THIS BOOK IS OF POOR LEGIBILITY DUE TO LIGHT PRINTING THROUGH OUT IT'S ENTIRETY.

THIS IS AS RECEIVED FROM THE CUSTOMER.

MRP EXPLOSION FOR THE COMMODORE COMPUTER

by

Karen Marie May

B.S., Florida State University, 1975

B.S. I.E., Kansas State University, 1982

A Master's Thesis

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

Kansas State University

Manhattan, Kansas

1983

Approved by:

L. E. Grosh

Major Professor

LD
2668
.T4
1983
M39
Vol. 1
c. 2

A11202 609827

TABLE OF CONTENTS

		Page
LIST OF FIGURES		xvi
ACKNOWLEDGMENTS		xxi
1.	INTRODUCTION	1
2.	MRP	3
2.1	Definition	3
2.2	History	8
2.3	Benefits of MRP	12
2.4	Sample Problem	15
2.4.1	Level Assignment	15
2.4.2	MRP Explosion	28
3.	Data Files	35
3.1	Data Requirements	35
3.2	Interconnection Between the Files	38
3.3	Data Files	41
3.3.1	Overview	41
3.3.2	Date File bom	42
3.3.3	Data File partid	46
3.3.4	Date File partstatus	50
3.3.5	Date File SR	56
3.3.6	Date File horizon	59
3.3.7	Date File DD	63
3.3.8	Data Files MPS, XO, and SP	65
4.	Input Programs	72
4.1	Introduction	72

4.2	PartFile	73
4.2.1	Introduction to PartFile	73
4.2.2	Subroutines in PartFile	74
4.2.2.1	Overview of Subroutines	74
4.2.2.2	Subroutine "setupadd"	76
4.2.2.2.1	Purpose	76
4.2.2.2.2	Variables	77
4.2.2.2.3	Description of Flow	80
4.2.2.2.4	Example of Nonrandomized File With Directional Keys	81
4.2.2.2.5	Example of Randomized File With Directional Keys	85
4.2.2.3	Subroutine "updown"	88
4.2.2.3.1	Purpose	88
4.2.2.3.2	Variables	90
4.2.2.3.3	Description of Flow	93
4.2.2.3.4	Example of Nonrandomized File With Up/Down Keys	96
4.2.2.3.5	Example of Randomized File With Up/Down Keys	98
4.2.2.4	Subroutine "keycheck"	101
4.2.2.4.1	Purpose	101
4.2.2.4.2	Variables	103
4.2.2.4.3	Description of Flow	106
4.2.2.4.4	Example of Nonrandomized File With Up/Down Keys for All Parts	107
4.2.2.4.5	Example of Randomized File With Up/Down Keys for All Parts	107
4.2.2.5	Subroutine "initialize"	107
4.2.2.5.1	Purpose	107

4.2.2.5.2 Variables.....107

4.2.2.5.3 Description of Flow.....109

4.2.2.5.4 Example of an Initialized Partstatus Data File.....110

4.3 BOMfile.....110

4.3.1 Introduction to BOMfile.....110

4.3.2 Subroutines in BOMfile.....113

4.3.2.1 Overview of Subroutines.....113

4.3.2.2 Subroutine "newbom".....116

4.3.2.2.1 Purpose.....116

4.3.2.2.2 Variables.....116

4.3.2.2.3 Description of Flow.....120

4.3.2.2.4 Example - Creating a bom File.....121

4.3.2.3 Subroutine "childkeys".....125

4.3.2.3.1 Purpose.....125

4.3.2.3.2 Variables.....125

4.3.2.3.3 Description of Flow.....127

4.3.2.3.4 Example - Assigning ckey and BofM.....128

4.3.2.4 Subroutine "findsys".....132

4.3.2.4.1 Purpose.....132

4.3.2.4.2 Variables.....132

4.3.2.4.3 Description of Flow.....133

4.3.2.4.4 Example - Finding the System Part Number.....135

4.3.2.5 Subroutine "addbom".....136

4.3.2.5.1 Purpose.....136

4.3.2.5.2 Variables.....137

4.3.2.5.3 Description of Flow.....139
4.3.2.5.4 Example - Adding Records to bom File.....140
4.3.2.6 Subroutine "changebom".....141
4.3.2.6.1 Purpose.....141
4.3.2.6.2 Variables.....146
4.3.2.6.3 Description of Flow.....147
4.3.2.6.4 Example - Changing bom Records.....148
4.3.2.7 Subroutine "pchange".....149
4.3.2.7.1 Purpose.....149
4.3.2.7.2 Variables.....152
4.3.2.7.3 Description of Flow.....154
4.3.2.8 Subroutine "cchange".....155
4.3.2.8.1 Purpose.....155
4.3.2.8.2 Variables.....155
4.3.2.8.3 Description of Flow.....157
4.3.2.9 Subroutine "qchange".....157
4.3.2.9.1 Purpose.....157
4.3.2.9.2 Variables.....157
4.3.2.9.3 Description of Flow.....158
4.4 DataPlus.....159
4.4.1 Introduction to DataPlus.....159
4.4.2 Subroutines in DataPlus.....160
4.4.2.1 Overview of Subroutines.....160
4.4.2.2 Subroutine "findsys".....163
4.4.2.3 Subroutine "IandSR".....163
4.4.2.3.1 Purpose.....163

4.4.2.3.2 Variable.....163
4.4.2.3.3 Description of Flow.....164
4.4.2.4 Subroutine "ISRnew".....164
4.4.2.4.1 Purpose.....164
4.4.2.4.2 Variables.....165
4.4.2.4.3 Description of Flow.....167
4.4.2.4.4 Example for Subroutine "ISRnew".....168
4.4.2.5 Subroutine "srchange".....169
4.4.2.5.1 Purpose.....169
4.4.2.5.2 Variables.....169
4.4.2.5.3 Description of Flow.....169
4.4.2.6 Subroutine "dayorqty".....171
4.4.2.6.1 Purpose.....171
4.4.2.6.2 Variables.....171
4.4.2.6.3 Description of Flow.....172
4.4.2.6.4 Example.....173
4.4.2.7 Subroutine "delSR".....174
4.4.2.7.1 Purpose.....174
4.4.2.7.2 Variables.....176
4.4.2.7.3 Description of Flow.....178
4.4.2.7.4 Example.....179
4.4.2.8 Subroutine "addSR".....180
4.4.2.8.1 Purpose.....180
4.4.2.8.2 Variables.....180
4.4.2.8.3 Description of Flow.....184

4.4.2.8.4 Example.....185
4.4.2.9 Subroutine "ohchange".....186
4.4.2.9.1 Purpose.....186
4.4.2.9.2 Variables.....186
4.4.2.9.3 Description of Flow.....188
4.4.2.10 Subroutine "leadandlot".....189
4.4.2.10.1 Purpose.....189
4.4.2.10.2 Variables.....189
4.4.2.10.3 Description of Flow.....189
4.4.2.11 Subroutine "leadtimeandlotrule".....190
4.4.2.11.1 Purpose.....190
4.4.2.11.2 Variables.....190
4.4.2.11.3 Description of Flow.....191
4.4.2.11.4 Example.....192
4.4.2.12 Subroutine 'leadtime'.....194
4.4.2.12.1 Purpose.....194
4.4.2.12.2 Variables.....194
4.4.2.12.3 Description of Flow.....195
4.4.2.12.4 Example.....196
4.4.2.13 Subroutine "lotrule".....197
4.4.2.13.1 Purpose.....197
4.4.2.13.2 Variables.....197
4.4.2.13.3 Description of Flow.....198
4.4.2.13.4 Example.....199
4.4.2.14 Subroutine "horizon".....200
4.4.2.14.1 Purpose.....200

4.4.2.14.2 Variables....200
4.4.2.14.3 Description of Flow....200
4.4.2.15 Subroutine "equalbucket"....201
4.4.2.15.1 Purpose....201
4.4.2.15.2 Variables....201
4.4.2.15.3 Description of Flow....201
4.4.2.15.4 Example....202
4.4.2.16 Subroutine "unequalbucket"....202
4.4.2.16.1 Purpose....202
4.4.2.16.2 Variables....202
4.4.2.16.3 Description of Flow....203
4.4.2.16.4 Example....203
4.5 DataPlus2....206
4.5.1 Introduction to DataPlus2....206
4.5.2 Subroutines of DataPlus2....206
4.5.2.1 Overview of Subroutines....206
4.5.2.2 Subroutine "findsys"....208
4.5.2.3 Subroutine "MSdemands"....208
4.5.2.3.1 Purpose....208
4.5.2.3.2 Variables....208
4.5.2.3.3 Description of Flow....208
4.5.2.4 Subroutine "MSnew"....208
4.5.2.4.1 Purpose....208
4.5.2.4.2 Variables....209
4.5.2.4.3 Description of Flow....211

4.5.2.4.4 Example....211

4.5.2.5 Subroutine "MSadd"....214

4.5.2.5.1 Purpose....214

4.5.2.5.2 Variables....214

4.5.2.5.3 Description of Flow....215

4.5.2.5.4 Example....216

4.5.2.6 Subroutine "MSchange"....217

4.5.2.6.1 Purpose....217

4.5.2.6.2 Variables....217

4.5.2.6.3 Description of Flow....220

4.5.2.6.4 Example....221

4.5.2.7 Subroutine "XOdemands"....222

4.5.2.7.1 Purpose....222

4.5.2.7.2 Variables....222

4.5.2.7.3 Description of Flow....222

4.5.2.8 Subroutine "XOnew"....223

4.5.2.8.1 Purpose....223

4.5.2.8.2 Variables....223

4.5.2.8.3 Description of Flow....225

4.5.2.8.4 Example....225

4.5.2.9 Subroutine "XOadd"....225

4.5.2.9.1 Purpose....225

4.5.2.9.2 Variables....226

4.5.2.9.3 Description....227

4.5.2.9.4 Example....228

4.5.2.10 Subroutine "XOchange"....228

4.5.2.10.1 Purpose....228
4.5.2.10.2 Variables....228
4.5.2.10.3 Description of Flow....230
4.5.2.10.4 Example....231
4.5.2.11 Subroutine "SPdemands"....231
4.5.2.11.1 Purpose....231
4.5.2.11.2 Variables....231
4.5.2.11.3 Description of Flow....231
4.5.2.12 Subroutine "SPnew"....232
4.5.2.12.1 Purpose....232
4.5.2.12.2 Variables....232
4.5.2.12.3 Description of Flow....234
4.5.2.12.4 Example....234
4.5.2.13 Subroutine "SPadd"....235
4.5.2.13.1 Purpose....235
4.5.2.13.2 Variables....235
4.5.2.13.3 Description of Flow....236
4.5.2.13.4 Example....237
4.5.2.14 Subroutine "SPchange"....237
4.5.2.14.1 Purpose....237
4.5.2.14.2 Variables....237
4.5.2.14.3 Description of Flow....239
4.5.2.14.4 Example....240
5. Explosion and Output Programs....241
5.1 Introduction....241

5.2 Level....242

5.2.1 Introduction to Level....242

5.2.2 Subroutines to Level....242

5.2.2.1 Overview....242

5.2.2.2 Subroutine "LandSinitial"....245

5.2.2.2.1 Purpose....245

5.2.2.2.2 Variables....245

5.2.2.2.3 Description of Flow....247

5.2.2.3 Subroutine "Lassign"....247

5.2.2.3.1 Purpose....247

5.2.2.3.2 Variables....248

5.2.2.3.3 Description of Flow....249

5.2.2.4 Subroutine "Sremove"....250

5.2.2.4.1 Purpose....250

5.2.2.4.2 Variables....250

5.2.2.4.3 Description of Flow....252

5.2.2.5 Subroutine "Levelconnect"....252

5.2.2.5.1 Purpose....252

5.2.2.5.2 Variables....253

5.2.2.5.3 Description of Flow....255

5.3 MRP....256

5.3.1 Introduction....256

5.3.2 Subroutines in MRP....259

5.3.2.1 Overview....259

5.3.2.2 Subroutine "DDkey"....261

5.3.2.2.1 Purpose....261

5.3.2.2.2 Variables....263
5.3.2.2.3 Description of Flow....264
5.3.2.3 Subroutine "setupHorizon"....264
5.3.2.3.1 Purpose....264
5.3.2.3.2 Variables....264
5.3.2.3.3 Description of Flow....265
5.3.2.4 Subroutine "MPStransfer"....265
5.3.2.4.1 Purpose....265
5.3.2.4.2 Variables....265
5.3.2.4.3 Description of Flow....267
5.3.2.5 Subroutine "XOtransfer"....268
5.3.2.5.1 Purpose....268
5.3.2.5.2 Variables....268
5.3.2.5.3 Description of Flow....269
5.3.2.6 Subroutine "SPtransfer"....269
5.3.2.6.1 Purpose....269
5.3.2.6.2 Variables....270
5.3.2.6.3 Description of Flow....271
5.3.2.7 Subroutine "GrossRequirements"....271
5.3.2.7.1 Purpose....271
5.3.2.7.2 Variables....271
5.3.2.7.3 Description of Flow....272
5.3.2.8 Subroutine "SRtransfer"....272
5.3.2.8.1 Purpose....272
5.3.2.8.2 Variables....272

5.3.2.8.3 Description of Flow.....274
5.3.2.9 Subroutine "OHandSR".....274
5.3.2.9.1 Purpose.....274
5.3.2.9.2 Variables.....274
5.3.2.9.3 Description of Flow.....275
5.3.2.10 Subroutine "LotSize.....276
5.3.2.10.1 Purpose.....276
5.3.2.10.2 Variables.....276
5.3.2.10.3 Description of Flow.....277
5.3.2.11 Subroutine "L4L".....277
5.3.2.11.1 Purpose.....277
5.3.2.11.2 Variables.....277
5.3.2.11.3 Description of Flow.....278
5.3.2.12 Subroutine "FPQ".....278
5.3.2.12.1 Purpose.....278
5.3.2.12.2 Variables.....278
5.3.2.12.3 Description of Flow.....280
5.3.2.13 Subroutine "OrderRelease".....280
5.3.2.13.1 Purpose.....280
5.3.2.13.2 Variables.....281
5.3.2.13.3 Description of Flow.....283
5.3.2.14 Subroutine "OutPut".....284
5.3.2.14.1 Purpose.....284
5.3.2.14.2 Variables.....284
5.3.2.14.3 Description of Flow.....285
5.3.2.15 Subroutine "SIX".....287

5.3.2.15.1 Purpose.....287

5.3.2.15.2 Variables.....287

5.3.2.15.3 Description of Flow.....288

5.3.2.16 Subroutine "FIVE",...,"ONE".....289

5.3.2.16.1 Purpose.....289

5.3.2.16.2 Variables.....290

5.3.2.16.3 Description of Flow.....290

5.3.2.17 Subroutine "OrderWrite".....290

5.3.2.17.1 Purpose.....290

5.3.2.17.2 Variables.....291

5.3.2.17.3 Description of Flow.....291

5.3.2.18 Subroutine "DependentDemands".....292

5.3.2.18.1 Purpose.....292

5.3.2.18.2 Variables.....292

5.3.2.18.3 Description of Flow.....295

5.4 SystemStatus.....296

5.4.1 Introduction.....296

5.4.2 Subroutines in SystemStatus.....296

5.4.2.1 Overview.....296

5.4.2.2 Subroutine "PartsList".....298

5.4.2.2.1 Purpose.....298

5.4.2.2.2 Variables.....298

5.4.2.2.3 Description of Flow.....298

5.4.2.3 Subroutine "sortedparts.....299

5.4.2.3.1 Purpose.....299

5.4.2.3.2 Variables.....299

5.4.2.3.3 Description of Flow.....299

5.4.2.4 Subroutine "unsortedparts".....301

5.4.2.4.1 Purpose.....301

5.4.2.4.2 Variables.....301

5.4.2.4.3 Description of Flow.....302

5.4.2.5 Subroutine "BOMlist".....302

5.4.2.5.1 Purpose.....302

5.4.2.5.2 Variables.....302

5.4.2.5.3 Description of Flow.....304

5.4.2.6 Subroutine "MRPflow".....305

5.4.2.6.1 Purpose.....305

5.4.2.6.2 Variables.....305

5.4.2.6.3 Description of Flow.....307

6. User's Guide.....308

6.1 Steps for Explosion.....308

6.2 Editing Guide.....322

6.2.1 To create new partid file.....322

6.2.2 To add more parts to the partid file.....323

6.2.3 To create new bom file.....323

6.2.4 To add more bom records.....324

6.2.5 To change bom records.....325

6.2.6 To revise inventory quantities and scheduled receipt information.....326

6.2.7 To change on-hand inventory quantity.....327

6.2.8 To change scheduled receipt record.....328

6.2.9 To delete scheduled receipt record.....329

6.2.10 To add scheduled receipt records.....330

6.2.11 Equal Buckets for Planning Horizon.....331

6.2.12 Unequal Buckets for Planning Horizon.....332

6.2.13 To create new MS file.....332

6.2.14 To add master production schedule demand records.....334

6.2.15 To change master production schedule demands records..334

6.2.16 To create new XO data file.....335

6.2.17 To add external order demand records.....335

6.2.18 To change external order demands records.....335

6.2.19 To create new SP data file.....335

6.2.20 To add service part demand records.....335

6.2.21 To change service part demand records.....336

6.2.22 To enter new lead time and lotting rules.....336

7. Conclusion.....337

7.1 Implementation of a MRP System.....337

7.2 Software Capabilities.....339

7.3 The Commodore Computer.....340

Footnotes.....342

Bibliography.....346

Appendix A - Programs (Debugging).....349

Appendix B - Programs (Working).....436

Appendix C - Flowchart.....499

Appendix D - Miscellaneous.....660

LIST OF FIGURES

		Page
Figure 2.1	Part Status Information for Sample Problem.....	16
Figure 2.2	Modular Bills-of-Materials for Sample Problem.....	17
Figure 2.3	Master Production Schedule for Sample Problem.....	18
Figure 2.4	Iteration 1 = Level 1.....	20
Figure 2.5	Iteration 2 = Level 2.....	21
Figure 2.6	Iteration 3 = Level 3.....	22
Figure 2.7	Iteration 4 = Level 4.....	22
Figure 2.8	Assembly Level Iteration 1 = Level 1.....	24
Figure 2.9	Iteration 2 = Level 2.....	25
Figure 2.10	Iteration 3 = Level 3.....	26
Figure 2.11	Iteration 4 = Level 4.....	27
Figure 2.12	Part A Matrix.....	30
Figure 2.13	Part G Matrix.....	31
Figure 2.14	Part J Matrix.....	32
Figure 2.15	Part E Matrix.....	33
Figure 2.16	Part H Matrix.....	34
Figure 3.1	File Links.....	43
Figure 3.2	Unit Numbers and Record Lengths.....	44
Figure 3.3	Data File bom.....	45
Figure 3.4	Sample Problem bom File.....	47
Figure 3.5	Data File partid.....	48
Figure 3.6	Sample Problem partid File.....	51
Figure 3.7	Data File partstatus.....	53
Figure 3.8	Sample Problem partstatus File.....	57

Figure 3.9 Data File SR..... 58
Figure 3.10 Sample Problem SR File..... 60
Figure 3.11 Data File horizon..... 61
Figure 3.12 Initialized horizon File..... 62
Figure 3.13 Data File DD..... 64
Figure 3.14 Sample of DD File..... 66
Figure 3.15 Data File MPS..... 67
Figure 3.16 Data File XO..... 68
Figure 3.17 Data File SP..... 69
Figure 3.18 Sample Problem MPS File..... 70
Figure 3.19 Sample Problem XO File..... 71
Figure 3.20 Sample Problem SP File..... 71
Figure 4.1 Part File Subroutines..... 75
Figure 4.2 Stage 3 partid with Directional Keys..... 82
Figure 4.3 Stage 4 partid with Directional Keys..... 82
Figure 4.4 Stage 5 partid with Directional Keys..... 82
Figure 4.5 Stage 6 partid with Directional Keys..... 83
Figure 4.6 Stage 3 Randomized partid with Directional Keys..... 86
Figure 4.7 Stage 4 Randomized partid with Directional Keys..... 86
Figure 4.8 Stage 5 Randomized partid with Directional Keys..... 86
Figure 4.9 Stage 6 Randomized partid with Directional Keys..... 87
Figure 4.10 Stage 7 Randomized partid with Directional Keys..... 87
Figure 4.11 Binary Tree..... 89
Figure 4.12 Binary File..... 89
Figure 4.13 Stage 2 partid With Up/Down Keys..... 99

Figure 4.14 Stage 3 partid With Up/Down Keys..... 99

Figure 4.15 Stage 2 Randomized partid With Up/Down Keys.....102

Figure 4.16 Stage 3 Randomized partid with Up/Down Keys.....108

Figure 4.17 The partid File After keycheck.....108

Figure 4.18 The Randomized partid File After keycheck.....108

Figure 4.19 Initialized partstatus File.....111

Figure 4.20 BOMfile Subroutines.....115

Figure 4.21 Step 7 bom File.....123

Figure 4.22 Step 10 bom File.....123

Figure 4.23 Step 13 bom File.....123

Figure 4.24 Step 18 bom File.....124

Figure 4.25 The bom File with ckey Assigned.....129

Figure 4.26 BofM and BOM in partstatus File.....130

Figure 4.27 Step 5 Additions to bom File.....142

Figure 4.28 Step 6 Additions to bom File.....142

Figure 4.29 Step 7 Additions to bom File.....143

Figure 4.30 Step 8 Additions to bom File.....143

Figure 4.31 Step 9 Additions to bom File.....144

Figure 4.32 Completed Sample Problem bom File.....144

Figure 4.33 All the numkids, BofM and BOM Assigned.....145
for Sample Problem

Figure 4.34 Parent Changed on bom Record.....150

Figure 4.35 Child Changed on bom Record.....150

Figure 4.36 Quantity Changed on bom Record.....151

Figure 4.37 DataPlus Subroutines.....161

Figure 4.38 The partstatus with Sample Problem Inventory.....170
and SR Keys Included

Figure 4.39 SR Data File with Sample Problem Scheduled Receipts...170
Figure 4.40 SR Data File After the Changes.....175
Figure 4.41 SR With 'a, 120, 12' Deleted.....181
Figure 4.42 SR With 'f, 105, 100' Deleted.....181
Figure 4.43 SR With 'c, 95, 200' Deleted.....182
Figure 4.44 SR After Additions Made.....187
Figure 4.45 The partstatus With Lead Times and Lot Rules.....193
Figure 4.46 The partstatus With Lead Times Changes.....193
Figure 4.47 The partstatus With Lot Rule Changes.....204
Figure 4.48 The horizon With Equal Buckets.....204
Figure 4.49 Unequal Horizon Buckets.....205
Figure 4.50 Unequal Buckets in horizon File.....205
Figure 4.51 DataPlus2 Subroutines.....208
Figure 4.52 MPS File; Plus partstatus File With MPSkey Assigned...213
Figure 4.53 MPS File With Additions.....218
Figure 4.54 MPS File With Changes.....218
Figure 5.1 Level Subroutines.....244
Figure 5.2 The partstatus With level and levelkey Assigned.....244
Figure 5.3 MRP Subroutines.....262
Figure 5.4 SystemStatus Subroutines.....297
Figure 5.5 Sorted Company Part Number List.....300
Figure 5.6 Unsorted Company Part Number List.....300
Figure 5.7 BOMlist.....303
Figure 5.8 MRPflow List.....303
Figure 6.1 CRT/Keyboard Unit.....309

Figure 6.2 Disk Drive Unit.....309

Figure 6.3 Top View of Commodore Tractor Printer 8023P.....310

Figure 6.4 Microprocessor Switches.....310

Figure 6.5 Positioning of Floppy Disk.....311

Figure 6.6 Steps for Explosion.....321

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. L. E. Grosh, my advisor, Drs. C. L. Hwang, M. Vaithianathan, and Mr. B. P. Mignano, professors in the Industrial Engineering Department. But, most of all, I want to say thanks to my husband, John L. May, Ph.D., for his love, patience, and support.

THIS BOOK
CONTAINS
NUMEROUS PAGES
WITH THE ORIGINAL
PRINTING BEING
SKEWED
DIFFERENTLY FROM
THE TOP OF THE
PAGE TO THE
BOTTOM.

THIS IS AS RECEIVED
FROM THE
CUSTOMER.

1. INTRODUCTION

The objective of this master of science thesis is to develop software capable of performing the basic MRP explosion. MRP is the abbreviation for Materials Requirement Planning. Basically, Materials Requirement Planning is in one sense simply a inventory planning technique used to answer the basic inventory questions. The basic inventory questions include: What parts to order? What quantity to order? When to place the order? In another sense, MRP starts out as an inventory technique and then expands into many other areas as well, such as capacity planning, financial planning, personnel planning, etc.

To answer those inventory questions, the MRP explosion uses the information from part's status (lead time, on hand inventory quantity, etc.), bills-of-materials, and the master production schedule.[1] The MRP explosion manipulates the information and makes simple calculations to determine the data required for placing the part's orders.

More specifically, the objectives for this M.S. thesis center around the simpler MRP system which includes designing the data files, creating the input programs, writing the MRP explosion programs, and developing the formatted output for the results. The data file is where the information from the part status, the bills-of-materials, and the master production schedule are stored. The input programs work interactively with the MRP user to create the data file. The MRP explosion programs include one file to assign a level to all the parts and one file to do the MRP calculations. The formatted output is designed to aid the MRP user in the interpretation of the results.

As stated by D. R. Schuchts, "The basic processing of an MRP system is extremely simple..."[2] The complexity arises from the multitude of design factors, the file organization and file management. Some of the design factors mentioned by Urban Wemmerlov are "Length of the planning horizon, Time - bucket size, Lot - sizing techniques, Safety Stocks, Pegged requirements, Firm planned orders, Lead times, Replanning approaches, Replanning frequencies, Rescheduling polices, Preparation of the master schedule, Capacity planning, Relations to the financial system, Priority control."[3]

Joseph Orlicky comments that "The importance of file organization and file management to the success of a computer-based system is great, particularly so because of a universal tendency on the part of management to underestimate both the importance and the requirements of this part of the system. Lack of file integrity is one important reason why some MRP systems installed in industry have failed to live up to expectations. Emphasis must therefore be put on system files, their organization maintenance, and accessibility. Computer manufacturers have invested heavily in the development of file management programs (data-base software), a tool that helps greatly in coping with the problem of maintaining file integrity."[4]

The software has already been successfully developed and expanded with many additional facets. In fact, the MRP system has been successfully implemented by many different companies. But, the software is still a big money-maker. One reason for the development of the software is that a MRP system makes an excellent teaching tool. Another reason is to investigate the possibility of a microcomputer being capable of handling the software.

2. MRP

2.1 Definition

For most business concerns, the task at hand is to make and sell a product to customers at a profit. To make the product, the required inventories (subassemblies, components, raw materials, etc.) are known. But, should the order size be large or small; and should the order be placed today or tomorrow? The correct answer revolves around the situation of increasing profits.

The goal of attaining increased profits is improved by decreasing inventory costs. In solving this problem, there are two conflicting solutions. On one side as pointed out by Roy L. Allen, "...It is possible to reduce inventory by purchasing more frequently, in smaller lots. However, the processing of many small orders to the distributor's vendors and the increased receiving load would be likely to result in serious disruption of operations. Further, a substantially smaller inventory might incur risk of unacceptable delays in filling customer orders."[5] The small inventory quantities may result in lower storage costs, fewer obsolescence losses, and less spoilage costs. But, the small inventory quantities may, also, result in a loss of a sale to an impatient customer(s) and larger purchasing costs.

On the other side of the conflict, as stated by L. S. Shealy, "... Inventory eats up large quantities of working capital; if inventory turnover is improved, the need for capital is reduced. Finding adequate working capital is probably the largest single problem facing expansion today and in the future."[6] A large inventory results favorably in being able to satisfy customers on a moment's notice and almost never having to

experience a stockout. But large inventories, also, result unfavorably in larger storage costs, higher taxes, and greater wastes due to spoilage. Therefore, the answer to how much inventory lies at some compromised point where the amount isn't too large or too small.

To resolve the order quantity problem, J. Orlicky states that, "In most manufacturing businesses, the bulk of the total inventory is in raw materials, components parts, and subassemblies, all largely subject to dependent demands. Such demand can of course be calculated. Dependent demand need not, and should not, be forecast, as it can be precisely determined from the demand for those items that are its sole cause".[7]

The sole product having an independent demand, is that generally sold to the customer. The dependent demands are the subassemblies, components, and raw materials that go into making products with independent demands. Knowing the quantity of independent demands and the different dependent quantities needed per independent demand, the total quality of dependent demands needed is determined by means of simple multiplication.

Thus, "The objective of MRP is to minimize on-hand physical stocks of raw materials and component parts while providing assurance of the availability of needed stocks for planned production runs with a high degree of confidence",[8] as pointed out by R. J. Campbell and T. M. Porcano. Another point of view by D. R. Schuchts is "The basic objectives of an MRP System are equally simple: order the right materials (purchased and manufactured) to arrive at the right time and is sufficient quantity to produce the products, as scheduled; and, by doing so, establish and maintain the proper priorities for material replenishment."[9]

To meet the objectives of MRP, "The requirements for 'exploding' the production schedule into meaningful information on material and component acquisition", as summarized by R. J. Campbell and T. M. Porcano, "are as follows:

1. a reliable and up-to-date production schedule detailed by product or product grouping;
2. knowledge of reliable lead times for acquisition of raw materials from vendors and fabrication of component parts plus assembly in the factory;
3. bills of materials with up-to-date engineering specifications."[10].

The master production schedule presents a time phased list of the product quantities needed. The bills-of-materials, BOM, contain the quantity of the subparts needed per unit of products. The lead times for each subpart item provides the bases for determining when the components are actually needed.

J. P. Morgan explains that "The materials requirements plan then takes the input from the master production schedule and shows for each individual item or component the starting inventory, the requirements, the open orders, and planned order releases. The projected inventory balances are then calculated for each item."[11] R. J. Campbell and T. M. Porcano explain further that "The MRP system produces time-phased records for each component part or raw material by specifying for a series of time periods (weekly over the next year, for example) various items such as (1) stock requirements by week, (2) scheduled receipts or completion of component parts, (3) planned on-hand inventory, and (4) planned order releases for

specific time periods."[12] These calculations are often referred to as the MRP explosion.

The information generated by the MRP explosion states what to order, how much to order and when to order. However, as J. Orlicky points out, "In the modern view, the most important function of an MRP system is not inventory control but its ability to plan and replan priorities, i.e., to schedule and reschedule orders."[13] This is MRP in the expanded sense. The output's uses and functions are almost endless. One area for its use, J. P. Morgan outlines, is that "The capacity requirements plan calculates the standard hours of production, by work center and by time period, required to satisfy the master production schedule."[14] D. R. Schuchts discusses purchasing and receiving systems, capacity and priority control systems, and shop floor reporting and control systems,[15] as a few areas where the MRP outputs are used. In a Modern Material Handling article, "The shipping plan...The master production schedule...The material requirements plan...The purchase commitments report...a shop capacity report...The customer-order forecast...customer-order status...Product cost reports...Expense budget reports..."[16] are mentioned as extensions from the simple MRP system.

"...a computer is required only because the mass of data to be gathered, sorted, correlated, and constantly changed/updated is overwhelming on a manual basis."[17] In summary, J. Greene writes that the "MRP Computer Software...generates requirements by time periods for finished products, assemblies, subassemblies, and raw material from an input of order and forecast information. To be more specific, the program will do

the following:

Determine gross requirements for finished products.

Determine net finished product requirements.

Determine net component requirements.

Plan order sizes.

Offset requirements considering lead times.

Maintain and update the requirements plan (requirements alternations).

Provide a review of planned orders (interrupt) and adjustments to planned orders (reentry) which is referred to as a conversational planning.

Provide management by exceptions."[18]

To define MRP (Materials Requirements Planning) is a complex task. J. C. Anderson et al. explain MRP in the following way:"...In the simplest sense two extreme views of MRP could be offered: First, that MRP is a narrow concept involving the parts explosion process, i.e., the process of exploding the demand for independent demand items into their necessary components. In this sense, MRP has facilitated order launching. At the other extreme is a broad definition which says that Material Requirements Planning is a concept that involves the entire manufacturing control system. It starts with parts explosion and order launching, but it goes far beyond that to capacity planning, purchasing, shop floor control, costing, and interfaces with a variety of other necessary functions to effectively operate the manufacturing business."[19] The focus of this thesis is on the first, narrow concept of MRP. After the first, narrow concept of MRP is established, the second broader definition is the next

level up - an extension - of the simpler concept.

2.2 History

G. Plossl claims that MRP has been used "...Ever since man began fashioning objects out of various materials; he did this by asking what was required to make what he desired, what did he already have, what else was needed?..."[20] In preparation for dinner, a good cook uses a simplified MRP. The number of diners is known, the menu and their ingredients is known, and cooking times are known; this information is used by the cook to determine what to buy at the grocery store, how much to buy, and when to start the cooking.

"Although MRP was the earliest material planning technique, there was a radical departure from its clear, basic logic beginning in the 1920s and continuing through the 1940s. Products became more complex with more components, backlogs of firm customer orders dwindled and there was greater dependence upon forecasts..."[21], as observed by G. Plossl. The manual calculations of determining the answers to the inventory questions with MRP were becoming more and more cumbersome and time consuming.

As an alternative to the simplicity of MRP, "...the 'cult of order point' arose in the '40s and persisted in the '50s up to the mid-'60s..."[22] The mathematician made an attempt to solve the inventory cost problem by using a mathematical model. Unfortunately, the real world more often than not doesn't mesh nicely with the model.

Orlicky, Plossl, and Wight point out that "An important distinction between Order Point systems and Material Requirement Planning lies in the fact that the order point/order quantity approach is part based whereas

MRP is product-oriented."[23] Therefore, another disadvantage to using order point methods is that the ordering of the parts has no correlation with the resulting end-product. For example, the product A is composed of 2 B parts and 1 C part. The stockroom contains 10,000,000 B parts and no C parts. If a customer wants to purchase 1,000,000 units of product A, the order won't be filled right away, if at all, because without the C parts product A cannot be made.

In discussing the history of inventory management, J. Orlicky observes that "The commercial availability of computers in the mid-1950s ushered in a new era of business information processing, with a profound impact of the new technology on the conduct of operations. Nowhere has the impact probably been greater, at least potentially, than in the area of manufacturing logistics, i.e., inventory management and production planning..."

"The first computer applications, around 1960, in the area of manufacturing inventory management represented the beginning of a break with tradition..."[24]

During this period, various industries, such as the aerospace and electronic fields, were presenting a more and more of a challenge. Therefore, as J. Orlicky points out "...the following mainstays of conventional inventory management became subject to reappraisal as to their validity, relevance, and applicability of manufacturing inventories:

1. The concept of stock replenishment
2. All techniques built around reorder points
3. The square-root approach to the economic order quantity
4. The analysis and categorization of inventory by function

5. The conventional notion of aggregate inventory management
6. The ABC inventory classification."[25]

Therefore, as a result, the ideas of MRP were again reviewed as a possible way to resolve the inventory problem. With the use of the computer, the time problems of the MRP technique would be eliminated. The cumbersome problem could be more easily sorted out by the computer.

J. Orlicky states that, "In the area of manufacturing inventory management, the most successful innovations are embodied in what has become known as materials requirements planning (MRP) systems."[26] G. Plossl feels, however, that "Rather than the birth of MRP we must speak of a rebirth..."[27] On the other hand, D. A. Davarajan claims "The main features that form the basis of MRP are not new... However, the integration of time-phased requirements generation with computerized, arithmetical operations has given a new dimension to production scheduling and inventory control problems in manufacturing operations involving large numbers of components and operations. The computer's ability to process a large quantity of data in a short time has become the catalyst for manufacturing industries to use the MRP system extensively for inventory control and planning of priorities and capacities..."[28]

The old inventory techniques used by Materials Requirement Planning to answer what to order, how much to order, and when to order is not new. However, the computer applications have enabled the simple aspects (or outputs) of MRP to be expanded and used in every area of the business' concern.

The return to MRP has not occurred overnight and still needs to be sold to many of the order point diehards. J. Orlicky claims "The lag of literature became painfully evident to me when, in early 1973...I wanted to list all written material pertaining to the subject. I found that the entire MRP listerature consisted of twenty-six items--good, bad, and indifferent--all of which were either articles, excerpts, special reports, or trade-press "testimonials." Tutorial material on MRP basics was lacking entirely..."[29]

However, J. C. Anderson et al. feels that, "Interest is growing in MRP system development across the country in a variety of firms and industries. The demands, complexities, and critical interdependencies of our productive systems appear to require a more informed, disciplined, planned and controlled approach which MRP system can, in part, facilitate."

"The design of the MRP system varies widely in breadth of definition and particular features incorporated. Both definition and system features seem to depend more upon particular company characteristics and needs rather than maturity of usage..."[30]

In retrospect, G. Plossl points out that "...MRP has had a fascinating, tumultuous history. However, it is no longer the star in our firmament or the savior in our heaven. It's just a member of the team with other tools and techniques we have to help us attack our problems..."[31] In other words, MRP does not create an absolute solution to the inventory (or business) decision problems but, MRP is a big help to the decision maker.

2.3 Benefits of MRP

Some of the benefits, which may be derived from using the MRP systems, are varied. The source of benefits depends upon the sophistication of the MRP system and the receptiveness by the company's personnel. Some of the different points of view are as follows:

D. F. Blumberg feels that "...MRP is only one of the elements in the overall chain associated with the management of the inventory process..."[32] Then, he states "In summary, MRP systems can provide a part of the mechanism necessary to improve productivity in terms of efficient utilization of raw material and semifinished goods inventory. However, an MRP system by itself will not be successful unless it is balanced with and tied to optimal manufacturing strategy and forecasts and linked to an order entry and finished goods inventory management system..."[33]

"Symptoms of Manufacturing Problems," D. B. Boxx points out are:

"* Low inventory turns

* Severe parts shortage

* Poor customer service

* Low labor productivity

* Selling at a loss or marginal profitability

* Competitive posture changing to unfavorable"[34]

Then, he claims that "...the ability MRP gives us to look into the future can reduce inventory by telling us to bring in only the quantities we need when we need them..."

"...reduce purchasing costs by allowing us to get the price advantages associated with annual buys and write fewer purchase orders and to schedule the inventory to be received only when we need it..."

"...improve vendor relationships and receive more realistic lead times by giving the vendor visibility into our future requirements so that the vendor can better plan for the future demands upon his shop..."

"...improve labor productivity by having the direct labor force work on only those items that require their labor today. This can substantially reduce the number of direct labor people and WIP inventories required because only those items that are needed are being worked on..."

"...MRP will allow us to quote more realistic dates to our customers because we have better visibility into the future...tells us exactly which customer orders will be affected by the inevitable changes that will occur..."[35]

R. J. Campbell and T. M. Porcano believe that, "The results obtained should include:

1. minimal dollars tied up in on-hand inventory;
2. fewer production run disruptions due to stock-outs of required material(s);
3. reduction of added costs associated with rush or expedited orders both firm vendors and within the firm's own shop;
4. reduction in handling costs of orders as a result of advance planning on order release;
5. reduction in storage costs associated with inventories."[36]

A Modern Material Handling article presented the observations that "MRP simulations help us to fine tune manufacturing resource planning--keeping production schedules consistent with top management's financial and production plan..."[37]

J. Orlicky claims that "MRP systems are a highly effective tool of manufacturing inventory management for the following reasons:

- Inventory investment can be held to a minimum.
- An MRP system is change-sensitive, reactive.
- The system provides a look into the future, on an item-by-item basis.
- Under material requirements planning, inventory control is action-oriented rather than clerical bookkeeping-oriented.
- Order quantities are related to requirements.
- The timing of requirements, coverage, and order actions is emphasized."[38]

L. S. Shealy acknowledges that "This case study of material requirements planning and the bottom line at Star Manufacturing Company shows how MRP, when implemented, helped increase profits significantly while providing better inventory control."[39] Then, to answer the questions "What were the benefits of MRP?", L. S. Shealy's response is "First and foremost, lower inventory requirements. In addition, the following were achieved:

* Inventory counts were more accurate and took less time.
* Obsolete items inventory was reduced.
* It took less time to review a purchase order...
* Ability to control order quantity. MRP provides the ability to order in quantities that are standard to its suppliers..."[40]

2.4 Sample Problem

The sample problem appeared in a take home exam for "Advanced Production and Inventory Control" taught by L. E. Grosh at Kansas State University. This problem and variations of the problem is presented for the following purposes:

1. to show how the part's level number is assigned;
2. to explain the different steps to performing the MRP explosion;
3. to explain the different data files;
4. to test the software;
5. to explain how the software works;
6. to show how to use the software.

The part status information is tabulated in Figure 2.1. The bills-of-materials is presented in Figure 2.2. Finally, the production schedule, which includes all the master production schedule (MPS) demands, the external order (XO) demands, and Service Parts (SP) demands, is listed in Figure 2.3.

2.4.1 Level Assignment

Each part in inventory is assigned a level number which indicates its dependency relationship with the other parts in inventory. On the subject of part leveling, J. Orlicky gives the following advice "The key to the entire material requirements planning (replanning) process is the linkage between parent-and component-item records. There is only one logical link between items on contiguous levels of the product structure, and that is the parent planned--order release and two component gross requirement. These coincide in time, because the component item(s) must be planned to

Parts	On-hand Inventory	Scheduled Receipts Day	Qyt	Lead Time (days)	Lotting rule	Lotting Series (days)
a	10	100	12	10	L4L	5
b	0	95 105	825 270	20	FPQ	10
c	0	95 105	210 250	15	FPQ	10
d	320	--	--	20	FPQ	35
e	66	100	67	15	FPQ	10
f	100	105	200	20	FPQ	35
g	10	--	--	10	L4L	5
h	10	100	23	10	FPQ	10
i	75	--	--	15	FPQ	20
j	3	--	--	5	FPQ	10

Abbreviations:

L4L = Lot-for-Lot

FPQ = Fixed Period Quantity

Figure 2.1

Part Status Information for Sample Problem

THIS BOOK CONTAINS NUMEROUS PAGES WITH ILLEGIBLE PAGE NUMBERS THAT ARE CUT OFF, MISSING OR OF POOR QUALITY TEXT.

THIS IS AS RECEIVED FROM THE CUSTOMER.

Parent	Child	Quantity
a	b	2
	c	1
	e	2
c	d	1
	b	1
e	f	1
	c	2

Parent	Child	Quantity
g	f	1
	b	4
	c	2
	h	1
h	i	1
i	f	3
	d	4
j	b	1
	i	2

Figure 2.2

Modular Bills-of-Materials for Sample Problem

Period / day	19	20	21	22	23	24	25	26	27	28	29
Part	95	100	105	110	115	120	125	130	135	140	145
a	10	12	14	14	16	16	18	18	20	20	20
g	9	1	10	12	11	12	13	12	11	14	10
j	3	4	4	5	3	10	2	3	4	4	3
External Orders											
e			5			10		2		6	
i		4			3		2				9
Service Parts											
c	2	2	3	2	2	3	2	2	3	2	2
i	1	0	1	0	1	0	1	0	1	0	1

Note: One period = 5 word days

Figure 2.3

Master Production Schedule for Sample Problem

be available at the time the parent order is released for production, at which time the component item(s) will be consumed.."

"The standard technique for maximizing processing efficiency is to process all the items on a given level before addressing their component on the next lower level. This is known as level-by-level processing..."[41]

In the "Advanced and Inventory Control" class taught by L. E. Grosh, a procedure for assigning levels was outlined. The procedure requires the use of the parts list and a bills-of-materials list. The form of the bills-of-materials (BOM) should be a paired list of parent (assembled or made) parts and child (component) parts. One of the parent and child pairs may be referred to as a BOM couplet. The steps of the procedure are as follows:

"1. Identify the parts in the parts list that do not appear in the parts made column of the BOM couplets list. These parts belong to this level.

2. Working with the parts assigned to this level, remove all BOM couplets with these part numbers.

3. Recopy the reduced list.

4. Proceed to the next level (back to 1) if neither list is empty otherwise stop."

This procedure was applied to the sample problem. See Figure 2.4 to 2.7. The results are that parts b, d, and f are assigned to level 1; parts c and i are assigned to level 2; parts e, h, and j are assigned to level 3; and parts a and g are assigned to level 4. With this method, part a and g are classified as the independent demands parts; and the

BOM Couplets

Parts List	Part Made	Part Needed	
a	a	b	* *
b*	a	c	
c	a	e	
d*	c	d	* *
e	c	b	* *
f*	e	f	* *
g	e	c	
h	g	f	* *
i	g	b	* *
j	g	c	
	g	h	
	h	i	
	i	f	* *
	i	d	* *
	j	b	* *
	j	i	

Note: * Part not in part made list

** BOM couplet to be eliminated

Figure 2.4

Iteration 1 = Level 1

BOM Couplets

Parts List	Part Made	Part Needed	
a	a	c	* *
c*	a	e	
e	e	c	* *
g	g	c	* *
h	g	h	
i*	h	i	* *
j	j	i	* *

Note: * Part not in part made list

** BOM couplet to be eliminated

Figure 2.5

Iteration 2 = Level 2

BOM Couplets

Parts List	Part Made	Part Needed	
a	a	e	* *
e*	g	h	* *
g			
h*			
j*			

Note: * Part not in part made list

** BOM couplet to be eliminated

Figure 2.6

Iteration 3 = Level 3

BOM Couplets

Parts List	Part Made	Part Needed
a		
g		

Note: There are no BOM couplets left. The algorithm stops.

Figure 2.7

Iteration 4 = Level 4

other parts are classified as the dependent demand parts.

However, J. Orlicky advises that the "...low-level coding technique. The lowest level at which any inventory item appears is identified through analysis of the bill of material file, and this information--the low-level code--is added either to the bill of material or to the items inventory record. In the level-by-level requirements-computation process, the processing of the item is then delayed until the lowest level on which it appears is reached. At that point, all the possible gross requirements stemming from parent items at any of the higher levels have been established and the need for multiple retrieval and processing has been forestalled..."[42] In reference to the sample problem, this means that the parts a and g, with the maximum level number 4, are to be processed first; then, parts e, h and j are to be processed next; and so on.

Unfortunately, without knowing ahead of time the number of levels that might be assigned, telling the computer where to begin processing is a little difficult. Therefore, the algorithm in the Level program (described in Section 5.2) is the reverse of the one just described. Thus, the MRP processing begins not with the parts with the maximum level number, but with the parts with the level number of one.

The reverse algorithm, used in the Level program, contains the following steps:

1. Find the parts that don't appear in the child's part list and do not have a level number already assigned.
2. Eliminate the BOM record for which the part appears in the parent's part list.

Parts List	Level Number	BOM List Parent	Child	
a	Ø 1	a	b	* *
b	0	a	c	* *
c	0	a	e	* *
d	0	c	d	
e	0	c	b	
f	0	e	f	
g	Ø 1	e	c	
h	0	g	f	* *
i	0	g	b	* *
j	Ø 1	g	c	* *
		g	h	* *
		h	i	
		i	f	
		i	d	
		j	b	* *
		j	i	* *

Note: ** BOM record to be eliminated

Figure 2.8 Assembly Level

Iteration 1 = Level 1

Parts List	Level Number	BOM List Parent	Child	
a	1	c	d	
b	0	c	b	
c	0	e	f	* *
d	0	e	c	* *
e	Ø 2	h	i	* *
f	0	i	f	
g	1	i	d	
h	Ø 2			
i	0			
j	1			

Note: ** BOM record to be eliminated

Figure 2.9

Iteration 2 = Level 2

Parts List	Level Number	BOM List Parent	Child	
a	1	c	d	* *
b	0	c	b	* *
c	Ø 3	i	f	* *
d	0	i	d	* *
e	2			
f	0			
g	1			
h	2			
i	Ø 3			
j	1			

Note: ** BOM record to be eliminated

Figure 2.10

Iteration 3 = Level 3

Parts List	Level Number	BOM List Parent	BOM List Child
a	1		
b	Ø 4		
c	3		
d	Ø 4		
e	2		
f	Ø 4		
g	1		
h	2		
i	3		
j	1		

Figure 2.11

Iteration 4 = Level 4

3. If all parts have a level number assigned, stop. If not, go back to step 1.

This procedure was applied to the sample problem. See Figure 2.8 to 2.11. The results are parts a, g, and j are assigned to level 1; e and h are assigned to level 2; c and i are assigned to level 3; and b, d, and f are assigned to level 4. The independent demand parts are a, g, and j; and the other parts are the dependent demand parts. The MRP processing beginning at level 1 starts with parts a, g, and j; then, parts e and h; and so on.

2.4.2 MRP Explosion

For each part the MRP explosion consists of the following steps:

Step 1. Place the quantities of demands into their appropriate time period. The source of the demands may come from either the production schedule and/or dependent demands from the lower levels.

Step 2. Calculate the gross requirements by adding all the production schedule demands and the dependent demands.

Step 3. List the scheduled receipts into their appropriate time slot.

Step 4. Extend on hand quantities as far into the future as possible. On-hand quantity equals gross requirements less the scheduled receipts.

Step 5. Determine net requirements, which are equal to the gross requirement not covered by the scheduled receipts and on hand inventory quantity.

Step 6. Use the specified lotting technique to figure the lot size.

Step 7. Adjust the lot size by the parts lead time to determine the correct order release time.

The MRP explosion calculations are usually summarized in a requirement chart, or MRP matrix. The explosion has been done manually for parts a, g, j, e, and h. The results are shown in Figures 2.12 to 2.16. The following abbreviations are used in the charts:

MPS - master production schedule demands

XO - external order demands;

SP - service parts demands;

DD_{nij} = x - dependent demands for part i from source j at the ratio of x - i pats per j part;

GR - gross requirements;

SR - scheduled receipts,

OH - on hand quantities;

NR - net requirements;

LR - lot size requirements;

OR - order release requirements.

PART A

Leadtime = 10 days

Lotting Rule = L4L

On-Hand Inventory = 10 units

Period	19	20	21	22	23	24	25	26	27	28	29
Day	95	100	105	110	115	120	125	130	135	140	145
MPS	10	12	14	14	16	16	18	18	20	20	20
GR	10	12	14	14	16	16	18	18	20	20	20
SR		12									
OH	--	--									
NR			14	14	16	16	18	18	20	20	20
LR			14	14	16	16	18	18	20	20	20
OR	14	14	16	16	18	18	20	20	20		

Figure 2.12

Part A Matrix

PART G

Leadtime = 10 days

Lotting Rule = L4L

On-Hand Inventory = 10 units

Period	19	20	21	22	23	24	25	26	27	28	29
Day	95	100	105	110	115	120	125	130	135	140	145
MS	9	1	10	12	11	12	13	12	11	14	10
GR	9	1	10	12	11	12	13	12	11	14	10
SR											
OH	1	0									
NR			10	12	11	12	13	12	11	14	10
LR			10	12	11	12	13	12	11	14	10
OR	10	12	11	12	13	12	11	14	10		

Figure 2.13

Part G Matrix

PART J

Leadtime = 5 days

Lotting Rule = FPQ for 10 days

On-Hand Inventory = 3 units

Period	19	20	21	22	23	24	25	26	27	28	29
Day	95	100	105	110	115	120	125	130	135	140	145
MS	3	4	4	5	3	10	2	3	4	4	3
GR	3	4	4	5	3	10	2	3	4	4	3
SR											
OH	0										
NR		4	4	5	3	10	2	3	4	4	3
LR		8		8		12		7		7	
OR	8		8		12		7		7		

Figure 2.14

Part J Matrix

PART E

Leadtime = 15 days

Lotting Rule = FPQ for 10 days

On-Hand Inventory = 66 units

Period	19	20	21	22	23	24	25	26	27	28	29
Day	95	100	105	110	115	120	125	130	135	140	145
XO			5			10		2		6	
$DD_{nea}=2$	28	28	32	32	36	36	40	40	40		
GR	28	28	37	32	36	46	40	42	40	6	
SR		67									
OH	38	77	40	8	0						
NR					28	46	40	42	40	6	
LR					74		82		46		
OR		74		82		46					

Figure 2.15

Part E Matrix

PART H

Leadtime = 10 days

Lotting Rule = FPQ for 10 days

On-Hand Inventory = 10 units

Period	19	20	21	22	23	24	25	26	27	28	29
Day	95	100	105	110	115	120	125	130	135	140	145
$DD_{nHG=1}$	10	12	11	12	13	12	11	14	10		
GR	10	12	11	12	13	12	11	14	10		
SR		23									
OH	0	11	0								
NR				12	13	12	11	14	10		
LR				25		23		24			
OR		25		23		24					

Figure 2.16

Part H Matrix

3. Data Files

3.1 Data Requirements

For the Material Requirements Planning system to operate, a large amount of timely data is required. The types of data may vary somewhat depending upon the sophistication of the MRP system. The general input categories include the master production schedule, the bill-of-material, and all the parts information. For the MRP explosion, the planning horizon information includes the time period specifications as well as all the gross requirements, net requirements, etc., per period. After the explosion is complete, some of the output information includes the overdue quantities, the specifications for the order to be placed, the demand requirements for the lower levels, etc.

For the basic MRP system designed, the data within the general categories is as follows:

A. Master production schedule

1. Dates when master production schedule items are needed.
2. Quantity of master production schedule items needed.
3. Dates when external order items are needed.
4. Quantity of external items needed.
5. Dates when service part items are needed.
6. Quantity of service part items needed.

B. Bills-of-material

1. Parent part made.
2. The child parts required to make parent part.
3. The quantity of child parts per parent part.

C. Part Information

1. Part number.
2. On-hand inventory quantity.
3. Scheduled receipts-the data and quantity expected.
4. Lead time.
5. Lotting technique plus any additional information needed to perform the technique.
6. Level number assigned to part.

D. Planning horizon

1. Number of periods.
2. Size of periods.
3. Dates associated with period.

E. During MRP explosion for each time period.

1. Master production schedule quantity
2. External order quantities
3. Service part quantities
4. Dependent demand quantities
5. Gross requirements
6. Scheduled receipts
7. On-hand inventory quantity
8. Net requirements
9. Lot size requirements
10. Order release requirements

F. After MRP explosion

1. Past due
2. Dependent demands
3. Specifications for order to be released.

The management of the data is a complex task because of its quantity and its timeliness. The inventory system is very dynamic and much of the data is always in need of updating. Accuracy of the information is also important, since garbage-in results only in garbage-out. In designing the software, the data is to be managed with direct access files stored onto the disks and with interactive programs used for editing the file.

J. Orlicky recommends that "...The file management functions ...consist of the following:

- File creation
- File organization
- Providing for file access
- File updating (i.e., transaction processing)
- File maintenance
- Establishing inquiry capability
- Report generation"[43]

"An MRP system's files (database) in general, and the inventory status records in particular, contain a wealth of information that provides an opportunity for extracting or further processing the data for a whole spectrum of possible outputs,"[44] observes J. Orlicky. Therefore, the goal in designing the files is ease for the extraction of information from the files.

Pertaining to design considerations, J. C. Hershauer and R. D. Eck feel, "To efficiently and effectively support corporate planning, it is suggested that computer software for an extended MRP system be developed to satify the following criteria:

1. Rapid and Easy System Modification...
2. Extensive What-If Capabilities...
3. Rapid and Easy Report Modification..."[45]

Then, A. K. Kochhar recommends that "...The essential requirements of a database system are that the data should be organized and integrated so that they represent the natural relationships between various data items, and can easily be accessed and used. Also the database should minimize the duplication of data..."[46]

3.2 Interconnections Between the Files*

The MRP system data is arranged into the direct access files:

1. bom
2. partid
3. partstatus
4. SR
5. horizon
6. DD
7. MPS
8. XO
9. SP

The bom data file contains the information for the bills-of-materials. The partid data file has the part numbers listed. The partstatus data file has a record of information for each part, which includes on-hand inventory, lead time, lotting rule, the pointers to additional information, etc. The SR data file has the scheduled receipt records. The horizon data file contains a record for each period in the planning horizon. This file is the storage location for the gross requirements, net requirements, etc., during the MRP explosion. The DD data file contains the records of dependent demands imposed by the parent part onto the child parts. The MPS data file contains the master production schedule (end-item) demand records. The XO data file contains the external order demand records. The SP data file contains the service part demand records.

To minimize the duplication of data, pointers have been used to connect the data pertaining to a given part number. Also, to reduce storage space requirements, the long company part number is assigned a system (or computer) part number. If a company part number is 10 alphanumeric digits long, then there is a possibility of 36^{10} (over 3.66×10^{15}) different part numbers. Suppose, there are less than a 1000 parts in inventory. Then the part can be identified with a 3 digit number rather than the longer 10 digits.

The connection between the partid data file and the partstatus data file is the system number. If the company part y is assigned a system

*Note: The file names are underlined; the subroutine names are placed in quotes; and the variable names are underlined. Also, a variable of A is not the same as a in Waterloo Fortran.

part number x, then the company part number and the system part number are stored on record number x in the partid. On record number x in the partstatus data file, additional information about the company part number y is stored.

The partstatus data file has 2 links to the records in the bom data file. The first link is with the BofM quantity. The BofM, stored on the record, is the record number link to where the part appears in the bom child parts list. The BOM, stored on the record in the partstatus data file, is the record number link to where the part appears in the bom parent parts list.

As well as links from the partstatus record to the bom data file, there is linkage from the bom record to the partstatus data file. On each bom record, there is a system part number for the parent part and a system part number for the child part. The system part number is equal to the record number in either the partid or partstatus data file, where additional information about either the parent part or child part is obtained.

The pointer SR, found on the partstatus record x, is a record number in the SR data file where a scheduled receipt record for the part with system number x is found. If the pointer SR has the value of -1, then the part with system number x does not have any scheduled receipts on file.

The pointer MPSkey, found on the partstatus record x, is a record number in the MPS data file where the master production schedule demand record for the part with system number x is found. If the pointer MPSkey has the value of -1, then the part with system number x does not have any master production schedule demands on file.

The pointer xorderkey, found on the partstatus record x, is a record number in the XO data file where an external order demand record for the part with system number x is found. If the pointer xorderkey has the value of -1, then the part with system number x does not have any external order demand on file.

The pointer DDkey, found on the partid record x, is a record number in the DD data file where the order release requirements for the part with system number x is found. The quantity of order release requirements is used to determine how many dependent demand (or child) parts are needed to satisfy the required number of parent parts. If DDkey = -1, the parent part with system number x has not placed any dependent demand requirements to be filled.

The horizon does not have any pointers to or from the other data files. During the MRP explosion, the data, required by the part being processed, is stored on the records in the data file.

Note: The pointers for the different data files are summarized in Figure 3.1.

3.3 Data Files

3.3.1 Overview

Each of the data files, described in the Section 3.2, is created interactively with specially designed program files and is a random access file. Much of the time, the data required by a computer program is supplied on data cards. In this case, the data cards for the MRP explosion are the records in the random access data files.

With the use of interactive programs, the MRP user doesn't create the records directly but indirectly. The data entry is done through the responses to the question posed by the computer programs. The question appears on the CRT screen and the response is typed into the keyboard. The MRP user specifies the information to be stored and the computer does the actual storing. Whatever pointers and keys for data manipulation are required, they are determined by the computer program.

To use the open, close, read or write commands, the file requires a unit number. Each one of the files is assigned a unique unit number. See Figure 3.2.

3.3.2 Data Files bom

The bills-of-materials are stored into the data file bom. A summary of the information in this file appears in Figure 3.3. This file is edited with the use of BOMfile. Upon record Ø, the value for next appears, which is the number of the next empty record in the file.

On each of the other records in the bom, there are six pieces of information. The variables most often associated with the information are parent (or p), child (or c), qty (or q), pkey, ckey and stack (or s). The value of parent (or p) is the system part number for the parent part or the part to be made. The value of child (or c) is the system part number for the child part or the part needed to make the parent part. The value of pty (or q) is the number of child parts required for each parent part.

The pkey value is the key to the next child part of the parent part. In other words, if parent part x is on record j with its child part y, then the value of pkey on record j is equal to a bom record number where

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

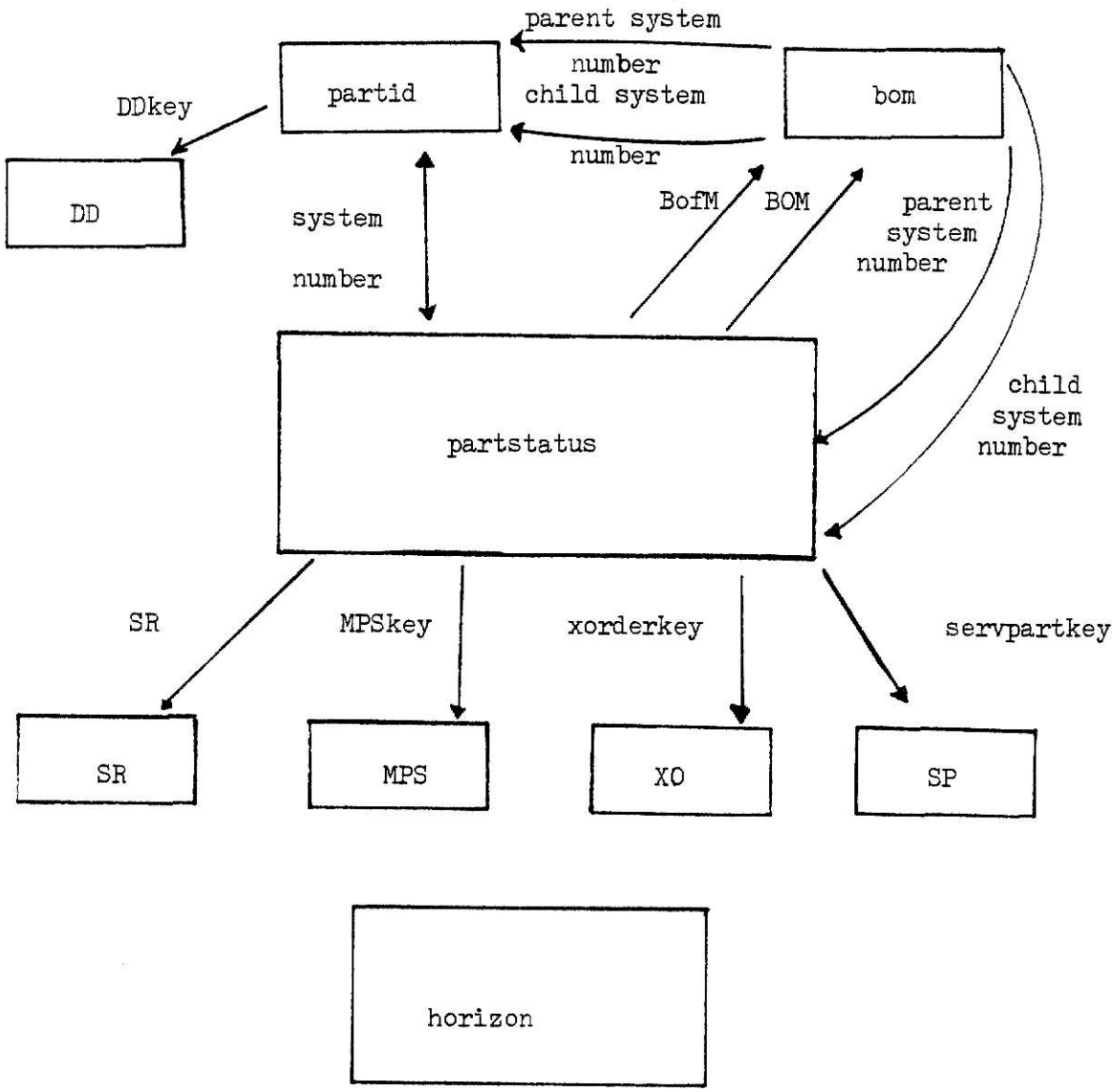


Figure 3.1

File Links

File Unit Number	File Name	Record Length
1	bom	27
2	partid	35
3	partstatus	100
4	SR	18
5	'CRT'	—
6	'printer'	--
7	horizon	98
8	DD	17
9	MPS	18
10	XO	18
11	SP	18

Figure 3.2

Unit Numbers and Record Lengths

Unit = 1

File = '(f:27)bom,rel'

Record Length = 27

Record: Ø

Heading: 'Bills-of-Material Data File'

Format code:	Variable	Use/notes
i4	next	next available record

Record: 1-9999

Format code:	Variable	Use/Notes
i3	parent, p	System # of parent part
i3	child, c	System # of child part
i4	qty, q	Number of child parts/parent part
i4	pkey	Key to other children with the same parent
i4	ckey	Key to other parents with the same child
i4	stack,s	Yes = 1; no = 0; used in level assigning algorithms, initialized at 1

Figure 3.3

Data File bom

parent part x appears again, but with child part ($z \neq y$). If pkey = -1, there are no other child parts linked with the parent part x. The value of pkey is assigned to the bom record by the program BOMfile.

The ckey value is the key to the next parent for the child part. In other words, if child part x is on record j with its parent part y, then the value of ckey on record j is equal to a bom record number where the child part x appears again, but with parent part x ($x \neq y$). If ckey = -1, there are no other parent parts linked with the child part x. The value of ckey is determined by the subroutine "childkeys" in the BOMfile.

The stack (or s) value is used by the Level program for the purpose of assigning level number to each part. The initialized value is one. This denotes that the bom record is still considered in the bom list. If the value is changed to zero, then the bom record has been eliminated from the bom list.

The bom data file for the sample problem appears in Figure 3.4. Note: the stack value does not appear because the Level program where it is initialized has not been run.

3.3.3 Data File partid

Both of the parts lists are stored into the partid file. A summary of the information on this file appears in Figure 3.5. This file is edited by means of the PartFile. On record Ø, the values of first, next, mid and add are stored. On each of the other records, the values of c, s, d, p, u d, and DDkey for each part number are stored.

Since the company part numbers are not stored in order, they are sorted by the program PartFile. Instead of creating a second file with

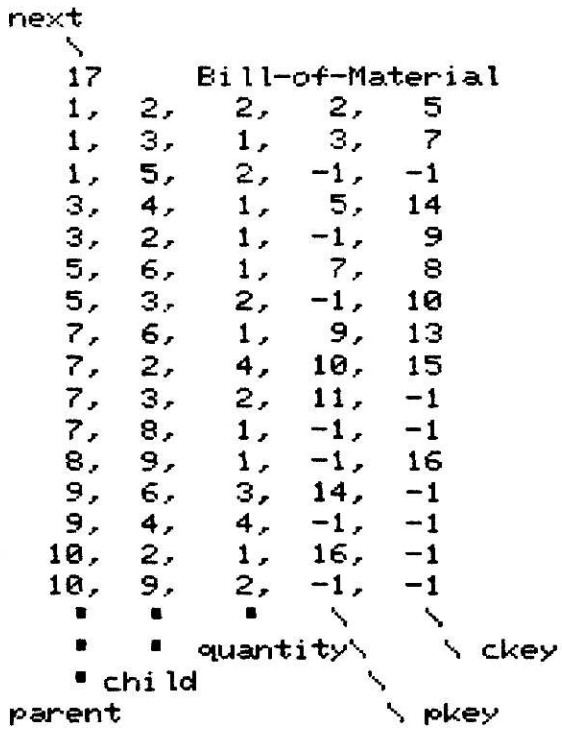


Figure 3.4
Sample Problem bom File

Unit = 2

File = '(f:35)partid,rel'

Record length = 35

Record: Ø

Heading: 'Partid Data File'

Format Code	Variable	Use/Notes
i3	first	Part with ordinal number one
i3	next	Next available record
i3	mid	Midpoint; binary search starting point
i3	add	Where new additions started

Record: 1 - 999

Format Code	Variable	Use/Notes
a10	c, num	Company or user number
i3	s, sys	Computer assigned or system number
i3	key, k	Location of next part in ordered sequence
i3	p, pos	Ordinal number of the part
i3	up, u	Up directional key for binary search
i3	down, d	Down directional key for binary search
i4	DDkey, ddkey	Pointer to dependent demands produced by part; initialized at -1

Figure 3.5

Data File partid

the company part numbers listed in order, their order is established by the pointer first and directional key k. The pointer first, stored on record Ø, is equal to the record number for the company part number with ordinal number one. Each of the records for the company part numbers has a directional key k which points to the next company part number in the sorted queue.

The value of next, on record Ø, is equal to the next empty or available record in the partid file. The value of add, on the record Ø, is equal to the recrod number where the first addition was made. The record number to be initialized for the partstatus data file are the record numbers add to next -1.

Since the company part numbers (c) tend to be long, to conserve space the company part number has a system part number (s or sys) assigned. But, since the company part numbers are recorded in a random order, the problem of how to find the system part number for a specified company part number is created. One method is to read each record in the partid data file one at a time, but this is a slow process. An alternative method is a binary search.

The binary search is the algorithm to be used to convert the company part number, x, into its system part number, x_{sys}. The binary search algorithm involves the following steps:

Step 1. Set record number, R, equal to mid, the midpoint of the company part number list.

Step 2. Read the part information on record number, R. The information includes z, the company part number, z_{sys}, the system part number assigned to z, z_{up}, the up directional key assigned to z; and

z .pa down the directional key assigned to z.

Step 3. Compare x and z.

If x = z, then $x_{sys} = z_{sys}$; stop.

If x < z, then $R = z_{up}$; go to Step 2.

If x > z, then $R = z_{down}$; go to Step 2.

The values of mid, p, u and d are all used to perform the binary search. The value of mid, stored on record Ø, is equal to the midpoint of the file. The p value is equal to the ordinal number or position in line for the company part number. The u and d values are the up branch keys and the down branch keys for the binary search.

The DDkey or ddkey variable is equal to the record number of dependent demands or order release requirements created by part c and stored into the DD data file. The value of DDkey is initialized and determined by the program file MRP. If DDkey is equal to -1, then part c doesn't have any order releases requiring dependent demand parts.

The partid data file for the sample problem appears in Figure 3.6. Note: the DDkeys, assigned by the MRP program file, does not appear.

3.3.4 Data File partstatus

The partstatus data file contains additional part information plus most of the pointers to the other data files. A summary of the information in this file appears in Figure 3.7. This file is initialized by the program PartFile. Part information, such as lead time, and MPS demands, is supplied with the use of the programs DataPlus and DataPlus2. The values of level and levelkey are determined by the program Level.

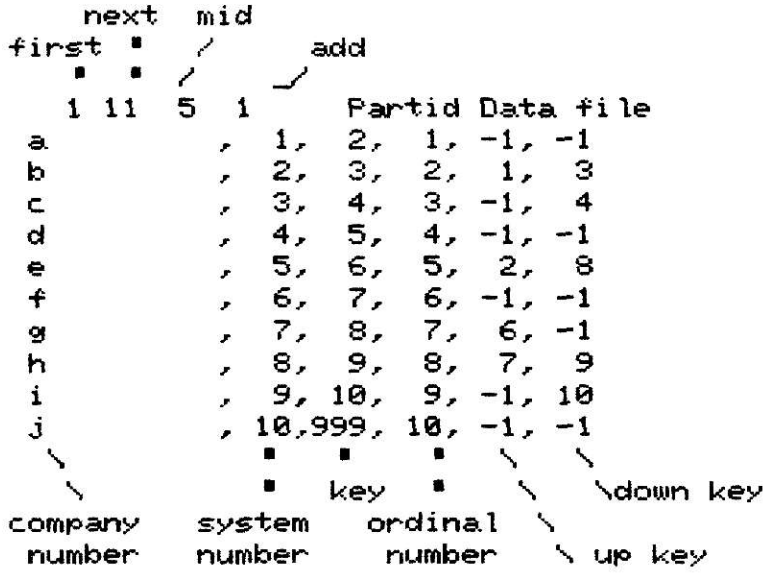


Figure 3.6
Sample Problem partid File

On record Ø, the values of next and first are stored. The value of next is equal to the next available record in the partstatus file. The total number of records in the file is equal to the quantity of next -1. The value of first is equal to the record number or the system part number of the part to be processed first by the MRP explosion.

On each record of the partstatus data file, the variables are the following:

1. oh
2. SR
3. numkids
4. BOM
5. BofM
6. lead
7. lotr
8. lotp
9. scost
10. ucost
11. ccost
12. ccper
13. pu
14. pup
15. MPSkey
16. xorderkey
17. servpartkey
18. level
19. levelkey

Unit = 3

File = '(f:100)partstatus,rel'

Record length = 100

Record: Ø

Heading: 'Partstatus Data File'

Format Code	Variable	Use/Notes
i3	next	
i3	first	

Record: 1 - 999

Format Code	Variable	Initial Value	Use/Notes
f8.0	oh	0	On hand inventory
i4	SR	-1	Pointer to Schedule Receipt Records
i4	numkids	-1	Number of Different children per parent
i4	BOM	-1	Pointer to first appearance as parent in bom
i4	BofM	-1	Pointer to first appearance as child in bom
i4	lead	5	Lead time; expressed in number of days

Figure 3.7

Data File partstatus

Format Code	Variable	Initial Value	Use/Notes
i1	lotr	1	Lotting rule code (1 = L4L; 2 = FPQ; 3 = EOQ)
i4	lotp	5	Number days in lotting period; needed by L4L and FPQ techniques
f8.2	scost	100.0	Setup cost; expressed in dollars and cents
f8.4	ucost	1.0	Unit cost, expressed in dollars and cents
f8.4	ccost	1.0	Carrying cost; expressed in dollars per time unit
i1	ccper	2	Time unit used in expressing carrying cost; (1 = day; 2 = week (5 days); 3 = month (30 days); 4 = year)
f8.0	pu	100.0	Periodal usage of part; expressed in number of units per time unit
i1	pup	4	Time unit used in expressing periodal usage; (1 = day; 2 = week; 3 = month; 4 = year)
i3	MPSkey	-1	Pointer to MPS or end item demand records
i3	xorderkey	-1	Pointer to external order demand records
i3	servpartkey	-1	Pointer to service part demand records
i3	level	-1	Part's level number
i3	levelkey	-1	Key to next part to be exploded

Figure 3.7 (continued)

The value oh is equal to the on-hand inventory quantity. That is, oh is the quantity of parts in the stockroom. The value of SR is equal to the record number in the SR data file where the scheduled receipts can be found. If SR = -1, there are no schedule receipts expected for the part.

The value of numkids is equal to the number of different components (or child parts) used to make the assembly (or parent part). If the value of numkids = -1, the part is either raw material or a purchased part since the part has no child parts.

Both BofM and BOM are pointers to records in the bom data file. The value of BofM is a record number where the part appears in the child parts list. The value of BOM is a record number where the part appears in the parent parts list. If either BofM or BOM are equal to -1, then the part doesn't appear in either the child or parent parts list, respectively.

The value of lead is equal to the lead time for the part. The lead time is to be expressed in number of days. The time unit days is used because the periods in the planning horizon may vary in size.

The value of lotr is equal to the lotting rule code which may be any integer from 0 to 9. The lotting rules, used to determine lot size, are assigned a number. That number is then stored onto the partstatus record. The lot-for-lot technique has been assigned the number one and the fixed period quantity technique has been assigned the number two. Since the fixed period quantity requires the period length for lot coverage to be specified, the number of days for covered is stored into the value of lotp.

Lotting techniques, such as Economic Order Quantity, added at some future date will require the values for scost, ucost, ccost, ccper, pu and

pup. The value of scost is equal to the setup cost ($/setup). The value of ucost is equal to the unit cost ($/unit). The value of ccost is carrying cost ($) of the unit over some time period. The code for the time period, used to express ccost, is stored into the variable ccper. The values for ccper are 1 for day, 2 for week, 3 for month, and 4 for year. The value of pu is the periodal usage rate (number of units/time period). The code for time period, used to express pu, is stored into pup. The codes for pup are the same as those for ccper.

The MPSkey, the xorderkey and the servparkey are the pointers to the MPS, XO and SP data files, respectively. The MPS file contains the records for the master production schedule, the end-item, demands. The XO file is the storage file for the external order demands. The SP file has the service part demands. If the pointer is equal to -1, the part doesn't have any records in the data file being pointed to.

The value of level is the part level number assigned to the part. The value of levelkey is equal to the record number of the next part to be processed by the MRP explosion.

The partstatus data file for the sample problem has been created and is shown in Figure 3.8.

3.3.5 Data File SR

All of the scheduled receipts are stored into data file SR. A summary of the information, stored in this file, appears in Figure 3.9.

On record Ø, the values of next and fillin are stored. The value of next is equal to the next available empty record in the SR data file. The number of records in the file is equal to next -1. From the process of

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

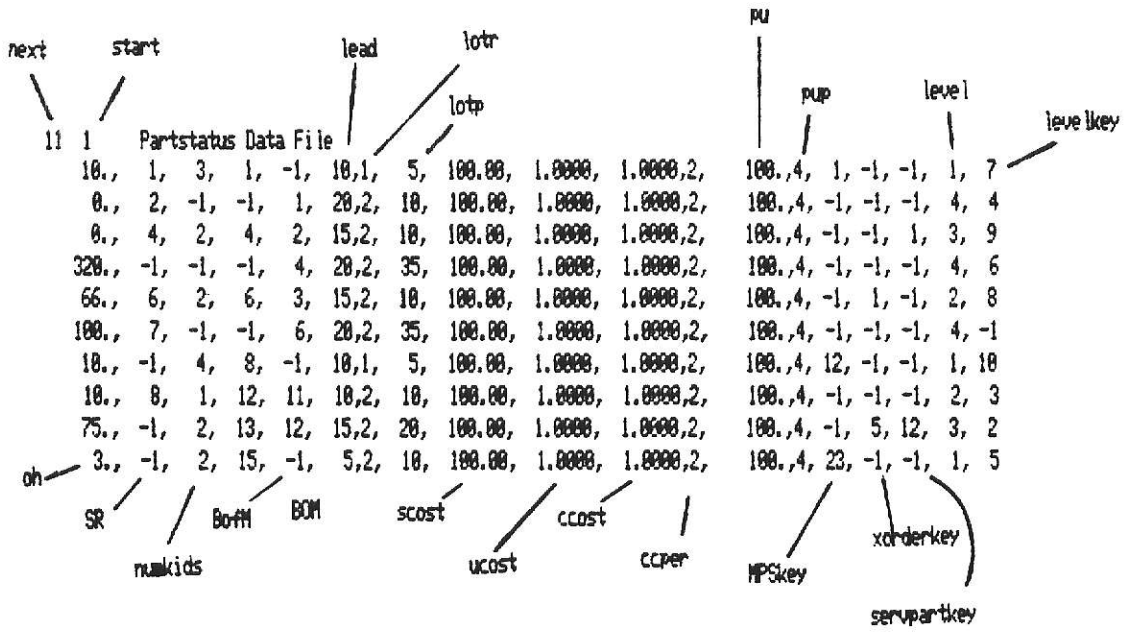


Figure 3.8
Sample Problem partstatus file

Unit = 4

File = '(f:18)SR,rel'

Record length = 18

Record: Ø

Heading: 'Scheduled Receipt Data File'

Format Code	Variable	Use/Notes
i4	next	Next available record; last record plus one
i4	fillin	Key to empty record in middle of file

Record: 1 - 9999

Format Code	Variable	Use/Notes
i4	day	Date when scheduled receipt is expected
f8.0	demand	Quantity of parts on order
i4	key	Key to next scheduled record for given part

Figure 3.9

Data File SR

deleting records, the value of fillin is established. In other words, the value of fillin is equal to the record number of a record in the SR file where new data may be written.

On each record, the values for a specific scheduled receipt is listed. The values include day, demand and key. The value of day is the date from the shop calendar when the scheduled receipt is expected. The value of demand is the quantity of the parts ordered.

The key values is the key to the next scheduled receipt for the part. If key = -1, there are no more scheduled receipt records in SR for the part. Or, the value of key may be equal to the record number for the next SR record where a scheduled receipt has been deleted.

The scheduled receipt of the sample problem appears in Figure 3.10.

3.3.6 Data File horizon

The horizon file is the data file used to process the part's requirements by the MRP explosion. A summary of the information for the file is in Figure 3.11. A sample of the file is shown in Figure 3.12. On record Ø, there is the value for buckets which is the number of periods, or number of records, in the horizon. Each record in the horizon file represents a period in the planning horizon. On each record in the horizon stores the values for the following variables:

1.	period	2.	date	3.	dMPS	4.	dXO
5.	dSP	6.	sumDD	7.	GR	8.	SR
9.	OH	10.	NS	11.	LR	12.	OR

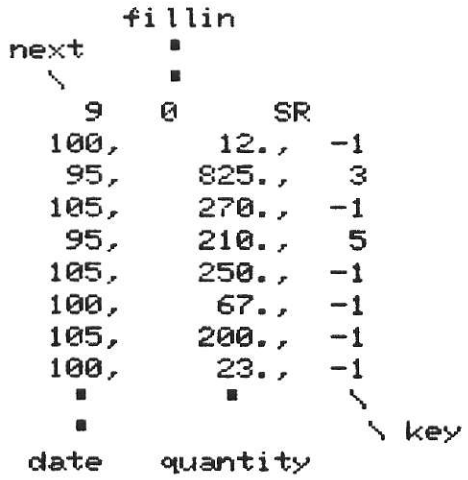


Figure 3.10
Sample Problem
SR file

Unit = 7

File = '(f:98)horizon,rel'

Record length = 98

Record: Ø

Heading: 'Planning Horizon'

Format Code	Variable	Use/Notes
i3	buckets	Number of buckets or planning periods used in horizon

Record: 1 - 999*

Format Code	Variable	Use/Notes
i3	period	Number of bucket or period
i4	date	First day of bucket or period
f8.0	dMPS	MPS or end item demands
f8.0	dXO	External Order demands
f8.0	dSP	Service Part demands
f8.0	sumDD	Total of all dependent demands
f8.0	GR	Gross requirements
f8.0	SR	Scheduled receipt quantity
f8.0	OH	On-hand inventory quantity
f8.0	NR	Net requirements
f8.0	LR	Lot size requirements
f8.0	OR	Order release

Figure 3.11

Data File horizon

*Note: Except for period and date all others have initial value of 0.

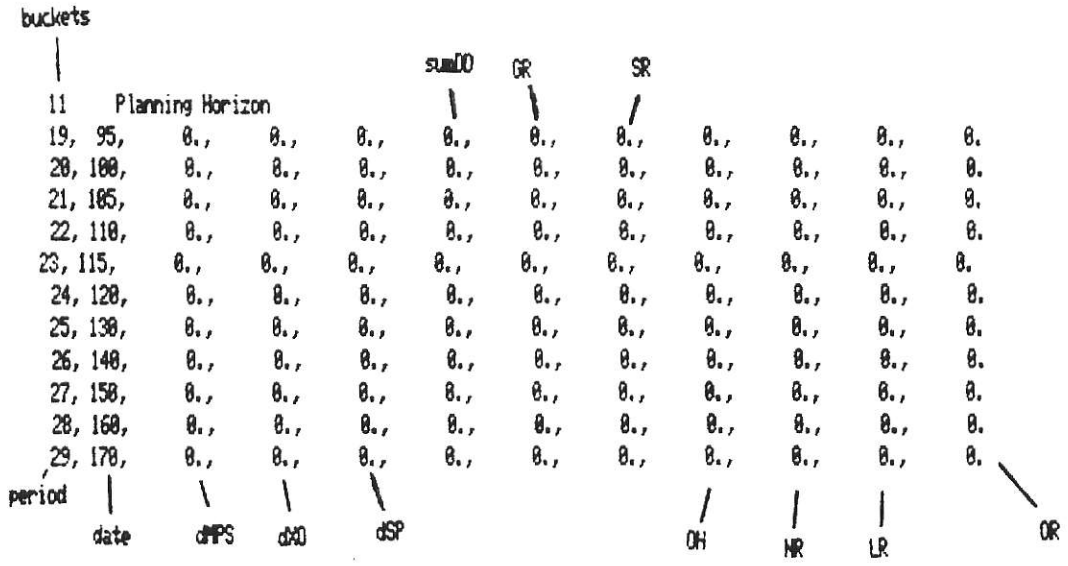


Figure 3.12
Initialized horizon file

The period value is the period number assigned to that record. The date value is the shop calendar date for the first day of the period. All the other variables on the horizon records are initialized to zero.

The value of dMPS is the quantity of master production schedule demands needed. The value of dXO is the quantity of external order demand items needed. The value of dSP is the quantity of service parts needed. The value of sumDD is equal to the total sum of dependent demands.

The gross requirements value is stored into variable GR. GR = dMPS + dXO + dSP + sumDD. The value of SR is equal to the number parts expected in the scheduled receipt. The value of OH is equal to the on-hand inventory quantity. The value of NR is equal to the net requirements quantity. The value of LR is equal to the lot size requirements. The value of OR is equal to the size of the order release.

3.3.7 Data File DD

All the dependent demands (or the size of order releases) are stored into the DD file. A summary of the information stored in the file appears in Figure 3.13. On record Ø, only the name of the file is written. On each of the other records in the file appear the values for the variable part, day, and qDD. The part value is equal to the system part number that is placing the dependent demand. The value of day is the shop calendar date that the dependent demand requirements are due. The value of qDD is equal to the order release quantity. The number of child parts needed is equal to the number of parent parts needed (qDD) times the number of child parts per parent part (q on the bom record).

For sample of the file, see Figure 3.14.

Unit = 8

File = '(f:17)DD,rel'

Record length = 17

Record: Ø

Heading: 'Dependent Demand file'

Record: 1 - n

Format Code	Variable	Use/Notes
i3	part	System number of part placing dependent demands
i4	day	Date when demand is due
f8.0	qDD	Quantity of demands required

Figure 3.13

Data File DD

3.3.8 Data Files MPS, XO, and SP

All the master production schedule, or end item, demand records are stored into the MPS data file. A summary of the information in the file appears in Figure 3.15 and a sample of the file is in Figure 3.18.

All the external order demand records are stored into the XO data file. A summary of the information in the file appears in Figure 3.16 and a sample of the file is in Figure 3.19.

All the service part demand records are stored into the SP data file. A summary of the information in the file appears in Figure 3.17 and a sample of the file is in Figure 3.20.

All three of the data files are set up the same. On record Ø is the value of next which is the next available record in the data file. On the other records, there is the values for day, demand and key. The variable day is the shop calendar date when the parts are needed. The variable demand is equal to the quantity of parts needed. The variable key is equal to the next record number in the data file with a demand record for the same part. If key = -1, there is no more demands for the part.

```
DD file
 1, 135,      20.
 1, 130,      20.
 1, 125,      20.
 1, 120,      18.
 1, 115,      18.
 1, 110,      16.
 1, 105,      16.
 1, 100,      14.
 1,  95,      14.
 7, 135,      10.
 7, 130,      14.
 7, 125,      11.
 7, 120,      12.
 7, 115,      13.
 7, 110,      12.
 7, 105,      11.
 7, 100,      12.
 7,  95,      10.
10, 135,       7.
10, 125,       7.
10, 115,      12.
10, 105,       8.
10,  95,       8.
 5, 120,      46.
 5, 110,      82.
 5, 100,      74.
 8, 120,      24.
 8, 110,      23.
 8, 100,      25.
 3, 125,       4.
 3, 115,      93.
 3, 105,     181.
 3,  95,      43.
 9, 115,      25.
 9,  95,      80.
 .    .       .
 .   date     quantity
part
number
```

Figure 3.14
Sample of DD file

Unit = 9

File = '(f:18)MPS,rel'

Record length = 18

Record: Ø

Heading: 'MPS Demand file'

Format Code	Variable	Use/Notes
i4	next	Next free record

Record: 1 - 9999

Format Code	Variable	Use/Notes
i4	day	Date when MPS or end item demands are due
f8.0	demand	Quantity of parts needed
i4	key	Key to next demand for the part given; last demand denoted by -1

Figure 3.15

Data File MPS

Unit = 10

File = '(f:18)XO,rel'

Record length = 18

Record: Ø

Heading: 'External Order Demand file'

Format Code	Variable	Use/Notes
i4	next	Next free record

Record 1 = 9999

Format Code	Variable	Use/Notes
i4	day	Date when External Order demands are due
f8.0	demand	Quantity of parts needed
i4	Key	Key to next demand for the part given; last demand denoted by -1

Figure 3.16

Data File XO

Unit = 11

File = '(f:18)SP,rel'

Record length = 18

Record: Ø

Heading: 'External Order Demand file'

Format Code	Variable	Use/Notes
i4	next	Next free record

Record 1 = 9999

Format Code	Variable	Use/Notes
i4	day	Date when Service Part demands are due
f8.0	demand	Quantity of parts needed
i4	Key	Key to next demand for the part given; last demand denoted by -1

Figure 3.17

Data File SP

```
next
 ▪
 ▪
 34      MPS
 95,      10.,   2
100,      12.,   3
105,      14.,   4
110,      14.,   5
115,      16.,   6
120,      16.,   7
125,      18.,   8
130,      18.,   9
135,      20.,  10
140,      20.,  11
145,      20.,  -1
 95,       9.,  13
100,       1.,  14
105,      10.,  15
110,      12.,  16
115,      11.,  17
120,      12.,  18
125,      13.,  19
130,      12.,  20
135,      11.,  21
140,      14.,  22
145,      10.,  -1
 95,       3.,  24
100,       4.,  25
105,       4.,  26
110,       5.,  27
115,       3.,  28
120,      10.,  29
125,       2.,  30
130,       3.,  31
135,       4.,  32
140,       4.,  33
145,       3.,  -1
 ▪          ▪    \
 ▪      quantity  \
date               \ key
```

Figure 3.18
Sample Problem
MPS file

```
 next
  ▪
  ▪
  9      XO
105,       5.,    2
120,      10.,    3
130,       2.,    4
140,       6.,   -1
100,       4.,    6
115,       3.,    7
125,       2.,    8
145,       9.,   -1
 ▪          ▪      \
 ▪       quantity   \
date                  key
```

Figure 3.19
Sample Problem
XO file

```
next
  ▪
  ▪
 18      SP
 95,        2.,    2
100,        2.,    3
105,        3.,    4
110,        2.,    5
115,        2.,    6
120,        3.,    7
125,        2.,    8
130,        2.,    9
135,        3.,   10
140,        2.,   11
145,        2.,   -1
 95,        1.,   13
105,        1.,   14
115,        1.,   15
125,        1.,   16
135,        1.,   17
145,        1.,   -1
 ▪          ▪       \
 ▪       quantity    \
date                   key
```

Figure 3.20
Sample Problem
SP file

4.0 Input Programs

4.1 Introduction

The partid, bom, partstatus, SR, MPS, XO, and SP data files are created interactively with the use of the various input programs. The computer interacts (operates) under the direction of the programmer to create and/or edit the numerous data files. The computer asks questions to which the programmer responds by specifying the kind of information to be entered and to type in the appropriate data when it is asked for. Since the systems of pointers and keys are necessary to reduce file storage requirements and to facilitate obtaining data during the explosion, the interactive arrangement is more desirable than the programmer having to create the data file directly.

There are four different input programs. They include the following:

1. PartFile
2. BOMfile
3. DataPlus
4. DataPlus2

The PartFile input program is used to create or append the partid data file and to initialize records in the partstatus data file. (See Appendix B, page 437 for the working file and Appendix A, page 350 for the debugging file.)

The BOMfile input program is called upon to establish and/or edit the bom data file. (See Appendix B, page 442 for the working file and Appendix A, page 358 for the debugging file.)

The DataPlus input program interacts with the programmer to store the lead time, lotting rule, inventory, schedule receipts, and planning horizon information. (See Appendix B, page 450 for the working file and see Appendix A, page 370 for the debugging file.)

For each of the input files, PartFile, Bomfile, etc., two files have been written. One of the files is a working file which is to be run for creating the data files. The other is a fully edited debugging file filled with pertinent comments for understanding the program logic and for debugging, as necessary. The two different files have been created since memory space is critical and comments eat up memory space. The working file has a few comments to help match its sections with the corresponding sections in the debugging file.

4.2 PartFile

4.2.1 Introduction to PartFile

PartFile is the input program which is run first by the programmer to create the parts list stored in the partid file. Then, each time a part needs to be added to the list, the PartFile is run. There is actually a dual parts list in the partid file - the company parts numbers and the corresponding systems parts numbers. With the use of the PartFile input program, the MRP user supplies the company part numbers interactively and the computer assigns the system part numbers. The PartFile program has the responsibility for storing both lists of numbers and then determining the necessary directional keys, the ordinal numbers for each company part number, and the set of up/down keys for the binary search. In addition, for each part listed in the partid file, the PartFile initializes a record

for more information pertaining to the part in the partstatus data file.

In summary, the PartFile has three major functions which are:

(1) to edit (create, append) the partid file;

(2) to determine the different keys (directional and up/down);

(3) to initialize the records in the partstatus file.

4.2.2 Subroutines in PartFile

4.2.2.1 Overview of Subroutines

The PartFile contains 4 subroutines. They include the following:

1. subroutine "setupadd",
2. subroutine "updown",
3. subroutine "key check",
4. subroutine "initialize".

(See Figure 4.1.)

The subroutine "setupadd" is an interactive routine which reads the company part numbers supplied by the programmer; assigns the system number to the company part numbers; and establishes a set of directional keys which link company part numbers with the ordinal number n to the company part number with the ordinal number n+1. The flowchart is in Appendix C, page 504. Also, see section 4.2.2.2.

The subroutine "updown" is employed to determine as many of the up and down keys as possible to make the binary search possible. The up and down keys are established with the use of an absolute distance method. The flow chart is in Appendix C, page 507. Also, see section 4.2.2.3.

The subroutine "keycheck" is called upon by the subroutine "updown" to assign those up and down keys which could not be assigned with the use

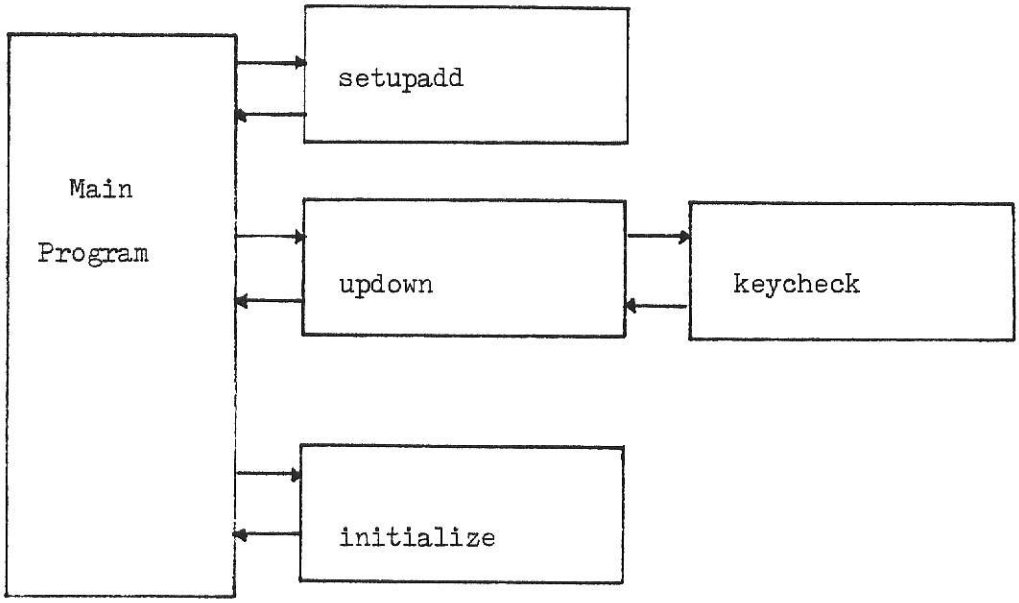


Figure 4.1

PartFile Subroutines

of the absolute distance method. The flow chart is in Appendix C, page 511. Also, see section 4.2.2.4.

The subroutine "initialize" creates a partstatus file record for each of the parts listed in the partid file. The company part number with system number i has its additional part information listed upon the record number i in the partstatus file. For flowchart, see Appendix C, page 513. Also, see section 4.2.2.5.

4.2.2.2 Subroutine "setupadd"

4.2.2.2.1 Purpose

The subroutine "setupadd" serves two major purposes. The first purpose is to create the partid file and the second is to append an existing partid file. In a dynamic manufacturing environment, new products are always being phased into the system so the ability to include additional parts without having to recreate the entire parts list is essential.

When the subroutine "setupadd" asks "new or add?", the user specifies the file's status. Then, the computer obtains the next system number; locates the next available record; etc. Now the computer responds with 'Part? or quit' and the computer part numbers are typed into the keyboard. The computer stores the parts as entered into the partid data file. Each of the company (or user) part numbers is assigned a system (or computer) number by the subroutine.

To obtain a successful MRP system, the duplication of part numbers should not exist. The "setupadd" subroutine checks each new part number entered to make sure that it is not already listed in the partid file.

The manipulation of part information requires that the computer be able to convert from the part's company number to its system number. To read sequentially through the list is a slow process. A better alternative is to use a binary search method. The system of search keys is started in the "setupadd" subroutine. The company part numbers are ordered alphanumerically with numbers first (i.e., 0, 1, 2, ... 9, a, b,

directional key which points to the company part number with ordinal number one; the company part number with ordinal number one has a directional key which points to the company part number with ordinal number two, and company part number with ordinal number n points to the company part number with ordinal number n + 1. The last company part number in the ordered queue has a directional key of 999.

4.2.2.2.2 Variables

add — Integer type; range of values 1 to 999. The value of add is stored on record Ø of the partid file. The add variable is the record number of the first new addition to the partid file. To initialize records in the partstatus file, add is the starting point.

answer — Character type; answer ε {new, add}. The variable answer is used to store the response to question 'new or add?' The answer relays what decision (or patch) to make for the initial values of first, next, etc.

first — Integer type; range of values 1 to 999. The value of first is stored on record Ø of the partid file. The variable first is equal to the record number of the company part

number with ordinal number one.

key — Integer type; range of values 1 to 999. The key value is equal to the record (location) number where the read is to take place. From this location, the values for variables - oldnum, oldsys, oldkey - are obtained.

mid — Integer type; values range from 1 to 999. The record number, mid, is the first stopping point for the binary search. The value of mid is stored on record Ø.

newkey — Integer type; range of values 1 to 999. The value of newkey is equal to the record number of the company part number with ordinal number n + 1. The newkey quantity is stored on the record of newnum, which has ordinal number n.

newnum — Character type; maximum of 10 alphanumerical digits. The newnum is the name of the variable where the company part number for the new addition to the parts list is stored.

newsys — Integer type; the range of values is 1 to 999. The system number, newsys, is assigned to the newnum. The values of newnum and newsys are stored on the same record in the partid file. Record number = system number.

next — Integer type; range of values 1 to 999. Record Ø is the storage location of variable next. The location of the next empty (or available) record is equal to next.

old — Integer type; takes on values Ø to (next -1). The number of reads done before the newkey value is obtained and stored into the variable old.

oldnum — Character type; 10 digits maximum. The company (or user) part number on record key is stored in oldnum. The variable is used for the comparison when determining the assignment value for newkey.

oldkey — Integer type; values range from 1 to 999. The record number of the company part number of ordinal number n + 1 is placed in the variable oldkey. The next record number with company part numbers for comparison is oldkey.

oldoldkey — Integer type. The range of values is 1 to 999. The record location of company part number with next ordinal numbers is retained by this variable. The oldoldkey is part of the set of numbers compared at record = oldoldkey.

oldoldnum — Character type; maximum of 10 alphanumeric digits. The variable oldooldnum is the company part number which was previously used for comparison. In an ordered queue of company part numbers, oldoldnum would appear before oldnum.

oldoldsys — Integer type; values range from 1 to 999. The system number assigned to oldoldnum is oldoldsys. Also, oldoldsys is the record number of the three variables - oldoldnum, oldoldsys, oldolskey.

oldsys — Integer type; range of values 1 to 999. The system number associated with oldnum is oldsys. The value of oldsys is equal to the record number with the three variables - oldnum, oldsys, oldkey.

4.2.2.2.3 Description of Flow*

The flow of the subroutine "setupadd" begins with the user being asked 'new or add?' The response is typed in at the keyboard and stored into the variable answer. If the value of answer is "new"; the assumption that a new file is to be created is made and the variables first, next, mid and add are set equal to one. All four of these values are stored onto record Ø of the partid file. The variable newsys, the system number, is initialized at zero, and newkey, the direction key, is assigned 999. The last company part number in the ordered queue is assigned the directional key of 999.

If the variable answer is "add", the assumption is that a partid file already exists and the status of the file is obtained from record Ø. Record number Ø supplies the values for first, next, mid, and add. The value of add is changed to next. The variable add keeps track of the record number where the first addition to the file is made so that records in the partstatus file can be initialized without destroying the information on the existing records. The newsys value is assigned next -1.

Now 'Part? or quit,' appears on the CRT, the computer is ready for the parts to be typed in. The name of the new part is stored into the variable newnum. If newnum = 'quit', the assumption is made that no more parts are to be listed and control returns to the main program. If newnum ≠ 'quit', the values of newsys and next are incremented by one. The new next value is stored on record Ø. Then, the search for the newkey value

---

* For flowchart, see Appendix C, page 504.

is begun. If newsys = 1, only one part has been submitted and no ordering of company part numbers is necessary. Let newkey = 999.

If newsys ≠ 1, the company part number, newnum, is compared with the company part numbers already in the partid data file to determine its place in the line. The record number, first, is read and the information stored into oldnum, oldsys, and oldkey. The values newnum and oldnum are compared for order. If newnum = oldnum, the newnum is not entered into the file since duplication of part numbers is not permitted; and the message "part number has already been used to name a part!!!" is sent to the CRT. If newnum > oldnum, the search continued to record number oldkey. The variables oldnum, oldsys and oldkey are stored for future use into the variables oldoldnum, oldoldsys, and oldoldkey, respectively. The search continues until either newnum < oldnum or oldkey = 999. If newnum < oldnum and new = first, newnum is now first in the queue; newkey = oldsys and first = newsys. If newnum < oldnum and newkey ≠ first, i.e., oldoldnum < newnum < oldnum, then oldoldkey = newsys and newkey = oldsys. If oldkey = 999, then newkey = 999 and oldkey = newsys; i.e., newnum is now the last part in the queue.

4.2.2.2.4 Example of Nonrandomized File With Directional Keys

The nonrandomized partid data file has been created in stages to the assignment of the directional keys by the subroutine "setupadd". The stages are as follows:

Stage 1. On the CRT screen appears the question: 'Is part file to be established? or appended? yes, no,' the programmer types in "yes" and presses the return key.

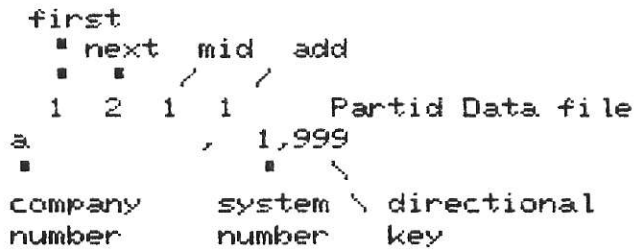


Figure 4.2
Stage 3 partid
With Directional Keys

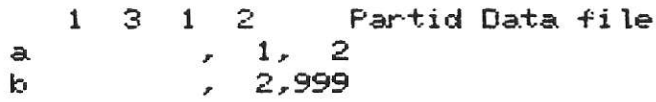


Figure 4.3
Stage 4 partid
With Directional Keys

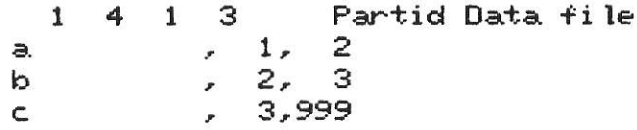


Figure 4.4
Stage 5 partid
With Directional Keys

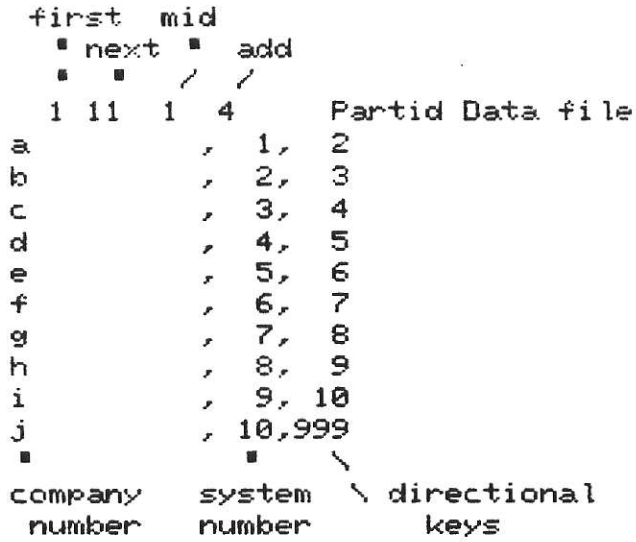


Figure 4.5
Stage 6 partid
With Directional Keys

Stage 2. To the next question: 'new or add?', the response is "new" and the return key is pressed.

Stage 3. Then, "Part? or quit", appears on the screen, the first company part number "a" is entered, and the return key is pressed. A copy of the partid data file, as it now appears, is shown in Figure 4.2. Note: Record Ø contains the values for first = 1, next = 2, mid = 1, add = 1, plus the name of the file. Record 1 has listed on it the company part number "a" with its system number, 1, and direction key, 999.

Stage 4. "Part? or quit," appears and the company part number "b" is entered. See Figure 4.3. Record 2 has been added to the file with the company part number, "b", its system number 2, and directional key, 999. The next on record Ø has been changed to 3. On record 1, the directional key for part "a" is now 2 pointing to part "b" on record 2.

Stage 5. "Part? or quit," gets the response of company part "c" being entered. See Figure 4.4. Record 3 with "c" has been included into the partid data file.

Stage 6. The rest of the company parts a, b, c, .., j are entered in order. Then, 'quit' is typed in as a part number indicating no more company part numbers are to be entered. Before the execution of the subroutine "updown" starts, the partid data file appears as it does in Figure 4.5.

4.2.2.2.5 Example of Randomized File with Directional Keys

The randomized partid data file has been created in stages to show the development of the directional keys by the subroutine "setupadd". The stages are as follows:

Stage 1. Same as in example 4.2.2.4.

Stage 2. Same as in example 4.2.2.4.

Stage 3. "Part? or quit" appears on the screen, the first company part number "e" is entered and the return key is pressed. Note the only difference between Figure 4.2 and Figure 4.6 is that "a" has been replaced by an "e".

Stage 4. "Part? or quit" appears and the company part "i" is entered. Note the only difference between Figure 4.3, and Figure 4.7 is the "a" and "b" have been replaced by "e" and "i", respectively.

Stage 5. "Part? or quit" gets the response of "g" being entered. Figure 4.4, and Figure 4.8 are except for "a", "b", "c" are replaced by "e", "i", "g", respectively; and the directional keys are different. Since "e" < "g" < "i" when ordered, the directional key of "e" points to "g" in record 3, the directional key of "g" points to "i" in record 2; and the directional key of "i" is 999, which means that it is the last company part number in the ordered list.

Stage 6. "Part? or quit" appears and the company part number "d" is entered. See Figure 4.9. Since the order for the parts is now "d" < "e" < "g" < "i", the first value on record Ø has been changed to 4 pointing to part "d" on record 4. The directional keys have also been changed to indicate the correct order of the parts.

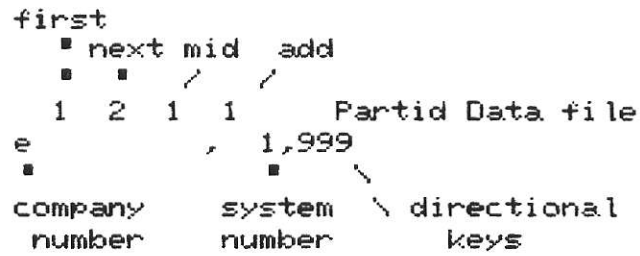


Figure 4.6
Stage 3 Randomized partid
With Directional Keys

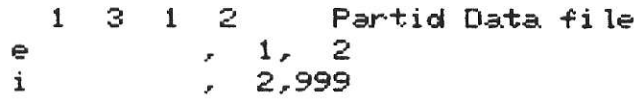


Figure 4.7
Stage 4 Randomized partid
With Directional Keys

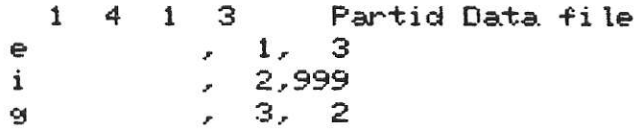


Figure 4.8
Stage 5 Randomized partid
With Directional Keys

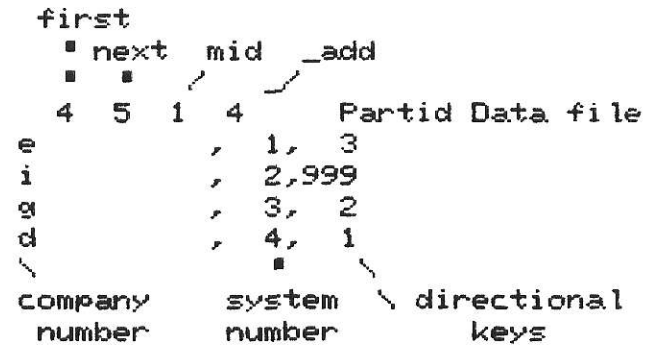


Figure 4.9
Stage 6 Randomized partid
With Directional Keys

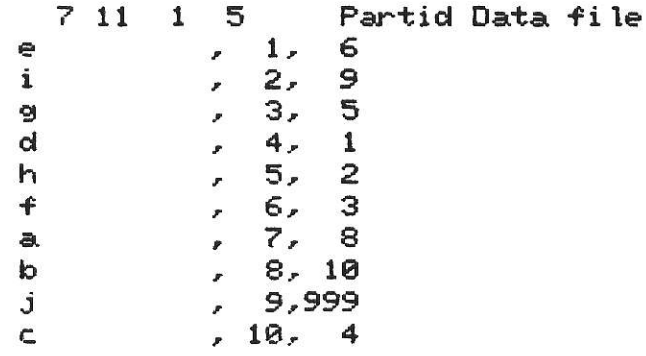


Figure 4.10
Stage 7 Randomized partid
With Directional Keys

Stage 7. The company part numbers are entered as follows: "e", "i", "g", "d", "h", "f", "a", "b", "j", "c" and finally quit. See Figure 4.10. When ordered, "a" on record 7 is first in the list so the value first on record Ø has been changed to 7. The directional keys are such that "a" points to "b", "b" points to "c", ..., "i" points to "j", and "j" is last with 999.

4.2.2.3 Subroutine "updown"

4.2.2.3.1 Purpose

The main purpose of the "updown" subroutine is to assign the up key and the down key for the binary search. (See Section 3.3.3.) Let x be the company part number being sought and let z be the company part number read from the file, the up key is the record number to try next when $x < z$ and the down key is the record number to try next when $x > z$. To assign the up and down keys, the "updown" subroutine employs an absolute distance method. Then, the subroutine "keycheck" is called upon to determine the remaining keys.

The absolute distance method was developed to determine placement of the branches for the binary tree. The branches of the binary tree split to form two branches generally. For example, the binary tree has a root that separates to form two branches, branch 1 and branch 2; branch 1 splits to form branch 11 and branch 12; branch 2 splits to form branch 21 and branch 22; and so on. (See Figure 4.11.) When applied to the records in a file, the root is the middle record of the file; branch 1 is the 1/4 record of the file; branch 2 is the 3/4 record of the file; etc. (See Figure 4.12.)

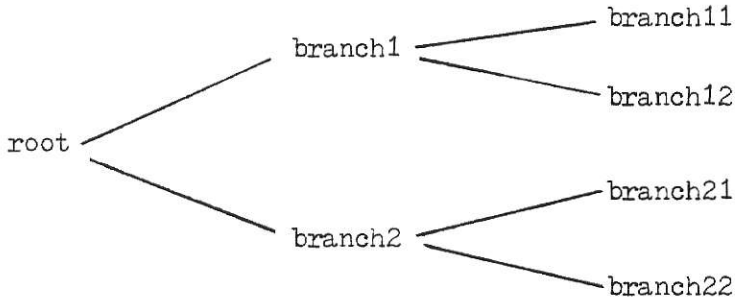


Figure 4.11
Binary Tree

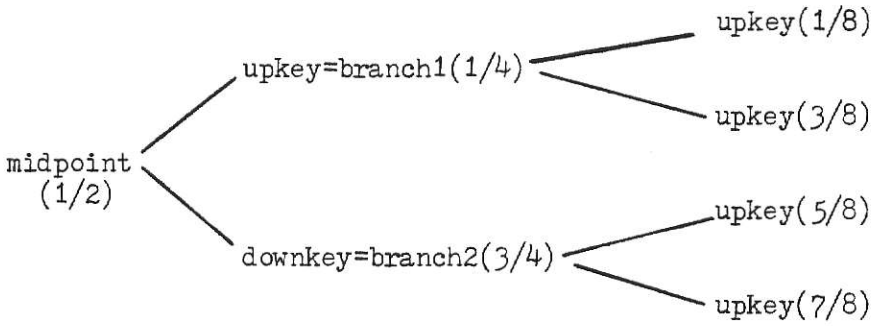


Figure 4.12
Binary File

The midpoint of the file, mid, is equal to one half the number of records in the file, n = next -1. Branch 1, on the up key for the midpoint, is determined by subtracting some distance, dist, from the midpoint; and branch 2, or the down key for the midpoint is determined by adding that same distance, dist, to the midpoint. That distance, dist, is equal to one-half the distance between the midpoint and the start of the file. Branch 11, or the upkey for branch 1, is branch 1 minus some distance, dist.; branch 12, or the down key for branch 1, is branch 1 plus dist; branch 21, or the up key for branch 2, is branch 2 minus dist; and branch 22, or the down key for branch 2, is branch 2 plus dist. For branch 11 to dist is equal to one half the distance between branch 1 and the record of the file. Then, dist is adding and subtracting from branch 1 and 2 to determine the position of their branches. The absolute distance method can continue to be used as long as dist $\geq$ 1 and n $\geq$ 0. See examples 4.2.2.3.4 and 4.2.2.3.5.

Since the company part numbers have not been listed in order, the absolute distance method can't be based on the record number of the part. Instead, the algorithm is based on the part's ordinal number. For example, if there are 10 records in the file and 5 is the midpoint, then the company part number with ordinal number 5 is the midpoint of the file.

4.2.2.3.2 Variables

add — Integer type; range of values 1 to 999. The add variable is the record number of the first addition to the partid file the value of add is stored on record Ø.

dist — Integer; range of values 0 to 250. The variable dist is the absolute distance, or step size, use to position the branches

of the binary tree.

down — Integer type; values range from 1 to 999. The down variable is used to read, store and write the value of the down key, as well as checking the record for assigned up/down keys.

downkey — Integer type; values range from 1 to 999. The ordinal number for the downward branch of the binary tree is downkey. The value pos (downkey) is equal to the location of the company part number with the ordinal number downkey.

first — Integer type; range of values from 1 to 999. The variable first is the key that points to the record containing the company part number with ordinal number one. Record Ø is the storage locaton for first.

i — Integer type; i is the index variable for all four do loops.

j — Integer type; values range from 1 to 1000. The variable j is equal to the number of branches, which will be assigned up/down keys.

jexp — Integer type; values range from 0 to 10. The value of jexp is equal to the power to which 2 is raised for determining the numbers of branches which will have up/down keys assigned. The value of jexp is initialized at zero and is incremented by one after each set of calculations.

k — Integer value; values range from 1 to 999. The variable k is used to read, store and write the directional key for the company part number.

key — Integer type; values range from 1 to 999. The variable key is the number of the record read; and is used in the do loop

for assigning ordinal numbers to the company part number. The key points to the company part number with the next ordinal number.

key1 — Integer type; values range from 1 to 999. The value key1 is equal to pos (upkey) and is the number of the record checked for up/down keys.

key2 — Integer type; values range from 1 to 999. The value key2 is equal to pos (downkey) and is the number of the record checked for up/down keys.

m — Integer array (30 x 1); values in the array range from 1 to 999. The new branches are stored in the array m.

md — Integer array (30 x 1); values in the array range from 1 to 999. The old root or branches are stored in the array md. The md(1) branch is used to calculate the absolute distance, dist, value.

mid — Integer type; values range from 1 to 999. The record number, mid, is the starting point for the binary search. The variable mid is the root, midpoint, of the binary tree, and is equal to half the number of records, n, in the file plus 1. The value of mid is stored on record Ø.

n — Integer type, values range from 1 to 999. The variable n is equal to the number of records in the file that need up/down keys. The value is initialized by next -1 and is decreased by j x 2 before each set of calculations are done.

next — Integer type; values range from 1 to 999. The first empty record number in the partid file is equal to next.

num — Character type; maximum 10 alphanumeric digits. The company part number is read, stored and written with the variable num.

p — Integer value; ranges from 1 to 999. The variable p is used to read, store and write the ordinal number for the company part number.

pos — Integer array; 999 x 1; values in the array range from 1 to 999. The record number for the company part number with the ordinal number i is stored into pos(i).

sys — Integer type; values range from 1 to 999. The system part number is read, stored and written with the variable sys.

up — Integer type; values range from 1 to 999. The up variable is used to read, store and write the value of the up key, as well as checking the record for assigned up/down keys.

upkey — Integer type; values range from 1 to 999. The ordinal number for the upward branch of the binary tree is upkey. The value pos (upkey) is equal to the location of the part with the ordinal number upkey.

4.2.2.3.3 Description of Flow*

Before the absolute distance method can operate, numerous variables are initialized, the ordinal number for the company part numbers are determined and the absolute distance is calculated. The "updown" subroutine

* For flowchart, see Appendix C, page 507.

begins by setting the elements of the old branches array, m, and new branches' array, md, to zero. Next, the status of the partid data file is obtained from record Ø which include the variables - first, next, mid, add. Then, with a do loop, the up/down keys are set to zero.

The ordinal number for each of the company part numbers is determined based on the facts that record Ø has the pointer for the company part number with ordinal number one; and the directional key for the part with ordinal number n points to the part with ordinal number n + 1. When the ordinal numbers are being determined, the record numbers for the company part number are stored into the array pos according to their ordinal numbers. The storing of this information is handled by the statement pos(i) = sys, where pos is the array name; i is the ordinal number; sys is the record number of the company part number with ordinal number i. The company part number with ordinal number n has its record number stored in pos (n).

Before the absolute distance method can start, the root or midpoint, mid, of the file is determined. This is accomplished by dividing the number of records in the file plus one by two. For example, if there are 10 records in the file the midpoint is 5. That is the midpoint has the ordinal number 5. The ordinal number is stored into the first element of array md; md(1) = mid = 5; and the location of the midpoint, mid = pos (mid), is stored onto the record Ø of the partid data file.

In order to keep track of the number of records to which up/down keys are assigned and the number of records reached by the up/down keys, the steps are as follows:

Step 1. Initialize jexp to zero;

Step 2. Raise 2 to the power of jexp and store the results in j, the number of records to which up/down keys are assigned.

Step 3. Subtract the number of records reached, $j \neq 2$, from the number of records left to be reached, n, and store the results into n.

Step 4. Test to see if more destinations, number of records, can be reached than are possible, $n \leq 0$

Step 4a. If $n \leq 0$, call the subroutine "keycheck", and return to the main program.

Step 4b. If $n > 0$, continue to Step 5.

Step 5. Calculate absolute distance; ...; replace old branches by new branches.

Step 6. Increment jexp by 1 and go to step.

As long as there are enough records, which can be assigned up and down keys the absolute distance method is used. The absolute distance is dist = (md(1) + 1)/2 where md (1) is the ordinal number closest to zero. If dist < 1, a new distination will not be reached by the absolute distance method; and the subroutine "keycheck" is called.

If dist $\geq$ 1 and n $\geq$ Ø, the absolute distance method begins. First, the ordinal numbers of the up and down keys are determined:

upkey = md(i) - dist

downkey = md(i) + dist

Where upkey is the ordinal number for the up branch for md(i); md(i) is the ordinal number of the old branch (or root); and dist is the absolute distance.

A couple of checks are performed to make sure that the destinations from the keying are acceptable. Check 1: the upkey and downkey to see if they have gone beyond the bounds of the file; upkey $\leq \emptyset$ or downkey $\geq$ next. If either upkey or downkey are outside the bounds, they are brought inside the boundaries of the file. Check 2: does the destination of the upkey or the downkey already have a set of up/down keys; i.e., up $> \emptyset$ and down $> \emptyset$ If yes, the destination of the up or down key is adjusted by moving in one record at a time until a record without up/down keys is found.

After a set of valid keys has been obtained, the ordinal numbers for the new branches are stored into the array m. The record number for the company part numbers are retrieved from the array pos; i.e., key = pos (md(i); upkey = pos(upkey), and downkey = pos(downkey). The upkey and downkey are listed on record number key; and, on record numbers, upkey and downkey, the up and down are changed to -1. Whenever up = -1 or down = -1, it indicates that the binary search can reach that part record.

Next, the new branches in array m replace the old branches in array md. The jexp is incremented by one. Then, again steps are made to determine whether or not the absolute distance method can be used.

4.2.2.3.4 Example of Nonrandomized File with Up/Down Keys

The nonrandomized partid data file has been stopped after certain stages of development to show the creation of the up/down keys for the binary search. The directional keys for the 10 company parts have already been established by the "setupadd" subroutine (see Figure 4.5.) The stages to create the up/down keys are as follows:

Stage 1. All the up and down keys are initialized with the value zero.

Stage 2. The midpoint of the file is determined and its branches are marked by changing their up and down keys to -1.

Calculations:

n = next - 1 = 11 - 1 = 10

mid = (n + 1)/2 = (10 + 1)/2 = 5

md(1) = mid = 5

mid = pos (mid() = 5

jexp = 0

j = 2 * * jexp = 2 * * 0 = 1

n = n - (j * 2) = 10 - (1 * 2) = 8

dist = (md (1) + 1)/2 = (5 + 1)/2 = 3

upkey = md(1) - dist = 5 - 3 = 2

downkey = md(1) + dist = 5 + 3 = 8

The file will appear as it does in Figure 4.13, if the program was stopped at this stage.

Stage 3. The next set of branches is determined.

Calculations:

md(1) = 2

md (2) = 8

jexp = jexp + 1 = 0 + 1 = 1

j = 2 ** jexp = 2 * * 1 = 2

n = n - (j * 2) = 8 - (2 * 2) = 4

dist = (md(1) + 1)/2 = (2 + 1)/2 = 1

upkey = (md(1) - dist = 2 - 1 = 1

downkey = md(1) + dist = 2 + 1 = 3

upkey = md(2) - dist = 8 - 1 = 7

downkey = md (2) + dist = 8 + 1 = 9

The file will appear as it does in Figure 4.14, if the program was stopped at this stage.

Stage 4. The test of $n \leq \emptyset$ is positive so no more sets of branches can be determined by the absolute distance method.

Calculations:

md(1) = 1 md(3) = 7

md(2) = 3 md(4) = 9

jexp = jexp + 1 = 1 + 1 = 2

j = 2 * * jexp = 2 * * 2 = 4

n = n - (j * 2) = 4 - (4 * 2) = -4

Since $n < \emptyset$, control of the program returns to the main program without records "d", "f", and "j" being linked for the binary search.

4.2.2.3.5 Example of Randomized File With Up/Down Keys

The randomized partid data file has been stopped after certain stages of development to show the creation of the up/down keys for the binary search. The directional keys for the 10 randomized company parts have already been established by the "setupadd" subroutine (see Figure 4.10.) The stages to create the up/down keys are as follows:

Stage 1. All the up and down keys are initialized at zero.

Stage 2. The midpoint of the file is determined and its branches are marked by changing their up and down keys to -1.

```
  1 11  5  1      Partid Data file
a          ,  1,  2,  1,  0,  0
b          ,  2,  3,  2, -1, -1 <-  up branch
c          ,  3,  4,  3,  0,  0    ▪
d          ,  4,  5,  4,  0,  0    ▪
e          ,  5,  6,  5,  2,  8 >-  file midpoint
f          ,  6,  7,  6,  0,  0    ▪
g          ,  7,  8,  7,  0,  0    ▪
h          ,  8,  9,  8, -1, -1 <-   down branch
i          ,  9, 10,  9,  0,  0
j          , 10,999, 10,  0,  0
                     ▪   \     \ down keys
                ordinal  \
                 number   \ up keys
```

Figure 4.13
Stage 2 partid
With Up/Down Keys

```
  1 11  5  1      Partid Data file
a          ,  1,  2,  1, -1, -1 <-\  up branch for b
b          ,  2,  3,  2,  1,  3>--▪
c          ,  3,  4,  3, -1, -1 <-/  down branch for b
d          ,  4,  5,  4,  0,  0
e          ,  5,  6,  5,  2,  8
f          ,  6,  7,  6,  0,  0
g          ,  7,  8,  7, -1, -1 <-\   up branch for h
h          ,  8,  9,  8,  7,  9>--▪
i          ,  9, 10,  9, -1, -1 <-/   down branch for h
j          , 10,999, 10,  0,  0
```

Figure 4.14
Stage 3 partid
With Up/Down Keys

Calculations:

n = next - 1 = 11 - 1 = 10

mid = (n + 1)/2 = (10 + 1)/2 = 5

md (1) = mid = 5

mid = pos (mid) = 1

jexp = 0

j = 2 * * jexp = 2 * * 0 = 1

n = n - (j * 2) = 10 -(1 * 2) = 8

dist = (md (1) + 1)/2 = (5 + 1)/2 = 3

upkey = md(1) - dist = 5 - 3 = 2

downkey = md(1) + dist = 5 + 3 = 8

upkey = pos (upkey) = pos (2) = 8

downkey = pos (downkey) = pos (8) = 5

The file will appear as it does in Figure 4.15, if the program was stopped at this stage.

Stage 3. The next set of branches is determined.

Calculations:

md(1) = 2

md(2) = 8

jexp = jexp + 1 = 0 + 1 = 1

j = 2 * * jexp = 2 * * 1 = 2

n = n - (j * 2) = 8 - (2 * 2) = 4

dist = (md (1) + 1)/2 = (2 + 1)/2 = 1

upkey = md(1) - dist = 2 + 1 = 3

upkey = pos (upkey) = pos(1) = 7

downkey = pos (downkey) = pos (3) = 10

upkey = md(2) - dist = 8 - 1 = 7

downkey = md (2) + dist = 8 - 1 = 7

downkey = md (2) + dist = 8 + 1 = 9

drpkey = pos (upkey) = pos (7) = 3

downkey = pos(downkey = pos (9) = 2

The file will appear as it does in Figure 4.16, if the program was stopped at this stage.

Stage 4. Same as Stage 4 in Example 4.2.2.3.4.

4.2.2.4 Subroutine "keycheck"

4.2.2.4.1 Purpose

The absolute distance method for assigning the up and down keys is not always able to determine enough keys to point to all parts record. Therefore, a check for the records not linked and a keying procedure is needed to insure that all parts can be reached with a binary search. The subroutine "keycheck" has the responsibility to make sure that all parts can be reached when using a binary search and if not, make the necessary corrections.

The "keycheck" subroutine starts with the record number 1 of the partid data file and works its way to the last record number, next - 1. If the up key and the down key are equal to zero, then the company part number with the ordinal number, x, has no up or down key pointing to it. The company part number with ordinal number, x - 1, is checked first for the possibility of a key pointing to the company part number, x. If the linking is possible, then the appropriate key numbers are assigned and the next record in the file is checked. If keying is not possible, the company part number

```
  7 11   1   1      Partid Data file
e            ,  1,  6,  5,  8,  5 >----    file midpoint
i            ,  2,  9,  9,  0,  0    ▪ ▪
g            ,  3,  5,  7,  0,  0    ▪ ▪
d            ,  4,  1,  4,  0,  0    ▪ ▪
h            ,  5,  2,  8, -1, -1 <--  ▪    down branch
f            ,  6,  3,  6,  0,  0      ▪
a            ,  7,  8,  1,  0,  0      ▪
b            ,  8, 10,  2, -1, -1 <----    up branch
j            ,  9,999, 10,  0,  0
c            , 10,  4,  3,  0,  0
                        ▪   \     \ down keys
                   ordinal \
                    number   \ up keys
```

Figure 4.15
Stage 2 Randomized partid
With Up/Down Keys

```
  7 11   1   1      Partid Data file
e            ,  1,  6,  5,  8,  5
i            ,  2,  9,  9, -1, -1 <---    down branch for h
g            ,  3,  5,  7, -1, -1 <- ▪    up branch for h
d            ,  4,  1,  4,  0,  0   ▪ ▪
h            ,  5,  2,  8,  3,  2>----
f            ,  6,  3,  6,  0,  0
a            ,  7,  8,  1, -1, -1 <--     up branch for b
b            ,  8, 10,  2,  7, 10>--▪
j            ,  9,999, 10,  0,  0    ▪
c            , 10,  4,  3, -1, -1 <--     down branch for b
```

Figure 4.16
Stage 3 Randomized partid
With Up/Down Keys

with ordinal number, x + 1, is checked for a possible link. If the "key-check" subroutine is unable to link all the parts, a message is sent to the computer operator and then a manual intervention to link all the records may be necessary.

4.2.2.4.2 Variables

down — Integer type; values range from 1 to 999. The variable down is used to read, store and write the down directional key when hunting for a place for a link.

downkey — Integer type; values range from 1 to 999. The downkey is the down directional key for the binary search found on the record that needs to be linked. If downkey = 0, there should be no parts after it in the queue for comparison. If downkey = -1, a binary search does reach the record for comparison. If downkey $\geq$ 1, then there are additional parts in the last part of the queue to which it is linked.

i — Integer type; values range from 1 to 999. The index of the do loop is i which has an initial value of one, has an increment of one, and has a stopping criteria of greater than next -1.

k — Integer type; values range from 1 to 999. The variable k is used to read, store and write the directional key for the company part number.

key — Integer type; values range from 1 to 999. The record number, key, is the location of the company part number which possibly needs a link for the binary search.

key1 — Integer type; values range from 1 to 999. The variable key1 is the record number for the part number with ordinal number, loc -1.

key2 — Integer type; values range from 1 to 999. The variable key2 is the record number for the part number with ordinal number, loc + 1.

loc — Integer type; values range from 1 to 999. The ordinal number of the company part number on record key is loc. This particular company part number is the one that possibly needs to be linked for the binary search.

next — Integer type, subroutine argument; values range from 1 to 999. The number of records in the partid data file is next -1.

num — Character type; maximinim of 10 alphanumeric digits. The company part number is read, stored, and written with the variable num.

p — Integer type; values range from 1 to 999. The variable p is used to read, store and write the ordinal number when hunting for a place for a link.

pos — Integer array; 999 x 1; subroutine argument; values in the array range from 1 to 999. The record number for the company part number with the ordinal number, say loc, can be found in pos (loc).

sys — Integer type; values range from 1 to 999. The system part numbers are read, stored, and written with the variable sys.

up — Integer type; values range from 1 to 999. The variable up is used to read, store and write the up directional key when hunting for a place for a link.

upkey — Integer type; values range from 1 to 999. The upkey is the up directional key for the binary search; is used when checking for linkages. If upkey = 0, there should be no parts before it in the queue. If upkey = -1, a binary search can reach the part for comparison. If upkey $\geq$ 1, then there are additional parts before it in the queue to which to part is linked.

4.2.2.4.3 Description of Flow*

To check for records not linked for the binary search, the subroutine "keycheck" uses a do loop which reads each part record in the partid data file one by one to test whether or not the binary search can reach that record. If the upkey and downkey are equal to zero, then the part record cannot be reached by the binary search. Therefore, the up/down keys need to be appended so that a linkage does exist.

To find a record with which a key may be changed, the procedure is as follows: Given the part has the ordinal number of loc ≠ 1 and loc ≠ last record in the file. The first step: the part with ordinal number of loc - 1 is tested for possible linkage. If down = -1, then down is assigned the record number for the part with ordinal number of loc. If down ≠ -1, then the part with ordinal number of loc +1 is tested. If up = -1, then up on record loc + 1 is assigned the record number for the part with the ordinal number of loc. When linkage can be made, the upkey and downkey for the part with ordinal number of loc is changed to -1. If down ≠ -1 for the part with ordinal number loc + 1, then the subroutine "keycheck will print the message 'not all the records can be keyed' and stop. If this happens, the programmer will need to assign the keys directly.

If the ordinal number for the part is loc = 1, there will not be a part with the ordinal number of loc -1. Thus, only the part with ordinal number loc +1 is tested for possible linkage. If loc is equal to the last number part in the file, next -1, then only a part with the ordinal number loc -1 can provide for possible linkage.

---

* For flowchart, see Appendix C, page 511.

4.2.2.4.4 Example of Nonrandomized File With Up/Down Keys for All Parts

The nonrandomized example 4.2.2.3.4 is fixed by the subroutine "keycheck" so that all parts are linked by up/down keys for a binary search. For the results, see Figure 4.17. Note: the down key of part "c" points to "d"; the up key of "g" points to "f"; and the down key of "i" points to "j".

4.2.2.4.5 Example of Randomized File With Up/Down Keys for All Parts

The randomized partid file example 4.2.2.3.5 is fixed by the subroutine "keycheck" so that all parts are linked by up/down keys for a binary search. For the results, see Figure 4.18. Note: similiar to example 4.2.2.4.4 the down key of part "c" points to "d"; the up key of "g" points to "f"; and the down key of "i" points to "j".

4.2.2.5 Subroutine "Initialize"

4.2.2.5.1 Purpose

The main purpose of the subroutine "initialize" is to establish one record in the partstatus data file for each of new parts listed in the partid data file. For the details on the partstatus data file see Section 3.3.4.

4.2.2.5.2 Variables

add — Integer type; values range from 1 to 999. The first entry of the new parts for the partid data file is the record number add.

first — Integer type; values range from 1 to 999. The record number for the part with ordinal number one is equal to

```
  1 11  5  1      Partid Data file
a          ,  1,  2,  1, -1, -1
b          ,  2,  3,  2,  1,  3
c          ,  3,  4,  3, -1,  4
d          ,  4,  5,  4, -1, -1
e          ,  5,  6,  5,  2,  8
f          ,  6,  7,  6, -1, -1
g          ,  7,  8,  7,  6, -1
h          ,  8,  9,  8,  7,  9
i          ,  9, 10,  9, -1, 10
j          , 10,999, 10, -1, -1
```

Figure 4.17
The partid file
After 'keycheck'

```
  7 11  1  1      Partid Data file
e          ,  1,  6,  5,  8,  5
i          ,  2,  9,  9, -1,  9
g          ,  3,  5,  7,  6, -1
d          ,  4,  1,  4, -1, -1
h          ,  5,  2,  8,  3,  2
f          ,  6,  3,  6, -1, -1
a          ,  7,  8,  1, -1, -1
b          ,  8, 10,  2,  7, 10
j          ,  9,999, 10, -1, -1
c          , 10,  4,  3, -1,  4
```

Figure 4.18
The Randomized partid
File After 'keycheck'

first.

i — Integer type; values range from 1 to 999. The index of the do loop is the variable i. Also, i is the record number of the direct file for writing the initial values.

mid — Integer type; values range from 1 to 999. The first record to be read for a binary search is mid. The variable mid is the record number for the midpoint of the company part number list.

next — Integer type; values range from 1 to 999. The record where the next new part can be written. The number of the records in the file is equal to next -1.

4.2.2.5.3 Description of Flow*

The subroutine "initialize" proceeds as follows: First, the status of the partid data file is checked by reading the record number Ø. The record Ø in the partid data file supplies the values for the variables first, next, mid, add. The values of first and mid aren't of much use to the subroutine "initialize", but next -1, is the number of records needed in both the partid and the partstatus data file; and add conveys the record number where the last new addition was made. In other words, the records numbered from add to next -1 need to be established in the partstatus data file. The value add is used rather than just starting with record number one, so that when new parts are included into the files the information previously stored on the partstatus records will not be destroyed.

*For flowchart, see Appendix C, p. 513.

After the values for next and add are obtained, the value of next is listed on record Ø in the partstatus data file. Then, the records from add to next -1 are established. For the starting values stored on each record, see section 3.3.4.

4.2.2.5.4 Example of An Initialized Partstatus Data File

After the "updown" subroutine and the "keycheck" subroutine are finished with the partid data file, the subroutine "initialize" is run. For either the randomized example 4.2.2.4.5 or the nonrandomized example 4.2.2.4.4, the partstatus data file will be the same. See Figure 4.19.

4.3 BOMfile

4.3.1 Introduction to BOMfile

After the partid file has been created by the PartFile input program, the BOMfile is run to create the bills-of-material (bom) data file. The bom data file is a list of parent parts with their quantity of child parts. The bom file has two internal sets of keys: set one, pkey, links the parent part to all its different child parts; and, set two, ckey, links the child part to all its different parent parts. The parent part, or assembly, places demand requirements onto the child parts, or components. Each line in the bom file includes p, c, q, pkey, ckey: Where p is the parent part; c is the child part; q is the number of c's per 1 p; pkey is the link to other children for parent p; and ckey is the link to the other parents of child c. For more details on the dom data file, see Section 3.3.2.

Since any given part may be both a parent and/or child parts, there are two pointers stored on the part's record in the partstatus data

11 Partstatus Data File 111

```
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
```

Figure 4.19
Initialized partstatus file

file. One pointer, BOM, links to the parts first appearance as a parent part in the bom file. The other pointer, BofM, links to the parts first appearance as a child part in the bom file. If the part is neither a parent part nor child part in the bom file, the BOM or BofM pointers will retain their initial value of -1.

In order to obtain a list of child parts for a given parent part, the procedure is to first read the parts record in the partstatus data file and get the value of BOM. Then, for the first child, read the line in the bom data file with the record number equal to BOM. Next, read the line in the bom data file with record number equal to pkey for the second child part. By continuing to read line (pkey) by line (pkey) in the bom file until pkey = -1, a list of all the children may be obtained. When pkey = -1, the last child part for the given parent part has been reached.

Similarily, to obtain a list of the parent parts for a given child part, first read the parts' record in the partstatus data file. Then, read the line in the bom data file with record number equal to BofM. Next, read the line in the bom data file with record number equal to ckey, and continue reading line (ckey) by line (ckey) until ckey = -1. When ckey = -1, the last parent part for the given child part has been reached

With the BOMfile, the bom data file can be edited in the following ways:

1. to create a new file;
2. to add parent/child combinations;
3. to change the parent part on an existing bom record;
4. to change the child part on an existng bom record;

5. to change the quantity of child parts per parent part on an existing bom record.

4.3.2 Subroutines in BOM file

4.3.2.1 Overview of Subroutines

The BOMfile contains 8 subroutines. They include the following:

1. Subroutine "newbom";
2. Subroutine "childkeys";
3. Subroutine "findsys";
4. Subroutine "addbom";
5. Subroutine "changebom";
6. Subroutine "pchange";
7. Subroutine "cchange";
8. Subroutine "qchange".

See Figure 4.20. The main program of BOMfile requests that the user specify what needs to be done (new, add, change) to the bom file. Depending upon the users' specifications the appropriate subroutines are called.

The subroutine "newbom" is an interactive routine which is designed to obtain from the users' the parent part, the child part and the quantity of child parts per parent part, to assign a link to the next child for the parent part; and to store the information in the bom data file. For flowchart, see Appendix C, page 516. Also, see Section 4.3.2.2.

The subroutine "childkeys" is the subprogram that creates the link between a given child part and all of its parents. The points, BofM, in the partstatus data file is, also, determined by this subroutine. For flowchart, see Appendix C, page 519. Also, see Section 4.3.2.3.

The subroutine "findsys" is a binary search routine that searches ou the given company part number to retrieve its system number. For flowchart, see Appendix C, page 521. Also, see Section 4.3.2.4.

The subroutine "addbom" is an interactive subroutine which obtains additional bill-of-material information, stores the information into the bom data file and adjusts the necessary pointers and keys. For flowchart, see Appendix C, page 523. Also, see Section 4.3.2.5.

The subroutine "changebom" is an interactive program which asks the user to specify how the information presently and how the information should exist in the bom data file. Then, the appropriate subroutines are called to make the changes by the subroutine "changebom". For flowchart, see Appendix C, page 526. Also, see Section 4.3.2.6.

The subroutine "pchange" is called by the subroutine "changebom" whenever the parent part on records in the bom data file needs to be changed. The change of the parent part involves replacing the system number of the new parent for the old parent part as well as fixing the keys and pointers. For flowchart, see Appendix C, page 529. Also, see Section 4.3.2.7.

The subroutine "cchange" is called by the subroutine "changebom" whenever the child part on the records in the bom data file needs to be changed. The change of the child part involves replacing the system number of the new child for the old child as well as fixing the keys and

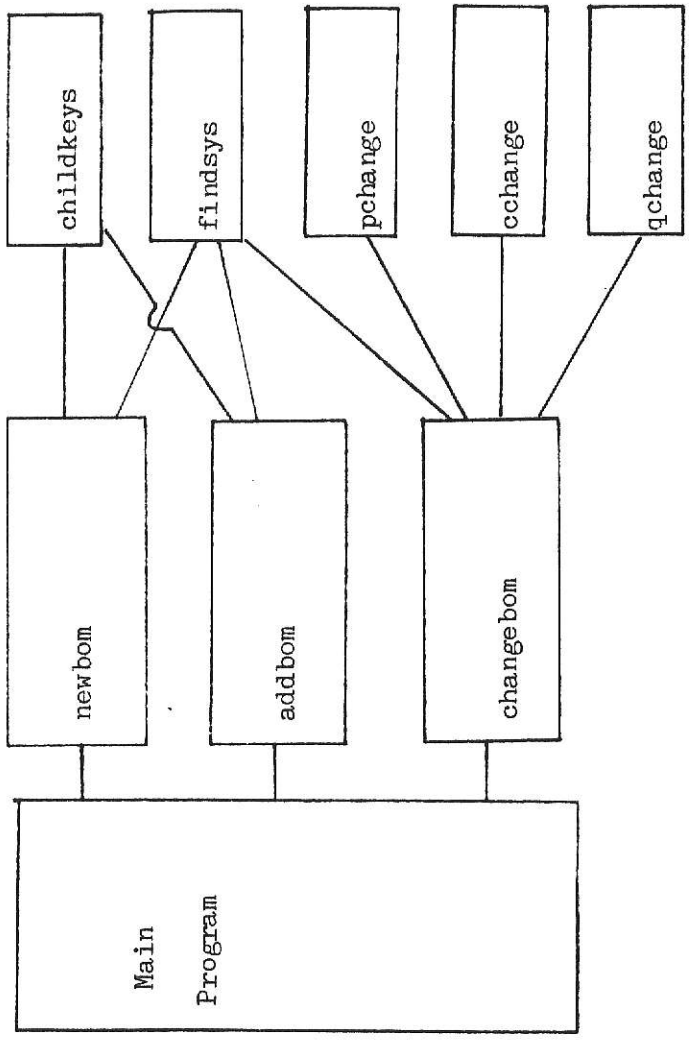


Figure 4.20
BOMfile Subroutines

pointers. For flowchart, see Appendix C, page 532. Also, see Section 4.3.2.8.

The subroutine "qchange" is called by the subroutine "changebom" whenever the quantity of child parts per parent part on the records in the bom data file needs to be changed. For flowchart, see Appendix C, page 534. Also, see Section 4.3.2.9.

4.3.2.2 Subroutine "newbom"

4.3.2.2.1 Purpose

The subroutine "newbom" is an interactive subprogram which initializes the bom data file. The programmer is asked to specify the parent and number of different children the parent has. The programmer is, also, asked to specify the child and the number of child parts per parent. The subroutine "newbom" determines the pkey (the link to all the children for the given part) and BOM (the pointer on the parts' record in the partstatus data file).

Since the part numbers supplied by the programmer are the company part numbers, the subroutine "findsys" is called upon to search out the system part number. After all the bills-of-material information is entered, the subroutine "childkeys" is called upon to determine the ckey's, the link of the child part to all the parent parts', and to assign the BofM's in the partstatus data file, which are used to link the part to its first appearance as a child part in the bom data file.

4.3.2.2.2 Variables

BofM — Integer type; range of values -1 to 9999. The record number of the first appearance of the part in the child's

bom list is stored into the variable BofM.

BOM — Integer type; range of values -1 to 9999. The record number of first appearance of the part in the parent's bom list is stored into the variable BOM.

ccost — Real type; maximum of 8 digits; format code-f8.4. The variable ccost is the carrying cost for the part, expressed in dollars per unit of time.

ccper — Integer type; set of values are {1,2,3,4} where 1 = day, 2 = week, 3 = month, 4 = year. The variable ccper specifies the time unit used to express ccost.

child — Character type; maximum of 10 alphanumeric digits. The company part number for the child part is stored in the variable child.

csys — Integer type; range of values 1 to 999. The system part number for the child part number is read, stored and written with the variable csys.

i — Integer type; range of values 1 to 9999. The do loop index is variable i. The names of the children and their quantities are obtained in the do loop.

key — Integer type; range of values -1 to 9999. The record in the bom data file, where the information is to be listed, is the record number key.

lead — Integer type; range of values 1 to 9999. The lead time for the part is expressed in days and stored in the variable lead.

level — Integer type; values range from 1 to 999. The part's level number is stored in the variable level.

levelkey — Integer type; values range fom -1 to 999. The keys for directing the flow of the explosion is stored into the variable levelkey.

lotp — Integer type; values range from 1 to 9999. The number of days per lotting period is stored in the variable lotp.

lotr — Integer type; set of values {1,2}. The lotting rule to be used to determine lot size is stored in the varible lotr. 1 = Lot-for-lot and 2 = Fixed Period Quantity.

MPSkey — Integer type; range of values -1 to 999. The record number, for the first MPS (end item) demend in the MPS data file, is stored in the variable MPSkey.

nextkidkey — Integer type; range of values -1 to 9999. The location/record number of the next child for the parent given is kept track of with the variable nextkidkey. The last child is denoted by nextkidkey = -1.

num — Integer type; range of values -1 to 9999. The value of num is the dummy variable for reading numkids prior to number being assigned to it.

numkids — Integer type; range of values -1 to 9999. The numkids is the number of different children the parent requires for its assembly. For example, numkids = 3 if 1 part a is made from 2 b's, 1 c and 2 e's.

oh — Real type; maximum of 8 digits; the format code is f8.0. The oh is the variable for the on hand inventory

quantity.

parent — Character type; maximum of 10 alphanumeric digits. The variable parent is the company part number for the parent part.

pBOMkey — Integer type; range of values 1 to 9999. The next available record is the bom data file is stored in the variable pBOMkey. The variable pBOMkey is also used in assigning the value of BOM in the partstatus data file.

psys — Integer type; range of values 1 to 999. The system part number for the parent part number is handled by the variable psys.

pu — Real type; maximum of 8 digits; format code is f8.0. The number of units for the periodal usage per time unit is stored into the variable pu.

pup — Integer type; set of values included {1,2,3,4} where 1 = day, 2 = week, 3 = month, and 4 = year. The variable pup is used to store the time unit used to express pu.

qty — Integer type; range of values 1 to 9999. The number of child parts per parent part is read, stored and written with the variable qty.

scost — Real type; maximum 8 digits; f8.2 for format code. The setup cost for manufacturing the part is stored into the variable scost.

servpartkey — Integer type; range of values -1 to 999. The record number for the first Service Part (SP) demand in the SP

data file is stored in the variable servpartkey.

SR — Integer type; range of values -1 to 9999. The SR variable is the pointer (the record number) to the first record of Scheduled Receipts for the given part. If SR = -1, there are no scheduled receipts for the part.

ucost — Real type; maximum of 8 digits; format code -f8.4. The unit cost for the part is stored into the variable ucost.

xorderkey — Integer type; range of values -1 to 999. The record number for the first external order (XO) demand in the XO data file is stored in the variable xorderkey.

4.3.2.2.3 Description of Flow*

The subroutine "newbom" interactively creates the bom (Bills-of-Materials) data file with the following steps:

Step 1. Initialize the variables pBOMkey (the next available record) to 1; key (the record for listing information) to Ø; psys (the parent part's system number) to Ø; and csys (the child part's system number to Ø).

Step 2. Ask the programmer for the company part number for the parent part, parent, and for the number of different children children, numkids, for the specified parent part. If parent = 'quit', the assumption is that no more material is to be entered into the bom data file. With no more additions, the "childkeys" subroutine is called to assign values to BofM in the partstatus

* For flowchart, see Appendix C, page 516.

data file and to the ckey in the bom data file. Then, control returns to the main program.

Step 3. Find the parent's system part number, psys, which is performed by the subroutine "findsys". If psys = -1, the parent part is not listed in partid data file.

Step 4. Store the values of numkids and BOM on the part's record in the partstatus data file.

Step 5. Obtain the child part's from the programmer; get the child's system part number with the "findsys" subroutine; assign pkey, the pointer to the parent's next child or -1 if last child; and store the information on a record in the bom data file.

4.3.2.2.4 Example - Creating a bom File

The following example shows the steps performed interactively by the MRP User and the computer to create the bom file. The partid file, used to obtain the system part numbers, is the nonrandomized file in Figure 4.17. The information on parts a, c, e, and g in Figure 2.2 are entered as follows:

Step 1. The MRP User starts running the BOMfile.

Step 2. Computer prints 'Does BOM need to be created, appended or existing information altered? Type either new, add, or change?

Step 3. The MRP User types in 'new'.

Step 4. Computer prints 'parent, # of children or quit, Ø to stop'. MRP user types 'a, 3'.

Step 5. Computer prints 'child, quantity' and the MRP User types in 'b, 2'.

Step 6. Computer prints 'child, quantity' and the MRP User types in 'c,1'.

Step 7. Computer prints 'child, quantity' and the MRP User types in 'e, 2'. See Figure 4.21.

Step 8. Computer prints 'parent, # of children or quit, Ø to stop'. MRP User types 'c, 2'.

Step 9. Computer prints 'child, quantity' and MRP User types in 'd, 1'.

Step 10. Computer prints 'child, quantity' and the MRP User types in 'b, 1'. See Figure 4.22.

Step 11. Computer prints 'parent, # of children or quit, Ø to stop'. MRP User types 'e, 2'.

Step 12. Computer prints 'child, quantity' and MRP User types 'f, 1'.

Step 13. Computer prints 'child, quantity' and the MRP User types 'c, 2'. See Figure 4.23.

Step 14. Computer prints 'parent, # of children of quit, Ø to stop'. MRP User types 'q,4'.

Step 15. Computer prints 'child, quantity' and MRP User types 'f,1'.

Step 16. Computer prints 'child, quantity' and MRP User types 'b, 4'.

Step 17. Computer prints 'child, quantity' and MRP User types 'c, 2'.

Step 18. Computer prints 'child, quantity' and MRP User types 'h, 1'. See Figure 4.24.

```
next
 ▪
 ▪
 4      Bill-of-Material
1,  2,    2,    2,  -1
1,  3,    1,    3,  -1
1,  5,    2,   -1,  -1
 ▪   ▪     \     \     \ ckey
 ▪  child   \     \ pkey
parent       \ quantity
```

Figure 4.21
Step 7 bom File

```
 6      Bill-of-Material
1,  2,    2,    2,  -1
1,  3,    1,    3,  -1
1,  5,    2,   -1,  -1
3,  4,    1,    5,  -1
3,  2,    1,   -1,  -1
```

Figure 4.22
Step 10 bom File

```
 8      Bill-of-Material
1,  2,    2,    2,  -1
1,  3,    1,    3,  -1
1,  5,    2,   -1,  -1
3,  4,    1,    5,  -1
3,  2,    1,   -1,  -1
5,  6,    1,    7,  -1
5,  3,    2,   -1,  -1
```

Figure 4.23
Step 13 bom File

```
next
  ▪
  ▪
 12      Bill-of-Material
 1,   2,    2,    2,   -1
 1,   3,    1,    3,   -1
 1,   5,    2,   -1,   -1
 3,   4,    1,    5,   -1
 3,   2,    1,   -1,   -1
 5,   6,    1,    7,   -1
 5,   3,    2,   -1,   -1
 7,   6,    1,    9,   -1
 7,   2,    4,   10,   -1
 7,   3,    2,   11,   -1
 7,   8,    1,   -1,   -1
 ▪    ▪     \     \     \ ckey
 ▪  child    \     \ pkey
parent       quantity
```

Figure 4.24
Step 18 bom File

Step 19. Computer prints 'parent, # of children or quit, Ø to stop'. MRP User types 'quit, Ø'.

4.3.2.3 Subroutine "childkeys"

4.3.2.3.1 Purpose

The subroutine "childkey" has the responsibility to link the child parts of the bom data file to all its different parents and to list BofM, the part's first appearance in the bom data file as a child part, on the part's record in the partstatus file. The subroutine "childkeys" is called by the subroutines "newbom", "addbom", and "cchange".

4.3.2.3.2 Variables

BofM	See section 4.3.2.2.2.
BOM	See section 4.3.2.2.2
c	Integer type; values range from 1 to 999. The system number of the child part on the bom record is stored into c.
ccost	See section 4.3.2.2.2
ccper	See section 4.3.2.2.2.
ch	Integer type; values range from 1 to 999. The child part, on the bom record where the child part was last found, is stored into the variable ch.
i	Integer type; values range from 1 to 9999. The variable i is the do loop index and the record number where p, c, q, ke and key are read in the bom data file.
ii	Integer type; values range from 1 to 9999. The index of one of the do loops is ii. The do loop with ii as index

searches for the other locations of c.

ke — Integer type; values range from -1 to 9999. The key, which links the parent part with its different child parts, is ke.

kee — Integer type; values range from -1 to 9999. During the ii do loop, the location of the next child for the given parent is kee.

key — Integer type; values range from -1 to 9999. The pointer to the next parent part for the child part is key.

keys — Integer type; values range from -1 to 9999. The location of the next parent for the given child is read and stored in the variable keys.

ki — Integer type; values range from 1 to 9999. The variable ki is the location where the last c to be linked is stored.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — See Section 4.3.2.2.2.

nc — Integer type; range of values -1 to 9999. The variable nc is the number of different children the parent needs for its assembly.

next	Integer type; range of values 1 to 9999. The next available record in the partstatus or bom data files is next. The number of records in the file is equal to next -1.
oh	See Section 4.3.2.2.2.
p	Integer type; values range from 1 to 999. The parent part on the bom record is stored into the variable p.
pa	Integer type; values range from 1 to 999. The parent part, on the record where the child part was last found, is stored in pa.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2
q	Integer type; values range from 1 to 9999. The number of c's per p is stored in the variable q.
qt	Integer type; values range from -1 to 9999. The number of child parts, ch, per parent part, pa, are stored in the variable qt.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.3.2.3.3. Description of Flow*

The subroutine "childkeys" has the responsibility to link the child part with its different parent parts. In other words, the objective is

---

* For flowchart, see Appendix C, page 519.

to be able to find all the parent parts for any given child part. The partstatus record will contain a pointer (BofM) to the first bom record where the first child part appears and each bom record will have a key (ckey) which points to the next record with the child on it. The last bom record with the child on it will have a key (ckey) equal to -1.

Since the subroutines "newbom", "addbom" and "cchange" all call upon the "childkeys" subroutines, the first step is to initialize or reinitialize the BofM pointer in the partstatus data file; i.e., BofM = -1.

The second step is to read each record in the bom data file -- p, c, q, key, key. If key = -1, each of the records that follow are read -- pa, ch, qt, kee, keys. If c = ch, a link is established by assigning the record number where ch was found to key. After all possible links for old c are made, the old c is replaced by a new c on the next bom record.

4.3.2.3.4 Example -- Assigning ckey and BofM

When Step 19 in Example 4.3.2.2.4 has been executed, the computer calls subroutine "childkeys". When "childkeys" is through, the bom data file will appear as it does in Figure 4.25, and the partstatus data file will appear as it does in Figure 4.26.

In summary, the "childkeys" subroutine performs the following steps:

Step 1. Reads record number 1 in the bom file;

p = 1 ke = 2

c = 2 key = -1

q = 2

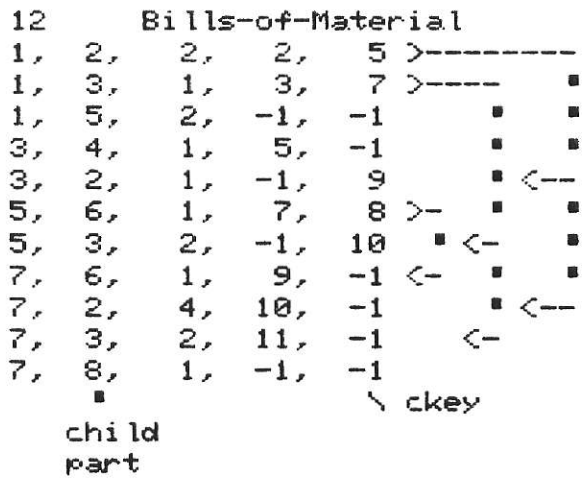


Figure 4.25
The bom File
With ckey Assigned

```
  0., -1,  3,  1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1, -1, -1,  1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1,  2,  4,  2,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1, -1, -1,  4,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
0., -1,  2,  6,  3,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1, -1, -1,  6,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1,  4,  8, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1, -1, -1, 11,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  0., -1, -1, -1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
```

numkids BOM BofM

Figure 4.26
BofM and BOM in
partstatus file

Step 2. Sets ki = 1

Step 3. Since key = -1 and BofM = -1, BofM = 1

Step 4. Reads record number 2 in bom file. Since ch = 3, c ≠ ch.

Step 5. Reads record number 3 in bom file. Since ch = 5, c ≠ ch.

Step 6. Reads record number 4 in bom file. Since ch = 4, c ≠ ch.

Step 7. Reads record number 5 in bom file. Since ch = 2 = c, ckey = key = 5 for record 1. Also, ki = 5, p = 3, c = 2, q = 1, ke = -1.

Step 8. Reads record 6 in bom file. Since ch = 6, c ≠ ch.

Step 9. Reads record 7 in bom file. Since ch = 3, c ≠ ch.

Step 10. Reads record 8 in bom file. Since ch = 6, c ≠ ch.

Step 11. Reads record 9 in bom file. Since ch = 2 = c, key = ckey = 9 for record 5. Also, ki = 9, p = 7, c = 2, q = 4, ke = 10.

Step 12. Reads record 10 in bom file. Since ch = 3, ch ≠ c.

Step 13. Reads record 11 in bom file. Since ch = 8, ch ≠ c.

Step 14. Reads record number 2 in the bom file;

p = 1 ke = 3

c = 3 key = -1

q = 1

Step 15. Since key = -1 and on record 3 of partstatus file BofM = -1, BofM = 2.

Similarily, the procedure continues until all ckey's and BofM's are assigned.

4.3.2.4 Subroutine "findsys"

4.3.2.4.1 Purpose

The subroutine "findsys has the responsibility to find the system part number for a given company part number. A binary search is used to locate the company part number record, which also contains the system part number. Both of the parts lists, the company part number and the system part numbers, are stored in the partid data file. For more details on the partid data file, see Section 3.3.3.

4.3.2.4.2 Variables

down — Integer type; values range from -1 to 999. The downward key for the binary search is stored into the variable down.

first — Integer type; values range from 1 to 999. The variable first is the record number of the company part number with ordinal number one.

k — Integer type; values range from 1 to 999. The variable k is the record number for the read in the partid data file.

key — Integer type; values range from 1 to 999. The variable key is the directional key which points to the part with ordinal number n + 1 given u has ordinal number n.

mid — Integer type; values range from 1 to 999. The variable mid is the midpoint record number of the company part number list.

next — Integer type; values range from 1 to 999. The variable

	next is the next available record in the partid data file. The quantity next -1 is the number of records in the file.
part	Character type; subroutine argument; maximum of 10 alphanumeric digits. The company part number that is being sought is stored in the variable part.
pos	Integer type; values range from 1 to 999. The ordinal number of the company part number is stored into the variable pos.
s	Integer type; values range from 1 to 999. The system part number from record number k is stored into the variable s.
sys	Integer type; subroutine argument; values range from -1 to 999. The system part number for part is stored into the variable sys. If the company part number can't be found in the partid data file, sys = -1.
u	Character type; maximum of 10 alphanumeric characters. The company part number from record number k is stored into the variable u.
up	Integer type; values range from -1 to 999. The upward key for the binary search is stored into the variable up.

4.3.2.4.3 Description of Flow*

The subroutine "findsys" is assigned the task of finding the system part number for a given company part number. The partid data file,

* For flowchart, see Appendix C, page 521.

which contains both sets of part numbers, has been equipped with a set of up and down binary search keys. To find the sys (system number) for part (the company part number), the binary search algorithm in the "findsys" subroutine involves the following steps:

Step 1. Read record Ø in the partid data file (first, next, mid and add); and store the midpoint of the company part number list into the variable mid

Step 2. Evaluate k = mid, where k is the record number in the partid file to be read and mid is the midpoint of the file.

Step 3. Read record k and store the data into the variables u, s, ..., up, down (for definitions see variable list).

Step 4. Company part with u.

Step 4a. If part = u, the desired company part number has been found. The system part number for part is s.

Step 4b. If part < u, the company part number part is located in the upper branches of the binary tree. Therefore, up is the next record number to be read in the search for part. However, if up = -1, there are no more upper branches and part is not in the partid file. If up ≠ -1, set k = up and go to Step 3.

Step 4c. If part > u, the company part number part can be found in the lower branches of the binary tree. Therefore, down is the next record number to be read in the search for part. However, if down = -1, there are no more lower branches and part is not in the partid file. If down ≠ -1, set k = down and go to Step 3.

4.3.2.4.4 Example - Finding the System Part Number

Given part = 'j' and the randomized partid data file in Figure 4.8, the "findsys" subroutine finds the system part number for 'j' with the following steps:

Step 1. Read record Ø in the partid file; first = 7; next = 11, and mid = 1.

Step 2. K = mid = 1

Step 3. Read record K = 1 in partid file;

u = e,

s = 1,

key = 6,

pos = 5,

up = 8,

down = 5.

Step 4. Since part = j > u = 3, K = down = 5.

Step 5. Read record K = 5 in partid file;

u = h,

s = 5,

key = 2,

pos = 8,

up = 3,

down = 2.

Step 6. Since part = j > u = h, K = down = 2.

Step 7. Read record K = 2 in partid file;

u = i,

s = 2,

key = 9,

pos = 9,

up = -1,

down = 9.

Step 8. Since part = j > u = i, K = down = 9.

Step 9. Read record K = 9 in partid file;

u = j,

s = 9,

key = 999,

pos = 10,

up = -1,

down = -1,

Since part = u = j, the system part number for j is s = 9.

4.3.2.5 Subroutine "addbom"

4.3.2.5.1 Purpose

When a bom (Bills-of-Material) data file has ready been established, the subroutine "addbom" is available to the programmer for including new part's assemblies and to append the stored information. The subroutine works interactively with the MRP user. The MRP user is asked to provide the information for the new bom record; i.e., the parent part, the child part, and the number of child parts per parent part. Then, the computer under the direction of the supplied routine makes the appropriate adjustments to the pointers (BOM and BofM) and the keys (pkey and ckey).

4.3.2.5.2 Variables

BofM — See Section 4.3.2.2.2

BOM — See Section 4.3.2.2.2.

c — Character type; maximum of 10 alphanumeric digits. The child company part number, which is to be added to the bom data file, is stored into the variable c.

cc — Integer type; range of values 1 to 999. The child part is read, stored, and written with the variable cc.

cckey — Integer type; range of values 1 to 9999. The variable cckey is the key that links the child part to its other parent parts.

ccost — See Section 4.3.2.2.2.

ccper — See Section 4.3.2.2.2.

csys — Integer type; values range from -1 to 999. The system number for c is stored into the variable csys. If csys = -1, the company part number for c is not in the partid file.

i — Integer type; range of values 1 to 999. The variable i is the index variable for the do loop which resets the ckey to -1.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — See Section 4.3.2.2.2.

next — Integer type; values range from 1 to 9999. The next free

	record number in the bom data file is stored into the variable next.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
p	Character type; maximum of 10 alphanumeric digits. The parent company part number, which is to be added to the bom data file, is stored into the variable p.
pp	Integer type; range of values 1 to 999. The parent part is read, stored, and written with the variable pp.
ppkey	Integer type; range of values 1 to 9999. The variable ppkey is the key that links the parent part to its other child parts.
psys	Integer type; values range from 1 to 999. The system number for p is stored into the variable psys.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
q	Integer type; values range from 1 to 9999. The number of c's per p is stored into the variable q.
qq	Integer type; range of values 1 to 9999. The number of child parts (cc) per parent part (pp) is stored into the variable qq.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.

where — Integer type; values range from 1 to 9999. The number of records in the bom data file is stored into the variable where. The variable where is, also, the record number for the new addition.

xorderkey — See Section 4.3.2.2.2.

4.3.2.5.3 Description of Flow*

The additions to the bom data file are made with the use of the subroutine "addbom" in the BOMfile input file. The parent part, the child part, and the quantity of child parts per parent are supplied interactively by the programmer. The difficulty in making the addition is adjusting the keys and pointers properly.

The subroutine "addbom" consists of the following steps:

Step 1. Find out the next available record number in the bom data file. This is done by reading next off the record Ø in the bom file.

Step 2. Reset the ckey, the link for different parents for the child part, to -1. This is done so that the subroutine "childkeys" will be able to correctly make the linkages.

Step 3. Calculate the number of records in the bom data file; i.e., where = next -1.

Step 4. Obtain the new addition from the MRP user. The question 'parent, child, qty to be added or quit, q, Ø to stop' is asked and the answer is stored into the variables p, c and q.

---

*For flowchart, see Appendix C, page 523.

Step 5. If p ≠ 'quit; go onto Step 6. If p = 'quit'; list the next available record on the record Ø of the bom data file; call the subroutine "childkeys" to establish the ckey's and BofM's; and stop.

Step 6. Find the system numbers psys and csys for the company part numbers p and c, respectively.

Step 7. Increment the record number where by +1.

Step 8. Read the record in the partstatus data file for the parent part.

Step 9. Adjust the number of child parts, numkids, the parent part has. The parent part has no children if numkids = -1.

Step 10. Check the value BOM.

Step 10a. If BOM = -1, there are no previous child parts. Set BOM = where, the record number of its first appearance as a parent part in the bom data file. Set pkey = -1 and ckey = -1. Record the new addition into the bom file. Go to Step 4.

Step 10b. If BOM ≠ -1, there are previous child parts. Locate the last child for the parent part in the bom data file and adjust pkey by setting it equal to where. Record the new addition into the bom data file with pkey = ckey = -1.

4.3.2.5.4 Example - Adding Records to bom File

The following example shows the steps performed interactively by the MRP User and the computer to add records to the bom file. Prior to adding the new records, the bom file appears as it does in Figures 4.25 and the partstatus file appears as it does in Figure 4.26. The BOM information for parts h, i, and j in Figure 2.2 are entered as follows:

Step 1. The MRP User starts running the BOMfile.

Step 2. Computer prints 'Does BOM need to be created, appended or existing information altered? type either new, add or change?'

Step 3. The MRP User types in 'add'.

Step 4. All ckey's in bom are reset to -1.

Step 5. Computer prints 'parent, child, qty to be added or quit, q, Ø to stop' and the MRP User types 'h, i, 1'. See Figure 4.27.

Step 6. Computer prints 'parent, child, qty...' and the MRP User types 'i, f, 3'. See Figure 4.28.

Step 7. Computer prints 'parent, child, qty...' and the MRP User types 'i, d, 4'. See Figure 4.29.

Step 8. Computer prints 'parent, child, qty...' and the MRP User types 'j, b, 1'. See Figure 4.30.

Step 9. Computer prints 'parent, child, qty...' and the MRP User types 'j, i, 2'. See Figure 4.31.

Step 10. Computer prints "parent, child, qty to be added or quit, q, Ø to stop' and the MRP User types in 'quit, q, Ø'. The subroutine 'childkeys' is called for the bom file, see Figure 4.32 and for the partstatus file see Figure 4.33.

4.3.2.6. Subroutine "changebom"

4.3.2.6.1 Purpose

The subroutine "changebom" enables the MRP User to correct errors that have invaded the bom data file. If either the parent part, the child part and/or quantity of child parts per parent parts is incorrect, they can be corrected under the direction of the "changebom" subroutine. The

```
13      Bills-of-Material
1,   2,    2,    2,   -1
1,   3,    1,    3,   -1
1,   5,    2,   -1,   -1
3,   4,    1,    5,   -1
3,   2,    1,   -1,   -1
5,   6,    1,    7,   -1
5,   3,    2,   -1,   -1
7,   6,    1,    9,   -1
7,   2,    4,   10,   -1
7,   3,    2,   11,   -1
7,   8,    1,   -1,   -1
8,   9,    1,   -1,   -1   <===   addition
```

Figure 4.27
Step 5
Additions to bom File

```
14      Bill-of-Material
1,   2,    2,    2,   -1
1,   3,    1,    3,   -1
1,   5,    2,   -1,   -1
3,   4,    1,    5,   -1
3,   2,    1,   -1,   -1
5,   6,    1,    7,   -1
5,   3,    2,   -1,   -1
7,   6,    1,    9,   -1
7,   2,    4,   10,   -1
7,   3,    2,   11,   -1
7,   8,    1,   -1,   -1
8,   9,    1,   -1,   -1
9,   6,    3,   -1,   -1 <==   addition
```

Figure 4.28
Step 6
Additions to bom File

```
15      Bills-of-Material
1,   2,    2,    2,   -1
1,   3,    1,    3,   -1
1,   5,    2,   -1,   -1
3,   4,    1,    5,   -1
3,   2,    1,   -1,   -1
5,   6,    1,    7,   -1
5,   3,    2,   -1,   -1
7,   6,    1,    9,   -1
7,   2,    4,   10,   -1
7,   3,    2,   11,   -1
7,   8,    1,   -1,   -1
8,   9,    1,   -1,   -1
9,   6,    3,   14,   -1
9,   4,    4,   -1,   -1   <===    addition
```

Figure 4.29
Step 7
Additions to bom File

```
16      Bills-of-Material
1,   2,    2,    2,   -1
1,   3,    1,    3,   -1
1,   5,    2,   -1,   -1
3,   4,    1,    5,   -1
3,   2,    1,   -1,   -1
5,   6,    1,    7,   -1
5,   3,    2,   -1,   -1
7,   6,    1,    9,   -1
7,   2,    4,   10,   -1
7,   3,    2,   11,   -1
7,   8,    1,   -1,   -1
8,   9,    1,   -1,   -1
9,   6,    3,   14,   -1
9,   4,    4,   -1,   -1
10,  2,    1,   -1,   -1   <===   addition
```

Figure 4.30
Step 8
Additions to bom File

```
 17      Bills-of-Material
 1,   2,    2,    2,   -1
 1,   3,    1,    3,   -1
 1,   5,    2,   -1,   -1
 3,   4,    1,    5,   -1
 3,   2,    1,   -1,   -1
 5,   6,    1,    7,   -1
 5,   3,    2,   -1,   -1
 7,   6,    1,    9,   -1
 7,   2,    4,   10,   -1
 7,   3,    2,   11,   -1
 7,   8,    1,   -1,   -1
 8,   9,    1,   -1,   -1
 9,   6,    3,   14,   -1
 9,   4,    4,   -1,   -1
10,   2,    1,   16,   -1
10,   9,    2,   -1,   -1      <===   addition
```

Figure 4.31
Step 9
Additions to bom File

```
 17      Bills-of-Material
 1,   2,    2,    2,    5
 1,   3,    1,    3,    7
 1,   5,    2,   -1,   -1
 3,   4,    1,    5,   14
 3,   2,    1,   -1,    9
 5,   6,    1,    7,    8
 5,   3,    2,   -1,   10
 7,   6,    1,    9,   13
 7,   2,    4,   10,   15
 7,   3,    2,   11,   -1
 7,   8,    1,   -1,   -1
 8,   9,    1,   -1,   16
 9,   6,    3,   14,   -1
 9,   4,    4,   -1,   -1
10,   2,    1,   16,   -1
10,   9,    2,   -1,   -1
                      \ ckey assigned
```

Figure 4.32
Completed Sample
Problem bom File

```
0., -1,  3,  1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1,  1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1,  2,  4,  2,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1,  4,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1,  2,  6,  3,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1, -1, -1,  6,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1,  4,  8, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1,  1, 12, 11,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1,  2, 13, 12,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
0., -1,  2, 15, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,  100.,4, -1, -1, -1, -1, -1
```



Figure 4.33
All the numkids, BofM, and BOM
Assigned for Sample Problem

subprogram requests from the programmer a list of a parent part, a child part, and a quantity of the childparts per parent part that are on the old bom record. Then, a request is made for a list of a parent part, a child part, and a quantity of child parts per parent for the new bom record. To actually make the change in the old bom record, the subroutines "findsys", "pchange", "cchange" and "qchange" are called upon by the subroutine "changebom" as they are needed.

Note: "changebom" is designed with the capability to replace all or any part of the bom record.

4.3.2.6.2 Variables

find	Integer type; set of possible values include {0, 1} where 1 = pair found and Ø = pair not found. The variable find is used as a flag for when the old parent and child part combination can be found or not in the bom data file.
newc	Character type; maximum of 10 alphanumeric digits. The child's company part number, for the corrected bom record, is stored into the variable newc.
newcsys	Integer type; values range from -1 to 999. The system part number for newc is stored into newcsys.
newp	Character type; maximum of 10 alphanumeric digits. The parents' company part number, for the corrected bom record, is stored into the variable newp.
newpsys	Integer type; values range from -1 to 999. The system part number for newp is stored into newpsys.

newq — Integer type; values range from 1 to 9999. The number of newc's per newp is stored into the variable newq.

oldc — Character type; maximum of 10 alphanumeric digits. The child's company part number, of the bom record to be changed, is stored into oldc.

oldcsys — Integer type; values range from -1 to 999. The system part number for oldc is stored into the variable oldcsys.

oldp — Character type; maximum of 10 alphanumeric digits. The parent's company part number, of the bom record to be changed, is stored into the variable oldp.

oldpsys — Integer type; values range from -1 to 999. The system part number for oldp is stored into the variable oldpsys.

oldq — Integer type; values range from 1 to 9999. The member of oldc's per oldp is stored into the variable oldq.

4.3.2.6.3 Description of Flow*

In order to make the required corrections to the bom data file, the subroutine "changebom" has the following steps:

Step 1. Obtain the old parent part, the old child part and the old quantity of child parts per parent part from the MRP User. If the parent part has been named 'quit', the assumption is that there are no more corrections; and stops. If the parent part has not been named 'quit', continue to Step 2.

Step 2. Call the subroutine "findsys" to obtain the system part numbers for the old parent part and the old child part.

---

*For flowchart, see Appendix C, page 526.

Step 3. Obtain the new parent part, the new child part and the new quantity of child parts per parent part. If the parent part has been named 'quit', the assumption is that no correction will be made after all; and return to Step 1. If the parent part has not been named 'quit', continue to Step 4.

Step 4. Call the subroutine "findsys" to obtain the system part numbers for the new parent part and the new child part.

Step 5. Set flag for finding the old parent and child part to one; i.e., find = 1. That is, assume they have been found.

Step 6. If oldp ≠ newp, call the subroutine "pchange".

Step 7. Check the value of find. If find = Ø, the old parent and child bom record wasn't found, go to Step 1.

Step 8. If oldc ≠ newc, call subroutine "cchange".

Step 9. (Same as Step 7.)

Step 10. If oldq ≠ newq, call subroutine "qchange" and return to Step 1.

Step 11. Stop.

4.3.2.6.4 Example - Changing bom Record

The "changebom" subroutine is designed to change on the bom records either the parent part, the child part, the quantity of child parts per parent part, or any combination of the three items. This example shows step-by-step the change of bom record 'g, b, 4' to 'e, d, 2'. The steps are as follows:

Step 1. The MRP User starts running BOMfile. See Figure 4.32 for bom file before the change takes place.

Step 2. The computer prints 'Enter parent, child, qty as it now appears' and the MRP User types in 'g, b, 4'.

Step 3. The computer uses the 'findsys' subroutine to find the system part numbers; oldpsys = 7 and oldcsys = 2.

Step 4. The computer prints 'Enter parent, child, qty as it should appear' and the MRP User types in 'e, d, 2'.

Step 5. The computer uses the "findsys" subroutine to find the system part numbers; newpsys = 5 and newcsys = 4.

Step 6. The computer calls the subroutine 'pchange'. See Figure 4.34.

Step 7. The computer calls the subroutine 'cchange'. See Figure 4.35.

Step 8. The computer calls the subroutine 'qchange'. See Figure 4.36.

4.3.2.7 Subroutine "pchange"

4.3.2.7.1 Purpose

When called by the subroutine "changebom", the subroutine "pchange" searches out the bom record with th parent part record that needs to be corrected; replaces the oldpsys with the newpsys; and adjust the pointer (BOM) and the key (pkey) accordingly.

```
 17      Bills-of-Material                150
 1,   2,    2,    2,    5
 1,   3,    1,    3,    7
 1,   5,    2,   -1,   -1
 3,   4,    1,    5,   14
 3,   2,    1,   -1,    9
 5,   6,    1,    7,    8
 5,   3,    2,    9,   10
 7,   6,    1,   10,   13
 5,   2,    4,   -1,   15  <==  PARENT PART CHANGED
 7,   3,    2,   11,   -1          was g(7) now e(5)
 7,   8,    1,   -1,   -1
 8,   9,    1,   -1,   16
 9,   6,    3,   14,   -1
 9,   4,    4,   -1,   -1
10,   2,    1,   16,   -1
10,   9,    2,   -1,   -1
```

Figure 4.34
Parent Changed on bom Record

```
 17      Bills-of-Material
 1,   2,    2,    2,    5
 1,   3,    1,    3,    7
 1,   5,    2,   -1,   -1
 3,   4,    1,    5,    9
 3,   2,    1,   -1,   15
 5,   6,    1,    7,    8
 5,   3,    2,    9,   10
 7,   6,    1,   10,   13
 5,   4,    4,   -1,   14  <=== CHILD PART CHANGED
 7,   3,    2,   11,   -1       -was b(2) now d(4)
 7,   8,    1,   -1,   -1       -ckey changed
 8,   9,    1,   -1,   16
 9,   6,    3,   14,   -1
 9,   4,    4,   -1,   -1
10,   2,    1,   16,   -1
10,   9,    2,   -1,   -1
```

Figure 4.35
Child Changed on bom Record

```
 1,  2,   2,   2,   5
 1,  3,   1,   3,   7
 1,  5,   2,  -1,  -1
 3,  4,   1,   5,   9
 3,  2,   1,  -1,  15
 5,  6,   1,   7,   8
 5,  3,   2,   9,  10
 7,  6,   1,  10,  13
 5,  4,   2,  -1,  14  <=== QUANTITY CHANGED
 7,  3,   2,  11,  -1        was 4 now 2
 7,  8,   1,  -1,  -1
 8,  9,   1,  -1,  16
 9,  6,   3,  14,  -1
 9,  4,   4,  -1,  -1
10,  2,   1,  16,  -1
10,  9,   2,  -1,  -1
```

Figure 4.36
Quantity Changed on bom Record

4.3.2.7.2 Variables

BofM — See Section 4.3.2.2.2.

BOM — See Section 4.3.2.2.2.

c — Integer type; values range from 1 to 999. The child part on the bom record is read, stored and written with the variable c.

cl — Integer type; range of values 1 to 999. The child part of parent p on the last record number, keyl, is stored into the variable cl.

ccost — See Section 4.3.2.2.2.

ccper — See Section 4.3.2.2.2.

ckey — Integer type; values range from -1 to 9999. The record number for the next parent part of child c is read, stored and written with the variable ckey.

ckeyl — Integer type; range of values -1 to 9999. The ckey on record number keyl is stored into the variable ckeyl.

find — Integer type; subroutine argument; set of possible values include {0, 1} where 1 = pair found and Ø = pair not found. The variable find is used as a flag for when the old parent and child part combination can be found or not in the bom data file.

key — Integer type; values range from 1 to 9999. The variable key is the record number in the bom data file to be read in search of the old parent and child part pair.

keyl — Integer type; range of values 1 to 9999. The record number

for the last child of parent p is stored into the variable keyl.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MRSkey — See Section 4.3.2.2.2.

n — Integer type; values range from 0 to 9999. The variable n keeps track of the number of reads needed to find to old parent and child pair.

newpsys — Integer type; subroutine argument; values range from +1 to 999. The system part number for the parent part, to replace oldpsys, is stored into the variable newpsys.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

oldcsys — Integer type; subroutine argument; values range from 1 to 999. The system part number for the child part, on the bom record with the parent part error, is stored into the variable oldcsys.

oldpsys — Integer type; subroutine argument; values range from 1 to 999. The system part number, for the parent part to be changed, is stored into the variable oldpsys.

p — Integer type; values range from 1 t 999. The parent part on the bom record is read, stored and written with the variable p.

pkey	Integer type; values range from -1 to 9999. The record number for the next child of parent p is read, stored and written with the variable pkey.
pkey1	Integer type; range of values -1 to 9999. The pkey on record number key1 is stored into variable pkey1.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
q	Integer type; values range from 1 to 9999. The number of c's per p on the bom record is read, stored and written with the variable q.
q1	Integer type; range of values 1 to 9999. The number of c1's per parent p on the record key1 is stored into the variable q1.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.3.2.7.3 Description of Flow*

The subroutine "pchange" consists of the following steps:

Step 1. Locate the old parent and child record in the bom data file. The variable BOM on the parent part record in the partstatus data file and the pkey on the bom records direct the search through the records to find the old parent and child record.

---

*For flowchart see Appendix C, page 529.

Step 2. Adjust numkids on the old parent parts record in the partstatus data file; i.e., numkids = numkids -1.

Step 3. Correct the old parent keys.

Step 4. Correct the new parent keys.

4.3.2.8 Subroutine "cchange"

4.3.2.8.1 Purpose

When called by the subroutine "changebom", the subroutine "cchange" will search out the old parent and child part record in the bom data file; replace the old child part by the new child part; correct the pointer (BofM); and call subroutine "childkeys" to the child part keys (ckey).

4.3.2.8.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
c	See Section 4.3.2.7.2
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
ckey	See Section 4.3.2.7.2.
find	See Section 4.3.2.7.2.
i	Integer type; values range from 1 to 9999. The do loop index variable is i. This do loop reassigns ckey to -1.
key	See Section 4.3.2.7.2.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.

lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
next	Integer type; values range from 1 to 9999. The number of files in bom data file is next -1.
newcsys	Integer type; subroutine argument; values range from 1 to 999. The newcsys variable is the system number of the child part that will replace oldcsys.
newpsys	Integer type; subroutine argument; values range from 1 to 999. The system part number for the parent part is stored into the variable newpsys.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
oldcsys	Integer type; subroutine argument; values range from 1 to 999. The oldcsys variable is the system number for the child part to be replaced.
P	See Section 4.3.2.7.2.
pkey	See Section 4.3.2.7.2.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
q	See Section 4.3.2.7.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.

xorderkey See Section 4.3.2.2.2.

4.3.2.8.3 Decription of Flow*

To change the child part on the bom record, the subroutine "cchange" follows these steps.

Step 1. Locate the old parent and child part record using the pointer BOM and the key pkey.

Step 2. Replaces the old child part with the new child part.

Step 3. Reset the ckey values to -1.

Step 4. Call subroutine "childkeys" to correct the ckey's and BofM's.

Step 5. Stop.

4.3.2.9 Subroutine "qchange"

4.3.2.9.1 Purpose

When called by the subroutine "changebom", the "qchange" subroutine proceeds to locate the old parent and child part record in the bom data file; and, then, corrects the old quantity of child parts for each parent part with the new quantity of child parts for each parent part.

4.3.2.9.2 Variables

c See Section 4.3.2.7.2.

ckey See Section 4.3.2.7.2.

find See Section 4.3.2.7.2.

key See Section 4.3.2.7.2.

---

*For flowchart, see Appendix C, page 532.

newcsys	Integer type; subroutine argument; values range from 1 to 999. The system part number for the child part is stored into variable newscsys.
newpsys	Integer type; subroutine argument; values range frm 1 to 999. The system number for the parent part is stored into variable newpsys.
newq	Integer type; subroutine argument; values range from 1 to 9999. The value of newc is equal to the number of newcsys per newpsys.
p	See Section 4.3.2.7.2.
pkey	See Section 4.3.2.7.2.
q	See Section 4.3.2.7.2.

4.3.2.9.3 Description of Flow*

When the subroutine "changebom" calls upon the subroutine "qchange" to change the old number of child parts per parent part, the procedure is as follows:

Step 1. Locate the parent, child part record in the bom data file with the pointer BOM and the keys pkey.

Step 2. Replace the old quantity, oldq, with the new quantity, newq.

* For flowchart, see Appendix C, page 534.

4.4 DataPlus

4.4.1 Introduction to DataPlus

Once the partid file is established, the bom file is created, and the partstatus file is initialized, there still remains a variety of information that needs to be supplied by the MRP User. The DataPlus interactively works with the MRP User to handle the information for (1) on hand inventory quantities and scheduled receipts, (2) lead times and lotting rules, and (3) the dates for the planning horizon.

The DataPlus begins by asking the MRP User which of the three areas he wishes to edit. If (1) on hand inventory and scheduled receipts are chosen, then there is the option to either reenter all on hand inventory quantities and information on the scheduled receipts, or just change scheduled receipt information, or just change on hand inventory quantities. If changing scheduled receipt information, then the MRP User may either change existing scheduled receipt records, delete scheduled receipt records or add new scheduled receipt records.

When the (2) lead times and lotting rules are to be edited, then there is the option for the MRP User to enter lead times or lotting rules or both. When (3) planning horizon dates are to be submitted, the MRP User has the choice of letting the completer assign equal bucket sizes or interactively creating unequal bucket sizes for the planning horizon. If the computer is allowed to create equal buckets, each planning horizon period will be assigned a 5 day work week.

4.4.2 Subroutines In DataPlus

4.4.2.1 Overview of Subroutines

The DataPlus input file consists of the following subprograms:

1. subroutine "findsys";
2. subroutine "IandSR";
3. subroutine "ISRnew";
4. subroutine "srchange";
5. subroutine "dayorqty";
6. subroutine "delSR";
7. subroutine "addSR";
8. subroutine "ohchange";
9. subroutine "leadandlot";
10. subroutine "leadtimeandlotrule";
11. subroutine "leadtime";
12. subroutine "lotrule";
13. subroutine "horizon";
14. subroutine "equalbucket";
15. subroutine "unequalbucket";

See Figure 4.37

The main program inquires as to whether inventory and/or scheduled receipts, or lead times and lotting rules, or periods for the planning horizon, then depending upon the response from the MRP User the appropriate subroutines are called upon.

The subroutine "findsys" is like the subroutine "findsys" in the BOMfile. The subroutine does a binary search to retrieve the system part number for the company part number.

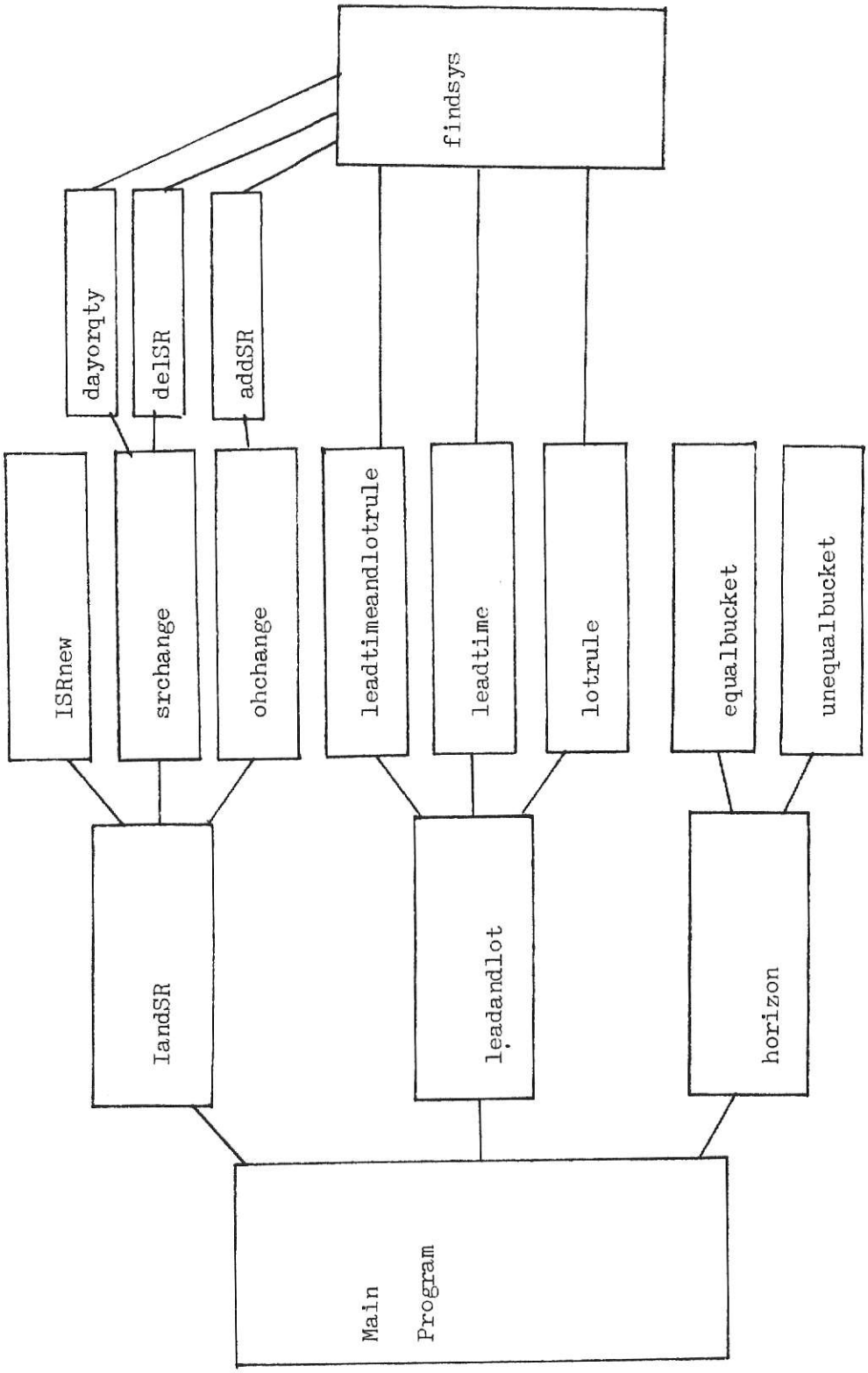


Figure 4.37
DataPlus Subroutines

The subroutine "IandSR" asks the MRP User to specify the type of editing to be done to the on hand inventory and the scheduled receipt (SR) information. There is a choice of re-entering all new information for both on hand inventory quantities and SR information, or just entering the on hand inventory quantities, or just entering the SR information.

The subroutine "ISRnew" reinitializes all the part records in the partstatus file; i.e., oh = Ø and SR = -1. Then the message 'part, inventory quantities, # of SR or quit, Ø, Ø to stop' is sent to the CRT screen. The on hand inventory quantity is stored on the specified part's partstatus record. Next, the expected date and expected number of units for each SR are requested and stored into the SR data file.

The subroutine "srchange" requests the type of changes to be made to the SR file and then calls the appropriate subprograms. The three subprograms called by "srchange" are either the "dayorqty" subroutine, the "delSR" subroutine, or the "addSR" subroutine. The "dayorqty" directs changes to the existing SR data file records. The "delSR" directs the elimination of the SR data file records. The "addSR" directs the addition of new SR data file records.

The subroutine "ohchange" is responsible for helping the MRP User to correct the on hand inventory quantities that have been stored onto the part's partstatus record.

When called by the main program, the subroutine "leadandlot" asks the MRP User to specify whether he wishes to enter either lead times or lotting rules or both. The subroutine "leadtimeandlotrule" directs the obtaining and storing of both the lead times and lotting rules need for the MRP explosion. The subroutine "leadtime" obtains and stores just the

lead times; and the subroutine "lotrule" obtains and stores information on just the lotting rules.

The subroutine "horizon" asks the MRP User to specify the number of buckets for the planning horizon, to state the number for the current period, and to choose either equal or unequal bucket sizes. If equal buckets are chosen, the subroutine "equalbucket" assigns a 5 day work week to each bucket in the planning horizon. If unequal buckets are chosen, then the subroutine "unequalbucket" asks the MRP User to specify the date for each bucket in the planning horizon.

4.4.2.2 Subroutine "findsys" See Section 4.3.2.4.

4.4.2.3 Subroutine "IandSR"

4.4.2.3.1 Purpose

After the MRP User has specified he wishes to enter on hand inventory quantities (oh) or scheduled receipt (SR) information, then the subroutine "IandSR" asks the MRP User to specify further whether he wishes to create a new on hand inventory list and a new SR data file, or to change SR data file records, or to change oh on hand inventory quantities.

4.4.2.3.2 Variable

answer — Character type; answer ε {new, SR, onhand, quit}. The variable answer is used to store the MRP User's editing preferences.

4.4.2.3.3 Description of Flow*

The subroutine "IandSR" presents the following statement to the MRP User" to create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory, type onhand. To stop, type quit'. The response is stored into the variable answer. If answer = 'new', then the subroutine 'ISRnew" is called. If answer = 'SR; the subroutine "srchange" is called. If answer = "onhand; the subroutine "ohchange" is called. If answer = 'quit', the program control returns to the main program.

4.4.2.4 Subroutine "ISRnew"

4.4.2.4.1 Purpose

When the MRP User indicates that both the information of on hand inventory quantities and the scheduled receipts is to be entered, the subroutine "ISRnew" is called. The on hand inventory quantity is stored into the partstatus data file. (See Section 3.3.4 for more information on the partstatus data file.) If the part has scheduled receipts, the 'ISRnew' lists a pointer on the partstatus record to the first SR record in the SR data file. The expected date of arrival and expected quantity of units for the scheduled receipt, as well as a key to the next SR record for the part, are stored in the SR data file. (For more information on the SR data file, see Section 3.3.5.)

---

*For flowchart, see Appendix C, page 536.

4.4.2.4.2 Variables

BofM — See Section 4.3.2.2.2.

BOM — See Section 4.3.2.2.2.

ccost — See Section 4.3.2.2.2.

ccper — See Section 4.3.2.2.2.

day — Integer type; values range from 1 to 9999. The expected due date of the scheduled receipt is stored into the variable day.

fillin — Integer type; values range from 1 to 9999. The variable fillin is stored on record Ø of the SR data file. The value of fillin is the record number of an empty record in the SR data file where a previous SR has been deleted.

i — Integer type; values range from 1 to sr. The do loop, which stores the SR in the SR data file, has the index variable of i.

key — Integer type; values range from -1 to 9999. The key is the record number of the next SR for the part. If key = -1, there are no more SR for the part.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — See Section 4.3.2.2.2.

next — Integer type; values range from 1 to 999. The quantity next -1 is the number of records in the partstatus data

	file. Also, next is the next available record in the SR data file.
numkids	See Section 4.3.2.2.2.
oh	Real type; format code f8.0. The variable oh is the quantity of on hand inventory stored on the partstatus record.
part	Character type; maximum of 10 alphanumeric digits. The variable part is the company part number being ordered or inventory quantity being specified.
pu	See Section 4.3.2.2.2.
qty	Real type; format type f8.0. The expected number of units for the SR is stored into the variable qty.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
sr	Integer type; values range from 1 to number of scheduled receipts per part. The sr is the variable where the number of SR for a particular part is stored.
SR	Integer type; values range from -1 to 9999. The pointer, to the part's scheduled receipts records in the SR data file, is the variable SR stored on the part's partstatus record.
SRkey	Integer type; values range from 1 to 9999. The variable SRkey is the record number where the first SR for the part is stored in the SR data file.
sys	Integer type; range of values from -1 to 999. The system

part number for part is stored into the variable sys. The part's partstatus record number is equal to sys. If sys = -1, part is not in the partid data file.

ucost — See Section 4.3.2.2.2.

xorderkey — See Section 4.3.2.2.2.

4.4.2.4.3 Description of Flow*

To create the SR file and to initialize on hand inventory, the following steps are performed by the subroutine 'ISRnew'.

Step 1. The oh and SR in the partstatus data file are reinitialized to 0. and -1, respectively.

Step 2. The fillin pointer is initialized at Ø.

Step 3. The record numbers, SRkey, for the SR data file are initialized at Ø.

Step 4. The user is asked to specify the 'part, inventory quantity, # of SR or quit, Ø, Ø to stop' and the response is stored into the variables part, inv, sr, respectively.

Step 5. If part = 'quit', return to main program.

Step 6. The subroutine "findsys" is called to find the system part number (sys) for the company part number (part).

Step 7. If sys = -1, go to Step 4.

Step 8. The inv is stored onto the partstatus record number sys.

Step 9. If sr > 0, the scheduled receipt information, date and quantity, is obtained interactively from the MRP User and stored in the SR data file.

---

*For flowchart, See Appendix C, page 537.

Step 10. Go to Step 4.

4.4.2.4.4 Example for Subroutine "ISRnew"

To illustrate the use of the "ISRnew" subroutine, the on hand invetory and scheduled receipts of the sample problem are entered. The steps are as follows:

Step 1. The statement - 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rule, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit.' - appears on the CRT screen. The MRP User types 'invandsr'.

Step 2. To create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory quantities, type on hand. To stop, type quit'. appears on the CRT screen and the MRP User types 'new'.

Step 3. The 'part, inventory quantity, # of SR or quit, Ø, Ø to stop' receives the response of a, 10, 1.

Step 4. To 'day expected, quantity expected', the MRP User types in '100, 12'.

Step 5. To 'part, inventory quantity, # of SR or quit, Ø, Ø to stop', the MRP User types in 'b, Ø, 2'.

Step 6. To 'day expected, quantity expected', the MRP User types in '95, 825'.

Step 7. To 'day expected, quantity expected', the MRP User tyes in '105, 270'

.
.
.

Step n. To 'part, inventory quantity, # of SR or quit, 9 Ø to stop', the MRP User types in 'i, 75, Ø'.

Step n+1. To 'part, inventory quantity, # of SR or quit, Ø, Ø to stop', the MRP User types in 'j, 3, Ø'.

Step n+2. To 'part, inventory quantity, # of SR or quit, Ø, Ø to stop', the MRP User types in 'quit, Ø, Ø'.

The results are the partstatus data file as it appears in Figure 4.38 and the SR data file as it appears in Figure 4.39.

4.4.2.5 Subroutine "srchange"

4.4.2.5.1 Purpose

Once the MRP User has specified that records in the SR data file are to be changed, then the MRP User is asked to further specify how the records are to be changed. The MRP User has the option to change existing SR records, to delete SR records or to add SR records.

4.4.2.5.2 Variables

answer — Character type, answer ε {change, delete, add quit}. The variable answer stores the response to the type of editing for the SR data file.

4.4.2.5.3 Description of Flow*

The subroutine 'srchange' asks the MRP User to respond to the following statement "To change record in SR file, type change. To delete record in the SR file, type delete. To include new records, type add. To

---

* For flowchart, see Appendix C, page 540.

```
11      Partstatus Data File
  10.,  1,  3,  1, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   0.,  2, -1, -1,  1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   0.,  4,  2,  4,  2,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
 320., -1, -1, -1,  4,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  66.,  6,  3,  6,  3,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
 100.,  7, -1, -1,  6,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  10., -1,  3,  8, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  10.,  8,  1, 12, 11,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  75., -1,  2, 13, 12,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   3., -1,  2, 15, -1,  5,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   |       \
  oh        SR
```

Figure 4.38
The partstatus With Sample Problem
Inventory and SR Keys Included

```
next
   ▪  fillin
   ▪   ▪
   9   0      SR
100,      12.,  -1
 95,     825.,   3
105,     270.,  -1
 95,     105.,   5
105,     250.,  -1
100,      67.,  -1
105,     200.,  -1
100,      23.,  -1
 ▪         ▪       \ key
date
        quantity
```

Figure 4.39
SR Data File With Sample Problem
Scheduled Receipts

stop, type quit'. The response is stored into the variable answer. If answer = 'change', the subroutine "dayorqty" is called. If answer = 'delete', the subroutine "delSR" is called. If answer = 'add', the subroutine 'addSR' is called. If answer = 'quit', the control returns to the subroutine 'IandSR'.

4.4.2.6 Subroutine "dayorqty"

4.4.2.6.1 Purpose

When the subroutine "dayorqty" is called upon, the MRP User is able to change existing SR data file records. If the scheduled receipts due date or quantity due has to be changed, the change is made with the subroutine "dayorqty" and the entire SR data file won't have to be recreated.

4.4.2.6.2 Variables

k	Integer type; values range from 1 to 9999. The SR record number, read in search for the record to be changed, is stored in the variable k.
key	Integer type; values range from -1 to 9999. The next SR for the part is stored on record number key. If key = -1, there are no more SR for the part specified.
newday	Integer type; values range from 1 to 9999. The expected date, for the corrected SR record, is stored into the variable newday.
newqty	Real type; format code f8.0. The expected quantity of parts, for the corrected SR record is stored into the

variable newqty.

oh — Real type; format code f8.0. The on hand inventory quantity is stored into the variable oh.

oldday — Integer type; values range from 1 to 9999. The expected date, on the SR record to be changed, is stored into the variable oldday.

oldqty — Real type; format code f8.0. The expected quantity of parts, on the SR record to be changed, is stored into the variable oldqty.

part — Character type; maximum of 10 alphanumeric digits. The company part number, which has had a change in scheduled receipt information, is stored into the variable part.

SR — Integer type; values range from -1 to 9999. The pointer, to the SR data file records for part, is the variable SR stored in the partstatus data file. If SR = -1, there are no scheduled receipt files for that part.

sys — Integer type; range of values -1 to 999. The system number of part is stored into the variable sys. If sys = -1, then part is not in the partid data file.

4.4.2.6.3 Description of Flow*

To change records in the SR data file, the subroutine "dayorqty" does the following steps:

Step 1. Ask the MRP User to specify the 'part, to be changed or quit to stop', and store the response into part.

---

* For flowchart, see Appendix C, page 542.

Step 2. If part = 'quit', go to subroutine "srchange".

Step 3. Call subroutine "findsys" to find the system part number (sys) for the company part number (part).

Step 4. Ask for the SR record information as it now exists and for the SR record information as it should exist.

Step 5. Find the old SR record that needs to be changed.

Step 6. Make the change; old = new

Step 7. Go to Step 1.

4.4.2.6.4 Example

To illustrate the use of the subroutine 'dayorqty', the SR data file created in Example 4.4.2.4.4. will have the following changes made: (1) from a, 100, 12 to a, 120, 12; (2) from f, 105, 200 to f, 105, 100; and (3) from c, 105, 250 to c, 95, 200. To make the changes the steps are as follows:

Step 1. To 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit', the MRP User response 'invandsr'.

Step 2. The statement -- 'To create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory quantities, type onhand. To stop, type quit'. -- receives the response 'SR'.

Step 3. The statement -- 'To change record in SR file, type change. To delete record in SR file, type delete. To include new records in SR, type add. To stop, type quit'. -- gets the response 'change'.

Step 4. To the 'part to be changed or quit to stop', the MRP User types 'a'.

Step 5. To the 'day, qty of SR as it now appears', the MRP User types '100, 12'.

Step 6. To the 'day, qty of SR as it should appear', the MRP User types '120, 12'.

Step 7. To the 'part to be changed or quit to stop', the MRP User types 'f'.

Step 8. To the 'day, qty of SR as it now appears', the MRP User types '105, 200'.

Step 9. To the 'day, qty of SR as it should appear', the MRP User types '105, 100'.

Step 10. To the 'part to be changed or quit to stop', the MRP User types 'c'.

Step 11. To the 'day, qty of SR as it now appears', the MRP User types '105, 250'.

Step 12. To the 'day, qty of SR as it should appear', the MRP User types '95, 200'.

Step 13. To the 'part to be changed or quit to stop', the MRP User types 'quit'.

The results are shown in Figure 4.40.

4.4.2.7 Subroutine "delSR"

4.4.2.7.1 Purpose

When the subroutine "delSR" is called by the subroutine "srchange", the MRP User is able to delete cancelled scheduled receipts from the SR

```
next
   *  fillin
   *    *
   9    0      SR
120,       12.,   -1
 95,      825.,    3
105,      270.,   -1
 95,      105.,    5
 95,      200.,   -1
100,       67.,   -1
105,      100.,   -1
100,       23.,   -1
  *          *      \ key
date
          quantity
```

Figure 4.40
SR Data File
After the Changes

data file. The records deleted are linked with a system of keys and a pointer on record Ø, which enables the record to be reused when scheduled receipts are added to the SR data file by subroutine "addSR".

4.4.2.7.2. Variables

BofM — See Section 4.3.2.2.2.

BOM — See Section 4.3.2.2.2.

ccost — See Section 4.3.2.2.2.

ccper — See Section 4.3.2.2.2.

day — Integer type; values range from 1 to 9999. The expected date of a scheduled receipt is read and stored with the variable day.

dayl — Integer type; values range from 1 to 9999. The values of day on record number kl is stored into the variable dayl.

fillin — Integer type; values range from Ø to 9999. The record number of an empty SR data file record is stored into the variable fillin on record Ø.

k — Integer type; values range from 1 to 9999. The record number of the SR record, that was deleted, is stored into the variable k. The value of k is determined during the search for the SR record to be deleted.

kl — Integer type; values range from -1 to 999. The value of key on record number kf.

key — Integer type; values range from -1 to 9999. The next SR record for part is stored into the variable key. If key = -1, there are no more SR records.

kf	Integer type; values range from 1 to 999. The record number kf is a deleted record in theSR data file.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
n	Integer type. The number of reads done to find the SR record to be deleted, is handled by the variables n.
next	Integer type; values range from 1 to 999. The quantity next -1 is the number of records in the SR data file.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
oldday	Integer type; values range from 1 to 999. The day expected, on the SR to be deleted, is stored into the variable oldday.
oldqty	Real type; format code f8.0. The quantity of units expected, on the SR to be deleted, is stored into the variable oldqty.
part	Character type; maximum of 10 alphanumeric digits. The part is the company part number which has had the scheduled receipt cancelled.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.

qty	Real type; format code f8.0. The expected quantity of units on a scheduled receipt is read and stored with the variable qty.
qty1	Real type; format code f8.0. The value of qty on record k1 is stored in the variable qty1.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.4.2.6.2.
sys	Integer type; values range from 1 to 999. The system part number for part is stored into the variable sys.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.4.2.7.3 Description of Flow*

To delete a record in the SR data file the subroutine "delSR" performs the following steps:

Step 1. Ask for the 'part to be deleted or quit to stop' and store the answer into part.

Step 2. If part = 'quit', return to the subroutine "srchange".

Step 3. Call the subroutine "findsys" to locate the system part number (sys) for part.

Step 4. Ask the MRP User to specify the 'day, qty as it now appears in the SR file' and store the response in oldday and oldqty, respectively.

Step 5. Find the file that is to be deleted.

---

* For flowchart, see Appendix C, page 544.

Step 6. Adjust the pointer fillin on record Ø and the keys to the next deleted SR record.

Step 7. Go to Step 1.

4.4.2.7.4. Example

To illustrate the use of "delSR" subroutine, the SR data file generated will have the following deletions made: (1) a, 120, 12; (2) f, 105, 100; and (3) c, 95, 200. To make the deletions the steps are as follows:

Step 1. To 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit', the MRP User types 'invandsr'.

Step 2. The statement - 'To create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory quantities, type onhand. To stop, type quit? -- receives the response 'SR'.

Step 3. The statement -- 'To change record in SR file, type change. To delete record in SR file, type delete. To include new records in SR, type add. To stop, type quit' -- gets the response 'delete'.

Step 4. To 'part to be deleted or quit to stop', the MRP User types 'a'.

Step 5. To 'day, qty as it now appears in SR file', type MRP User types '120, 12'. See Figure 4.41.

Step 6. To 'part to be deleted or quit to stop', the MRP User types 'f'.

Step 7. To 'day, qty as it now appears in SR file', the MRP User types '105, 100'. See Figure 4.42.

Step 8. To 'part to be deleted or quit to stop', the MRP User types 'a'.

Step 9. To 'day, qty as it now appears in SR file', the MRP User types '85, 200'. See Figure 4.43.

Step 10. To 'part to be deleted or quit to stop', the MRP User types 'quit'.

4.4.2.8 Subroutine "addSR"

4.4.2.8.1 Purpose

When the subroutine "addSR" is called by the subroutine "srchange", the MRP User is able to add new scheduled receipt records to the SR data file.

4.4.2.8.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
d	Integer type; values range from 1 to 9999. The dl value on record fillin is stored into the variable d.
dl	Integer type; values range from 1 to 9999. The variable dl is the expected scheduled receipt date on record k.
fillin	Integer type; values range from 0 to 9999. The record number of a scheduled receipt, that has been deleted, is stored into the variable fillin.

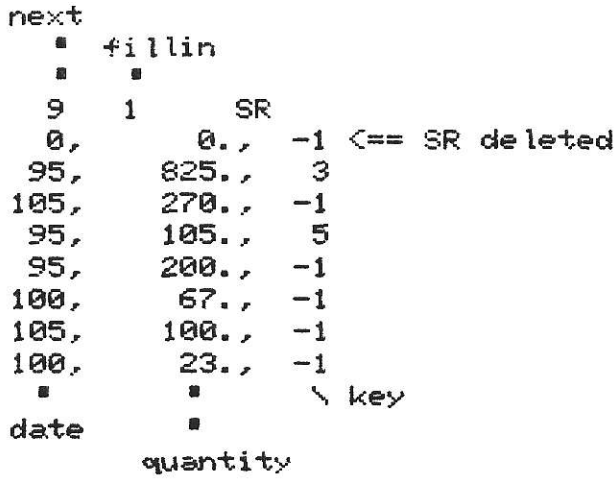


Figure 4.41
SR With 'a,120,12' Deleted

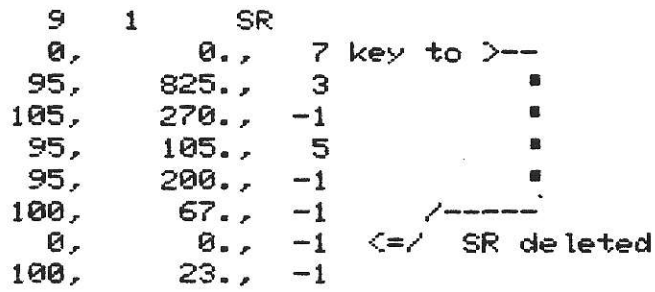


Figure 4.42
SR With 'f,105,100' Deleted

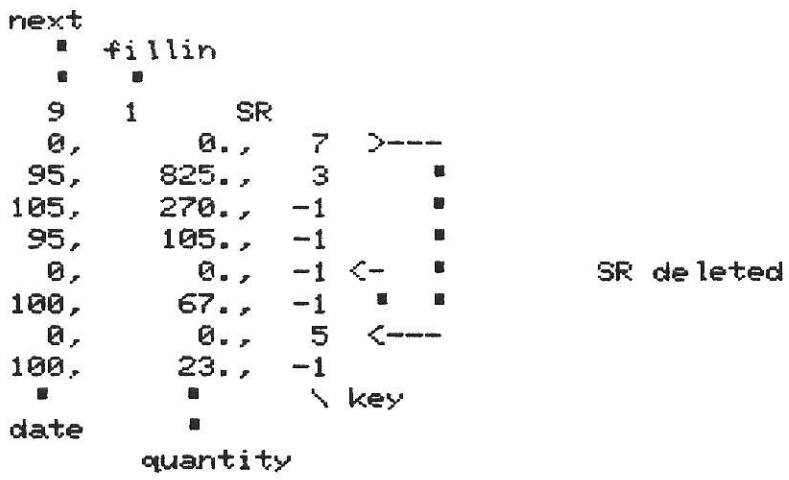


Figure 4.43
SR With 'c,95,200' Deleted

k	Integer type; values range from 1 to 9999. The record number k is the last SR record for part. Also, the record number, of the next fillin record in the SR data file, is stored into the variable k.
kl	Integer type; values range from -1 to 9999. The record number of the next SR is stored into the variable kl. If kl =1, there are no more SR for that part.
key	Integer type; values range from -1 to 9999. The location of the new SR record is the record number key.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
newday	Integer type; values range from 1 to 9999. The expected date of the SR is stored into the variable newday.
newqty	Real type; format code f8.0. The expected quantity of units in the SR is stored into the variable newqty.
next	Integer type; range of values 1 to 9999. The next available record in the SR data file is stored into the variable next.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	Character type; maximum of 10 alphanumeric digits. The company part number, for the scheduled receipt to be added,

	is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
q	Real type; format code f8.0. The value of ql on record fillin is stored into the variable q.
ql	Real type; format code f8.0. The quantity of units expected, located on record k in the SR data file, is stored into the variable ql.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.4.2.6.2.
sys	Integer type; range of values -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, the part is not in the partid data file.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.4.2.8.3 Description of Flow*

To add new SR records to the SR data file, the subroutine "addSR" does the following steps:

Step 1. Asks for the 'part to be added or quit to stop' and stores the response into part.

Step 2. If part = 'quit', return to subroutine "srchange".

Step 3. Asks for the 'day,qty' for the new SR records; and stores the responses into variable newday and newqty, respectively.

---

* For flowchart, see Appendix C, page 548.

Step 4. Records the new scheduled receipt information. The location depends on the values of fillin and next. If fillin = Ø, then next is the record number for the new SR record. But if fillin > Ø, then fillin is the record number for the new SR record.

Step 5. Fix the values for variables next, fillin, SR and key.

Step 6. Go to Step 1.

4.4.2.8.4 Example

To illustrate the use of the 'addSR' subroutine, the scheduled receipts -(1) a, 100, 12, (2) f, 105, 200, (3) c, 105, 250, and (4) j, 100, 10 -- are added. The steps are as follows:

Step 1. To 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit', the MRP User types 'invandsr'.

Step 2. The statement --'To create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory quantities, type onhand. To stop, type quit' -- receives the response 'SR'.

Step 3. The statement - 'To change record in SR file, type change. To delete record in SR file, type delete. To include new records in SR, type add. To stop, type quit', -- get the response 'add'.

Step 4. To 'part to be added or quit to stop', the MRP User types'a'.

Step 5. To 'day, qty', the MRP User types '100, 12'.

Step 6. To 'part to be added or quit to stop', the MRP User types 'f'.

Step 7. To 'day, qty', the MRP User types '105, 200'.

Step 8. To 'part to be added or quit to stop', the MRP User types 'c'.

Step 9. To 'day, qty', the MRP User types, '105, 250'.

Step 10. To 'part to be added or quit to stop', the MRP User types 'j'.

Step 11. To 'day, qty', the MRP User types '100, 10'.

Step 12. To 'part to be added or quit to stop', the MRP User types 'quit'.

For results, see Figure 4.44.

4.4.2.9 Subroutine "ohchange"

4.4.2.9.1 Purpose

When the subroutine "IandSR" calls the subroutine "ohchange", the MRP User is able to interactively correct the on hand inventory quantities which are stored on the partstatus data file record.

4.4.2.9.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.

```
next
   *  fillin
   *   *
 10    0      SR
100,      12.,   -1   <=   addition 1
 95,     825.,    3
105,     270.,   -1
 95,     105.,    5
105,     250.,   -1   <== addition 3
100,      67.,   -1
105,     200.,   -1   <=== addition 2
100,      23.,   -1
100,      10.,   -1   <==== addition 4
  *        *       \ key
date       *
        quantity
```

Figure 4.44
SR After Additions Made

levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	Character type, maximum of 10 alphanumeric digits. The part is the company part number which needs its on hand inventory quantity changed.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
qty	Real type; format code f8.0. The corrected on hand inventory quantity is stored into the variable qty.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; values range from -1 to 999. The system part number of part is stored into the variable sys. If sys = -1, then part is not in the partid data file.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.4.2.9.3. Description of Flow*

To change on hand inventory quantities, the subroutine "ohchange" performs the following steps:

---

* For flowchart, see Appendix C, page 552.

Step 1. Ask the MRP User for the 'part, on hand quantity or quit, Ø' and store the response into the variables part, qty, respectively.

Step 2. If part = 'quit', return to subroutine "IandSR".

Step 3. Call "findsys" subroutine to locate the system part number (sys) for part.

Step 4. If sys = -1. go to Step 1.

Step 5. Set oh = qty on the part's partstatus record.

Step 6. Go to Step 1.

4.4.2.10 Subroutine 'leadandlot'

4.4.2.10.1 Purpose

When the MRP User indicates the desire to edit lead times and lotting rules, the subroutine 'leadandlot' asks the MRP User to specify either just lead times are to be edited, or just lotting rules are to be editted, or both lead times and lotting rules are to be edited.

4.4.2.10.2 Variables

answer — Character type; answer ε {ltlr, lt, lr, quit}. The variable answer contains the response to the question on the type of editing to be done

4.4.2.10.3. Description of Flow*

The MRP User is asked to respond to the following "To enter both lead time and lotting rule, type ltlr. To enter just lead time, type lt. To enter just lotting rule, type lr. To stop, type quit'. The response is stored into the variable answer. If answer = 'ltlr', the subroutine

* For flowchart, see Appendix C, page 553.

"leadtimesandlotrule" is called. If answer = 'lt', the subroutine "leadtime" is called. If answer = 'lr', the subroutine "lotrule" is called, and if answer = 'quit', the program control returns to the main program.

4.4.2.11 Subroutine 'leadtimeandlotrule'

4.4.2.11.1 Purpose

When the MRP User wants to enter both lead times and lotting rules, the subroutine "leadtimeandlotrule' is designed to work interactively with the MRP User to accomplish this task.

4.4.2.11.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
lead	Integer type; values range from 1 to 9999. The lead time in day is stored into the variable lead.
leadtime	Integer type; values range from 1 to 9999. The number of days for lead time of part is stored into the variable leadtime.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	Integer type; values range from 1 to 9999. The number of days for FPQ and L4L is stored into the variable lotp.
lotr	Integer type; values range from 0 to 9. The lotting rule is coded with a single digit number; 1 = L4L, 2 = FPQ.

lotrule	Character type; lotrule {L4L, FPQ}. The lotrule is the lotting technique for determining the lot size of part.
MPSkey	See Section 4.3.2.2.2.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	Character type; maximum of 10 alphanumeric digits. The company part number, which is to have its lead time and lot rule specified, is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; range of values -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, part is not in the partid data file.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.4.2.11.3 Description of Flow*

To enter both lead times and lotting rules, the subroutine "leadtimeandlotrule" performs the following steps:

Step 1. Ask the MRP User to specify the 'part, lead time (days), lot rule (L4L, FPQ)' and store the response into part, leadtime, lotrule, respectively.

---

* For flowchart, see Appendix C, page 554.

Step 2. If part = 'quit', return to subroutine "leadandlot".

Step 3. Call "findsys" to obtain the system part number (sys) for part.

Step 4. If sys = -1, go to Step 1.

Step 5. Obtain any additional information needed for lotting rule.

Step 6. Store information on partstatus record number sys.

Step 7. Go to Step 1.

4.4.2.11.4 Example

To illustrate the use of subroutine "leadtimeandlotrule", the lead times and lotting rules from the sample problem are entered. The steps are as follows:

Step 1. To 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph, To stop, type quit', the MRP User types 'ltandlr'.

Step 2. The statement - 'To enter both lead time and lotting rule, type ltlr. To enter just lead time, type lt. To enter just lotting rule, type lr. To stop, type quit', - gets the response 'ltlr'.

Step 3. To 'part, lead time (days), lot rule (L4L, FPQ) or quit, Ø, q to stop', the MRP User types 'a, 10, L4L'.

Step 4. To 'part, lead time (days), lot rule (L4L, FPQ) or quit, Ø, q to stop', the MRP User types, 'b, 20, FPQ'.

Step 5. To 'period (days) for FPQ', the MRP User types '10'.

.
.
.

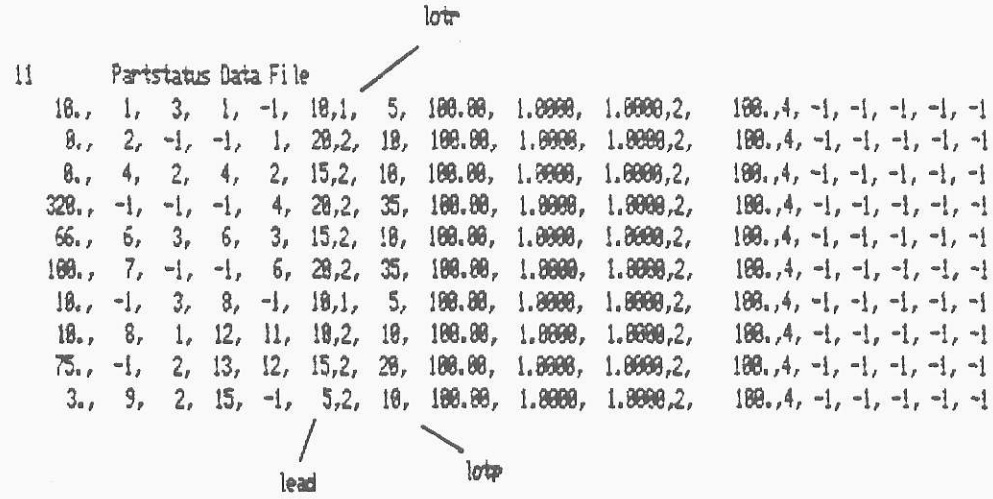


Figure 4.45
The partstatus With Lead Times
and Lot Rules

Note: L4L = 1 and FPQ = 2

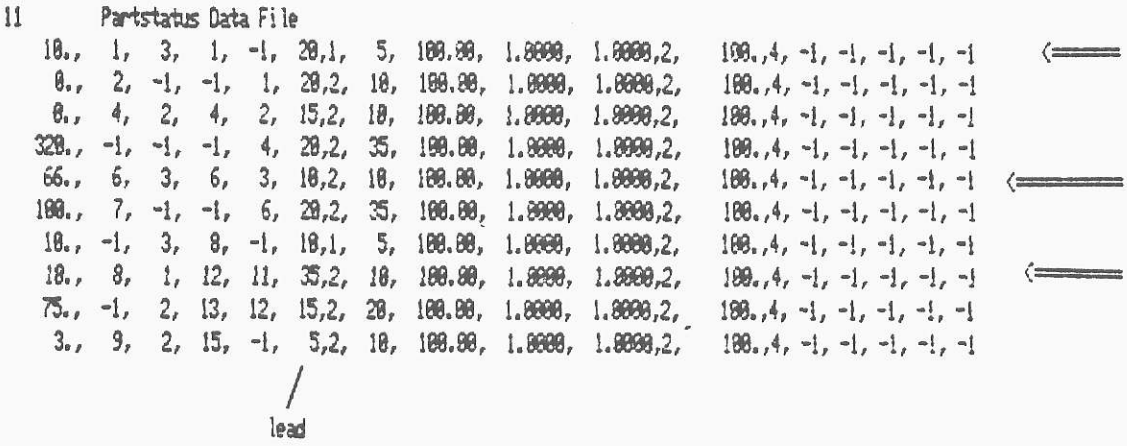


Figure 4.46
The partstatus With Lead Time Changes

Step n. To 'part, lead time (days), lot rule (L4L< FPQ) or quit, Ø, q to stop', the MRP User types 'j, 5, FPQ'.

Step n+1. To 'period (days) for FPQ', the MRP User types '10'.

Step n+2. To 'part, lead time (days), lot rule (L4L, FPQ) or quit, Ø, q to stop', the MRP User types 'quit, Ø, q'.

4.4.2.12 Subroutine 'leadtime'

4.4.2.12.1 Purpose

When the MRP User needs to edit only the lead times, then the subroutine "leadtime" is designed to interact with the MRP User to accomplish this task.

4.4.2.12.2 Variables

BoM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
lead	See Section 4.3.2.2.2.
Leadtime	Integer type; values range from 1 to 9999. The lead time in days is stored into the variable Leadtime.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2. .
MPSkey	See Section 4.3.2.2.2.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.

part	Character type; maximum of 10 alphanumeric digits. The company part number, for the part whose lead time is to be entered, is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; range of values -1 to 999. The system part number of part is stored into the variable sys. If sys = -1, part is not in the partid data file.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.4.2.12.3 Description of Flow*

To enter just lead times, the subroutine "leadtime" performs the following steps:

Step 1. Ask the MRP for the 'part, lead time (days) or quit, Ø to stop' and store the response into variables part, Leadtime, respectively.

Step 2. If part = 'quit', return to subroutine "leadandlot".

Step 3. Call the subroutine "findsys" to obtain the system part number (sys) for part.

Step 4. If sys = -1, go to Step 1.

---

* For flowchart, see Appendix C, page 557.

Step 5. Set lead = Leadtime on record number sys in the partstatus data file.

Step 6. Go to Step 1.

4.4.2.12.4 Example

To illustrate the use of subroutine "leadtime", the partstatus data file generated in Example 4.4.2.11.4 will have the following lead time changes made:

(1) for a from 10 to 20;

(2) for e from 15 to 10; and

(3) for h from 10 to 35.

To make the changes, the steps are as follows:

Step 1. To 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit', the MRP User types 'ltandlr'.

Step 2. The statement - 'To enter both lead time and lotting rule, type ltlr. To enter just lead tme, type lt. To enter just lotting rule, type lr. To stop, type quit,' - gets the response 'lt'.

Step 3. To 'part, lead time (days) or quit, Ø to stop', the MRP User types 'a, 20'.

Step 4. To 'part, lead time (days) or quit, Ø to stop', the MRP User types 'e, 10'.

Step 5. To 'part, lead time (days) or quit, Øto stop', the MRP User types 'h, 35'.

Step 6. To 'part, lead time (days) or quit, Ø to stop', the MRP User types 'quit, Ø'.

For results, see Figure 4.46.

4.4.2.13 Subroutine "lotrule"

4.4.2.13.1 Purpose

When the MRP User needs to edit only the lotting rules, then the subroutine 'lotrule' is designed to work interactively with the MRP User to accomplish this task.

4.4.2.13.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
Lotrule	Character type; Lotrule $\in$ {L4L, FPQ}. The lotrule is the variable used to store lotting technique for determining the lot size of part.
MPSkey	See Section 4.3.2.2.2.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.

part	Character type; maximum of 10 alphanumeric digits. The company part number, which is to have its lotting rule specified, is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.4.2.13.2 Description of Flow*

To enter just lotting rules. the subroutine "lotrule" executes the following steps:

Step 1. Ask the MRP User for the 'part. lot rule (L4L, FPQ) or quit, q to stop', and store the response into the variables part, Lotrule, respectively.

Step 2. If part = 'quit'. return to subroutine "leadandlot".

Step 3. Call subroutine "findsys" to get the system part number (sys) for part.

Step 4. If sys = -1. go to Step 1.

Step 5. Obtain additional information for the lotting rule if necessary.

---

* For flowchart, see Appendix C, page 558.

Step 6. Store the information on record number sys in the partstatus data file.

Step 7. Go to Step 1.

4.4.2.13.4 Example

To illustrate the use of the subroutine "lotrule", the partstatus data file generated in Example 4.4.2.12.4 will have the following lot rules changed:

(1) for b from FPQ, 10 to FPQ, 20;

(2) for d from FPQ, 35 to L4L;

(3) for g from L4L to FPQ, 10.

To make the changes the steps are as follows:

Step 1. To 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Period for the planning horizon, type ph. To stop, type quit', the MRP User types 'ltandlr'.

Step 2. The statement - 'To enter both lead time and lotting rule, type ltlr. To enter just lead time, type lt. To enter just lotting rule, type lr. To stop, type quit', -- gets the response 'lr'.

Step 3. To 'part, lotrule (L4L, FPQ) or quit, q to stop', the MRP User types 'b,FPQ'.

Step 4. To 'period (days) for FPQ', the MRP User types '20'.

Step 5. To 'part, lot rule (L4L, FPQ) or quit, q to stop', the MRP User types 'd, L4L'.

Step 6. To 'part, lot rule (L4L, FPQ) or quit, q to stop', the MRP User types 'g, FPQ'.

Step 7. To 'period (days) for FPQ', the MRP User types '10'.

For the results, see Figure 4.47.

4.4.2.14 Subroutine "horizon"

4.4.2.14.1 Purpose

The subroutine "horizon" has the responsibility to interact with the MRP User to setup the dates for the planning horizon.

4.4.2.14.2 Variables

answer — Character type; answer ∈ {yes, no}. The response to question 'Equal bucket size? yes or no', is stored into the variable answer.

buckets — Integer type. The number of periods, or buckets, in the planning horizon is stored into the variable buckets.

nperiod — Integer type; values range from 1 to 999. The number of the current period is stored into the variable nperiod.

4.4.2.14.3 Description of Flow*

To setup the planning horizon dates, the subroutine "horizon" does the following steps:

Step 1. Ask the MRP User to specify the 'number of buckets?' and store the response into the variable buckets.

Step 2. Ask the MRP User for the planning horizon to 'Start with period number', and store the response into the variable nperiod.

---

* For flowchart, see Appendix C, page 560.

Step 3. Ask the MRP User "Equal bucket size? yes or no', and store the response into the variable answer.

Step 4. If answer = 'yes' call subroutine "equalbucket".

Step 5. If answer = 'no', call subroutine "unequalbucket".

4.4.2.15 Subroutine "equalbucket"

4.4.2.15.1 Purpose

To create a planning horizon with equal buckets, which has size of five days per bucket, the MRP User specifies the number of buckets and the period number for the current period. Then the subroutine "equalbucket" does the rest.

4.4.2.15.2 Variables

buckets	See Section 4.4.2.14.2
dayone	Integer type; values range from 1 to 9999. The first day of the planning bucket is stored into the variable dayone.
i	Integer type; values range from 1 to buckets. To do loop index and the record numbers in the horizon file are handled by the variable i.
nperiod	See Section 4.4.2.14.2.

4.4.2.15.3 Description of Flow*

To create a planning horizon of equal bucket sizes, the subroutine "equalbucket" performs the following steps:

Step 1. The values of buckets and nperiod are obtained from the subroutine "horizon".

* For flowchart, see Appendix C, page 562.

Step 2. Write the value of buckets on record Ø of the horizon file.

Step 3. Assign dayone = period number *5 to each bucket and record in horizon file.

4.4.2.15.4 Example

To illustrate the use of the subroutine "horizon" with the subroutine "equalbucket", the planning horizon for the sample problem is developed with the following steps:

Step 1. To 'What do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Period for the planning horizon, type ph. To stop, type quit,' the MRP User types 'ph'.

Step 2. To 'number of buckets?' the MRP User types '11'.

Step 3. To 'Start with period number', the MP User types '19'.

Step 4. To 'Equal bucket size? yes or no'. the MRP User types 'yes'.

The results are shown in Figure 4.48.

4.4.2.16 Subroutine "unequalbucket"

4.4.2.16.1 Purpose

When the MRP User desires to have a planning horizon without equal buckets of 5 days per bucket, then the subroutine "unequalbuckets" works interactively with the MRP User to assign different dates to each bucket.

4.4.2.16.2 Variables

buckets	See Section 4.4.2.14.2.
dayone	See Section 4.4.2.15.2.
i	See Section 4.4.2.15.2.

nperiod See Section 4.4.2.14.2.

4.4.2.16.3 Description of Flow*

To assign dates to the unequal buckets in the planning horizon, the subroutine "unequalbucket" performs the following steps:

Step 1. The values of buckets and nperiod are obtained from the subroutine "horizon".

Step 2. The value of buckets is written on record Ø.

Step 3. For each planning period, the computer asks 'day one is __' and stores the information in the horizon file.

4.4.2.16.4 Example

To illustrate the use of the subroutine "horizon" with the subroutine "unequalbucket", the planning horizon, with the dates per periods shown in Figure 4.49, is developed by the following steps:

Step 1. See Step 1 in Example 4.4.2.15.4.

Step 2. See Step 2 in Example 4.4.2.15.4.

Step 3. See Step 3 in Example 4.4.2.15.4.

Step 4. To 'Equal bucket size? yes or no', the MRP User types 'no'.

Step 5. To 'Period 19 day one is ___', the MRP User types '95'.

Step 6. To 'Period 20 dayone is ___', the MRP User types '100'.

.
.
.

Step n. To 'Period 28 dayone is ___', the MRP User types '160'.

Step n+1. To 'Period 29 dayone is ___', the MRP User tyes '170'.

The results are shown in Figure 4.50.

* For flowchart, see Appendix C, page 563.

```
buckets
 .
 .
11      Planning Horizon
19,  95
20, 100
21, 105
22, 110
23, 115
24, 120
25, 125
26, 130
27, 135
28, 140
29, 145
 .      \ day one
period
number
```

Figure 4.48
The horizon With Equal Buckets

```
                                    lotr          lotp
11      Partstatus Data File
   10.,  1,  3,  1, -1, 20,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
    0.,  2, -1, -1,  1, 20,2, 20, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1   <=====
    0.,  4,  2,  4,  2, 15,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  320., -1, -1, -1,  4, 20,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1   <=====
   65.,  6,  3,  6,  3, 10,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  100.,  7, -1, -1,  6, 20,2, 35, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   10., -1,  3,  9, -1, 10,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1   <=====
   10.,  8,  1, 12, 11, 35,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   75., -1,  2, 13, 12, 15,2, 20, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
    3.,  9,  2, 15, -1,  5,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
```

Figure 4.47
The partstatus With Lot Rule Changes

Period	_Day_
19	95
20	100
21	105
22	110
23	115
24	120
25	130
26	140
27	150
28	160
29	170

Figure 4.49
Unequal horizon Buckets

```
buckets
 .
 .
11      Planning Horizon
19,  95
20, 100
21, 105
22, 110
23, 115
24, 120
25, 130
26, 140
27, 150
28, 160
29, 170
 .      \ day one
period
number
```

Figure 4.50
Unequal Buckets in horizon Files

4.5 DataPlus2

4.5.1 Introduction to DataPlus2

The input file called DataPlus2 has the responsibility to help the MRP User to edit the MPS data file, the XO data file, and the SP data file. The MPS data file contains the information from the Master Production Schedule end item demands. The XO data file contains the information on the External Order demands. The SP data file contains the information on the Service Part (or Spare Part) demands. The editing optons for each file is to create a new file, to add new records to the file, and to change an existing record.

4.5.2 Subroutines of DataPlus2

4.5.2.1 Overview of Subroutines

The DataPlus2 input file consists of the following subprograms:

1. Subroutine "findsys",
2. Subroutine "MSdemands",
3. Subroutine "MSnew",
4. Subroutine "MSadd",
5. Subroutine "MSchange",
6. Subroutine "XOdemands",
7. Subroutine "XOnew",
8. Subroutine "XOadd",
9. Subroutine "XOchange",
10. Subroutine "SPdemands",
11. Subroutine "SPnew",
12. Subroutine "SPadd",

13. Subroutine "SPchange".

See Figure 4.51.

The main program inquires about whether the MRP User wishes to edit the MS (Master Schedule, Master Production Schedule, end-items) demands or XO (External Order) demands or SP (Service Part, Spare Part) demands.

If MS demands are specified for editing, then the subroutine "MSdemands" is called. The "MSdemands" subroutine asks the MRP User to specify further as to whether he wishes to create a new MPS data file, to add new records to the MPS data file, or to change existing records in the MPS data file. To create a new MPS data file, the subroutine "MSnew" is used. To add new records to the MPS data file, the subroutine "MSadd" is used. To change existing records in the MPS data file, the subroutine "MSchange" is called by subroutine "MSdemands".

When XO demands are specified for editing, then the "XOdemands" subroutine calls "XOnew", "XOadd", or "XOchange" depending on the editing preference indicated by the MP User. Similarily, when SP demands are indicated for editing, then the subroutines "SPdemands", "SPnew", SPadd", or "SPchange" are available to accomplish this task.

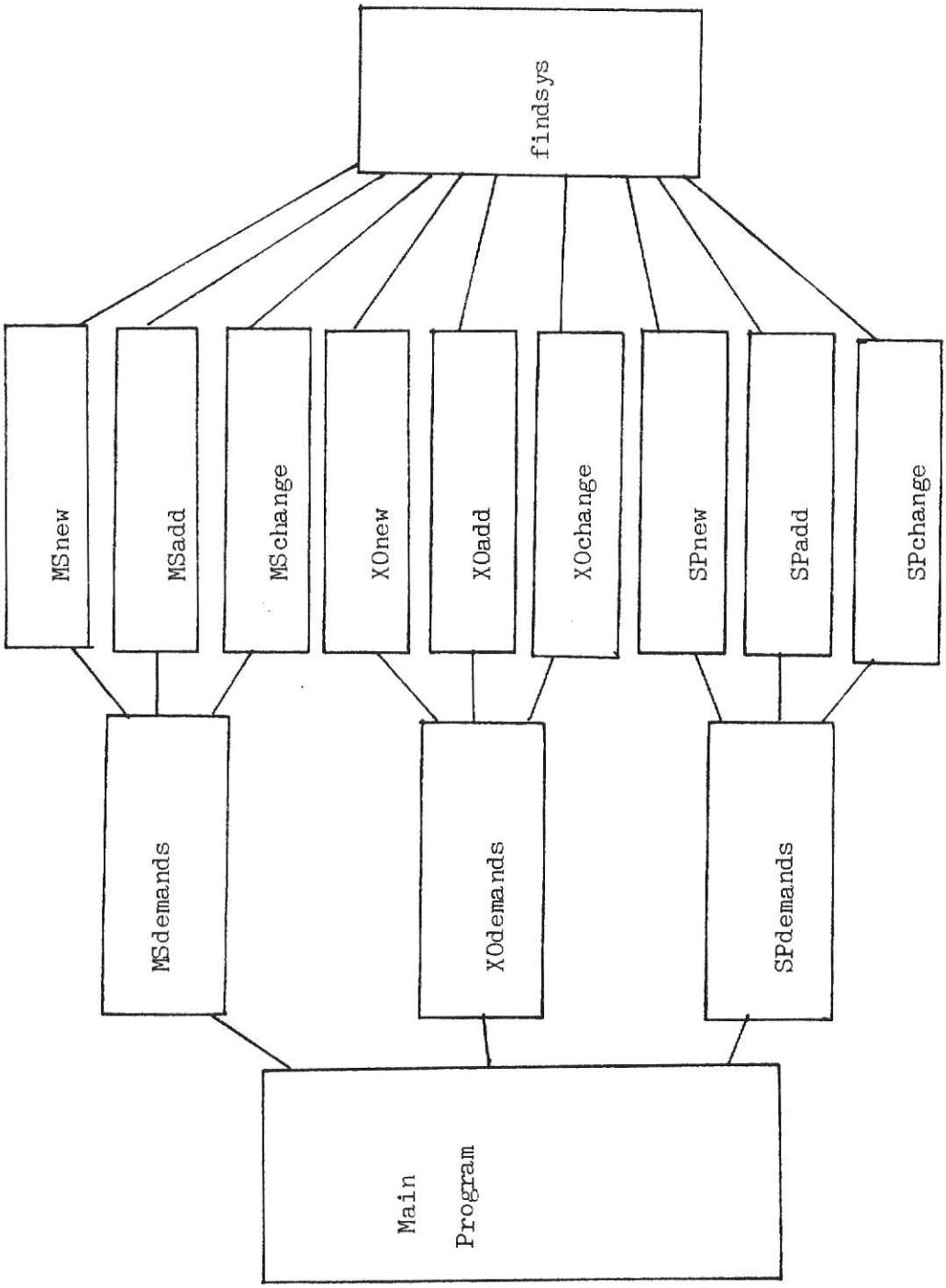


Figure 4.51
DataPlus2 Subroutines

4.5.2.2 Subroutine "findsys"

See Section 4.3.2.4.

4.5.2.3 Subroutine "MSdemands"

4.5.2.3.1 Purpose

When the MRP User states that the MPS data file is to be edited, the subroutine "MSdemands" asks whether the purpose of the editing is to create a new file, add new file records, or change file records.

4.5.2.3.2 Variables

answer — Character type; answer {new, add, change, quit}. The response to the editing preference is stored into the variable answer.

4.5.2.3.3 Description of Flow*

The subroutine "MSdemands" asks the MRP User 'new, add, change, or quit MS demand list'. The response is stored into the varible answer.

If answer = 'new', then the subroutine "MSnew" is called.

If answer = 'add', the the subroutine "MSadd" is called.

If answer = 'change', then the subroutine "MSchange" is called.

If answer = 'quit', the program control returns to the main program.

4.5.2.4 Subroutine "MSnew"

4.5.2.4.1 Purpose

When a new MPS data file is to be created, the subroutine "MSnew" works interactively with the MRP User to accomplish this task.

---

* For flowchart, see Appendix C, page 565.

4.5.2.4.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
D	Real type; format code f8.0. The demand on record mskey is stored into the variable D.
Da	Integer type; values range from 1 to 9999. The date on record mskey is stored into the variable Da.
date	Integer type; values range from 1 to 9999. The due date for the MPS demand is stored into the variable date.
demand	Real type; format code f8.0. The quantity of parts needed to satisfy the MPS demand is stored into the variable demand.
k	Integer type; values range from 1 to 9999. The keyms on record mskey is stored into the variable k.
keyms	Integer type; values range from +1 to 9999. The variable keyms is the key placed on record mskey to point to the next MPS demand for part.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.

mskey	Integer type; values range from 1 to 9999. The variable mskey is the record number in the MPS data file where the demand information is written.
n	Integer type. The number of demand entries for a given part is stored into the variable n.
next	Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of records in the partstatus data file.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	Character type; maximum of 10 alphanumeric digits. The company part number with the MPS demands is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; values range from -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, then part is not in the partid data file.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.5.2.4.3 Description of Flow*

To create a new MPS data file, the subroutine "MSnew" performs the following steps:

Step 1. Reinitialize the MPSkey in the partstatus data file to -1.

Step 2. Initialize both mskey and keyms to Ø.

Step 3. Ask the MRP User 'part or quit to stop' and store the response into the variable part.

Step 4. If part = 'quit', return to "MSdemands" subroutine.

Step 5. Call subroutine "findsys" to obtain the system part number (sys) for part.

Step 6. If sys = -1, go to Step 3.

Step 7. Read record number sys in the partstatus data file.

Step 8. Obtain and store the MPS demands for part. List the record number for the first entry on the partstatus record sys. Denote the last entry for part with keyms = -1.

Step 9. Go to Step 3.

4.5.2.4.4 Example

To illustrate the use of the subroutine "MSnew", some of the Master Production schedule for parts a, g, and j in the sample problem are stored into the MPS data file. The steps are as follows:

Step 1. The statement - 'To enter MSdemands, type MS. To enter external orders, type XO. To enter service parts, type SP. To stop, type quit', -- gets the response of 'MS'.

---

* For flowchart, see Appendix C, page 566.

Step 2. To 'new, add, change, or quit MS demands list?', the MRP User types 'new'.

Step 3. To 'part or quit to stop', the MRP User types 'a'.

Step 4. To 'date, demand or -1, Ø to stop', the MRP User types '95, 10'.

Step 5. To 'date, demand or -1, Ø to stop', the MRP User types '100, 12'.

Step 6. To 'date, demand or -1, Ø to stop', the MRP User types '105, 14'.

Step 7. To 'date, demand or -1, Ø to stop', the MP User types '-1, Ø'.

Step 8. To 'part or quit to stop', the MRP User types 'g'.

Step 9. To 'date, demand or -1, Ø to stop', the MRP User types '95, 9'.

Step 10. To 'date, demand or -1, Ø to stop', the MRP User types '100, 1'.

Step 11. To 'date, demand or -1, Ø to stop', the MRP User types '-1, Ø'.

Step 12. To 'part or quit to stop', the MRP User types 'j'.

Step 13. To 'date, demand or -1, Ø to stop', the MRP User types '95, 3'.

Step 14. To 'date, demand or -1, Ø to stop', the MRP User types '-1, Ø.

Step 15. To 'part or quit to stop', the MRP User types 'quit'.

For results, see Figure 4.52.

```
11      Partstatus Data File
   10.,  1,  3,  1, -1, 20,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4,  3, -1, -1, -1, -1
    0.,  2, -1, -1,  1, 20,2, 20, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
    0.,  4,  2,  4,  2, 15,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  320., -1, -1, -1,  4, 20,1,  5, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   66.,  6,  3,  6,  3, 10,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
  100.,  7, -1, -1,  6, 20,2, 35, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   10., -1,  3,  8, -1, 10,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4,  4, -1, -1, -1, -1
   10.,  8,  1, 12, 11, 35,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
   75., -1,  2, 13, 12, 15,2, 20, 100.00, 1.0000, 1.0000,2,   100.,4, -1, -1, -1, -1, -1
    3.,  9,  2, 15, -1,  5,2, 10, 100.00, 1.0000, 1.0000,2,   100.,4,  6, -1, -1, -1, -1
```

```
next
  ▪
  ▪
  7       MPS
 95,       10.,    2
100,       12.,    3
105,       14.,   -1
 95,        9.,    5
100,        1.,   -1
 95,        3.,   -1
  ▪          ▪       \ key
 date    quantity
needed
```

Figure 4.52
MPS File, Plus
partstatus File With
MPSkey Assigned

4.5.2.5 Subroutine "MSadd"

4.5.2.5.1 Purpose

To add records to the MS data file, the subroutine "MSadd" works interactively with the MRP User to accomplish this task.

4.5.2.5.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
D	Real type; format code f8.0. The quantity of part needed is stored into variable D.
Da	Integer type; values range from 1 to 9999. Due date for part is stored into the variable Da.
date	See Section 4.5.2.4.2
demand	See Section 4.5.2.4.2
k	Integer type; values range from 1 to 9999. The record number for the read, used to find last demand listed in MPS file, is stored into the variable k.
key	Integer type; values range from -1 to 999. Part's next MPS demand appears on record number key. If key = -1, then there are no more MPS demands for part.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.

lotr	See Section 4.3.2.2.2.
MPSkey	Integer type, values range from -1 to 9999. The record number in the MPS data file where the first MPS demand is stored into the variable MPSkey. If MPSkey = -1, no demand records for part are on file.
next	Integer type; values range from 1 to 9999. The variable next is equal to the next available record number.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	See Section 4.5.2.4.2.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	See Section 4.5.2.4.2
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.5.2.5.3 Description of Flow*

To add new records to the MPS data file, the subroutine "MSadd" performs the following steps:

Step 1. Read the next available record number for the MPS data file from record Ø.

---

* For flowchart, see Appendix C, page 569.

Step 2. Ask the MRP User for the 'part, date, demand quantity or quit, Ø, Ø to stop' and store the response into variables part, date, demand, respectively.

Step 3. If part = 'quit', return to subroutine "MSdemands".

Step 4. Call upon subroutine "findsys" to obtain the system part number (sys) for part.

Step 5. If sys = -1, go to Step 2.

Step 6. Write the new addition on record next.

Step 7. Link the new demand for part to the last demand listed in the MPS data file.

Step 8. Go to Step 2.

4.5.2.5.4 Example

To illustrate the use of the subroutine "MSadd", the following data:

part	day	quantity
g	105	10
j	100	4
j	105	4

is added to the MPS data file developed in Example 4.5.2.4.4. The steps are as follows:

Step 1. The statement - 'To enter MSdemands, type MS. To enter external orders, type XO. To enter service parts, type SP. To stop, type quit', - gets the response of 'MS'.

Step 2. To 'new, add, change, or quit MSdemands list?', the MRP User types 'add'.

Step 3. To 'part, date, demand quantity or quit, Ø, Ø to stop', the MRP User types 'g, 105, 10'.

Step 4. To 'part, date, demand quantity or quit, Ø, Ø to stop', the MRP User types 'j, 100, 4'.

Step 5. To 'part, date, demand quantity or quit, Ø, Ø to stop', the MRP User types 'j, 105, 4'.

Step 6. To 'part, date, demand quantity or quit, Ø, Ø to stop', the MRP User types 'quit, Ø, Ø'.

The results are shown in Figure 4.53.

4.5.2.6 Subroutine "MSchange"

4.55.2.6.1 Purpose

To change MPS demand records in the MPS data file, the subroutine "MSchange" works interactively with the MRP User to accomplish this task.

4.5.2.6.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
date	Integer type; values range from 1 to 9999. The MPS demand date, read from record k, is stored into the variable date.
demand	Real type; format code f8.0. The MPS demand quantity, read on record k, is stored into the variable demand.
k	Integer type; values range from 1 to 9999. The record number, read in search of the old MPS record, is stored into the variable k.

```
next
  .
  .
 10      MPS
 95,      10.,    2
100,      12.,    3
105,      14.,   -1
 95,       9.,    5
100,       1.,    7
 95,       3.,    8
105,      10.,   -1
100,       4.,    9
105,       4.,   -1
  .        .       \ key
 date    quantity
needed
```

Figure 4.53
MPS File With Additions

```
next
  .
  .
 10      MPS
 95,      10.,    2
100,      12.,    3
110,      20.,   -1   <== change
100,       9.,    5   <==== change
100,       1.,    7
 95,       3.,    8
105,      10.,   -1
100,       4.,    9
105,       4.,   -1
  .        .       \ key
 date    quantity
needed
```

Figure 4.54
MPS File With Changes

key — Integer type; value range from -1 to 9999. The MPS demand key to the next demand for part is stored into the variable key. If key = -1, no more MPS demands are in the file for part.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — Integer type; values range from -1 to 9999. The variable MPSkey is the pointer to the next MPS demand for part. If MPSkey = -1, nomore demands are on file.

newdate — Integer type; values range from 1 to 9999. The new date for the MPS record is stored into the variable newdate.

newdemand — Real type; format code f8.0. The new demand quantity for the MPS record is stored into the variable newdemand.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

olddate — Integer type; values range from 1 to 9999. The date of the MPS demand to be changed is stored into the variable olddate.

olddemand — Real type; format code f8.0. The demand quantity of the MPS demand to be changed is stored into the variable olddemand.

part — Character type; maximum of 10 alphanumeric digits. The company part number, for the part whose demand is to be

	changed is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; values range from -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, then part is not in the partid file.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.5.2.6.3 Description of Flow*

To change MPS data file records, the subroutine "MSchange" performs the following steps:

Step 1. Ask the MRP User for the 'part, date, demand quantity to be changed or quit, Ø, Ø to stop' and store the responses into variables part, olddate, olddemand, respectively.

Step 2. If part = 'quit', return to subroutine "MSdemands".

Step 3. Call subroutine "findsys" to obtain the system part number (sys) for part.

Step 4. If sys = -1, go to Step 1.

Step 5. Ask the MRP User for the 'correct date, correct demand quantity' and store the responses into variables newdate, newdemand, respectively.

* For flowchart, see Appendix C, page 571.

Step 6. Find the MPS demand record.

Step 7. Set olddate = newdate and olddemand = newdemand.

Step 8. Go to Step 1.

4.5.2.6.4 Example

To illustrate the use of the subroutine "MSchange", the MPS data file generated in Example 4.5.2.5.2 receives the following changes:

	OLD		NEW	
part	day	quantity	day	quantity
a	105	14	110	20
g	95	9	100	9

To make the changes, the steps are as follows:

Step 1. The statement - 'To enter MS demands, type MS. To enter external orders, type XO. To enter service parts, type SP. To stop, type quit', - gets the response of 'MS'.

Step 2. To 'new, add, change, or quit MS demands list?', the MRP User types 'add'.

Step 3. To 'part, date, demand quantity to be changed or quit, Ø, Ø to stop', the MRP User tyes 'a, 100, 14'.

Step 4. To 'correct date, correct demand quantity', the MRP User types '110, 20'.

Step 5. To 'part, date, demand quantity to be changed or quit, Ø, Ø to stop', the MRP User tyes 'g, 95, 9'.

Step 6. To 'correct date, correct demand quantity', the MRP User tyes '100, 9'.

Step 7. To 'part, date, demand quantity to be changed or quit, Ø, Ø to stop', the MRP User tyes 'quit, Ø, Ø'.

The results are shown in Figure 4.54.

4.5.2.7 Subroutine "XOdemands"

4.5.2.7.1 Purpose

When the MRP User states that the XO data file is to be edited, the subroutine "XOdemands" asks whether the purpose of the editing is to create a new file, add new file records, or change file records.

4.5.2.7.2 Variables

answer — Character type; answer ε {new, add, change, quit}. The response to the editing preference is stored into the variable answer.

4.5.2.7.3 Description of Flow*

The subroutine "XOdemands" asks the MRP User 'new, add, change, or quit XO demand list'. The response is stored into the variable answer. If answer = 'new', then the subroutine "XOnew" is called. If answer = 'add', then the subroutine "XOadd" is called. If answer = 'change', then the subroutine "XOchange" is called. If answer = 'quit', the program control returns to the main program.

---

* For flowchart, see Appendix C, page 573.

4.5.2.8 Subroutine "XOnew"

4.5.2.8.1 Purpose

When a new XO data file is to be created, the subroutine "XOnew" works interactively with the MRP User to accomplish this task.

4.5.2.8.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
D	Real type; format code f8.0. The demand on record XOkey is stored into the variable D.
Da	Integer type; values range from 1 to 9999. The date on record XOkey is stored into the variable Da.
date	Integer type; values range from 1 to 999. The due date for the XO demand is stored into the variable date.
demand	Real type; format code f8.0. The quantity of parts needed to satify the XO demand is stored into the variable demand.
k	Integer type; values range from 1 to 9999. The keyxo on record xokey is stored into the variable k.
keyxo	Integer type; values range from 1 to 9999. The variable keyxo is the key placed on record xokey to point to the next XO demand for part.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.

lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
n	Integer type. The number of demand entries for a given part is stored into the variable n.
next	Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of records in the partstatus data file.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	Character type; maximum of 10 alphanumeric digits. The company part number with the XO demands is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; values range from -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, then part is not in the partid data file.
ucost	See Section 4.3.2.2.2.
xokey	Integer type; values range from 1 to 9999. The variable xokey is the record number in the XO data file where the demand information is written.
xorderkey	See Section 4.3.2.2.2.

4.5.2.8.3 Description of Flow*

To create a new XO data file, the subroutine "XOnew" performs the following steps:

Step 1. Reinitialize the xorderkey in the partstatus data file to -1.

Step 2. Initialize both xokey and keyxo to Ø.

Step 3. Ask the MP User 'part or quit to stop' and store the response into the variable part.

Step 4. If part = 'quit', return to "XOdemands" subroutine.

Step 5. Call subroutine "findsys" to obtain the system part number (sys) for part.

Step 6. If sys = -1, go to Step 3.

Step 7. Read record number sys in the partstatus data file.

Step 8. Obtain and store the XO demands for part. List the record number for the first entry on the partstatus record sys. Denote the last entry for part with keyxo = -1.

Step 9. Go to Step 3.

4.5.2.8.4 Example

See Example 4.5.2.4.4.

4.5.2.9 Subroutine "XOadd"

4.5.2.9.1 Purpose

To add records to the XO data file, the subroutine, "XOadd" works interactively with the MRP User to accomplish this task.

---

* For flowchart, see Appendix C, page 574.

4.5.2.9.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
D	Real type, format code f8.0. The quantity of part needed is stored into variable D.
Da	Integer type; values range from 1 to 9999. Due date for part is stored into the variable Da.
date	See Section 4.5.2.8.2.
demand	See Section 4.5.2.8.2.
k	Integer type; values range from 1 to 9999. The record number for the read, used to find last demand listed in XO file, is stored into the variable k.
key	Integer type; values range from -1 to 999. Part's next XO demand appears on record number key. If key = -1, then there are no more XO demands for part.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
next	Integer type; values range from 1 to 9999. The variable next is equal to the next available record number.
numkids	See Section 4.3.2.2.2.

oh	See Section 4.3.2.2.2.
part	See Section 4.5.2.8.2
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	See Section 4.5.2.8.2.
ucost	See Section 4.3.2.2.2.
xorderkey	Integer type; values range from -1 to 9999. The record number in the XO data file were the first XO demand is stored into the variable xorderkey. If xorderkey = -1, no demand records for part are on file.

4.5.2.9.3 Description of Flow*

To add new records to the XO data file, the subroutine "XOadd" performs the following steps:

Step 1. Read the next available record number for the XO data file from record Ø.

Step 2. Ask the MRP User for the 'part, date, demand quantity or quit, Ø, Ø to stop' and store the responses into variables part, date, demand, respectively.

Step 3. If part = 'quit', return to subroutine "XOdemands".

---

* For flowchart, see Appendix C, page 577.

Step 4. Call upon subroutine "findsys" to obtain the system part number (sys) for part.

Step 5. If sys = -1, go to Step 2.

Step 6. Write the new addition on record next.

Step 7. Link the new demand for prt to the last demand listed in the XO data file.

Step 8. Go to Step 2.

4.5.2.9.4 Example

See Example 4.5.2.5.4.

4.5.2.10 Subroutine "XOchange"

4.5.2.10.1 Purpose

To change XO demand records in the XO data file, the subroutine "XOchange" works interactively with the MRP User to accomplish this task.

4.5.2.10.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
date	Integer type; values range from 1 to 9999. The XO demand date, read from record k, is stored into the variable date.
demand	Real type; format code f8.0. The XO demand quantity, read from record k, is stored into the variable demand.
k	Integer type; values range from 1 to 9999. The record number, read in search of the old XO record, is stored into

the variable k.

key — Integer type; values range from -1 to 9999. The XO demand key to the next demand for part is stored into the variable key. If key = -1, no more XO demands are in the file for part.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — See Section 4.3.2.2.2.

newdate — Integer type; values range from 1 to 9999. The new date for the record is stored into the variable newdate.

newdemand — Real type; format code f8.0. The new demand quantity for the XO record is stored into the variable newdemand.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

olddate — Integer type; values range from 1 to 9999. The date of the XO demand to be changed is stored into the variable olddate.

olddemand — Real type; format code f8.0. The demand quantity of the XO demand to be changed is stored into the variable olddemand.

part — Character type; maximum of 10 alphanumeric digits. The company part number, for the part whose demand is to be changed, is stored into the variable part.

pu — See Section 4.3.2.2.2.

pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
sys	Integer type; values range from -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, then part is not in the partid file.
ucost	See Section 4.3.2.2.2.
xorderkey	Integer type; values range from -1 to 9999. The variable xorderkey is the pointer to the next XO demand for part. If xorderkey = -1, no more demands are on file.

4.5.2.10.3 Description of Flow*

To change XO data file records, the subroutine "XOchange" performs the following steps:

Step 1. Ask the MRP User for the 'part, date, demand quantity to be changed or quit, Ø, Ø to stop' and store the responses into variables part, olddate, olddemand, respectively.

Step 2. If part = 'quit', return to subroutine "XOdemands".

Step 3. Call subroutine "findsys" to obtain the system part number (sys) for part.

Step 4. If sys = -1, go to Step 1.

Step 5. Ask the MRP User for the 'correct date, correct demand quantity' and store the responses into variables newdate, newdemand, respectively.

---

* For flowchart, see Appendix C, page 579.

Step 6. Find the XO demand record.

Step 7. Set olddate = newdate and olddemand = newdemand.

Step 8. Go to Step 1.

4.5.2.10.4 Example

See Example 4.5.2.6.4.

4.5.2.11 Subroutine "SPdemands"

4.5.2.11.1 Purpose

When the MRP User states that the SP data file is to be edited, the subroutine "SPdemands" asks whether the purpose of the editing is to create a new file, add new file records, or change file records.

4.5.2.11.2 Variables

answer — Character type; answer ε {new, add, change, quit}. The response to the editing preference is stored into the variable answer.

4.5.2.11.3 Description of Flow*

The subroutine "SPdemands" ask the MRP User 'new, add, change, or quit SP demand list'. The response is stored into the variable answer. If answer = 'new', then the subroutine "SPnew" is called. If answer = 'add', then the subroutine "SPadd" is called. If answer = 'change', then the subroutine "SPchange" is called. If answer = 'quit', the program control returns to the main program.

---

* For flowchart, see Appendix C, page 581.

4.5.2.12 Subroutine "SPnew"

4.5.2.12.1 Purpose

When a new SP data file is to be created, the subroutine "SPnew" works interactively with the MRP User to accomplish this task.

4.5.2.12.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
D	Real type; format code f8.0. The demand on record spkey is stored into the variable D.
Da	Integer type; values range from 1 to 9999. The date on record spkey is stored into the variable Da.
date	Integer type; values range from 1 to 9999. The due date fo the SP demand is stored into the variables date.
demand	Real type; format code f8.0. The quantity of parts needed to satisfy the SP demand is stored into the variable demand.
k	Integer type; values range from 1 to 9999. The keysp on record spkey is stored into the variable k.
keysp	Integer type; values range from 1 to 9999. The variable keysp is the key placed on record spkey to point to the next SP demand for part.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.

levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
n	Integer type. The number of demand entries for a given part is stored into the variable n.
next	Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of records in the partstatus data file.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	Character type; maximum of 10 alphanumeric digits. The company number with the MPS demands is stored into the variable part.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
spkey	Integer type; values range from 1 to 9999. The variable spkey is the record number in the SP data file where the demand information is written.
SR	See Section 4.3.2.2.2.
sys	Integer type; values range from -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, then part is not in the partid data file.
ucost	See Section 4.3.2.2.2.

xorderkey See Section 4.3.22.2.2.

4.5.2.12.3 Description of Flow*

To create a new SP data file, the subroutine "SPnew" performs the following steps:

Step 1. Reinitialize the servpartkey in the partstatus data file to -1.

Step 2. Initialize both spkey and keysp to Ø.

Step 3. Ask the MRP User 'part or quit to stop' and store the response into the variable part.

Step 4. If part = 'quit', return to "SPdemands" subroutine.

Step 5. Call subroutine "findsys" to obtain the system part number (sys) for part.

Step 6. If sys = -1, go to Step 3.

Step 7. Read record number sys in the partstatus data file.

Step 8. Obtain and store the SP demands for part. List the record number for the first entry on the partstatus record sys. Denote the last entry for part with keysp = -1.

Step 9. Go to Step 3.

4.5.2.12.4 Example

See Example 4.5.2.4.4.

---

* For flowchart, see Appendix C, page 582.

4.5.2.13 Subroutine "SPadd"

4.5.2.13.1 Purpose

To add records to the SP data file, the subroutine "SPadd" works interactively with the MRP User to accomplish this task.

4.5.2.13.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
D	Real type; format code f8.0. The quantity of part needed is stored into variable D.
Da	Integer type; values range from 1 to 9999. Due date for part is stored into the variable Da.
date	See Section 4.5.2.12.2.
demand	See Section 4.5.2.12.2.
k	Integer type; values range from 1 to 9999. The record number for the read, used to find last demand listed in SP file, is stored into the variable k.
key	Integer type; values range from -1 to 999. Part's next SP demand appears on record number key. If key = -1, then there are no more SP demands for part.
lead	See Section 4.3.2.2.2.
level	See Section 4.3.2.2.2.
levelkey	See Section 4.3.2.2.2.
lotp	See Section 4.3.2.2.2.

lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
next	Integer type; values range from 1 to 9999. The variable next is equal to the next available record number.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
part	See Section 4.5.2.12.2.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
scost	See Section 4.3.2.2.2.
servpartkey	Integer type; values range from -1 to 9999. The record number in the SP data file where the first SP demand is stored into the variable servpartkey. If servpartkey = -1, no demand records for part are on file.
SR	See Section 4.3.2.2.2.
sys	See Section 4.5.2.12.2.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

4.5.2.13.3 Description of Flow*

To add new records to the SP data file, the subroutine "SPadd" performs the following steps:

Step 1. Read the next available record number for the SP data file from record Ø.

---

* For flowchart, see Appendix C, page 585.

Step 2. Ask the MRP User for the 'part, date, demand quantity or quit, Ø, Ø to stop' and store the responses into variables part, date, demand, respectively.

Step 3. If part = 'quit', return to subroutine "SPdemands".

Step 4. Call upon subroutine "findsys" to obtain the system part number (sys) for part.

Step 5. If sys = -1, go to Step 2.

Step 6. Write the new addition on record next.

Step 7. Link the new demand for part to the last demand listed in the SP data file.

Step 8. Go to Step 2.

4.5.2.13.4 Example

See Example 4.5.2.5.4

4.5.2.14 Subroutine "SPchange"

4.5.2.14.1 Purpose

To change SP demand records in the SP data file, the subroutine "SPchange" works interactively with the MRP User to accomplish this task.

4.5.2.14.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
date	Integer type; values range from 1 to 9999. The SP demand date, read from record k, is stored into the variable date.

demand — Real type; format code f8.0. The SP demand quantity, read on record k, is stored into the variable demand.

k — Integer type; values range from 1 to 9999. The record number, read in search of the old SP record, is stored into the variable k.

key — Integer type; values range from -1 to 9999. The SP demand key to the next demand for part is stored into the variable key. If key = -1, no more SP demands are in the file for part.

lead — See Section 4.3.2.2.2.

level — See Section 4.3.2.2.2.

levelkey — See Section 4.3.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — See Section 4.3.2.2.2.

newdate — Integer type; values range from 1 to 9999. The new date for the SP record is stored into the variable newdate.

newdemand — Real type; format code f8.0. The new demand quantity for the SP record is stored into the variable newdemand.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

olddate — Integer type; values range from 1 to 9999. The date of the SP demand to be changed is stored into the variable olddate.

olddemand — Real type; format code f8.0. The demand quality of the SP demand to be changed is stored into the variable olddemand.

part — Character type; maximum of 10 alphanumeric digits. The

company part number, for the part whose demand is to be changed, is stored into the variable part.

pu — See Section 4.3.2.2.2.

pup — See Section 4.3.2.2.2.

scost — See Section 4.3.2.2.2.

servpartkey — Integer type; values range from -1 to 9999. The variable servpartkey is the pointer to the next SP demand for part. If servpartkey = -1, no more demands are on file.

SR — See Section 4.3.2.2.2.

sys — Integer type; values range from -1 to 999. The system part number for part is stored into the variable sys. If sys = -1, the part is not on the partid file.

ucost — See Section 4.3.2.2.2.

xorderkey — See Section 4.3.2.2.2.

4.5.2.14.3 Description of Flow*

To change SP data file records, the subroutine "SPchange" performs the following steps:

Step 1. Ask the MRP User for the 'part, date, demand quantity to be changed or quit, Ø, Ø to stop' and store the responses into variables part. olddate. olddemand, respectively.

Step 2. If part = 'quit', return to subroutine "SPdemands".

Step 3. Call subroutine "findsys" to obtain the system part number (sys) for part.

---

* For flowchart, see Appendix C, page 587.

Step 4. If sys = -1, go to Step 1.

Step 5. Ask the MRP User for the 'correct data, correct demand quantity' and store the responses into variables newdate, newdemand, respectively.

Step 6. Find the SP demand record.

Step 7. Set olddate = newdate and olddemand = newdemand.

Step 8. Go to Step 1.

4.5.2.14.4 Example

See Example 4.5.2.6.4.

5.0 Explosion and Output Programs

5.1 Introduction

The Explosion and Output Programs include: (1) Level, (2) MRP, and (3) SystemStatus. The Level program assigns levels to all the parts and determines the order flow for the MRP explosion. To assign levels, the parent parts at the top of the bills-of-materials tree receives a level assignment of 1. These parts have no dependent demands placed on them and are not in the child parts list in the bom data file. For more information on the algorithm, see Section 2.2. To establish the order flow for the MRP explosion, a pointer to the first part with level = 1 is stored onto record of the partstatus data file. Then, the levelkey for that part is assigned the record number of the next part in the list with level = +1. The levelkey, for the last part with level = 1, is set equal to the record number of the first part with level = 2. For more information on levelkey, see Section 3.3.4.

The MRP program performs the Material Requirements Planning calculations for each of the parts. The MRP matrix is printed out for each part as well as a list of Dependent Demands with their sources and a purchase order when requirements appear in the current order release bucket.

The SystemStatus program contains a variety of output programs, such as ones to print, parts lists and bills-of-materials list. The file is designed so that the MRP user can select the program needed interactively.

5.2 Level

5.2.1 Introduction to Level

The Level program assigns a level to each part and determines the order of flow for the MRP explosion. To assign the part levels, the partid data file and the bom data file are needed to run the Level program. For the basic algorithm used to assign part levels, see Section 2.2.

5.2.2 Subroutines in Level

5.2.2.1 Overview

The Subroutines in the Level program include:

1. LandSinitial;
2. Lassign;
3. Sremove;
4. Levelconnect.

(See Figure 5.1)

The flow of the program is directed by the Main Program. The steps are as follows:

Step 1. Call subroutine "LandSinitial" to initialize level and levelkey to -1, and to assign a flag (stack or s) to each record in the bom data file.

Step 2. Let the level number (L) be equal to 0.

Step 3. Increment the L by 1.

Step 4. Call subroutine "Lassign" to determine what parts are not in the child parts list and assign them to part level L.

Step 5. Call subroutine "Sremove" to determine which bom records to

remove from the stack, i.e., set s = 0.

Step 6. Check. If all parts have a level assigned, go to Step 7. If not, go to Step 3.

Step 7. Call subroutine "Levelconnect" to establish the order of flow for the MRP explosion.

The variables in the Main Program include:

BofM - key to first appearance as child part in bom file;

BOM - key to first appearance as parent part in bom file;

ccost - carrying cost;

ccper - time unit for ccost;

i - do loop index and record number in the partstatus file;

j - not all parts have a level assigned if j = 1 and all parts have a level assigned if j = 0;

L - level number to be assigned to parts which qualify;

lead - lead time;

level - part level;

levelkey - key for MRP flow;

lotp - time in days for lot-for-lot and fixed period quantity;

lotr - lotting rule;

MPSkey - key to MPS demands;

nextbom - next available bom record;

nextpart - next available partstatus record;

numkids - number of different components;

oh - on hand inventory;

pu - periodal usage;

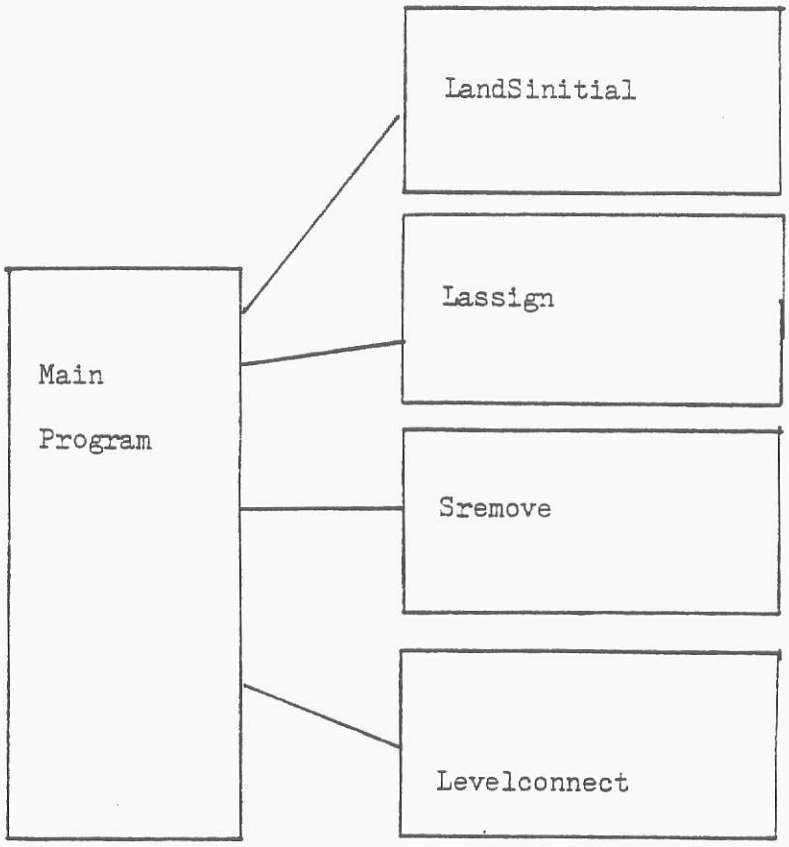


Figure 5.1
Level Subroutines

```
11  1    Partstatus Data File
   10.,   1,   3,   1,  -1,  20,1,   5,  100.00,  1.0000,  1.0000,2,   100.,4,  3, -1, -1,  1,  7
    0.,   2,  -1,  -1,   1,  20,2,  20,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  4,  4
    0.,   4,   2,   4,   2,  15,2,  10,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  3,  9
  320.,  -1,  -1,  -1,   4,  20,1,   5,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  4,  6
   65.,   6,   3,   6,   3,  10,2,  10,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  2,  8
  100.,   7,  -1,  -1,   6,  20,2,  35,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  4, -1
   10.,  -1,   3,   8,  -1,  10,2,  10,  100.00,  1.0000,  1.0000,2,   100.,4,  4, -1, -1,  1, 10
   10.,   8,   1,  12,  11,  35,2,  10,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  2,  3
   75.,  -1,   2,  13,  12,  15,2,  20,  100.00,  1.0000,  1.0000,2,   100.,4, -1, -1, -1,  3,  2
    3.,   9,   2,  15,  -1,   5,2,  10,  100.00,  1.0000,  1.0000,2,   100.,4,  6, -1, -1,  1,  5
```

level

levelkey

Figure 5.2
The partstatus With level and
levelkey Assigned

pup - time unit for periodal usage;

scost - set cost;

servpartkey - key to Service Part demands;

SR - key to scheduled receipt demands;

ucost - unit cost;

xorderkey - key to external order demands.

The program is run for the sample problem and the results are in Figure 5.2.

5.2.2.2 Subroutine "LandSinitial"

5.2.2.2.1 Purpose

The subroutine "LandSinitial" has the responsibilty to set both level and levelkey in the partstatus data file to -1 and to assign a flag of stack = 1 to each of the bom record. The assumption is that the bom record has not been removed from the stack if stack = 1.

5.2.2.2.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
c	Integer type; values range from 1 to 999. The child part on the bom record is stored into the variable c.
ckey	Integer type; values range from 1 to 9999. The key to the next parent part for c is stored into the variable ckey.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
lead	See Section 4.3.2.2.2.

level — Integer type; values range from 1 to 9999. The part level is stored into the variable level.

levelkey — Integer type; values range from -1 to 999. The record number to the next part to be exploded is stored into the variable levelkey. If levelkey = -1, the part is the last one to be exploded.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSKey — See Section 4.3.2.2.2.

nextbom — Integer type; values range from 1 to 9999. The quantity of nextbom -1 is equal to the number of records in the bom file.

nextpart — Integer type; values range from 1 to 999. The quantity of nextpart is equal to the number of parts in the partstatus data file.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

op — Integer type; values range from 1 to 999. The parent part on the bom record is stored into the variable p.

pkey — Integer type; values range from 1 to 9999. The key to the next child part for p is stored into the variable pkey.

pu — See Section 4.3.2.2.2.

pup — See Section 4.3.2.2.2.

q — Integer type; values range from 1 to 9999. The quantity of c per p is stored into the variable q.

scost — See Section 4.3.2.2.2.

servpartkey See Section 4.3.2.2.2.

SR See Section 4.3.2.2.2.

ucost See Section 4.3.2.2.2.

xorderkey See Section 4.3.2.2.2.

5.2.2.2.3 Description of Flow*

To set level = -1, levelkey = -1, and stack = 1, the subroutine "LandSinitial" performs the following steps:

Step 1. Read record 0 in the partstatus data file and store the value into variable nextpart. Note: the number of records in the file is equal to nextpart - 1.

Step 2. Read each record in the partstatus data file; set level = -1 and levelKey = -1; and rewrite to the file.

Step 3. Read record 0 in the bom data file and store the value into variable next bom. Note: the number of records in the file is equal to next bom - 1.

Step 4. Read each record in the bom data file; set stack = 1; and rewrite to the file.

5.2.2.3 Subroutine "Lassign"

5.2.2.3.1 Purpose

The subroutine "Lassign" assigns level (L) to the parts that qualify for that level. To qualify, the part must not already have a level number and not be among the list of child parts that are flagged to be still in the bom record list.

---

* For flowchart, see Appendix C, page 590.

5.2.2.3.2 **Variables**

assign — Real type; assign ϵ {Ø, 1}.
If assign = 0, the part will not be assigned to level L.
If assign = 1, the part will be assigned to level L.

BofM — See Section 4.3.2.2.2

BOM — See Section 4.3.2.2.2

c — See Section 5.2.2.2.2

ckey — See Section 5.2.2.2.2

ccost — See Section 4.3.2.2.2

ccper — See Section 4.3.2.2.2

i — Integer type; values range from 1 to 999. The do loop index and record number in the partstatus file is stored into the variable i. The system part number is equal to i.

j — Integer type; values range from 1 to 9999. The do loop index and record number in bom file is stored into the variable j.

L — Integer type. The level number, to assign to the part that qualifies, is stored into the variable L.

lead — See Section 4.3.2.2.2.

level — Integer type; values range from 1 to 999. The part level is stored into the variable level.

levelkey — Integer type; values range from 1 to 999. The record number to the next part to be exploded is stored into the variable levelkey. If levelkey = 1, the part is the last one to be exploded.

lotp	See Section 4.3.2.2.2.
lotr	See Section 4.3.2.2.2.
MPSkey	See Section 4.3.2.2.2.
nextbom	See Section 5.2.2.2.2.
nextpart	See Section 5.2.2.2.2.
numkids	See Section 4.3.2.2.2.
oh	See Section 4.3.2.2.2.
p	See Section 5.2.2.2.2.
pkey	See Section 5.2.2.2.2.
pu	See Section 4.3.2.2.2.
pup	See Section 4.3.2.2.2.
q	See Section 5.2.2.2.2.
s	Integer type; s ϵ {Ø}. If s = 1, then the bom record is in the stack. If s = 0, then the bom record is not in the stack.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.
xorderkey	See Section 4.3.2.2.2.

5.2.2.3.3 Description of Flow*

To determine which parts qualify for level (L), the subroutine "Lassign" performs the following steps:

Step 1. Let i = 0

*For flow chart, see Appendix C, page 592.

Step 2. Set i = i + 1 and read record i in the partstatus file.

Step 3.. If level ≠ 1, go to Step 2.

Step 4. Set assign = 1.

Step 5. Read records in the bom data file. If the part is in the child's list and s = 1, then set assign = Ø,

Step 6. If assign = 1, set level = L and record on partstatus record i.

Step 7. If i ≥ nextpart - 1, stop; otherwise, go to Step 2.

5.2.2.4 Subroutine "Sremove"

5.2.2.4.1 Purpose

The subroutine "Sremove" flags the bom records which qualify to be removed from the stack. To qualify for removal (i.e., s = 0) flag, the part is assigned to the most recent level (L) and is listed in the parent parts list in the bom data file.

5.2.2.4.2 Variables

BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
c	Integer type; values range from 1 to 999. The child part on the bom record is stored into the variable c.
ckey	Integer type; values range from 1 to 9999. The key to the next parent part for c is stored into the variable ckey.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
i	Integer type; values range from 1 to 999. The do loop index and record number in the partstatus file is stored

into the variable i. The system part number is equal to i.

j — Integer type; values range from 1 to 9999. The do loop index and record number in bom file is stored into the variable j.

L — Integer type. The level number, to assign to the part that qualified, is stored into the variable L.

lead — See Section 4.3.2.2.2.

level — See Section 5.2.2.2.2.

levelkey — See Section 5.2.2.2.2.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — See Section 4.3.2.2.2.

nextbom — See Section 5.2.2.2.2.

nextpart — See Section 5.2.2.2.2.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

p — Integer type; values range from 1 to 999. The parent part on the bom record is stored into the variable p.

pkey — Integer type; values range from 1 to 9999. The key to the next child part for p is stored into the variable pkey.

pu — See Section 4.3.2.2.2.

pup — See Section 4.3.2.2.2.

q — Integer type; values range from 1 to 9999. The quantity of c per p is stored into the variable q.

s — Integer type; s {0, 1}. If s = 1, then the bom record is in the stack. If s = 0, then the bom record is not in

	the stack.
scost	See Section 4.3.2.2.2.
servpartkey	See Section 4.3.2.2.2.
SR	See Section 4.3.2.2.2.
ucost	See Section 4.3.2.2.2.
Xorderkey	See Section 4.3.2.2.2.

5.2.2.4.3 Description of Flow*

To flag the bom record as being removed from the stack, the subroutine "Sremove" performs the following steps:

Step 1. Set i = 0.

Step 2. Set i = i + 1 and read record i in the partstatus file.

Step 3. If level ≠ L, go to Step 2.

Step 4. Read each record in the bom file. If the part is in the parent list, then set s = 0.

Step 5. If i ≥ nextpart 1, stop; otherwise, go to Step 2.

5.2.2.5 Subroutine "Levelconnect"

5.2.2.5.1 Purpose

The subroutine "Levelconnect" establishes the order of flow for the MRP explosion. The parts with level = 1 are linked first; then, the last part with level = 2; then, the parts with level = 2 are linked; and so on. The flow is set up with the pointer start on record Ø and the value of levelkey in the partstatus file. (For more details, see Section 3.3.4.)

---

* For flowchart, see Appendix C, page 594.

5.2.2.5.2 Variables

BofM	See Section 4.3.2.2.2.
BofM1	The BofM on the last record, assigned to a levelkey, is stored into the variable BofM1.
BOM	See Section 4.3.2.2.2.
BOM1	The BOM on the last record, assigned to a levelkey, is stored into the variable BOM1.
ccost	See Section 4.3.2.2.2.
ccost1	The ccost on the last record, assigned to a levelkey, is stored into the variable ccost1.
ccper	See Section 4.3.2.2.2.
ccper1	The ccper on the last record, assigned to a levelkey, is stored into the variable ccper1.
i	See Section 5.2.2.3.2.
i1	The i for the last record, assigned to a levelkey, is stored into the variable i1.
L	See Section 5.2.2.3.2.
lead	See Section 4.3.2.2.2.
lead1	The lead on the last record, assigned to a levelkey, is stored into the variable lead1.
level	See Section 5.2.2.2.2.
level1	The level on the last record, assigned to a levelkey, is stored into variable level1.
levelkey	See Section 5.2.2.2.2.
Lmax	Integer type. The maximum part level assigned is stored into the variable Lmax.

lotp	See Section 4.3.2.2.2
lotpl	The lotp on the last record, assigned to a levelkey, is stored into the variable lotpl.
lotr	See Section 4.3.2.2.2.
lotrl	The lotr on the last record, assigned to a levelkey, is stored into the variable lotrl.
MPSkey	See Section 4.3.2.2.2.
MPSkeyl	The MPSkey on the last record, assignd to a levelkey is stored into the variable MPSkeyl.
n	Integer type. The number of passes through the loop are stored into the variable n.
nextpart	See Section 5.2.2.2.2.
numkids	See Section 4.3.2.2.2.
numkids	The numkids on the last record, assigned to a levelkey, is stored into the variable numkidsl.
oh	See Section 4.3.2.2.2.
ohl	The oh on the last record, assigned to a levelkey, is stored into the variable ohl.
pu	See Section 4.3.2.2.2.
pul	The pu on the last record, assigned to a levelkey, is stored into the variable pul.
pup	See Section 4.3.2.2.2.
pupl	The pup on the last record, assigned to a levelkey, is stored into the variable pupl.
scost	See Section 4.3.2.2.2.

scostl	The scost on the last record, assigned to a levelkey, is stored into the variable scostl.
servpartkey	See Section 4.3.2.2.2.
servpartkeyl	The servpartkey on the last record, assigned to a levelkey, is stored into the variable servpartkeyl.
SR	See Section 4.3.2.2.2.
SRl	The SR on the last record, assigned to a levelkey, is stored into the variable SRl.
ucost	See Section 4.3.2.2.2.
ucostl	The ucost on the last record, assigned to a levelkey, is stored into the variable ucostl
xorderkey	See Section 4.3.2.2.2.
xorderkey	The xorderkey on the last record, assigned to a levelkey, is stored into the variable xorderkey.

5.2.2.5.3 Description of Flow*

To establish the order flow for the MRP explosion, the "Levelconnect" subroutine performs the following steps:

Step 1. Let Lmax = L

Step 2. Set L ≠ Ø, n = Ø, and i = Ø.

Step 3. Increment level number; L = L + 1.

Sstep 4. Set i = i + 1 and read record i the partstatus file.

Step 5. If level ≠ L, go to Step 4.

Step 6. Increment number of times through loop; n = n + 1.

---

* For flowchart, see Appendix C, page 595.

Step 7. If L = 1 and n = 1, list record number on record Ø of the partstatus file.

Step 8. If L > 1 or n > L ., set levelkey, on the last partstatus record linked, equal to record number i.

Step 9. If i ≥ nextpart -1, go to Step 3; otherwise, go to Step 4.

5.3 MRP

5.3.1 Introduction

After all the data files have been set up and the Level file run, the MRP calculations are ready to be done. The MRP explosion file is designed to calculate the MRP matrix for each part, print out the dependent demands and their source; print out the MRP matrix; and create a purchase order when necessary.

The MRP explosion is directed by the main program in the MRP file. The steps to perform the explosion are as follows:

Step 1. Initialize the DD (dependent demand) file.

Step 2. Send "Material Requirements Planning" heading to printer.

Step 3. Ask the MRP User for the company name to be placed on the purchased orders.

Step 4. Call subroutine "DDkey" to initialize the key to dependent demand records stored in the DD datafile. The key, itself, is stored on the part's record in the partid file. In other words, the demands created by part x are stored into the DD data file and a DDkey is listed on the partid record for part x.

Step 5. Read record Ø in the partstatus data file to obtain the part to be exploded first (start).

Step 6. Set parttoexplode = start.

Step 7. Read record number parttoexplode in the partstatus data file. (See Section 3.3.4 for the details on the information stored on the partstatus record.)

Step 8. Read record number parttoexplode on the partid data file to obtain the company part number.

Step 9. List company part number, system part number, lead time, and on hand inventory quantity on the printer output.

Step 10. Call the subroutine "setupHorizon" to set up the MRP matrix stored in the horizon file.

Step 11. If MPSkey ≠ -1, call subroutine "MPStransfer". The MPSkey is the key to the Master Production Schedule (MPS) demands. If MPSkey ≠ -1, demands exist for parttoexplode and need to be transferred to the MRP matrix.

Step 12. If xorderkey ≠ -1, call subroutine "XOtransfer". The variable xorderkey is the key to the External Order (XO) demands.

Step 13. If servpartkey ≠ -1, call subroutine "SPtransfer". The variable servpartkey is the key to the Service Part (SP) demands.

Step 14. If BofM = -1, the parttoexplode has dependent demands (DD) placed on it. Therefore, call subroutine "DependentDemands".

Step 15. Call subroutine "GrosssRequirements" to calculate the gross requirements for each period. Gross requirements is equal to the sum of MPS demands plus XO demands plus SP demands plus the DD.

Step 16. If SR ≠ -1, then the parttoexplode has scheduled receipts planned. To transfer them to the MRP matrix, call subroutine "SRtransfer".

Step 17. Call subroutine "OHandNR" to determine the on hand inventory and the net requirements for the MRP matrix.

Step 18. Call subroutine "LotSize" to determine the lot sizes for parttoexplode. The subroutine "L4L" (Lot-for-lot) is called if lotr = 1 or the subroutine "FPQ" (Fixed Period Quantity) is called if lotr = 2.

Step 19. Call subroutine "OrderRelease" to determine time for order placement by adjusting the lot size with the lead time.

Step 20. Call subroutine "OutPut" to direct the output of the MRP matrix sent to the printer. The "OutPut" subroutine has the sub-routines "SIX", "FIVE", "FOUR", "THREE", "TWO", and "ONE" to help with the data arrangement.

Step 21. If levelkey = -1, stop. Otherwise, let parttoexplode = levelkey and return to Step 7.

The variables in the MRP main program are defined as follows:

BofM - first appearance in bom child's list;

BOM - first appearance in bom parent's list;

buckets - the number of periods in the planning horizon;

c - the company part number;

ccost - carrying cost ($/time period);

ccper - time period for ccost;

ddkey - the next available record number in the DD data file;

lead - lead time (in days);

level - part level number;

levelkey - record number for next part to explode and last part to

explode is marked with levelkey = -1;

lotp - number of days for L4L or FPQ;

lotr - lotting rule;

MPSkey - key to MPS demands;

name - the company name;

nextpart - next available record number in the partstatus data file;

numkids - number of different components;

oh - on hand inventory;

parttoexplode - the system (record) number of the part to be exploded;

pu - periodal usage (units/time period);

pup - time period for pu;

scost - set up cost;

servpartkey - key to SP demands;

SR - key to scheduled receipts;

start - record number of first part to explode;

ucost - unit cost

xorderkey - key to XO demands.

5.3.2 Subroutines in MRP

5.3.2.1 Overview

The MRP explosion file consists of the following subroutines:

1. DDkey
2. setupHorizon
3. MPStransfer
4. XOtransfer

5. SPtransfer
6. GrossRequirements
7. SRtransfer
8. OHandNR
9. LotSize
10. L4L
11. FPQ
12. OrderRelease
13. OutPut
14. SIX
15. FIVE
16. FOUR
17. THREE
18. TWO
19. ONE
20. OrderWrite
21. DependentDemands

(See Figure 5.3)

The action in the MRP file is directed by the main program (See Section 5.3.1). In brief, the duties of the subroutines are as follows: The subroutine "DDkey" initializes the variable DDkey in the partid data file to -1. The subroutine "setupHorizon" initializes the values in the MRP matrix to Ø.

The subroutines "MPStransfer", "stransfer", "SPtransfer" and "DependentDemands" add to the MRP matrix the Master Production Schedule (MPS) demands, the External Order (XO) demands, the Service Part (SP)

demands, and the dependent demands (DD), respectively. The subroutine "GrossRequirements" determines the gross requirements for each period in the MRP matrix.

The subroutine "SRtransfer" puts the scheduled receipts into the MRP matrix. Then, the "OHandNR" subroutine projects the on hand inventory and the next requirements into the future. After the projection is done, the subroutine "LotSize" with the help of "L4L" and "FPQ" determine the lot size requirements. Finally, the "OrderRelease" subroutine adjusts the lot size requirement with lead time to determine when to place the order. The dependent demands produced by a part are stored into the DD data file by the subroutine "OrderRelease".

The output sent to the printer is directed by the subroutine "Out-Put". The MRP matrix stored into the horizon data file is the transposed matrix of the output matrix. Therefore, the subroutines "SIX", "FIVE", "FOUR", "THREE", "TWO", and "ONE" rearrange the data for the printer. If a purchase order is required, the "OrderWrite" subroutine produces the order.

5.3.2.2 Subroutine "DDkey"

5.3.2.2.1 Purpose

The subroutine "DDKey" initializes the DDKey's in the partid data file to -1. The DDkey on record i is the key to the dependent demands produced by part i.

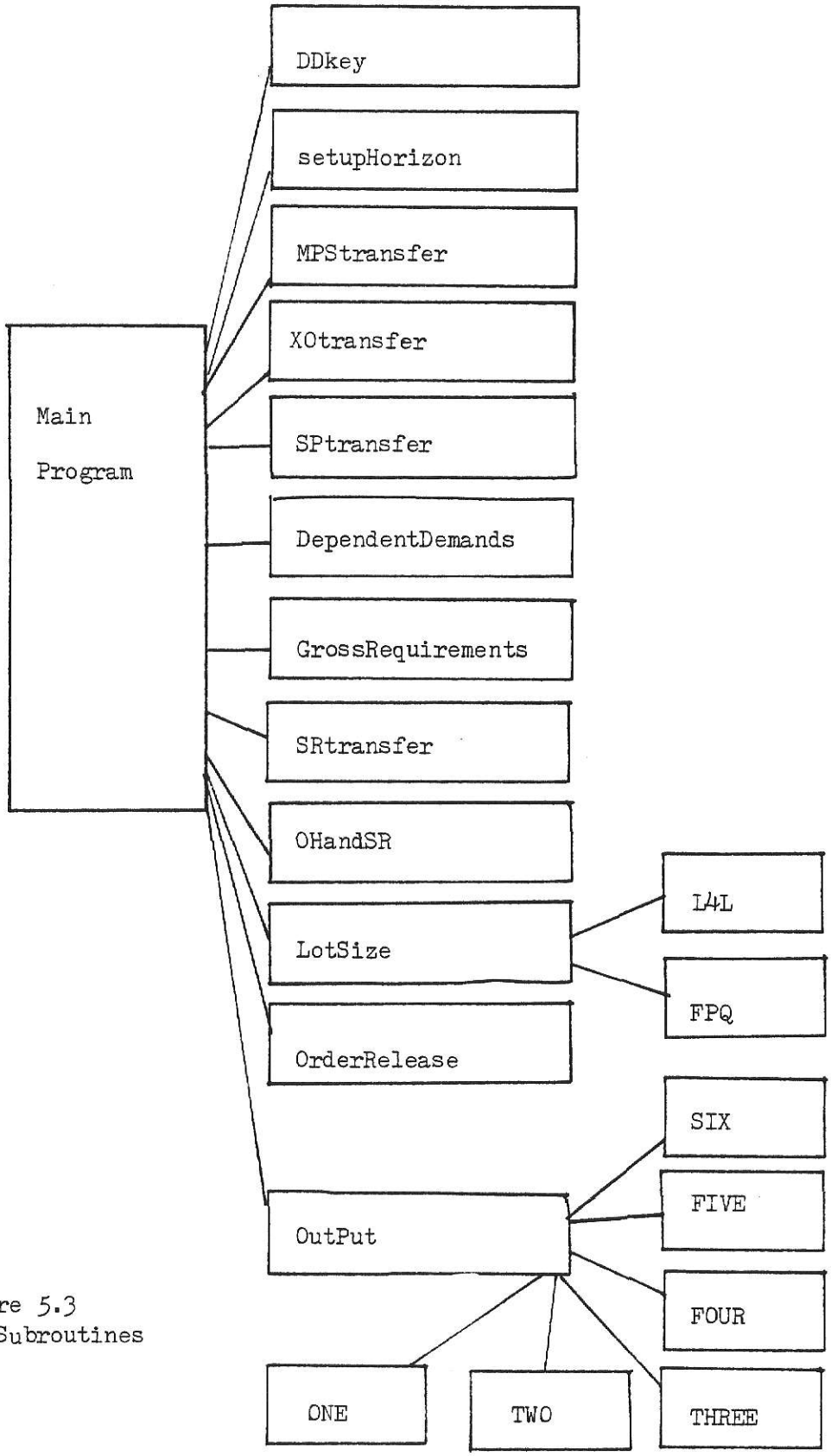


Figure 5.3
MRP Subroutines

5.3.2.2.2 Variables

c	Character type; maximum of 10 alphanumeric digits. The company part number on record i is stored into c.
d	Integer type; values range from 1 to 999. The down branch for the binary search is stored into d.
first	Integer type; values range from 1 to 999. The value of first is stored on record Ø of the partid data file and is the record number for c with ordinal number one.
i	Integer type; values range from 1 to 999. The do loop index and the record number to read is stored into the variable i.
k	Integer type; values range from 1 to 999. The directional key is stored into k.
next	Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of records in the partid data file.
p	Integer type; values range from 1 to 999. The ordinal number for c is stored into p.
sys	Integer type; values range from 1 to 999. The system part number on record i is stored into sys.
u	Integer type; values range from 1 to 999. The up branch for the binary search is stored into u.

5.3.2.2.3 Description of Flow*

To initialize DDKey in the partid data file, the subroutine "DDkey" uses a do loop to read and rewrite each record one-by-one setting DDKey to -1.

5.3.2.3 Subroutine "setupHorizon"

5.3.2.3.1 Purpose

The subroutine "setupHorizon" initializes to zero the values in the MRP matrix stored in the horizon data file. The values in the matrix include:

1. the Master Production Schedule demands (dMPS);
2. the External Order demands (dXO);
3. the Service Part demands (dSP);
4. the sum of dependent demands (sumDD);
5. the gross requirements (GR);
6. scheduled receipts (SR);
7. on hand inventory (OH);
8. net requirements (NR);
9. lot size requirements (LR);
10. order requirements (OR).

5.3.2.3.2 Variables

buckets — Integer type; values range from 1 to 999. The number of periods or records in the horizon is equal to buckets.

---

* For flowchart, see Appendix C, page 601.

date	Integer type; values range from 1 to 9999. The first day of the period is stored into date.
i	Integer type; values range from 1 to 999. The do loop index and the record number to read is stored into variable i.
period	Integer type; values range from 1 to 999. The period number is stored into the variable period.

5.3.2.3.3 Description of Flow*

The values in the MRP matrix are all set to Ø by a do loop in the "setupHorizon" subroutine. For each record, the do loop reads the period and date from the record; and then rewrites the period and date to the record adding all other values of the MRP matrix set to zero.

5.3.2.4 Subroutine "MPStransfer"

5.3.2.4.1 Purpose

The subroutine "MPStransfer" handles the transfer of the Master Production Schedule demands from the MPS data file to the MRP matrix in the horizon file.

5.3.2.4.2 Variables

buckets	See Section 5.3.2.3.2.
date	Integer type; values range from 1 to 9999. The first day of period is stored into date.
datel	Integer type; values range from 1 to 9999. The date on recordi is stored into the variable datel.

*For flowchart, see Appendix C, page 602.

day	Integer type; values range from 1 to 9999. The need date of the MPSdemand is stored into the variable day.
demand	Real type; format code f8.0. The demand quantity is stored into the variable demand.
dMPS	Real type; format code f8.0. The MPS demand quantity is stored into dMPS.
dSP	Real type; format code f8.0. The SP demand quantity is stored into dSP.
dXO	Real type; format code f8.0. The XO demand quantity is stored into dXO.
GR	Real type; format code f8.0. The gross requirements is stored into the variable GR.
i	Integer type; values range from 1 to 999. The do loop index and the record number read in the horizon file is stored into i.
j	Integer type; values range from 1 to 999. The next record in the horizon data file is j = i + 1.
k	Integer type; values range from 1 to 999. The record number k in the MPS data file has the MPS demand information.
key	Integer type; values range from -1 to 999. The key is the record number for the next MPS demand for the part. If key = -1, there are no more demands.
LR	Real type; format code f8.0. The lot size requirement is stored into the variable LR.
MPSkey	Integer type; values range from 1 to 999. The MPSkey is

the key to the Master Production Schedule demands in the MPS data file.

NR — Real type; format code f8.0. The next requirements quantity is stored into the varible NR.

OH — Real type; format code f8.0. The on hand inventory quantity is stored into the variable OH.

OR — Real type; format code f8.0. The order release quantity is stored into the variable OR.

period — Integer type; values range from 1 to 999. The period number is stored into period.

periodl — Interger type; values range from 1 to 999. The period on record j is stored into the variable periodl.

SR — Real type; format code f8.0. The scheduled receipt quantity is stored into the variable SR.

sumDD — Real type; format code f8.0. The sum total of the dependent demands is stored into the variable sumDD.

5.3.2.4.3 Description of Flow*

To transfer the Master Production Schedule demands to the horizon data file, the subroutine "MPStransfer" performs the following steps:

Step 1. Let k = MPSkey.

Step 2. Read record k in the MPS data file and store the information into the variables day, demand and key.

Step 3. Locate the period where the MPS demand belongs; and add it to the other MPS demands for that period.

*For flowchart, see Appendix C, page 603.

Step 4. If key = -1, return to main program. Otherwise, set k = key and go to Step 2.

5.3.2.5 Subroutine "XOtransfer"

5.3.2.5.1 Purpose

The subroutine "XOtransfer" handles the transfer of the External Order demands from the XO data file to the MRP matrix in the horizon file.

5.3.2.5.2 Variables

buckets	See Section 5.3.2.3.2
date	See Section 5.3.2.4.2.
datel	See Section 5.3.2.4.2.
day	Integer type; values range from 1 to 9999. The need date of the XO demand is stored into the variable day.
demand	See Section 5.3.2.4.2.
dMPS	See Section 5.3.2.4.2.
dSP	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.
i	See Section 5.3.2.4.2.
j	See Section 5.3.2.4.2.
k	Integer type; values range from 1 to 999. The record number k in the XO data file has the XO demand information.
key	Integer type; values range from -1 to 999. The key is the record number for the next XO demand for the part. If key = -1, there are no more demands.
LR	See Section 5.3.2.4.2.

NR	See Section 5.3.2.4.2.
OH	See Section 5.3.2.4.2.
OR	See Section 5.3.2.4.2.
period	See Section 5.3.2.4.2.
SR	See Section 5.3.2.4.2.
sumDD	See Section 5.3.2.4.2.
xorderkey	Integer type; values range from 1 to 999. The xorderkey is the key to the External Order demands in the XO data file.

5.3.2.5.3 Description of Flow*

To transfer the External Order demands to the horizon data file, the subroutine "XOtransfer" performs the following steps:

Step 1. Let k = xorderkey.

Step 2. Read record k in the XO data file and store the information into the variable day, demand and key.

Step 3. Locate the period where the XO demand belongs; and add it to the other XO demands for that period.

Step 4. If key = -1, return to the main program. Otherwise, set k = key and go to Step 2.

5.3.2.6 Subroutine "SPtransfer"

5.3.2.6.1 Purpose

The subroutine "SPtransfer" handles the transfer of the Service Part demands from the SP data file to the MRP matrix in the horizon file.

---

*For flowchart, see Appendix C, page 605.

5.3.2.6.2 Variables

buckets	See Section 5.3.2.3.2.
date	See Section 5.3.2.4.2.
date1	See Section 5.3.2.4.2.
day	Integer type; values range from 1 to 9999. The need date of the SP demand is stored into the variable day.
demand	See Section 5.3.2.4.2.
dMPS	See Section 5.3.2.4.2.
dSP	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.
i	See Section 5.3.2.4.2.
j	See Section 5.3.2.4.2.
k	Integer type; values range from 1 to 999. The record number k in the SP data file has the SP demand information.
key	Integer type; values range from -1 to 999. The key is the record number for the next SP demand for the part. If key = -1, there are no more demands.
LR	See Section 5.3.2.4.2.
NR	See Section 5.3.2.4.2.
OH	See Section 5.3.2.4.2.
OR	See Section 5.3.2.4.2.
period	See Section 5.3.2.4.2.
servpartkey	Integer type; values range from 1 to 999. The servpartkey is the key to the Service Part demands in the SP data file.
SR	See Section 5.3.2.4.2.

sumDD See Section 5.3.2.4.2.

5.3.2.6.3 Description of Flow*

To transfer the Service Part demands to the horizon data file, the subroutine "SPtransfer" performs the following steps:

Step 1. Let k = servpartkey.

Step 2. Read record k in the SP data file and store the informatio into the variables day, demand and key.

Step 3. Locate the period where the SP demand belongs; and add it to the other SP demands for that period.

Step 4. If key = -1, return to the main program. Otherwise, set k = key and go to Step 2.

5.3.2.7 Subroutine "GrossRequirements"

5.3.2.7.1 Purpose

The gross requirements for each period in the planning horizon is calculated by the subroutine "GrossRequirements".

5.3.2.7.2 Variables

buckets	See Section 5.3.2.3.2
date	See Section 5.3.2.4.2.
dMPS	See Section 5.3.2.4.2.
dSP	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.

*For flowchart, see Appendix C, page 607.

i	Integer type; values range from 1 to 999. The do loop index and the record number read is stored into the variable i.
LR	See Section 5.3.2.4.2.
NR	See Section 5.3.2.4.2.
OH	See Section 5.3.2.4.2.
OR	See Section 5.3.2.4.2.
period	See Section 5.3.2.4.2.
SR	See Section 5.3.2.4.2.
sumDD	See Section 5.3.2.4.2.

5.3.2.7.3 Description of Flow*

To calculate the gross requirements for each period, the subroutine "GrossRequirements" uses a do loop to read the horizon record; adds the values dMPS, dXO, dSP and sumDD; stores the result into GR; and rewrites the values to the same horizon record.

5.3.2.8 Subroutine "SRtransfer"

5.3.2.8.1 Purpose

The subroutine "SRtransfer" handles the transfer of the scheduled receipt quantities from the SR data file to the MRP matrix in the horizon file.

5.3.2.8.2 Variables

buckets	See Section 5.3.2.3.2.
date	See Section 5.3.2.4.2.

*For flowchart, see Appendix C, page 609.

datel	See Section 5.3.2.4.2.
day	Integer type; values range from 1 to 9999. The need date of the SR demands is stored into the variable day.
demand	See Section 5.3.2.4.2.
dMPS	See Section 5.3.2.4.2.
dSP	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.
i	See Section 5.3.2.4.2.
j	See Section 5.3.2.4.2.
k	Integer type; values range from 1 to 999. The record number k in the SR data file has the SR demand information.
key	Integer type; values range from -1 to 999. The key is the record number for the next SR demand for the part. If key = -1, there are no more demands.
LR	See Section 5.3.2.4.2.
NR	See Section 5.3.2.4.2.
OH	See Section 5.3.2.4.2.
OR	See Section 5.3.2.4.2.
period	See Section 5.3.2.4.2.
SR	See Section 5.3.2.4.2.
SRkey	Integer type; values range from 1 to 9999. The SRkey is the key to the scheduled receipts records in the SR data file.
sumDD	See Section 5.3.2.4.2.

5.3.2.8.3 Description of Flow*

To transfer the scheduled receipts to the horizon data file, the subroutine "SRtransfer" performs the following steps:

Step 1. Let k = SRkey.

Step 2. Read record k in the SR data file and store the information into the variable day, demand, and key.

Step 3. Locate the period where the SR belongs; and add it to the other SR demands for that period.

Step 4. If key = -1, return to the main program. Otherwise, set k = key and go to Step 2.

5.3.2.9 Subroutine

5.3.2.9.1 Purpose

The "OHandNR" subroutine projects the on hand inventory quantities and the net requirements into the future periods.

5.3.2.9.2 Variables

buckets	See Section 5.3.2.3.2
date	See Section 5.3.2.4.2.
dMPS	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.
i	Integer type; values range from 1 to 999. The do loop index and the record number read is stored into the variable i.
LR	See Section 5.3.2.4.2

*For flowchart, see Appendix C, page 610.

net — Real type; format code f8.0. The quantity of either OH or NR is calculated from GR - (SR + net) is stored into the varible net. If net < Ø, OH = -net; and if net > Ø, NR = net.

NR — See Section 5.3.2.4.2.

oh — Real type; format code f8.0. The quantity of on hand inventory in the stockroom currently is stored into the variable oh.

OH — See Section 5.3.2.4.2.

OR — See Section 5.3.2.4.2.

period — See Section 5.3.2.4.2.

SR — See Section 5.3.2.4.2.

sumDD — See Section 5.3.2.4.2.

5.3.2.9.3 Description of Flow*

To project the on hand inventory quantities and the net requirement into the future, the subroutine "OHandNR" performs the following steps:

Step 1. Set net = oh.

Step 2. Let i = Ø.

Step 3. Increment i by 1.

Step 4. Read record i in horizon data file.

Step 5. Calculate GR - (SR + net) and store the result into net.

Step 6. If net < Ø, set OH = -net and net = -net. Go to Step 8.

Step 7. If net > Ø, set NR = net and net = Ø.

Step 8. Write the new values on record i in the horizon file.

*For flowchart, see Appendix C, page 612.

Step 9. If i ≤ buckets, go to Step 3. Otherwise, return to the main program.

5.3.2.10 Subroutine "LotSize"

5.3.2.10.1 Purpose

The subroutine "LotSize" call the proper lotting technique subroutine to determine the lot size.

5.3.2.10.2 Variables

buckets	See Section 5.3.2.3.2
ccost	Real type; format code f8.4. The carrying cost ($/time unit) is stored into ccost.
ccper	Integer type; values range from 0 to 9. The time unit for ccost is stored into the variable ccper.
lotp	Integer type; values range from 1 to 9999. The number of days per lotting period is stored into lotp.
lotr	Integer type; values range from 0 to 9. The lotting rule code is stored into the variable lotr.
pu	Real type; format code f8.0. Ther periodal usage (units/time period) is stored into the variable pu.
pup	Integer type; values range from 0 to 9. The time period for pu is stored into the variable pup.
scost	Real type; format code f8.2. The setup cost is stored into the variable scost.
ucost	Real type; format code f8.4. The unit cost is stored into the variable ucost.

5.3.2.10.3 Description of Flow*

The subroutine "LotSize" knows the proper lotting rule to call depending on the value of lotr. If lotr = 1, the lot-for-lot lotting technique is requested and the subroutine "L4L" is called. If lotr = 2, the fixed period quantity technnique is requested and the subroutine "FPQ" is called.

5.3.2.11 Subroutine "L4L"

5.3.2.11.1 Purpose

The subroutine "L4L" contains the algorithm necessary for determining the lot size with the lot-for-lot technique.

5.3.2.11.2 Variables

buckets	See Section 5.3.2.3.2.
date	See Section 5.3.2.4.2.
dMPS	See Section 5.3.2.4.2.
dSP	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.
i	Integer type; values range from 1 to 999. The do loop index and record number to read is stored into the variable i.
LR	Real type; format code f8.0. The lot size requirement is stored into the variable LR.
NR	Real type; format code f8.0. The net requirements quantity is stored into the variable NR.

*For flowchart, see Appendix C, page 613.

OH	See Section 5.3.2.4.2.
OR	See Section 5.3.2.4.2.
period	See Section 5.3.2.4.2.
SR	See Section 5.3.2.4.2.
sumDD	See Section 5.3.2.4.2.

5.3.2.11.3 Description of Flow*

To perform the lot-for-lot technique, the subroutine "L4L" uses a do loop to read the record; set LR = NR; and rewrite the information back onto the same record for all the records in the horizon data file.

5.3.2.12 Subroutine "FPQ"

5.3.2.12.1 Purpose

The subroutine "FPQ" contains the algorithm necesssary for determining the lot size with the fixed period quantity technique.

5.3.2.12.2 Variables

buckets	See Section 5.3.2.3.2.
date	Integer type; values range from 1 to 9999. The first day of period is stored into date.
date1	The value of date on record j is stored into the variable date1.
dMPS	See Section 5.3.2.4.2.
dMPS1	The value of dMPS on record j is stored into the variable dMPS1.
dSP	See Section 5.3.2.4.2.

*For flowchart, see Appendix C, page 613.

dSP1	The value of dSP on record j is stored into the variable dSP1.
dXO	See Section 5.3.2.4.2.
dXO1	The value of dXO on record j is stored into the variable dXO1
GR	See Section 5.3.2.4.2.
GR1	The value of GR on record j is stored into the variable GR1.
i	Integer type; values range from 1 to 999. The loop increment and the record number read is stored into the variable i.
j	Integer type; values range from 1 to 999. The record number j is read to add NR to the NR on record i.
lotp	See Section 5.3.2.10.2.
LR	Real type; format code f8.0. The lot size requirements is stored into the variable LR.
LR1	The value of LR on record j is stored into the variable LR1.
NR	Real type; format code f8.0. The net requirements quantity is stored into the variable NR.
NR1	The value of NR on record j is stored into the variable NR1.
OH	See Section 5.3.2.4.2.
OH1	The value of OH on record j is stored into the variable OH1.
OR	See Section 5.3.2.4.2.
OR1	The value of OR on record j is stored into the variable

	OR1.
SR	See Section 5.3.2.4.2.
SR1	The value of SR on record j is sstored into the variable SR1.
sumDD	See Section 5.3.2.4.2.
sumDD1	The value of sumDD on record j is stored into the variable sumDD1.

5.3.2.12.3 Description of Flow*

To perform the fixed period quantity lotting technique, the subroutine "FPQ" does the following:

Step 1. Set i = Ø.

Step 2. Increment i by 1.

Step 3. If i > buckets, return to the main program.

Step 4. Read record i in the horizon data file.

Step 5. Is NR = Ø, go to Step 2?

Step 6. When lotp ≤ time spanned, LR = NR. Otherwise, LR = NR + LR.

Step 7. If i > buckets, return to the main program. Otherwisse, go to Step 4.

5.3.2.13 Subroutine "OrderRelease"

5.3.2.13.1 Purpose

The subroutine "OrderRelease" time adjusts the lot sizes by the lead time to determine the order release.

---

*For flowchart, see Appendix C, page 615.

5.3.2.13.2 Variables

buckets	See Section 5.3.2.3.2. The starting point of the do loop for variable i is buckets.
c	Character type; maximum of 10 alphanumeric digits. The company part number is stored into the variable c.
currentperiod	Integer type; values range from 1 to 999. The number for the current period is stored into variable currentperiod.
d	Integer type; values range from -1 to 999. The down key for the binary search is stored into the variable d.
date	See Section 5.3.2.4.2
date1	The date on record j is stored into variable date1.
DDkey	Integer type; values range from 1 to 9999. The record number, used in the DD file, is stored into the variable DDkey. The DDkey is the pointer to the DD's produced by the part.
dMPS	See Section 5.3.2.4.2.
dMPS1	The dMPS on record j is stored into variable dMPS1.
dSP	See Section 5.3.2.4.2.
dSP1	The dSP on record j is stored into the varible dSP1
dXO	See Section 5.3.2.4.2.
dXO1	The dXO on record j is stored into the variable dXO1.
GR	See Section 5.3.2.4.2.
GR1	The GR on record j is stored into the variable GR1.
i	Integer type; values range from 1 to 999. The do loop index and the record number read for adjustment is stored into i.

j	Integer type. A do loop index variable is j. The do loop determines the time placement for the order.
k	Integer type; values range from 1 to 999. The directional key is stored into variable k.
leadtime	Integer type; values range from 1 to 9999. The lead time (in days) is stored into leadtime.
LR	See Section 5.3.2.4.2.
LR1	The LR on record j is stored into the variable LR1.
n	Integer type. The number of dependent demands stored for the part is kept track of in the variable n.
next	Integer type; values range from 1 to 999. The record number, read for adjustment, is stored into the variable next. The starting point of the do loop with variable j is next.
NR	See Section 5.3.2.4.2.
NR1	The NR on record j is stored into variable NR1.
OH	See Section 5.3.2.4.2.
OH1	The OH on record j is stored into variable OH1.
or	Real type; format code f8.0. The or variable is equal to the size of the order to be released.
OR	See Section 5.3.2.4.2.
OR1	The OR on record j is stored into variable OR1.
p	Integer type; values range from 1 to 999. The ordinal number for c is stored into p.
parttoexplode	Integer type; values range from 1 to 9999. The part to

explode is stored into the variable parttoexplode.

pastdue — Real type; format code f8.0. The total requirements, which are over due, are stored into the variable pastdue.

period — See Section 5.3.2.4.2.

period1 — The period on record j is stored into variable period1.

SR — See Section 5.3.2.4.2.

SR1 — The SR on record j is stored into variable SR1.

sumDD — See Section 5.3.2.4.2.

sumdd1 — The sumDD on record j is stored into variable sumDD1.

sys — Integer type; values range from 1 to 999. The system part number for c is stored into sys.

totdays — Integer type; values range from 1 to 9999. The number of days spanned is stored into totdays.

u — Integer type; values range from -1 to 999. The up key for the binary search is stored into the variable u.

5.3.2.13.3 Description of Flow*

To time adjust for the order release, the subroutine "OrderRelease" performs the following steps:

Step 1. Set n = Ø and pastdue = Ø.

Step 2. Read record number 1 in the horizon file to obtain the values for currentperiod and today.

Step 3. Set i = buckets.

Step 4. Read record i in the horizon file.

Step 5. Determine placement of order release for LR on record i.

*For flowchart, see Appendix C, page 617.

Step 6. Subtract 1 from i.

Step 7. If i ≥ Ø, go to Step 4.

Step 8. Sum up pastdue value from the LR's which can't be adjusted because lead time is too long.

Step 9. Return to main program

5.3.2.14 Subroutine "OutPut"

5.3.2.14.1 Purpose

The subroutine "OutPut" directs the printing of the results in the MRP matrix. The MRP matrix, stored in the horizon file, is the transpose of how the results is to be presented. Therefore, the data needs to be rearranged. A maximum of 6 periods at a time are rearranged by the subroutine "SIX". If the remaining periods are less than 6, then either "FIVE", "FOUR", "THREE", "TWO", or "ONE" handles the rearrangement. If a purchase order is required, the subroutine "OrderWrite" is called.

5.3.2.14.2 Variables

buckets — Integer type; values range from 1 to 999. The maximum number of periods is stored into buckets.

columns — Integer type; values range from 1 to 999. The number of periods left with data to be printed is stored into the variable columns.

date — Integer type; values range from 1 to 9999. The date of the order placement is stored into date.

i — Integer type; values range from 1 to 999. The period number, or record number, in the horizon is stored into the variable i.

leadtime	Integer type; values range from 1 to 9999. The lead time (in days) is stored into the variable leadtime.
n	Integer type. The number of subroutines used is kept track of in the variable n.
name	Character type; maximum of 12 alphanumeric digits. The company name is stored into the variable name.
orderdate	Integer type; values range from 1 to 9999. The date of the order placement is stored into orderdate.
orderperiod	Integer type; values range from 1 to 999. The period number for the order is stored into the variable orderperiod.
ordersize	Real type; format code f8.0. The number of units ordered is stored into ordersize.
parttoexplode	Integer type; values range from 1 to 999. The system number for the part exploded is stored into the variable parttoexplode.
period	Integer type; values range from 1 to 999. The period number for the order is stored into the variable period.
size	Real type; format code f8.0. The number of units ordered is stored into size.

5.3.2.14.3 Description of Flow*

To direct the print out of the MRP matrix and the purchase order, the subroutine "OutPut" performs the following steps:

---

*For flowchart, see Appendix C, page 621.

Step 1. Set n and i equal to Ø.

Step 2. Initialize columns = buckets at the maximum number of periods.

Step 3. Decrement columns by 6.

Step 4. If columns $\geq$ Ø, there is 6 periods to print call subroutine "SIX".

Step 5. If columns = -1, there are 5 periods left to print out call subroutine "FIVE".

Step 6. If columns = -2, there are 4 periods left to print out call subroutine "FOUR".

Step 7. If columns = -3, there are 3 periods left to print out call subroutine "THREE".

Step 8. If columns = -4, there are 2 periods left to print out call subroutine "TWO"

Step 9. If columns = -5, there is 1 period left to print out call subroutine "ONE".

Step 10. If n = 1, store the information period number, date and OR requirements from the most current period into the variables period, dates and size, respectively.

Step 11. If columns $\leq$ Ø, there is no more data to print out; see if order needs to be produced; and return to main program.

Step 12. Go to Step 3.

5.3.2.15 Subroutine "SIX"

5.3.2.15.1 Purpose

When called by the subroutine "OutPut", the subroutines "SIX" rearranges the data for the next 6 periods and sends the information formatted to the printer.

5.3.2.15.2 Variables

date*x*	Integer type; values range from 1 to 9999. The first day of the period for column *x* is stored into date*x*.
dMPS*x*	Real type; format code f8.0. The MPS demand quantity for column number *x* is stored into the variable dMPS*x*.
dSP*x*	Real type; format code f8.0. The SP demand quantity for column number *x* is stored into dSP*x*.
dXO*x*	Real type; format code f8.0. The XO demand quantity for column number *x* is stored into dXO*x*.
GR*x*	Real type; format code f8.0. The gross requirements for column number *x* is stored into GR*x*.
i	See Section 5.3.2.14.2
LR*x*	Real type; format code f8.0. The lot size requirements for column *x* is stored into LR*x*.
n	See Section 5.3.2.14.2.
NR*x*	Real type; format code f8.0. The net requirements for column *x* is stored into the variable NR*x*.
OH*x*	Real type; format code f8.0. The on hand inventory quantity for column *x* is stored into OH*x*.
OR*x*	Real type; format code f8.0. The order requirements for

column x is stored into ORx.

orderdate — See Section 5.3.2.14.2.

orderperiod — See Section 5.3.2.14.2.

periodx — Integer type; values range from 1 to 999. The period number for column x is stored into the variable periodx.

SRx — Real type; format code f8.0. The scheduled receipt for column x is stored into SRx.

sumDDx — Real type; format code f8.0. The sum of dependent demands for column x is stored into the variable sumDDx.

5.3.2.15.3 Description of Flow*

To print out the data for 6 periods, the subroutine "SIX" performs the following steps:

Step 1. Increment n by 1.

Step 2. Increment i by 1.

Step 3. Read record i in the horizon file and store the information into period1, date1, ..., OR1.

Step 4. Increment i by 1.

Step 5. Read record i in the horizon file and store the information into period2, date2, ..., OR2.

Step 6. Increment i by 1.

Step 7. Read record i in the horizon file and store the information into period3, date3, ..., OR3.

Step 8. Increment i by 1.

---

*For flowchart, see Appendix C, page 623.

Step 9. Read record i in the horizon data file and store the information into period4, date4, ..., OR4.

Step 10. Increment i by 1.

Step 11. Read record i in the horizon file and store the information into period5, date5, ..., OR5.

Step 12. Increment i by 1.

Step 13. Read record i in the horizon file and store the information into period6, date6, ..., OR6.

Step 14. Write out the period's information.

Step 15. Write out the date's information.

Step 16. Write out the dMPS's information.

Step 17. Write out the dXO's information.

Step 18. Write out the dSP's information.

Step 19. Write out the sumDD's information.

Step 20. Write out the GR's information.

Step 21. Write out the SR's information.

Step 22. Write out the OH's information.

Step 23. Write out the NR's information.

Step 24. Write out the LR's information.

Step 25. Write out the OR's information.

Step 26. Return to the subroutine "OutPut".

5.3.2.16 Subroutines "FIVE", ..., "ONE"

5.3.2.16.1 Purpose

The subroutines "FIVE", "FOUR", "THREE", "TWO", and "ONE" are needed to print out the data whenever there are less than 6 periods of

information. When 5 columns need to be rearranged and printed, the subroutine "FIVE" is called by the subroutine "OutPut". If 4 columns, then subroutine "FOUR" is called...If 1 column, then subroutine "ONE" is called.

5.3.2.16.2 Variables

See Section 5.3.2.15.2

5.3.2.16.3 Description of Flow*

To handle 5 columns, the subroutine "FIVE" follows the same steps as "SIX" outlined in Section 5.3.2.1.2 skipping Steps 12 and 13.

To handle 4 columns, the subroutine "FOUR" does the same steps as in "SIX" skipping numbers 10 to 13.

To handle 3 columns, the subroutine "THREE" does the same steps as in "SIX" skipping numbers 8 to 13.

To handle 2 columns, the subroutine "TWO" does the same steps as in "SIX" skipping numbers 6 to 13.

To handle 1 column, the subroutine "ONE" does the same steps as in "SIX" skipping numbers 4 to 13.

5.3.2.17 Subroutine "OrderWrite"

5.3.2.17.1 Purpose

The subroutine "OrderWrite" is designed to print a formatted order. The order includes the company name, the name of the part, the date the order was placed, the size of the order, and the expected due date of the order.

---

*For flowchart, see Appendix C, page 628-648.

5.3.2.17.2 Variables

date	See Section 5.3.2.14.2.
duedate	Integer type; values range from 1 to 999. The expected due date of the order is stored into duedate.
i	Integer type. The variable i is the do loop index varible.
leadtime	Integer type; values range from 1 to 999. The lead time (in days) is stored into the variable leadtime.
name	Character type; maximum of 12 alphanumeric digits. The company name is stored into the variable name.
part	Character type; maximum of 10 alphanumeric digits. The company part number is stored into part.
parttoorder	Integer type; values range from 1 to 999. The system part number to be ordered is stored into parttoorder. Note: parttoexplode = partoorder.
period	See Section 5.3.2.14.2
size	See Section 5.3.2.14.2.

5.3.2.17.3 Description of Flow

To produce a order, the subroutine "OrderWrite" follows these steps:

Step 1. Read the company part name from record parttoorder in the partid data file.

Step 2. Calculate expected due date by adding date plus leadtime.

Step 3. Write company name (name).

Step 4. Write "Purchase Order"

Step 5. Write period and date when order is placed.

Step 6. Write part and the number of units needed (size).

Step 7. Write expected duedate.

Step 8. Return to "OutPut" subroutine.

5.3.2.18 Subroutine "DependentDemands"

5.3.2.18.1 Purpose

The subroutine "DependentDemands" transfers the DD placed on the part by its parent parts into the MRP matrix stored into the horizon file.

5.3.2.18.2 Variables

BofM	Integer type; values range from 1 to 9999. The value of BofM is stored on the part's record in the partstatus file. The BofM is the bom record number where the part appears in the child parts list.
buckets	See Section 5.3.2.3.2.
c	Character type; maximum of 10 alphanumeric digits. The company part number is stored into c.
child	Integer type; values range from 1 to 999. The child part on the bom record is stored into the variable child.
ckey	Integer type; values range from 1 to 9999. The record number, of the next parent for child, is stored into ckey.
d	Integer type; values range from 1 to 999. The down key for the binary search is stored into d.
date	See Section 5.3.2.4.2.
day	Integer type; values range from 1 to 9999. The due date of the dependent demands is stored into day.
DDkey	Integer type; values range from 1 to 9999. The pointer to the dependent demand records is stored into the variable

	DDkey.
dMPS	See Section 5.3.2.4.2.
dSP	See Section 5.3.2.4.2.
dXO	See Section 5.3.2.4.2.
GR	See Section 5.3.2.4.2.
i	Integer type; values range from 1 to 999. The do loop index and the record number, read in the horizon file, is stored into the variable i.
k	Integer type; values range from 1 to 999. The directional key is stored into k.
kBOM	Integer type; values range from 1 to 9999. The record number, to read in the bom file, is stored into the variable kBOM.
kDD	Integer type; values range from 1 to 9999. The record number, to read in the DD data file, is stored into the variable kDD.
LR	See Section 5.3.2.4.2.
NR	See Section 5.3.2.4.2.
OH	See Section 5.3.2.4.2.
OR	See Section 5.3.2.4.2.
p	Integer type; values range from 1 to 999. The ordinal number for c is stored into the variable p.
parent	Integer type; values range from 1 to 999. The parent part on the bom record is stored into the variable parent.
part	Integer type; values range from 1 to 999. The system part

number on record kDD is stored into part.

period — See Section 5.3.2.4.2.

pkey — Integer type; values range from 1 to 9999. The record number, of the next child for parent, is stored into pkey.

qDD — Real type; format code f8.0. The number of units required by part is stored into qDD.

qtyc — Integer type; values range from 1 to 9999. The number of child per parent is stored into qtyc.

qtyDD — Real type; format code f8.0. The total number of child parts needed to meet qDD is stored into qtyDD.

s — Integer type; values range from 1 to 999. The system part number is stored in s.

SSR — See Section 5.3.2.4.2.

sumDD — Real type; format code f8.0. The total of all dependent demands is stored into the MRP matrix is accumulated in the variable sumDD.

u — Integer type; values range from 1 to 999. The up key for the binary search is stored into u.

5.3.2.18.3 Description of Flow*

To transfer the dependent demands to the MRP matrix, the subroutine "DependentDemands" does the following steps:

Step 1. Let kBOM = BofM.

Step 2. Read record kBOM in the bom file and obtain the values for parent, child, qtyc, pkey, and ckey.

Step 3. Read record parent in the partid file and obtain the values for c, s, k, p, u, d, and DDkey.

Step 4. If DDkey = -1, the parent requires no child parts at this time. Therefore, go to Step 11.

Step 5. Set kDD = DDkey.

Step 6. Read record kDD in the DD file to obtain the values for part, day, and qDD.

Step 7. If part ≠ parent, go to Step 11.

Step 8. Calculate the total number of child parts needed; i.e., qtyDD = qtyc * qDD.

Step 9. Include in correct bucket of the MRP matrix.

Step 10. Increment kDD by 1 and return to Step 6.

Step 11. If ckey = -1, there are no more parent parts who require this child part. Therefore, return to main program.

Step 12. Set kBOM = ckey and return to Step 2.

*For flowchart, see Appendix C, page 649.

5.4 SystemStatus

5.4.1 Introduction

To obtain the listings of various information stored away into the data files, the SystemStatus output program is designed to retrieve the information for the MRP User. The information, that can be retrieved interactively, is the following:

1. the company parts list ordered;
2. the company parts list as entered;
3. the bills-of-materials list;
4. the company parts list sorted by the MRP flow.

5.4.2 Subroutines in SystemStatus

5.4.2.1 Overview

The subroutines in the SystemStatus output file include:

1. PartsList,
2. sortedparts,
3. unsortedparts,
4. BOMlist,
5. MRPflow.

(See Figure 5.4.)

The main program begins by asking the MRP User to specify the type of information needed. If a parts list is needed, then the MRP User types in 'part'. The subroutine "PartsList" is called and it asks for the type of parts list desired. If the ordered company parts list is desired, the subroutine "sortedparts" produces the list after the MRP User types in 's'. If the as entered company parts list is desired, the subroutine

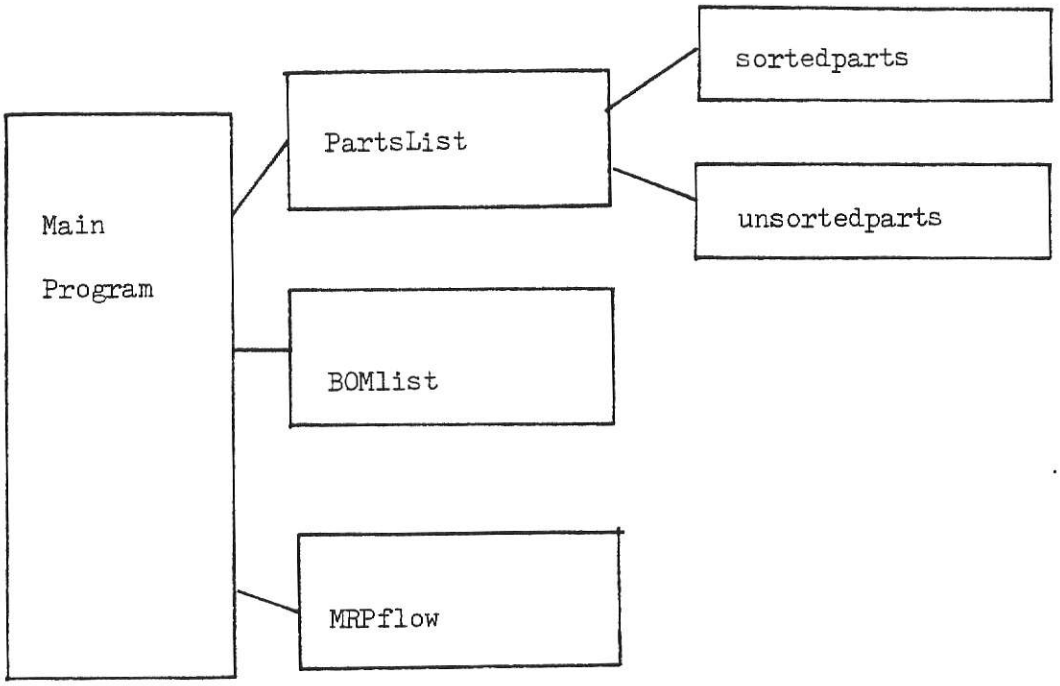


Figure 5.4
SystemStatus
Subroutines

"unsortedparts" prints out the list after the MRP User types in 'u'.

If the bills-of-materials list is needed, then the MRP User types in 'BOM'. The subroutine "BOMlist" prints out the list. If a list of parts ordered by the flow of the MRP explosion is wanted, then the MRP User types in 'flow'. This parts list is produced by the subroutine "MRPflow".

5.4.2.2 Subroutine "PartsList"

5.4.2.2.1 Purpose

The subroutine "PartsList" asks the MRP User to specify the type of parts list desired. The choice is either a company parts list as entered or sorted.

5.4.2.2.2 Variables

answer — Character type; answer $\in$ {s, u, quit}. The MRP User response to the type of parts list is stored into the variable answer.

5.4.2.2.3 Description of Flow*

To direct the printing of the desired parts list, the subroutine "PartsList" asks the MRP User to respond to the following statements:

'To get sorted list, type s.
To get as entered list, type u.
To stop, type quit'.

The MRP User types in his response at the keyboard and it is stored into the variable answer. If answer = 's', he subroutine "sortedparts" is called. If answer = 'u', the subroutine "unsortedparts" is called.

*For flowchart, see Appendix C, page 653.

5.4.2.3 Subroutine "sortedparts"

5.4.2.3.1 Purpose

The subroutine "sortedparts" prints out the ordered company parts list. (The program run for the randomized file in Example 4.2.2.2.5 produces the results shown in Figure 5.5.)

5.4.2.3.2 Variables

c	Character type; maximum of 10 alphanumeric digits. The company part number is stored into variable c.
first	Integer type, values range from 1 to 999. The company part number, with ordinal number one, is stored on record number first.
k	Integer type; values range from 1 to 999. Given c has ordinal number i, the record number, for the company part number with ordinal number i + 1, is stored into k.
key	Integer type; values range from 1 to 999. The record number, read in the partid file, is stored into the variable key.
s	Integer type; values range from 1 to 999. The system part number for c is stored into the variable s.

5.4.2.3.3 Description of Flow*

To print out a sorted parts list, the subroutine "sortedparts" does the following steps:

Step 1. Read record Ø in the partid file to obtain the value of first.

*For flowchart, see Appendix C, page 654.

```
Parts_List
a
b
c
d
e
f
g
h
i
j
```

Figure 5.5
Sorted Company
Part Number List

```
Parts_List
e
i
g
d
h
f
a
b
j
c
```

Figure 5.6
Unsorted Company
Part Number List

Step 2. Set key = first, the record number of the company part number with ordinal number one.

Step 3. Read record key in the partid file for the company part number (c) and k.

Step 4. List c on printer.

Step 5. If k = 999, there are the last part in the list has been reached. Therefore, return to subroutine "PartsList".

Step 6. Set key = k, next part in list.

Step 7. Go to Step 3.

5.4.2.4 Subroutine "unsortedparts"

5.4.2.4.1 Purpose

The subroutine "unsortedparts" prints out the company parts list as it was entered. (The program run for the randomized file in Example 4.2.2.2.5 produces the results shown in Figure 5.6.)

5.4.2.4.2 Variables

c	See Section 5.4.2.3.2
first	See Section 5.4.2.3.2.
i	Integer type; values range from 1 to 999. The do loop index and the record number read is stored into the variable i.
next	Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of records in the partid file.

5.4.2.4.2 Description of Flow*

To print out the list of parts as it was entered, the subroutine "unsortedparts" performs the following steps:

Step 1. Read record Ø in the partid file to obtain the values of next.

Step 2. Set i = Ø.

Step 3. Increment i by 1.

Step 4. If i > next -1, return to the subroutine "PartsList".

Step 5. Read record i in the partid file to obtain the valu of c.

Step 6. List c on the printer.

Step 7. Go to Step 3.

5.4.2.5 Subroutine "BOMlist"

5.4.2.5.1 Purpose

The subroutine "BOMlist" prints out the bills-of-materials list. The bills-of-materials is stored into the bom data file by the part's system number. However, the bills-of-materials, produced by the "BOMlist" subroutine, is the company parts numbers for both the parent and child parts. (See sample print out in Figure 5.7.)

5.4.2.5.2 Variables

c	Integer type; values range from 1 to 999. The system number for the child part, on the bom record i is stored into c.
child	Character type; maximum of 10 alphanumeric digits. The company part number for c is stored into child.

*For flowchart, see Appendix C, page 655.

Bills of Material

Parent	Child	Quantity
a	b	2
a	c	1
a	e	2
c	d	1
c	b	1
e	f	1
e	c	2
g	f	1
e	d	2
g	c	2
g	h	1
h	i	1
i	f	3
i	d	4
j	b	1
j	i	2

Figure 5.7
BOM list

Parts sorted by MRP flow order.

a
g
j
e
h
c
i
b
d
f

Figure 5.8
MRP Flow List

i	Integer type; values range from 1 to 999. The do loop index and the record number, read in the bom file, is stored into the variable i.
next	Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of records in the bom data file.
p	Integer type; values range from 1 to 999. The system part number for the parent part, on the bom record i, is stored into p.
parent	Character type; maximum of 10 alphanumeric digits. The company part number for p is stored into parent.
pkey	Integer type; values range from -1 to 9999. The record number for the next child part for the parent part, is stored into the variable pkey.

5.4.2.5.3 Description of Flow*

To print out the bills-of-materials, the subroutine "BOMlist" performs the following steps:

Step 1. Read record Ø in the bom file to obtain the value for next.

Step 2. Let i = Ø.

Step 3. Increment i by 1.

Step 4. If i > next -1, return to main program.

Step 5. Read record i in the bom file for the parent system part number (p) and the child system part number (c).

*For flowchart, see Appendix C, page 656.

Step 6. Obtain the company part numbers for p and c from the partid file; and store the values into parent and child, respectively.

Step 7. List the bills-of-materials record (parent, child, qty) at the printer.

Step 8. Go to Step 3.

5.4.2.6 Subroutine "MRPflow"

5.4.2.6.1 Purpose

The subroutine "MRPflow" prints out the list of company part numbers. The parts are sorted according to their MRP processing order. In other words, the MRP flow order. (For sample print out, see Figure 5.8.)

5.4.2.6.2 Variables

answer	Character type; answer ε {yes, no}. The response to the question about the level assignment is stored into answer.
BofM	See Section 4.3.2.2.2.
BOM	See Section 4.3.2.2.2.
c	See Section 5.4.2.3.2.
ccost	See Section 4.3.2.2.2.
ccper	See Section 4.3.2.2.2.
first	Integer type; values range from 1 to 999. The record number, of the first part at level = 1, is stored into first.
k	Integer type; values range from 1 to 999. The record k, of the partstatus file, is read for the value of levelkey.
lead	See Section 4.3.2.2.2.

level — Integer type. The part level number is stored into the variable level.

levelkey — Integer type; values range from -1 to 999. The record number, for the next part to be exploded, is stored into the variable levelkey. If levelkey = -1, the last part in the list has been reached.

lotp — See Section 4.3.2.2.2.

lotr — See Section 4.3.2.2.2.

MPSkey — Integer type; values range from -1 to 999. The key to the Master Production Schedule demands is MPSkey.

next — Integer type; values range from 1 to 999. The quantity of next -1 is equal to the number of parts in the partstatus file.

numkids — See Section 4.3.2.2.2.

oh — See Section 4.3.2.2.2.

pu — See Section 4.3.2.2.2.

pup — See Section 4.3.2.2.2.

SR — See Section 4.3.2.2.2.

SPkey — Integer type; values range from -1 to 999. The key to the Service Part demands is SPkey.

XOkey — Integer type; values range from -1 to 9999. The key to the External Order demands is XOkey.

5.4.2.6.3 Description of Flow*

To print out the parts according to the MRP processing order, the subroutine "MRPflow" performs the following steps:

Step 1. Ask the MRP User if he has run the Level program. If no, return to main program.

Step 2. Read record Ø in the partstatus file for the value of first.

Step 3. Set k = first.

Step 4. Read record k in the partstatus file to obtain the value of levelkey.

Step 5. Read record k in the partid file for the company part number (c).

Step 6. Transmit the value of c to the printer.

Step 7. If levelkey = -1, return to main program.

Step 8. Let k = levelkey.

Step 9. Go to Step 4.

*For flowchart, see Appendix C, page 658.

6. User's Guide

6.1 Steps for Explosion

The step-by-step instructions to the MRP explosion is explained in this section. The steps to doing specific types of data file editing are explained in the next section. Before going any further, it is recommended that the MRP user browse Section 3 where the data files are described, and Sections 4 and 5 where the functions of the programs are discussed. Also, reading "Chapter 7 Waterloo MicroEditor" in the System Overview: Commodore SuperPET is recommended.

The steps are as follows:

Step 1. Turn on the CRT/keyboard unit.

When facing the CRT screen, the turn-on switch is on the back left-hand side. See Figure 6.1.

Step 2. Turn on the disk drive unit.

On Model 8050, the turn-on switch is on the back left-hand side when facing the front of the unit. See Figure 6.2.

Step 3. Turn on the printer unit.

On Tractor Printer 8023P, the turn-on switch is on the right side panel. See Figure 6.3.

Step 4. Set the microprocessor (three-stage toggle) switches.

The switches are located on the lower right-side panel of the CRT/keyboard unit. See Figure 6.4. Facing the 2 switches, the one on the left is positioned straight down for the R/W option. The other switch on the right is positioned to the left under the 6809 option.

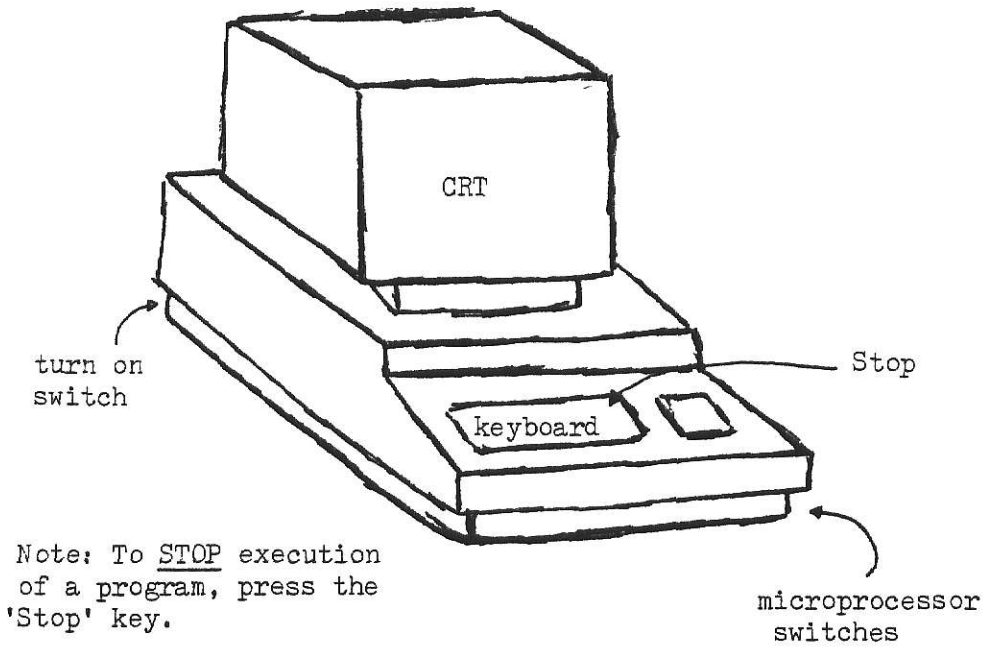
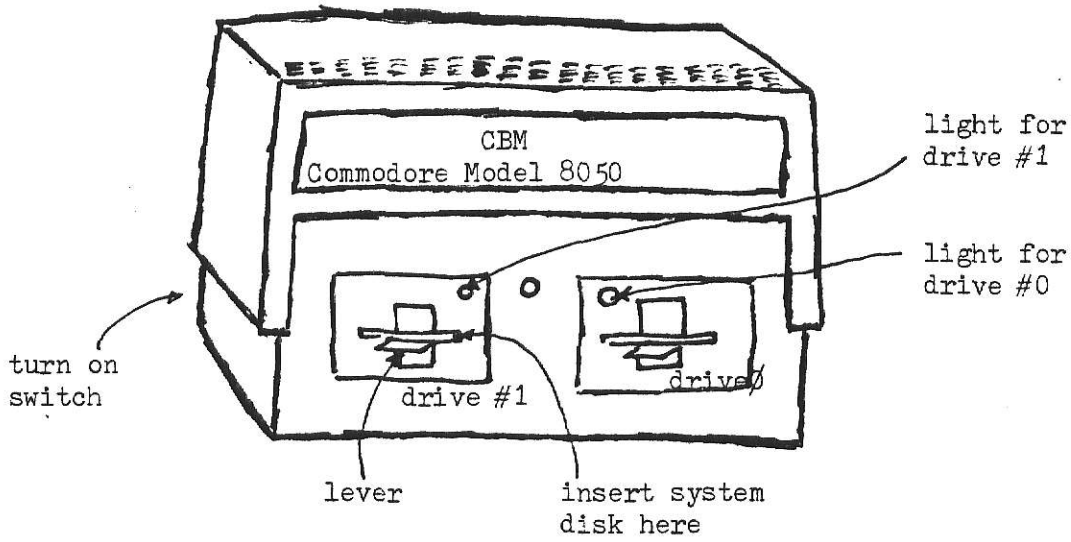


Figure 6.1
CRT/keyboard Unit



Warnings: 1. Remove all disks before turning the disk unit off.
2. If the light for either drive #1 or #0 is green, don't remove disks

Figure 6.2
Disk Drive Unit

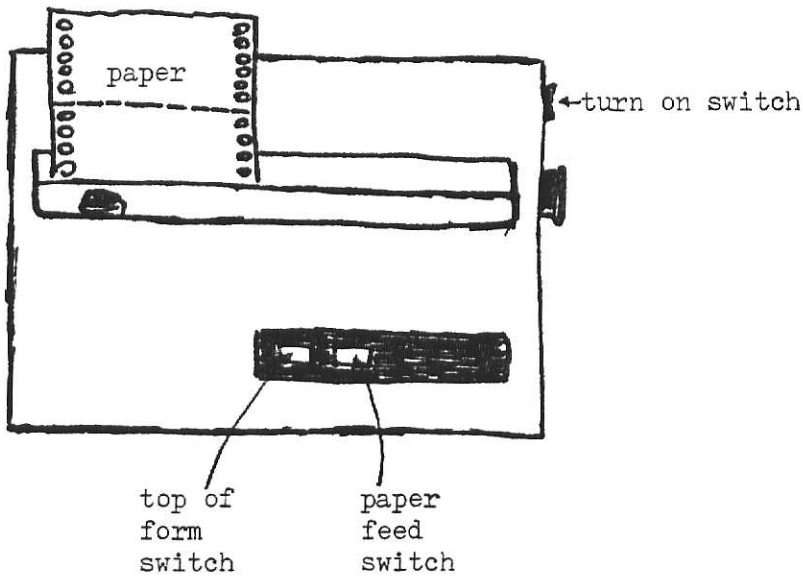


Figure 6.3
Top View of Commodore
Tractor Printer 8023P

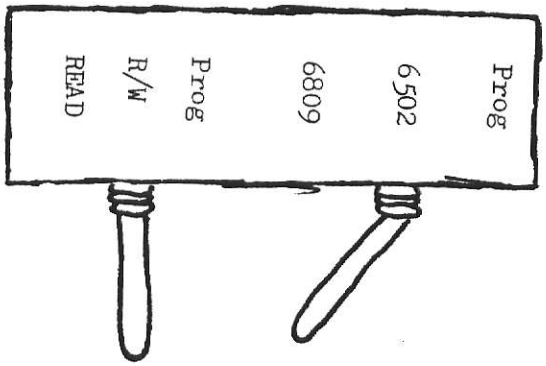


Figure 6.4
Microprocessor Switches

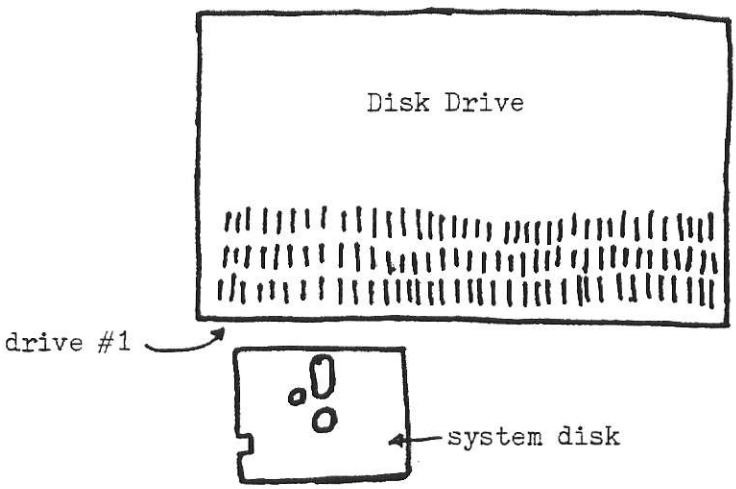


Figure 6.5
Positioning of Floppy Disk

Step 5. Load system floppy disk into Disk Drive #1.

See Figure 6.5 for the correct positioning of the disk. Insert the disk into the Disk Drive and press down on the lever. The system floppy disk contains the Waterloo Fortran which needs to be loaded into the computer's memory.

Step 6. Load the Waterloo Fortran into the computer's memory.

The CRT screen should contain the following:

"Waterloo microSystems
Select:
setup
monitor
apl
basic
edit
fortran
pascal
development."

If not, return to Step 4. Otherwise, type 'f' and press the return key. After the fortran has been loaded, the message "Waterloo microFORTRAN Version 1.1... Press <Return>" will appear on the screen. Press the return key. If a 'Program not found' message appears on the screen, make sure the disk is properly inserted and has the Waterloo Fortran on it.

Step 7. Load the MRP disk into Disk Drive #0.

Step 8. Create the partid data file.

Step 8a. Type 'g PartFile' and press the return key. This loads the program PartFile in the computer's memory.

Step 8b. Type 'run' and press the return key. This starts the program executing.

Step 8c. The question 'Is part file to be established? or appended? yes, no' appears on the CRT screen. Type 'yes' and press the return key.

Step 8d. The question 'new or add?' appears on the CRT screen. Type 'new' and press the return key.

Step 8e. The question 'Part? or quit.' appears on the CRT screen. Type the part number to be entered and press the return key. When all the parts have been entered, type 'quit' and press the return key.

Step 8f. '...Stop' appears on the CRT screen. This indicates that a normal termination has taken place. Press return key.

Step 8g. Type '*d' and press the return key. This clears the PartFile program out of the computer's memory.

Step 9. Create the bom record.

Step 9a. Type 'g BOMfile' and press the return key. This loads the program BOMfile into the computer's memory.

Step 9b. Type 'run' and press the return key. This starts the program executing.

Step 9c. The question 'Does BOM need to be created, appended or existing information altered? type either new, add, or change.' appears on the CRT screen. Type 'new' and press the return key.

Step 9d. The statement 'parent, # of children or quit, Ø to stop' appears on the CRT screen. Type the name of the parent part and the number of different child parts (or components) required to make the parent. Press the return key.

Step 9e. The statement 'child, quantity' appears on the CRT screen. Type the name for the child part and the quantity of child parts per parent part. Press the return key. NOTE: Continue working with Steps 9d and 9e until all the bill-of-material information has been supplied. When all the information is entered, type 'quit, Ø' in response to the statement 'parent, # of children or quit, Ø or stop'. Press the return key.

Step 9f. When '...Stop' appears on the CRT screen, the program has terminated normally. Press the return key.

Step 9g. Type '*d' and press the return key. This clears the BOMfile program out of the computer's memory.

Step 10. Enter the on-hand inventory quantities and the scheduled receipts data.

Step 10a. Type 'g DataPlus' and press the return key. This loads the DataPlus program file into the computer's memory.

Step 10b. Type 'run' and press the return key.

Step 10c. The message 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit.' appears on the CRT screen. Type 'invandsr' and press the return key.

Step 10d. The message 'To create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory quantities, type onhand. To stop, type quit.' appears on the CRT screen. Type 'new' and press the return key.

Step 10e. The statement 'part, inventory quantity, # of SR or quit, Ø, Ø to stop' appears on the CRT screen. Type the part number, the quantity of parts in the stockroom, and the number of scheduled receipts planned for the part. Press the return key.

Step 10f. The statement 'day expected, quantity expected' appears on the CRT screen. Type the requested specifics for the scheduled receipts. Press the return key. Note: Continue working with Steps 10e and 10f until all the inventory and scheduled receipt data are entered. To stop, type 'quit, Ø, Ø' to the statement in Step 10e and press the return key.

Step 10g. The message in Step 10 d appears on the CRT screen. Type 'quit' and press the return key.

Step 11. Enter the lead time and lotting rule data.

Step 11a. The message in Step 10c appears on the screen. Type 'ltandlr' and press the return key.

Step 11b. The message 'To enter both lead time and lotting rule, type ltlr. To enter just lead time, type lt. To enter just lotting rule, type lr. To stop, type quit.' appears on the CRT screen. Type 'ltlr' and press the return key.

Step 11c. The message 'Leadtime was initialized at five days. Lotting rule was initialized at L4L (lot-for-lot). part, leadtime (days), lot rule (14L, FPQ) or quit, Ø, q to stop.' appears on the

CRT screen. Type the part number, its lead time with the unit of days, and the lotting rule either 'L4L' or 'FPQ'. Press the return key.

If the lotting rule is FPQ, the 'period (days) for FPQ' message appears on the screen. Type the total number of days the lotting period covers. Press the return key.

Note: Continue working with Step 11c until all the data has been entered. Then, type 'quit, Ø, q to stop and press the return key.

Step 11d. The message in 11b appears on the CRT screen. Type 'quit' and press the return key.

Step 12. Enter the dates for the planning horizon.

Step 12a. The message in Step 10c appears on the CRT screen. Type 'ph' and press the return key.

Step 12b. The question 'number of buckets?' appears on the CRT screen. Type the number of buckets (or periods) needed for the planning horizon and press the return key. Then, statement 'start with period number', appears, type the number for the first period in the planning horizon and press the return key.

Step 12c. The question 'Equal bucket size? yes or no.' appears on the CRT screen. Type the answer into the keyboard and press the return key.

Step 12d. If 'yes' was typed in Step 12c, go to Step 12e. If 'no' was typed, the statement 'Period period number day one is __' appears on the screen for each period number. Type the response in at the keyboard and press the return key each time.

Step 12e. The message in Step 10c appears on the CRT screen. Type 'quit' and press the return key.

Step 12f. The word '...Stop' appears on the CRT screen to indicate that a normal termination has occurred. Press the return key.

Step 12g. Type '*d' and press the return key. The DataPlus program is removed from the computer's memory.

Step 13. Enter the MPS demand information.

Step 13a. Load the DataPlus2 program by typing 'g DataPlus2' and pressing the return key.

Step 13b. Start the program executing by typing 'run' and pressing the return key.

Step 13c. When the message 'To enter MS demands, type MS. To enter external orders, type XO. To enter service parts, type SP. To stop, type quit.", type 'MS' and press the return key.

Step 13d. When the message 'new, add, change, or quit MS demand list?' appears on the CRT screen, type 'new' and press the return key.

Step 13e. When the statement 'part or quit to stop' appears on the CRT screen, type in the part number for the part to be produced and press the return key.

Step 13f. When the statement 'date, demand or -1, Ø to stop' appears on the CRT, type in the date and quantity of the part demand for the part indicated in Step 13e; and press the return key.

Note: Enter the demand specifications with Step 13f. When all the demands for the part type in at Step 13e have been entered, type '-1, Ø' and press the return key. Then, the message in Step 13e appears

again for the next part number to be entered. If all the demands for all the parts have been entered, type 'quit' and press the return key.

Step 13g. After 'quit' has been typed to the message 'part or quit to stop', the message in Step 13d appears on the CRT screen. Type 'quit' and press the return key.

Step 14. Enter external order demand.

Step 14a. When the message in Step 13c appears on the CRT screen, type 'XO' and press the return key.

Step 14b. When the message 'new, add, change or quit XO demand list?' appears on the CRT screen, type 'new' and press the return key.

Step 14c. When the statement 'part or quit to stop' appears on the CRT screen, type in the part number for the part to be produced and press the return key.

Step 14d. When the statement 'date, demand or -1 Ø to stop' appears on the CRT, type in the date and quantity of the part indicated in Step 14c; and press the return key.

Note: Enter the demand specifications with Step 14d. When all the demands have been entered for the part typed in at Step 14c, type '-1, Ø' and press the return key. The message in Step 14c appears again for the next part number to be entered. If all the demand for all the parts have been entered, type 'quit' and press the return key.

Step 14e. After 'quit' has been typed to the message 'part or quit

to stop', the message in Step 14b appears on the CRT screen. Type 'quit' and press the return key.

Step 15. Enter service part demands.

Step 15a. When the message in Step 13c appears on the CRT screen, type 'SP' and press the return key.

Step 15b. When the message 'new, add, change, or quit SP demand list?' appears on the CRT screen, type 'new' and press the return key.

Step 15c. Do Steps 14c to 14e.

Step 15f. When the statement in Step 13c appears on the CRT screen, type 'quit' and press the return key.

Step 15g. When '...Stop' appears on the CRT screen, the DataPlus2 has terminated normally. Press the return key.

Step 15h. Type '*d', then the program DataPlus2 is removed from the computer's memory.

Step 16. Run the Level program.

Step 16a. Load the Level program into the computer's memory by typing 'g Level' and pressing the return key.

Step 16b. Type 'run' and press the return key to start the program executing.

Step 16c. When '...Stop' appears on the CRT screen, the execution has terminated normally. Press the return key.

Step 16d. Type '*d' to clear the program Level out of the computer's memory.

Step 17. Run the MRP program.

Step 17a. Load the MRP program into the computer's memory by typing

'g MRP' and pressing the return key. Note: Make sure the printer is on.

Step 17b. Type 'run' and press the return key to start the program executing.

Step 17c. When the "company name (up to 12 characters in length)?' appears on the CRT screen, type in the name for the order form and press the return key.

Note: The MRP explosion is being performed. Be patient it will take a little while.

Step 17d. When '...Stop' appears on the CRT screen, the program has terminated normally. Press the return key.

Step 17e. Type '*d' and press the return key to clear the MRP program out of the computer's memory.

Step 18. Log-off.

Step 18a. Type 'bye' and press the return key.

Step 18b. Remove both disks from disk drive.

Step 18c. Turn off all three pieces of equipment - the printer, the disk drive, and the CRT/keyboard. Note: See Figure 6.6 for a summary of the steps.

1. Turn on CRT/keyboard Unit
2. Turn on Disk Drive
3. Turn on Printer
4. Set the two Microprocessor Switches
5. Load System Disk into Drive #1
6. Load FORTRAN into Computer
7. Load MRP Disk into Drive #Ø
8. Create partid File
9. Create bom File
10. Enter Inventory and Scheduled Receipt Data
11. Enter Lead Time and Lotting Rule Data
12. Enter Planning Horizon Specifications
13. Enter MPS Demands
14. Enter XO Demands
15. Enter SP Demands
16. Run Level
17. Run MRP
18. Log off

Figure 6.6 Steps for Explosion

6.2 Editing Guide

6.2.1 To Create New partid File

To create a new partids file, perform the following steps:

Step 1. Perform Steps 1 to 7 in Section 6.1 if they haven't already been done.

Step 2. If Step 1 has already been done, type '*d' and press the return key to clear out any MRP programs in the computer's memory.

Step 3. Type 'g PartFile' and press the return key to load the program PartFile into the computer's memory.

Step 4. Type 'run' and press the return key to start the program executing.

Step 5. When the message 'Is part file to be established? or appended? yes, no' appears on the CRT screen, type 'yes' and press the return key.

Step 6. When the message 'new or add?' appears on the CRT screen, type 'new' and presss the return key.

Step 7. Whenever the message 'Part? or quit' appears on the CRT screen, type in one part name; and press the return key. Each time the message appears another part number may be entered. When no additional parts are to be entered, type 'quit' and press the return key.

Step 8. When '...Stop' appears on the CRT screen, press the return key.

Step 9. Type '*d' and press the return key to clear the program PartFile out of the computer's memory.

6.2.2 To Add More Parts to the partid File

To add more parts to the partid file, perform the steps outlined in Section 6.2.1 with the exception of Step 6. For Step 6, type 'add' and press the return key.

6.2.3 To Create New bom File

To create a new bom file, perform the following steps:

Step 1. Perform Steps 1 to 7 in Section 6.1 if they haven't already been done.

Step 2. If Step 1 has already been done, type '*d' and press the return key.

Step 3. Load the BOMfile into the computer's memory by typing 'g BOMfile' and pressing the return key.

Step 4. Type 'run' and press the return key. When the message 'Does BOM need to be created, appended or existing information altered? type either new, add, or change'. appears on the CRT screen, type 'new' and press the return key.

Step 5. When the statement 'parent, # of children or quit, Ø to stop' appears on the CRT screen, type the name for the parent part and the number of different kinds of children the parent assembly requires. Press the return key.

Step 6. When the statement 'child, quantity' appears on the CRT screen, type the name of the child parts and the number of that child part required by the parent assembly; and press the return key.

Note: Continue typing in the appropriate part names and quantities for the statements listed in Steps 5 and 6. When no more bill-of-

material information is to be entered, type 'quit, Ø' in response to the statement in Step 5.

Step 7. When '...Stop' appears on the BOMfile program has terminated normally, press the return key.

Step 8. Type '*d' and press the return key to clear the BOMfile out of the computer's memory.

6.2.4 To Add More bom Records

To add more bom records, perform the following steps:

Step 1. Perform Steps 1 to 7 in Section 6.1 if they haven't already been done.

Step 2. If Step 1 has already been done, type '*d' and press the return key.

Step 3. Load the BOMfile into the computer's memory by typing 'g BOMfile' and pressing the return key. Type 'run' and press the return key to start the program executing.

Step 4. When the message 'Does BOM need to be created, appended or existing information altered? type either new, add, or change' appears on the CRT screen, type 'add' and press the return key.

Step 5. When the statement 'parent, child, qty to be added or quit, q, Ø to stop' appears on the CRT screen, type in the parent part's name, the child part's name, and the quantity of child parts per parent part; and press the return key.

Note: Additional bom records are created by each response to the statement in Step 5. When not additional bom records are needed, type 'quit, q, Ø' and press the return key.

Step 6. When '...Stop' appears on the CRT screen, press the return key.

Step 7. Type '*d' and press the return key to clear out the BOMfile from the computer's memory.

6.2.5 To Change bom Records

To change records in the bom file, perform the following steps:

Step 1. Perform Steps 1 to 7 in Section 6.1 if they haven't already been done.

Step 2. If Step 1 has already been done, type '*d' and press the return key.

Step 3. Load the BOMfile into the computer's memory by typing 'g BOMfile' and pressing the return key. Type 'run' and press the return key to start the program executing.

Step 4. When the message 'Does BOM need to be created, appended or existing information altered? type either new, add or change' appears on the CRT screen, type 'change' and press the return key.

Step 5. When the message 'Enter parent, child, qty as it now appears or quit, q, Ø to stop' appears on the CRT screen, type all three pieces of information on the bom record to be changed.

Step 6. When the message 'Enter parent, child, qty as it should appear or quit, q, Ø to stop' appears on the CRT screen, type in all three pieces of information for corrected bom record. Type 'quit, q, Ø' if an error in Step 5 has been noticed.

Note: Continue working with Steps 5 and 6 until all necessary corrections for the bom file have been made. To stop, type 'quit, q,

Ø' to the message listed in Step 5.

Step 7. When '...Stop' appears on the CRT screen, press the return key.

Step 8. Type '*d' and press the return key to clear the BOMfile out of the computer's memory.

6.2.6 To Revise Inventory Quantities and Scheduled Receipt Information

To completely update all the on-hand inventory and scheduled receipt data, perform the following steps:

Step 1. Perform Steps 1 to 7 in Section 6.1 if they haven't been done.

Step 2. If Step 1 has already been done, type '*d' and press the return key to clear out any MRP programs in the computer's memory.

Step 3. Type 'g DataPlus' and press the return key to load the program DataPlus into the computer's memory.

Step 4. Type 'run' and press the return key to start the program executing.

Step 5. When 'What data do you wish to enter? Inventory and/or scheduled receipts, type invandsr. Lead time and lotting rules, type ltandlr. Periods for the planning horizon, type ph. To stop, type quit.' appears on the CRT screen, type 'invandsr' and press the return key.

Step 6. When 'To create SR file and initialize on hand inventory, type new. To change SR file data, type SR. To change on hand inventory quantities, type on hand. To stop, tpe quit' appears, type 'new' and press the return key.

Step 7. When 'part, inventory quantity, # of SR or quit, Ø, Ø to stop' appears, type the part number, the number of units in the stockroom, and the number of scheduled receipts planned for that part. Press the return key.

Step 8. When 'day expected, quantity expected' appears, type in the shop calendar date and the quantity of units expected in the scheduled receipt. Press the return key. Note: Continue responding to the statements in Steps 7 and 8 until all information has been entered. When no more information is to be entered, type 'quit, Ø, Ø' and press the return key.

Step 9. When the statement 'To create SR file and initialize on hand inventory...' appears, type 'quit' and press the return key.

Step 10. When the statement 'What data do you wish to enter?...' appears type 'quit' and press the return key.

Step 11. When '...Stop' appears, press the return key.

Step 12. Type '*d' and press the return key to clear the DataPlus program out of the computer's memory.

6.2.7 To Change on-hand Inventory Quantity

To change only the on-hand inventory quantities, perform the following steps:

Step 1. Do Steps 1 to 5 in Section 6.2.6.

Step 2. When 'To create SR file...' appears on the CRT screen, type 'onhand' and press the return key.

Step 3. When 'part, on hand quantity or quit, Ø' appears on the CRT screen, type the part's name and the number of parts in the stockroom.

Then, press the return key.

Step 4. When 'To create SR file...' appears, type 'quit' and press the return key.

Step 5. When 'What data do you wish to enter?...' appears, type 'quit' and press the return key.

Step 6. When '...Stop' appears, press the return key.

Step 7. Type '*d' and press the return key to clear the DataPlus program out of the computer's memory.

6.2.8 To Change Scheduled Receipt Record

To change a scheduled receipt record, perform the following steps:

Step 1. Do Steps 1 to 5 in Section 6.2.6.

Step 2. When 'To create SR file...' appears on the CRT screen, type 'SR' and press the return key.

Step 3. When 'To change record in SR file, type change. To delete record in SR file, type delete. To include new records in SR, type add. To stop, type quit' appears type 'change' and press the return key.

Step 4. When 'part to be changed or quit to stop' appears, type in the part's name which has the scheduled receipt needing to be changed.

Step 5. When 'day, qty' of SR as it now appears' is written on the screen, type the date and quantity of parts on the scheduled receipt record to be changed. Press the return key.

Step 6. When 'day, qty of SR as it should appear' is written on the CRT screen, type in the shop calendar date and the quantity of parts for the corrected scheduled receipt record. Press the return key.

Note: When no more scheduled receipts records are to be changed, type 'quit' to the statement' part to be changed or quit to stop'. Press the return key.

Step 7. When 'To change record in SR file...' appears, type 'quit' and press the return key.

Step 8. When 'What do you wish to enter?...' appears, type 'quit' and press the return key.

Step 9. When '...Stop' appears on the CRT screen, press the return key.

Step 10. Type '*d' and press the return key to clear the DataPlus program out of the computer's memory.

6.2.9 To Delete Scheduled Receipt Record

To delete scheduled receipts records, perform the following steps:

Step 1. Do Steps 1 to 5 in Section 6.2.6.

Step 2. When 'To create SR file...' appears, type 'SR' and press the return key.

Step 3. When 'To change record in SR file...' appears, type 'delete' and press the return key.

Step 4. When the message 'part to be deleted or quit to stop' appears on the CRT, type the name of the part whose scheduled receipt is to be deleted. Press the return key.

Step 5. When 'day, qty as it now appears in the SR file' appears, type the date when expected and the quantity of parts expected for the scheduled receipt to be deleted. Press the return key.

Note: When no more deletions are to be made, type 'quit' to the message

'part to be deleted or quit to stop' and press the return key.

Step 6. When 'To change record in SR file...' appears on the screen, type 'quit' and press the return key.

Step 7. When 'What do you wish to enter?...' appears, type 'quit' and press the return key.

Step 8. When '...Stop' appears on the CRT screen, press the return key.

Step 9. Type '*d' and press the return key to clear the DataPlus program out of the computer's memory.

6.2.10 To Add Scheduled Receipt Records

To add scheduled receipt records to the SR data file, do the following steps:

Step 1. Do Steps 1 to 5 in Section 6.2.6.

Step 2. When 'To create SR file...' appears on the CRT, type 'SR' and press the return key.

Step 3. When 'To change record in SR file...' appears, type 'add' and press the return key.

Step 4. When 'part to be added or quit to stop' appears on the CRT, type the name of the part with the SR To be added. Press the return key.

Step 5. When 'day, qty' appears on the CRT screen, type the specific date and quantity of parts expected in the scheduled receipt. Press the return key.

Note: Type 'quit' and press the return key to the message 'part to be added or quit to stop' when no more scheduled receipt additions are to be

made.

Step 6. When 'To change record...' appears on the CRT screen, type 'quit' and press the return key.

Step 7. When 'What do you wish to enter?...' appears, type 'quit' and press the return key.

Step 8. When '...Stop' appears on the screen, press the return key.

Step 89. Type '*d' and press the return key to clear the DataPlus program out of the computer's memory.

6.2.11 Equal Buckets for Planning Horizon

To create a planning horizon with equal buckets with a five day work week, perform the following steps:

Step 1. Do Steps 1 to 4 in Section 6.2.6.

Step 2. When 'What data do you wish to enter?...' appears, type 'ph' and press the return key.

Step 3. When 'number of buckets?' appears on the CRT, type the number of periods or buckets desired for the planning horizon. Press the return key.

Step 4. When 'Start with period number' appears, type the period number that the first bucket in the planning horizon should be. Press the return key.

Step 5. When 'Equal bucket size? yes or no' appears, type 'yes' and press the return key.

Step 6. When 'What data do you wish to enter?...' appears, type 'quit' and press the return key.

Step 7. When '...Stop' appears on the CRT screen, press the return

3 2 1

MRP EXPLOSION FOR THE COMMODORE COMPUTER

by

Karen Marie May

B.S., Florida State University, 1975

B.S. I.E., Kansas State University, 1982

A Master's Thesis

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

Kansas State University

Manhattan, Kansas

1983

Approved by:

L. C. Frash

Major Professor

vol. 2

LD
2668
.T4
1983
M39
vol. 2
c. 2

A11202 609835

key.

Step 8. Type '*d' and press the return key to clear the program DataPlus out of the computer's memory.

6.2.12 Unequal Buckets for planning horizon

To enter dates and period numbers for planning horizon without equal bucket size, perform the following steps:

Step 1. Do Steps 1 to 4 in Section 6.2.11.

Step 2. When 'Equal bucket size? yes or no' appears, type 'no' and press the return key.

Step 3. For each 'Period number day one is ___' that appears on the CRT screen, type the date for the first day of that period. Press the return key.

Step 4. After the last entry for Step 3 is done, the message 'What data do you wish to enter?...' appears on the CRT screen. type 'quit' and press the return key.

Step 5. When '...Stop' appears on the CRT screen, press the return key.

Step 6. Type '*d' and press the return key to clear the program DataPlus out of the computer's memory.

6.2.13 To Create New MS File

To create a new MS data file, perform the following steps:

Step 1. Do Steps 1 to 7 in Section 6.1 if they haven't been done.

Step 2. If Step 1 has already been done, type '*d' and press the return key to clear out any MRP programs in the computer's memory.

Step 3. Type 'g DataPlus2' and press the return key to load the

program DataPlus2 into the computer's memory.

Step 4. Type 'run' and press the return key to start the program executing.

Step 5. When "To enter MS demands..." appears on the CRT screen, type 'MS' and press the return key.

Step 6. When 'new, add, change, or quit MS demands list?' appears, type 'new' and press the return key.

Step 7. When 'part or quit to stop' appears, type the name of the part with the master production schedule demands and press the return key.

Step 8. When 'date, demand or -1, Ø to stop' appears, type the due date and quantity of parts needed. Press the return key.

Note: When no more demands are required for the part entered in Step 7, type '-1, Ø' and press the return key. When no more parts are required to satisfy MS demands, type 'quit' to the message 'part or quit to stop'. Press return key.

Step 9. When 'new, add, change or quit MS demand list' appears, type 'quit' and press the return key.

Step 10. When 'To enter MS demands...' appears on the CRT screen, type 'quit' and press the return key.

Step 11. When '...Stop' appears, press the return key.

Step 12. Type '*d' and press the return key to clear the DataPlus2 out of the computer's memory.

6.2.14 To add master, production schedule demand records

To add records into the MS data file, perform the following steps:

Step 1. Do Steps 1 to 5 in Section 6.2.13.

Step 2. When 'new, add, change, or quit MS demand list?' appears, type 'add' and press the return key.

Step 3. When 'part, date, demand quantity or quit, Ø, Ø to stop' appears on the CRT screen, type the name of the part needed, the due date, and how many. Press the return key. When no more additions are to be placed, type 'quit, Ø, Ø' and press the return key.

Step 4. When 'new, add, change, or quit MS demands list' appears, type 'quit' and press the return key.

Step 5. Do Steps 10 to 12 in Section 6.2.13.

6.2.15 To change master production schedule demand records

To change records in the MS data file, perform the following steps:

Step 1. Do Steps 1 to 5 in Section 6.2.13.

Step 2. When 'new, add, change, or quit MS demand list?' appears, type 'change' and press the return key.

Step 3. When 'part, date, demand quantity to be changed or quit, Ø, Ø to stop' appears, type the information for the MS record to be changed and press the return key.

Step 4. When 'correct date, correct demand quantity' appears type the corrected master production schedule information for the part listed in Step 3. Press the return key.

Note: Continue making the appropriate responses to Steps 3 and 4 until all necessary changes have been made. Then, type 'quit, Ø, Ø' and press

the return key.

Step 5. Do Steps 10 to 12 in Section 6.2.13.

6.2.16 To Create New XO Data File

See Section 6.2.13. Except for typing 'XO' in Step 5, the procedure is the same.

6.2.17 To Add External Order Demand Records

See Section 6.2.14. Except for typing 'XO' when 'To enter MS demands, type MS. To enter external orders, type XO. To enter service parts, type SP. To stop, type quit.' appears on the screen, the procedure is the same.

6.2.18 To Change External Order Demand Records

See Section 6.2.15. Except for typing 'XO' when 'To enter MS...' appears on the screen, the procedure is the same.

6.2.19 To Create New SP Data File

See Section 6.2.13. Except for typing 'SP' in Step 5, the procedure is the same.

6.2.20 To Add Service Part Demand Records

See Section 6.2.14. Except for typing 'SP' when 'To enter MS demands, type MS. To enter external orders, type XO. To enter service parts, type SP. To stop, type quit' appears on the screen, the procedure is the same.

6.2.21 To Change Service Part Demand Records

See Section 6.2.15. Except for typing 'SP' when 'To enter MS...' appears on the screen, the procedure is the same.

6.2.22 To Enter New Lead Times and Lotting Rules

To enter new lead times and lotting rules, perform the following steps:

Step 1. Do Steps 1 to 4 in Section 6.2.6.

Step 2. When 'What data do you wish to enter?...' appears on the CRT screen, type 'ltandlr'. Press the return key.

Step 3. When 'To enter both lead time and lotting rule, type ltlr. To enter just lead time, type lt. To enter just lotting rule, type lr. To stop, type quit'. appears on the screen, type 'ltlr' and press the return key.

Step 4. When 'Leadtime... part, lead time (days), lot rule (L4L, FPQ) or quit, Ø, q to stop' appears on the CRT, type in the part number, its lead time in number of days and either 'L4L' or 'FPQ' for its lotting rule. Press the return key.

Step 5. If 'FPQ' was requested for the part's lotting rule, then 'period (days) for FPQ' will appear on the CRT screen. Type in the number of days for the fixed period and press the return key.

Step 6. When 'To enter both lead time and lotting rule...' appears, type 'quit' and press the return key.

Step 7. Do Steps 10 to 12 in Section 6.2.6.

Note: See Appendix D for additional editing informaion.

7. CONCLUSION

7.1 Implementation of an MRP System

Since MRP System failures are great, the achievement requires a great deal of planning and preparation before implementation. A. J. Forster and E. Y. P. Hsu state "The principal activities are: (1) bringing the bills of material up-to-date, re-structuring them, and converting them; (2) establishing good material flow and control, locked stockrooms; (3) developing a well thought out manufacturing planning system which is translated into an effective, realistic master schedule; and lastly, (4) the most important, the training of the people."[47] Then, once implementation has taken to maintain an ongoing successful MRP system, G. Plossl wrote "...in 1975, I listed the requirements for effective control, calling for less attention to techniques and emphasing the foundations needed for effective systems. Here's what I saw as the five essentials:

1. An effective, integrated system.
2. A realistic master production schedule to drive it.
3. Accurate data on which the system can work.
4. Qualified people to operate the system and to operate the business with the system.
5. An effective organization of these individuals concentrating on executing the plan, not replanning themselves out of trouble."[48]

Many MRP authorities claim that the computer program does not make a successful MRP system. But, it is those people, who implement, maintain and run the MRP system, that determine its success or failure. A. K. Kochhar states "User involvement in all aspects of system development work

is a primary requirement for ensuring the success of any computer application...An important consideration in the implementation of a computer system is the design of supporting manual and clerical procedures..."[49] If the computer system is difficult and cumbersome to work with, the people (MRP Users) will dislike it and will not support the MRP system dooming it to failure. R. W. Hall explains further that "...The real challenge is to make the formal system a tool which can respond to legitimate demands made on it. Otherwise the MRP-based system will go the way of all bureaucratic bungles."[50]

To reiterate a requirement for success, the accuracy of inventory records is essential. An article in Modern Material Handling points out that "While computers and software are pre-requisites for installing MRP, so, too, are accurate records. You must make sure that the data you crank in is both the right kind of data and accurate. Bills-of-material, for example, must be complete and specific down to the last nut and bolt."[51] To insure an error free MRP system, D. A. Devarajan advises that 'A workable system requires that errors are hunted and eliminated. A person or task force specifically responsible for that function will greatly accelerate the establishment of a workable system. However, once the system is in operation, the necessity for such an activity will depend upon several factors that characterize the ability to maintain the system under control. The dynamic nature of the business, product changes, and personnel changes, will dictate the answer."[52]

7.2 Software Capabilities

For maintenance of the data files, software for the MRP explosion has been written keeping clerical ease in mind. This software can interactively work with the MRP User to do the following tasks:

1. Create and add to a parts list with up to 999 parts.
2. Create, add, and change existing up to 9999 bill of material records.
3. Edit easily the lead time information.
4. Choose between lot-for-lot or fixed period quantity lot sizing techniques.
5. Edit easily the on hand inventory quantities.
6. Create, add, delete, and change scheduled receipt data.
7. Set up a planning horizon with either equal buckets (5 days per bucket) or unequal buckets.
8. Create, add, or change the master production schedule information.
9. Create, add, or change the external orders information.
10. Create, add, or change the service parts information.
11. Obtain a computer generated MRP explosion output, which is complete with a MRP matrix for each part, dependent demand sources, purchase orders, etc.

Typically for most MRP systems, the capabilities listed above are only a beginning. The inventory (production) system being modeled will require additional capabilities. Some might include the following: 1. to delete parts from the parts list; 2. to delete bill of materials records; 3. to include additional lot sizing techniques, such as E.O.Q.,

PPB, etc.; and 4. to retrieve information made available by the data files such as total component lists per assembly. (Note: much of the information required by E.O.Q., P.P.B, etc., has been included in the design of the partstatus data file. See Section 3.3.4.)

7.3 The Commodore Computer

Since MRP systems have been successfully maintained on a large computer, the question is whether or not a microcomputer can handle the MRP system. A. K. Kochhar comments that "a large mainframe computer may be used to meet the data processing requirements in a number of different functional areas. The mainframe computer can also collect summary information, required for overall planning and production scheduling, from mini-computers used in individual production lines or departments...

"Mini-computers can be used in an on-line mode, to carry out general purpose data processing applications. For example a mini-computer could be dedicated to the performance of production planning and control work in individual production lines. Such a system can provide up-to-date information for the production controllers and thereby help in the task of management decision making. Similar mini-computers can be used in other functional areas."[53] In other words, possibly the microcomputer can function in the smaller work areas, then a summary is supplied to the big mainframe.

The Commodore Superpet microcomputer was used for the development of the software for this thesis. In using this computer, the main advantages are the Waterloo Fortran and the direct access files maintained by this computer system. The Waterloo Fortran is easy to understand and use;

contains if-then-else statements; and has formatting capabilities. However, some of the disadvantages are memory limitations, poor documentation and storage and retrieval problems.

F. D. Boswell, et. al., state that "The system memory consists of read/write memory (RAM) into which programs and data may be loaded and/or executed... The read/write memory consists of 32768 bytes (32 K) of "standard" memory which is always accessible by a computer program. Additionally, there are 65536 bytes (64 K) of read/write memory which is organized into 16 "banks" of 4096 bytes (4 K) each. At any one time only one bank is accessible by a computer program..."[54] The documentation on how to use the Waterloo Fortran is pretty fair. But, documentation for formatting a disk or operating the printer, etc., is almost non-existent. Mostly, the manuals are written for the Commodore 4.0 Basic, which is different from the Waterloo systems. Thus, the user is left to spend many hours trying to figure out simple editing tasks. In data manipulation the following problems arose with the use of negative values on a direct access record and the misinterpretation of data on records. The first problem was solved by using the comma between each piece of data. The second problem was solved by using additional open and close statements. More open and close statements, however, add time to the execution.

The main problem with the software is the long execution time. Once all the data is entered and the level and levelkey determined for each part, one MRP matrix takes about 10 minutes to produce. If some of the open and close statements were eliminated, possibly the execution time would decrease.

FOOTNOTES

[1]David R. Schuchts, ""Give MRP a Kiss" Keep It Smooth and Simple," Production and Inventory Management, Third Qtr, 1979, p. 69.

[2]Ibid., p. 69.

[3]Urban Wemmerlov, "Design Factors in MRP Systems: A Limited Survey," Production and Inventory and Management, Fourth Qtr, 1979, p. 15.

[4]Joseph Orlicky, Material Requirements Planning (New York: McGraw-Hill Book Company, 1975), p. 189-190.

[5]H. B. Maynard, editor, Industrial Engineering Handbook (New York: McGraw-Hill Book Company, 1971), p. 8-86.

[6]L. S. Shealy, "MRP and the Bottom Line," Production and Inventory Management, Third Qtr, 1979, p. 59.

[7]Orlicky, Material Requirements Planning, p. 22-23.

[8]Robert J. Campbell and Thomas M. Porcano, "The Contribution of Material Requirements Planning (MRP) to Budgeting and Cost Control," Production and Inventory Management, Second Qtr, 1979, p. 63.

[9]Schuchts, p. 69.

[10]Campbell and Porcano, p. 64.

[11]James P. Morgan, "MRP breaks wth past patterns of failure," Purchasing, July 24, 1980, p. 53.

[12]Campbell and Porcano, p. 64.

[13]Joseph A. Orlicky, "Rescheduling With Tomorrow's MRP System," Production and Inventory Management, Second Qtr, 1976, p. 38.

[14]Morgan, p. 70.

[15]Schuchts, p. 70.

Footnotes (continued)

[16]"At Tennant - MRPII gets the 'impossible' answers!", Modern Material Handling, Sept. 1979, p. 86-87.

[17]Schuchts, p. 69.

[18]James H. Greene, Production and Inventory Control Systems and Decisions, (Homewood, IL: Richard D. Irwin, Inc., 1974), p. 249.

[19]John C. Anderson, Roger G. Schroeder, Sharon E. Tupy, and Edna M. White, "Material Requirements Planning Systems: The State of the Art," Production and Inventory Management, Fourth Qtr, 1982, p. 56-58.

[20]George W. Plossl, "MRP Yesterday, Today, and Tomorrow," Production and Inventory Management, Third Qtr, 1980, p. 1.

[21]Ibid.

[22]Ibid.

[23]Orlicky, Plossl, and Wight, "Structuring the Bill of Material for MRP," Production and Inventory Management, Fourth Qtr, 1972, p. 19.

[24]Orlicky, Materials Requirement Planning, p. 3.

[25]Ibid., p. 5.

[26]Ibid., p. 4.

[27]Possl, p. 1.

[28]Dave A. Devarajan, "Care and feeding of MRP," Industrial Engineer, 10, No. 8 (Aug. 1978), p. 22.

[29]Orlicky, Materials Requirement Planning, p.x.

[30]Anderson, et al., p. 65.

[31]Plossl, p. 10.

[32]Donald F. Blumberg, "Factors Affecting the Design of a Successful MRP System," Production and Inventory Management, Fourth Qtr, 1980, p. 51.

Footnotes (continued)

[33]Ibid., p. 57.

[34]Douglas B. Boxx, ""MRP - Selling a Dead Horse" (Or How to Convince Management to Invest in MRP)", Production and Inventory Management, Second Qtr, 1979, p. 51.

[35]Ibid., p. 52.

[36]Campbell and Porcano, p. 63-64.

[37]"At Tennant - MRPII gets the 'impossible' answers!", p. 90.

[38]Orlicky, Material Requirements Planning, p. 47.

[39]Shealy, p. 59.

[40]Ibid., p. 64.

[41]Orlicky, Material Requirements Planning, p. 85.

[42]Ibid., p. 63.

[43]Ibid., p. 93.

[44]Ibid., p. 142.

[45]James C. Herchauer and Roger D. Eck, "Extended MRP Systems For Evaluating Master Schedules and Materials Requirements Plans," Production and Inventory Management, Second Qtr, 1980, p. 55-56.

[46]A. K. Kochhar, Development of Computer-Based Production Systems, (London, England: Edward Arnold (Publishers) Ltd., 1979), p. 64.

[47]A. J. Forster and Enrico Y. P. Hsu, "Material Requirements Planning (MRP) The Fine Line Between Success and Failure," APICS 1976 Conference Proceedings, 1976, p. 429.

[48]Plossl, p. 5.

[49]Kochhar, p. 7.

Footnotes (continued)

[50]Robert W. Hall, "Getting the Commitment of Top Management," Production and Inventory Management, First Qtr, 1977, p. 3.

[51]"The essential MRP tools: a note of caution!," Modern Materials Handling, Jan. 6, 1981, p. 82.

[52]Devarajan, p. 27.

[53]Kochhar, p. 36.

[54]F. D. Boswell, et al., System Overview: Commodore SuperPET (Waterloo, Ontario, Canada: Waterloo Computing Systems Limited), 1981, p. 10.

BIBLIOGRAPHY

Anderson, John C., Roger G. Schroeder, Sharon E. Tupy, and Edna M. White. "Material Requirements Planning Systems: The State of the Art," Production and Inventory Management, Fourth Qtr, 1982, pp. 51-67.

"At Tennant - MRPII gets the 'impossible answers!". Modern Material Handling, Sept. 1979, pp. 88-92.

Blumberg, Donald F. "Factors Affecting the Design of a Successful MRP System," Production and Inventory Management, Fourth Qtr, 1980, pp. 50-62.

Boswell, F. D., T. R. Grove, K. I. McPhee, and J. W. Welch. System Overview: Commodore SuperPET. Waterloo, Ontario, Canada: Waterloo Computing Systems Limited, 1981.

Box, Douglas B. ""MRP - Selling a Dead Horse" (Or How to Convince Management to Invest in MRP)", Production and Inventory Management, Second Qtr, 1979, pp. 49-62.

Campbell, Robert J., and Thomas M. Porcano. "The Contributions of Material Requirements Planning (MRP) to Budgeting and Cost Control," Production and Inventory Management, Second Qtr, 1979, pp. 63-71.

Devarajan, Dave A. "Care and feeding of MRP," Industrial Engineer, 10, No. 8, Aug 1978, pp. 22-27.

"The essential MRP tools: A note of caution!," Modern Material Handling, Jan 6, 1981, p. 82.

Forster, A.J. and Enrico Y. P. Hsu, "Material Requirements Planning (MRP) The Fine Line Between Success and Failure," APICS 1976 Conference Proceedings, pp. 421-446.

Greene, James H. Production and Inventory Control Systems and Decisions. Homewood, IL: Richard D. Irwin, Inc., 1974.

Hall, Robert W. "Getting the Commitment of Top Management," Production and Inventory Management, First Qtr, 1977, pp. 1-9.

Hershauer, James C. and Roger D. Eck. "Extended MRP Systems For Evaluating Master Schedules and Materials Requirement Plans," Production and Inventory Management, Second Qtr, 1980, pp. 53-66.

Kochhar, A. K. Development of Computer-Based Production Systems. London, England: Edward Arnold (Publishers) Ltd, 1979.

Maynard, H. B., editor. Industrial Engineering Handbook. New York: McGraw-Hill Book Company, 1971.

Morgan, James P. "MRP breaks with past patterns of failure," Purchasing, July 24, 1980, pp. 48-54.

Orlicky, Joseph. Material Requirements Planning. New York: McGraw-Hill Book Company, 1975.

Orlicky, Joseph. "Rescheduling with Tomorrow's MRP System," Production and Inventory Management, Second Qtr, 1976, pp. 38-48.

Orlicky, Joseph A., George W. Plossl, and Oliver W. Wight. "Structuring the Bill of Material for MRP," Production and Inventory Management, Fourth Qtr, 1972, pp. 19-42.

Plossl, George W. "MRP Yesterday, Today and Tomorrow," Production and Inventory Management, Third Qtr, 1980, p. 1.

Schuchts, David R. ""Give MRP a Kiss" Keep It Smooth and Simple." Production and Inventory Management, Third Qtr, 1979, pp. 69-80.

Shealy, L. S. "MRP and the Bottom Line." Production and Inventory Management, Third Qtr, 1979, pp. 58-67.

Wemmerlov, Urban. "Design Factors in MRP Systems: A Limited Survey." Production and Inventory Management. Fouth Qtr, 1979, pp. 15-35.

APPENDIX A

PROGRAMS (DEBUGGING)

```
*
*   File: PartFile
*
*   Programmer: Karen M. May    Date: May 11, 1983
*
*   Function: establish and append partid file; initialize
*             partstatus file
*
*   Variables:
*     add - where new additions to partid file start
*     answer - reply to questions
*     down - down directional key for binary search
*     first - part with ordinal number one
*     k,key,keyi - location of next part in ordered list (i=1,2)
*     mid - midpoint; binary search starting point
*     next - next record free for additional parts
*     num - company or user part numbers (character)
*     p - ordinal position of the company part number
*     sys - system or computer part number (integer)
*     up - up directional key for binary search
*
*   Subprograms: setupadd, updown, keycheck, initialize
*
      character answer
*   Check to make sure partid file is to editted.
10    print,'Is part file to be established? or appended? yes,no'
      read,answer
      if answer = 'yes'
        open(unit=2,file='(f:35)partid,rel',access='DIRECT')
        call setupadd
        call initialize
        call updown
        close(unit=2)
      elseif answer = 'no'
        go to 20
      else
        go to 10
      endif
20    stop
      end
*
      subroutine setupadd
*
*   The routine reads company part numbers,assigns system numbers,
*   and establishes sequential keys for company part numbers.
*
*   Variables
*     newkey - pointer for the new part number
*     newnum - company part number to be added to file
*     newsys - system part number for the new company part
*     old - number of records compared
*     oldkey - pointer for 'oldnum' part already in file
*     oldnum - company part number already in file
*     oldsys - system part number belonging to oldnum
*     oldoldkey - pointer for last part used in comparison
*     oldoldnum - company part number used last
*     oldoldsys - system part number used last
*
*   Formats
10203 format(3i3)
```

```
203   format(3i3,8x,'Partid Data File')
10204 format(4i3)
204   format(4i3,5x,'Partid Data File')
20203 format(a10,2(',',i3))
*
      character newnum,oldnum,oldoldnum,answer
      integer first,oldsys,oldoldsys,oldkey,oldoldkey,old,add
*
*  Get initial values depending on new or old file.
*
5     print,'new or add?'
      read, answer
      if answer = 'new'
         first = 1
         next = 1
         mid = 1
         add = 1
         write(2,00204,rec=0)first,next,mid,add
         newsys = 0
         newkey = 999
      elseif answer = 'add'
         read(2,10203,rec=0)first,next,mid
         add = next
         write(2,00204,rec=0)first,next,mid,add
         newsys = next - 1
      else
         goto 5
      endif
loop

*     Type in new number. When no more parts, just type quit.
*
10    print,
      read,newnum
*
*  No more additional parts?
*
      quitif newnum = 'quit'
*
*  Increment system number and next available record number.
*
      newsys = newsys + 1
      next = next + 1
      write(2,00204,rec=0)first,next,mid,add
*
*     When the file only has one part,no comparison is necessary.
*
      if newsys = 1
         write(2,20203,rec=newsys) newnum,newsys,newkey
         goto 10
      endif
*
*     When more than one part,compare and assign key.
*
      old = 0
      key = first
20    read(2,*,rec=key)oldnum,oldsys,oldkey
*
*     Part numbers should not be duplicated.
*
```

```
      if newnum = oldnum
         print,newnum,' has already been used to name a part!!!'
         newsys = newsys - 1
         next = next - 1
         write(2,00204,rec=0)first,next,mid,add
         goto 10
*
*     New part < old part, key new to old part. If old part was
*   first, then new part becomes first. If new part is not first,
*   change key on preceding part.
*
      elseif newnum < oldnum
         newkey = oldsys
         if newkey = first
            first = newsys
            write(2,10204,rec=0)first,next,mid,add
         endif
         if old >= 1
            oldoldkey = newsys
            write(2,20203,rec=oldoldsys)oldoldnum,oldoldsys,
&              oldoldkey
         endif
         write(2,20203,rec=newsys)newnum,newsys,newkey
         goto 10
*
*     New part > old part. If old part is last, new part is now
*   last. Otherwise, continue with next part.
*
      else
         if oldkey = 999
            newkey = oldkey
            oldkey = newsys
            write(2,20203,rec=oldsys)oldnum,oldsys,oldkey
            write(2,20203,rec=newsys)newnum,newsys,newkey
            goto 10
         else
            key = oldkey
            oldoldkey = oldkey
            oldoldsys = oldsys
            oldoldnum = oldnum
            old = old + 1
            goto 20
         endif
      endif
endloop
return
end
*
      subroutine updown
*
*    The up/down keys are assigned for the binary search.
*
*   Variables:
*     dist - absolute distance for step to up/down keys
*     downkey - down direction for the binary tree search
*     j - 1/2 number of calculations per dist value
*     jexp - number of keys assigned by calculation method
*     keyi - sequential pointer for company part (i=1,2)
*     m - locations for j+1 set of calculations; old branches
*     md - locations for j set of calculations; new branches
```

```
*     n - number of records in file left for keying
*     pos - ordinal number of the company part numbers
*     upkey - up direction for the binary tree search
*
      character num
      integer pos(999),first,sys,dist,upkey,downkey,up,down,m(30),
&        p,md(30),add
*  Formats
10204 format(4i3)
204   format(4i3,5x,'Partid Data File')
10203 format(3i3)
203   format(3i3,8x,'Partid Data  File')
20204 format(a10,3(',',i3))
20206 format(a10,5(',',i3))
*
*  Initialize values.
*
      do i = 1,30                                          :mandmd
         m(i) = 0
         md(i) = 0
      enddo                                                :mandmd
      read(2,10204,rec=0)first,next,mid,add
      key = first
*
*  Calculate the ordinal number for each company part number.
*
      do i=1,next-1,1                                      :posnumber
         read(2,*,rec=key)num,sys,key
         pos(i) = sys
         write(2,20204,rec=sys)num,sys,key,i
      enddo                                                :posnumber
*
*  Initialize the up/down keys.
*
      do i = 1,next-1,1                                    :upanddown
         read(2,*,rec=i)num,sys,k,p
         write(2,20206,rec=i)num,sys,k,p,0,0
      enddo                                                :upanddown
*
*   Determine the file's midpoint, which is the starting point
* for the binary search.
*
      n = next - 1
      mid = (n+1)/2
      md(1) = mid
      mid = pos(mid)
*
*  Place location of midpoint on record 0.
*
      write(2,00204,rec=0)first,next,mid,add
*
*  Initialize the exponent of 2.
*
      jexp = 0
*
*   Can more keys be assigned by the calculation method.
*
10    j = 2**jexp
      n = n - (j*2)
      if (n <= 0)                                          :toofew
```

```
         call keycheck(next,pos)
         goto 9999
      endif                                                :toofew
*
*    Calculate absolute distance to the next key.
*
      dist = (md(1) + 1)/2
      if (dist < 1)                                        :toosmall
         call keycheck(next,pos)
        goto 9999
      endif                                                :toosmall
*
*    Calculate up/down keys; mark destinations with -1.
*
      do i = 1,j,1
*
*   Where to locate up/down keys.
*
         upkey = md(i) - dist
         downkey = md(i) + dist
*
*   Make sure values are still in the file's range.
*
20       if (upkey <= 0)                                   :range
            upkey = upkey + 1
            quitif upkey=md(i)                             :range
            goto 20
         endif                                             :range
30       if (downkey >= next)                              :inrange
            downkey = downkey - 1
            quitif downkey =md(i)                          :inrange
            goto 30
         endif                                             :inrange
*
*   Which record number contains upkey?
*
40       key1 = pos(upkey)
         read(2,*,rec=key1)num,sys,k,p,up,down
*
*   Don't key here if algorithm already reaches this part.
*
         if (up > 0 .and. down > 0)                        :allkeyed
            upkey = upkey + 1
            quitif upkey = md(i)                           :allkeyed
            goto 40
         endif                                             :allkeyed
*
*   Which record number contains downkey?
*
50       key2 = pos(downkey)
         read(2,*,rec=key2)num,sys,k,p,up,down
*
*   Don't key here if algorithm already reaches this part.
*
         if (up > 0 .and. down > 0)                        :keyed
            downkey = downkey - 1
            quitif downkey = md(i)                         :keyed
            goto 50
         endif                                             :keyed
*
```

```
*    Store for later calculations.
*
          m(2*i-1) = upkey
          m(2*i) = downkey
*
*    Write up/down keys on root and assign -1 to its branches.
*
          key = pos(md(i))
          upkey = pos(upkey)
          downkey = pos(downkey)
          read(2,*,rec=key)num,sys,k,p
          write(2,20206,rec=key)num,sys,k,p,upkey,downkey
          read(2,*,rec=upkey)num,sys,k,p
          write(2,20206,rec=upkey)num,sys,k,p,-1,-1
          read(2,*,rec=downkey)num,sys,k,p
          write(2,20206,rec=downkey)num,sys,k,p,-1,-1
       enddo
*
*   Store new branches as old branches.
*
       do i=1,j*2,1                                          :storemi
          md(i) = m(i)
       enddo                                                 :storemi
*
*   Increment exponent of j.
*
       jexp = jexp + 1
       goto 10
9999   return
       end
*
       subroutine keycheck(next,pos)
*
*    All parts are checked for up/down keys(i.e.not up=down=0).
*    If not, part is keyed above(if down=-1) or below(if up=-1)?
*    No, then print 'not all records can be keyed' and stop.
*
*   Variables
*     loc - ordinal number of company part number
*     pos - location for parts with specific ordinal numbers
*
       character num
       integer pos(999),upkey,downkey,up,down,sys,p
* Format
20206  format(a10,5(',',i3))
*
*   Check each record to see if reached by a key.
*
       do i = 1,next-1,1                                     :allrecords
          key = i
          read(2,*,rec=key)num,sys,k,loc,upkey,downkey
*
*   Not reached? If upkey and downkey equal 0.
*
          if (upkey = 0 .and. downkey = 0)                   :notkeyed
*
*   If first in queue, check records below for linking.
*
             if (loc = 1) goto 20
*
```

```
*  Where is the record below?
*
            key1 = pos(loc-1)
            read(2,*,rec=key1)num,sys,k,p,up,down
*
*  Can it be used for linking below?
*
            if (down = -1)                                  :checkabove
               write(2,20206,rec=key1)num,sys,k,p,up,i
               read(2,*,rec=key)num,sys,k,p
*
*  Make note that the record can be reached now.
*
               write(2,20206,rec=key)num,sys,k,p,-1,-1
               goto 90
            endif                                           :checkabove
            if (loc = next-1)
*
*  Can't link above and if last in queue can't link below.
*
               print,'not all records can be keyed up and down'
               stop
            endif
*
*  Can it be linked with the above record?
*
20          key2 = pos(loc + 1)
            read(2,*,rec=key2)num,sys,k,p,up,down
            if (up = -1 )                                   :checkbelow
*
*  It can be linked above?
*
               write(2,20206,rec=key2)num,sys,k,p,i,down
               read(2,*,rec=key)num,sys,k,p
               write(2,20206,rec=key)num,sys,k,p,-1,-1
              goto 90
            else                                            :checkbelow
*
*  Link can't be established.
*
               print,'not all records can be keyed'
               stop
            endif                                           :checkbelow
         endif                                              :notkeyed
90    enddo                                                 :allrecords
999   return
      end
*
      subroutine initialize
*
* The following procedure initializes the partstatus file.
*
*  Variables:
*     add - where the new additions start in partid file
*     first - first part of sorted company number
*     mid - midpoint of sorted company numbers
*     next - where the next empty record appears
*
      integer add,first
*
```

```
* How many records need to be initialized?
*
      read(2,10204,rec=0)first,next,mid,add
10204 format(4i3)
*
* Next available on record 0? Initialize the records.
*
      open(unit=3,file='(#:100)partstatus.rel',access='DIRECT')
      write(3,10301,rec=0)next
301   format(i3,8x,'Partstatus Data File')
      do i = add,next-1,1
         write(3,20319,rec=i)0.,-1,-1,-1,-1,5,1,5,100.,1.,1.,2,
&           100.,4,-1,-1,-1,-1,-1
20319    format(f8.0,',',5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      enddo
      close(unit=3)
      return
      end
```

```
*
*  file - BOMfile
*
*  Programmer: Karen M. May     Date: May 20, 1983
*
*  function - create bill of material(bom) file; append bom;
*         correct data in bom
*
*  Variables:
*    answer - reply to  questions
*    BofM - key to first appearance as child in bom file
*    BOM - key to first appearance as parent in bom file
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time used to express carrying cost
*    child,c,ch,cc - components that make up assembly
*    ckey - bom pointer to next parent of the same child
*    csys - system part number for child
*    lead - lead time in days
*    level - part level
*    levelkey - link for MRP flow
*    lotp - period in days for L4L or FPQ
*    lotr - lotting rule for explosion
*    MPSkey - key to master production schedule demands
*    numkids - number of different components for parent
*    oh - on hand inventory quantity
*    parent,p,pa,pp - assembly
*    pkey - bom pointer to next child of the same parent
*    psys - parent's system part number'
*    pu - periodal usage (units/unit of time)
*    pup - time unit for periodal usage
*    qty - quantity of child parts/parent part
*    servpartkey - key to service part demands
*    SR - key to scheduled receipts
*    xorderkey - key to external order demands
*
*  Subprograms: newbom, addbom, findsys, childkeys, changebom,
*    pchange, cchange, qchange
*
      character answer
      open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
10    print,'Does BOM need to be created,appended or existing'
      print,'information altered?type either new,add,or change.'
      read, answer
      if answer = 'new'
         call newbom
         go to 99
      elseif answer = 'add'
         call addbom
         go to 99
     elseif answer = 'change'
         call changebom
         go to 99
      else
         go to 10
     endif
99    close(unit=1)
      close(unit=2)
      close(unit=3)
```

```
      stop
      end
*
      subroutine newbom
*
*  The bom file is created with this subprogram. List parent &
* number of different children; then, children & quantity.
*
*  Variables:
*    BofM - key to first appearance as child in bom file
*    BOM - key to first appearance as parent in bom file
*    child - component part of assembly
*    csys - system number of child part
*    key - record number in bom
*    nextkidkey - point to next child of parent
*    num,numkids - number of different parent components
*    oh - on hand inventory quantity
*    parent - assembly
*    pBOMkey - number of records used in bom file
*    psys - parent's system part number'
*    qty -  quantity of child/parent
*    SR - key to scheduled receipts
*
      character parent,child
      integer pBOMkey,psys,csys,qty,first,ccper,pup,SR,xorderkey,
&        servpartkey
*  Formats
101    format(i4,5x,'Bills-of-Materials')
20105 format (2(i3,','),i4,2(',',i4))
20319 format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&         ',',i1,',',f8.0,',',i1,5(',',i3))
*
*  Initialize
*
      pBOMkey = 1
      key = 0
      psys = 0
      csys = 0
*
*  Number of bom records for this parent part?
*
10    print,'parent,# of children or quit,0 to stop'
      read(5,*) parent,numkids
      if parent = 'quit'                                  :fixckey
*
*  Link partstatus record BofM and bom record ckey.
*
         call childkeys
         go to 9999
     endif                                                :fixckey
*
*  Find system number for the company(user) number.
*
     call findsys(parent,psys)
*
*  Is company part not in the partid?
*
     if (psys=-1) go to 10
*
*  Assign pointer BOM in the partstatus file.
```

```
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=psys)oh,SR,num,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      write(3,20319,rec=psys)oh,SR,numkids,pBOMkey,-1,lead,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
       do i=1,numkids,1                                    :readchild
*
*   Increment record number.
*
         key = key + 1
*
*   Increment key to next child part.
*
         nextkidkey = key + 1
20       print,'child,quantity'
         read(5,*)child,qty
*
*   Find the child's system number.
*
        call findsys(child,csys)
*
*   Isn't system number in partid?
*
         if (csys=-1) go to 20
*
*   Denote the last child by pkey = -1.
*
         if (i=numkids) nextkidkey = -1
         write(1,20105,rec=key)psys,csys,qty,nextkidkey,-1
      enddo                                                :readchild
      pBOMkey = key + 1
      write(1,00101,rec=0) pBOMkey
      go to 10
9999  return
      end
*
      subroutine childkeys
*
*   This procedure assigns ckeys in the bom.
*
*   Variables:
*     BofM - key to first appearance of child in bom file
*     BOM - key to first appearance of parent in bom file
*     c - child part just read on the bom record
*     ch - previous child part read on the bom record
*     i - do loop index for reading bom records
*     ii - do loop index for finding records with same child
*     ke - pkey on record i
*     kee - pkey on record ii
*     key - ckey on record i
*     keys - ckey on record ii
*     ki - last record ii containing same child
*     nc - number of different components
*     oh - on hand inventory quantity
*     p - parent part just read on the bom record
*     pa - previous parent part read on the bom record
```

```
*     q - quantity of child parts/parent part on record i
*     qt - quantity of child parts/parent part on record ii
*     SR - key to scheduled receipts
*
      integer p,c,q,SR,BOM,pa,ch,qt,BofM,ccper,pup,xorderkey,
&        servpartkey
*  Format
10101 format(i4)
20105 format(2(i3,','),i4,2(',',i4))
20319 format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&       ',',i1,',',f8.0,',',i1,5(',',i3))
*
*  Make sure all BofM keys are -1.
*
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file'(f:100)partstatus.rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,nc,BOM,BofM,lead,lotr,lotp,scost,
&           ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&           level,levelkey
         write(3,20319,rec=i)oh,SR,nc,BOM,-1,lead,lotr,lotp,scost,
&           ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&           level,levelkey
         close(unit=3)
      enddo
*
*  How many records are in the bom?
*
      read(1,10101,rec=0)next
      close(unit=3)
      do i=1,next-1,1                                    :startingpoint
      read(1,*,rec=i)p,c,q,ke,key
*
*  Has a link already been created?
*
      if key = -1                                        :lastone
         close(unit=3)
         open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
         read(3,*,rec=c)oh,SR,nc,BOM,BofM,lead,lotr,lotp,scost,
&           ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&           level,levelkey
*
*  Assign BofM in partstatus record.
*
         if BofM = -1                                    :firstone
            write(3,20319,rec=c)oh,SR,nc,BOM,i,lead,lotr,lotp,
&              scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&              level,levelkey
            ki=i
*
*  Other locations where child part appears?
*
            do ii=i+1,next-1,1                           :search
               read(1,*,rec=ii)pa,ch,qt,kee,keys
               if c = ch                                 :lastread
                  write(1,20105,rec=ki)p,c,q,ke,ii
                  ki=ii
                  p = pa
```

```
                  c = ch
                  q = qt
                  ke = kee
                endif                                          :lastread
              enddo                                            :search
           endif                                               :firstone
        endif                                                  :lastone
        enddo                                                  :startingpoint
        return
        end
*
        subroutine findsys(part,sys)
*
*  This algorithm find system part number (sys) for company
* part number(part).
*
*  Variables:
*    down - down branch for binary search
*    first - company part number with ordinal number one
*    k - next record to try in search for 'part'
*    key - directional key for company part number
*    mid - midpoint of company part number
*    next - next available record in partid file
*    part - company (user) number that is sought
*    pos - ordinal number for company part number
*    s - system (computer) part number on record k
*    sys - system number for 'part'
*    u - company (user) number on record k
*    up - up branch for binary search
*
        character part,u
        integer sys,first,s,pos,up,down
*  format
10203 format(3i3)
*
*  Get the scoop on the partid file.
*
     read(2,10203,rec=0)first,next,mid
*
*  Try mid(midpoint) of company part list. if not 'part', where?
*
        k = mid
10      read(2,*,rec=k)u,s,key,pos,up,down
        if part = u                                            :found
           sys = s
           return
        endif                                                  :found
        if part < u                                            :whichdirection
           if up = -1                                          :notinfile
              print,part,' not in partid file'
              sys = -1
              return
           endif                                               :notinfile
           k = up
           go to 10
        else                                                   :whichdirection
           if down = -1                                        :infileornot
              print,part,' not in partid file'
              sys = -1
              return
```

```
         endif                                                  :infileornot
         k = down
         go to 10
      endif                                                     :whichdirection
      return
      end
*
      subroutine addbom
*
*  Additional records are included in bom file.
*
*  Variables:
*    BofM - key to first appearance as child in bom file
*    BOM - key to first appearance as parent in bom file
*    c - child part for new bom record
*    cc - 'c' in do loop to reset ckey
*    cckey - key to next parent of the same child
*    csys - child's system part number
*    next - next available record in file
*    numkids - number of different components in parent
*    oh - on hand inventory quantity
*    p - parent part for new bom record
*    pp - 'p' in do loop to reset ckey
*    psys - parent's system part number
*    ppkey - key to the next child of the same parent
*    q - number of c's/p on new bom record
*    qq - 'q' in do loop to reset ckey
*    SR - key to scheduled receipts
*    where - next empty record
*
      integer q,where,psys,csys,BOM,BofM,SR,pp,cc,qq,ppkey,cckey,
&        add,ccper,pup,xorderkey,servpartkey
      character p,c
*  Format
101    format(i4,5x,'Bills-of-Material')
10101 format(i4)
20319 format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&        ',',i1,',',f8.0,',',i1,5(',',i3))
20105 format(2(i3,','),i4,2(',',i4))
      read(1,10101,rec=0)next
      do i = 1,next-1,1                                         :resetcckey
         read(1,*,rec=i)pp,cc,qq,ppkey,cckey
         write(1,20105,rec=i)pp,cc,qq,ppkey,-1
      enddo                                                     :resetcckey
      where = next - 1
10    print,'parent,child,qty to be added or quit,q,0 to stop'
      read(5,*)p,c,q
      if p = 'quit'                                             :nomore
         next = where + 1
         write(1,10101,rec=0)next
         call childkeys
         go to 999
      endif                                                     :nomore
*
*  Find system numbers.
*
      call findsys(p,psys)
      call findsys(c,csys)
*
*  Can't find one  of the system numbers?
```

```
*
      if (psys = -1.or.csys = -1) go to 10
*
*   Next empty record?
*
      where = where + 1
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=psys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*   New parent? Yes, if numkids = -1.
*
      if (numkids <> -1) numkids = numkids + 1
      if (numkids = -1) numkids = 1
*   Where to find the other children?
      if BOM = -1                                          :previous listing
         write(3,20319,rec=psys)oh,SR,numkids,where,BofM,lead,
&           lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         write(1,20105,rec=where)psys,csys,q,-1,-1
         go to 10
      else                                                 :previous listing
         write(3,20319,rec=psys)oh,SR,numkids,BOM,BofM,lead,
&           lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         l = BOM
20       read(1,*,rec=l)pp,cc,qq,ppkey,cckey
         if ppkey = -1                                     :last child
            write(1,20105,rec=l)pp,cc,qq,where,cckey
            write(1,20105,rec=where)psys,csys,q,-1,-1
            go to 10
         else                                              :last child
            BOM = BOM + 1
            l = ppkey
            go to 20
         endif                                             :last child
      endif                                                :previous listing
999   return
      end
*
      subroutine changebom
*
*   This routine directs the changes made to the bom file.
*
*   Variables:
*     find - flag for old pair found or not; yes=1 and no=0
*     newc - new child part for the bom record
*     newcsys - system part number for newc
*     newp - new parent part for the bom record
*     newpsys - system part number for newp
*     newq - quantity for number of newc/newp
*     oldc - old child part on bom record
*     oldcsys - system part number for oldc
*     oldp - old parent part on bom record
*     oldpsys - system part number for oldp
*     oldq - quantity for number of oldc/oldp
*
      character oldp,oldc,newp,newc
```

```
      integer oldpsys,oldcsys,oldq,find
*
*  What record is to be changed?
*
10    print,'Enter parent,child,qty as it now appears'
      print,'or quit,q,0 to stop.'
      read(5,*)oldp,oldc,oldq
*
*  No more corrections?
*
      if (oldp = 'quit') go to 999
*
*  Search for system numbers.
*
      call findsys (oldp,oldpsys)
      call findsys (oldc,oldcsys)
*
*  Can't find one of the system numbers?
*
      if (oldpsys = -1.or.oldcsys = -1) go to 10
*
*  How to change the bom record?
*
20    print,'Enter parent,child,qty as it should appear'
      print,'or quit,q,0 to stop.'
      read(5,*)newp,newc,newq
*
*  Old material entered wrong?
*
      if (newp = 'quit') go to 10
*
*  Search for system numbers.
*
      call findsys(newp,newpsys)
      call findsys(newc,newcsys)
      if (newpsys = -1.or.newcsys = -1) go to 20
*
*  The parent,child pair is in the bom file?
*
      find = 1
*
*  Change parent part?
*
      if(oldp <> newp) call pchange(oldpsys,oldcsys,newpsys,find)
      if (find=0) go to 10
      find = 1
*
*  Change child part?
*
      if (oldc <> newc) call cchange(newpsys,oldcsys,newcsys,find)
      if (find=0) go to 10
      find = 1
*
*  Change quantity child per parent?
*
      if (oldq <> newq) call qchange(newpsys,newcsys,newq,find)
      go to 10
999   return
      end
*
```

```
      subroutine pchange(oldpsys,oldcsys,newpsys,find)   366
*
*   The parent part is changed with this subprogram.
*
*   Variables:
*     BofM - key to first appearance as parent in bom file
*     BOM - key to first appearance as child in bom file
*     c - child on bom record for comparison in search
*     ckey - key to same child of different parent
*     c1 - child on last bom record read
*     ckey1 - ckey on last bom record read
*     find - flag for pair found or not; 1 and 0, respectively
*     key - record number of read
*     key1 - key on last bom record read
*     n - number of reads to locate the old child/parent pair
*     newpsys - system part number of newp
*     numkids - number of different components in parent
*     oh - on hand inventory quantity
*     oldcsys - system part  number of oldc
*     oldpsys - system part  number of oldp
*     p - parent part on the bom file record
*     pkey - key to next child of the same parent
*     pkey1 - pkey on last bom record read
*     q - number of c's/p
*     q1 - Number of c1's/p'
*     SR - pointer to scheduled receipts
*
      integer SR,BOM,BofM,ccper,pup,p,c,q,pkey,ckey,c1,q1,pkey1,
&        oldpsys,oldcsys,oldq,find,ckey1,xorderkey,servpartkey
*   format
20105    format(2(i3,','),i4,2(',',i4))
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*
*   Find old parent,child pair in the bom file.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=oldpsys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*   How many reads were needed to find pair.
*
      n = 0
*
*   BOM is first bom with parent. If BOM=-1, no children.
*
      if BOM = -1
         find = 0
         print,'old parent,child pair is not in BOM file'
         go to 999
      endif
      key = BOM
30    read(1,*,rec=key)p,c,q,pkey,ckey
      if oldcsys <> c                                    :findchild
         if pkey = -1                                    :infileornot
            find = 0
            print,'parent,child pair not in BOM file'
            go to 999
         else                                            :infileornot
```

```
            n = n + 1
            key1 = key
            key = pkey
            c1 = c
            q1 = q
            pkey1 = -1
            ckey1 = ckey
            go to 30
         endif                                              :infileornot
      endif                                                 :findchild
*
*  Found parent,child pair. Old parent = New parent.
*
      numkids = numkids - 1
      if (numkids = 0) numkids = -1
      write(3,20319,rec=oldpsys)oh,SR,numkids,BOM,BofM,lead,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if (pkey<>-1) pkey1 = pkey
      if (n>0) write(1,20105,rec=key1)p,c1,q1,pkey1,ckey1
*
*  Let the child be last child for the new parent.
*
      write(1,20105,rec=key)newpsys,c,q,-1,ckey
      read(3,*,rec=newpsys)oh,SR,numkids,BOM,BofM,lead,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if BOM = -1                                           :previouslisting
         write(3,20319,rec=newpsys)oh,SR,1,key,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         go to 999
      else                                                  :previouslisting
         numkids = numkids + 1
         write(3,20319,rec=newpsys)oh,SR,numkids,BOM,BofM,lead,
&           lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         k = BOM
*
*  Link the last child to other new parent children.
*
40       read(1,*,rec=k)p,c,q,pkey,ckey
         if pkey = -1                                       :lastchild
            write(1,20105,rec=k)p,c,q,key,ckey
            go to 999
        else                                               :lastchild
            k = pkey
            go to 40
         endif                                              :lastchild
      endif                                                 :previouslisting
999   return
      end
*
      subroutine cchange(newpsys,oldcsys,newcsys,find)
*
*  The child part in the bom is changed with this subprogram.
*
*  Variables:
*    BofM - key to first appearance as child in bom file
*    BOM - key to first appearance as parent in bom file
```

```
*     c - child part on bom record in old pair search
*     ckey - pointer to next parent of same child
*     find - flag for pair found or not; 1 or 0, respectively
*     newcsys - system part number for newc
*     newpsys - system part number for newp
*     numkids - number of different components in parent
*     oh - on hand inventory quantity
*     oldcsys - system part number for oldc
*     p - parent part on bom record in old pair search
*     pkey - pointer to next child of same parent
*     q - number of children/parent in old pair search
*     SR - pointer to scheduled receipts
*
      integer SR,BOM,BofM,ccper,pup,p,c,q,pkey,ckey,find,oldcsys,
&        oldq,xorderkey, servpartkey
*  Format
10101    format(i4)
10301    format(i3)
20105    format(2(i3,','),i4,2(',',i4))
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
*
*  Find parent,child pair.
*
      read(3,*,rec=newpsys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        level,levelkey
      if BOM = -1
         find = 0
         print,'old parent,child pair is not in bom file'
         go to 999
      endif
      key = BOM
50    read(1,*,rec=key)p,c,q,pkey,ckey
*
*  Child found? No more children if pkey = -1.
*
      if  oldcsys <> c                                    :findoldchild
         if pkey = -1                                     :infileornot
            find = 0
            print,'old parent,child pair is not in BOM file'
            go to 999
         else                                             :infileornot
            key = pkey
            go to 50
         endif                                            :infileornot
      endif                                               :findoldchild
*
*  Pair has been found.
*
      write(1,20105,rec=key)p,newcsys,q,pkey,ckey
*
*  Reset ckey to -1 before calling childkeys.
*
      read(1,10101,rec=0) next
      do i = 1,next-1,1                                   :resetckey
         read(1,*,rec=i)p,c,q,pkey,ckey
         write(1,20105,rec=i)p,c,q,pkey,-1
```

```
      enddo                                                  :resetckey
      call childkeys
999   return
      end
*
      subroutine qchange(newpsys,newcsys,newq,find)
*
*  This algorithm corrects the quantity on bom record.
*
*  Variables:
*    BOM - key to first appearance as parent in bom file
*    c - child part read on bom record
*    ckey - key to same child of different parent
*    find - flag for pair found or not; 1 and 0, respectively
*    key - record read in bom file
*    newpsys - system part number of newp
*    newq - new number of newc/newp
*    numkids - number of different children/parent
*    oh - on hand inventory quantity
*    p - parent part read on bom record
*    pkey - key to next child of same parent
*    q - number of c's/p'
*    SR - pointer to scheduled receipts
*
      integer find,SR,BOM,p,c,q,pkey,ckey
*  format
20105    format(2(i3,','),i4,2(',',i4))
*
*  Find the old parent,child pair.
*
      read(3,*,rec=newpsys)oh,SR,numkids,BOM
      if BOM = -1
         find = 0
         print,'old parent,child pair is not in BOM file'
         go to 999
      endif
      key = BOM
10    read(1,*,rec=key)p,c,q,pkey,ckey
*
*   When pair is found, change the quantity.
*
      if newcsys = c                                         :findchild
         write(1,20105,rec=key)newpsys,newcsys,newq,pkey,ckey
         go to 999
      else                                                   :findchild
         if pkey = -1                                        :infileornot
            find = 0
            print,'old parent,child pair not in BOM file'
            go to 999
         else                                                :infileornot
            key = pkey
            go to 10
         endif                                               :infileornot
      endif                                                  :findchild
999   return
      end
```

```
*
*  File - DataPlus
*
*  Karen M. May      Date: May 31, 1983
*
*  Function - to edit lead time, lotting rules, and dates
* for the planning horizon
*
*  Variables:
*    answer - response to questions
*    BofM - key to first appearance as child part
*    BOM - key to first appearance as parent part
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time for expressing carrying cost
*    fillin - empty records from 1-next
*    lead - lead time (in days)
*    level - part level
*    levelkey - link for MRP flow
*    lotp - time in days for L4L or FPQ
*    lotr - lotting rule for explosion
*    MPSkey - key to master production schedule demands
*    next - next free record
*    numkids - number of different components in parent
*    oh - on hand inventory
*    pu - periodal usage(units/unit of time)
*    pup - time unit for periodal usage
*    servpartkey - key to service parts demands
*    SR - key to scheduled receipts
*    xorderkey - key to external orders
*
*  Subprograms: findsys,IandSR,ISRnew,srchange,dayorqty,delSR,
* addSR,ohchange,leadandlot,leadtimeandlotrule,leadtime,lotrule,
* horizon,equalbucket,unequalbucket
*
      character answer
10    print,'What data do you wish to enter?'
      print,'Inventory and/or scheduled receipts,type invandsr.'
      print,'Lead time and lotting rules,type ltandlr.'
      print,'Periods for the planning horizon, type ph.'
      print,'To stop, type quit.'
      read, answer
      if answer = 'invandsr'
         call IandSR
         go to 10
      elseif answer = 'ltandlr'
         call leadandlot
         go to 10
      elseif answer =  'ph'
         call horizon
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   stop
      end
*
      subroutine findsys(part,sys)
*
```

```
*  This algorithm finds the system number (sys) for the company
* number (part).
*
*  Variables:
*    down - down branch for binary search
*    first - company part number with ordinal number one
*    k - next record to try in search for 'part'
*    key - directional key for company part number
*    mid - midpoint of company part number
*    next - next available record in partid file
*    part - company (user) number that is sought
*    pos - ordinal position for the company number
*    s - system (computer) part number on record k
*    sys - system number for 'part'
*    u - the company (user) number on record k
*    up - upkey for binary search
*
      character part,u
      integer sys,first,s,pos,up,down
*
*  Format
*
10203   format(3i3)
*
*  Check statistics on partid file
*
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10203,rec=0)first,next,mid
*
*  Try midpoint first
*
      k = mid
10    read(2,*,rec=k)u,s,key,pos,up,down"
      if part = u                                          :found
         sys = s
         close(unit=2)
         return
      endif                                                :found
      if part < u                                          :whichdirection
         if up = -1                                        :notinfile
           print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :notinfile
         k = up
         go to 10
      else                                                 :whichdirection
        if down = -1                                      :infileornot
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                            :infileornot
         k = down
         go to 10
      endif                                               :whichdirection
      close(unit=2)
      return
      end
```

```
*
      subroutine IandSR
*
*  The user specifies what editing is to be done.
*
*  Variable: answer - response to questions
*
       character answer
10    print,'To create SR file and initialize oh hand inventory,'
      print,'type new. To change SR file data,type SR.'
      print,'To change on hand inventory quantities,type onhand.'
      print,'To stop,type quit.'
      read,answer
      if answer = 'new'
         call ISRnew
         go to 10
      elseif answer = 'SR'
         call srchange
         go to 10
      elseif answer = 'onhand'
         call ohchange
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
      subroutine ISRnew
*
*  Both oh(on hand inventory) and SR(scheduled receipts) are
* entered.
*
*  Variables:
*    day - expected due date of the order
*    fillin - empty records from 1-next
*    inv - amount of inventory in stockroom
*    next - next free record in partstatus file
*    oh - on hand inventory; reset to 0
*    part - company part ordered
*    qty - order quantity
*    sr - number of scheduled receipts for part
*    SR - key to scheduled receipts; reset to -1
*    SRkey - location of SRs for 'part'
*
      character part,answer
      integer sr,oh,SR,BOM,BofM,ccper,pup,day,SRkey,sys,fillin,
&        xorderkey,servpartkey
      real inv
*  Formats
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
20403    format(i4,',',f8.0,',',i4)
402      format(2i4,5x,'SR')
10402    format(2i4)
10301    format(i3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=4,file='(f:18)SR,rel',access='DIRECT',recl=18)
```

```
      read(3,10301,rec=0)next
      close(unit=3)
*
*   Set (reset) oh and SR to default values.
*
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)0.,-1,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         close(unit=3)
      enddo
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
*
*   Keep track of empty records between 1 and next.
*
      fillin = 0
*
*   Where can the new SR be written?
*
      SRkey = 0
20    print,'part,inventory quantity,# of SR or quit,0,0 to stop'
      read(5,*)part,inv,sr
*
*   No more?
*
      if (part = 'quit') go to 999
*
*   Find system number.
*
      call findsys(part,sys)
*
*   Part not in partid file?
*
      if (sys = -1) go to 20
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
*
*   No scheduled receipts expected?
*
      if sr = 0                                              :SRornot
         write(3,20319,rec=sys)inv,-1,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         go to 20
      else                                                   :SRornot
         do i = 1,sr,1                                       :writeSR
            SRkey = SRkey + 1
*
*   Link partstatus to SR file.
*
            if(i=1) write(3,20319,rec=sys)inv,SRkey,numkids,BOM,
&              BofM,lead,lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&              MPSkey,xorderkey,servpartkey,level,levelkey
```

```
*
*  Get the specifics on the SR.
*
           print,'day expected,quantity expected'
           read(5,*) day,qty
*
*  Link all SR for given part together.
*
           key = SRkey + 1
           if (i=sr) key = -1
           write(4,20403,rec=SRkey)day,qty,key
           next = SRkey + 1
*
*  List the next available record for SR file.
*
           write(4,00402,rec=0)next,fillin

        enddo                                                :writeSR
        go to 20
     endif                                                   :SRornot
999  close(unit=3)
     close(unit=4)
     return
     end
*
     subroutine srchange
*
*  Changes for the SR file are specified.
*
*  Variables: answer - reply to the questions
*
     character answer
10   print,'To change record in SR file,type change.'
     print,'To delete record in SR file,type delete.'
     print,'To include new records in SR,type add.'
     print,'To stop,type quit.'
     read,answer
     if answer = 'change'
        call dayorqty
        go to 10
     elseif answer = 'delete'
        call delSR
        go to 10
     elseif answer = 'add'
       call addSR
        go to 10
     elseif answer = 'quit'
        go to 999
     else
       go to 10
     endif
999  return
     end
*
     subroutine dayorqty
*
*  The quantity &/or expected due date are changed.
*
*  Variables:
*    day - date on record k
```

```
*     k - record read in search of old SR
*     key - next SR for the part given
*     newday - new date for oldday
*     newqty - new quantity for oldqty
*     oh - on hand inventory quantity
*     oldday - expected due date to be changed
*     oldqty - quantity ordered to be changed
*     part - company part with SR to be changed
*     qty - quantity on record k
*     SR - key to scheduled receipts
*     sys - system number for part
*
      character part
      integer oldday,SR,BOM,BofM,sys
      real newqty
*  Formats
20403    format(i4,',',f8.0,',',i4)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=4,file='(f:18)SR,rel',access='DIRECT',recl=18)
*
*  Which part has SR to change?
*
10    print,'part to be changed or quit to stop'
      read,part
*
*  No more changes?
*
      if (part = 'quit') go to 999
*
*  Find system number.
*
     call findsys(part,sys)
*
*  Can't find part?
*
      if (sys=-1) go to 10
*
*  What SR needs changing?
*
      print,'day,qty of SR as it now appears'
      read(5,*) oldday,oldqty
*
*  How should SR be changed?
*
      print,'day,qty of SR as it should appear'
      read(5,*) newday,newqty
*
*  Find the record to be changed.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR
      if (SR = -1)
*
*  If SR = -1, there are no SR's for the given part.
*
         print,'old SR not in file'
         go to 10
      endif
      k = SR
```

```
*
*  Find old SR?
*
20    read(4,*,rec=k)day,qty,key
*
*  Make the change.
*
      if day = oldday .and. qty = oldqty
         write(4,20403,rec=k) newday,newqty,key
         go to 10
      else
         if key = -1
*
*  When key = -1, no more records for part.
*
            print,'old SR not in file'
            go to 10
         else
*
*  Try record.
*
            k = key
            go to 20
         endif
      endif
999   close(unit=3)
      close(unit=4)
      return
      end
*
      subroutine delSR
*
*  SR are deleted with this subprogram.
*
*  Variables:
*     day - date on record k
*     d1 - date for SR on record kf
*     day1 - date on record k1
*     fillin - empty records between 1-next
*     k - record read in search for order to delete
*     key - next SR for given part
*     kf - key on last fillin read
*     k1 - key to next fillin
*     n - number of records read to find SR
*     oh - on hand inventory quantity
*     oldday - date on deleted order (SR)
*     oldqty - quantity on deleted order (SR)
*     part - company part to delete
*     qty - quantity on record k
*     q1 - quantity on record kf
*     qty1 - quantity on record k1
*     SR - key to scheduled receipts for part
*     sys - system number for part
*
      character part
      integer oldday,SR,BOM,BofM,ccper,pup,day,day1,fillin,d1,sys,
&        xorderkey,servpartkey
*  Formats
402      format(2i4,5x,'SR')
10402    format(2i4)
```

```
20403   format(i4,',',f8.0,',',i4)
20319   format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&          ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      open(unit=4,file='(f:18)SR.rel',access='DIRECT',recl=18)
*
*  What part has the SR to be deleted?
*
10    print,' part to be deleted or quit to stop'
      read,part
*
*  No more SRs to be deleted?
*
     if (part = 'quit') go to 999
*
*  Find system number.
*
     call findsys(part,sys)
*
*  Can't find part on file?
*
      if (sys = -1) go to 10
      print,'day,qty as it now appears in SR file'
*
*  Which SR is deleted?
*
      read(5,*)oldday,oldqty
*
*   Find SR to be deleted.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper, pu,pup,MPSkey,xorderkey,servpartkey,
&         level,levelkey
*
*  Number of records read to find SR.
*
      n = 0
      if  SR = -1                                          :anyonfile
*
*  If SR = -1, there aren't any SR on file.
*
         print,'old SR not in file'
         go to 10
      endif                                                :anyonfile
      k = SR
*
*  Find old SR.
*
20    read(4,*,rec=k)day,qty,key
      if day = oldday .and. qty = oldqty                   :findoldSR
         write(4,20403,rec=k)0,0.,-1
*
*  Link last record to next one in the file.
*
         if (n > 0) write(4,20403,rec=k1) day1,qty1,key
*
*  Delete first SR? Key partstatus record to second record.
*
```

```
        if (n = 0) write(3,20319,rec=sys)oh,key,numkids,BOM,BofM,
&          lead,lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&          xorderkey,servpartkey,level,levelkey
        read(4,10402,rec=0)next,fillin
*
*  Is there an empty record?
*
        if fillin = 0                                      :otherempties
*
*  Where is first empty record.
*
           write(4,00402,rec=0)next,k
           go to 10
        else                                               :otherempties
          kf = fillin
*
*  Link empty records together.
*
30         read(4,*,rec=kf)d1,q1,k1
           if k1 = -1                                      :anymore
              write(4,20403,rec=kf)d1,q1,k
              go to 10
           else                                            :anymore
              kf = k1
              go to 30
           endif                                           :anymore
       endif                                              :otherempties
     else                                                  :findoldSR
*
*  What record to check next?
*
        if key = -1                                        :more
           print,'old SR not in file'
           go to 10
        else                                               :more
           k1 = k
           k = key
           day1 = day
           qty1 = qty
           n = n + 1
           go to 20
        endif                                              :more
     endif                                                 :findoldSR
999  close(unit=3)
     close(unit=4)
     return
     end
*
     subroutine addSR
*
*  As new SR are made, they can be included in the SR,rel file
* with this routine.
*
*  Variables:
*    d - date on record fillin
*    d1 - date on record k
*    fillin - link to empty records in SR file
*    k - record read to find last SR; next fillin record
*    key - link between last SR and new SR
*    k1 - next SR for given part
```

```
*     newday - date order expected
*     newqty - quantity being ordered
*     next - next available record in SR file
*     oh - on hand inventory
*     part - company part ordered
*     q - quantity on record fillin
*     q1 - quantity on record k
*     SR - key to scheduled receipts
*     sys - system number for part
*
      character part
      integer fillin,SR,BOM,BofM,ccper,pup,d1,sys,xorderkey,
&        servpartkey
      real newqty
*  Formats
402      format(2i4,5x,'SR')
10402    format(2i4)
20403    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      open(unit=4,file='(f:18)SR.rel',access='DIRECT',recl=18)
*
*  Part ordered?
*
10    print,' part to be added or quit to stop'
      read, part
*
*  No more orders?
*
      if (part = 'quit') go to 999
*
*  Get system number.
*
      call findsys(part,sys)
*
*  Can't find the part?
*
      if (sys = -1) go to 10
*
*  New order.
*
      print, ' day, qty '
      read(5,*) newday,newqty
*
*  Find the empty records.
*
      read(4,10402,rec=0)next,fillin
*
*  Other SR's.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      if fillin = 0                                    :empties
         key = next
         write(4,20403,rec=key)newday,newqty,-1
         next = next + 1
```

```
        write(4,00402,rec=0)next,fillin
*
*  Link new SR to old SRs.
*
        if SR = -1                                          :listed
           write(3,20319,rec=sys)oh,key,numkids,BOM,BofM,lead,
&              lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&              xorderkey,servpartkey,level,levelkey
           go to 10
        else                                                :listed
           k = SR
20         read(4,*,rec=k)d1,q1,k1
           if k1 = -1                                       :anymore
              write(4,20403,rec=k)d1,q1,key
              go to 10
           else                                             :anymore
              k = k1
              go to 20
           endif                                            :anymore
        endif                                               :listed
     else                                                  :empties
        read(4,*,rec=fillin)d,q,k
        write(4,20403,rec=fillin)newday,newqty,-1
*
*  Aren't any more empty records?
*
        if (k = -1) k = 0
        write(4,00402,rec=0)next,k
*
*  No other SRs if SR = -1.
*
        if SR = -1                                          :any listed
           write(3,20319,rec=sys)oh,fillin,numkids,BOM,BofM,lead,
&              lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&              xorderkey,servpartkey,level,levelkey
           go to 10
        else                                                :any listed
           k = SR
30         read(4,*,rec=k)d1,q1,k1
           if k1 = -1                                       :more
              write(4,20403,rec=k)d1,q1,fillin
              go to 10
           else                                             :more
              k = k1
              go to 30
           endif                                            :more
        endif                                               :any listed
     endif                                                  :empties
999  close(unit=3)
     close(unit=4)
     return
     end
*
     subroutine ohchange
*
* The oh value is corrected with the following routine.
*
*  Variables:
*    oh - old inventory quantity
*    part - company number for oh
```

```
*     qty - corrected inventory quantity
*     SR - key to scheduled receipts
*     sys - system number for part
*
      character part
      integer SR,BOM,BofM,ccper,pup,sys,xorderkey,servpartkey
*
*   Formats
*
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
*
*   What record needs to be changed? To what quantity?
*
10    print,'part,on hand quantity or quit,0 '
      read(5,*) part,qty
*
*   No more?
*
      if (part = 'quit') go to 999
*
*   Find system number.
*
      call findsys(part,sys)
*
*   Can't find the part?
*
      if (sys  = -1) go to 10
*
*   Change only oh(on hand inventory).
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      write(3,20319,rec=sys)qty,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      go to 10
999   close(unit=3)
      return
      end
*
      subroutine leadandlot
*
*   The type of lead time and lot rule editing is specified.
*
*   Variable: answer - reply to question
*
      character answer
10    print,'To enter both lead time and lotting rule,type ltlr.'
      print,'To enter just lead time, type lt.'
      print,'To enter just lotting rule, type lr.'
      print,'To stop, type quit.'
      read, answer
      if answer = 'ltlr'
         call leadtimeandlotrule
         go to 10
```

```
      elseif answer = 'lt'
         call leadtime
         go to 10
      elseif answer = 'lr'
         call lotrule
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
       subroutine leadtimeandlotrule
*
*   The lead time and lotting rules are supplied using routine.
*
*   Variables:
*     Leadtime - time to produce part (days)
*     lotp - period length for L4L or FPQ (days)
*     lotr - lotting rule code number
*     Lotrule - rule for lot size; character type
*     part - company part id; character
*     sys - system number assigned to the part
*
      character part,lotrule,timeunit,timeperiod
      integer SR, BOM,BofM,ccper,pup,sys,xorderkey,servpartkey
*
*   format
*
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      print,'Leadtime was initialized at 5 days.'
      print,'Lotting rule was initialized at L4L(lot-for-lot).'
10    print,'part,lead time(days), lot rule(L4L,FPQ)'
      print,' or quit,0,q to stop.'
      read(5,*) part,leadtime,lotrule
*
*   No more?
*
      if (part='quit') go to 999
*
*   Find system number.
*
      call findsys(part,sys)
*
*   Part not on file?
*
      if (sys = -1) go to 10
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      lead = leadtime
*
*   Need additional information for lotting rule.
*
```

```
      if lotrule = 'L4L'
         lotr = 1
         lotp = 5
      elseif lotrule = 'FPQ'
         lotr = 2
         print,'period(days) for FPQ'
         read,lotp
      else
         go to 10
      endif
*
*  List data on partstatus record.
*
      write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      go to 10
999   close(unit=3)
      return
      end
*
      subroutine leadtime
*
*  The lead times are edited with the following program.
*
*   Variables:
*    Leadtime - lead time in days
*    part - company part number
*    sys - system number for part
*
      character part
      integer sys,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey
*  format
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      print,'Lead time was initialized at 5 days.'
10    print,'part,lead time(days), or quit,0 to stop.'
      read(5,*)part,Leadtime
*
*  No more information?
*
      if (part = 'quit') go to 999
*
*  Find system number.
*
      call findsys(part,sys)
*
*  Part isn't on file?'
*
      if (sys=-1) go to 10
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,Leadtime,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      go to 10
```

```
999   close(unit=3)
      return
      end
*
      subroutine lotrule
*
*  The lotting rules are edited with the following subprogram.
*
*  Variables:
*     lotp - period length for L4L or FPQ
*     lotr - lotting rule code number
*     Lotrule - lot size rule;character
*     part - company part number
*     sys - system number for part
*
      character part,Lotrule,timeunit,timeperiod
      integer sys,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey
*
*  format
*
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      print,'Lotting rule was initialized at L4L(lot-for-lot).'
10    print,'part,lotrule(L4L,FPQ) or quit,q to stop.'
      read(5,*) part,Lotrule
*
*  No more addition information?
*
      if (part='quit') go to 999
*
*  Fetch system number.
*
      call findsys(part,sys)
*
*  Part isn't on file?'
*
      if (sys = -1) go to 10
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
*
*  Need additional information for the lotting rule.
*
      if Lotrule = 'L4L'
         lotr = 1
         lotp = 5
      elseif Lotrule = 'FPQ'
         lotr = 2
         print,'period(days) for FPQ'
         read,lotp
      else
         go to 10
      endif
*
*  List information on partstatus record.
*
      write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
```

```
&       scost,ucost,ccost,coper,pu,pup,MPSkey,xorderkey,
&       servpartkey,level,levelkey
      go to 10
999   return
      end
*
      subroutine horizon
*
*  The dates for the planning horizon are entered.
*
*  Variables
*    answer - reply to the questions
*    buckets - number of periods in planning horizon
*    nperiod - number of current period
*
      character answer
      integer buckets
*
*  Specify number of buckets(periods) in the planning horizon.
*
      print,'number of buckets?'
      read,buckets
*
*  Specify the current period.
*
      print,'Start with period number.'
      read,nperiod
*
*  Choice of equal (1 week/bucket) or unequal bucket sizes.
*
10    print,'Equal bucket size? yes or no.'
      read, answer
      if answer  = 'yes'
         call equalbucket(buckets,nperiod)
      elseif answer = 'no'
         call unequalbucket(buckets,nperiod)
      else
         go to 10
      endif
      return
      end
*
      subroutine equalbucket(buckets,nperiod)
*
*  The computer assigns to each period(1 week/bucket).
*
*  Variables:
*    buckets - number of periods in planning horizon
*    dayone - date(first day) of the bucket; from shop calendar
*    nperiod - period number
*
      integer buckets,dayone
*
*  Formats
*
701      format(i3,5x,'Planning Horizon')
20702    format(i3,',',i4)
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
*
*  List number of periods in planning horizon.
```

```
*
      write(7,00701,rec=0) buckets
      nperiod = nperiod -1
*
*   Assign day one to each bucket (1 week = 5 days).
*
      do i = 1,buckets,1
         nperiod = nperiod + 1
         dayone = nperiod
         write(7,20702,rec=i) nperiod,dayone
      enddo
      close(unit=7)
      return
      end
*
      subroutine unequalbucket(buckets,nperiod)
*
*   The computer assigns dates as spedifiecd by the user.
*
*   Variables:
*     buckets - number of periods in planning horizon
*     dayone - date(first day) of the bucket; from shop calendar
*     nperiod - period number
*
      integer buckets,dayone
*   Formats
701       format(i3,5x,'Planning Horizon')
20702     format(i3,',',i4)
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
*
*   List number of periods in planning horizon.
*
      write(7,00701,rec=0) buckets
      nperiod = nperiod - 1
*
*   Ask for date (day one) for each bucket.
*
      do i = 1,buckets,1
         nperiod = nperiod + 1
         print,'Period ',nperiod,' day one is ____'
         read, dayone
         write(7,20702,rec=i)nperiod,dayone
      enddo
      close(unit=7)
      return
      end
*
```

```
*
*  file - DataPlus2
*
*  Karen M. May     Date: June 1, 1983
*
*  function - to edit MPS,XO, and SP demands
*
*  Variables:
*    answer - response to inquiries
*    BofM - key to first appearance as child part
*    BOM - key to first appearance as parent part
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time for carrying cost
*    fillin - empty lines between record 1 and next
*    lead - lead time (in days)
*    lotp - time in days for L4L or FPQ
*    lotr - lotting rule
*    next - next free record in the file
*    numkids - number of different components in parent
*    oh - on hand inventory
*    pu - periodal usage(units/unit of time)
*    pup - time unit for periodal usage
*    SR - key to scheduled receipts
*
*  Subprograms: findsys,MSdemands,MSnew,MSadd,MSchange,XOdemands,
* XOnew,XOadd,XOchange,SPdemands,SPnew,SPadd,SPchange
*
      character answer
10    print,'To enter MS demands, type MS.'
      print,'To enter external orders, type  XO.'
      print,'To enter service parts, type SP.'
      print,'To stop, type quit.'
      read, answer
      if answer = 'MS'
         call MSdemands
         go to 10
      elseif answer = 'XO'
         call XOdemands
         go to 10
      elseif answer =  'SP'
         call SPdemands
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   stop
      end
*
      subroutine findsys(part,sys)
*
*  This algorithm finds the system number(sys) for the company
* part number(part).
*
*   Variables:
*    down - down branch for binary search
*    first - company part number with ordinal number one
*    k - next record to try in search for part
*    key - directional key for company part number
```

```
*     mid - midpoint of company part number
*     next - next available record in partid file
*     part  - the company (user) number that is sought
*     pos - ordinal position of the company number
*     s - system (computer) number on record k
*     sys - system number for part
*     u - company (user) number on record k
*     up - up branch for binary search
*
      character part,u
      integer sys,first,s,pos,up,down
*
*  format
10203    format(3i3)
*
*  Check statistics on partid file
*
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10203,rec=0)first,next,mid
*
*  Try midpoint first
*
      k = mid
10    read(2,*,rec=k)u,s,key,pos,up,down
      if part = u                                          :found
         sys = s
         close(unit=2)
         return
      endif                                                :found
      if part < u                                          :whichdirection
         if up = -1                                        :notinfile
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :notinfile
         k = up
         go to 10
      else                                                 :whichdirection
         if down = -1                                      :infileornot
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :infileornot
         k = down
         go to 10
      endif                                                :whichdirection
      close(unit=2)
      return
      end
*
      subroutine MSdemands
*
*  The type of editing for the MS file is specified.
*
*  Variable: answer - response to questions
*
      character answer
10    print,'new,add,change, or quit MS demand list?'
```

```
      read,answer
      if answer = 'new'
         call MSnew
         go to 10
      elseif answer = 'add'
         call MSadd
         go to 10
      elseif answer = 'change'
         call MSchange
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
      subroutine MSnew
*
*  The following routine is used to initialize the MPS file.
*
*  Variables:
*    D - demand when setting key on last MPS for part i
*    Da - date when setting key on last MPS for part i
*    date - due date for MPS demand
*    demand - quantity of part needed
*    keyms - link to next MPS demand
*    mskey - record's number for MPS file
*    n - number of demands for part i
*    next - number of records in partstatus file
*    part - part with MPS demand; company id
*    sys - system number for part
*
      character part
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,sys,
&        date,Da
*
*  Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
901      format(i4,5x,'MPS')
20903    format(i4,',',f8.0,',',i4)
*
*  Reset the MPSkey to -1
*
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10301,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,-1,xorderkey,
         close(unit=3)
      enddo
*
```

```
*   Set the pointer to zero.
*
      mskey = 0
      keyms = 0
*
*   What part has MPS demand?
*
      open(unit=9,file='(f:18)MPS.rel',access='DIRECT',recl=18)
10    print,'part or quit to stop'
      read,part
*
*   No more entries?
*
      if (part='quit') go to 999
*
*   Locate the system number for the part.
*
      call findsys(part,sys)
*
*   The part isn't in the files?'
*
      if (sys=-1) go to 10
*
*   Get the scoop on the part.
*
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*   Set the loop counter.
*
      n = 0
*
      loop
      n = n + 1
*
*   Get the specifics about the demand.
*
      print,'date,demand or -1,0 to stop'
      read(5,*)date,demand
*
*   No more demands for part i? Yes, set last record's key to -1.'
*
      if (date=-1)
         write(9,00901,rec=0)keyms
         read(9,*,rec=mskey)Da,D,k
         write(9,20903,rec=mskey)Da,D,-1
         go to 10
      endif
*
*   No, increment to next record.
*
      mskey = mskey + 1
*
*   Link partstatus record to MPS record.
*
      if (n=1)
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,mskey,xorderkey,
```

```
&           servpartkey,level,levelkey
         close(unit=3)
      endif
*
*  Increment key to next record.
*
      keyms = mskey + 1
      write(9,20903,rec=mskey)date,demand,keyms
      endloop
      go to 10
999   close(unit=9)
      return
      end
*
      subroutine MSadd
*
*  The following allows the user to append the MPS file.
*
*  Variables:
*    D - demand when setting key on last MPS for part i
*    Da - date when setting key on last MPS for part i
*    date - due date of the demand
*    demand - quantity of part needed
*    key - link to next MPS for part i
*    next - location of next available record
*    part - part with the MPS demand
*    sys - system number for part
*
      character part
      integer Da,date,sys,SR,BOM,BofM,ccper,pup,xorderkey,
&        servpartkey
*
*  Formats
901      format(i4,5x,'MPS')
10901    format(i4)
20903    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=9,file='(f:18)MPS,rel',access='DIRECT',recl=18)
      read(9,10901,rec=0)next
*
*  Get additions info (part,etc.).
*
10    print,'part,date,demand quantity or quit,0,0 to stop'
      read(5,*)part,date,demand
*
*  No more additions?
*
      if (part='quit') go to 999
*
*  Locate the system number.
*
      call findsys(part,sys)
*
*  Part not in the files?
*
      if (sys=-1) go to 10
      write(9,20903,rec=next)date,demand,-1
*
```

```
*  Link the old demands to the new demand.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*  if MPSkey = -1, no previous demands exist.
*
      if MPSkey = -1
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,next,xorderkey,
&           servpartkey,level,levelkey
         next = next + 1
         write(9,00901,rec=0)next
         go to 10
      else
*
*  Find the last record for part i.
*
         k = MPSkey
20       read(9,*,rec=k)Da,D,key
*
*  The last record has key = -1.
*
         if key = -1
            write(9,20903,rec=k)Da,D,next
            next = next + 1
            write(9,00901,rec=0)next
            go to 10
         else
            k = key
            go to 20
         endif
      endif
999   close(unit=3)
      close(unit=9)
      return
      end
*
      subroutine MSchange
*
*  With this subprogram, the user may correct MPS input errors.
*
*  Variables:
*     k - record number for search of old data
*     key - link to next MS demand
*     newdate - correct date
*     newdemand - correct demand
*     olddate - date to be changed
*     olddemand - demand to be changed
*     part - part with demand to be corrected
*     sys - system number of part
*
      character part
      real newdemand
      integer olddate,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        sys,date
*
```

```
*  Formats
20903    format(i4,',',f8.0,',',i4)
*
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      open(unit=9,file='(f:18)MPS.rel',access='DIRECT',recl=18)
*
*  Get information on part to be changed.
*
10    print,'part,date,demand quantity to be changed'
      print,'  or quit,0,0  to stop'
      read(5,*)part,olddate,olddemand
*
*  No more changes?
*
      if(part='quit') go to 999
*
*  Locate the system number for part i.
*
      call findsys(part,sys)
*
*  Part isn't in the file?'
*
      if(sys=-1) go to 10
*
*   How demand information is to be changed?
*
      print,'correct date, correct demand quantity'
      read(5,*)newdate,newdemand
*
*  Find the record that needs changing.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*  if MPSkey = -1, there are no records on file for part i.
*
      if MPSkey = -1
         print,'old date and demand not on file'
         go to 10
      else
*
*  Start search with record=MPSkey.
*
         k = MPSkey
20       read(9,*,rec=k)date,demand,key
         if (date=olddate.and.demand=olddemand)
*
*  When the old demand is found, make the change.
*
            write(9,20903,rec=k)newdate,newdemand,key
            go to 10
         else
            if key = -1
*
*  if key = -1, there are no more records for part i.
*
               print,'old date and demand not on file'
```

```
               go to 10
            else
*
*  continue search with next record=key.
*
               k = key
               go to 20
            endif
         endif
      endif
999   close(unit=3)
      close(unit=9)
      return
      end
*
      subroutine  XOdemands
*
*  The type of editing for the external Order(XO) file
* is specified.
*
*  Variable: answer - response to questions
*
      character answer
10    print,'new,add,change, or quit XO demand list?'
      read,answer
      if answer = 'new'
         call XOnew
         go to 10
      elseif answer  = 'add'
         call XOadd
         go to 10
      elseif answer  = 'change'
         call XOchange
         go to 10
      elseif answer  = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
      subroutine XOnew
*
*  A XO file can be initialized with the following subprogram.
*
*  Variables:
*    D - demand; setting key on last demand
*    Da - date; setting key on last demand
*    date - date when part is needed
*    demand - quantity of part needed
*    keyxo - key to next demand for part i
*    n - count of number of times through loop
*    next - how many records in partstatus file
*    part - part with XO demand
*    sys - system number of part
*    xokey - records increment for XO file
*
      character part
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
```

```
&       sys,date,Da,xokey
*  Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
1001     format(i4,5x,'XO')
21003    format(i4,',',f8.0,',',i4)
*
*  Reset the xorderkey to -1.
*
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,-1,servpartkey,
&           level,levelkey
         close(unit=3)
      enddo
*
*  Begin with the pointer and record number equal to 0.
*
      xokey = 0
      keyxo = 0
      open(unit=10,file='(f:18)XO.rel',access='DIRECT',recl=18)
*
*  What part has the external order demand?
*
10    print,'part or quit to stop'
      read,part
*
*  No more additional inputs?
*
      if (part='quit') go to 999
*
*  Find the system number for part i.
*
      call findsys(part,sys)
*
*  Part not on file?
*
      if (sys=-1) go to 10
*
*  Get the facts about the part.
*
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*  Set the loop count.
*
      n = 0
*
      loop
      n = n + 1
```

```
*
*   Get the specifics about the demand.
*
      print,'date,demand or -1,0 to stop'
      read(5,*)date,demand
*
*   No more entries?
*
      if (date = -1)
         write(10,01001,rec=0)keyxo
         read(10,*,rec=xokey)Da,D,k
         write(10,21003,rec=xokey)Da,D,-1
         go to 10
      endif
*
*   Increment record number.
*
      xokey = xokey + 1
      if (n=1)
*
*   Link the partstatus record to the XO record.
*
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xokey,
&           servpartkey,level,levelkey
         close(unit=3)
      endif
*
*   Increment key to the next record.
*
      keyxo = xokey + 1
      write(10,21003,rec=xokey)date,demand,keyxo
      end loop
      go to 10
999   close(unit=10)
      return
      end
*
      subroutine XOadd
*
*   The XO file can be appended with the following subprogram.
*
*   Variables:
*     D - demand; setting last record for part i
*     Da - date; setting last record for part i
*     date - date when part is needed
*     demand - quantity of part needed
*     next - number of records in XO file
*     part - part with additional demand
*     sys - system number for part
*
      character part
      integer  Da,date,sys,SR,BOM,BofM,ccper,pup,xorderkey,
&        servpartkey
*
*    Formats
1001     format(i4,5x,'XO')
11001    format(i4)
21003    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
```

```
&          ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=10,file='(f:18)XO,rel',access='DIRECT',recl=18)
      read(10,11001,rec=0)next
*
*  Get the specifics about the new addition.
*
10    print,'part,date,demand quantity or quit,0,0 to stop'
      read(5,*)part,date,demand
*
*  No more additions? stop.
*
      if (part='quit') go to 999
*
*  Get the system number.
*
      call findsys(part,sys)
*
*  Part not in file? correct?
*
      if (sys=-1) go to 10
      write(10,21003,rec=next)date,demand,-1
*
*  Link the old demand to the new demand.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if xorderkey = -1
*
*  if xorderkey = -1, there are no previous demands.
*
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,next,
&           servpartkey,level,levelkey
         next = next + 1
         write(10,01001,rec=0)next
         go to 10
      else
*
*  Search for the last record which is linked to new demand.
*
         k = xorderkey
20       read(10,*,rec=k)Da,D,key
         if key = -1
*
*  The last record has key = -1.
*
            write(10,21003,rec=k)Da,D,next
            next = next + 1
            write(10,01001,rec=0)next
            go to 10
         else
*
*  See if the next record(key) is last.
*
            k = key
            go to 20
```

```
         endif
      endif
999   close(unit=3)
      close(unit=10)
      return
      end
*
      subroutine XOchange
*
*  The XO file errors can be corrected with the following routine.
*
*   Variables:
*    key - next demand record for part i
*    k - record number of old demands
*    newdate - correct date
*    newdemand - correct demand
*    olddate - date to be changed
*    olddemand - demand to be changed
*    part - part with demand change
*    sys - system number of part
*
      character part
      real newdemand
      integer olddate,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        sys,date
*  Formats
21003    format(i4,',',f8.0,',',i4)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=10,file='(f:18)XO,rel',access='DIRECT',recl=18)
*
*  What part demand needs to be corrected?
*
10    print,'part,date,demand quantity to be changed'
      print,'   or quit,0,0 to stop'
      read(5,*)part,olddate,olddemand
*
*  No more changes? stop.
*
      if (part='quit') go to 999
*
*  Locate system number.
*
      call findsys(part,sys)
*
*  Part not in file? correction?
*
      if (sys=-1) go to 10
*
*  How is the demand to be changed?
*
      print,'correct date,correct demand quantity'
      read(5,*)newdate,newdemand
*
*  Find the old demand.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
```

```
      if xorderkey = -1
*
*   if xorderkey = -1, there are no records on file.
*
         print,'old date and demand not on file'
         go to 10
      else
*
*   Start search with record=xorderkey.
*
         k = xorderkey
20       read(10,*,rec=k)date,demand,key
         if (date=olddate.and.demand=olddemand)
            write(10,21003,rec=k)newdate,newdemand,key
            go to 10
         else
*
*   if key = -1, there are no more records for part i.
*
            if key = -1
               print,'old date and demand not on file'
               go to 10
            else
*
*   See if the next record is the old demand.
*
               k = key
               go to 20
            endif
         endif
      endif
999   close(unit=3)
      close(unit=10)
      return
      end
*
      subroutine SPdemands
*
*    The type of editing for the  Service Parts(SP) file is
* specified here.
*
*   Variable: answer - response to questions
*
      character answer
10    print,'new,add,change or quit SP demand list?'
      read,answer
      if answer  = 'new'
         call SPnew
         go to 10
      elseif  answer  = 'add'
         call SPadd
         go to 10
      elseif answer = 'change'
         call  SPchange
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
```

```
999   return
      end
*
      subroutine SPnew
*
*  The SP file can be initialized with this subprogram.
*
*  Variables:
*    date - date when part is needed
*    demand - quantity of part needed
*    keysp - increment of next record key
*    n - loop increment
*    next - number of records in partstatus file
*    part - company id for part with SP demand
*    spkey - next record location
*    sys - part's system number'
*
      character part
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,sys,
&        date,Da,spkey
*
*  Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i3,',',f8.0,',',i1,5(',',i3))
1101     format(i4,5x,'SP')
21103    format(i4,',',f8.0,',',i4)
*
*  Reset the servpartkey to -1.
*
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10301,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           -1,level,levelkey
         close(unit=3)
      enddo
*
*  Begin with the last record and pointer equal to zero.
*
      spkey = 0
      keysp = 0
      open(unit=11,file='(f:18)SP,rel',access='DIRECT',recl=18)
*
*   What part has the service part demand?
*
10    print,'part  or quit to stop'
      read,part
*
*  No more information to be entered? stop.
*
      if (part='quit') go to 999
*
*  Locate the system number.
```

```
*
      call findsys(part,sys)
*
*   Part not in file?
*
      if (sys=-1) go to 10
*
*   Get the facts about part i.
*
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*   Set the loop count.
*
      n  = 0
*
      loop
      n = n + 1
*
*    Get the specifics about SP demand.
*
      print,'date,demand or -1,0 to stop'
      read(5,*)date,demand
*
*    No more demands for part i?
*
      if (date=-1)
         write(11,01101,rec=0)keysp
         read(11,*,rec=spkey)Da,D,k
         write(11,21103,rec=spkey)Da,D,-1
         go to 10
      endif
*
*   Increment the record number.
*
      spkey = spkey + 1
      if n = 1
*
*   Link the partstatus record to the SP demand record.
*
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           spkey,level,levelkey
         close(unit=3)
      endif
*
*   Increment the key to next record.
*
      keysp = spkey + 1
      write(11,21103,rec=spkey)date,demand,keysp
      end loop
      go to 10
999   close(unit=11)
      return
      end
*
      subroutine SPadd
*
```

```
*  The SP file can be appended with the following.
*
*   Variables:
*    D - demand; used when linking records
*    Da - date; used when linking records
*    date - date when part needed
*    demand - quantity of part needed
*    k - record location for next read
*    key - next demand location for part i
*    next - number of records in SP file
*    part - part with SP demands
*    sys - part's system number'
*
      character part
      integer Da,date,sys,SR,BOM,BofM,ccper,pup,xorderkey,
&        servpartkey
*
*  Formats
1101     format(i4,5x,'SP')
21103    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.4,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=11,file='(f:18)SP,rel',access='DIRECT',recl=18)
      read(11,11101,rec=0)next
*
*  What addition demand needs to be entered?
*
10    print,'part,date,demand or quit,0,0, to stop'
      read(5,*)part,date,demand
*
*  No more additions? stop.
*
      if (part='quit') go to 999
*
*  Find the system number.
*
      call findsys(part,sys)
*
*  Part not in file? correction?
*
      if (sys=-1) go to 10
      write(11,21103,rec=next)date,demand,-1
*
*  Link the old demands to the new demand.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if servpartkey = -1
*
*  if servpartkey = -1, there are no previous demands.
*
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           next,level,levelkey
         next  = next + 1
```

```
         write(11,01101,rec=0)next
         go to 10
      else
*
*  Start search for the last record with record=servpartkey.
*
         k = servpartkey
20       read(11,*,rec=k)Da,D,key
         if key = -1
*
*  The last record has key = -1.
*
            write(11,21103,rec=k)Da,D,next
            next = next + 1
            write(11,01101,rec=0)next
            go to 10
         else
*
*  continue search for last record.
*
            k = key
            go to 20
         endif
      endif
999   close(unit=3)
      close(unit=11)
      return
      end
*
      subroutine SPchange
*
*  The SP input errors may be corrected with this routine.
*
*  Variables:
*    k - record number for read location
*    key - next record number for part i demand
*    newdate - date correction
*    newdemand - demand correction
*    olddate - date to be changed
*    olddemand - demand to be changed
*    part - part to be changed
*    sys - part's system number'
*
      character part
      real newdemand
      integer olddate,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        date,sys
*
*  Formats
21103    format(i4,',',f8.0,',',i4)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=11,file='(f:18)SP,rel',access='DIRECT',recl=18)
*
*  What demand needs correcting?
*
10    print,'part,date,demand quantity to be changed'
      print,'   or quit,0,0  to stop'
      read(5,*)part,olddate,olddemand
*
*  No more corrections? stop.
```

```
*
      if (part='quit') go to 999
*
*   Locate the system number.
*
      call findsys(part,sys)
*
*   Part not in file? correction?
*
      if (sys=-1) go to 10
*
*   What is the correction?
*
      print,'correct date,correct demand quantity'
      read(5,*)newdate,newdemand
*
*   Find the old demand.
*
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,Bo#M,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if servpartkey = -1
*
*   if servpartkey = -1, there are no demands for part i.
*
         print,'old date and demand not on file'
         go to 10
      else
*
*   Start search with record = servpartkey.
*
         k = servpartkey
20       read(11,*,rec=k)date,demand,key
         if (date=olddate.and.demand=olddemand)
*
*   When old demand is found, make the change.
*
            write(11,21103,rec=k)newdate,newdemand,key
            go to 10
         else
            if key = -1
*
*   if key = -1, there are no more demands for part i.
*
               print,'old date and demand not on file'
               go to 10
            else
*
*   continue search to next record(key).
*
               k = key
               go to 20
            endif
         endif
      endif
999   close(unit=3)
      close(unit=11)
      return
```

```
end
```

```
*
*  file - Level
*
*  Karen M. May      Date: June 3, 1983
*
*  function - assign part levels and link levels for explosion
*
*  Variables:
*    BofM - key to first appearance as child part
*    BOM - key to first appearance as parent part
*    c - child part on bom record
*    ckey - key to next parent part for c
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time for carrying cost
*    lead - lead time
*    level - part level
*    levelkey - key for MRP flow
*    lotp - time in days for L4L and FPQ
*    lotr - lotting rule
*    MPSkey - key to MPS demands
*    nextbom - next available bom record
*    nextpart - next available partstatus record
*    numkids - number of different components
*    oh - on hand inventory
*    p - parent part on bom record
*    pkey - key to next child part for p
*    pu - periodal usage(units/unit of time)
*    pup - time unit for periodal usage
*    q - number of c/p on bom record
*    scost - set up cost
*    servpartkey - key to Service Part demands
*    SR - key to scheduled receipts
*    ucost - unit cost
*    xorderkey - key to external orders
*
*  Subprograms: LandSinitial,Lassign,Sremove,Levelconnect
*
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey
      call LandSinitial(nextpart,nextbom)
*
*   Level Number increment if after Lassign and Sremove,parts
* still need levels assigned.
*
      L = 0
10    L = L + 1
*
*   What parts qualify to be assigned this level?
*
      call Lassign(L,nextpart,nextbom)
*
*  Eliminate B.O.M. records for next round?
*
      call Sremove(L,nextpart,nextbom)
*
*  Have all parts been assigned a level?
*
      j = 0
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
```

```
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         close(unit=3)
         if (level=-1) j = 1
      enddo
      if (j=1) go to 10
*
*  Set the MRP explosion flow.
*
      call Levelconnect(L,nextpart)
      end
*
      subroutine LandSinitial(nextpart,nextbom)
*
*  This subprogram initialize level = -1, levelkey = -1,
* and stack flag to 1.
*
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&           p,c,q,pkey,ckey
*
*  Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
20106    format(2(i3,','),i4,3(',',i4))
10101    format(i4)
*
*  Initialize level at -1; -1 means no level has been assigned
*
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10301,rec=0)nextpart
      close(unit=3)
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,-1,-1
         close(unit=3)
      enddo
*
*  Initialize stack at 1; 1 = in stack
*
      open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      read(1,10101,rec=0) nextbom
      do i = 1,nextbom-1,1
         read(1,*,rec=i)p,c,q,pkey,ckey
         write(1,20106,rec=i)p,c,q,pkey,ckey,1
      enddo
      close(unit=1)
      return
      end
*
      subroutine Lassign(L,nextpart,nextbom)
*
*  This algorithm assigns level L to the parts that qualify.
*
*  Variables:
```

```
*     assign - flag for part in list(1=yes;0=no)
*     i - system part number
*     s - stack flag (1=in;0=out)
*
      integer SR,BofM,BOM,ccper,pup,xorderkey,servpartkey,
&           p,c,q,pkey,ckey,s
*
*   format
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*
*   Check to see if part i is in the bom child list.
*
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
*
*   if level = -1,then part need a level assignment.
*
         if level = -1
*
*   Part is in child list if i = c, and in stack if s=1.
*
               assign = 1
               open(unit=1,file='(f:27)bom,rel',access='DIRECT')
               do j = 1,nextbom-1,1
                  read(1,*,rec=j)p,c,q,pkey,ckey,s
*
*   Found in the list?
*
                  if (i=c.and.s=1) assign=0
               enddo
               close(unit=1)
               if assign = 1
*
*   if not found in stack, assign part to level L.
*
                  write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,
&                    lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&                    MPSkey,xorderkey,servpartkey,L,levelkey
               endif
            endif
            close(unit=3)
         enddo
         return
         end
*
      subroutine Sremove(L,nextpart,nextbom)
*
*  Eliminate B.O.M. records from the stack when part with level L
* is in parent list.
*
*   Variable:
*     L - level number to be assigned
*
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        p,c,q,pkey,ckey,s
*
```

```
*  Formats
20106    format(2(i3,','),i4,3(',',i4))
*
*  Determine if bom record should be removed from the stack.
*
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         if level = L
            open(unit=1,file='(f:27)bom,rel',access='DIRECT')
            do j = 1,nextbom-1,1
               read(1,*,rec=j)p,c,q,pkey,ckey,s
               if (i=p) write(1,20106,rec=j)p,c,q,pkey,ckey,0
            enddo
            close(unit=1)
         endif
         close(unit=3)
         enddo
         return
         end
*
      subroutine Levelconnect(L,nextpart)
*
*    The subprogram connects levels in flow  order for MRP
*  explosion.
*
*  Variables:
*    L - level number for linking
*    Lmax - maximum part level assigned
*    n - number of times through loop
*
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        SR1,BOM1,BofM1,ccper1,pup1,xorderkey1,servpartkey1
*
*  Formats
302      format(2i3,5x,'Partstatus Data File')
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*
*  Highest level assigned is the stopping criteria.
*
      Lmax = L
*
*  Connect the lower levels first.
*
      L = 0
*
*  Increment times through loop.
*
      n = 0
      loop
      L = L + 1
*
*  Beyond maximum level? stop.
*
      quitif L > Lmax
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
```

```
      read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*  if level = L, set link.
*
      if level = L
         n = n + 1
*
*  Link after first read.
*
         if (L>1.or.n>1)
            write(3,20319,rec=i1)oh1,SR1,numkids1,BOM1,BofM1,
&              lead1,lotr1,lotp1,scost1,ucost1,ccost1,ccper1,
&              pu1,pup1,MPSkey1,xorderkey1,servpartkey1,level1,i
         endif
*
*  Store the first level 1 in record 0; the explosion starts here.
*
         if (L=1.and.n=1)
            write(3,00302,rec=0) nextpart,i
         endif
*
*  Remember the details on the last location linked.
*
         i1 = i
         oh1 = oh
         SR1  = SR
         numkids1  = numkids
         BOM1 = BOM
         BofM1 = BofM
         lead1 = lead
         lotr1 = lotr
         lotp1 = lotp
         scost1 = scost
         ucost1 = ucost
         ccost1 = ccost
         ccper1 = ccper
         pu1 = pu
         pup1 = pup
         MPSkey1 = MPSkey
         xorderkey1 = xorderkey
         servpartkey1 = servpartkey
         level1 = level
      endif
      close(unit=3)
   enddo
   endloop
   return
   end
```

```
*
*    file - MRP
*
*   Karen M. May      Date: June 6, 1983
*
*     BfoM - first appearance in bom child's list'
*     BOM - first appearance in bom parent's list'
*     buckets - number of periods in horizon
*     c - company part number
*     ccost - carrying cost ($/time period)
*     ccper - time period for ccost
*     ddkey - next available record in DD file
*     lead - lead time in days
*     level - part level number
*     levelkey - next part to explode
*     lotp - number of days for L4L and FPQ
*     lotr - lotting rule
*     MPSkey - key to MPS demand records
*     name - company name
*     nextpart - next available record in partstatus
*     numkids - number of different components
*     oh - on hand inventory
*     parttoexplode - record number of part to explode
*     pu - periodal usage (units/time period)
*     pup - time period for pu
*     scost - set up cost
*     servpartkey - key to SP demand records
*     SR - key to scheduled receipts
*     start - first part to explode
*     ucost - unit cost
*     xorderkey - key to XO demand records
*
*   Subprograms: DDkey,setupHorizon,MPStransfer,XOtransfer,
* SPtransfer,DependentDemands,GrossRequirements,SRtransfer,
* OHandNR,LotSize,L4L,FPQ,OrderRelease,OutPut,SIX,FIVE,FOUR,
* THREE,TWO,ONE,OrderWrite
*
      character c,name
      integer start,parttoexplode,SR,BofM,BOM,ccper,pup,
&             xorderkey,servpartkey,buckets,ddkey
*
*   format
10302    format(2i3)
10801    format( ' DD file ')
*
*   Initialize the DD file key.
*
      ddkey = 0
*
*   Label the DD file.
*
      open(unit=8,file='(f:17)DD,rel',access='DIRECT',recl=17)
      write(8,10801,rec=0)
      close(unit=8)
*
*   Heading for the MRP printer output.
*
      open(unit=6,file='printer')
      write(6,10600)
10600    format(16('*'),'Materials Requirement Planning',16('*'))
```

```
      close(unit=6)
*
*   The name is needed for the Purchase Order Heading.
*
      print,'Company Name (up to 12 characters in length)?'
      read,name
*
*   Initialize Dependent Demand key (DDkey) in partid file.
*
      call DDkey
*
*   Which part is exploded first?
*
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10302,rec=0) nextpart,start
      parttoexplode = start
      close(unit=3)
*
*   Get the scoop on the part to be exploded?
*
10    open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=parttoexplode)oh,SR,numkids,BOM,BofM,lead,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*
*   Headings for printout.
*
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,*,rec=parttoexplode)c
      close(unit=2)
      open(unit=6,file='printer')
      write(6,10601)c,parttoexplode
10601    format('Part Number ',a10,';System Number',i3)
      write(6,10602)lead,oh
10602    format('Lead Time = ',i4,' days; On Hand Inventory is ',
&           f8.0)
      close(unit=6)
*
*   Initialize values in horizon.
*
      call setupHorizon (buckets)
*
*   What demands are placed on this part?
*
      if(MPSkey<>-1) call MPStransfer(MPSkey,buckets)
      if(xorderkey<>-1) call XOtransfer(xorderkey,buckets)
      if(servpartkey<>-1) call SPtransfer(servpartkey,buckets)
      if(BofM<>-1)call DependentDemands(BofM,buckets,parttoexplode)
*
*   Total Demands.
*
      call GrossRequirements(buckets)
*
*   What orders have been placed?
*
      if(SR<>-1) call SRtransfer(SR,buckets)
*
*   Determine on have quantities and net requirements.
*
      call OHandNR(oh,buckets)
```

```
*
*  Determine the lot (or order) size.
*
      call LotSize(lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&        buckets)
*
*  Adjust order date according to lead time.
*
      call OrderRelease(buckets,lead,ddkey,parttoexplode)
*
*  print out results.
*
      call OutPut(buckets,lead,name,parttoexplode)
*
*  Are any parts left to be exploded?
*
      if(levelkey=-1) go to 999
      parttoexplode = levelkey
      close(unit=3)
      go to 10
999   stop
      end
*
      subroutine DDkey
*
*  Dependent demand key(DDkey) is initialized in the following
* subprogram.
*
*  Variables:
*    d - down key for binary search
*    k - key to next part
*    next - number of parts plus 1
*    p - ordinal number for part
*    sys - system part number
*    u - up key for binary search
*
      character c
      integer first,sys,p,u,d
*
*  Formats
10202    format(2i3)
20207    format(a10,5(',',i3),',',i4)
*
*  How many records need to be initialized?
*
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10202,rec=0)first,next
*
*  Initialize the required number of records.
*
      do i = 1,next-1,1
         read(2,*,rec=i)c,sys,k,p,u,d
         write(2,20207,rec=i)c,sys,k,p,u,d,-1
      enddo
      close(unit=2)
      return
      end
*
      subroutine setupHorizon(buckets)
*
```

```
*  The planning horizon is initialized with this routine.
*
*  Variables:
*    buckets - number of periods in horizon
*    date - first day of the period
*    period - planning period number
*
     integer buckets,period,date
*
*  Formats
10701   format(i3)
20712   format(i3,',',i4,10(',',f8.0))
*
     open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
     read(7,10701,rec=0)buckets
     close(unit=7)
     do i = 1,buckets,1
        open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
        read(7,*,rec=i)period,date
        write(7,20712,rec=i)period,date,0.,0.,0.,0.,0.,0.,0.,0.,
&          0.,0.
        close(unit=7)
     enddo
     return
     end
*
     subroutine MPStransfer(MPSkey,buckets)
*
*  The MPS(end item) demands are transferred to planning horizon.
*
*  Variables:
*    buckets - number of periods in horizon
*    date - first day of period n
*    date1 - first day of period n+1
*    day - date demand is needed
*    demand - number of units needed
*    dMPS - end item demands
*    k - location of MPS demand
*    key - location of next demand
*    MPSkey - location of first MPS demand
*    period - number of planning periods
*
     integer buckets,day,period,date,date1,period1
     real NR,LR
*
*  Formats
20712   format(i3,',',i4,10(',',f8.0))
*
*  Begin with the first MPS demand
*
     k = MPSkey
*
*  Get the details about the demands
*
     open(unit=9,file='(f:18)MPS.rel',access='DIRECT',recl=18)
10   read(9,*,rec=k)day,demand,key
*
*  Find the correct time period for the demand.
*
     do i = 1,buckets,1
```

```
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         j = i + 1
         if j > buckets
            date1 = 9999
            go to 15
         endif
         read(7,*,rec=j)period1,date1
15       if (date<=day.and.day<date1)
            demand = dMPS + demand
            write(7,20712,rec=i)period,date,demand,dXO,dSP,sumDD,
&              GR,SR,OH,NR,LR,OR
            close(unit=7)
            go to 20
         endif
         close(unit=7)
      enddo
*
*  Any more demands for part i?
*
20    if (key=-1) go to 999
      k = key
      go to 10
999   close(unit=9)
      return
      end
*
      subroutine XOtransfer(xorderkey,buckets)
*
*    The XO (external order) demands are transferred to the
* planning horizon.
*
*  Variable:
*    xorderkey - location of first XO demand
*
      integer buckets,day,period,date,date1,period1,xorderkey
      real NR,LR
*
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*
*  Begin with the first demand.
*
      k = xorderkey
*
*  Get the information about the demand.
*
      open(unit=10,file='(f:18)XO.rel',access='DIRECT',recl=18)
10    read(10,*,rec=k)day,demand,key
*
*  Locate the correct time period for the demand.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         j = i + 1
         if j > buckets
            date1 = 9999
```

```
            go to 15
         endif
         read(7,*,rec=j)period1,date1
15       if (date<=day.and.day<date1)
            demand = dXO + demand
            write(7,20712,rec=i)period,date,dMPS,demand,dSP,sumDD,
&              GR,SR,OH,NR,LR,OR
            close(unit=7)
            go to 20
         endif
         close(unit=7)
      enddo
*
*  Any more demand for part i?
*
20    if(key = -1) go to 999
      k = key
      go to 10
999   close(unit=10)
      return
      end
*
      subroutine SPtransfer(servpartkey,buckets)
*
*  The SP (Service Parts) demands are transferred to the
* planning horizon.
*
*  Variable:
*    servpartkey - location of first SP demand
*
      integer buckets,day,period,date,date1,period1,servpartkey
      real NR,LR
*
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*
*  Begin with the first demand.
*
      k = servpartkey
*
*  Get the details about the demand.
*
      open(unit=11,file='(f:18)SP.rel',access='DIRECT',recl=18)
10*   read(11,*,rec=k)day,demand,key
*
*  Locate the proper time period for the demand.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,NR,
&           LR,OR
         j = i + 1
         if j > buckets
            date1 = 9999
            go to 15
         endif
         read(7,*,rec=j)period1,date1
15       if (date<=day.and.day<date1)
            demand = dSP + demand
            write(7,20712,rec=i)period,date,dMPS,dXO,demand,sumDD,
```

```
&              GR,SR,OH,NR,LR,OR
            close(unit=7)
            go to 20
         endif
         close(unit=7)
      enddo
*
*  Any more demands for part i?
*
20    if (key=-1) go to 999
      k = key
      go to 10
999   close(unit=11)
      return
      end
*
      subroutine GrossRequirements(buckets)
*
*  The gross requirements for each period is calculated by the
* following procedure.
*
*  Variables:
*    buckets - number of periods in horizon
*    dMPS - end item (MPS) demands
*    dSP - service part demands
*    dXO - external order demands
*    GR - gross requirements
*    sumDD - sum total of dependent demands
*
      integer period,date,buckets
      real NR,LR
*
*  format
20712    format(i3,',',i4,10(',',f8.0))
*
*  Add the GR for each period.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         GR = dMPS + dXO + dSP + sumDD
         write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,
&           OH,NR,LR,OR
         close(unit=7)
      enddo
      return
      end
*
      subroutine SRtransfer(SRkey,buckets)
*
*  The scheduled receipts are transferred into the planning
* horizon.
*
*  Variable:
*    SRkey - location of first scheduled receipts
*
      integer buckets,SRkey,period,date,day,date1,period1
      real NR,LR
*  Formats
```

```
20712   format(i3,',',i4,10(',',f8.0))
*
*  Begin with first SR.
*
      k = SRkey
*
*  Get details about SR.
*
      open(unit=4,file='(f:18)SR,rel',access='DIRECT',recl=18)
10    read(4,*,rec=k)day,qty,key
*
*  Find proper period for SR in planning horizon.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         j = i + 1
         if j > buckets
            date1 = 9999
            go to 15
         endif
         read(7,*,rec=j)period1,date1
15       if (date<=day.and.day<date1)
            SR = SR + qty
            write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,
&              SR,OH,NR,LR,OR
            close(unit=7)
            go to 20
         endif
         close(unit=7)
      enddo
*
*  Any more SR?
*
20    if (key=-1) go to 999
      k = key
      go to 10
999   close(unit=4)
      return
      end
*
      subroutine OHandNR(oh,buckets)
*
*  The following calculates on hand and net requirements amounts.
*
*  Variables:
*    buckets - number of periods in horizon
*    net - amount of on hand inventory
*    NR - net requirements
*    oh - on hand inventory
*    OH - on hand inventory stored in bucket
*
      integer period,date,buckets
      real NR,LR,net
*
*  Formats
20712   format(i3,',',i4,10(',',f8.0))
*
*  Initialize net with on hand inventory.
```

```
*
      net = oh
*
*   Determine on hand inventory and net requirements for each per.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         net = GR - (SR + net)
         if (net<0)
            OH = -net
            net = -net
            go to 10
         endif
         if (net>0)
            NR = net
            net  = 0
         endif
10       write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,
&           OH,NR,LR,OR
         close(unit=7)
      enddo
      return
      end
*
      subroutine LotSize(lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&        buckets)
*
*   Proper Lot Size technique is requested by these statements.
*
*   Variables:
*     buckets - number of periods in horizon
*     ccper - time unit for ccost
*     ccost - carrying cost
*     lotp - number of days covered by FPQ or L4L
*     lotr - lotting rule
*     pu - periodal usage
*     pup - time period of pu
*     scost - set up cost
*     ucost - unit cost
*
      integer ccper,pup,buckets
      if (lotr = 1) call L4L(buckets)
      if (lotr = 2) call FPQ(buckets,lotp)
      return
      end
*
      subroutine L4L(buckets)
*
*   Lot-for-lot technique is performed by the subprogram.
*
*   Variables:
*     NR - net requirements
*     LR - lot size requirements
*
      integer period,date,buckets
      real NR,LR
*
*   format
```

```
20712   format(i3,',',i4,10(',',f8.0))
      open(unit=6,file='printer')
      write(6,10601)
10601 format('Lot Size is determined with Lot-for-Lot Technique')
*
      close(unit=6)
*
*  For each period, Lot Size = Net Requirements.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         LR = NR
         write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,
&           OH,NR,LR,OR
         close(unit=7)
      enddo
      return
      end
*
      subroutine FPQ(buckets,lotp)
*
*  The Fixed Period Quantity Lotting Technique is performed
* by the following subprogram.
*
*  Variables:
*    date - first day of first period
*    date1 - first day of period under consideration
*    i - number of buckets
*    j - next bucket under consideration
*    LR - lot size requirements
*    NR - net requirements
*    NR1 - net requirements of period under consideration
*    totdays - number of days since first period
*
      integer period,date,period1,date1,buckets,totdays
      real NR,LR,NR1,LR1
*
*  format
20712   format(i3,',',i4,10(',',f8.0))
*
      open(unit=6,file='printer')
      write(6,10601)
10601 format('Lot Size is determined with Fixed Period'
&        Algorithm.')'
      close(unit=6)
      i = 0
*
*  Find the first nonzero net requirement(NR).
*
      loop
      i = i + 1
      if (i> buckets) go to 999
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,NR,
&        LR,OR
      close(unit=7)
      if (NR <> 0) go to 20
      endloop
```

```
*
*  Sum NR until lot period length has been reached or until no
* buckets are left.
*
20    loop
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,NR,
&        LR,OR
      LR = NR
      j = i
30    write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&        NR,LR,OR
      close(unit=7)
      j = j + 1
*
*  No more buckets left?
*
      if (j > buckets) go to 999
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=j)period1,date1,dMPS1,dXO1,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      LR = LR + NR1
*
*  Reached lot period length?
*
      if lotp <= (date1 - date)
         close(unit=7)
         i = j
         go to 20
      else
         go to 30
      endif
      endloop
999   return
      end
*
     subroutine OrderRelease(buckets,leadtime,DDkey,parttoexplode)
*
*  The subprogram determines when order is to be released;stores
* DD; and places pointer for DD in partid file.
*
*  Variables:
*     currentperiod - first period in planning horizon
*     date - first day of period n
*     date1 - first day of period n-1
*     leadtime - lead time
*     LR - lot size
*     n - number of DD stored
*     next - number of buckets left in horizon
*     or - size of order release
*     OR1 - order release in period n-1
*     pastdue - NR which are overdue
*     today - first day of currentperiod
*     totdays - number of days until due date
*
      character c
      integer buckets,DDkey,parttoexplode,period,date,period1,
&        date1,currentperiod,today,totdays,sys,p,u,d,DD
      real NR,LR,NR1,LR1
*
```

```
*  Formats
20207    format(a10,5(',',i3),',',i4)
20712    format(i3,',',i4,10(',',f8.0))
20803    format(i3,',',i4,',',f8.0)
*
*   Initialize the number of DD stored.
*
      n = 0
*
*   Initialize the amount of NR overdue.
*
      pastdue = 0.
      open(unit=8,file='(f:17)DD,rel',access='DIRECT',recl=17)
*
*   Find out today's date.'
*
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
      read(7,*,rec=1)currentperiod,today
      close(unit=7)
      do i = buckets,1,-1
         next = i
*
*   What is the lot size or size of the order?
*
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         close(unit=7)
*
*   Is there enough time to make lead time adjustment?
*
         if (leadtime > (date - today)) go to 20
         do j = next,1,-1
            open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
            read(7,*,rec=j)period1,date1,dMPS1,dXO1,dSP1,sumDD1,
&              GR1,SR1,OH1,NR1,LR1,OR1
*
*   Has enough time elapsed for lead time adjustment?
*
            totdays = date - date1
            if (leadtime <= totdays)
               or = LR + OR1
               write(7,20712,rec=j)period1,date1,dMPS1;dXO1,dSP1,
&                 sumDD1,GR1,SR1,OH1,NR1,LR1,or
               close(unit=7)
               if (or <> 0)
*
*   Store DD for part i in DD file.
*
                  n = n + 1
                  DDkey = DDkey + 1
                  write(8,20803,rec=DDkey)parttoexplode,date1,or
                  if  n = 1
*
*   Link partid file to first DD stored.
*
                  open(unit=2,file='(f:35)partid,rel',access=
&                    'DIRECT')
                     read(2,*,rec=parttoexplode)c,sys,k,p,u,d,DD
                     write(2,20207,rec=parttoexplode)c,sys,k,p,u,
```

```
&                          d,DDkey
                        close(unit=2)
                      endif
                    endif
                    close(unit=7)
                    go to 15
                 endif
                 close(unit=7)
              enddo
15            continue
           enddo
           go to 999
*
*  Add up the overdue demands.
*
20         do k = next,1,-1
              open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
              read(7,*,rec=k)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&                NR,LR,OR
              pastdue = pastdue + LR
              close(unit=7)
           enddo
999        close(unit=8)
           open(unit=6,file='printer')
           write(6,10601)pastdue
10601         format('The amount of parts overdue is ',f8.0)
           close(unit=6)
           return
           end
*
           subroutine OutPut(buckets,leadtime,name,parttoexplode)
*
           character name
           integer buckets,orderperiod,orderdate,period,date,columns,
&             parttoexplode
           n = 0
           i = 0
           columns = buckets
10         columns = columns - 6
         if (columns>=0)call SIX(orderperiod,orderdate,ordersize,n,i)
         if (columns=-1)call FIVE(orderperiod,orderdate,ordersize,n,i)
         if (columns=-2)call FOUR(orderperiod,orderdate,ordersize,n,i)
         if (columns=-3)call THREE(orderperiod,orderdate,ordersize,n,i)
         if (columns=-4)call TWO(orderperiod,orderdate,ordersize,n,i)
         if (columns=-5)call ONE(orderperiod,orderdate,ordersize,n,i)
           if (n = 1)
              period = orderperiod
              date = orderdate
              size  = ordersize
           endif
           if (columns<=0) go to 999
           go to 10
999        if (size>0) call OrderWrite(period,date,size,leadtime,
&             parttoexplode,name)
           return
           end
*
           subroutine SIX(orderperiod,orderdate,ordersize,n,i)
*
           integer period1,period2,period3,period4,period5,period6,
```

```
&        date1,date2,date3,date4,date5,date6,orderperiod,
&        orderdate
      real LR1,LR2,LR3,LR4,LR5,LR6,NR1,NR2,NR3,NR4,NR5,NR6
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      orderdate = date1
      ordersize  = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      i = i + 1
      read(7,*,rec=i)period4,date4,dMPS4,dX04,dSP4,sumDD4,GR4,SR4,
&        OH4,NR4,LR4,OR4
      i = i + 1
      read(7,*,rec=i)period5,date5,dMPS5,dX05,dSP5,sumDD5,GR5,SR5,
&        OH5,NR5,LR5,OR5
      i = i + 1
      read(7,*,rec=i)period6,date6,dMPS6,dX06,dSP6,sumDD6,GR6,SR6,
&        OH6,NR6,LR6,OR6
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3,period4,period5,
&        period6
10601    format('_Period_',6(3x,i3,3x))
      write(6,10602)date1,date2,date3,date4,date5,date6
10602    format('_1st_day',6(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3,dMPS4,dMPS5,dMPS6
10603    format('__MPS___',6(1x,f8.0))
      write(6,10604)dX01,dX02,dX03,dX04,dX05,dX06
10604    format('___X0___',6(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3,dSP4,dSP5,dSP6
10605    format('___SP___',6(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3,sumDD4,sumDD5,sumDD6
10606    format('_sum_DD_',6(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3,GR4,GR5,GR6
10607    format('___GR___',6(1x,f8.0))
      write(6,10608)SR1,SR2,SR3,SR4,SR5,SR6
10608    format('___SR___',6(1x,f8.0))
      write(6,10609)OH1,OH2,OH3,OH4,OH5,OH6
10609    format('___OH___',6(1x,f8.0))
      write(6,10610)NR1,NR2,NR3,NR4,NR5,NR6
10610    format('___NR___',6(1x,f8.0))
     write(6,10611)LR1,LR2,LR3,LR4,LR5,LR6
10611    format('___LR___',6(1x,f8.0))
      write(6,10612)OR1,OR2,OR3,OR4,OR5,OR6
10612    format('___OR___',6(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
```

```
      end
*
      subroutine FIVE(orderperiod,orderdate,ordersize,n,i)
*
     integer period1,period2,period3,period4,period5,date1,date2,
&        date3,date4,date5,orderperiod,orderdate
      real NR1,NR2,NR3,NR4,NR5,LR1,LR2,LR3,LR4,LR5
      n = n + 1
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      orderdate = date1
      ordersize = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      i = i + 1
      read(7,*,rec=i)period4,date4,dMPS4,dX04,dSP4,sumDD4,GR4,SR4,
&        OH4,NR4,LR4,OR4
      i = i + 1
      read(7,*,rec=i)period5,date5,dMPS5,dX05,dSP5,sumDD5,GR5,SR5,
&        OH5,NR5,LR5,OR5
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3,period4,period5
10601    format('_Period_',5(3x,i3,3x))
      write(6,10602)date1,date2,date3,date4,date5
10602    format('_1st_day',5(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3,dMPS4,dMPS5
10603    format('__MPS___',5(1x,f8.0))
      write(6,10604)dX01,dX02,dX03,dX04,dX05
10604    format('___X0___',5(1x,f8.0))
     write(6,10605)dSP1,dSP2,dSP3,dSP4,dSP5
10605    format('___SP___',5(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3,sumDD4,sumDD5
10606    format('_sum_DD_',5(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3,GR4,GR5
10607    format('___GR___',5(1x,f8.0))
      write(6,10608)SR1,SR2,SR3,SR4,SR5
10608    format('___SR___',5(1x,f8.0))
      write(6,10609)OH1,OH2,OH3,OH4,OH5
10609    format('___OH___',5(1x,f8.0))
      write(6,10610)NR1,NR2,NR3,NR4,NR5
10610    format('___NR___',5(1x,f8.0))
      write(6,10611)LR1,LR2,LR3,LR4,LR5
10611    format('___LR___',5(1x,f8.0))
     write(6,10612)OR1,OR2,OR3,OR4,OR5
10612    format('___OR___',5(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
```

```
      end
*
      subroutine FOUR(orderperiod,orderdate,ordersize,n,i)
*
      integer period1,period2,period3,period4,date1,date2,date3,
&        date4,orderperiod,orderdate
      real NR1,NR2,NR3,NR4,LR1,LR2,LR3,LR4
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1
&        OH1,NR1,LR1,OR1
      orderperiod  = period1
      orderdate = date1
      ordersize  = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      i = i + 1
      read(7,*,rec=i)period4,date4,dMPS4,dX04,dSP4,sumDD4,GR4,SR4,
&        OH4,NR4,LR4,OR4
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3,period4
10601    format('_Period_',4(3x,i3,3x))
      write(6,10602)date1,date2,date3,date4
10602    format('_1st_day',4(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3,dMPS4
10603    format('__MPS___',4(1x,f8.0))
      write(6,10604)dX01,dX02,dX03,dX04
10604    format('___XO___',4(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3,dSP4
10605    format('___SP___',4(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3,sumDD4
10606    format('_sum_DD_',4(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3,GR4
10607    format('___GR___',4(1x,f8.0))
      write(6,10608)SR1,SR2,SR3,SR4
10608    format('___SR___',4(1x,f8.0))
      write(6,10609)OH1,OH2,OH3,OH4
10609    format('___OH___',4(1x,f8.0))
      write(6,10610)NR1,NR2,NR3,NR4
10610    format('___NR___',4(1x,f8.0))
      write(6,10611)LR1,LR2,LR3,LR4
10611    format('___LR___',4(1x,f8.0))
      write(6,10612)OR1,OR2,OR3,OR4
10612    format('___OR___',4(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine THREE(orderperiod,orderdate,ordersize,n,i)
```

```
*
      integer period1,period2,period3,date1,date2,date3,
&        orderperiod,orderdate
      real NR1,NR2,NR3,LR1,LR2,LR3
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod  = period1
      orderdate = date1
      ordersize = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i  + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3
10601    format('_Period_',3(3x,i3,3x))
      write(6,10602)date1,date2,date3
10602    format('_1st_day',3(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3
10603    format('__MPS___',3(1x,f8.0))
      write(6,10604)dX01,dX02,dX03
10604    format('___XO___',3(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3
10605    format('___SP___',3(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3
10606    format('_sum_DD_',3(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3
10607    format('___GR___',3(1x,f8.0))
      write(6,10608)SR1,SR2,SR3
10608    format('___SR___',3(1x,f8.0))
      write(6,10609)OH1,OH2,OH3
10609    format('___OH___',3(1x,f8.0))
      write(6,10610)NR1,NR2,NR3
10610    format('___NR___',3(1x,f8.0))
      write(6,10611)LR1,LR2,LR3
10611    format('___LR___',3(1x,f8.0))
      write(6,10612)OR1,OR2,OR3
10612    format('___OR___',3(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine TWO(orderperiod,orderdate,ordersize)
*
      integer period1,period2,date1,date2,orderperiod,orderdate
      real NR1,NR2,LR1,LR2
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
```

```
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      orderdate = date1
      ordersize = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2
10601    format('_Period_',2(3x,i3,3x))
      write(6,10602)date1,date2
10602    format('_1st_day',2(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2
10603    format('__MPS___',2(1x,f8.0))
      write(6,10604)dX01,dX02
10604    format('___XO___',2(1x,f8.0))
      write(6,10605)dSP1,dSP2
10605    format('___SP___',2(1x,f8.0))
      write(6,10606)sumDD1,sumDD2
10606    format('_sum_DD_',2(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2
10607    format('___GR___',2(1x,f8.0))
      write(6,10608)SR1,SR2
10608    format('___SR___',2(1x,f8.0))
      write(6,10609)OH1,OH2
10609    format('___OH___',2(1x,f8.0))
      write(6,10610)NR1,NR2
10610    format('___NR___',2(1x,f8.0))
      write(6,10611)LR1,LR2
10611    format('___LR___',2(1x,f8.0))
      write(6,10612)OR1,OR2
10612    format('___OR___',2(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine ONE(orderperiod,orderdate,ordersize,n,i)
*
      integer period1,date1,orderdate,ordersize
      real LR1,NR1
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      ordersize = OR1
      orderdate = date1
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1
```

```
10601    format('_Period_',3x,i3)
      write(6,10602)date1
10602    format('_1sy_day',3x,i4)
      write(6,10600)
      write(6,10603)dMPS1
10603    format('__MPS___',1x,f8.0)
      write(6,10604)dXO1
10604    format('___XO___',4x,f8.0)
      write(6,10605)dSP1
10605    format('___SP___',4x,f8.0)
      write(6,10606)sumDD1
10606    format('_sum_DD_',4x,f8.0)
      write(6,10600)
      write(6,10607)GR1
10607    format('___GR___',4x,f8.0)
      write(6,10608)SR1
10608    format('___SR___',4x,f8.0)
      write(6,10609)OH1
10609    format('___OH___',4x,f8.0)
      write(6,10610)NR1
10610    format('___NR___',4x,f8.0)
      write(6,10611)LR1
10611    format('___LR___',4x,f8.0)
      write(6,10612)OR1
10612    format('___OR___',4x,f8.0)
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine OrderWrite(period,date,size,leadtime,parttoorder,
&        name)
*
      character part,name
      integer period,date,parttoorder,duedate
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,*,rec=parttoorder)part
      close(unit=2)
      duedate = date + leadtime
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('*'))
      do i = 1,4,1
         write(6,10601)
10601    format('***',56x,'***')
      enddo
      write(6,10602)name
10602    format('***',22x,a12,22x,'***')
      write(6,10603)
10603    format('***',21x,'PURCHASE_ORDER',21x,'***')
         write(6,10601)
         write(6,10601)
      write(6,10604)period
10604    format('***',28x,'Period_placed_',i3,11x,'***')
      write(6,10605)date
10605    format('***',28x,'Date_placed___',i4,10x,'***')
         write(6,10601)
         write(6,10601)
      write(6,10606)part
10606    format('***',7x,'Part_Number_',a10,27x,'***')
```

```
      write(6,10601)
      write(6,10607)size
10607    format('***',7x,'Quantity____',f8.0,29x,'***')
      write(6,10601)
      write(6,10608)duedate
10608    format('***',7x,'Due_Date____',i4,33x,'***')
      do i = 1,4,1
         write(6,10601)
      enddo
      write(6,10600)
      return
      end
*
      subroutine DependentDemands(BofM,buckets,parttoexplode)
*
*  The dependent demands are transferred to MRP matrix.
*
*  Variables:
*    BofM - key to child parts list
*    child - child part on bom record
*    ckey - key to next parent part
*    day - due date of dependent demands
*    DDkey - key to DD records
*    kBOM - record number in bom file
*    kDD - record number in DD file
*    parent - parent part on bom record
*    part - system part number on DD record
*    qDD - number of parent units needed
*    qtyc - number of child/parent needed
*    qtyDD - number of childs needed
*    sumDD - sum total of dependent demands
*
      character c
      integer parent,child,qtyc,pkey,ckey,BofM,s,p,u,d,DDkey,
&        period,date,day,part,buckets,parttoexplode
      real NR,LR
      open(unit=6,file='printer')
      write(6,10600)
10600    format('Dependent Demand sources are as follows:')
*
*  What parents place the demands?
*
      kBOM = BofM
10    open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      read(1,*,rec=kBOM)parent,child,qtyc,pkey,ckey
      close(unit=1)
*
*  Has the parent placed demands?
*
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,*,rec=parent)c,s,k,p,u,d,DDkey
      close(unit=2)
      if DDkey = -1
*
*  No demands this time.
*
         write(6,10601)c
10601    format('Part_',a10,'Produces No Dependent Demands')
*
*  No more parent demands?
```

```
*
         if (ckey = -1) go to 999
         kBOM = ckey
         go to 10
      endif
      write(6,10602)c
10602    format('Part_',a10,5x,'_Date_',5x,'_Quantity_')
      kDD = DDkey
20    open(unit=8,file='(f:17)DD,rel',access='DIRECT',recl=17)
      read(8,*,rec=kDD)part,day,qDD
      close(unit=8)
*
*  Make sure the demands are from parent i.
*
      if (part<>parent)
*
*  No more parent demands?
*
         if (ckey = -1) go to 999
         kBOM = ckey
         go to 10
      endif
      qtyDD = qtyc * qDD
      write(6,10603)day,qtyDD
10603    format(21x,i4,7x,f8.0)
      sumDD = 0
*
*  Find the right time slot for the demands.
*
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         if (day = date)
            sumDD = sumDD + qtyDD
            write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,
&              SR,OH,NR,LR,OR
20712       format(i4,',',i3,10(',',f8.0))
            close(unit=7)
            kDD = kDD + 1
         endif
      close(unit=7)
      enddo
*
*  More demands?
*
      if (part=parent) go to 20
*
*  No more parents?
*
      if (ckey = -1) go to 999
      kBOM = ckey
      go to 10
999   close(unit=6)
      return
      end
*
```

```
*
*  file - System Status
*
*  Karen M. May      Date: June 7, 1983
*
*  function: to print out information from the data files
*
*  Variables:
*    answer - response to questions
*
*  Subprograms: PartsList,sortedparts,unsortedparts,BOMlist,
* MRPflow
*
      character answer
*
*  Type of system status information is requested.
*
10    print,'To obtain flow of explosion,type flow.'
      print,'To obtain list of parts,type part.'
      print,'To obtain list of Bills of Material,type BOM.'
      print,'To stop,type quit.'
      read,answer
      if answer = 'part'
         call PartsList
         go to 10
      elseif answer = 'BOM'
         call BOMlist
         go to 10
      elseif answer = 'flow'
         call MRPflow
         go to 10
      elseif answer  = 'quit'
         go to 999
      else
         go to 10
      endif
999   stop
      end
*
      subroutine PartsList
*
*  The order for the parts list is requested.
*
*  Variables:
*    answer - response to inquiry
*
      character answer
10    print,'To get sorted list,type s.'
      print,'To get as entered list,type u.'
      print,'To stop,type quit.'
      read,answer
      if answer = 's'
         call sortedparts
         go to 10
      elseif answer = 'u'
         call unsortedparts
         go to 10
      elseif answer = 'quit'
         go to 999
      else
```

```
         go to 10
      endif
999   return
      end
*
      subroutine sortedparts
*
*    Parts list is printed out with company part numbers sorted.
*
*  Variables:
*     c - company part number
*     first - company part number with ordinal number one
*     k - directional key to next part
*     key - record read in partid file
*     s - system part number
*
      character c
      integer first,next,s
      open(unit=6,file='printer')
      write(6,10600)
10600    format(5x,'Parts_List')
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10201,rec=0)first
10201    format(i3)
      key = first
10    read(2,*,rec=key)c,s,k
      write(6,10601)c
10601    format(5x,a10)
      if (k=999) go to 999
      key = k
      go to 10
999   close(unit=6)
      close(unit=2)
      return
      end
*
      subroutine unsortedparts
*
*  Parts list is printed out unsorted.  That is as entered.
*
*  Variables:
*     c - company part number
*     first - company part number with ordinal number one
*     i - do loop index; record number read
*     next  - next available record in partid file
*
      character c
      integer first,next
      open(unit=6,file='printer')
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10202,rec=0)first,next
10202    format(2i3)
      write(6,10600)
10600    format(5x,'Parts_List')
      do i = 1,next-1,1
         read(2,*,rec=i)c
         write(6,10601)c
10601    format(5x,a10)
      enddo
      close(unit=6)
```

```
      close(unit=2)
      return
      end
*
      subroutine BOMlist
*
*  The bills-of-materials is printed out by part module.
*
*  Variables:
*     c - child system part number
*     child - company part number for c
*     i - do loop index; record number read
*     next - next available record in bom file
*     p - parent system part number
*     parent - company part number for p
*     pkey - key to next child for parent
*     qty - number of c/p
*
      character parent,child
      integer p,c,qty,pkey
      open(unit=6,file='printer')
      write(6,10600)
10600    format(5x,'Bills of Material')
      write(6,10601)
10601    format(5x,36('-'))
      write(6,10602)
10602    format(5x,'__Parent__',3x,'__Child___',3x,'_Quantity_')
      write(6,10601)
      open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      read(1,10101,rec=0)next
      close(unit=1)
10101    format(i4)
      do i = 1,next-1,1
         open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
         read(1,*,rec=i)p,c,qty,pkey
         close(unit=1)
         open(unit=2,file='(f:35)partid,rel',access='DIRECT')
         read(2,*,rec=p)parent
         read(2,*,rec=c)child
         close(unit=2)
         write(6,10603)parent,child,qty
10603    format(5x,a10,3x,a10,6x,i4)
         if(pkey=-1) write(6,10601)
         enddo
      close(unit=6)
      return
      end
*
      subroutine MRPflow
*
*   The part-by-part flow pattern is printed out.
*
*  Variables:
*     answer - response to question
*     c - company part number
*     first - first part at level=1
*     k - record number read in partstatus
*     level - part level number
*     levelkey - key to the next part to be exploded
*     next - next available record in partstatus
```

```
*
      character c,answer
      integer  first,SR,BOM,BofM,ccper,pup,XOkey,SPkey
10    print,'Has level number been assigned? yes or no.'
      read,answer
      if answer = 'yes'
         go to 20
      elseif answer = 'no'
         go to 999
      else
         go to 10
      endif
20    open(unit=6,file='printer')
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10302,rec=0)next,first
10302    format(2i3)
      close(unit=3)
      write(6,10600)
10600    format(5x,'Parts sorted by MRP flow order.')
      k = first
30    open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=k)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,XOkey,SPkey,level,
&        levelkey
      close(unit=3)
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,*,rec=k)c
      close(unit=2)
      write(6,10601)c
10601    format(10x,a10)
      if (levelkey=-1) go to 999
      k = levelkey
      go to 30
999   close(unit=6)
      return
      end
```

APPENDIX B

PROGRAMS (WORKING).

```
*
*  File: PartFile
*
*  Programmer: Karen M. May     Date: May 11, 1983
*
*  Function: establish and append partid file; initialize
*            partstatus file
*
*  Variables:
*    add - where new additions to partid file start
*    answer - reply to questions
*    down - down directional key for binary search
*    first - part with ordinal number one
*    k,key,keyi - location of next part in ordered list (i=1,2)
*    mid - midpoint; binary search starting point
*    next - next record free for additional parts
*    num - company or user part numbers (character)
*    p - ordinal position of the company part number
*    sys - system or computer part number (integer)
*    up - up directional key for binary search
*
*  Subprograms: setupadd, updown, keycheck, initialize
*
      character answer
10    print,'Is part file to be established? or appended? yes,no'
      read,answer
      if answer = 'yes'
        open(unit=2,file='(f:35)partid,rel',access='DIRECT')
        call setupadd
        call initialize
        call updown
        close(unit=2)
      elseif answer = 'no'
        go to 20
      else
        go to 10
      endif
20    stop
end
*
      subroutine setupadd
*  The routine reads company part numbers,assigns system numbers,
*  and establishes sequential keys for company part numbers.
*  Formats
10203 format(3i3)
203   format(3i3,8x,'Partid Data File')
10204 format(4i3)
204   format(4i3,5x,'Partid Data File')
20203 format(a10,2(',',i3))
      character newnum,oldnum,oldoldnum,answer
      integer first,oldsys,oldoldsys,oldkey,oldoldkey,old,add
5     print,'new or add?'
      read, answer
      if answer = 'new'
         first = 1
         next = 1
         mid = 1
         add = 1
         write(2,00204,rec=0)first,next,mid,add
         newsys = 0
```

```
         newkey = 999
      elseif answer = 'add'
         read(2,10203,rec=0)first,next,mid
         add = next
         write(2,00204,rec=0)first,next,mid,add
         newsys = next - 1
      else
         goto 5
      endif
loop
*     Type in new number. When no more parts, just type quit.
10    print,
      read,newnum
      quitif newnum = 'quit'
      newsys = newsys + 1
      next = next + 1
      write(2,00204,rec=0)first,next,mid,add
*     When the file only has one part,no comparison is necessary.
      if newsys = 1
         write(2,20203,rec=newsys) newnum,newsys,newkey
         goto 10
      endif
*     When more than one part,compare and assign key.
      old = 0
      key = first
20    read(2,*,rec=key)oldnum,oldsys,oldkey
*     Part numbers should not be duplicated.
      if newnum = oldnum
         print,newnum,' has already been used to name a part!!!'
         newsys = newsys - 1
         next = next - 1
         write(2,00204,rec=0)first,next,mid,add
         goto 10
*     New part < old part, key new to old part.
      elseif newnum < oldnum
         newkey = oldsys
         if newkey = first
            first = newsys
            write(2,00204,rec=0)first,next,mid,add
         endif
         if old >= 1
            oldoldkey = newsys
            write(2,20203,rec=oldoldsys)oldoldnum,oldoldsys,
&              oldoldkey
         endif
         write(2,20203,rec=newsys)newnum,newsys,newkey
         goto 10
*     New part > old part.
      else
         if oldkey = 999
            newkey = oldkey
            oldkey = newsys
            write(2,20203,rec=oldsys)oldnum,oldsys,oldkey
            write(2,20203,rec=newsys)newnum,newsys,newkey
            goto 10
         else
            key = oldkey
            oldoldkey = oldkey
            oldoldsys = oldsys
            oldoldnum = oldnum
```

```
            old = old + 1
            goto 20
         endif
      endif
endloop
return
end
*
      subroutine updown
*    The up/down keys are assigned for the binary search.
      character num
      integer pos(999),first,sys,dist,upkey,downkey,up,down,m(30),
&        p,md(30),add
*  Formats
10204 format(4i3)
204    format(4i3,5x,'Partid Data File')
10203 format(3i3)
203    format(3i3,8x,'Partid Data File')
20204 format(a10,3(',',i3))
20206 format(a10,5(',',i3))
*  Initialize variables.
      do i = 1,30,1                                          :mandmd
         m(i) = 0
         md(i) = 0
      enddo                                                  :mandmd
      read(2,10204,rec=0)first,next,mid,add
      key = first
      do i=1,next-1,1                                        :posnumber
         read(2,*,rec=key)num,sys,key
         pos(i) = sys
         write(2,20204,rec=sys)num,sys,key,i
      enddo                                                  :posnumber
      do i = 1,next-1,1                                      :upanddown
         read(2,*,rec=i)num,sys,k,p
         write(2,20206,rec=i)num,sys,k,p,0,0
      enddo                                                  :upanddown
*    Determine the file's midpoint.
      n = next - 1
      mid = (n+1)/2
      md(1) = mid
      mid = pos(mid)
      write(2,00204,rec=0)first,next,mid,add
      jexp = 0
*    Can more keys be assigned by the calculation method.
10    j = 2**jexp
      n = n - (j*2)
      if (n <= 0)                                            :toofew
         call keycheck(next,pos)
         goto 9999
      endif                                                  :toofew
*    Calculate absolute distance to the next key.
      dist = (md(1) + 1)/2
      if (dist < 1)                                          :toosmall
         call keycheck(next,pos)
         goto 9999
      endif                                                  :toosmall
*    Calculate up/down keys; mark destinations with -1.
      do i = 1,j,1
         upkey = md(i) - dist
         downkey = md(i) + dist
```

```
20       if (upkey <= 0)                                    :range
            upkey = upkey + 1
            quitif upkey=md(i)                              :range
            goto 20
         endif                                              :range
30       if (downkey >= next)                               :inrange
            downkey = downkey - 1
            quitif downkey =md(i)                           :inrange
            goto 30
         endif                                              :inrange
40       key1 = pos(upkey)
         read(2,*,rec=key1)num,sys,k,p,up,down
         if (up > 0 .and. down > 0)                         :allkeyed
            upkey = upkey + 1
            quitif upkey = md(i)                            :allkeyed
            goto 40
         endif                                              :allkeyed
50       key2 = pos(downkey)
         read(2,*,rec=key2)num,sys,k,p,up,down
         if (up > 0 .and. down > 0)                         :keyed
            downkey = downkey - 1
            quitif downkey = md(i)                          :keyed
            goto 50
         endif                                              :keyed
*    Store for later calculations.
         m(2*i-1) = upkey
         m(2*i) = downkey
*    Write up/down keys on root and assign -1 to its branches.
         key = pos(md(i))
         upkey = pos(upkey)
         downkey = pos(downkey)
         read(2,*,rec=key)num,sys,k,p
         write(2,20206,rec=key)num,sys,k,p,upkey,downkey
         read(2,*,rec=upkey)num,sys,k,p
         write(2,20206,rec=upkey)num,sys,k,p,-1,-1
         read(2,*,rec=downkey)num,sys,k,p
         write(2,20206,rec=downkey)num,sys,k,p,-1,-1
      enddo
      do i=1,j*2,1                                          :storemi
         md(i) = m(i)
      enddo                                                 :storemi
      jexp = jexp + 1
      goto 10
9999  return
      end
*
      subroutine keycheck(next,pos)
*    All parts are checked for up/down key(i.e.not up=down=0).
*    If not, part is keyed above(if down=-1) or below(if up=-1)?
*    No, then print 'not all records can be keyed' and stop.
      character num
      integer pos(999),upkey,downkey,up,down,sys,p
* Format
20206 format(a10,5(',',i3))
*  Check each record to see if it can be reached.
      do i = 1,next-1,1                                     :allrecords
         key = i
         read(2,*,rec=key)num,sys,k,loc,upkey,downkey
         if (upkey = 0 .and. downkey = 0)                   :notkeyed
            if (loc = 1) goto 20
```

```
            key1 = pos(loc-1)
            read(2,*,rec=key1)num,sys,k,p,up,down
            if (down = -1)                                  :checkabove
*  Can be used for linking below?
               write(2,20206,rec=key1)num,sys,k,p,up,i
               read(2,*,rec=key)num,sys,k,p
               write(2,20206,rec=key)num,sys,k,p,-1,-1
               goto 90
            endif                                           :checkabove
            if (loc = next-1)
               print,'not all records can be keyed'
               stop
            endif
20          key2 = pos(loc + 1)
            read(2,*,rec=key2)num,sys,k,p,up,down
            if (up = -1 )                                   :checkbelow
*  Can be linked above.
               write(2,20206,rec=key2)num,sys,k,p,i,down
               read(2,*,rec=key)num,sys,k,p
               write(2,20206,rec=key)num,sys,k,p,-1,-1
               goto 90
            else                                            :checkbelow
               print,'not all records can be keyed'
               stop
            endif                                           :checkbelow
         endif                                              :notkeyed
90    enddo                                                 :allrecords
999   return
      end
      subroutine initialize
* The following procedure initializes the partstatus file.
      integer add,first
* How many records need to be initialized?
      read(2,10204,rec=0)first,next,mid,add
10204 format(4i3)
* Next available on record 0? Initialize the records.
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      write(3,00301,rec=0)next
301   format(i3,8x,'Partstatus Data File')
      do i = add,next-1,1
         write(3,20319,rec=i)0.,-1,-1,-1,-1,5,1,5,100.,1.,1.,2,
&           100.,4,-1,-1,-1,-1,-1
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      enddo
      close(unit=3)
      return
      end
```

```
*
*  File - BOMfile
*
*  Programmer: Karen M. May      Date: May 20, 1983
*
*  Function - create bill of material(bom) file; append bom;
*         correct data in bom
*
*  Variables:
*    answer - reply to  questions
*    BofM - key to first appearance as child in bom file
*    BOM - key to first appearance as parent in bom file
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time used to express carrying cost
*    child,c,ch,cc - components that make up assembly
*    ckey - bom pointer to next parent of the same child
*    csys - system part number for child
*    lead - lead time in days
*    level - part level
*    levelkey - link for MRP flow
*    lotp - period in days for L4L or FPQ
*    lotr - lotting rule for explosion
*    MPSkey - key to master production schedule demands
*    numkids - number of different components for parent
*    oh - on hand inventory quantity
*    parent,p,pa,pp - assembly
*    pkey - bom pointer to next child of same parent
*    psys - parent's system part number
*    pu - periodal usage (units/unit of time)
*    pup - time unit for periodal usage
*    qty - quantity of child parts/parent part
*    servpartkey - key to service parts demands
*    SR - key to scheduled receipts
*    xorderkey - key to external order demands
*
*  Subprograms: newbom, addbom, findsys, childkeys, changebom,
*     pchange, cchange, qchange
*
      character answer
      open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
10    print,'Does BOM need to be created,appended or existing'
      print,'information altered?type either new,add,or change.'
      read, answer
      if answer = 'new'
         call newbom
         go to 99
      elseif answer = 'add'
         call addbom
         go to 99
      elseif answer = 'change'
         call changebom
         go to 99
      else
         go to 10
      endif
99    close(unit=1)
      close(unit=2)
      close(unit=3)
```

```
      stop
      end
*
      subroutine newbom
*  The bom file is created with this subprogram. List parent &
* number of different children;then, children & quantity.
      character parent,child
      integer pBOMkey,psys,csys,qty,first,ccper,pup,SR,xorderkey,
&        servpartkey
*  Formats
101    format(i4,5x,'Bill-of-Material')
20105 format (2(i3,','),i4,2(',',i4))
20319 format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&        ',',i1,',',f8.0,',',i1,5(',',i3))
*  Initialize
      pBOMkey = 1
      key = 0
      psys = 0
      csys = 0
*  Number of bom records for this parent part?
10    print,'parent,# of children or quit,0 to stop'
      read(5,*) parent,numkids
      if parent = 'quit'                                    :fixckey
*  Link partstatus record BofM and bom record ckey.
         call childkeys
         go to 9999
      endif                                                 :fixckey
*  Find system number for the company(user) number.
      call findsys(parent,psys)
*  Is company part not in partid?
      if (psys=-1) go to 10
*  Assign pointer BOM in the partstatus file.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=psys)oh,SR,num,BOM,BofM,lead,lotr,lotp,scost,
&       ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&       level,levelkey
      write(3,20319,rec=psys)oh,SR,numkids,pBOMkey,-1,lead,lotr,
&       lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&       servpartkey,level,levelkey
      do i=1,numkids,1                                      :readchild
         key = key + 1
         nextkidkey = key + 1
20       print,'child,quantity'
         read(5,*)child,qty
*  Find the child's system number.
         call findsys(child,csys)
* Isn't system number in partid?
         if (csys=-1) go to 20
*  Denote the last child by pkey = -1.
         if (i=numkids) nextkidkey = -1
         write(1,20105,rec=key)psys,csys,qty,nextkidkey,-1
      enddo                                                 :readchild
      pBOMkey = key + 1
      write(1,00101,rec=0) pBOMkey
      go to 10
9999  return
      end
*
      subroutine childkeys
```

```
*  This procedure assigns ckeys in the bom.
      integer p,c,q,SR,BOM,pa,ch,qt,BofM,ccper,pup,xorderkey,
&        servpartkey
*  Format
10101 format(i4)
20105 format(2(i3,','),i4,2(',',i4))
20319 format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&       ',',i1,',',f8.0,',',i1,5(',',i3))
*  Make sure all BofM keys are -1.
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,nc,BOM,BofM,lead,lotr,lotp,scost,
&           ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&           level,levelkey
         write(3,20319,rec=i)oh,SR,nc,BOM,-1,lead,lotr,lotp,scost,
&           ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&           level,levelkey
         close(unit=3)
      enddo
*  How many records are in the bom?
      read(1,10101,rec=0)next
      do i=1,next-1,1                                    :startingpoint
      read(1,*,rec=i)p,c,q,ke,key
*  Has a link already been created?
      if key = -1                                        :lastone
         close(unit=3)
         open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
         read(3,*,rec=c)oh,SR,nc,BOM,BofM,lead,lotr,lotp,scost,
&           ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&           level,levelkey
*  Assign BofM in partstatus record.
         if BofM = -1                                    :firstone
            write(3,20319,rec=c)oh,SR,nc,BOM,i,lead,lotr,lotp,
&              scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&              servpartkey,level,levelkey
            ki=i
*  Other locations where child part appears?
            do ii=i+1,next-1,1                           :search
               read(1,*,rec=ii)pa,ch,qt,kee,keys
               if c = ch                                 :lastread
                  write(1,20105,rec=ki)p,c,q,ke,ii
                  ki=ii
                  p = pa
                  c = ch
                  q = qt
                  ke = kee
               endif                                     :lastread
            enddo                                        :search
         endif                                           :firstone
      endif                                              :lastone
      enddo                                              :startingpoint
      return
      end
      subroutine findsys(part,sys)
*  This algorithm finds system part number(sys) for company
* part number(part).
      character part,u
```

```
      integer sys,first,s,pos,up,down
*  Format
10203 format(3i3)
*  Get the scoop on the partid file
      read(2,10203,rec=0)first,next,mid
*  Try mid(midpoint) of company part list. If not 'part', where?
      k = mid
10    read(2,*,rec=k)u,s,key,pos,up,down
      if part = u                                          :found
         sys = s
         return
      endif                                                :found
      if part < u                                          :whichdirection
         if up = -1                                        :notinfile
            print,part,' not in partid file'
            sys = -1
            return
         endif                                             :notinfile
         k = up
         go to 10
      else                                                 :whichdirection
         if down = -1                                      :infileornot
            print,part,' not in partid file'
            sys = -1
            return
         endif                                             :infileornot
         k = down
         go to 10
      endif                                                :whichdirection
      return
      end
*
      subroutine addbom
*  Additional records are included in bom file.
      integer q,where,psys,csys,BOM,BofM,SR,pp,cc,qq,ppkey,cckey,
&        add,ccper,pup,xorderkey,servpartkey
      character p,c
*  Format
101    format(i4,5x,'Bills-of-Material')
10101 format(i4)
20319 format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&         ',',i1,',',f8.0,',',i1,5(',',i3))
20105 format(2(i3,','),i4,2(',',i4))
      read(1,10101,rec=0)next
      do i = 1,next-1,1                                    :resetcckey
         read(1,*,rec=i)pp,cc,qq,ppkey,cckey
         write(1,20105,rec=i)pp,cc,qq,ppkey,-1
      enddo                                                :resetcckey
      where = next - 1
10    print,'parent,child,qty to be added or quit,q,0 to stop'
      read(5,*)p,c,q
      if p = 'quit'                                        :nomore
         next = where + 1
         write(1,00101,rec=0)next
         call childkeys
         go to 999
      endif                                                :nomore
*  Find system numbers.
      call findsys(p,psys)
      call findsys(c,csys)
```

```
*   Can't find one of the system numbers?
      if (psys = -1.or.csys = -1) go to 10
*   Next empty record?
      where = where + 1
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=psys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*   New parent? Yes, if numkids = -1.
      if (numkids <> -1) numkids = numkids + 1
      if (numkids = -1) numkids = 1
*   Where are the other children?
      if BOM = -1                                          :previous listing
         write(3,20319,rec=psys)oh,SR,numkids,where,BofM,lead,
&           lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         write(1,20105,rec=where)psys,csys,q,-1,-1
         go to 10
      else                                                 :previous listing
         write(3,20319,rec=psys)oh,SR,numkids,BOM,BofM,lead,
&           lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         l = BOM
20       read(1,*,rec=l)pp,cc,qq,ppkey,cckey
         if ppkey = -1                                     :last child
            write(1,20105,rec=l)pp,cc,qq,where,cckey
            write(1,20105,rec=where)psys,csys,q,-1,-1
            go to 10
         else                                              :last child
            BOM = BOM + 1
            l = ppkey
            go to 20
         endif                                             :last child
      endif                                                :previous listing
999   return
      end
*
      subroutine changebom
*   This routine directs the changes made to the bom file.
      character oldp,oldc,newp,newc
      integer oldpsys,oldcsys,oldq,find
*   What record to change?
10    print,'Enter parent,child,qty as it now appears'
      print,'or quit,q,0 to stop.'
      read(5,*)oldp,oldc,oldq
*   No more corrections?
      if (oldp = 'quit') go to 999
*   Search for system numbers.
      call findsys (oldp,oldpsys)
      call findsys (oldc,oldcsys)
*   Can't find one of the system numbers?
      if (oldpsys = -1.or.oldcsys = -1) go to 10
*   How to change the bom record?
20    print,'Enter parent,child,qty as it should appear'
      print,'or quit,q,0 to stop.'
      read(5,*)newp,newc,newq
*   Old material entered wrong?
      if (newp = 'quit') go to 10
*   Search for system numbers.
```

```
      call findsys(newp,newpsys)
      call findsys(newc,newcsys)
      if (newpsys = -1.or.newcsys = -1) go to 20
*   The parent,child pair is in the bom file?
      find = 1
*   Change parent part?
      if(oldp <> newp) call pchange(oldpsys,oldcsys,newpsys,find)
      if (find=0) go to 10
      find = 1
*   Change child part?
      if (oldc <> newc) call cchange(newpsys,oldcsys,newcsys,find)
      if (find=0) go to 10
      find = 1
*   Change quantity child per parent?
      if (oldq <> newq) call qchange(newpsys,newcsys,newq,find)
      go to 10
999   return
      end
*
      subroutine pchange(oldpsys,oldcsys,newpsys,find)
*   The parent part is changed with this subprogram.
      integer SR,BOM,BofM,ccper,pup,p,c,q,pkey,ckey,c1,q1,pkey1,
&        oldpsys,oldcsys,oldq,find,ckey1,xorderkey,servpartkey
*   Format
20105    format(2(i3,','),i4,2(',',i4))
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*   Find old parent,child pair in the bom file.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=oldpsys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*   How many reads were needed to find pair.
      n = 0
*   BOM is first bom with parent. If BOM=-1, no children.
      if BOM = -1
         find = 0
         print,'old parent,child pair is not in BOM file'
         go to 999
      endif
      key = BOM
30    read(1,*,rec=key)p,c,q,pkey,ckey
      if oldcsys <> c                                  :findchild
         if pkey = -1                                  :infileornot
            find = 0
            print,'old parent,child pair not in BOM file'
            go to 999
         else                                          :infileornot
            n = n + 1
            key1 = key
            key = pkey
            c1 = c
            q1 = q
            pkey1 = -1
            ckey1 = ckey
            go to 30
         endif                                         :infileornot
      endif                                            :findchild
*   Found parent,child pair. Old parent = New parent.
```

```
      numkids = numkids - 1
      if (numkids = 0) numkids = -1
      write(3,20319,rec=oldpsys)oh,SR,numkids,BOM,BofM,lead,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if (pkey<>-1) pkey1 = pkey
      if (n>0) write(1,20105,rec=key1)p,c1,q1,pkey1,ckey1
*  Let child be last child for new parent.
      write(1,20105,rec=key)newpsys,c,q,-1,ckey
      read(3,*,rec=newpsys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if BOM = -1                                          :previous listing
         write(3,20319,rec=newpsys)oh,SR,1,key,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         go to 999
      else                                                 :previous listing
         numkids = numkids + 1
         write(3,20319,rec=newpsys)oh,SR,numkids,BOM,BofM,lead,
&           lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         k = BOM
*  Link new child to other new parent children.
40       read(1,*,rec=k)p,c,q,pkey,ckey
         if pkey = -1                                      :lastchild
            write(1,20105,rec=k)p,c,q,key,ckey
            go to 999
         else                                              :lastchild
            k = pkey
            go to 40
         endif                                             :lastchild
      endif                                                :previous listing
999   return
      end
*
      subroutine cchange(newpsys,oldcsys,newcsys,find)
*  The child part in the bom is changed with this subprogram.
      integer SR,BOM,BofM,ccper,pup,p,c,q,pkey,ckey,find,oldcsys,
&        oldq,xorderkey,servpartkey
*  Format
10101    format(i4)
10301    format(i3)
20105    format(2(i3,','),i4,2(',',i4))
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*  Find parent,child pair.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=newpsys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if BOM = -1
         find = 0
         print,'old parent,child pair is not in bom file'
         go to 999
      endif
      key = BOM
10    read(1,*,rec=key)p,c,q,pkey,ckey
*  Child found? No more children if pkey = -1.
```

```
      if  oldcsys <> c                                    :findoldchild
         if pkey = -1                                     :infileornot
            find = 0
            print,'old parent,child pair is not in BOM file'
            go to 999
         else                                             :infileornot
            key = pkey
            go to 50
         endif                                            :infileornot
      endif                                               :findoldchild
*  Pair has been found.
      write(1,20105,rec=key)p,newcsys,q,pkey,ckey
*  Reset ckey to -1 before calling childkeys.
      read(1,10101,rec=0) next
      do i = 1,next-1,1                                   :resetckey
         read(1,*,rec=i)p,c,q,pkey,ckey
         write(1,20105,rec=i)p,c,q,pkey,-1
      enddo                                               :resetckey
      call childkeys
999   return
      end
*
      subroutine qchange(newpsys,newcsys,newq,find)
*  The algorithm corrects quantity on bom record.
      integer find,SR,BOM,p,c,q,pkey,ckey
*  Format
20105    format(2(i3,','),i4,2(',',i4))
*  Find the old parent,child pair.
      read(3,*,rec=newpsys)oh,SR,numkids,BOM
      if BOM = -1
         find = 0
         print,'old parent,child pair is not in BOM file'
         go to 999
      endif
      key = BOM
10    read(1,*,rec=key)p,c,q,pkey,ckey
*  When pair is found, change the quantity.
      if newcsys = c                                      :findchild
         write(1,20105,rec=key)newpsys,newcsys,newq,pkey,ckey
         go to 999
      else                                                :findchild
         if pkey = -1                                     :infileornot
            find = 0
            print,'old parent,child pair not in BOM file'
            go to 999
         else                                             :infileornot
            key = pkey
            go to 10
         endif                                            :infileornot
      endif                                               :findchild
999   return
      end
```

```
*
*  File - DataPlus
*
*  Karen M. May      Date: May 31, 1983
*
*  Function - to edit lead time, lotting rules, and dates
* for the planning horizon
*
*  Variables:
*     answer - response to questions
*     BofM - key to first appearance as child part
*     BOM - key to first appearance as parent part
*     ccost - carrying cost ($/unit of time)
*     ccper - unit of time for expressing carrying cost
*     fillin - empty records from 1-next
*     lead - lead time (in days)
*     level - part level
*     levelkey - link for MRP flow
*     lotp - time in days for L4L or FPQ
*     lotr - lotting rule for explosion
*     MPSkey - key to master production schedule demands
*     next - next free record
*     numkids - number of different components in parent
*     oh - on hand inventory
*     pu - periodal usage(units/unit of time)
*     pup - time unit for periodal usage
*     servpartkey - key to service parts demands
*     SR - key to scheduled receipts
*     xorderkey - key to external orders
*
*  Subprograms: findsys,IandSR,ISRnew,srchange,dayorqty,delSR,
* addSR,ohchange,leadandlot,leadtimeandlotrule,leadtime,lotrule,
* horizon,equalbucket,unequalbucket
*
      character answer
10    print,'What data do you wish to enter?'
      print,'Inventory and/or scheduled receipts,type invandsr.'
      print,'Lead time and lotting rules,type ltandlr.'
      print,'Periods for the planning horizon, type ph.'
      print,'To stop, type quit.'
      read, answer
      if answer = 'invandsr'
         call IandSR
         go to 10
      elseif answer = 'ltandlr'
         call leadandlot
         go to 10
      elseif answer =  'ph'
         call horizon
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   stop
      end
*
      subroutine findsys(part,sys)
*  This algorithm finds the system number (sys) for the company
```

```
* number (part).
      character part,u
      integer sys,first,s,pos,up,down
*  Format
10203    format(3i3)
*  Check statistics on partid file
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10203,rec=0)first,next,mid
*  Try midpoint first
      k = mid
10    read(2,*,rec=k)u,s,key,pos,up,down
      if part = u                                          :found
         sys = s
         close(unit=2)
         return
      endif                                                :found
      if part < u                                          :whichdirection
         if up = -1                                        :notinfile
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :notinfile
         k = up
         go to 10
      else                                                 :whichdirection
         if down = -1                                      :infileornot
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :infileornot
         k = down
         go to 10
      endif                                                :whichdirection
      close(unit=2)
      return
      end
*
      subroutine IandSR
*  The user specifies what editing is to be done.
      character answer
10    print,'To create SR file and initialize oh hand inventory,'
      print,'type new. To change SR file data,type SR.'
      print,'To change on hand inventory quantities,type onhand.'
      print,'To stop,type quit.'
      read,answer
      if answer = 'new'
         call ISRnew
         go to 10
      elseif answer = 'SR'
         call srchange
         go to 10
      elseif answer = 'onhand'
         call ohchange
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
```

```
      endif
999   return
      end
*
      subroutine ISRnew
*  Both oh(on hand inventory) and SR(scheduled receipts) are
* entered.
      character part,answer
      integer sr,oh,SR,BOM,BofM,ccper,pup,day,SRkey,sys,fillin,
&        xorderkey,servpartkey
      real inv
*  Formats
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
20403    format(i4,',',f8.0,',',i4)
402      format(2i4,5x,'SR')
10402    format(2i4)
10301    format(i3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=4,file='(f:18)SR,rel',access='DIRECT',recl=18)
      read(3,10301,rec=0)next
      close(unit=3)
*  Set (reset) oh and SR to default values.
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)0.,-1,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         close(unit=3)
      enddo
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
*  Keep track of empty records between 1 and next.
      fillin = 0
*  Where can the new SR be written?
      SRkey = 0
20    print,'part,inventory quantity,# of SR or quit,0,0 to stop'
      read(5,*)part,inv,sr
*  No more?
      if (part = 'quit') go to 999
*  Find system number.
      call findsys(part,sys)
*  Part not in partid file?
      if (sys = -1) go to 20
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
*  No scheduled receipts expected?
      if sr = 0                                          :SRornot
         write(3,20319,rec=sys)inv,-1,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         go to 20
      else                                               :SRornot
         do i = 1,sr,1                                   :writeSR
            SRkey = SRkey + 1
```

```
*  Link partstatus to SR file.
           if(i=1) write(3,20319,rec=sys)inv,SRkey,numkids,BOM,
&             BofM,lead,lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&             MPSkey,xorderkey,servpartkey,level,levelkey
*  Get the specifics on the SR.
           print,'day expected,quantity expected'
           read(5,*) day,qty
*  Link all SR for given part together.
           key = SRkey + 1
           if (i=sr) key = -1
           write(4,20403,rec=SRkey)day,qty,key
           next = SRkey + 1
*  List the next available record for SR file.
           write(4,00402,rec=0)next,fillin
        enddo                                              :writeSR
        go to 20
     endif                                                 :SRornot
999  close(unit=3)
     close(unit=4)
     return
     end
*
     subroutine srchange
*  Changes for the SR file are specified.
     character answer
10   print,'To change record in SR file,type change.'
     print,'To delete record in SR file,type delete.'
     print,'To include new records in SR,type add.'
     print,'To stop,type quit.'
     read,answer
     if answer = 'change'
        call dayorqty
        go to 10
     elseif answer = 'delete'
        call delSR
        go to 10
     elseif answer = 'add'
        call addSR
        go to 10
     elseif answer = 'quit'
        go to 999
     else
        go to 10
     endif
999  return
     end
*
     subroutine dayorqty
*  The quantity &/or expected due date are changed.
     character part
     integer oldday,SR,BOM,BofM,sys
     real newqty
*  Formats
20403   format(i4,',',f8.0,',',i4)
     open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
     open(unit=4,file='(f:18)SR.rel',access='DIRECT',recl=18)
*  Which part has SR to change?
10   print,'part to be changed or quit to stop'
     read,part
*  No more changes?
```

```
      if (part = 'quit') go to 999
*   Find system number.
      call findsys(part,sys)
*   Can't find part?
      if (sys=-1) go to 10
*   What SR needs changing?
      print,'day,qty of SR as it now appears'
      read(5,*) oldday,oldqty
*   How should SR be changed?
      print,'day,qty of SR as it should appear'
      read(5,*) newday,newqty
*   Find the record to be changed.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR
      if (SR = -1)
*   If SR = -1, there are no SR's for the given part.
         print,'old SR not in file'
         go to 10
      endif
      k = SR
*   Find old SR?
20    read(4,*,rec=k)day,qty,key
*   Make the change.
      if day = oldday .and. qty = oldqty
         write(4,20403,rec=k) newday,newqty,key
         go to 10
      else
         if key = -1
*   When key = -1, no more records for part.
            print,'old SR not in file'
            go to 10
         else
*   Try record.
            k = key
            go to 20
         endif
      endif
999   close(unit=3)
      close(unit=4)
      return
      end
*
      subroutine delSR
*   SR are deleted with this subprogram.
      character part
      integer oldday,SR,BOM,BofM,ccper,pup,day,day1,fillin,d1,sys,
&        xorderkey,servpartkey
*   Formats
402       format(2i4,5x,'SR')
10402     format(2i4)
20403     format(i4,',',f8.0,',',i4)
20319     format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=4,file='(f:18)SR,rel',access='DIRECT',recl=18)
*   What part has the SR to be deleted?
10    print,' part to be deleted or quit to stop'
      read,part
*   No more SRs to be deleted?
```

```
      if (part = 'quit') go to 999
*   Find system number.
      call findsys(part,sys)
*   Can't find part on file?
      if (sys = -1) go to 10
      print,'day,qty as it now appears in SR file'
*   Which SR is deleted?
      read(5,*)oldday,oldqty
*    Find SR to be deleted.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM, lead, lotr, lotp,scost,
&        ucost,ccost,ccper, pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
*   Number of records read to find SR.
      n = 0
      if  SR = -1                                          :anyonfile
*   If SR = -1, there aren't any SR on file.
         print,'old SR not in file'
         go to 10
      endif                                                :anyonfile
      k = SR
*   Find old SR.
20    read(4,*,rec=k)day,qty,key
      if day = oldday .and. qty = oldqty                   :findoldSR
         write(4,20403,rec=k)0,0.,-1
*   Link last record to next one in the file.
         if (n > 0) write(4,20403,rec=k1) day1,qty1,key
*   Delete first SR? Key partstatus record to second record.
         if (n = 0) write(3,20319,rec=sys)oh,key,numkids,BOM,BofM,
&           lead,lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&           xorderkey,servpartkey,level,levelkey
         read(4,10402,rec=0)next,fillin
*   Are there any empty record?
         if fillin = 0                                     :otherempties
*   Where is first empty record.
            write(4,00402,rec=0)next,k
            go to 10
         else                                              :otherempties
            kf = fillin
*   Link empty records together.
30          read(4,*,rec=kf)d1,q1,k1
            if k1 = -1                                     :anymore
               write(4,20403,rec=kf)d1,q1,k
               go to 10
            else                                           :anymore
               kf = k1
               go to 30
            endif                                          :anymore
         endif                                             :otherempties
      else                                                 :findoldSR
*   What record to check next?
         if key = -1                                       :more
            print,'old SR not in file'
            go to 10
         else                                              :more
            k1 = k
            k = key
            day1 = day
            qty1 = qty
```

```
            n = n + 1
            go to 20
         endif                                          :more
      endif                                             :findoldSR
999   close(unit=3)
      close(unit=4)
      return
      end
*
      subroutine addSR
*  As new SR are made, they can be included in the SR,rel file
* with this routine.
      character part
      integer fillin,SR,BOM,BofM,ccper,pup,d1,sys,xorderkey,
&        servpartkey
      real newqty
*  Formats
402      format(2i4,5x,'SR')
10402    format(2i4)
20403    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=4,file='(f:18)SR,rel',access='DIRECT',recl=18)
*  Part ordered?
10    print,' part to be added or quit to stop'
      read, part
*  No more orders?
      if (part = 'quit') go to 999
*  Get system number.
      call findsys(part,sys)
*  Can't find the part?
      if (sys = -1) go to 10
*  New order.
      print, ' day, qty '
      read(5,*) newday,newqty
*  Find the empty records.
      read(4,10402,rec=0)next,fillin
*  Other SR's.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM, lead, lotr, lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      if fillin = 0                                     :empties
         key = next
         write(4,20403,rec=key)newday,newqty,-1
         next = next + 1
         write(4,00402,rec=0)next,fillin
*  Link new SR to old SRs.
         if SR = -1                                     :listed
            write(3,20319,rec=sys)oh,key,numkids,BOM,BofM, lead,
&              lotr, lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&              xorderkey,servpartkey, level, levelkey
            go to 10
         else                                           :listed
            k = SR
20          read(4,*,rec=k)d1,q1,k1
            if k1 = -1                                  :anymore
               write(4,20403,rec=k)d1,q1,key
```

```
               go to 10
            else                                                :anymore
               k = k1
               go to 20
            endif                                               :anymore
         endif                                                  :listed
      else                                                      :empties
         read(4,*,rec=fillin)d,q,k
         write(4,20403,rec=fillin)newday,newqty,-1
*  Aren't any more empty records?
         if (k = -1) k = 0
         write(4,00402,rec=0)next,k
*  No other SRs if SR = -1.
         if SR = -1                                             :anylisted
            write(3,20319,rec=sys)oh,fillin,numkids,BOM,BofM,lead,
&              lotr,lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,
&              xorderkey,servpartkey,level,levelkey
            go to 10
         else                                                   :anylisted
            k = SR
30          read(4,*,rec=k)d1,q1,k1
            if k1 = -1                                          :more
               write(4,20403,rec=k)d1,q1,fillin
               go to 10
            else                                                :more
               k = k1
               go to 30
            endif                                               :more
         endif                                                  :anylisted
      endif                                                     :empties
999   close(unit=3)
      close(unit=4)
      return
      end
*
      subroutine ohchange
* The oh value is corrected with the following routine.
      character part
      integer SR,BOM,BofM,ccper,pup,sys,xorderkey,servpartkey
*  Formats
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
*  What record needs to be changed? To what quantity?
10    print,'part,on hand quantity or quit,0 '
      read(5,*) part,qty
*  No more?
      if (part = 'quit') go to 999
*  Find system number.
      call findsys(part,sys)
*  Can't find the part?
      if (sys  = -1) go to 10
*  Change only oh(on hand inventory).
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      write(3,20319,rec=sys)qty,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
```

```
&       servpartkey,level,levelkey
      go to 10
999   close(unit=3)
      return
      end
*
      subroutine leadandlot
*  The type of lead time and lot rule editing is specified.
      character answer
10    print,'To enter both lead time and lotting rule,type ltlr.'
      print,'To enter just lead time, type lt.'
      print,'To enter just lotting rule, type lr.'
      print,'To stop, type quit.'
      read, answer
      if answer = 'ltlr'
         call leadtimeandlotrule
         go to 10
      elseif answer = 'lt'
         call leadtime
         go to 10
      elseif answer = 'lr'
         call lotrule
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
      subroutine leadtimeandlotrule
*  The lead time and lotting rules are supplied using routine.
      character part,lotrule,timeunit,timeperiod
      integer SR, BOM,BofM,ccper,pup,sys,xorderkey,servpartkey
*  Format
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      print,'Leadtime was initialized at 5 days.'
      print,'Lotting rule was initialized at L4L(lot-for-lot).'
10    print,'part,lead time(days), lot rule(L4L,FPQ)'
      print,' or quit,0,q to stop.'
      read(5,*) part,leadtime,lotrule
*  No more?
      if (part='quit') go to 999
*  Find system number.
      call findsys(part,sys)
*  Part not on file?
      if (sys = -1) go to 10
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      lead = leadtime
*  Need additional information for lotting rule.
      if lotrule = 'L4L'
         lotr = 1
         lotp = 5
```

```
      elseif lotrule = 'FPQ'
         lotr = 2
         print,'period(days) for FPQ'
         read,lotp
      else
         go to 10
      endif
   List data on partstatus record.
      write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      go to 10
999   close(unit=3)
      return
      end
*
      subroutine leadtime
*  The lead times are edited with the following program.
      character part
      integer sys,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey
*  Format
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      print,'Lead time was initialized at 5 days.'
10    print,'part,lead time(days), or quit,0 to stop.'
      read(5,*)part,Leadtime
*  No more information?
      if (part = 'quit') go to 999
*  Find system number.
      call findsys(part,sys)
*  Part isn't on file?
      if (sys=-1) go to 10
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
      write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,Leadtime,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      go to 10
999   close(unit=3)
      return
      end
*
      subroutine lotrule
*  The lotting rules are edited with the following subprogram.
      character part,Lotrule,timeunit,timeperiod
      integer sys,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey
*  Format
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      print,'Lotting rule was initialized at L4L(lot-for-lot).'
10    print,'part,lotrule(L4L,FPQ) or quit,q to stop.'
      read(5,*) part,Lotrule
*  No more addition information?
      if (part='quit') go to 999
*  Fetch system number.
```

```
      call findsys(part,sys)
*   Part isn't on file?
      if (sys = -1) go to 10
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&        ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,servpartkey,
&        level,levelkey
*   Need additional information for the lotting rule.
      if Lotrule = 'L4L'
         lotr = 1
         lotp = 5
      elseif Lotrule = 'FP0'
         lotr = 2
         print,'period(days) for FP0'
         read,lotp
      else
         go to 10
      endif
*   List information on partstatus record.
      write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      go to 10
999   return
      end
*
      subroutine horizon
*   The dates for the planning horizon are entered.
      character answer
      integer buckets
*   Specify number of buckets(periods) in the planning horizon.
      print,'number of buckets?'
      read,buckets
*   Specify the current period.
      print,'Start with period number.'
      read,nperiod
*   Choice of equal (1 week/bucket) or unequal bucket sizes.
10    print,'Equal bucket size? yes or no.'
      read, answer
      if answer  = 'yes'
         call equalbucket(buckets,nperiod)
      elseif answer = 'no'
         call unequalbucket(buckets,nperiod)
      else
         go to 10
      endif
      return
      end
*.
      subroutine equalbucket(buckets,nperiod)
*   The computer assigns to each period(1 week/bucket).
      integer buckets,dayone
*   Formats
701      format(i3,5x,'Planning Horizon')
20702    format(i3,',',i4)
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
*   List number of priods in planning horizon.
      write(7,00701,rec=0) buckets
      nperiod = nperiod -1
```

```
*  Assign day one to each bucket (1 week = 5 days).
      do i = 1,buckets,1
         nperiod = nperiod + 1
         dayone = nperiod * 5
         write(7,20702,rec=i) nperiod,dayone
      enddo
      close(unit=7)
      return
      end
*
      subroutine unequalbucket(buckets,nperiod)
*  The computer assigns dates as spedifiecd by the user.
      integer buckets,dayone
*  Formats
701      format(i3,5x,'Planning Horizon')
20702    format(i3,',',i4)
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
*  List number of periods in planning horizon.
      write(7,00701,rec=0) buckets
      nperiod = nperiod - 1
*  Ask for date (day one) for each bucket.
      do i = 1,buckets,1
         nperiod = nperiod + 1
         print,'Period ',nperiod,' day one is ____'
         read, dayone
         write(7,20702,rec=i)nperiod,dayone
      enddo
      close(unit=7)
      return
      end
*
```

```
*
*  File - DataPlus2
*
*  Karen M. May      Date: June 1, 1983
*
*  Function - to edit MPS,XO, and SP demands
*
*  Variables:
*    answer - response to inquiries
*    BofM - key to first appearance as child part
*    BOM - key to first appearance as parent part
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time for carrying cost
*    fillin - empty lines between record 1 and next
*    lead - lead time (in days)
*    lotp - time in days for L4L or FPQ
*    lotr - lotting rule
*    next - next free record in the file
*    numkids - number of different components in parent
*    oh - on hand inventory
*    pu - periodal usage(units/unit of time)
*    pup - time unit for periodal usage
*    SR - key to scheduled receipts
*
*  Subprograms: findsys,MSdemands,MSnew,MSadd,MSchange,XOdemands,
* XOnew,XOadd,XOchange,SPdemands,SPnew,SPadd,SPchange
*
      character answer
10    print,'To enter MS demands, type MS.'
      print,'To enter external orders, type  XO.'
      print,'To enter service parts, type SP.'
      print,'To stop, type quit.'
      read, answer
      if answer = 'MS'
         call MSdemands
         go to 10
      elseif answer = 'XO'
         call XOdemands
         go to 10
      elseif answer =  'SP'
         call SPdemands
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   stop
      end
*
      subroutine findsys(part,sys)
*  This algorithm finds the system number(sys) for the company
* part number(part).
      character part,u
      integer sys,first,s,pos,up,down
*  Format
10203    format(3i3)
*  Check statistics on partid file
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10203,rec=0)first,next,mid
```

```
*  Try midpoint first
      k = mid
10    read(2,*,rec=k)u,s,key,pos,up,down
      if part = u                                          :found
         sys = s
         close(unit=2)
         return
      endif                                                :found
      if part < u                                          :whichdirection
         if up = -1                                        :notinfile
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :notinfile
         k = up
         go to 10
      else                                                 :whichdirection
         if down = -1                                      :infileornot
            print,part,' not in partid file'
            sys = -1
            close(unit=2)
            return
         endif                                             :infileornot
         k = down
         go to 10
      endif                                                :whichdirection
      close(unit=2)
      return
      end
*
      subroutine MSdemands
*  The type of editing for the MS file is specified.
      character answer
10    print,'new,add,change, or quit MS demand list?'
      read,answer
      if answer = 'new'
         call MSnew
         go to 10
      elseif answer = 'add'
         call MSadd
         go to 10
      elseif answer = 'change'
         call MSchange
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
* .
      subroutine MSnew
*  The following routine is used to initialize the MPS file.
      character part
      integer SR,BOM,BofM,ccper,pup,xorderkey,serupartkey,sys,
&        date,Da
*  Formats
10301    format(i3)
```

```
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
901      format(i4,5x,'MPS')
20903    format(i4,',',f8.0,',',i4)
*  Reset the MPSkey to -1.
     open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
     read(3,10301,rec=0)next
     close(unit=3)
     do i = 1,next-1,1
        open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
        read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&          scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&          servpartkey,level,levelkey
        write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,
&          lotp,scost,ucost,ccost,ccper,pu,pup,-1,xorderkey,
&          servpartkey,level,levelkey
        close(unit=3)
     enddo
*  Set the pointer to zero.
     mskey = 0
     keyms = 0
*  What part has MPS demand?
     open(unit=9,file='(f:18)MPS.rel',access='DIRECT',recl=18)
10   print,'part or quit to stop'
     read,part
*  No more entries?
     if (part='quit') go to 999
*  Locate the system number for the part.
     call findsys(part,sys)
*  The part isn't in the files?
     if (sys=-1) go to 10
*  Get the scoop on the part.
     open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
     read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&       scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&       servpartkey,level,levelkey
*  Set the loop counter.
     n = 0
*
     loop
     n = n + 1
*  Get the specifics about the demand.
     print,'date,demand or -1,0 to stop'
     read(5,*)date,demand
*  No more demands for part i? Yes, set last record's key to -1.
     if (date=-1)
        write(9,00901,rec=0)keyms
        read(9,*,rec=mskey)Da,D,k
        write(9,20903,rec=mskey)Da,D,-1
        go to 10
     endif
*  No, increment to next record.
     mskey = mskey + 1
*  Link partstatus record to MPS record.
     if (n=1)
        write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&          lotp,scost,ucost,ccost,ccper,pu,pup,mskey,xorderkey,
&          servpartkey,level,levelkey
        close(unit=3)
     endif
```

```
*   Increment key to next record.
      keyms = mskey + 1
      write(9,20903,rec=mskey)date,demand,keyms
      end loop
      go to 10
999   close(unit=9)
      return
      end
*
      subroutine MSadd
*   The following allows the user to append the MPS file.
      character part
      integer Da,date,sys,SR,BOM,BofM,ccper,pup,xorderkey,
&        servpartkey
*   Formats
901       format(i4,5x,'MPS')
10901     format(i4)
20903     format(i4,',',f8.0,',',i4)
20319     format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=9,file='(f:18)MPS,rel',access='DIRECT',recl=18)
      read(9,10901,rec=0)next
*   Get additions info (part,etc.).
10    print,'part,date,demand quantity or quit,0,0 to stop'
      read(5,*)part,date,demand
*   No more additions?
      if (part='quit') go to 999
*   Locate the system number.
      call findsys(part,sys)
*   Part not in the files?
      if (sys=-1) go to 10
      write(9,20903,rec=next)date,demand,-1
*   Link the old demands to the new demand.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*   If MPSkey = -1, no previous demands exist.
      if MPSkey = -1
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,next,xorderkey,
&           servpartkey,level,levelkey
         next = next + 1
         write(9,00901,rec=0)next
         go to 10
      else
*   Find the last record for part i.
         k = MPSkey
20       read(9,*,rec=k)Da,D,key
*   The last record has key = -1.
         if key = -1
            write(9,20903,rec=k)Da,D,next
            next = next + 1
            write(9,00901,rec=0)next
            go to 10
         else
            k = key
            go to 20
```

```
        endif
      endif
999   close(unit=3)
      close(unit=9)
      return
      end
*
      subroutine MSchange
*  With this subprogram, the user may correct MPS input errors.
      character part
      real newdemand
      integer olddate,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        sys,date
*  Formats
20903    format(i4,',',f8.0,',',i4)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      open(unit=9,file='(f:18)MPS.rel',access='DIRECT',recl=18)
*  Get information on part to be changed.
10    print,'part,date,demand quantity to be changed'
      print,'  or quit,0,0  to stop'
      read(5,*)part,olddate,olddemand
*  No more changes?
      if(part='quit') go to 999
*  Locate the system number for part i.
      call findsys(part,sys)
*  Part isn't in the file?
      if(sys=-1) go to 10
*   How demand information is to be changed?
      print,'correct date, correct demand quantity'
      read(5,*)newdate,newdemand
*  Find the record that needs changing.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*  If MPSkey = -1, there are no records on file for part i.
      if MPSkey = -1
         print,'old date and demand not on file'
         go to 10
      else
*  Start search with record=MPSkey.
         k = MPSkey
20       read(9,*,rec=k)date,demand,key
         if (date=olddate.and.demand=olddemand)
*  When the old demand is found, make the change.
            write(9,20903,rec=k)newdate,newdemand,key
            go to 10
         else
            if key = -1
*  If key = -1, there are no more records for part i.
               print,'old date and demand not on file'
               go to 10
            else
*  Continue search with next record=key.
               k = key
               go to 20
            endif
         endif
      endif
```

```
999   close(unit=3)
      close(unit=9)
      return
      end
*
      subroutine  XOdemands
*  The type of editing for the External Order(XO) file is
* specified.
      character answer
10    print,'new,add,change, or quit XO demand list?'
      read,answer
      if answer = 'new'
         call XOnew
         go to 10
      elseif answer  = 'add'
         call XOadd
         go to 10
      elseif answer  = 'change'
         call XOchange
         go to 10
      elseif answer  = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
      subroutine XOnew
*  A XO file is initialized with the following subprogram.
      character part
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        sys,date,Da,xokey
*  Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
1001     format(i4,5x,'XO')
21003    format(i4,',',f8.0,',',i4)
*  Reset the xorderkey to -1.
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,-1,servpartkey,
&           level,levelkey
         close(unit=3)
      enddo
*  Begin with the pointer and record number equal to 0.
      xokey = 0
      keyxo = 0
      open(unit=10,file='(f:18)XO,rel',access='DIRECT',recl=18)
*  What part has the external order demand?
10    print,'part or quit to stop'
      read,part
```

```
*   No more addition inputs?
      if (part='quit') go to 999
*   Find the system number for part i.
      call findsys(part,sys)
*   Part not on file?
      if (sys=-1) go to 10
*   Get the facts about the part.
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*   Set the loop count.
      n = 0
*
      loop
      n = n + 1
*   Get the specifics about the demand.
      print,'date,demand or -1,0 to stop'
      read(5,*)date,demand
*   No more entries?
      if (date = -1)
         write(10,01001,rec=0)keyxo
         read(10,*,rec=xokey)Da,D,k
         write(10,21003,rec=xokey)Da,D,-1
         go to 10
      endif
*   Increment record number.
      xokey = xokey + 1
      if (n=1)
*   Link the partstatus record to the XO record.
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xokey,
&           servpartkey,level,levelkey
         close(unit=3)
      endif
*   Increment key to the next record.
      keyxo = xokey + 1
      write(10,21003,rec=xokey)date,demand,keyxo
      endloop
      go to 10
999   close(unit=10)
      return
      end
*
      subroutine XOadd
*   The XO file can be appended with the following subprogram.
      character part
      integer  Da,date,sys,SR,BOM,BofM,ccper,pup,xorderkey,
&        servpartkey
*    Formats
1001     format(i4,5x,'XO')
11001    format(i4)
21003    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      open(unit=10,file='(f:18)XO.rel',access='DIRECT',recl=18)
      read(10,11001,rec=0)next
*   Get the specifics about the new addition.
10    print,'part,date,demand quantity or quit,0,0 to stop'
```

```
      read(5,*)part,date,demand
*   No more additions? stop.
      if (part='quit') go to 999
*   Get the system number.
      call findsys(part,sys)
*   Part not in file? correct?
      if (sys=-1) go to 10
      write(10,21003,rec=next)date,demand,-1
*   Link the old demand to the new demand.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if xorderkey = -1
*   If xorderkey = -1, there are no previous demands.
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,next,
&           servpartkey,level,levelkey
         next = next + 1
         write(10,01001,rec=0)next
         go to 10
      else
*   Search for the last record which is linked to new demand.
         k = xorderkey
20       read(10,*,rec=k)Da,D,key
         if key = -1
*   The last record has key = -1.
            write(10,21003,rec=k)Da,D,next
            next = next + 1
            write(10,01001,rec=0)next
            go to 10
         else
*   See if the next record(key) is last.
            k = key
            go to 20
         endif
      endif
999   close(unit=3)
      close(unit=10)
      return
      end
*
      subroutine XOchange
*   The XO file errors can be corrected with the following routine.
      character part
      real newdemand
      integer olddate,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&        sys,date
*   Formats
21003    format(i4,',',f8.0,',',i4)
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      open(unit=10,file='(f:18)XO.rel',access='DIRECT',recl=18)
*   What part demand needs to be corrected?
10    print,'part,date,demand quantity to be changed'
      print,'   or quit,0,0 to stop'
      read(5,*)part,olddate,olddemand
*   No more changes? stop.
      if (part='quit') go to 999
*   Locate system number.
```

```
      call findsys(part,sys)
*   Part not in file? correction?
      if (sys=-1) go to 10
*   How is the demand to be changed?
      print,'correct date,correct demand quantity'
      read(5,*)newdate,newdemand
*   Find the old demand.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if xorderkey = -1
*   If xorderkey = -1, there are no records on file.
         print,'old date and demand not on file'
         go to 10
      else
*   Start search with record=xorderkey.
         k = xorderkey
20       read(10,*,rec=k)date,demand,key
         if (date=olddate.and.demand=olddemand)
            write(10,21003,rec=k)newdate,newdemand,key
            go to 10
         else
*   If key = -1, there are no more records for part i.
            if key = -1
               print,'old date and demand not on file'
               go to 10
            else
*   See if the next record is the old demand.
               k = key
               go to 20
            endif
         endif
      endif
999   close(unit=3)
      close(unit=10)
      return
      end
*
      subroutine SPdemands
*    Type of editing for the Service Parts(SP) file is specified.
      character answer
10    print,'new,add,change or quit SP demand list?'
      read,answer
      if answer  = 'new'
         call SPnew
         go to 10
      elseif  answer  = 'add'
         call SPadd
         go to 10
      elseif answer = 'change'
         call  SPchange
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
```

```
      end
*
      subroutine SPnew
*   The SP file can be initialized with this subprogram.
      character part
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,sys,
&        date,Da,spkey
*   Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
1101     format(i4,5x,'SP')
21103    format(i4,',',f8.0,',',i4)
*   Reset the servpartkey to -1.
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10301,rec=0)next
      close(unit=3)
      do i = 1,next-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           -1,level,levelkey
         close(unit=3)
      enddo
*   Begin with the last record and pointer equal to zero.
      spkey = 0
      keysp = 0
      open(unit=11,file='(f:18)SP,rel',access='DIRECT',recl=18)
*    What part has the service part demand?
10    print,'part  or quit to stop'
      read,part
*   No more information to be entered? stop.
      if (part='quit') go to 999
*   Locate the system number.
      call findsys(part,sys)
*   Part not in file?
      if (sys=-1) go to 10
*   Get the facts about part i.
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*   Set the loop count.
      n  = 0
*
      loop
      n = n + 1
*    Get the specifics about SP demand.
      print,'date,demand or -1,0 to stop'
      read(5,*)date,demand
*    No more demands for part i?
      if (date=-1)
         write(11,01101,rec=0)keysp
         read(11,*,rec=spkey)Da,D,k
         write(11,21103,rec=spkey)Da,D,-1
         go to 10
      endif
```

```
*   Increment the record number.
      spkey = spkey + 1
      if n = 1
*   Link the partstatus record to the SP demand record.
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           spkey,level,levelkey
         close(unit=3)
      endif
*   Increment the key to next record.
      keysp = spkey + 1
      write(11,21103,rec=spkey)date,demand,keysp
      end loop
      go to 10
999   close(unit=11)
      return
      end
*
      subroutine SPadd
*   The SP file can be appended with the following.
      character part
      integer Da,date,sys,SR,BOM,BofM,ccper,pup,xorderkey,
&        servpartkey
*   Formats
1101     format(i4,5x,'SP')
11101    format(i4)
21103    format(i4,',',f8.0,',',i4)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      open(unit=11,file='(f:18)SP,rel',access='DIRECT',recl=18)
      read(11,11101,rec=0)next
*   What addition demand needs to be entered?
10    print,'part,date,demand or quit,0,0, to stop'
      read(5,*)part,date,demand
*   No more additions? stop.
      if (part='quit') go to 999
*   Find the system number.
      call findsys(part,sys)
*   Part not in file? correction?
      if (sys=-1) go to 10
      write(11,21103,rec=next)date,demand,-1
*   Link the old demands to the new demand.
      close(unit=3)
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
      if servpartkey = -1
*   If servpartkey = -1, there are no previous demands.
         write(3,20319,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,
&           lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           next,level,levelkey
         next  = next + 1
         write(11,01101,rec=0)next
         go to 10
      else
*   Start search for the last record with record=servpartkey.
         k = servpartkey
20       read(11,*,rec=k)Da,D,key
```

```
          if key = -1
*   The last record has key = -1.
              write(11,21103,rec=k)Da,D,next
              next = next + 1
              write(11,01101,rec=0)next
              go to 10
          else
*   Continue search for last record.
              k = key
              go to 20
          endif
        endif
999     close(unit=3)
        close(unit=11)
        return
        end
*
        subroutine SPchange
*   The SP input errors may be corrected with this routine.
        character part
        real newdemand
        integer olddate,SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&           date,sys
*   Formats
21103     format(i4,',',f8.0,',',i4)
        open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
        open(unit=11,file='(f:18)SP.rel',access='DIRECT',recl=18)
*   What demand needs correcting?
10      print,'part,date,demand quantity to be changed'
        print,'   or quit,0,0  to stop'
        read(5,*)part,olddate,olddemand
*   No more corrections? stop.
        if (part='quit') go to 999
*   Locate the system number.
        call findsys(part,sys)
*   Part not in file? correction?
        if (sys=-1) go to 10
*   What is the correction?
        print,'correct date,correct demand quantity'
        read(5,*)newdate,newdemand
*   Find the old demand.
        close(unit=3)
        open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
        read(3,*,rec=sys)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
        if servpartkey = -1
*   If servpartkey = -1, there are no demands for part i.
            print,'old date and demand not on file'
            go to 10
        else
*   Start search with record = servpartkey.
            k = servpartkey
20          read(11,*,rec=k)date,demand,key
            if (date=olddate.and.demand=olddemand)
*   When old demand is found, make the change.
                write(11,21103,rec=k)newdate,newdemand,key
                go to 10
            else
                if key = -1
```

```
*  If key = -1, there are no more demands for part i.  474
                print,'old date and demand not on file'
                go to 10
             else
*  Continue search to next record(key).
                k = key
                go to 20
             endif
          endif
       endif
999    close(unit=3)
       close(unit=11)
       return
       end
```

```
*
*  File - Level
*
*  Karen M. May      Date: June 3, 1983
*
*  Function - assign part levels and link levels for explosion
*
*  Variables:
*    BofM - key to first appearance as child part
*    BOM - key to first appearance as parent part
*    c - child part on bom record
*    ckey - key to next parent part for c
*    ccost - carrying cost ($/unit of time)
*    ccper - unit of time for carrying cost
*    lead - lead time
*    level - part level
*    levelkey - key for MRP flow
*    lotp - time in days for L4L and FPQ
*    lotr - lotting rule
*    MPSkey - key to MPS demands
*    nextbom - next available bom record
*    nextpart - next available partstatus record
*    numkids - number of different components
*    oh - on hand inventory
*    p - parent part on bom record
*    pkey - key to next child part for p
*    pu - periodal usage(units/unit of time)
*    pup - time unit for periodal usage
*    q - number of c/p on bom record
*    scost - set up cost
*    servpartkey - key to Service Part demands
*    SR - key to scheduled receipts
*    ucost - unit cost
*    xorderkey - key to external orders
*
*  Subprograms: LandSinitial,Lassign,Sremove,Levelconnect
*
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey
      call LandSinitial(nextpart,nextbom)
*   Level Number increment if after Lassign and Sremove,parts
* still need levels assigned
      L = 0
10    L = L + 1
*    What parts qualify to be assigned this level?
      call Lassign(L,nextpart,nextbom)
*   Eliminate B.O.M. records for next round?
      call Sremove(L,nextpart,nextbom)
*   Have all parts been assigned a level?
      j = 0
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         close(unit=3)
         if (level=-1) j = 1
      enddo
      if (j=1) go to 10
*   Set the MRP explosion flow.
      call Levelconnect(L,nextpart)
```

```
      end
*
      subroutine LandSinitial(nextpart,nextbom)
*  This subprogram initialize level = -1, levelkey = -1,
* and stack flag to 1.
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&          p,c,q,pkey,ckey
*  Formats
10301    format(i3)
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&          ',',i1,',',f8.0,',',i1,5(',',i3))
20106    format(2(i3,','),i4,3(',',i4))
10101    format(i4)
*  Initialize level at -1; -1 means no level has been assigned
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10301,rec=0)nextpart
      close(unit=3)
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,
            lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
            servpartkey,-1,-1
         close(unit=3)
      enddo
*  Initialize stack at 1; 1 = in stack & 0 = not in stack.
      open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      read(1,10101,rec=0) nextbom
      do i = 1,nextbom-1,1
         read(1,*,rec=i)p,c,q,pkey,ckey
         write(1,20106,rec=i)p,c,q,pkey,ckey,1
      enddo
      close(unit=1)
      return
      end
*
      subroutine Lassign(L,nextpart,nextbom)
*  This algorithm assigns level L to the parts that qualify.
      integer SR,BofM,BOM,ccper,pup,xorderkey,servpartkey,
&          p,c,q,pkey,ckey,s
*  Format
20319    format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&          ',',i1,',',f8.0,',',i1,5(',',i3))
*  Check to see if part i is in the bom child list.
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
*  If level = -1,then part need a level assignment.
         if level = -1
*  Part is in child list if i = c, and in stack if s=1.
               assign = 1
               open(unit=1,file='(f:27)bom,rel',access='DIRECT')
               do j = 1,nextbom-1,1
                  read(1,*,rec=j)p,c,q,pkey,ckey,s
*  Found in the list?
                  if (i=c.and.s=1) assign=0
```

```
                enddo
                close(unit=1)
                if assign = 1
*   If not found in stack, assign part to level L.
                    write(3,20319,rec=i)oh,SR,numkids,BOM,BofM,lead,
&                       lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&                       MPSkey,xorderkey,servpartkey,L,levelkey
                endif
              endif
              close(unit=3)
            enddo
            return
            end
*
      subroutine Sremove(L,nextpart,nextbom)
*   Eliminate B.O.M. records from the stack when part with level L
* is in parent list.
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&         p,c,q,pkey,ckey,s
*   Formats
20106     format(2(i3,','),i4,3(',',i4))
*   Determine if bom record should be removed from the stack.
      do i = 1,nextpart-1,1
         open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
         read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&           scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&           servpartkey,level,levelkey
         if level = L
            open(unit=1,file='(f:27)bom,rel',access='DIRECT')
            do j = 1,nextbom-1,1
               read(1,*,rec=j)p,c,q,pkey,ckey,s
               if (i=p) write(1,20106,rec=j)p,c,q,pkey,ckey,0
            enddo
            close(unit=1)
         endif
         close(unit=3)
         enddo
         return
         end
*
      subroutine Levelconnect(L,nextpart)
*    The subprogram connects levels in flow  order for MRP
* explosipn.
      integer SR,BOM,BofM,ccper,pup,xorderkey,servpartkey,
&         SR1,BOM1,BofM1,ccper1,pup1,xorderkey1,servpartkey1
*   Formats
302       format(2i3,5x,'Partstatus Data File')
20319     format(f8.0,5(',',i4),',',i1,',',i4,',',f8.2,2(',',f8.4),
&           ',',i1,',',f8.0,',',i1,5(',',i3))
*   Highest level assigned is the stopping criteria.
      Lmax = L
*   Connect the lower levels first.
      L = 0
*   Increment times through loop.
      n = 0
      loop
      L = L + 1
*   Beyond maximum level? Stop.
      quitif L > Lmax
      do i = 1,nextpart-1,1
```

```
      open(unit=3,file='(f:100)partstatus.rel',access='DIRECT')
      read(3,*,rec=i)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,
&        scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*  If level = L, set link.
      if level = L
        n = n + 1
*  Link after first read.
        if (L>1.or.n>1)
          write(3,20319,rec=i1)oh1,SR1,numkids1,BOM1,BofM1,
&            lead1,lotr1,lotp1,scost1,ucost1,ccost1,ccper1,
&            pu1,pup1,MPSkey1,xorderkey1,servpartkey1,level1,i
        endif
*  Store the first level 1 in record 0; the explosion starts here.
        if (L=1.and.n=1)
          write(3,00302,rec=0) nextpart,i
        endif
*  Remember the details on the last location linked.
        i1 = i
        oh1 = oh
        SR1  = SR
        numkids1  = numkids
        BOM1 = BOM
        BofM1 = BofM
        lead1 = lead
        lotr1 = lotr
        lotp1 = lotp
        scost1 = scost
        ucost1 = ucost
        ccost1 = ccost
        ccper1 = ccper
        pu1 = pu
        pup1 = pup
        MPSkey1 = MPSkey
        xorderkey1 = xorderkey
        servpartkey1 = servpartkey
        level1 = level
      endif
      close(unit=3)
    enddo
    end loop
    return
    end
*
```

```
*
*    File - MRP
*
*  Karen M. May      Date: June 6, 1983
*
*     BfoM - first appearance in bom child's list
*     BOM - first appearance in bom parent's list
*     buckets - number of periods in horizon
*     c - company part number
*     ccost - carrying cost ($/time period)
*     ccper - time period for ccost
*     ddkey - next available record in DD file
*     lead - lead time in days
*     level - part level number
*     levelkey - next part to explode
*     lotp - number of days for L4L and FPQ
*     lotr - lotting rule
*     MPSkey - key to MPS demand records
*     name - company name
*     nextpart - next available record in partstatus
*     numkids - number of different components
*     oh - on hand inventory
*     parttoexplode - record number of part to explode
*     pu - periodal usage (units/time period)
*     pup - time period for pu
*     scost - set up cost
*     servpartkey - key to SP demand records
*     SR - key to scheduled receipts
*     start - first part to explode
*     ucost - unit cost
*     xorderkey - key to XO demand records
*
*  Subprograms: DDkey,setupHorizon,MPStransfer,XOtransfer,
* SPtransfer,DependentDemands,GrossRequirements,SRtransfer,
* OHandNR,LotSize,L4L,FPQ,OrderRelease,OutPut,SIX,FIVE,FOUR,
* THREE,TWO,ONE,OrderWrite
*
      character c,name
      integer start,parttoexplode,SR,BofM,BOM,ccper,pup,
&           xorderkey,servpartkey,buckets,ddkey
*  format
10302    format(2i3)
10801    format( ' DD file ')
*  Initialize the DD file key.
      ddkey = 0
*  Label the DD file.
      open(unit=8,file='(f:17)DD,rel',access='DIRECT',recl=17)
      write(8,10801,rec=0)
      close(unit=8)
*  Heading for the MRP printer output.
      open(unit=6,file='printer')
      write(6,10600)
10600    format(16('*'),'Materials Requirement Planning',16('*'))
      close(unit=6)
*  The name is needed for the Purchase Order Heading.
      print,'Company Name (up to 12 characters in length)?'
      read,name
*  Initialize Dependent Demand key (DDkey) in partid file.
      call DDkey
*  Which part is exploded first?
```

```
      open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,10302,rec=0) nextpart,start
      parttoexplode = start
      close(unit=3)
*   Get the scoop on the part to be exploded?
10    open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
      read(3,*,rec=parttoexplode)oh,SR,numkids,BOM,BofM,lead,lotr,
&        lotp,scost,ucost,ccost,ccper,pu,pup,MPSkey,xorderkey,
&        servpartkey,level,levelkey
*   Headings for printout
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,*,rec=parttoexplode)c
      close(unit=2)
      open(unit=6,file='printer')
      write(6,10601)c,parttoexplode
10601    format('Part Number ',a10,';System Number',i3)
      write(6,10602)lead,oh
10602    format('Lead Time = ',i4,' days; On Hand Inventory is ',
&           f8.0)
      close(unit=6)
*   Initialize values in horizon.
      call setupHorizon (buckets)
*   What demand are placed on this part?
      if(MPSkey<>-1) call MPStransfer(MPSkey,buckets)
      if(xorderkey<>-1) call XOtransfer(xorderkey,buckets)
      if(servpartkey<>-1) call SPtransfer(servpartkey,buckets)
      if(BofM<>-1)call DependentDemands(BofM,buckets,parttoexplode)
*   Total Demands.
      call GrossRequirements(buckets)
*   What orders have been placed?
      if(SR<>-1) call SRtransfer(SR,buckets)
*   Determine on have quantities and net requirements.
      call OHandNR(oh,buckets)
*   Determine the lot (or order) size.
      call LotSize(lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&        buckets)
*   Adjust order date according to lead time.
      call OrderRelease(buckets,lead,ddkey,parttoexplode)
*   Print out results.
      call OutPut(buckets,lead,name,parttoexplode)
*   Are any parts left to be exploded?
      if(levelkey=-1) go to 999
      parttoexplode = levelkey
      close(unit=3)
      go to 10
999   stop
      end
*
      subroutine DDkey
*   Dependent demand key(DDkey) is initialized in the following
* subprogram.
      character c
      integer first,sys,p,u,d
*   Formats
10202    format(2i3)
20207    format(a10,5(',',i3),',',i4)
*   How many recordes need to be initialized?
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10202,rec=0)first,next
*   Initialize the required number of records.
```

```
      do i = 1,next-1,1
         read(2,*,rec=i)c,sys,k,p,u,d
         write(2,20207,rec=i)c,sys,k,p,u,d,-1
      enddo
      close(unit=2)
      return
      end
*
      subroutine setupHorizon(buckets)
*  The planning horizon is initialized with this routine.
      integer buckets,period,date
*  Formats
10701    format(i3)
20712    format(i3,',',i4,10(',',f8.0))
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,10701,rec=0)buckets
      close(unit=7)
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date
         write(7,20712,rec=i)period,date,0.,0.,0.,0.,0.,0.,0.,0.,
&           0.,0.
         close(unit=7)
      enddo
      return
      end
*
      subroutine MPStransfer(MPSkey,buckets)
*  The MPS(end item) demands are transferred to planning horizon.
      integer buckets,day,period,date,date1,period1
      real NR,LR
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*  Begin with the first MPS demand
      k = MPSkey
*  Get the details about the demands
      open(unit=9,file='(f:18)MPS.rel',access='DIRECT',recl=18)
10    read(9,*,rec=k)day,demand,key
*  Find the correct time period for the demand.
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         j = i + 1
         if j > buckets
            date1 = 9999
            go to 15
         endif
         read(7,*,rec=j)period1,date1
15       if (date<=day.and.day<date1)
            demand = dMPS + demand
            write(7,20712,rec=i)period,date,demand,dXO,dSP,sumDD,
&              GR,SR,OH,NR,LR,OR
            close(unit=7)
            go to 20
         endif
         close(unit=7)
      enddo
*  Any more demands for part i?
20    if (key=-1) go to 999
```

```
      k = key
      go to 10
999   close(unit=9)
      return
      end
*
      subroutine XOtransfer(xorderkey,buckets)
*    The XO (external order) demands are transferred to the
* planning horizon.
      integer buckets,day,period,date,date1,period1,xorderkey
      real NR,LR
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*  Begin with the first demand.
      k = xorderkey
*  Get the information about the demand.
      open(unit=10,file='(f:18)XO,rel',access='DIRECT',recl=18)
10    read(10,*,rec=k)day,demand,key
*  Locate the correct time period for the demand.
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         j = i + 1
         if j > buckets
            date1 = 9999
            go to 15
         endif
         read(7,*,rec=j)period1,date1
15       if (date<=day.and.day<date1)
            demand = dXO + demand
            write(7,20712,rec=i)period,date,dMPS,demand,dSP,sumDD,
&             GR,SR,OH,NR,LR,OR
            close(unit=7)
            go to 20
         endif
         close(unit=7)
      enddo
*  Any more demand for part i?
20    if(key = -1) go to 999
      k = key
      go to 10
999   close(unit=10)
      return
      end
*
      subroutine SPtransfer(servpartkey,buckets)
*  The SP (Service Parts) demands are transferred to the
* planning horizon.
      integer buckets,day,period,date,date1,period1,servpartkey
      real NR,LR
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*  Begin with the first demand.
      k = servpartkey
*  Get the details about the demand.
      open(unit=11,file='(f:18)SP,rel',access='DIRECT',recl=18)
10    read(11,*,rec=k)day,demand,key
*  Locate the proper time period for the demand.
      do i = 1,buckets,1
```

```
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,NR,
&        LR,OR
      j = i + 1
      if j > buckets
         date1 = 9999
         go to 15
      endif
      read(7,*,rec=j)period1,date1
15    if (date<=day.and.day<date1)
         demand = dSP + demand
         write(7,20712,rec=i)period,date,dMPS,dXO,demand,sumDD,
&           GR,SR,OH,NR,LR,OR
         close(unit=7)
         go to 20
      endif
      close(unit=7)
   enddo
*  Any more demands for part i?
20    if (key=-1) go to 999
   k = key
   go to 10
999   close(unit=11)
   return
   end
*
   subroutine GrossRequirements(buckets)
*  The gross requirements for each period is calculated by the
* following procedure.
   integer period,date,buckets
   real NR,LR
*  Format
20712    format(i3,',',i4,10(',',f8.0))
*  Add the GR for each period.
   do i = 1,buckets,1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&        NR,LR,OR
      GR = dMPS + dXO + dSP + sumDD
      write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,
&        OH,NR,LR,OR
      close(unit=7)
   enddo
   return
   end
*
   subroutine SRtransfer(SRkey,buckets)
*  The scheduled receipts are transferred into the planning
* horizon.
   integer buckets,SRkey,period,date,day,date1,period1
   real NR,LR
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*  Begin with first SR.
   k = SRkey
*  Get details about SR.
   open(unit=4,file='(f:18)SR.rel',access='DIRECT',recl=18)
10    read(4,*,rec=k)day,qty,key
*  Find proper period for SR in planning horizon.
   do i = 1,buckets,1
```

```
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&        NR,LR,OR
      j = i + 1
      if j > buckets
         date1 = 9999
         go to 15
      endif
      read(7,*,rec=j)period1,date1
15    if (date<=day.and.day<date1)
         SR = SR + qty
         write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,
&           SR,OH,NR,LR,OR
         close(unit=7)
         go to 20
      endif
      close(unit=7)
   enddo
*  Any more SR?
20    if (key=-1) go to 999
   k = key
   go to 10
999   close(unit=4)
   return
   end
*
   subroutine OHandNR(oh,buckets)
*  The following calculates on hand and net requirements amounts.
   integer period,date,buckets
   real NR,LR,net
*  Formats
20712    format(i3,',',i4,10(',',f8.0))
*  Initialize net with on hand inventory.
   net = oh
*  Determine on hand inventory and net requirements for each per.
   do i = 1,buckets,1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&        NR,LR,OR
      net = GR - (SR + net)
      if (net<0)
         OH = -net
         net = -net
         go to 10
      endif
      if (net>0)
         NR = net
         net  = 0
      endif
10    write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,
&        OH,NR,LR,OR
      close(unit=7)
   enddo
   return
   end
*
   subroutine LotSize(lotr,lotp,scost,ucost,ccost,ccper,pu,pup,
&      buckets)
*  Proper Lot Size technique is requested by these statements.
    integer ccper,pup,buckets
```

```
      if (lotr = 1) call L4L(buckets)
      if (lotr = 2) call FPQ(buckets,lotp)
      return
       end
*
      subroutine L4L(buckets)
*  Lot-for-lot technique is performed by the subprogram.
      integer period,date,buckets
      real NR,LR
*  Format
20712    format(i3,',',i4,10(',',f8.0))
      open(unit=6,file='printer')
      write(6,10601)
10601 format('Lot Size is determined with Lot-for-Lot Technique')
      close(unit=6)
*  For each period, Lot Size = Net Requirements.
      do i = 1,buckets,1
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         LR = NR
         write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,
&           OH,NR,LR,OR
         close(unit=7)
      enddo
      return
      end
*
      subroutine FPQ(buckets,lotp)
*  The Fixed Period Quantity Lotting Technique is performed
* by the following subprogram.
      integer period,date,period1,date1,buckets,totdays
      real NR,LR,NR1,LR1
*  Format
20712    format(i3,',',i4,10(',',f8.0))
      open(unit=6,file='printer')
      write(6,10601)
10601 format('Lot Size is determined with Fixed Period'
&       Algorithm.')'
      close(unit=6)
      i = 0
*  Find the first nonzero net requirement(NR).
       loop
      i = i + 1
      if (i> buckets) go to 999
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,NR,
&       LR,OR
      close(unit=7)
      if (NR <> 0) go to 20
      endloop
*  Sum NR until lot period length has been reached or until no
* buckets are left.
20    loop
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
      read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,NR,
&       LR,OR
      LR = NR
      j = i
```

```
30    write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&        NR,LR,OR
      close(unit=7)
      j = j + 1
*  No more buckets left?
      if (j > buckets) go to 999
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
      read(7,*,rec=j)period1,date1,dMPS1,dXO1,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      LR = LR + NR1
*  Reached lot period length?
      if lotp <= (date1 - date)
         close(unit=7)
         i = j
         go to 20
      else
         go to 30
      endif
      endloop
999   return
      end
*
     subroutine OrderRelease(buckets,leadtime,DDkey,parttoexplode)
*  The subprogram determines when order is to be released;stores
* DD; and places pointer for DD in partid file.
      character c
      integer buckets,DDkey,parttoexplode,period,date,period1,
&        date1,currentperiod,today,totdays,sys,p,u,d,DD
      real NR,LR,NR1,LR1
*  Formats
20207    format(a10,5(',',i3),',',i4)
20712    format(i3,',',i4,10(',',f8.0))
20803    format(i3,',',i4,',',f8.0)
*  Initialize the number of DD stored.
      n = 0
*  Initialize the amount of NR overdue.
      pastdue = 0.
      open(unit=8,file='(f:17)DD,rel',access='DIRECT',recl=17)
*  Find out today's date.
      open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
      read(7,*,rec=1)currentperiod,today
      close(unit=7)
      do i = buckets,1,-1
         next = i
*  What is the lot size or size of the order?
         open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         close(unit=7)
*  Is there enough time to make lead time adjustment?
         if (leadtime > (date - today)) go to 20
         do j = next,1,-1
            open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
            read(7,*,rec=j)period1,date1,dMPS1,dXO1,dSP1,sumDD1,
&              GR1,SR1,OH1,NR1,LR1,OR1
*  Has enough time elapsed for lead time adjustment?
            totdays = date - date1
            if (leadtime <= totdays)
               or = LR + OR1
               write(7,20712,rec=j)period1,date1,dMPS1,dXO1,dSP1,
```

```
&                  sumDD1,GR1,SR1,OH1,NR1,LR1,or
                close(unit=7)
                if (or <> 0)
*  Store DD for part i in DD file.
                   n = n + 1
                   DDkey = DDkey + 1
                   write(8,20803,rec=DDkey)parttoexplode,date1,or
                   if  n = 1
*  Link partid file to first DD stored.
                   open(unit=2,file='(f:35)partid,rel',access=
&                     'DIRECT')
                      read(2,*,rec=parttoexplode)c,sys,k,p,u,d,DD
                      write(2,20207,rec=parttoexplode)c,sys,k,p,u,
&                        d,DDkey
                      close(unit=2)
                   endif
                endif
                close(unit=7)
                go to 15
             endif
             close(unit=7)
          enddo
15        continue
       enddo
       go to 999
*  Add up the overdue demands.
20     do k = next,1,-1
          open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
          read(7,*,rec=k)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&            NR,LR,OR
          pastdue = pastdue + LR
          close(unit=7)
       enddo
999    close(unit=8)
       open(unit=6,file='printer')
       write(6,10601)pastdue
10601     format('The amount of parts overdue is ',f8.0)
       close(unit=6)
       return
       end
*
       subroutine OutPut(buckets,leadtime,name,parttoexplode)
       character name
       integer buckets,orderperiod,orderdate,period,date,columns,
&         parttoexplode
       n = 0
       i = 0
       columns = buckets
10     columns = columns - 6
     if (columns>=0)call SIX(orderperiod,orderdate,ordersize,n,i)
     if (columns=-1)call FIVE(orderperiod,orderdate,ordersize,n,i)
     if (columns=-2)call FOUR(orderperiod,orderdate,ordersize,n,i)
     if (columns=-3)call THREE(orderperiod,orderdate,ordersize,n,i)
     if (columns=-4)call TWO(orderperiod,orderdate,ordersize,n,i)
     if (columns=-5)call ONE(orderperiod,orderdate,ordersize,n,i)
       if (n = 1)
          period = orderperiod
          date = orderdate
          size  = ordersize
       endif
```

```
      if (columns<=0) go to 999
      go to 10
999   if (size>0) call OrderWrite(period,date,size,leadtime,
&        parttoexplode,name)
      return
      end
*
      subroutine SIX(orderperiod,orderdate,ordersize,n,i)
      integer period1,period2,period3,period4,period5,period6,
&        date1,date2,date3,date4,date5,date6,orderperiod,
&        orderdate
      real LR1,LR2,LR3,LR4,LR5,LR6,NR1,NR2,NR3,NR4,NR5,NR6
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      orderdate = date1
      ordersize  = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      i = i + 1
      read(7,*,rec=i)period4,date4,dMPS4,dX04,dSP4,sumDD4,GR4,SR4,
&        OH4,NR4,LR4,OR4
      i = i + 1
      read(7,*,rec=i)period5,date5,dMPS5,dX05,dSP5,sumDD5,GR5,SR5,
&        OH5,NR5,LR5,OR5
      i = i + 1
      read(7,*,rec=i)period6,date6,dMPS6,dX06,dSP6,sumDD6,GR6,SR6,
&        OH6,NR6,LR6,OR6
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3,period4,period5,
&        period6
10601    format('_Period_',6(3x,i3,3x))
      write(6,10602)date1,date2,date3,date4,date5,date6
10602    format('_1st_day',6(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3,dMPS4,dMPS5,dMPS6
10603    format('__MPS___',6(1x,f8.0))
      write(6,10604)dX01,dX02,dX03,dX04,dX05,dX06
10604    format('___XO___',6(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3,dSP4,dSP5,dSP6
10605    format('___SP___',6(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3,sumDD4,sumDD5,sumDD6
10606    format('_sum_DD_',6(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3,GR4,GR5,GR6
10607    format('___GR___',6(1x,f8.0))
      write(6,10608)SR1,SR2,SR3,SR4,SR5,SR6
10608    format('___SR___',6(1x,f8.0))
      write(6,10609)OH1,OH2,OH3,OH4,OH5,OH6
10609    format('___OH___',6(1x,f8.0))
```

```
      write(6,10610)NR1,NR2,NR3,NR4,NR5,NR6
10610    format('___NR___',6(1x,f8.0))
      write(6,10611)LR1,LR2,LR3,LR4,LR5,LR6
10611    format('___LR___',6(1x,f8.0))
      write(6,10612)OR1,OR2,OR3,OR4,OR5,OR6
10612    format('___OR___',6(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine FIVE(orderperiod,orderdate,ordersize,n,i)
      integer period1,period2,period3,period4,period5,date1,date2,
&        date3,date4,date5,orderperiod,orderdate
      real NR1,NR2,NR3,NR4,NR5,LR1,LR2,LR3,LR4,LR5
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      orderdate = date1
      ordersize = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      i = i + 1
      read(7,*,rec=i)period4,date4,dMPS4,dX04,dSP4,sumDD4,GR4,SR4,
&        OH4,NR4,LR4,OR4
      i = i + 1
      read(7,*,rec=i)period5,date5,dMPS5,dX05,dSP5,sumDD5,GR5,SR5,
&        OH5,NR5,LR5,OR5
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3,period4,period5
10601    format('_Period_',5(3x,i3,3x))
      write(6,10602)date1,date2,date3,date4,date5
10602    format('_1st_day',5(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3,dMPS4,dMPS5
10603    format('__MPS___',5(1x,f8.0))
      write(6,10604)dX01,dX02,dX03,dX04,dX05
10604    format('___XO___',5(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3,dSP4,dSP5
10605    format('___SP___',5(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3,sumDD4,sumDD5
10606    format('_sum_DD_',5(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3,GR4,GR5
10607    format('___GR___',5(1x,f8.0))
      write(6,10608)SR1,SR2,SR3,SR4,SR5
10608    format('___SR___',5(1x,f8.0))
      write(6,10609)OH1,OH2,OH3,OH4,OH5
10609    format('___OH___',5(1x,f8.0))
      write(6,10610)NR1,NR2,NR3,NR4,NR5
```

```
10610   format('___NR___',5(1x,f8.0))
      write(6,10611)LR1,LR2,LR3,LR4,LR5
10611   format('___LR___',5(1x,f8.0))
      write(6,10612)OR1,OR2,OR3,OR4,OR5
10612   format('___OR___',5(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine FOUR(orderperiod,orderdate,ordersize,n,i)
      integer period1,period2,period3,period4,date1,date2,date3,
&        date4,orderperiod,orderdate
      real NR1,NR2,NR3,NR4,LR1,LR2,LR3,LR4
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1
&        OH1,NR1,LR1,OR1
      orderperiod  = period1
      orderdate = date1
      ordersize  = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      i = i + 1
      read(7,*,rec=i)period4,date4,dMPS4,dX04,dSP4,sumDD4,GR4,SR4,
&        OH4,NR4,LR4,OR4
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600   format(62('_'))
      write(6,10601)period1,period2,period3,period4
10601   format('_Period_',4(3x,i3,3x))
      write(6,10602)date1,date2,date3,date4
10602   format('_1st_day',4(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3,dMPS4
10603   format('__MPS___',4(1x,f8.0))
      write(6,10604)dX01,dX02,dX03,dX04
10604   format('___XO___',4(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3,dSP4
10605   format('___SP___',4(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3,sumDD4
10606   format('_sum_DD_',4(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3,GR4
10607   format('___GR___',4(1x,f8.0))
      write(6,10608)SR1,SR2,SR3,SR4
10608   format('___SR___',4(1x,f8.0))
      write(6,10609)OH1,OH2,OH3,OH4
10609   format('___OH___',4(1x,f8.0))
      write(6,10610)NR1,NR2,NR3,NR4
10610   format('___NR___',4(1x,f8.0))
      write(6,10611)LR1,LR2,LR3,LR4
10611   format('___LR___',4(1x,f8.0))
      write(6,10612)OR1,OR2,OR3,OR4
```

```
10612    format('___OR___',4(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine THREE(orderperiod,orderdate,ordersize,n,i)
      integer period1,period2,period3,date1,date2,date3,
&        orderperiod,orderdate
      real NR1,NR2,NR3,LR1,LR2,LR3
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod  = period1
      orderdate = date1
      ordersize = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      i = i  + 1
      read(7,*,rec=i)period3,date3,dMPS3,dX03,dSP3,sumDD3,GR3,SR3,
&        OH3,NR3,LR3,OR3
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2,period3
10601    format('_Period_',3(3x,i3,3x))
      write(6,10602)date1,date2,date3
10602    format('_1st_day',3(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2,dMPS3
10603    format('__MPS___',3(1x,f8.0))
      write(6,10604)dX01,dX02,dX03
10604    format('___XO___',3(1x,f8.0))
      write(6,10605)dSP1,dSP2,dSP3
10605    format('___SP___',3(1x,f8.0))
      write(6,10606)sumDD1,sumDD2,sumDD3
10606    format('_sum_DD_',3(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2,GR3
10607    format('___GR___',3(1x,f8.0))
      write(6,10608)SR1,SR2,SR3
10608    format('___SR___',3(1x,f8.0))
      write(6,10609)OH1,OH2,OH3
10609    format('___OH___',3(1x,f8.0))
      write(6,10610)NR1,NR2,NR3
10610    format('___NR___',3(1x,f8.0))
      write(6,10611)LR1,LR2,LR3
10611    format('___LR___',3(1x,f8.0))
      write(6,10612)OR1,OR2,OR3
10612    format('___OR___',3(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine TWO(orderperiod,orderdate,ordersize)
```

```
      integer period1,period2,date1,date2,orderperiod,orderdate
      real NR1,NR2,LR1,LR2
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      orderdate = date1
      ordersize = OR1
      i = i + 1
      read(7,*,rec=i)period2,date2,dMPS2,dX02,dSP2,sumDD2,GR2,SR2,
&        OH2,NR2,LR2,OR2
      close(unit=7)
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1,period2
10601    format('_Period_',2(3x,i3,3x))
      write(6,10602)date1,date2
10602    format('_1st_day',2(3x,i4,2x))
      write(6,10600)
      write(6,10603)dMPS1,dMPS2
10603    format('__MPS___',2(1x,f8.0))
      write(6,10604)dX01,dX02
10604    format('___X0___',2(1x,f8.0))
      write(6,10605)dSP1,dSP2
10605    format('___SP___',2(1x,f8.0))
      write(6,10606)sumDD1,sumDD2
10606    format('_sum_DD_',2(1x,f8.0))
      write(6,10600)
      write(6,10607)GR1,GR2
10607    format('___GR___',2(1x,f8.0))
      write(6,10608)SR1,SR2
10608    format('___SR___',2(1x,f8.0))
      write(6,10609)OH1,OH2
10609    format('___OH___',2(1x,f8.0))
      write(6,10610)NR1,NR2
10610    format('___NR___',2(1x,f8.0))
      write(6,10611)LR1,LR2
10611    format('___LR___',2(1x,f8.0))
      write(6,10612)OR1,OR2
10612    format('___OR___',2(1x,f8.0))
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine ONE(orderperiod,orderdate,ordersize,n,i)
      integer period1,date1,orderdate,ordersize
      real LR1,NR1
      n = n + 1
      open(unit=7,file='(f:98)horizon.rel',access='DIRECT')
      i = i + 1
      read(7,*,rec=i)period1,date1,dMPS1,dX01,dSP1,sumDD1,GR1,SR1,
&        OH1,NR1,LR1,OR1
      orderperiod = period1
      ordersize = OR1
      orderdate = date1
      close(unit=7)
```

```
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('_'))
      write(6,10601)period1
10601    format('_Period_',3x,i3)
      write(6,10602)date1
10602    format('_1sy_day',3x,i4)
      write(6,10600)
      write(6,10603)dMPS1
10603    format('__MPS___',1x,f8.0)
      write(6,10604)dX01
10604    format('___X0___',4x,f8.0)
      write(6,10605)dSP1
10605    format('___SP___',4x,f8.0)
      write(6,10606)sumDD1
10606    format('_sum_DD_',4x,f8.0)
      write(6,10600)
      write(6,10607)GR1
10607    format('___GR___',4x,f8.0)
      write(6,10608)SR1
10608    format('___SR___',4x,f8.0)
      write(6,10609)OH1
10609    format('___OH___',4x,f8.0)
      write(6,10610)NR1
10610    format('___NR___',4x,f8.0)
      write(6,10611)LR1
10611    format('___LR___',4x,f8.0)
      write(6,10612)OR1
10612    format('___OR___',4x,f8.0)
      write(6,10600)
      close(unit=6)
      return
      end
*
      subroutine OrderWrite(period,date,size,leadtime,parttoorder,
&        name)
      character part,name
      integer period,date,parttoorder,duedate
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,*,rec=parttoorder)part
      close(unit=2)
      duedate = date + leadtime
      open(unit=6,file='printer')
      write(6,10600)
10600    format(62('*'))
      do i = 1,4,1
         write(6,10601)
10601    format('***',56x,'***')
      enddo
      write(6,10602)name
10602    format('***',22x,a12,22x,'***')
      write(6,10603)
10603    format('***',21x,'PURCHASE_ORDER',21x,'***')
         write(6,10601)
         write(6,10601)
      write(6,10604)period
10604    format('***',28x,'Period_placed_',i3,11x,'***')
      write(6,10605)date
10605    format('***',28x,'Date_placed___',i4,10x,'***')
         write(6,10601)
```

```
         write(6,10601)
      write(6,10606)part
10606    format('***',7x,'Part_Number_',a10,27x,'***')
      write(6,10601)
      write(6,10607)size
10607    format('***',7x,'Quantity____',f8.0,29x,'***')
      write(6,10601)
      write(6,10608)duedate
10608    format('***',7x,'Due_Date____',i4,33x,'***')
      do i = 1,4,1
         write(6,10601)
      enddo
      write(6,10600)
      return
      end
*
      subroutine DependentDemands(BofM,buckets,parttoexplode)
      character c
      integer parent,child,qtyc,pkey,ckey,BofM,s,p,u,d,DDkey,
&        period,date,day,part,buckets,parttoexplode
      real NR,LR
      open(unit=6,file='printer')
      write(6,10600)
10600    format('Dependent Demand sources are as follows:')
*  What parents place the demands?
      kBOM = BofM
10    open(unit=1,file='(f:27)bom.rel',access='DIRECT',recl=27)
      read(1,*,rec=kBOM)parent,child,qtyc,pkey,ckey
      close(unit=1)
*  Has the parent placed demands?
      open(unit=2,file='(f:35)partid.rel',access='DIRECT',recl=35)
      read(2,*,rec=parent)c,s,k,p,u,d,DDkey
      close(unit=2)
      if DDkey = -1
*  No demands this time.
         write(6,10601)c
10601    format('Part_',a10,'Produces No Dependent Demands')
*  No more parent demands?
         if (ckey = -1) go to 999
         kBOM = ckey
         go to 10
      endif
      write(6,10602)c
10602    format('Part_',a10,5x,'_Date_',5x,'_Quantity_')
      kDD = DDkey
20    open(unit=8,file='(f:17)DD.rel',access='DIRECT',recl=17)
      read(8,*,rec=kDD)part,day,qDD
      close(unit=8)
*  Make sure the demands are from parent i.
      if (part<>parent)
*  No more parent demands?
         if (ckey = -1) go to 999
         kBOM = ckey
         go to 10
      endif
      qtyDD = qtyc * qDD
      write(6,10603)day,qtyDD
10603    format(21x,i4,7x,f8.0)
      sumDD = 0
*  Find the right time slot for the demands.
```

```
      do i = 1,buckets,1
        open(unit=7,file='(f:98)horizon,rel',access='DIRECT')
         read(7,*,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,SR,OH,
&           NR,LR,OR
         if (day = date)
            sumDD = sumDD + qtyDD
            write(7,20712,rec=i)period,date,dMPS,dXO,dSP,sumDD,GR,
&              SR,OH,NR,LR,OR
20712       format(i4,',',i3,10(',',f8.0))
            close(unit=7)
            kDD = kDD + 1
         endif
      close(unit=7)
      enddo
*  More demands?
      if (part=parent) go to 20
*  No more parents?
      if (ckey = -1) go to 999
      kBOM = ckey
      go to 10
999   close(unit=6)
      return
      end
*
```

```
*
*  File - System Status
*
*  Karen M. May      Date: June 7, 1983
*
*  Function: to print out information from the data files
*
*  Variables:
*    answer - response to questions
*
*  Subprograms: PartsList,sortedparts,unsortedparts,BOMlist,
* MRPflow
*
      character answer
*  Type of system status information is requested.
10    print,'To obtain flow of explosion,type flow.'
      print,'To obtain list of parts,type part.'
      print,'To obtain list of Bills of Material,type BOM.'
      print,'To stop,type quit.'
      read,answer
      if answer = 'part'
         call PartsList
         go to 10
      elseif answer = 'BOM'
         call BOMlist
         go to 10
      elseif answer = 'flow'
         call MRPflow
         go to 10
      elseif answer  = 'quit'
         go to 999
      else
         go to 10
      endif
999   stop
      end
*
      subroutine PartsList
*  The order for the parts list is requested.
      character answer
10    print,'To get sorted list,type s.'
      print,'To get as entered list,type u.'
      print,'To stop,type quit.'
      read,answer
      if answer = 's'
         call sortedparts
         go to 10
      elseif answer = 'u'
         call unsortedparts
         go to 10
      elseif answer = 'quit'
         go to 999
      else
         go to 10
      endif
999   return
      end
*
      subroutine sortedparts
*    Parts list is printed out with company part numbers sorted.
```

```
      character c
      integer first,next,s
      open(unit=6,file='printer')
      write(6,10600)
10600    format(5x,'Parts_List')
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10201,rec=0)first
10201    format(i3)
      key = first
10    read(2,*,rec=key)c,s,k
      write(6,10601)c
10601    format(5x,a10)
      if (k=999) go to 999
      key = k
      go to 10
999   close(unit=6)
      close(unit=2)
      return
      end

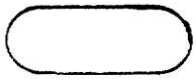
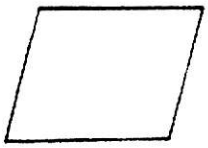
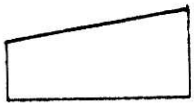
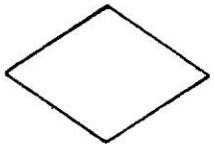
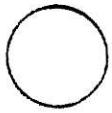
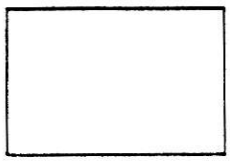
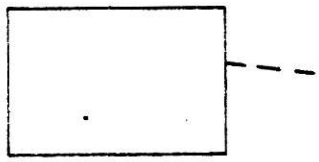
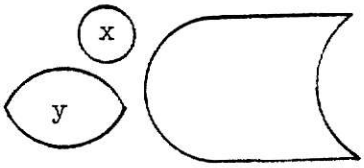
      subroutine unsortedparts
   Parts list is printed out unsorted.  That is as entered.
      character c
      integer first,next
      open(unit=6,file='printer')
      open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
      read(2,10202,rec=0)first,next
10202    format(2i3)
      write(6,10600)
10600    format(5x,'Parts_List')
      do i = 1,next-1,1
         read(2,*,rec=i)c
         write(6,10601)c
10601    format(5x,a10)
      enddo
      close(unit=6)
      close(unit=2)
      return
      end
*
      subroutine BOMlist
*  The bills-of-materials is printed out by part module.
      character parent,child
      integer p,c,qty,pkey
      open(unit=6,file='printer')
      write(6,10600)
10600    format(5x,'Bills of Material')
      write(6,10601)
10601    format(5x,36('-'))
      write(6,10602)
10602    format(5x,'__Parent__',3x,'__Child___',3x,'_Quantity_')
      write(6,10601)
      open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
      read(1,10101,rec=0)next
      close(unit=1)
10101    format(i4)
      do i = 1,next-1,1
         open(unit=1,file='(f:27)bom,rel',access='DIRECT',recl=27)
         read(1,*,rec=i)p,c,qty,pkey
         close(unit=1)
```

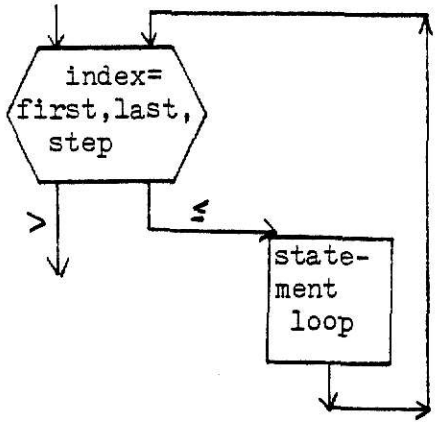
```
      open(unit=2,file='(f:35)partid,rel',access='DIRECT')
      read(2,*,rec=p)parent
      read(2,*,rec=c)child
      close(unit=2)
      write(6,10603)parent,child,qty
10603 format(5x,a10,3x,a10,6x,i4)
      if(pkey=-1) write(6,10601)
      enddo
   close(unit=6)
   return
   end
*
   subroutine MRPflow
*  The part-by-part flow pattern is printed out.
   character c,answer
   integer  first,SR,BOM,BofM,ccper,pup,XOkey,SPkey
10 print,'Has level number been assigned? yes or no.'
   read,answer
   if answer = 'yes'
      go to 20
   elseif answer = 'no'
      go to 999
   else
      go to 10
   endif
20 open(unit=6,file='printer')
   open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
   read(3,10302,rec=0)next,first
10302  format(2i3)
   close(unit=3)
   write(6,10600)
10600  format(5x,'Parts sorted by MRP flow order.')
   k = first
30 open(unit=3,file='(f:100)partstatus,rel',access='DIRECT')
   read(3,*,rec=k)oh,SR,numkids,BOM,BofM,lead,lotr,lotp,scost,
&     ucost,ccost,ccper,pu,pup,MPSkey,XOkey,SPkey,level,
&      levelkey
   close(unit=3)
   open(unit=2,file='(f:35)partid,rel',access='DIRECT',recl=35)
   read(2,*,rec=k)c
   close(unit=2)
   write(6,10601)c
10601  format(10x,a10)
   if (levelkey=-1) go to 999
   k = levelkey
   go to 30
999 close(unit=6)
   return
   end
```

APPENDIX C

FLOWCHARTS

Flowchart
Symbols

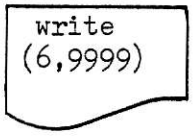
Symbol	Meaning
	terminal; start or stop
	output to the CRT screen
	input from the keyboard
	decision
	connector
	computation; process commands, etc.
	comments
	read/write from disk file or keyboard; x=unit number and y=record number
	output for the printer

Symbol	Meaning
	do loop*

Flowchart
Abbreviations

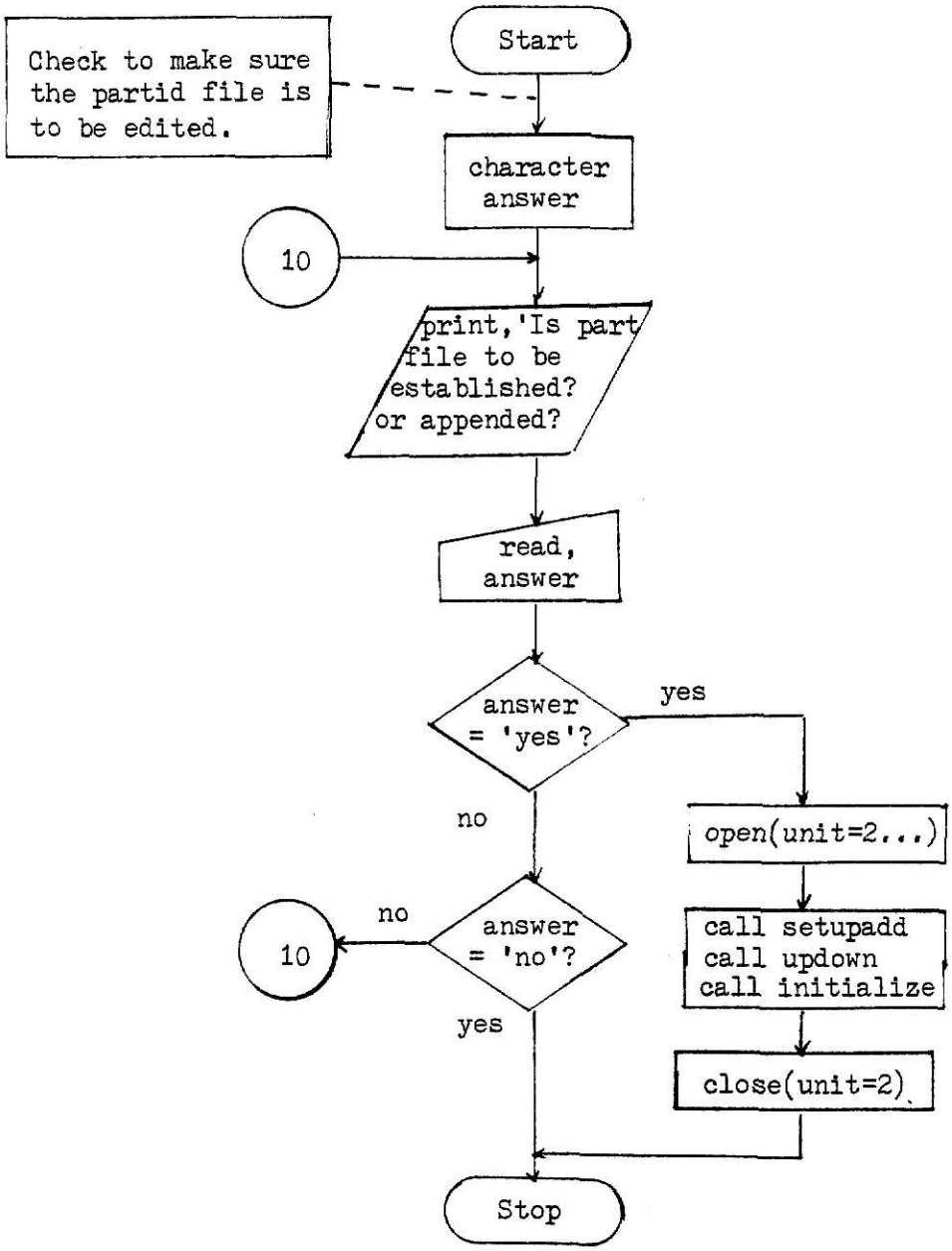
Abbreviation	Meaning
read(...) or write(...)	The (...) = (unit=x, format number, rec=y); x=integer for the unit number; y=integer variable for the record number
open(unit=x...)	x=unit number (integer)
(unit=x...)	Equals (unit=x,file="name", access='DIRECT',recl=z); x=unit number; "name" = the file name; z= record length

*This symbol appears in Applied FORTRAN Programming with Standard FORTRAN, WATFOR,WATFIV, and Structured WATFIV. (Belmont, Cal.: Wadsworth Publishing Company, Inc., 1977) p. 159, by Michael J. Merchant and John R. Sturgul.

Abbreviation	Meaning
	9999=format code number;
oh,...,levelkey	oh,SR,numkids,BofM,BOM, lead,lotr,lotp,scost, ucost,ccost,ccper,pu, pup,MPSkey,xorderkey, servpartkey,level, levelkey
period,...,OR	period,date,dMPS,dXO,dSP, sumDD,GR,SR,OH,NR,LR,OR

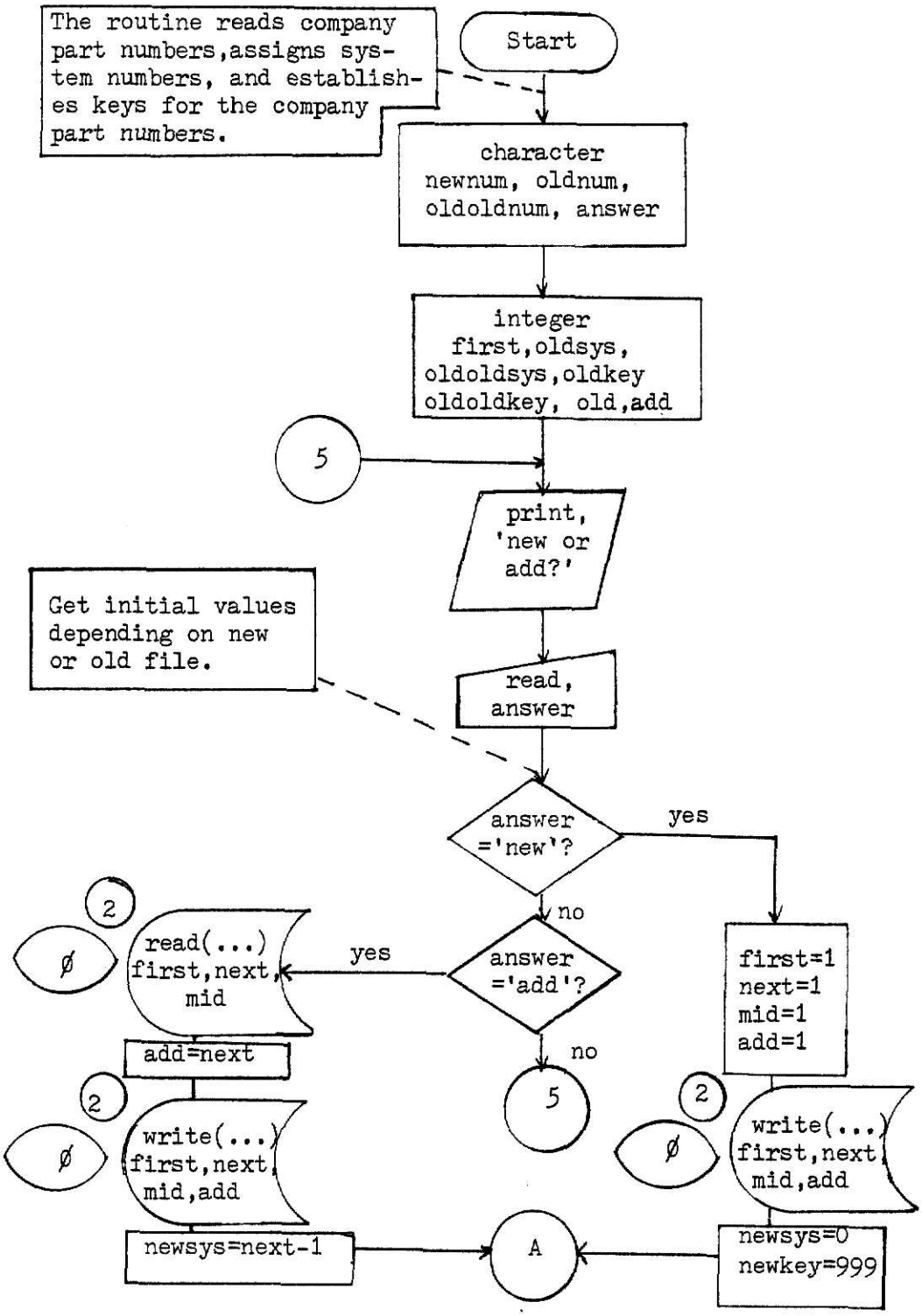
PartFile

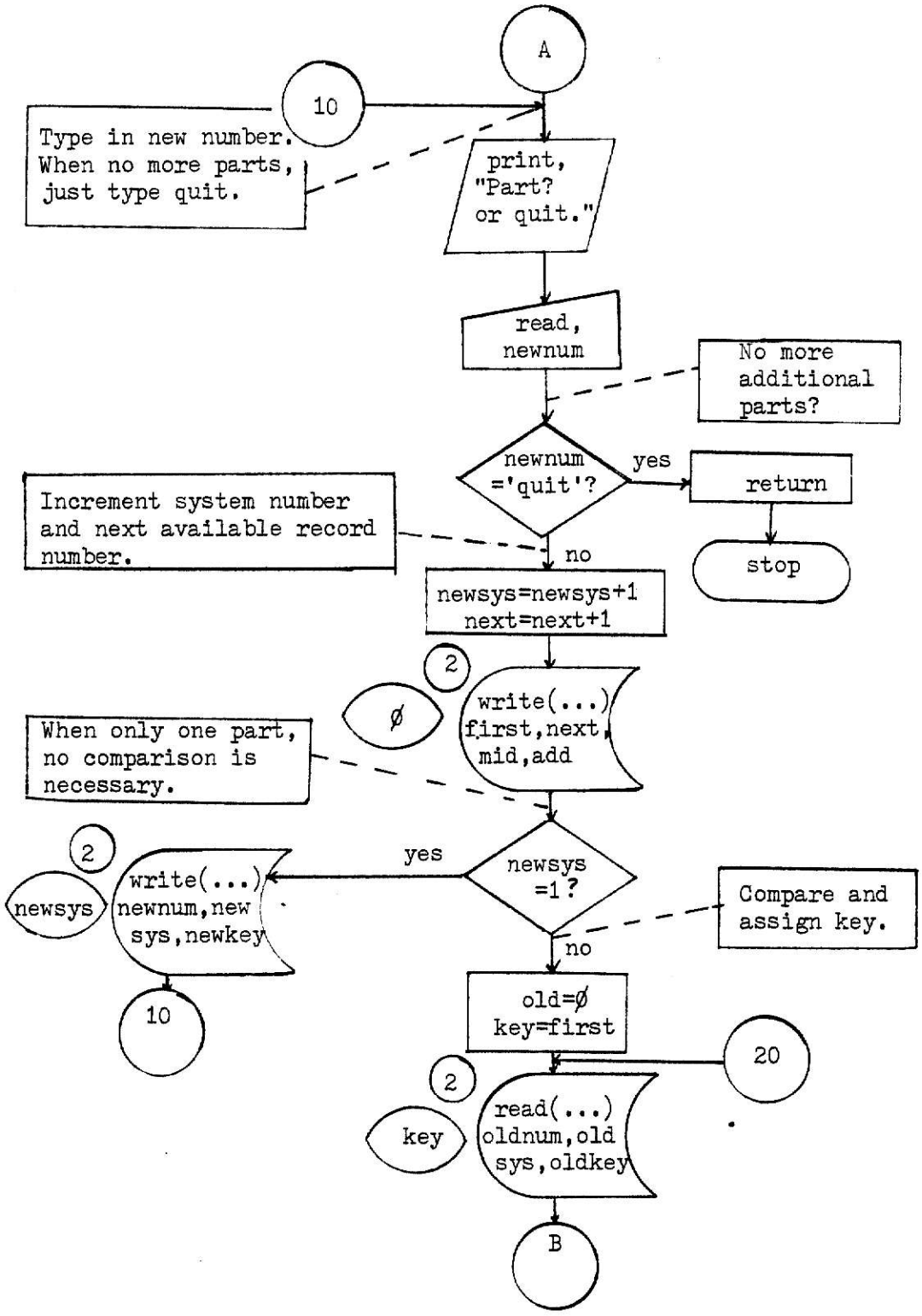
-Main Program-

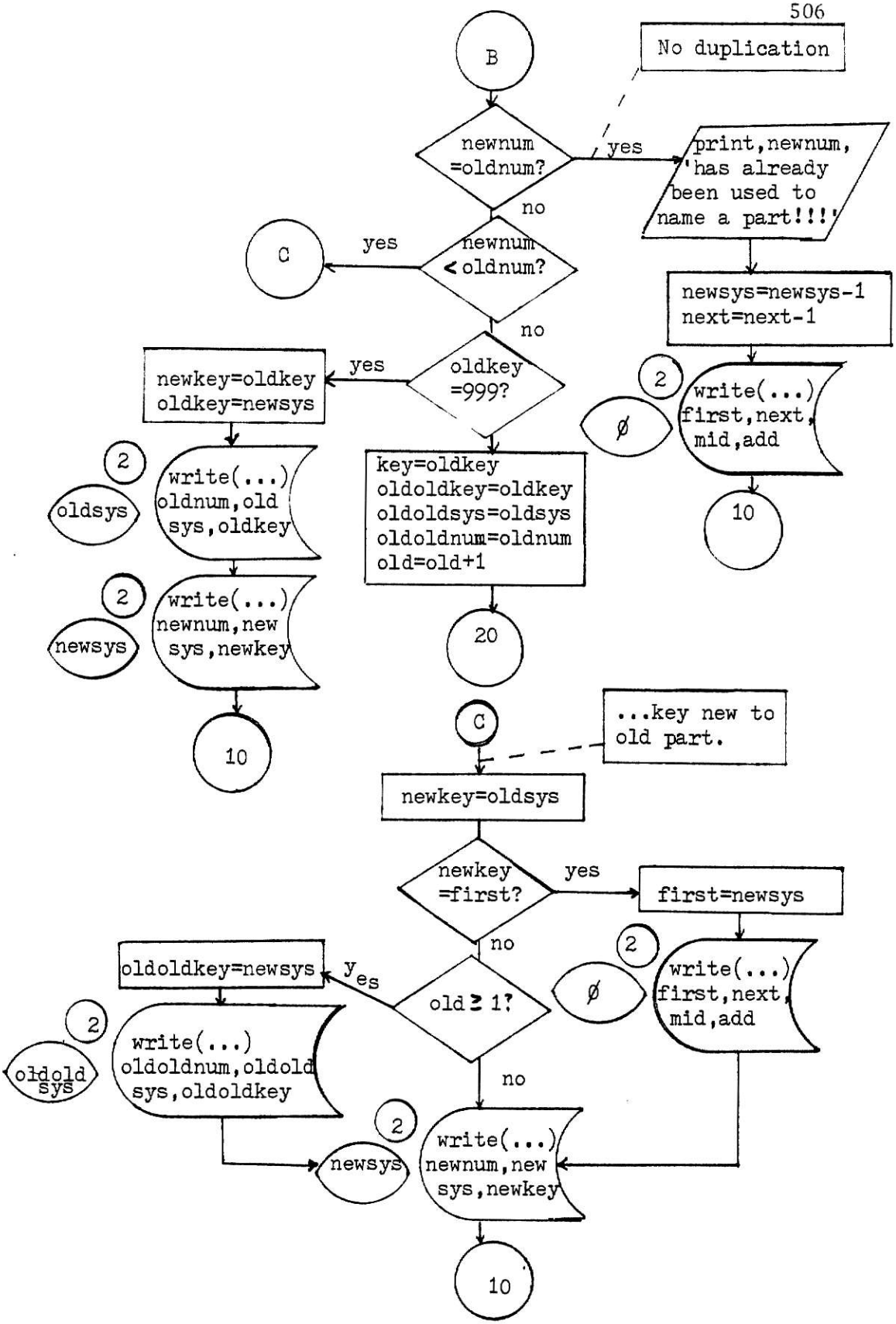


PartFile

-Subroutine setupadd-

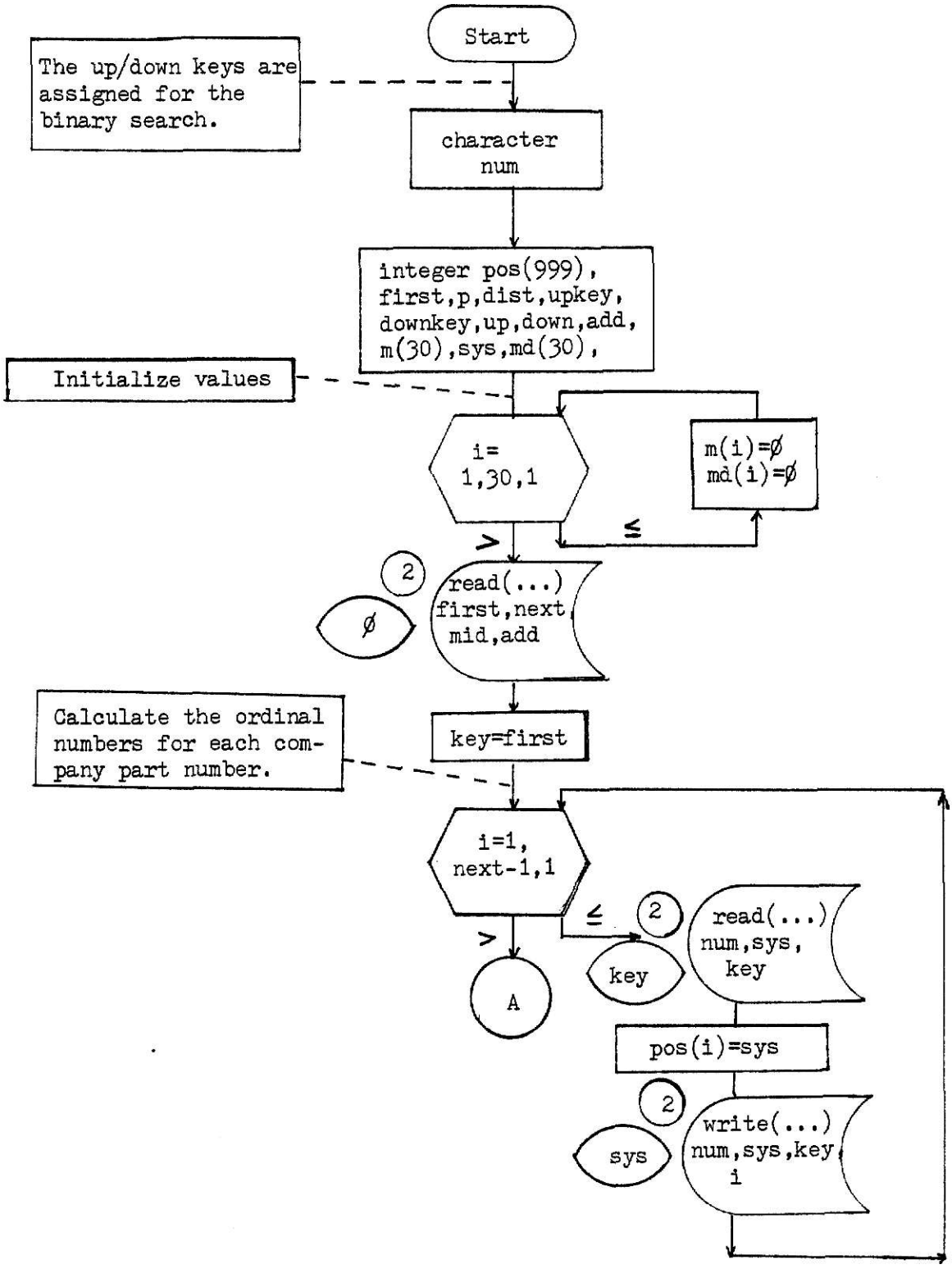


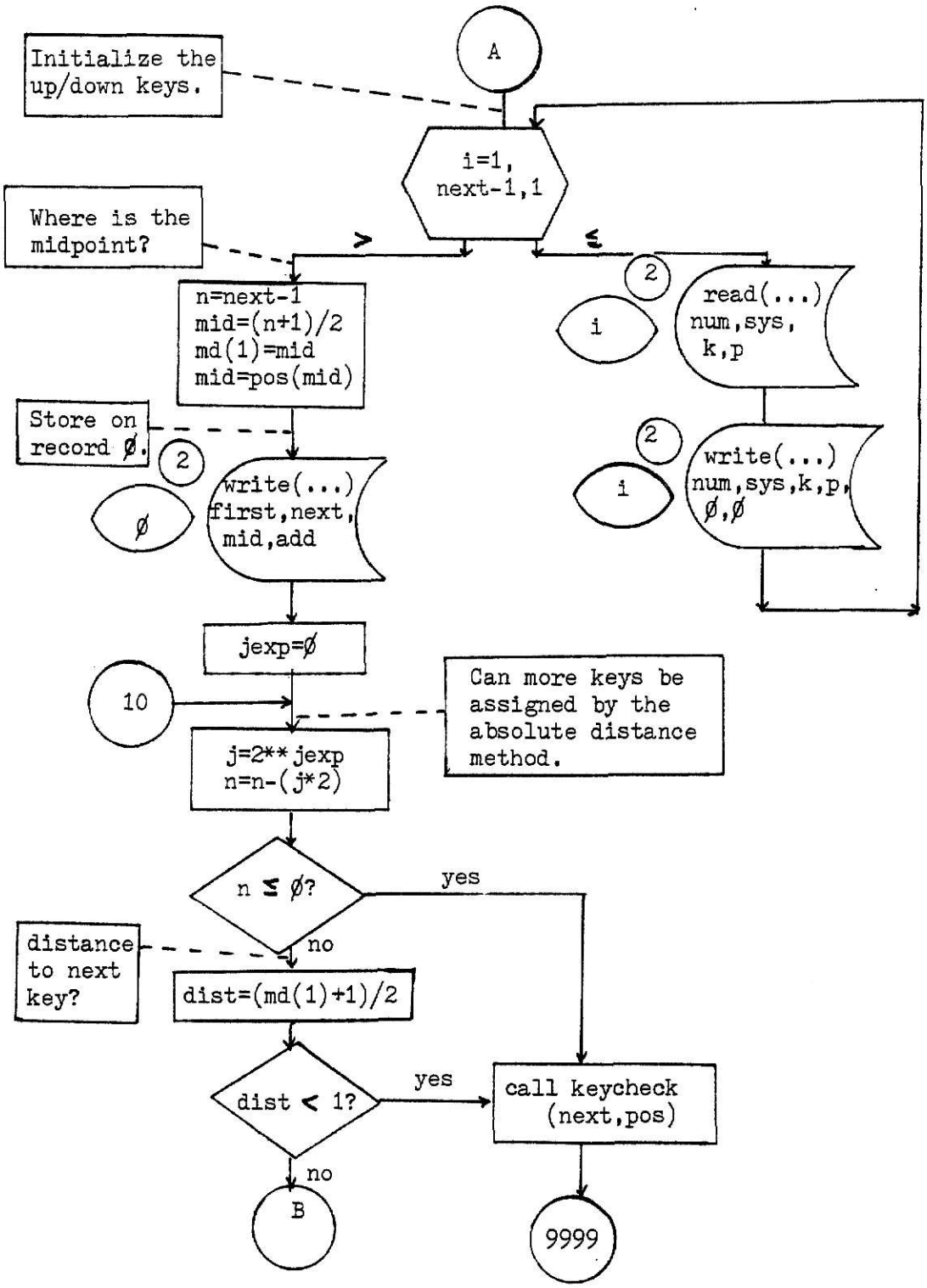

A
10
Type in new number.
When no more parts,
just type quit.
print,
"Part?
or quit."
read,
newnum
No more
additional
parts?
newnum
='quit'?
yes
return
stop
Increment system number
and next available record
number.
no
newsys=newsys+1
next=next+1
2
ø
write(...)
first,next,
mid,add
When only one part,
no comparison is
necessary.
2
newsys
write(...)
newnum,new
sys,newkey
yes
newsys
=1?
Compare and
assign key.
no
10
old=ø
key=first
20
2
key
read(...)
oldnum,old
sys,oldkey
B

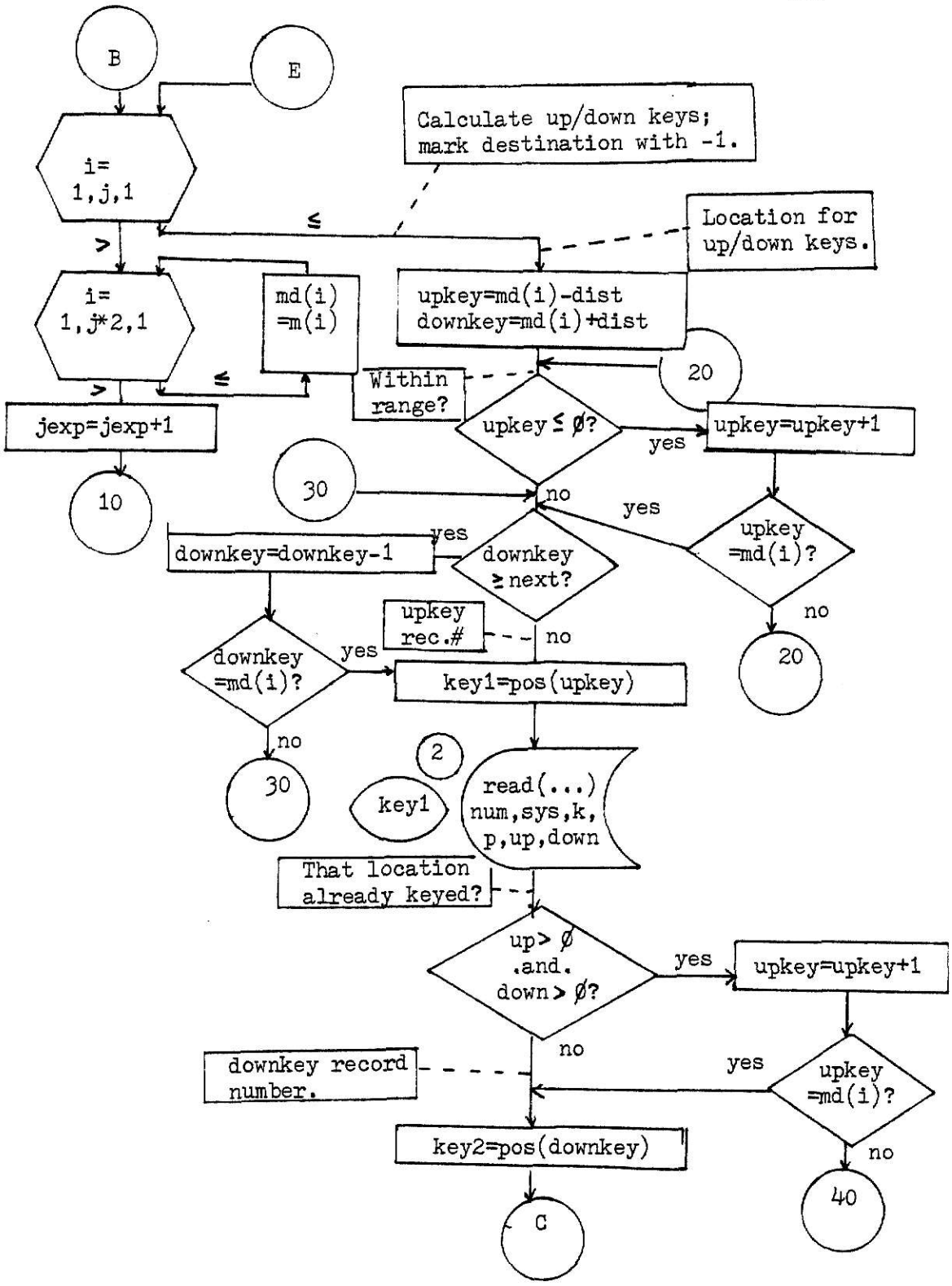

B
No duplication
newnum =oldnum?
yes
print,newnum, 'has already been used to name a part!!!'
no
C
yes
newnum < oldnum?
newsys=newsys-1
next=next-1
no
newkey=oldkey
oldkey=newsys
yes
oldkey =999?
2
write(...) first,next, mid,add
ø
10
2
write(...) oldnum,old sys,oldkey
oldsys
key=oldkey
oldoldkey=oldkey
oldoldsys=oldsys
oldoldnum=oldnum
old=old+1
2
write(...) newnum,new sys,newkey
newsys
20
10
C
...key new to old part.
newkey=oldsys
newkey =first?
yes
first=newsys
no
oldoldkey=newsys
yes
old ≥ 1?
2
write(...) first,next, mid,add
ø
2
write(...) oldoldnum,oldold sys,oldoldkey
oldold sys
no
2
write(...) newnum,new sys,newkey
newsys
10

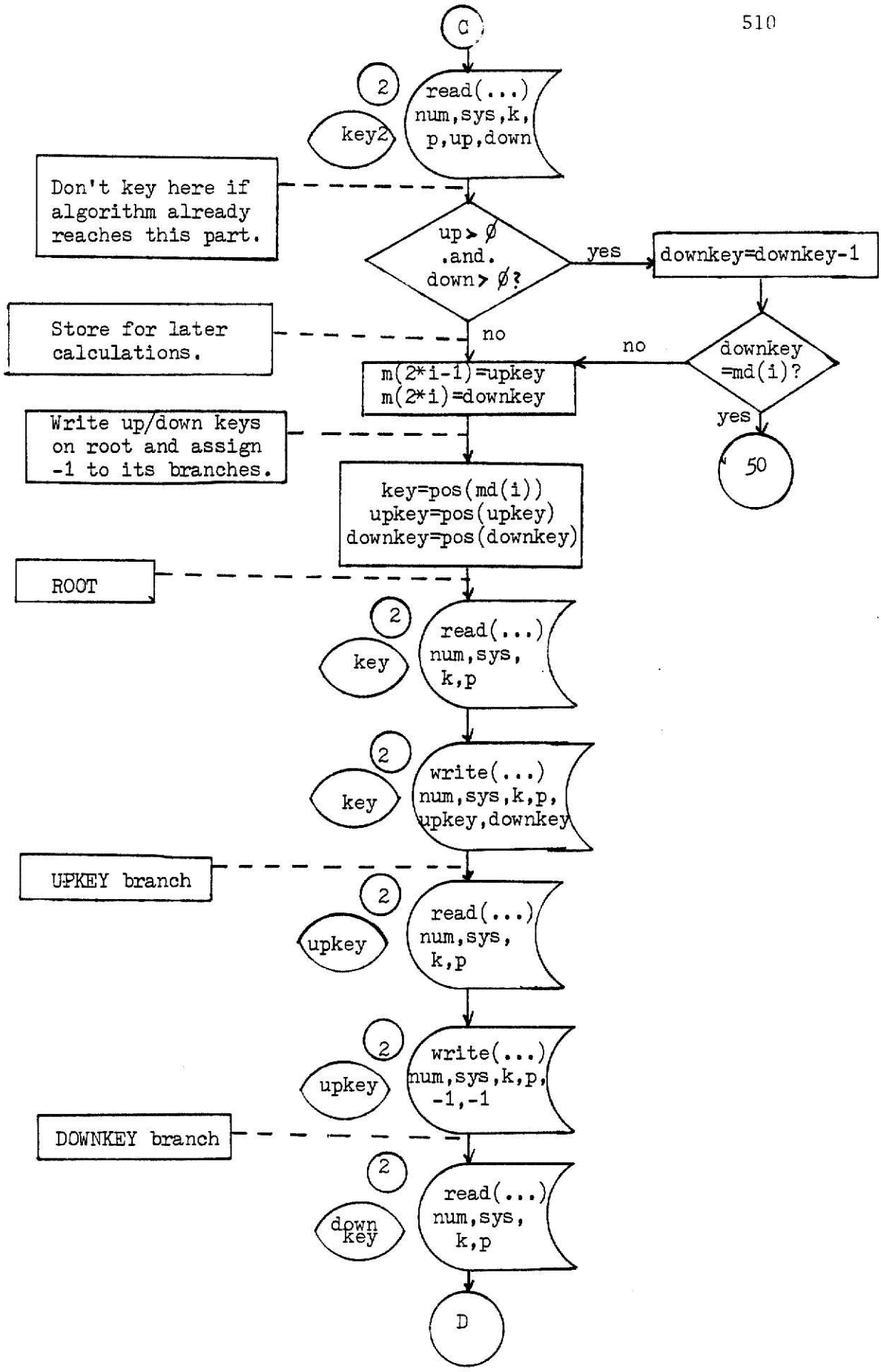
PartFile

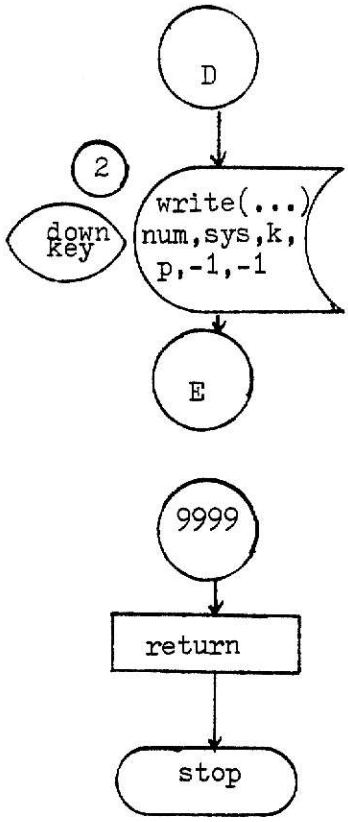
-Subroutine updown-




A
Initialize the up/down keys.
i=1, next-1,1
Where is the midpoint?
>
≤
n=next-1
mid=(n+1)/2
md(1)=mid
mid=pos(mid)
2
i
read(...) num,sys, k,p
Store on record ∅.
2
∅
write(...) first,next, mid,add
2
i
write(...) num,sys,k,p, ∅,∅
jexp=∅
10
Can more keys be assigned by the absolute distance method.
j=2** jexp
n=n-(j*2)
n ≤ ∅?
yes
no
distance to next key?
dist=(md(1)+1)/2
dist < 1?
yes
call keycheck (next,pos)
no
B
9999

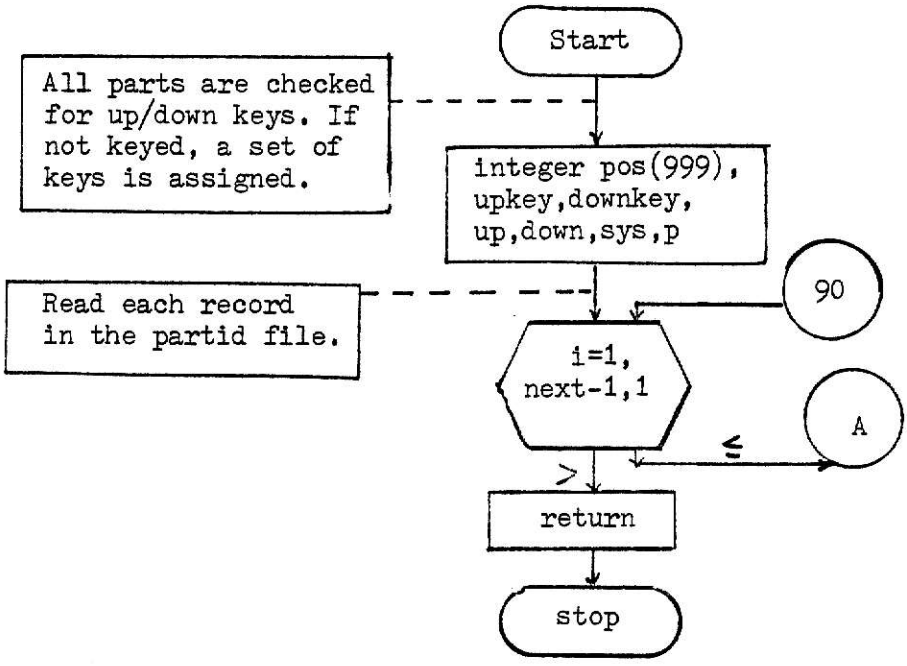

B
E
Calculate up/down keys; mark destination with -1.
i= 1,j,1
≤
Location for up/down keys.
>
i= 1,j*2,1
md(i) =m(i)
upkey=md(i)-dist
downkey=md(i)+dist
20
>
≤
Within range?
upkey ≤ ∅?
yes
upkey=upkey+1
jexp=jexp+1
10
30
no
yes
yes
downkey=downkey-1
downkey ≥ next?
upkey =md(i)?
no
upkey rec.#
downkey =md(i)?
yes
no
key1=pos(upkey)
20
no
30
2
key1
read(...) num,sys,k, p,up,down
That location already keyed?
up> ∅ .and. down> ∅?
yes
upkey=upkey+1
no
yes
downkey record number.
upkey =md(i)?
key2=pos(downkey)
no
40
C

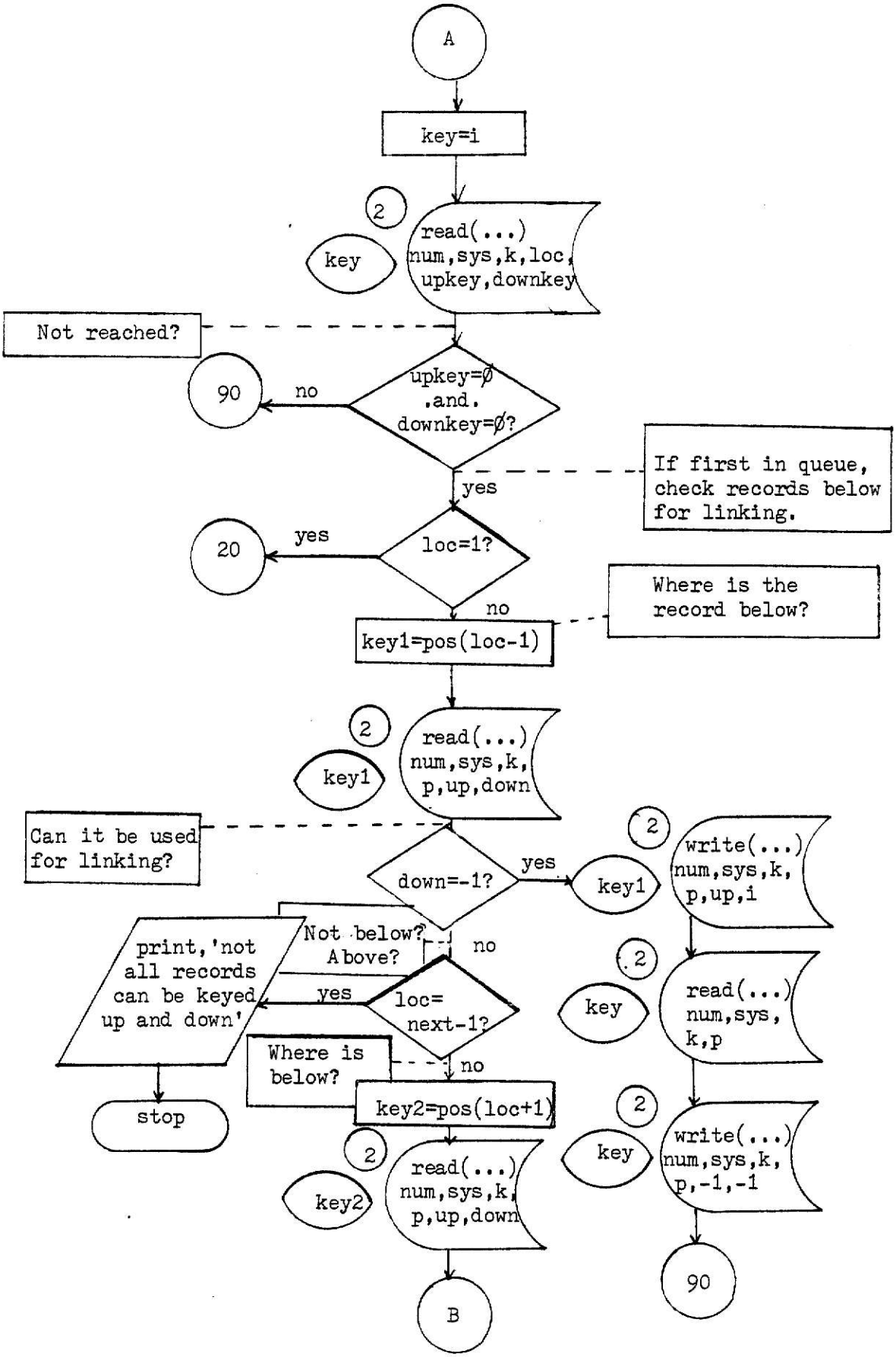

C
2
key2
read(...)
num,sys,k,
p,up,down
Don't key here if algorithm already reaches this part.
up > ∅
.and.
down > ∅?
yes
downkey=downkey-1
Store for later calculations.
no
no
downkey
=md(i)?
m(2*i-1)=upkey
m(2*i)=downkey
yes
50
Write up/down keys on root and assign -1 to its branches.
key=pos(md(i))
upkey=pos(upkey)
downkey=pos(downkey)
ROOT
2
key
read(...)
num,sys,
k,p
2
key
write(...)
num,sys,k,p,
upkey,downkey
UPKEY branch
2
upkey
read(...)
num,sys,
k,p
2
upkey
write(...)
num,sys,k,p,
-1,-1
DOWNKEY branch
2
down
key
read(...)
num,sys,
k,p
D

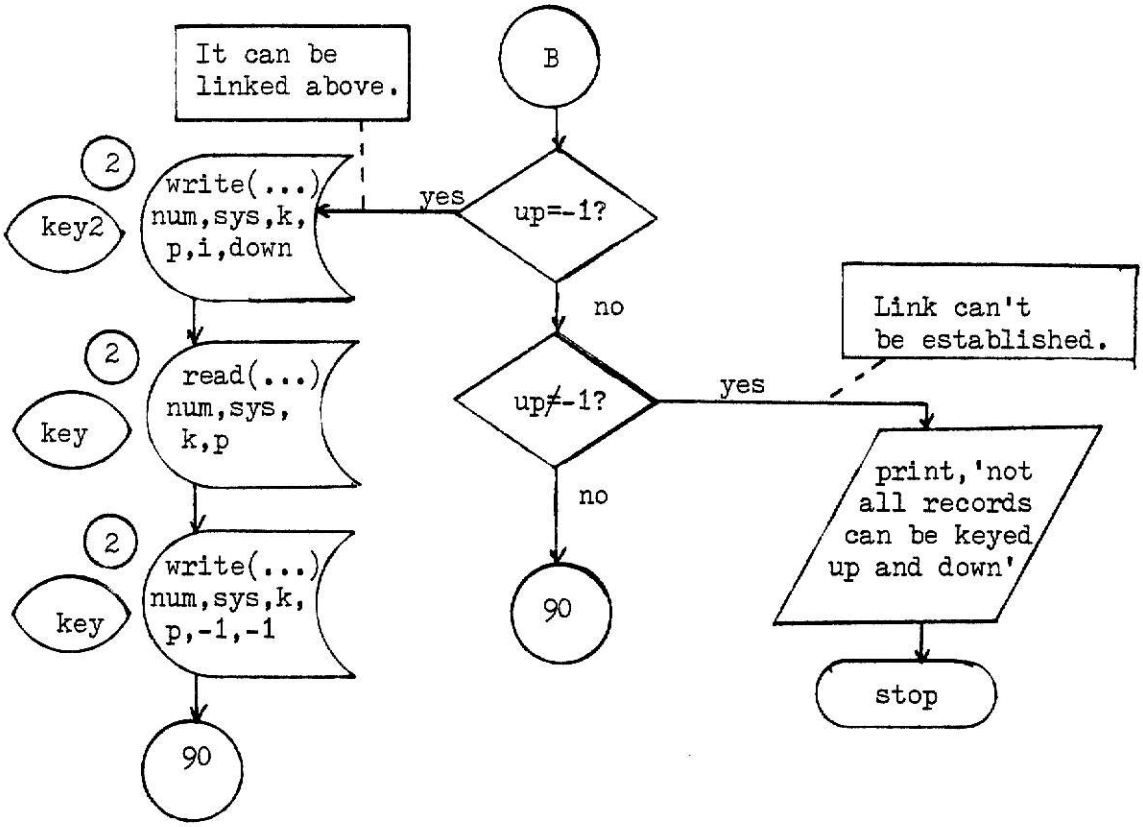


PartFile

-Subroutine keycheck-
Arguments: next,pos

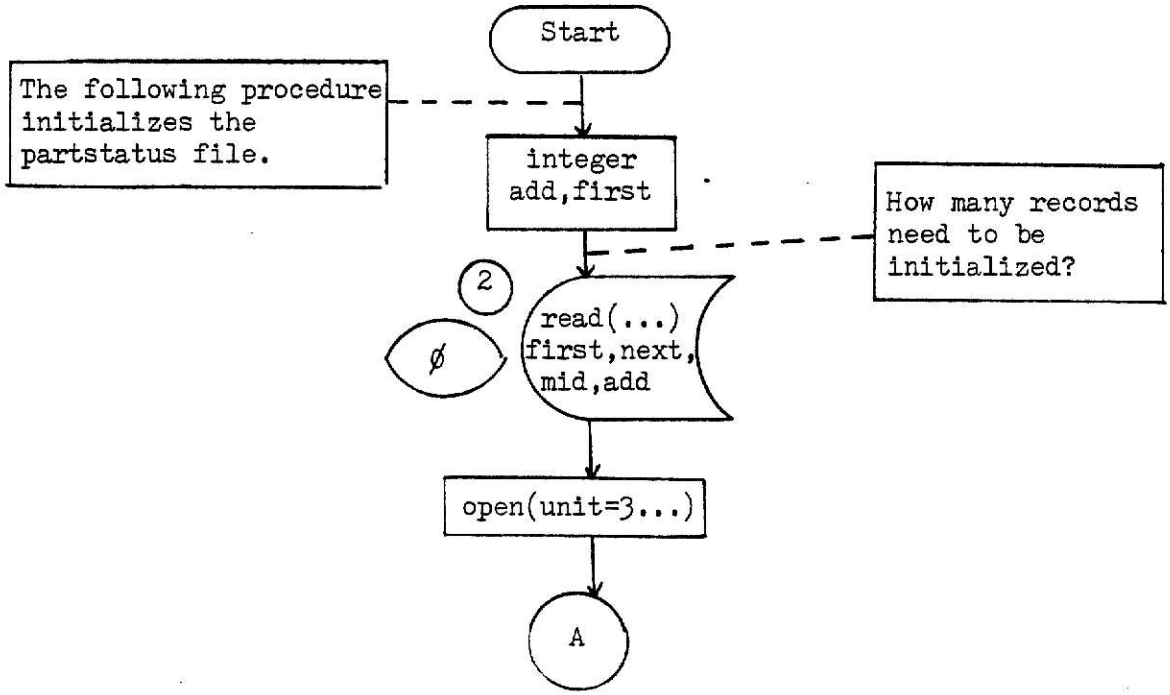


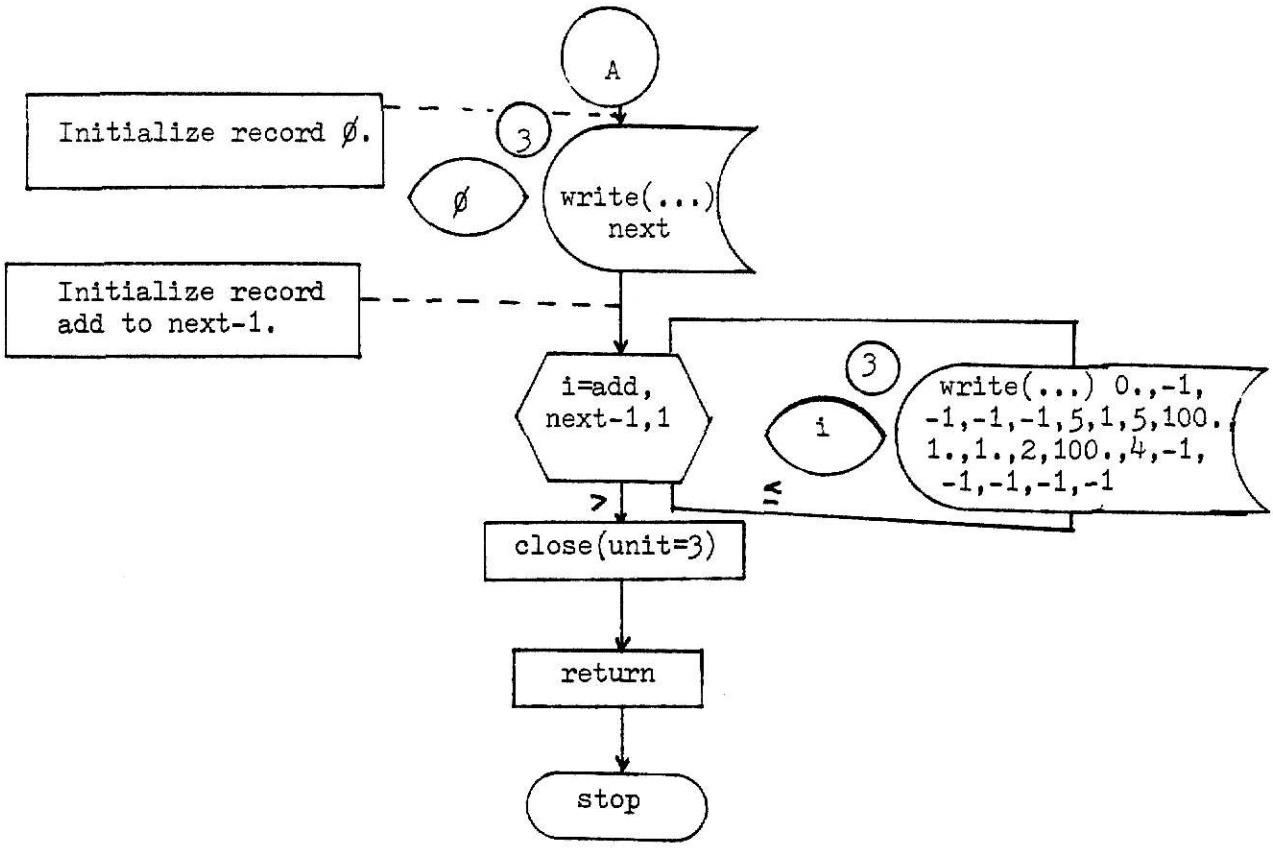

A
key=i
2
key
read(...)
num,sys,k,loc,
upkey,downkey
Not reached?
90
no
upkey=ø
.and.
downkey=ø?
If first in queue,
check records below
for linking.
yes
20
yes
loc=1?
Where is the
record below?
no
key1=pos(loc-1)
2
key1
read(...)
num,sys,k,
p,up,down
Can it be used
for linking?
down=-1?
yes
key1
2
write(...)
num,sys,k,
p,up,i
print,'not
all records
can be keyed
up and down'
Not below?
Above?
no
yes
loc=
next-1?
2
key
read(...)
num,sys,
k,p
Where is
below?
no
key2=pos(loc+1)
stop
2
key
write(...)
num,sys,k,
p,-1,-1
2
key2
read(...)
num,sys,k,
p,up,down
90
B



PartFile

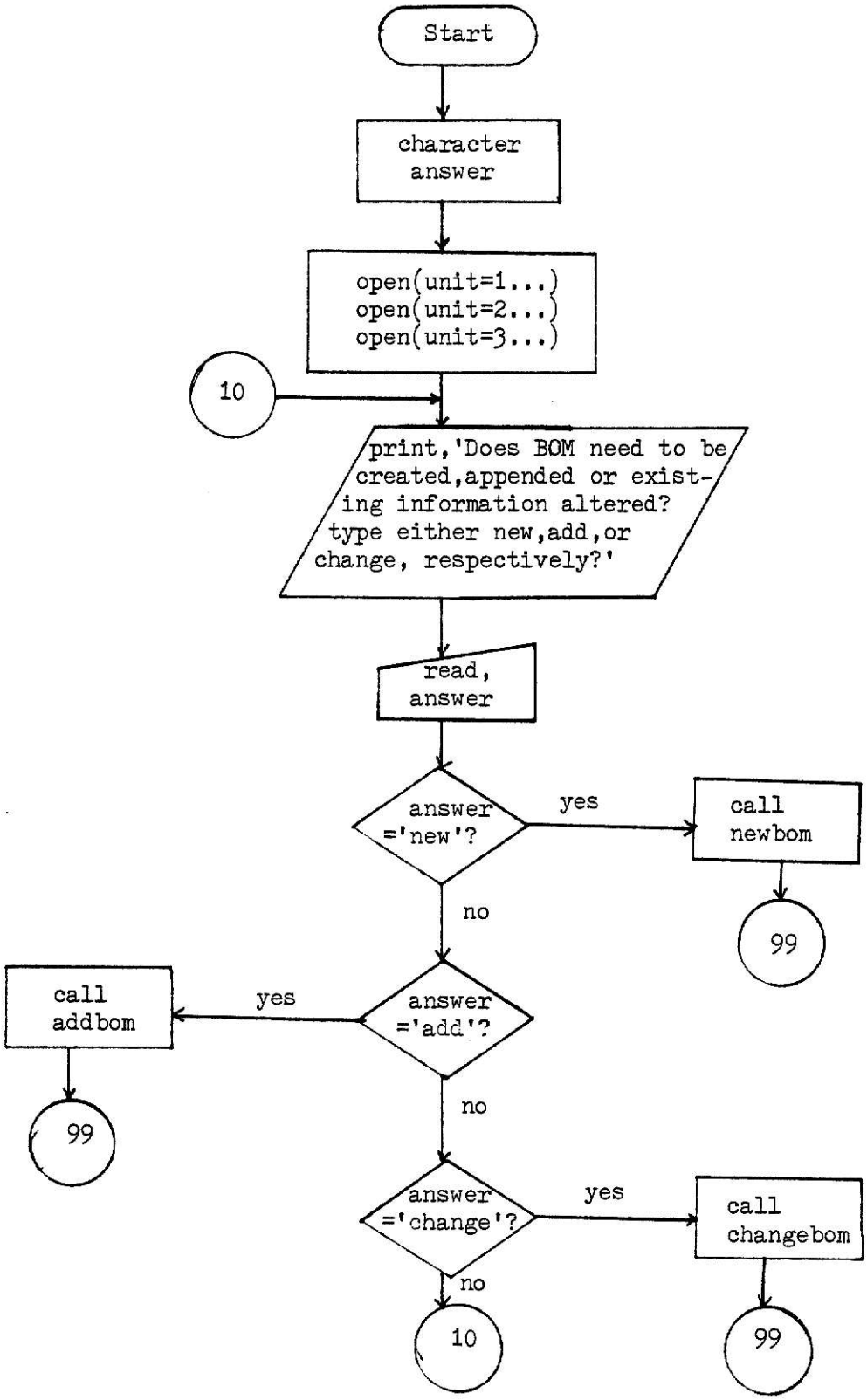
-Subroutine initialize-

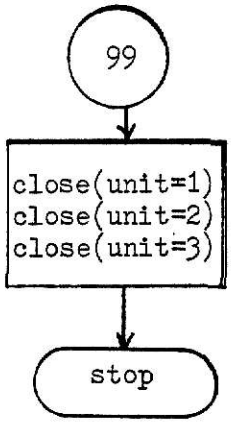



A
Initialize record ∅.
3
∅
write(...)
next
Initialize record
add to next-1.
i=add,
next-1,1
3
i
write(...) 0.,-1,
-1,-1,-1,5,1,5,100.,
1.,1.,2,100.,4,-1,
-1,-1,-1,-1
>
≤
close(unit=3)
return
stop

BOMfile

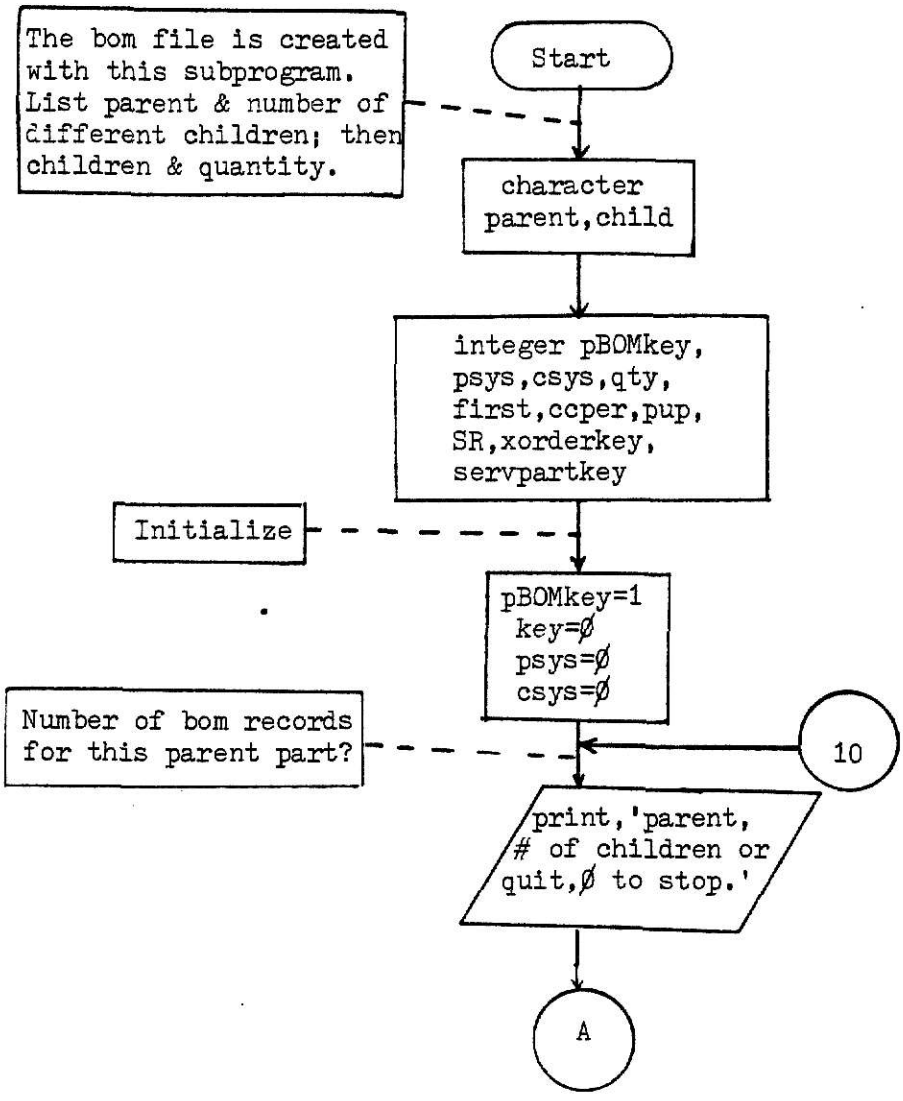
-Main Program-

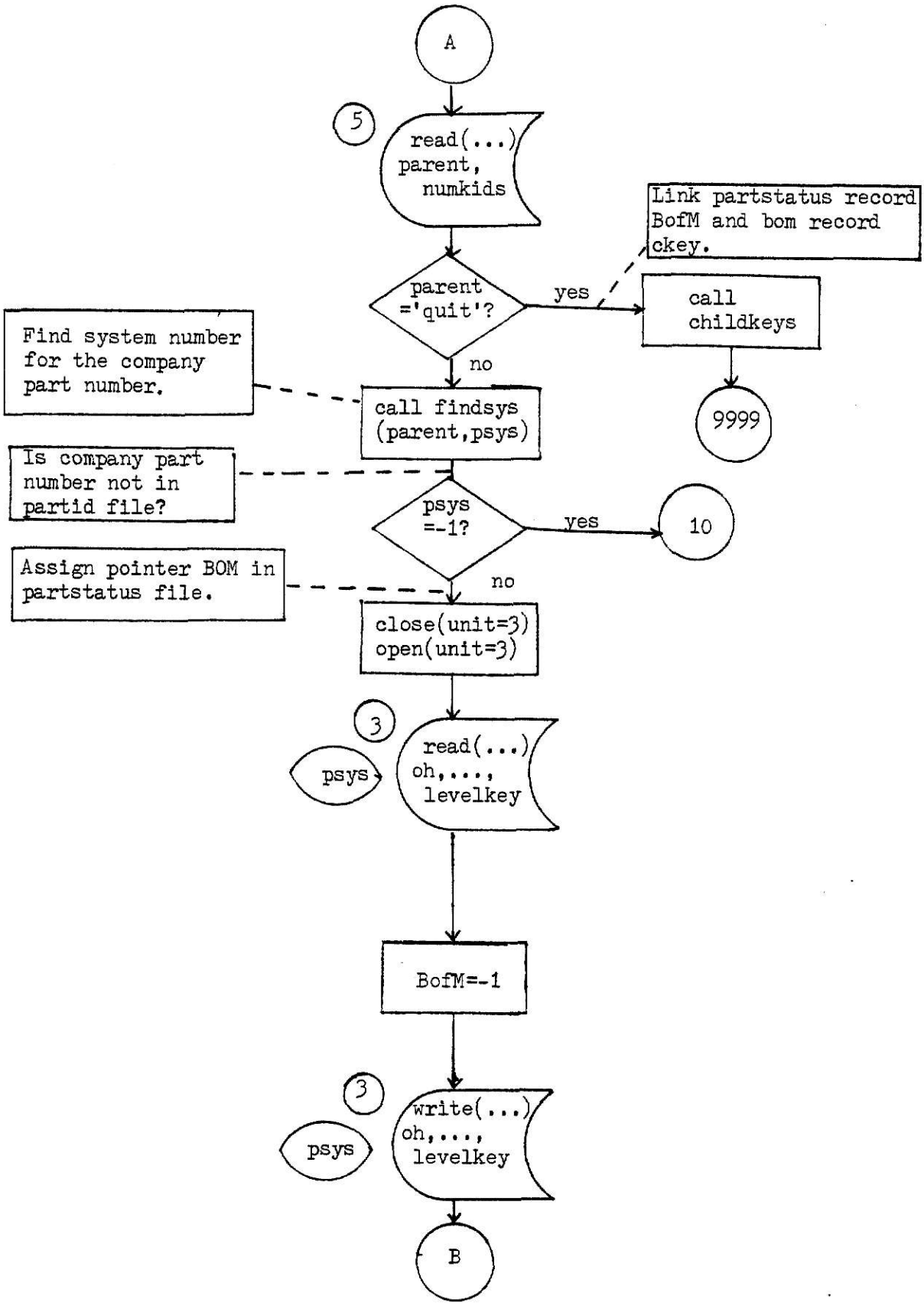


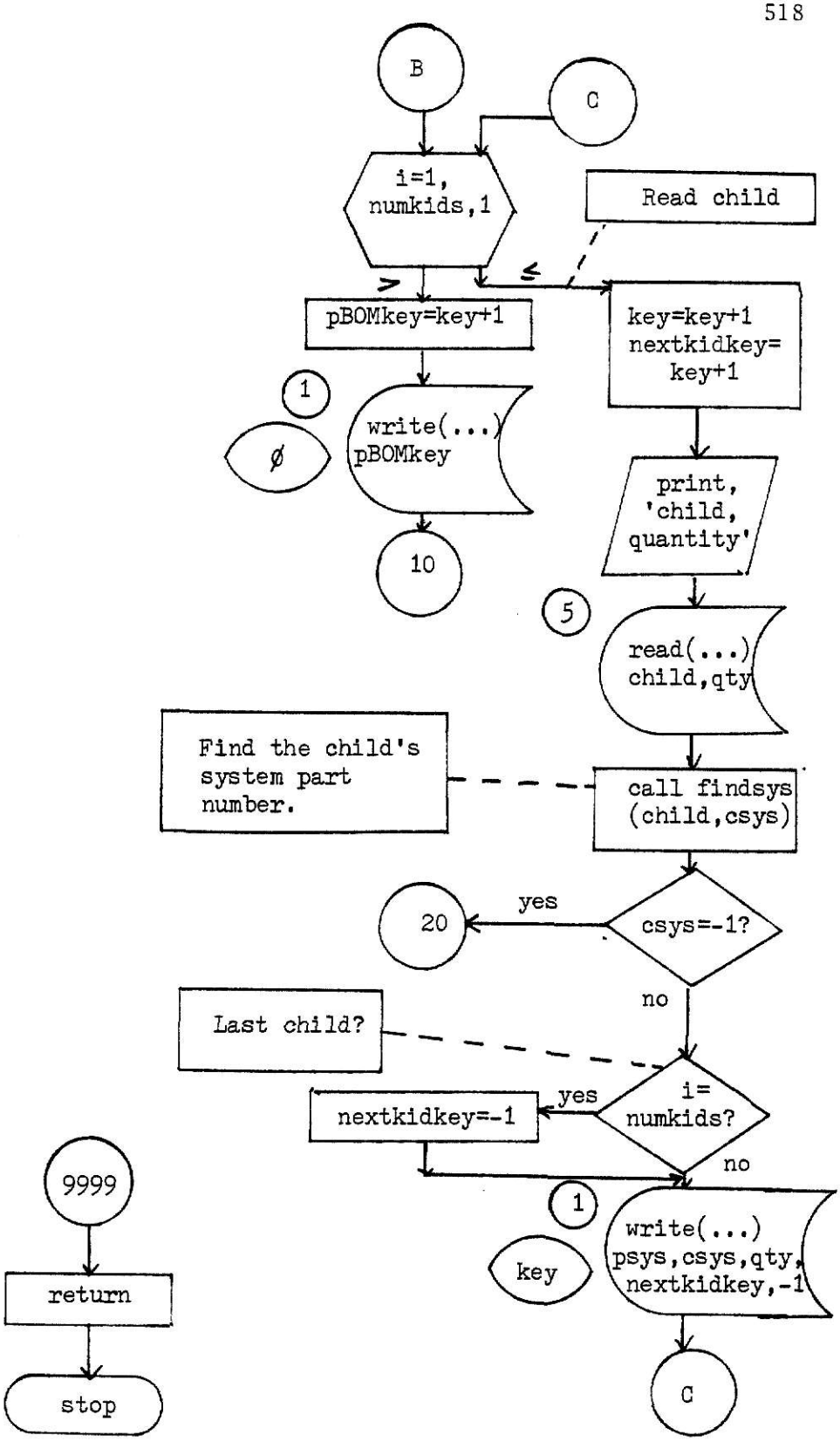


BOMfile

-Subroutine newbom-

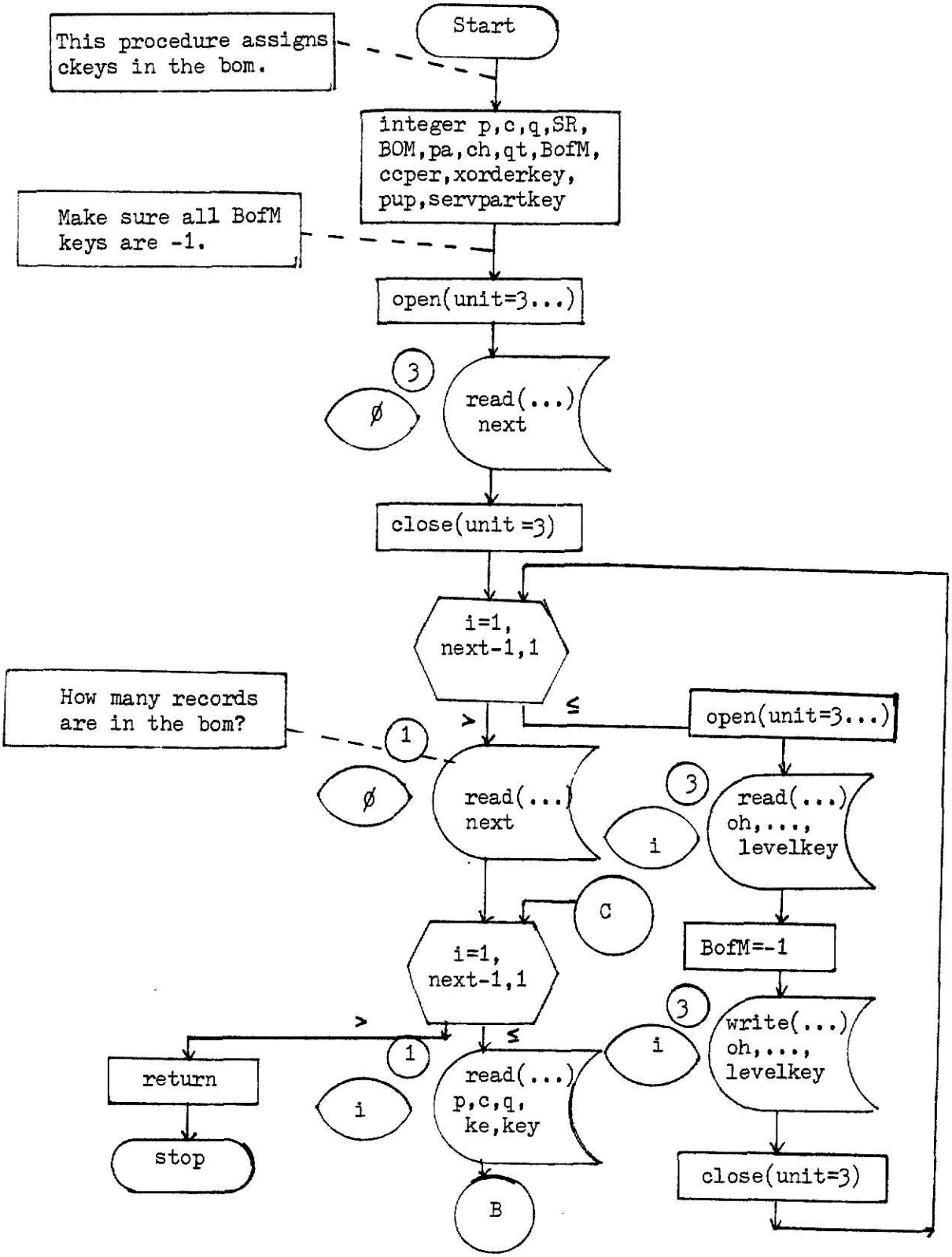


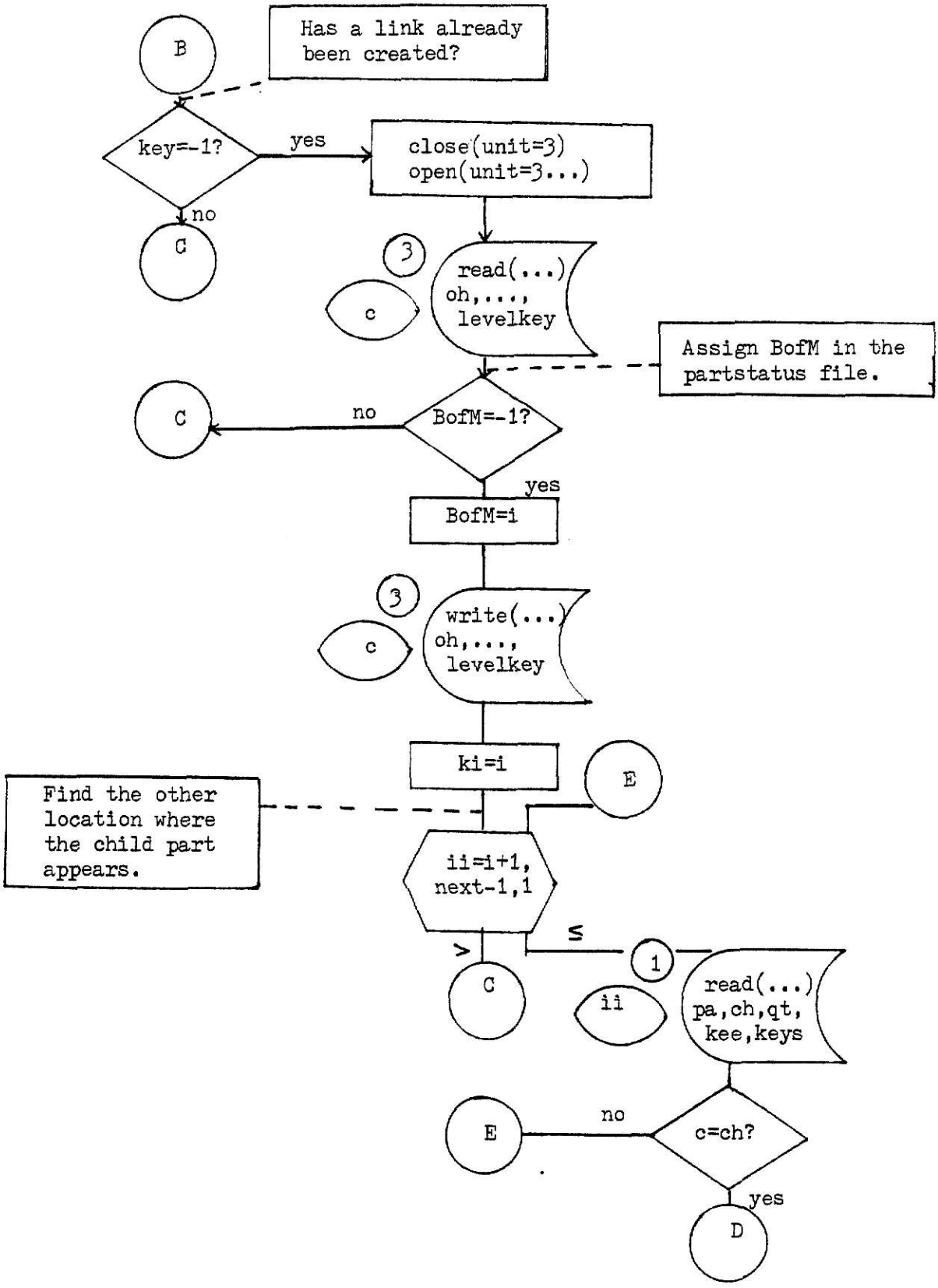

A
5
read(...)
parent,
numkids
Link partstatus record
BofM and bom record
ckey.
parent
='quit'?
yes
call
childkeys
Find system number
for the company
part number.
no
call findsys
(parent,psys)
9999
Is company part
number not in
partid file?
psys
=-1?
yes
10
Assign pointer BOM in
partstatus file.
no
close(unit=3)
open(unit=3)
3
psys
read(...)
oh,...,
levelkey
BofM=-1
3
psys
write(...)
oh,...,
levelkey
B

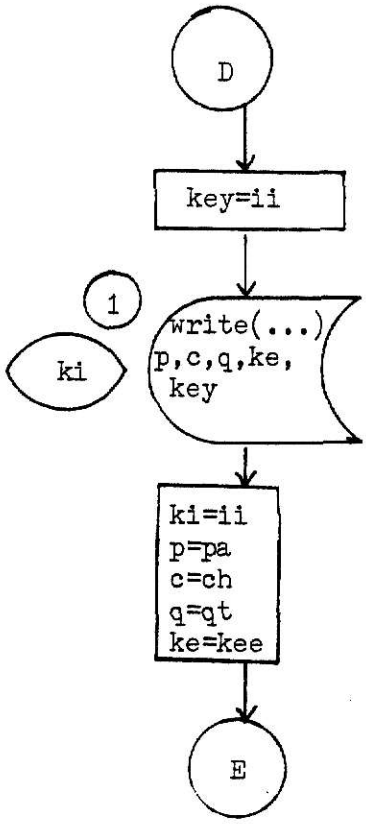

B
C
i=1,
numkids,1
Read child
>
≤
pBOMkey=key+1
key=key+1
nextkidkey=
key+1
1
ø
write(...)
pBOMkey
10
print,
'child,
quantity'
5
read(...)
child,qty
Find the child's
system part
number.
call findsys
(child,csys)
20
yes
csys=-1?
no
Last child?
i=
numkids?
yes
nextkidkey=-1
no
1
key
write(...)
psys,csys,qty,
nextkidkey,-1
C
9999
return
stop

BOMfile

-Subroutine childkeys-

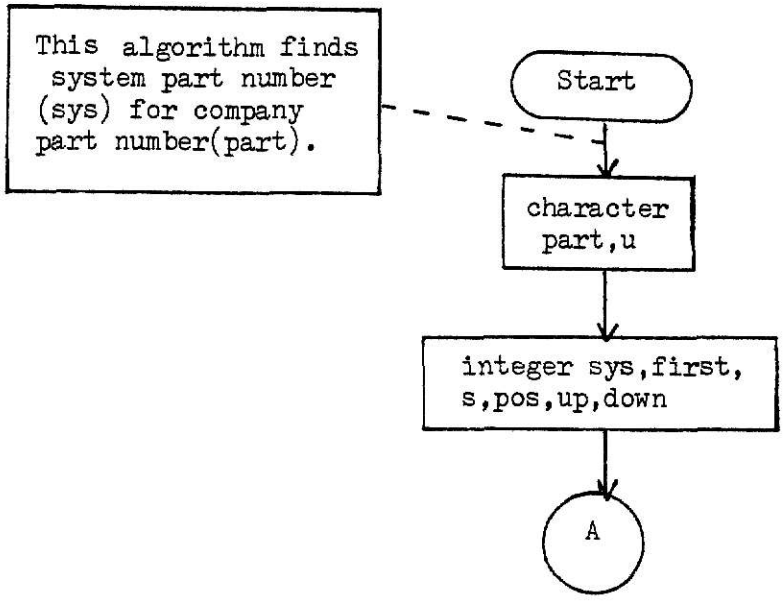


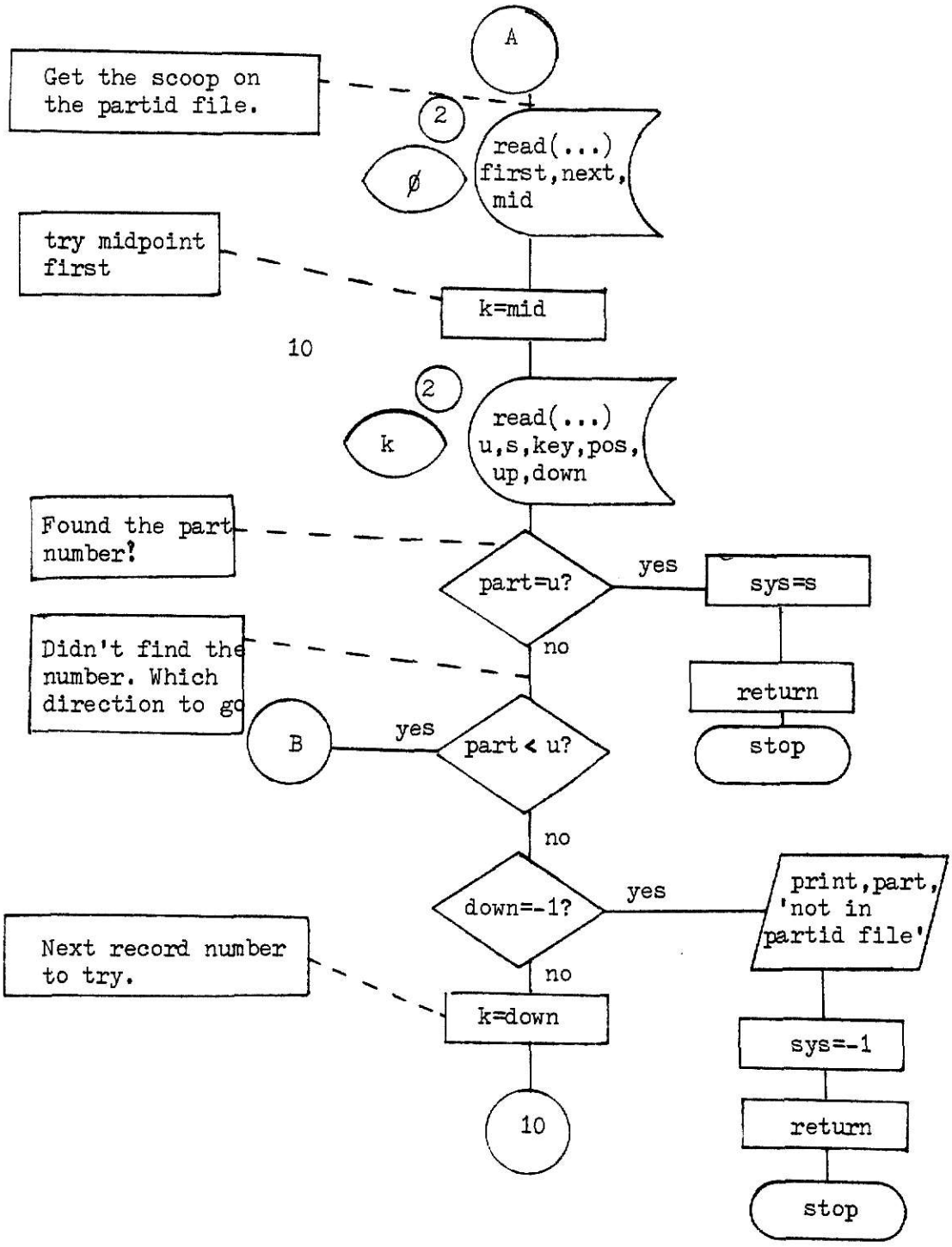

B
Has a link already been created?
key=-1?
yes
close(unit=3)
open(unit=3...)
no
C
3
c
read(...)
oh,...,
levelkey
Assign BofM in the partstatus file.
C
no
BofM=-1?
yes
BofM=i
3
c
write(...)
oh,...,
levelkey
ki=i
E
Find the other location where the child part appears.
ii=i+1,
next-1,1
>
≤
C
1
ii
read(...)
pa,ch,qt,
kee,keys
E
no
c=ch?
yes
D

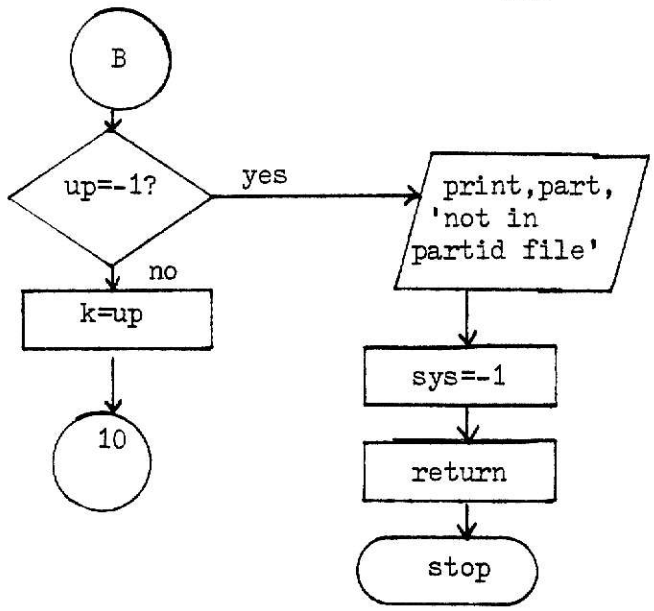


BOMfile

-Subroutine findsys
Arguments: part,sys

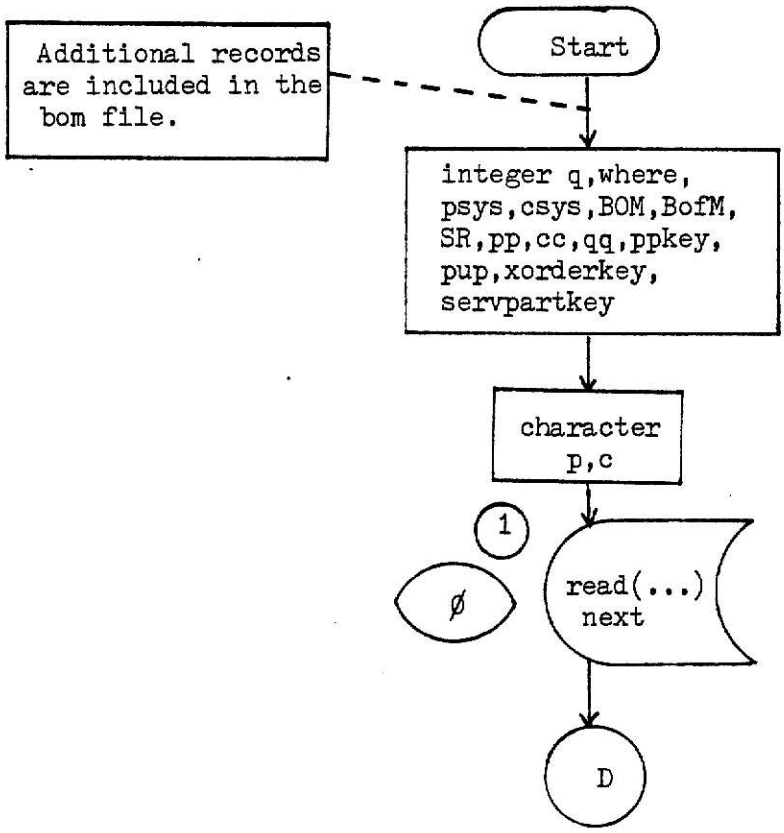


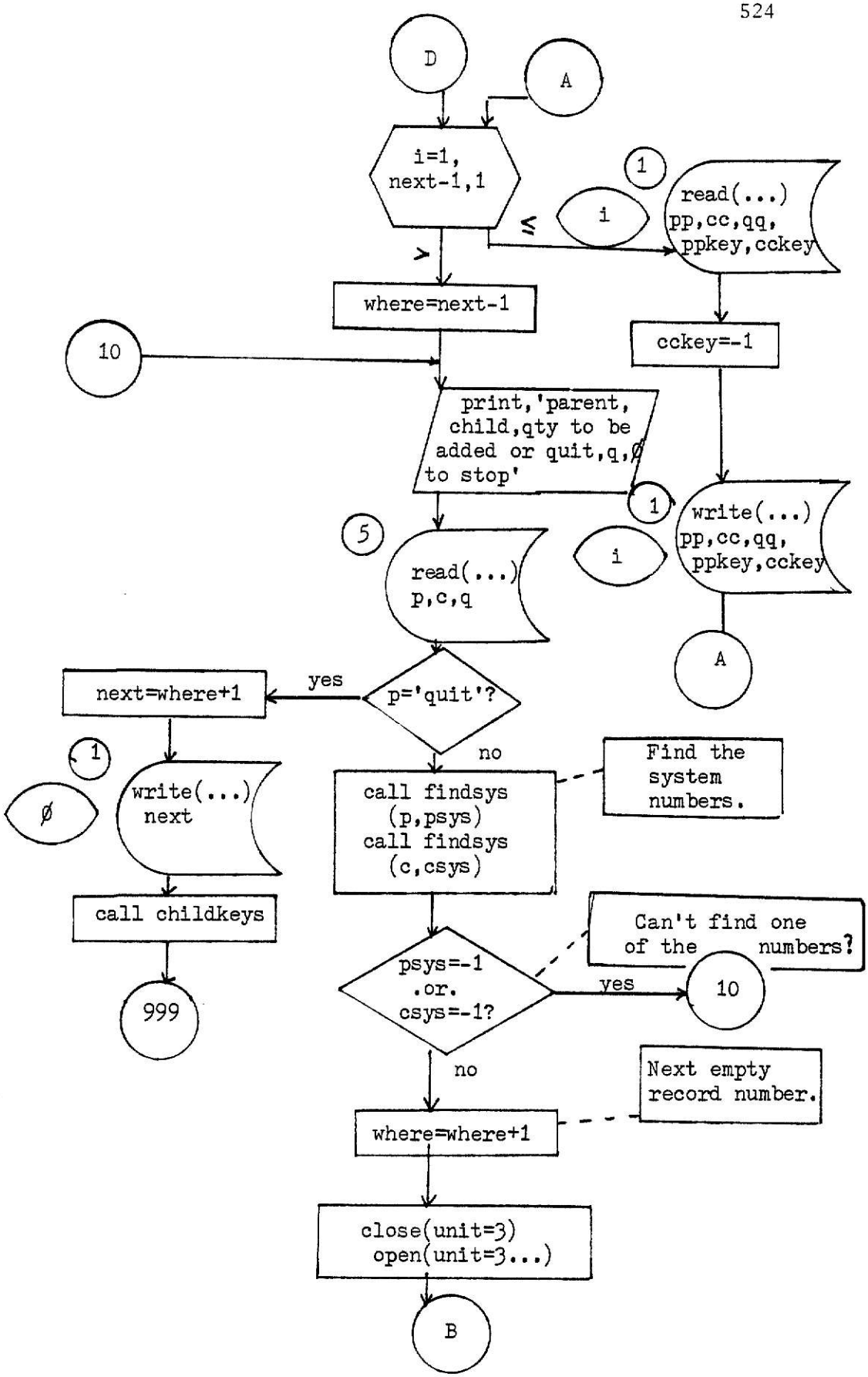

A
Get the scoop on the partid file.
2
ø
read(...) first,next, mid
try midpoint first
k=mid
10
2
k
read(...) u,s,key,pos, up,down
Found the part number!
part=u?
yes
sys=s
return
stop
Didn't find the number. Which direction to go
no
B
yes
part < u?
no
down=-1?
yes
print,part, 'not in partid file'
Next record number to try.
no
k=down
sys=-1
return
stop
10

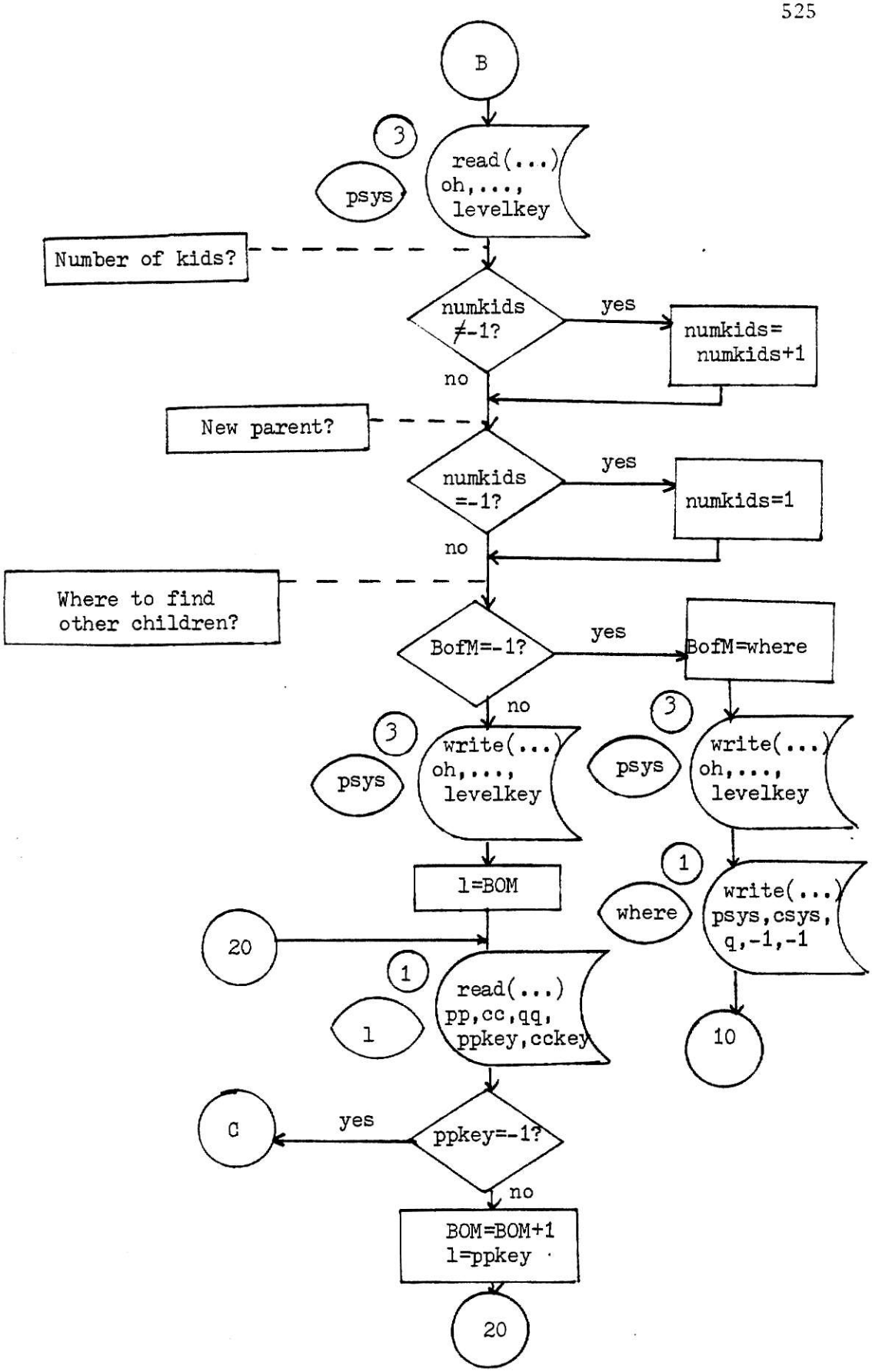


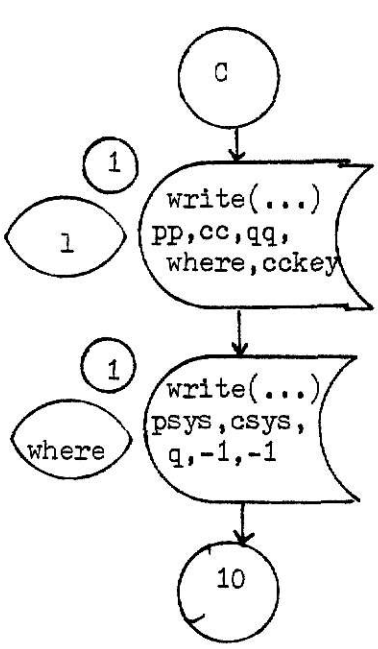
BOMfile

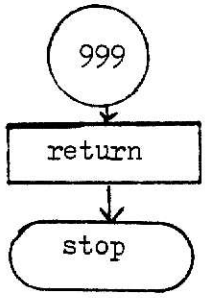
-Subroutine addbom-




D
A
i=1,
next-1,1
1
i
≤
read(...)
pp,cc,qq,
ppkey,cckey
>
where=next-1
cckey=-1
10
print,'parent,
child,qty to be
added or quit,q,Ø
to stop'
1
i
write(...)
pp,cc,qq,
ppkey,cckey
A
5
read(...)
p,c,q
p='quit'?
yes
next=where+1
no
Find the
system
numbers.
1
Ø
write(...)
next
call findsys
(p,psys)
call findsys
(c,csys)
call childkeys
Can't find one
of the numbers?
psys=-1
.or.
csys=-1?
yes
10
999
no
Next empty
record number.
where=where+1
close(unit=3)
open(unit=3...)
B

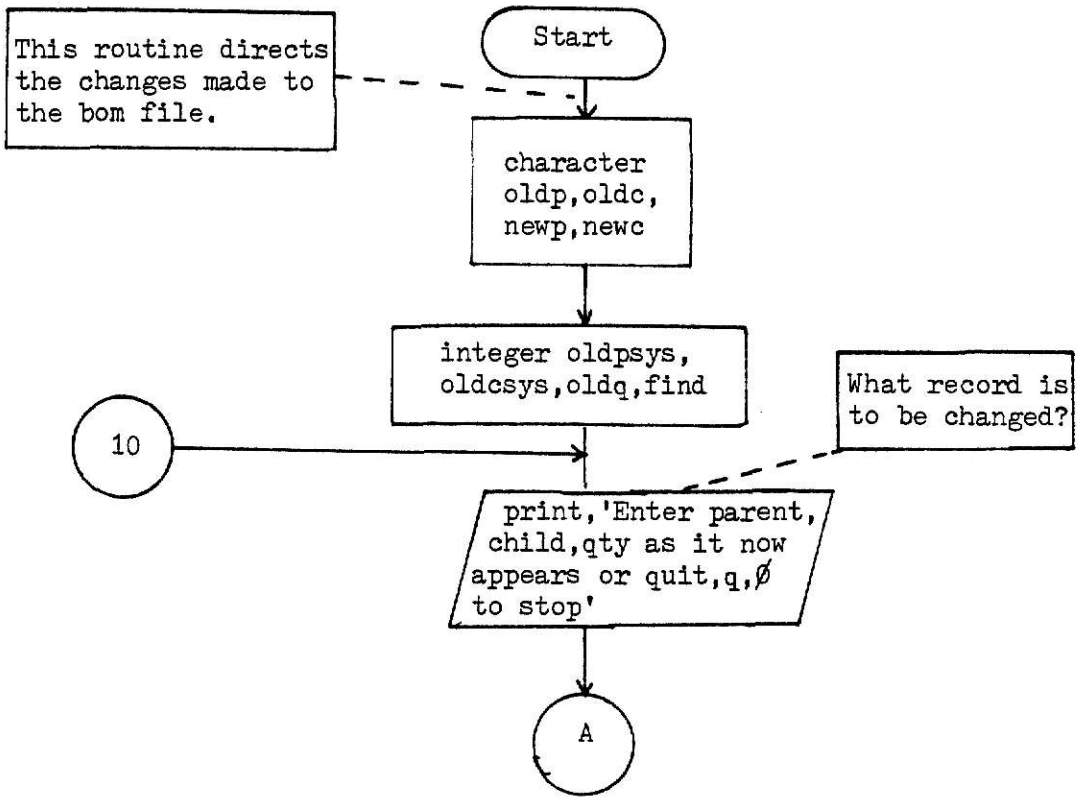

B
3
psys
read(...)
oh,...,
levelkey
Number of kids?
numkids
≠-1?
yes
numkids=
numkids+1
no
New parent?
numkids
=-1?
yes
numkids=1
no
Where to find
other children?
BofM=-1?
yes
BofM=where
no
3
psys
write(...
oh,...,
levelkey
3
psys
write(...
oh,...,
levelkey
l=BOM
1
where
write(...
psys,csys,
q,-1,-1
20
1
l
read(...)
pp,cc,qq,
ppkey,cckey
10
C
yes
ppkey=-1?
no
BOM=BOM+1
l=ppkey
20

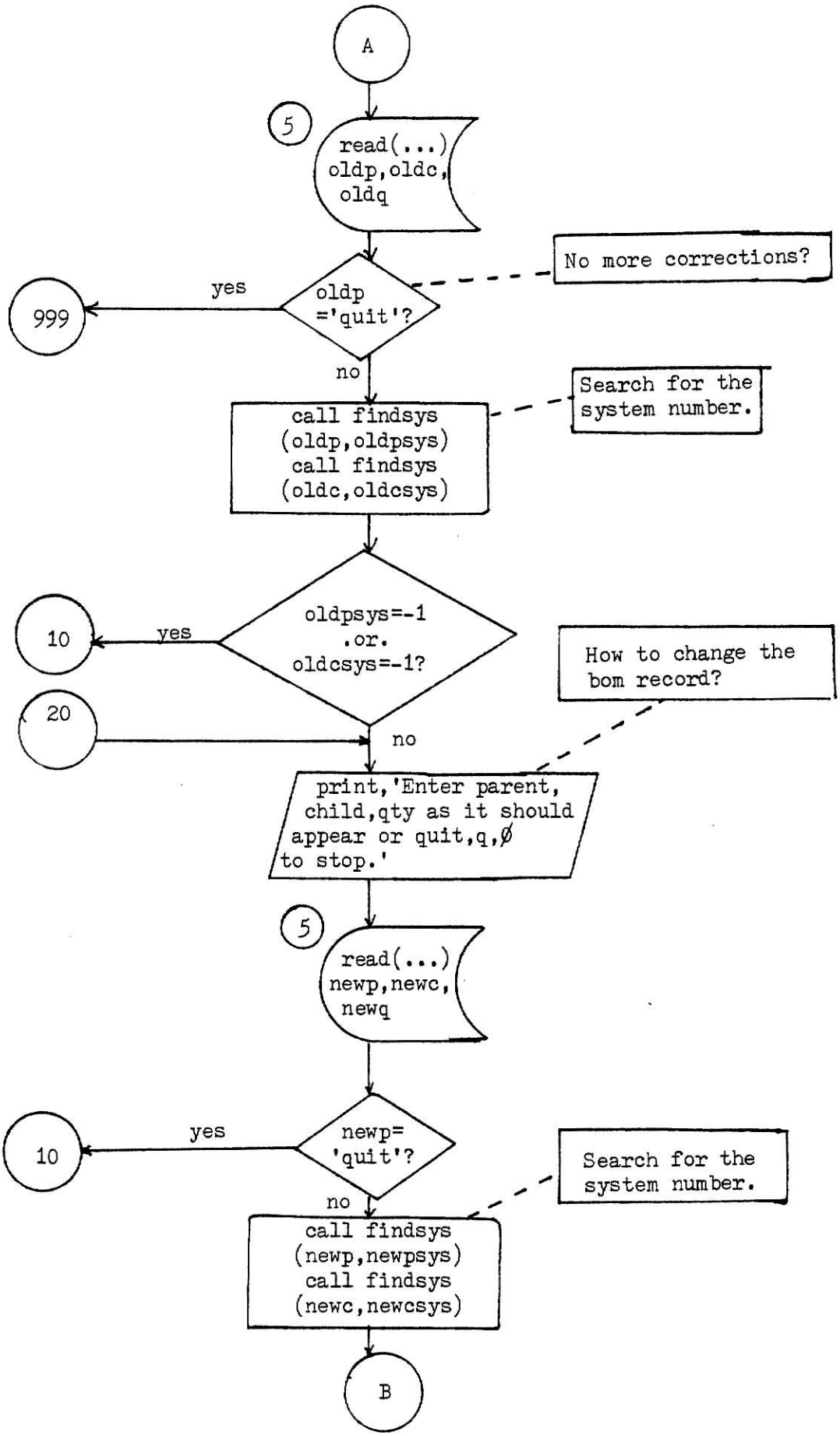


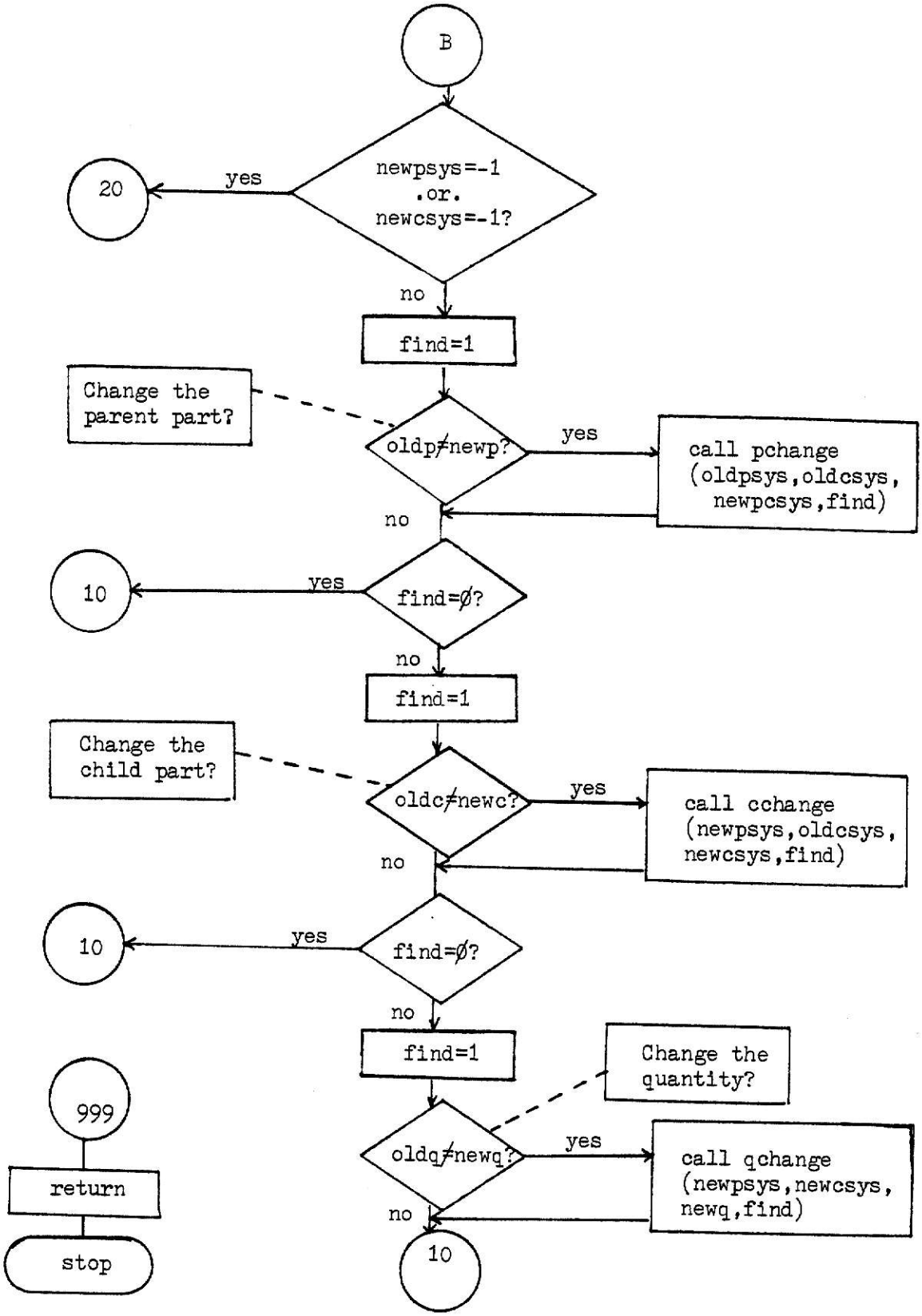


BOMfile

-Subroutine changebom-

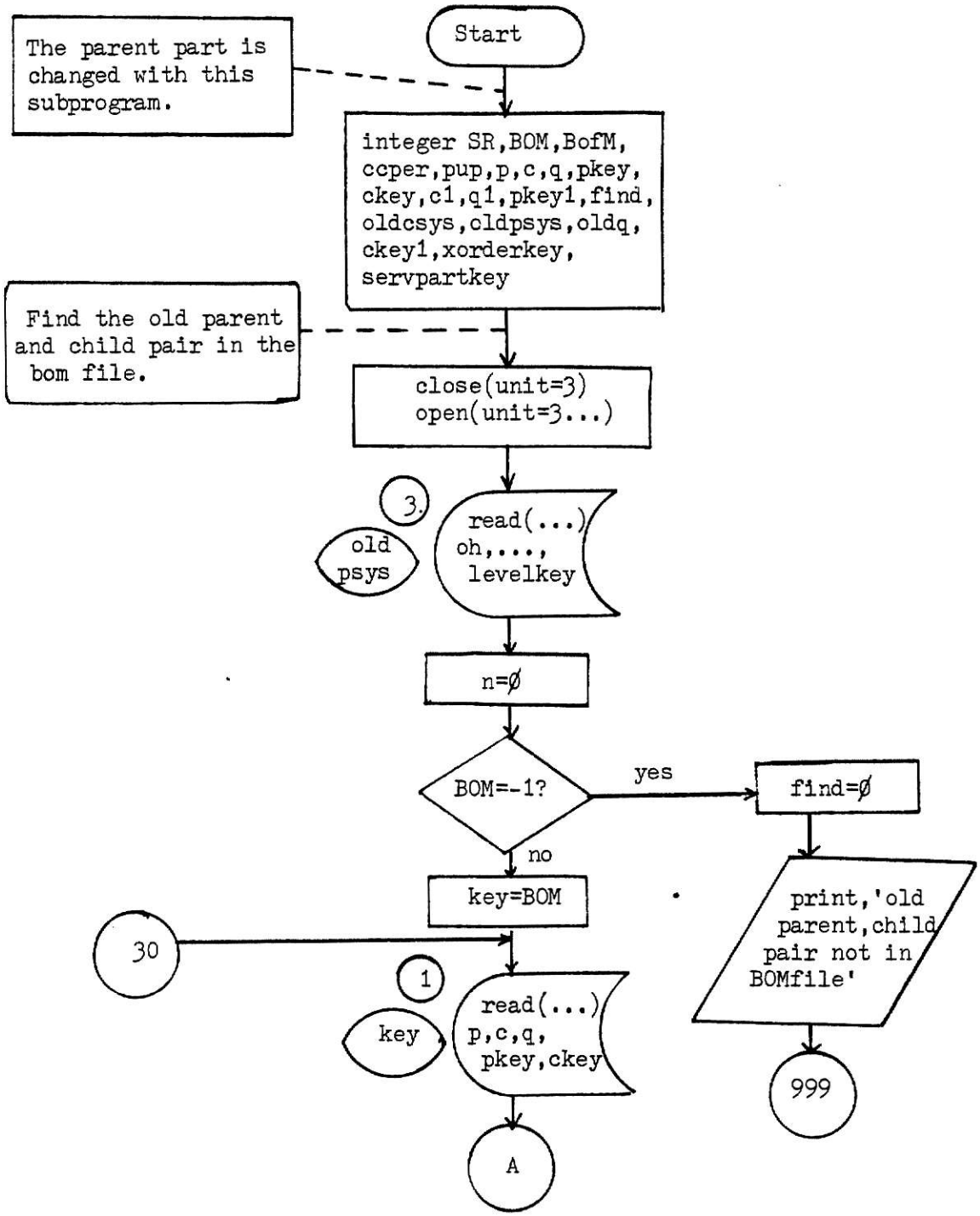


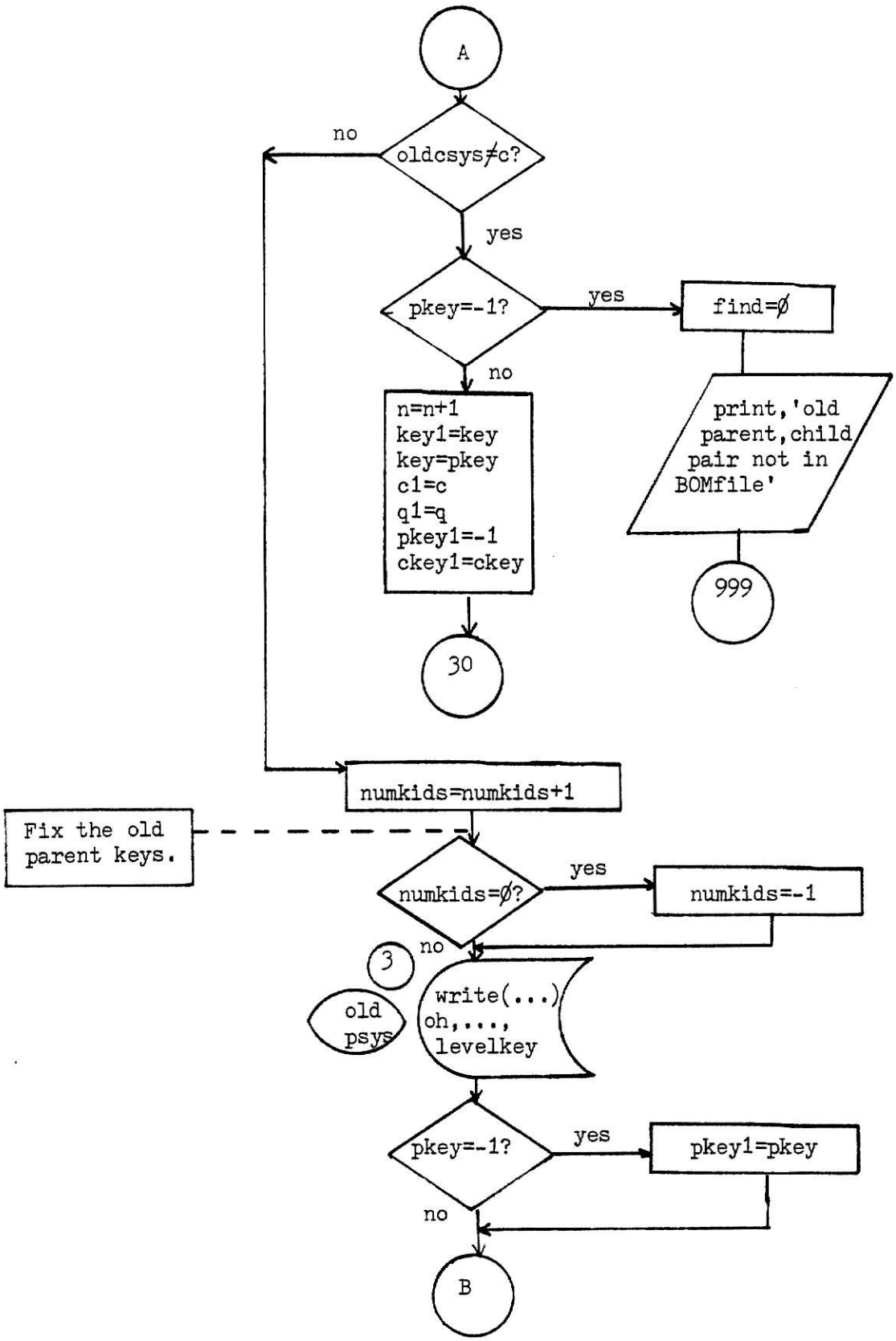

A
5
read(...)
oldp,oldc,
oldq
No more corrections?
yes
oldp
='quit'?
999
no
Search for the
system number.
call findsys
(oldp,oldpsys)
call findsys
(oldc,oldcsys)
oldpsys=-1
.or.
oldcsys=-1?
yes
10
How to change the
bom record?
20
no
print,'Enter parent,
child,qty as it should
appear or quit,q,Ø
to stop.'
5
read(...)
newp,newc,
newq
yes
newp=
'quit'?
10
Search for the
system number.
no
call findsys
(newp,newpsys)
call findsys
(newc,newcsys)
B

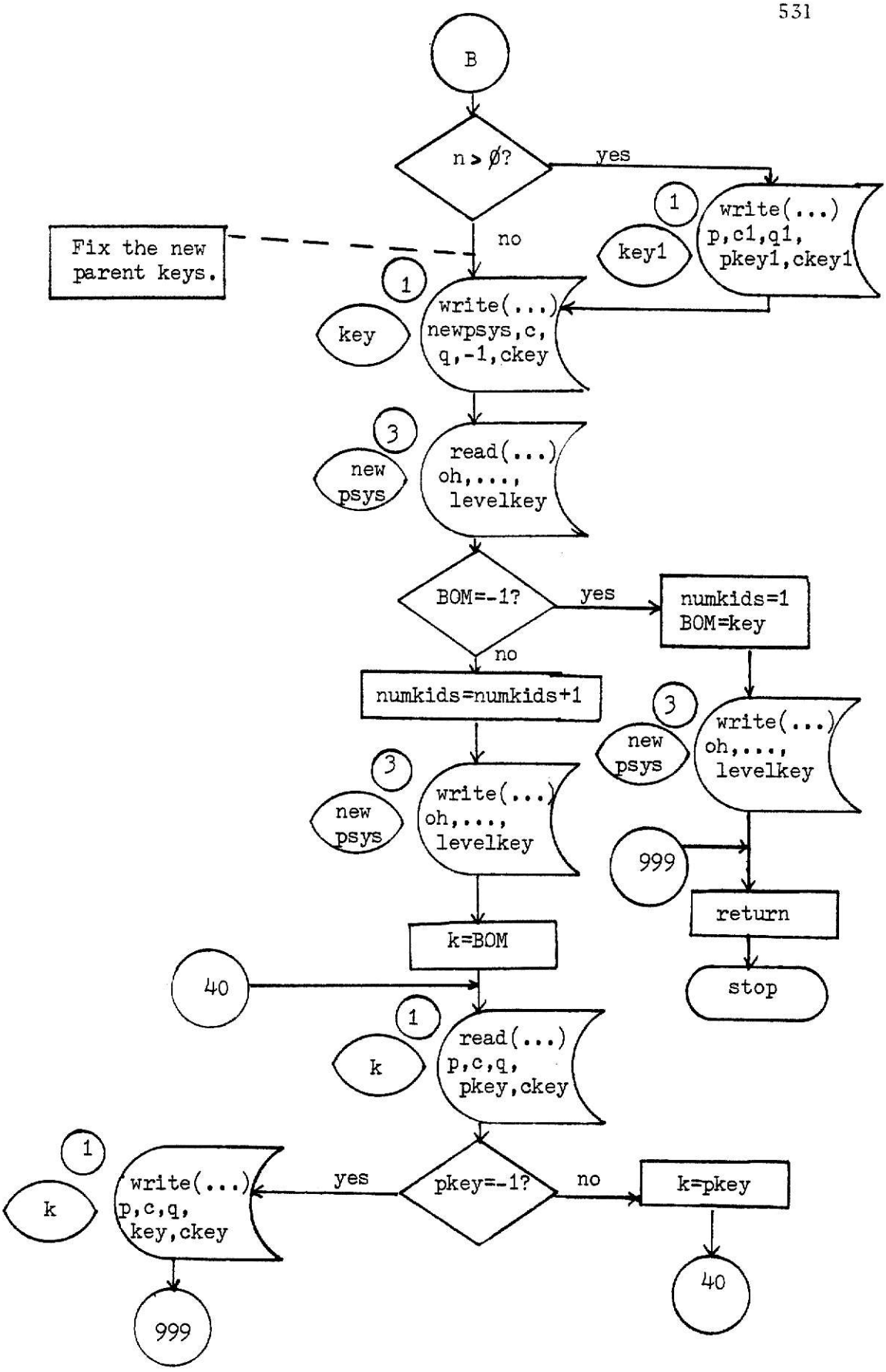

B
newpsys=-1
.or.
newcsys=-1?
yes
20
no
find=1
Change the
parent part?
oldp≠newp?
yes
call pchange
(oldpsys,oldcsys,
newpcsys,find)
no
10
yes
find=∅?
no
find=1
Change the
child part?
oldc≠newc?
yes
call cchange
(newpsys,oldcsys,
newcsys,find)
no
10
yes
find=∅?
no
find=1
Change the
quantity?
999
oldq≠newq?
yes
call qchange
(newpsys,newcsys,
newq,find)
return
no
10
stop

BOMfile

-Subroutine pchange-
Arguments: oldpsys,oldcsys
newpsys,find

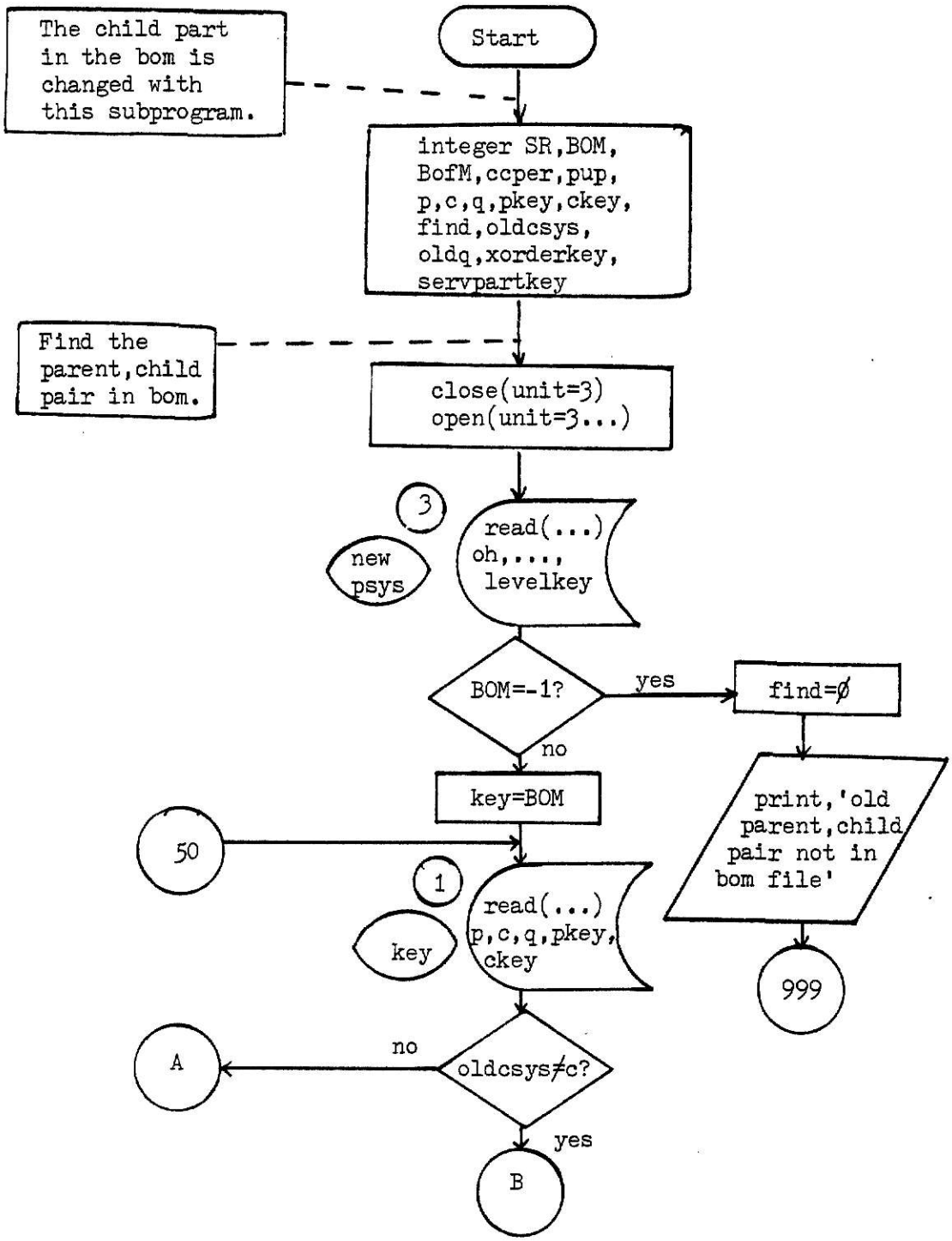


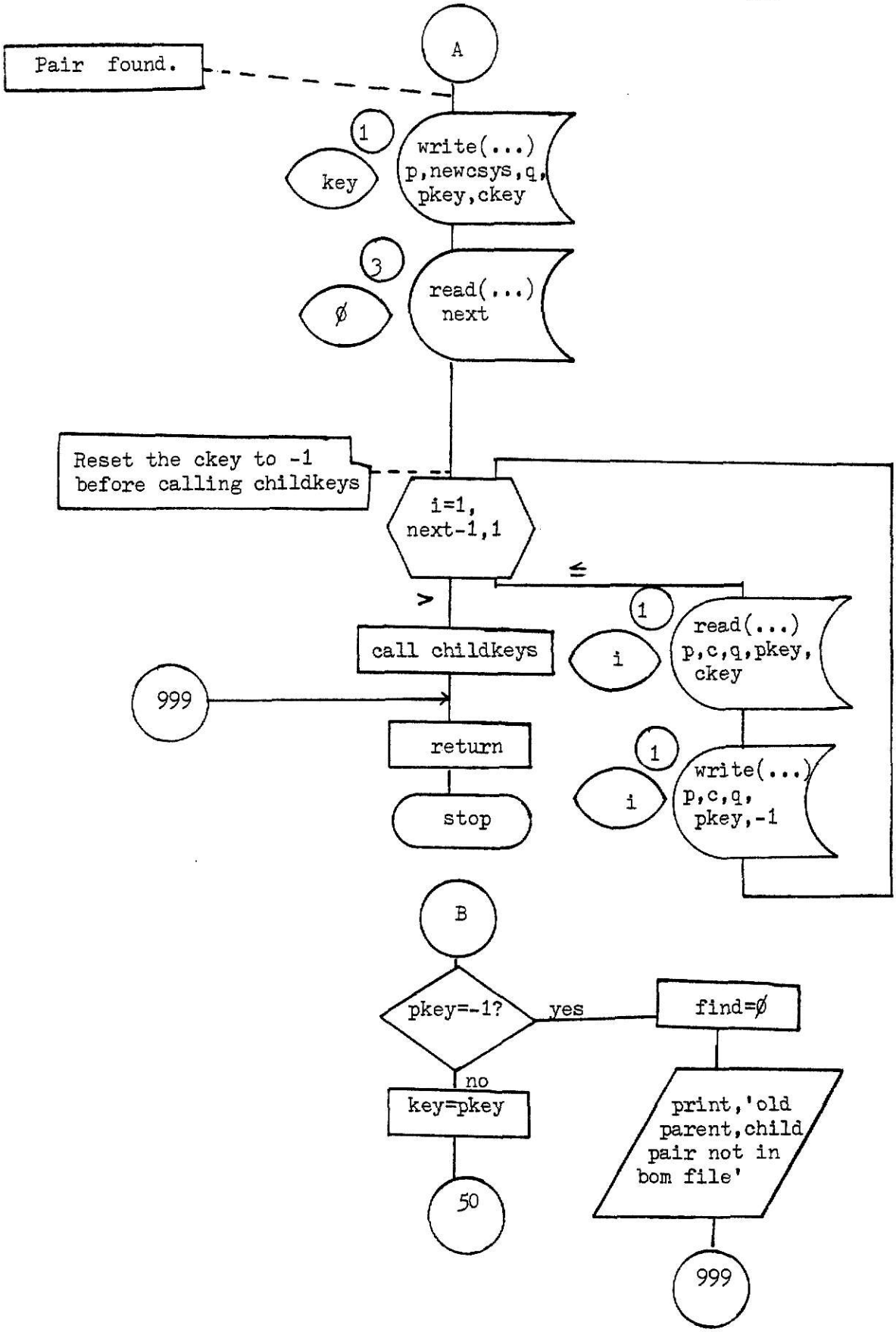

A
oldcsys≠c?
no
yes
pkey=-1?
yes
find=∅
no
n=n+1
key1=key
key=pkey
c1=c
q1=q
pkey1=-1
ckey1=ckey
print,'old parent,child pair not in BOMfile'
999
30
numkids=numkids+1
Fix the old parent keys.
numkids=∅?
yes
numkids=-1
3
no
old psys
write(...)
oh,...,
levelkey
pkey=-1?
yes
pkey1=pkey
no
B


B
n > Ø?
yes
no
1
write(...)
p,c1,q1,
pkey1,ckey1
key1
Fix the new
parent keys.
1
write(...)
newpsys,c,
q,-1,ckey
key
3
read(...)
oh,...,
levelkey
new
psys
BOM=-1?
yes
numkids=1
BOM=key
no
numkids=numkids+1
3
write(...)
oh,...,
levelkey
new
psys
3
write(...)
oh,...,
levelkey
new
psys
999
return
stop
k=BOM
40
1
read(...)
p,c,q,
pkey,ckey
k
1
write(...)
p,c,q,
key,ckey
k
yes
pkey=-1?
no
k=pkey
40
999

BOMfile

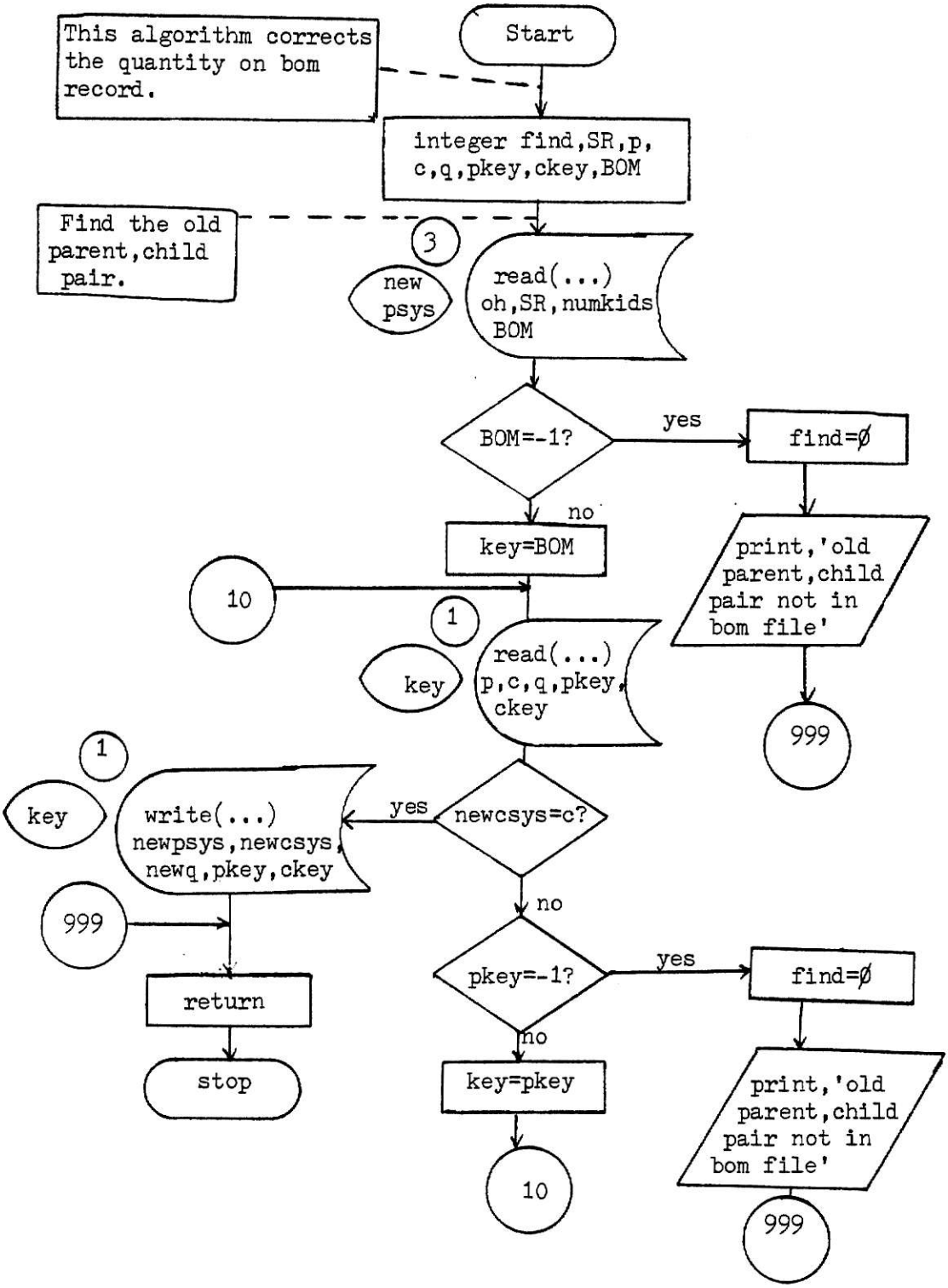
-Subroutine cchange-
Agruments: newpsys,oldcsys,
newcsys,find




Pair found.
A
1
key
write(...)
p,newcsys,q,
pkey,ckey
3
ø
read(...)
next
Reset the ckey to -1
before calling childkeys
i=1,
next-1,1
≤
>
call childkeys
999
return
stop
1
i
read(...)
p,c,q,pkey,
ckey
1
i
write(...)
p,c,q,
pkey,-1
B
pkey=-1?
yes
find=ø
no
key=pkey
50
print,'old
parent,child
pair not in
bom file'
999

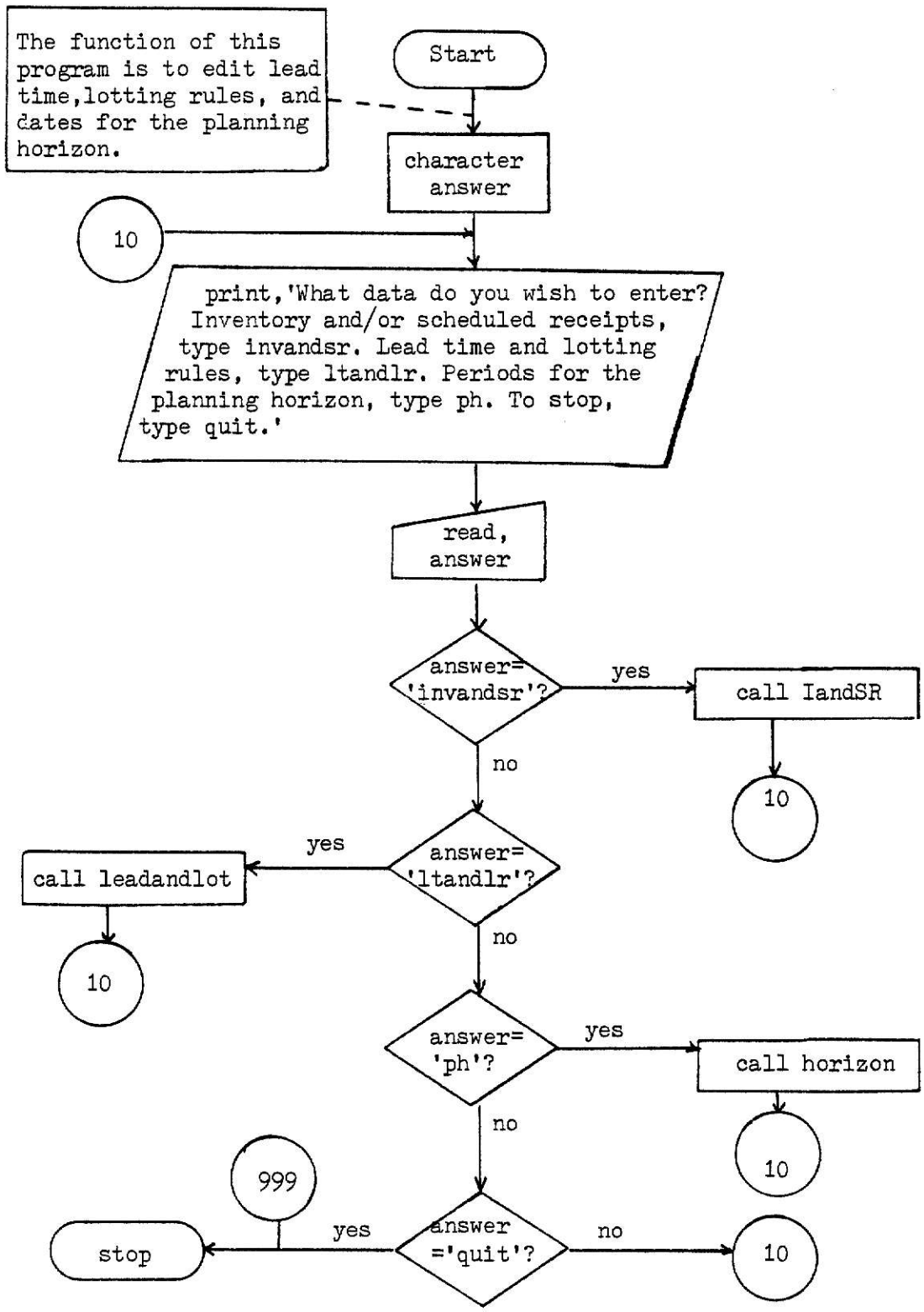
BOMfile

-Subroutine qchange-
Arguments: newpsys,newcsys,
newq,find



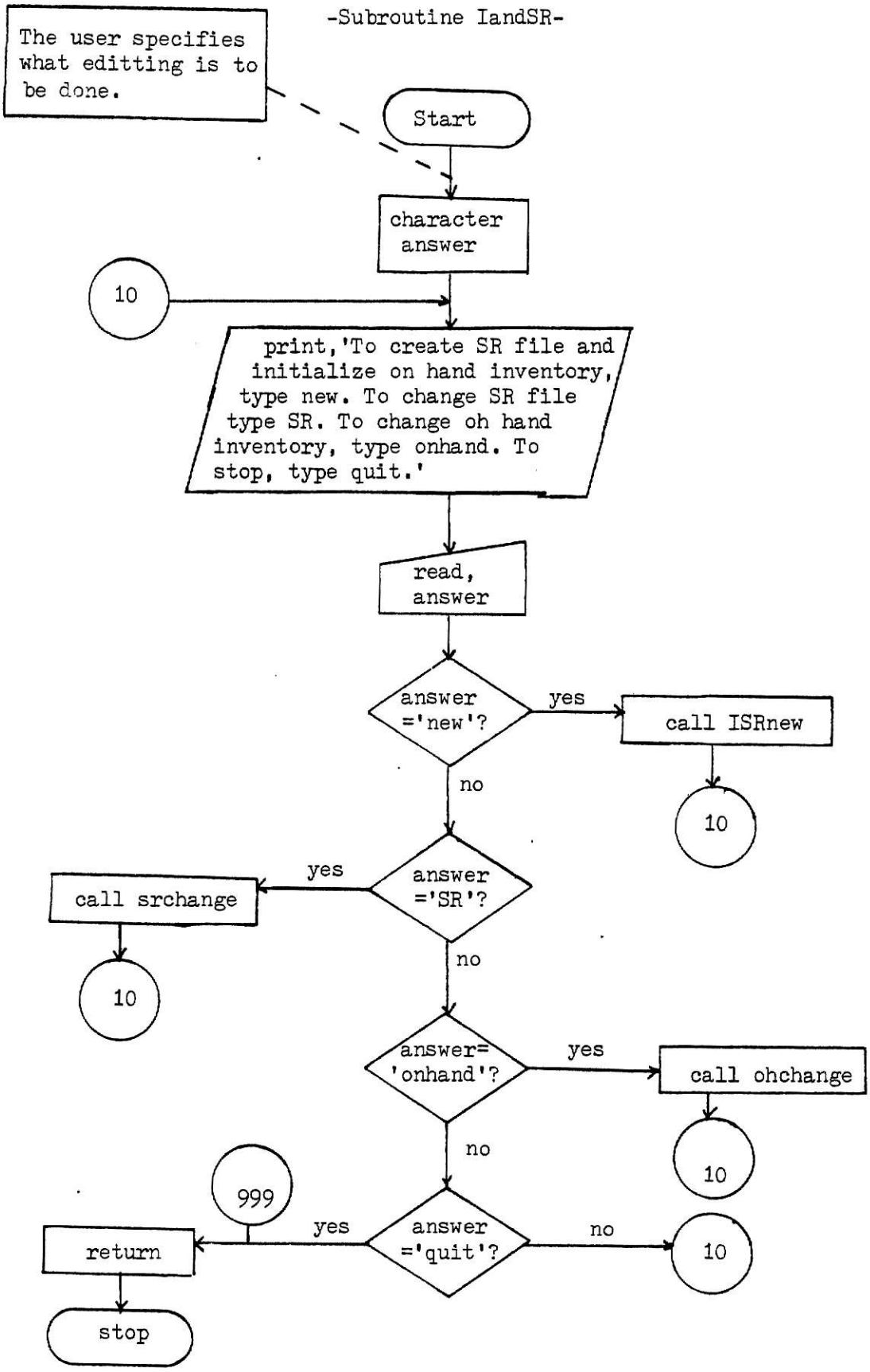
DataPlus

-Main Program



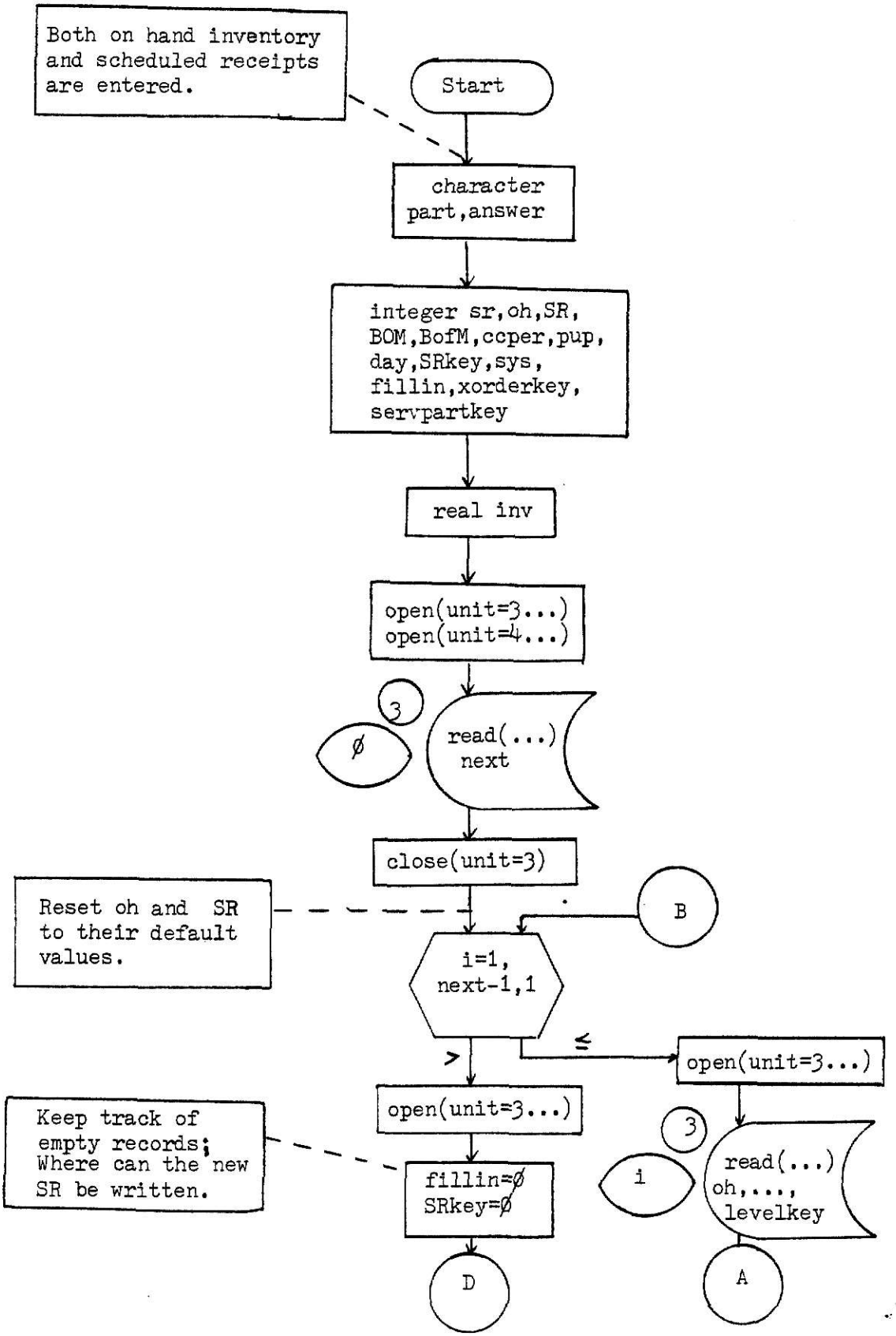
DataPlus

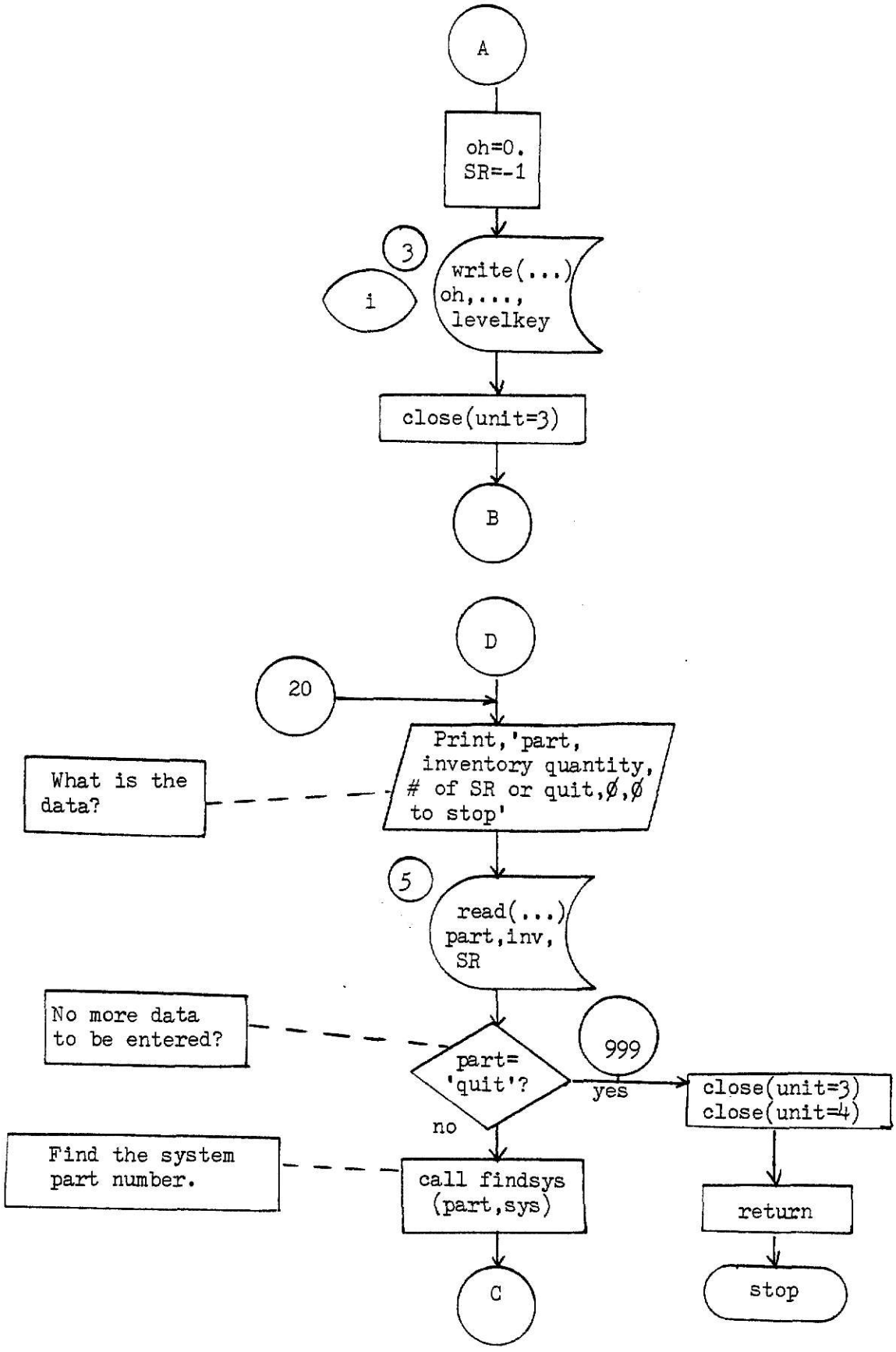
-Subroutine IandSR-

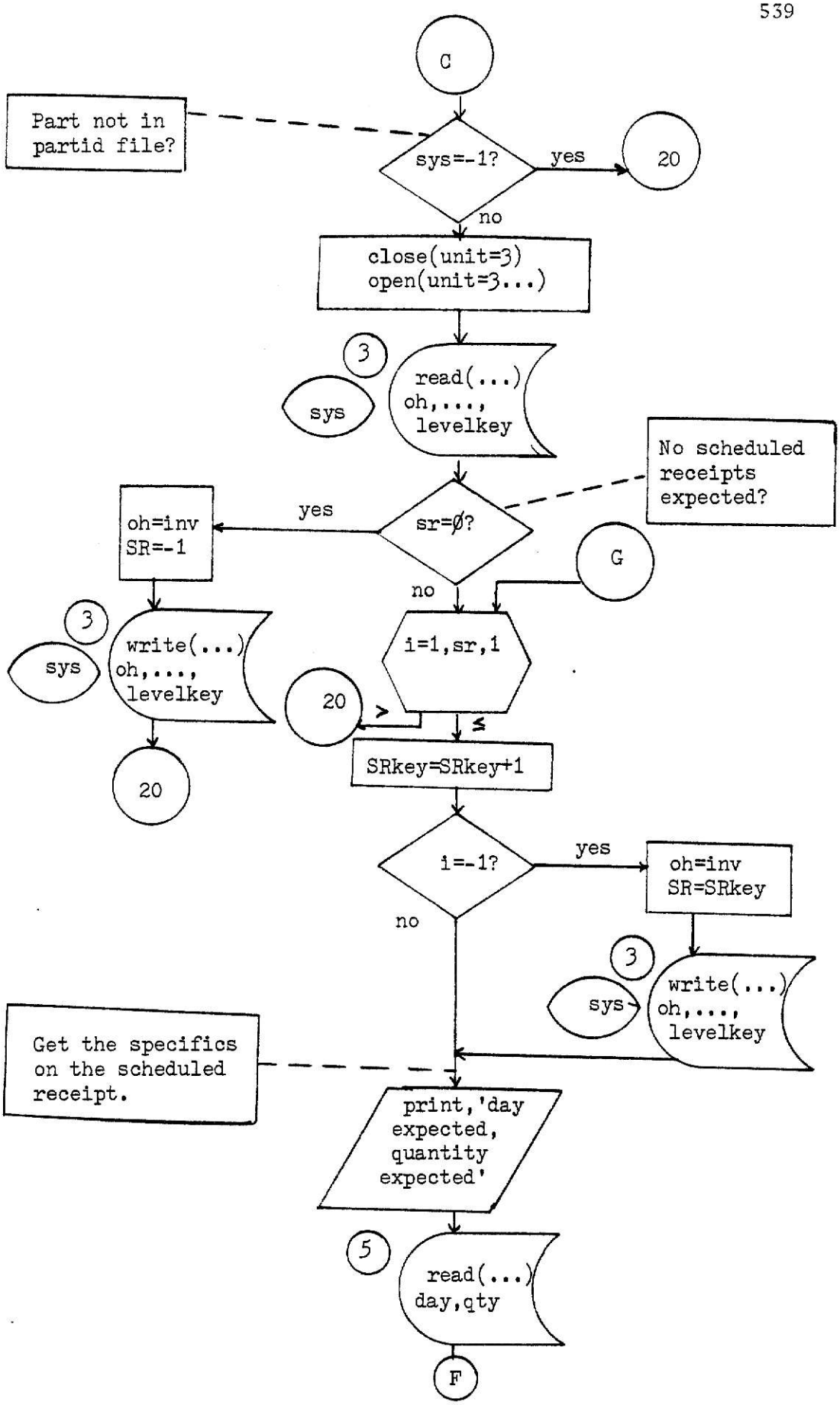


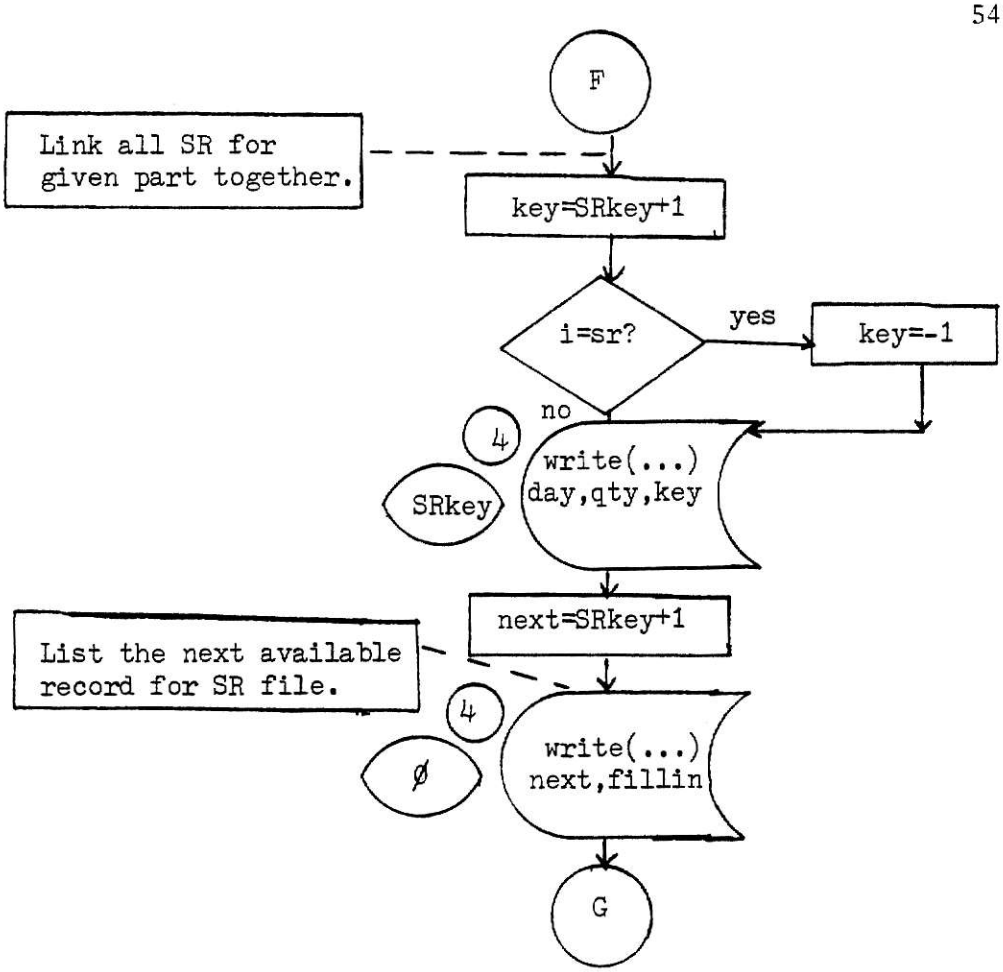
DataPlus

-Subroutine ISRnew-



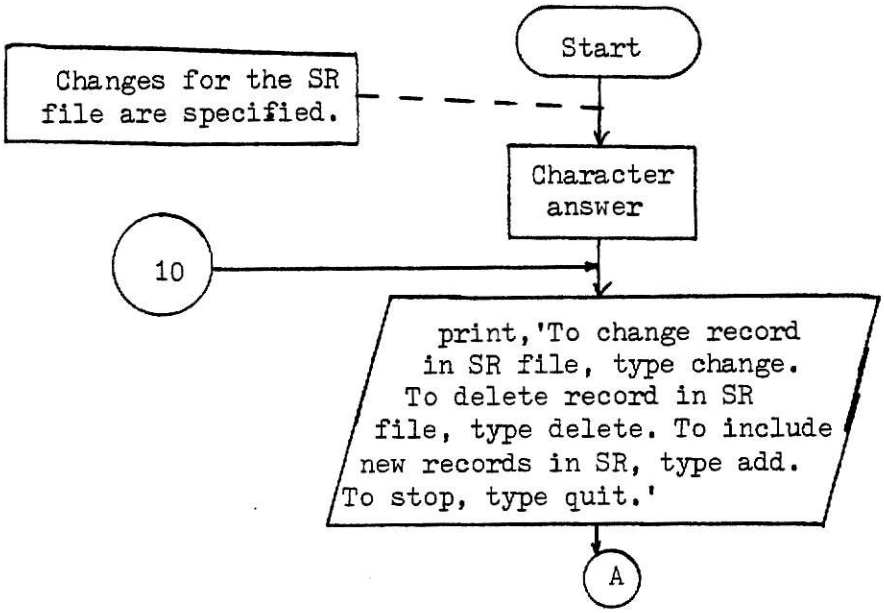

A
oh=0.
SR=-1
3
i
write(...)
oh,...,
levelkey
close(unit=3)
B
D
20
Print,'part,
inventory quantity,
# of SR or quit,Ø,Ø
to stop'
What is the
data?
5
read(...)
part,inv,
SR
No more data
to be entered?
part=
'quit'?
999
yes
no
close(unit=3)
close(unit=4)
Find the system
part number.
call findsys
(part,sys)
return
C
stop

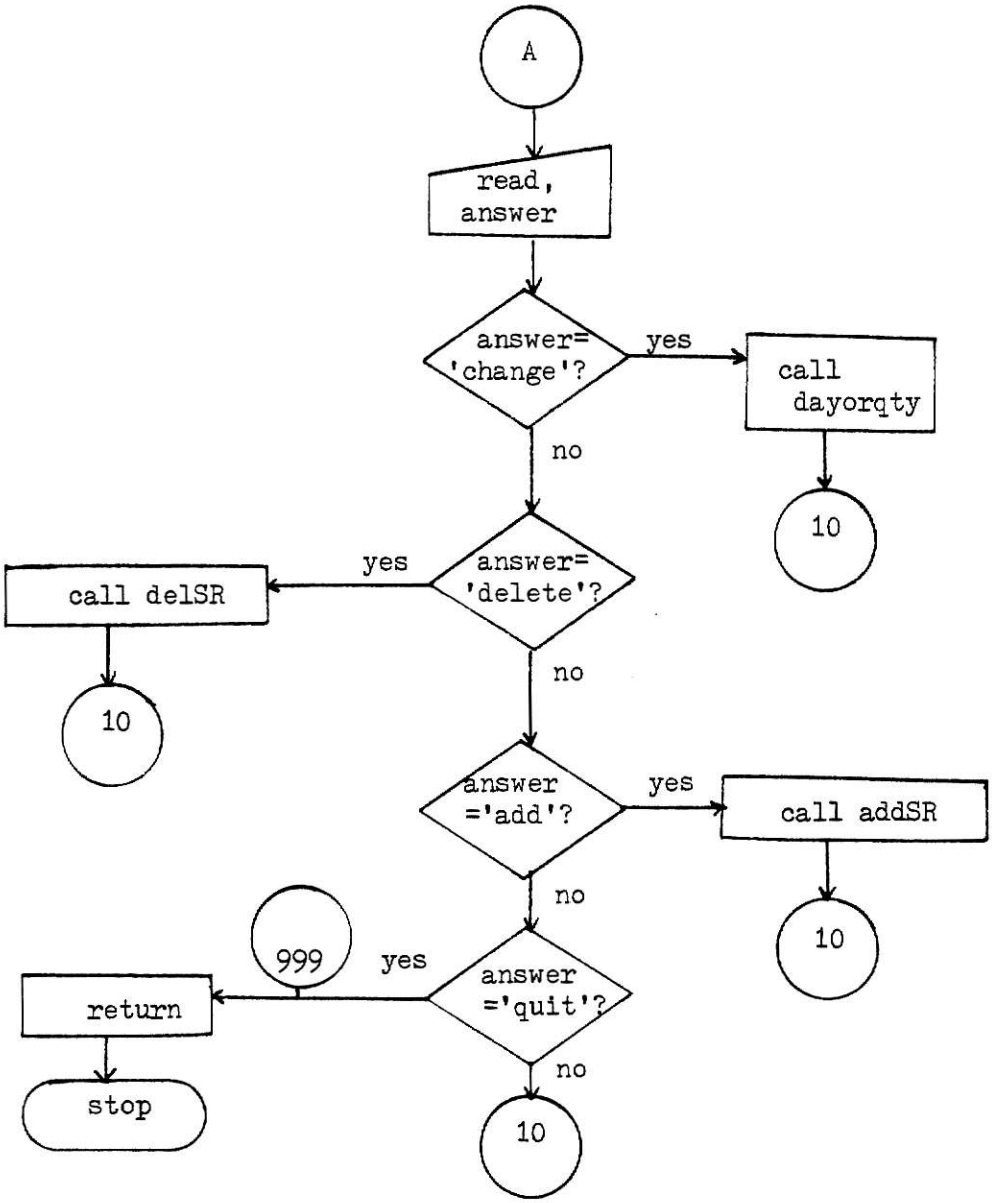

C
Part not in partid file?
sys=-1?
yes
20
no
close(unit=3)
open(unit=3...)
3
sys
read(...)
oh,...,
levelkey
No scheduled receipts expected?
sr=Ø?
yes
oh=inv
SR=-1
G
no
3
sys
write(...)
oh,...,
levelkey
i=1,sr,1
20
>
≤
20
SRkey=SRkey+1
i=-1?
yes
oh=inv
SR=SRkey
no
3
sys
write(...)
oh,...,
levelkey
Get the specifics on the scheduled receipt.
print,'day expected, quantity expected'
5
read(...)
day,qty
F



DataPlus

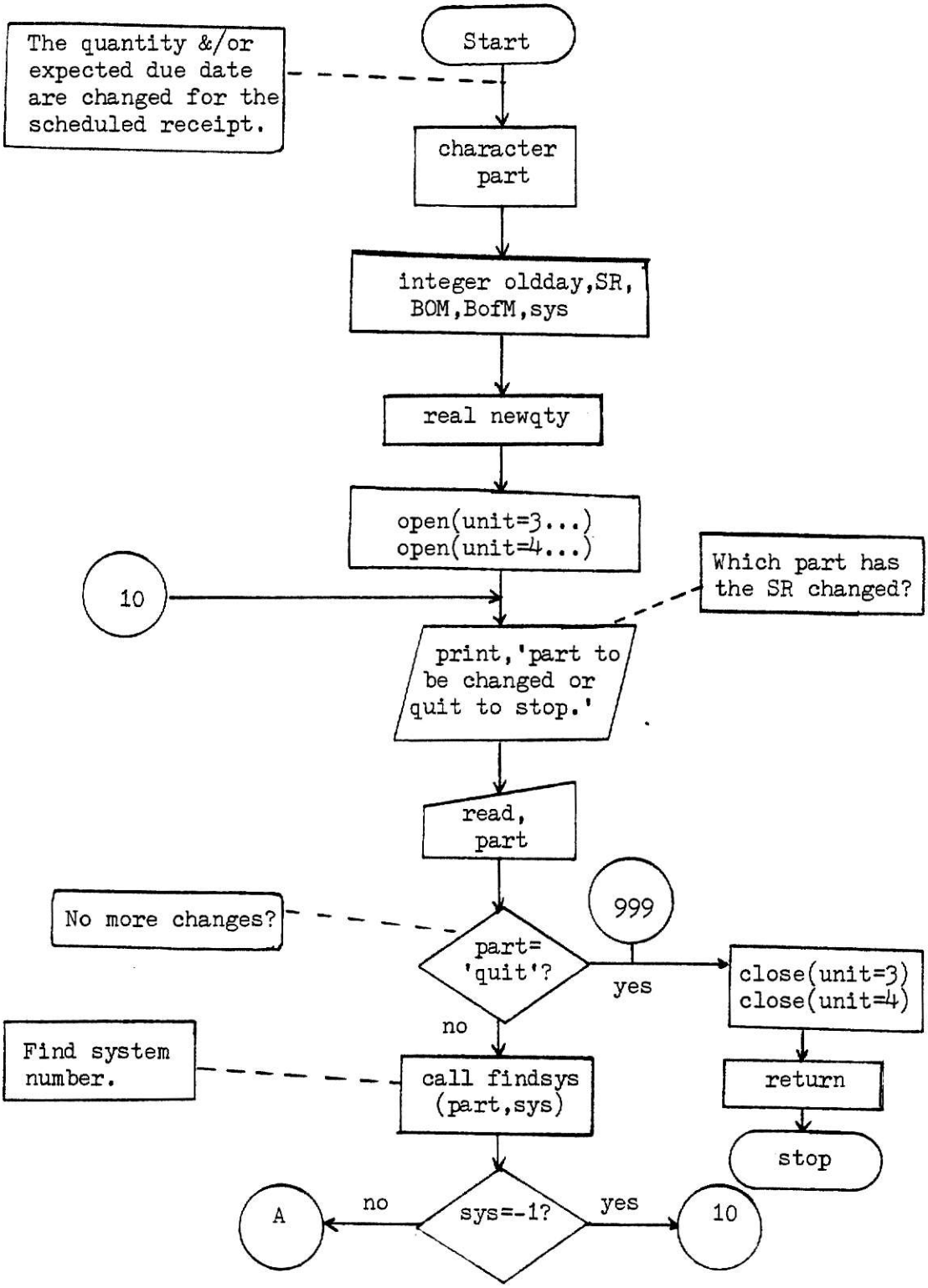
-Subroutine srchange-

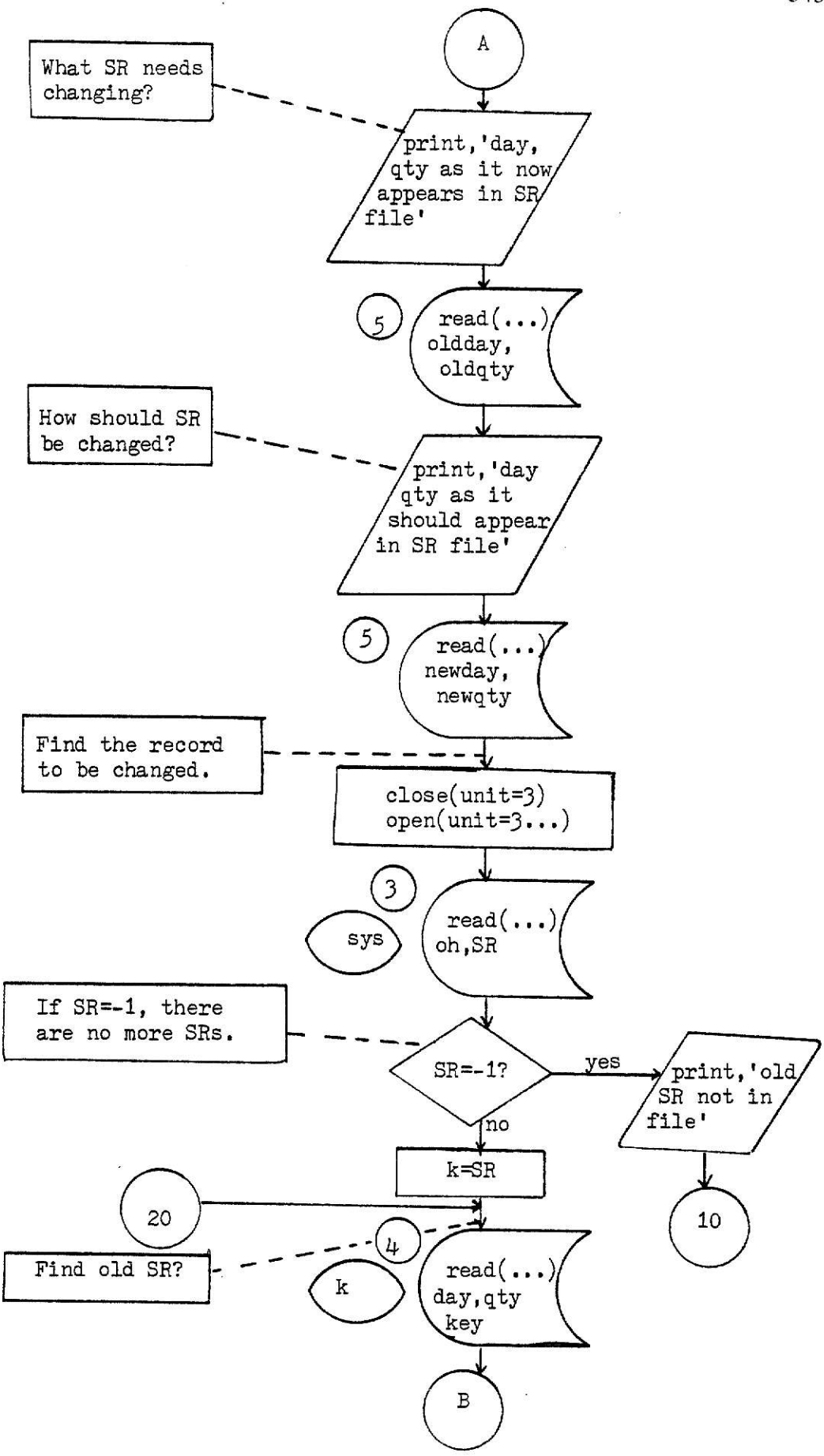


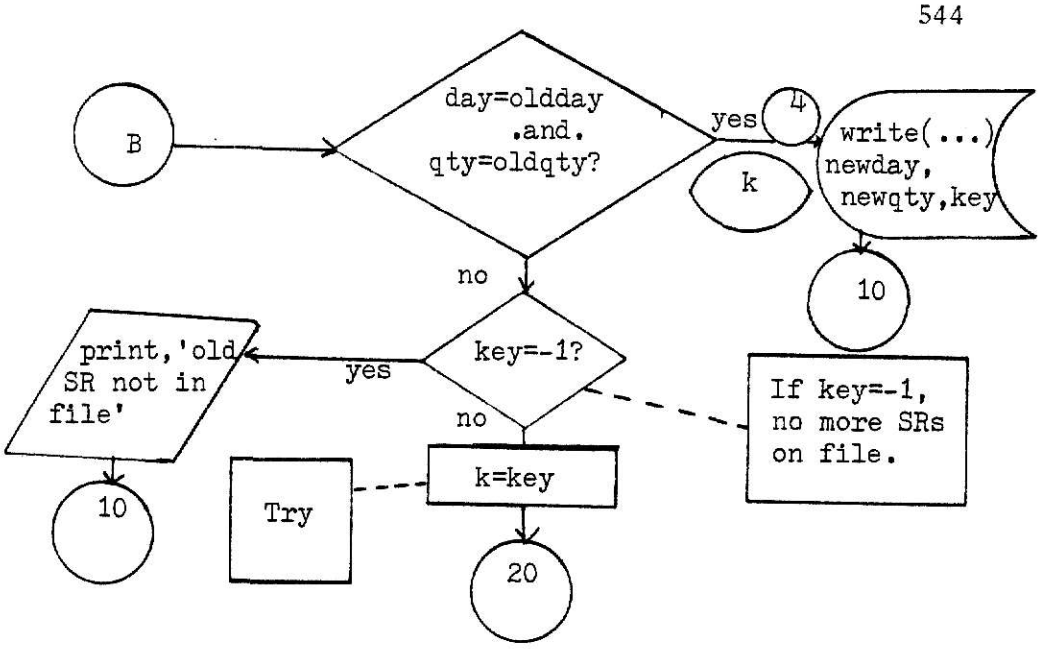

A
read,
answer
answer=
'change'?
yes
call
dayorqty
10
no
answer=
'delete'?
yes
call delSR
10
no
answer
='add'?
yes
call addSR
10
no
999
yes
answer
='quit'?
return
stop
no
10

DataPlus

-Subroutine dayorqty-

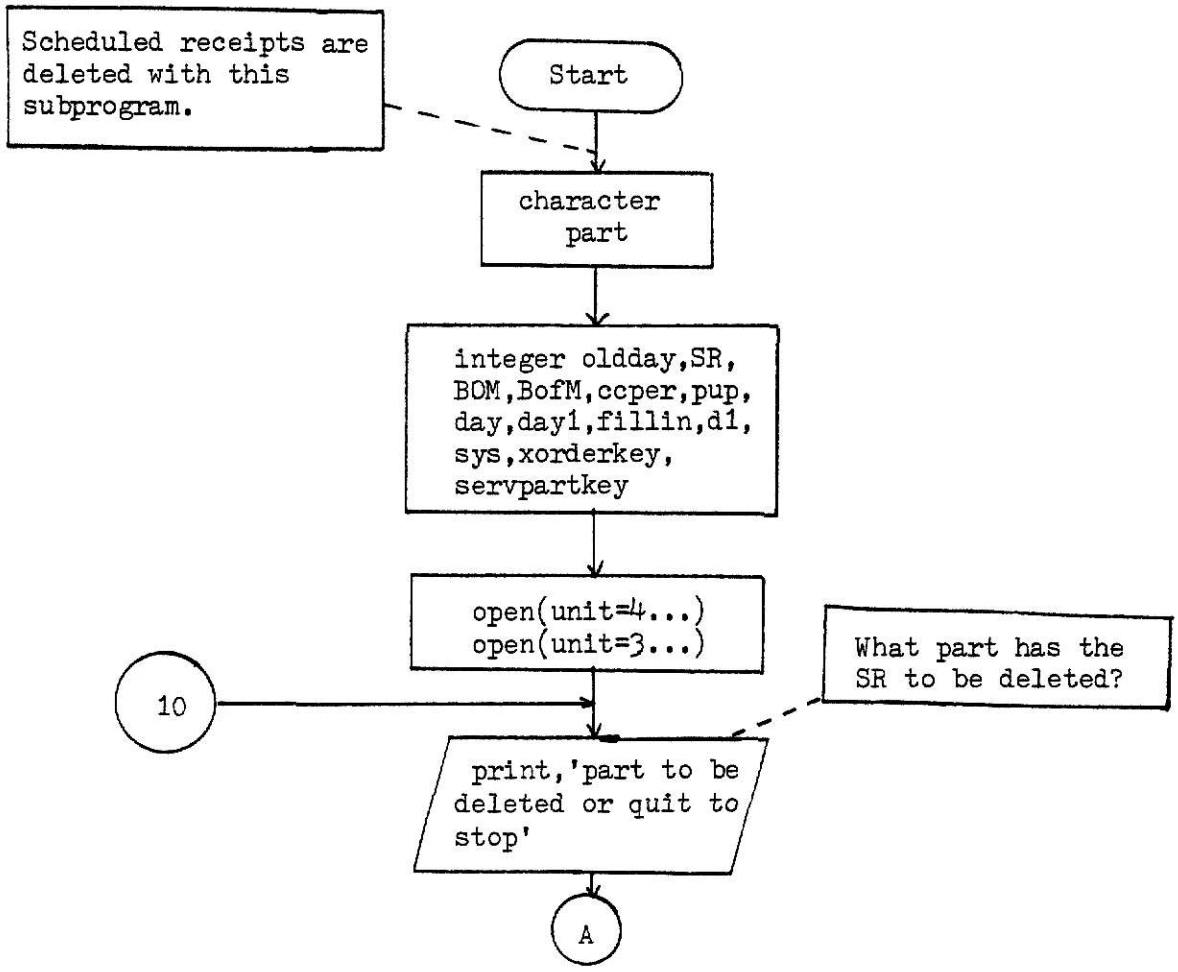


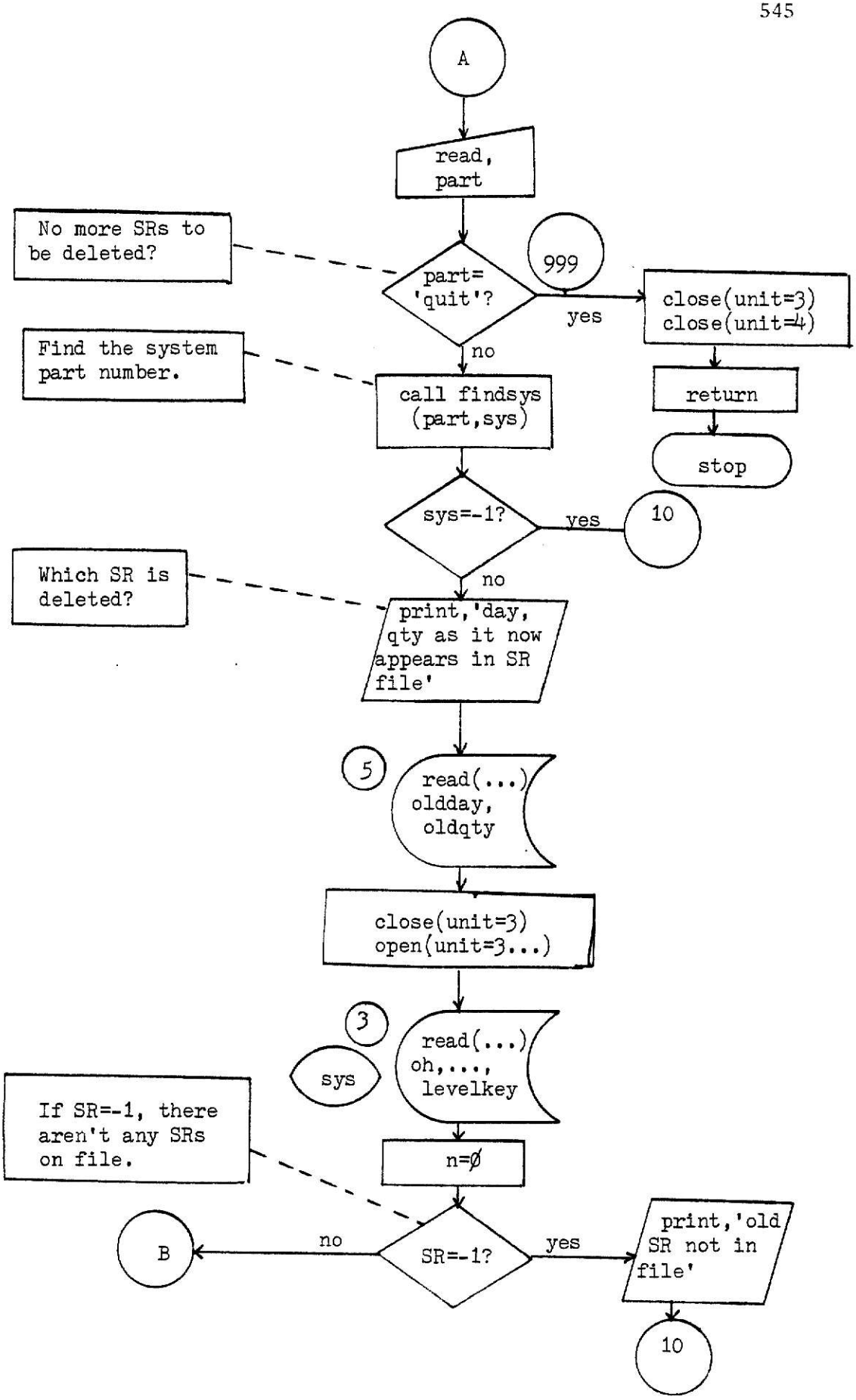

A
What SR needs changing?
print,'day, qty as it now appears in SR file'
5
read(...) oldday, oldqty
How should SR be changed?
print,'day qty as it should appear in SR file'
5
read(...) newday, newqty
Find the record to be changed.
close(unit=3) open(unit=3...)
3
sys
read(...) oh,SR
If SR=-1, there are no more SRs.
SR=-1?
yes
print,'old SR not in file'
no
k=SR
10
20
Find old SR?
4
k
read(...) day,qty key
B

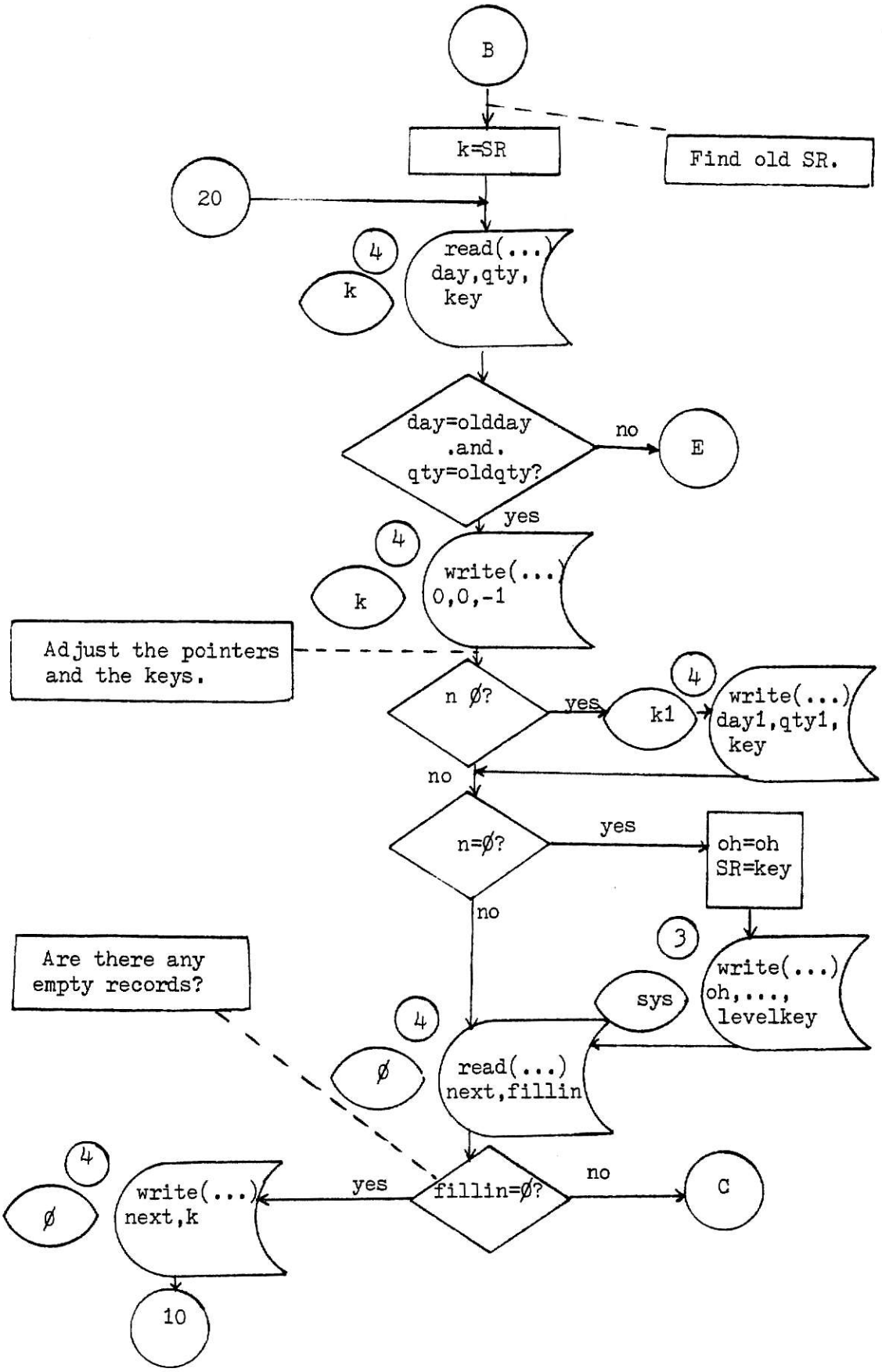


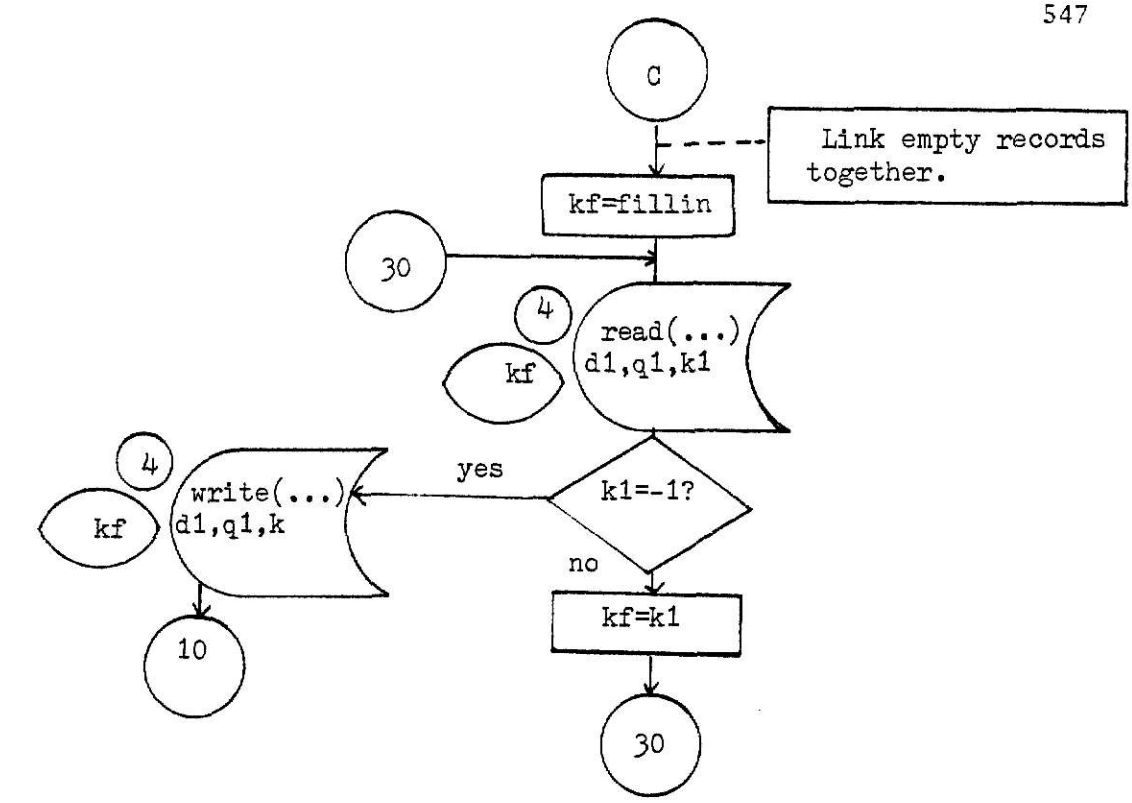
DataPlus

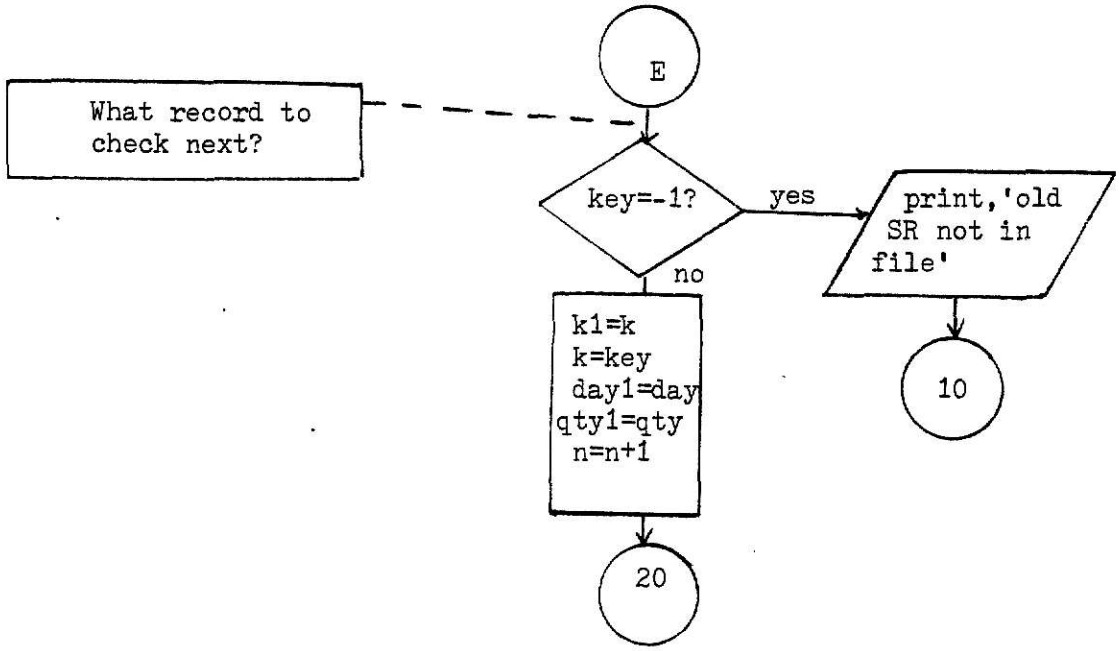
-Subroutine delSR-




A
read, part
No more SRs to be deleted?
part= 'quit'?
999
yes
close(unit=3)
close(unit=4)
return
stop
no
Find the system part number.
call findsys (part,sys)
sys=-1?
yes
10
no
Which SR is deleted?
print,'day, qty as it now appears in SR file'
5
read(...) oldday, oldqty
close(unit=3)
open(unit=3...)
3
sys
read(...) oh,..., levelkey
If SR=-1, there aren't any SRs on file.
n=ø
B
no
SR=-1?
yes
print,'old SR not in file'
10

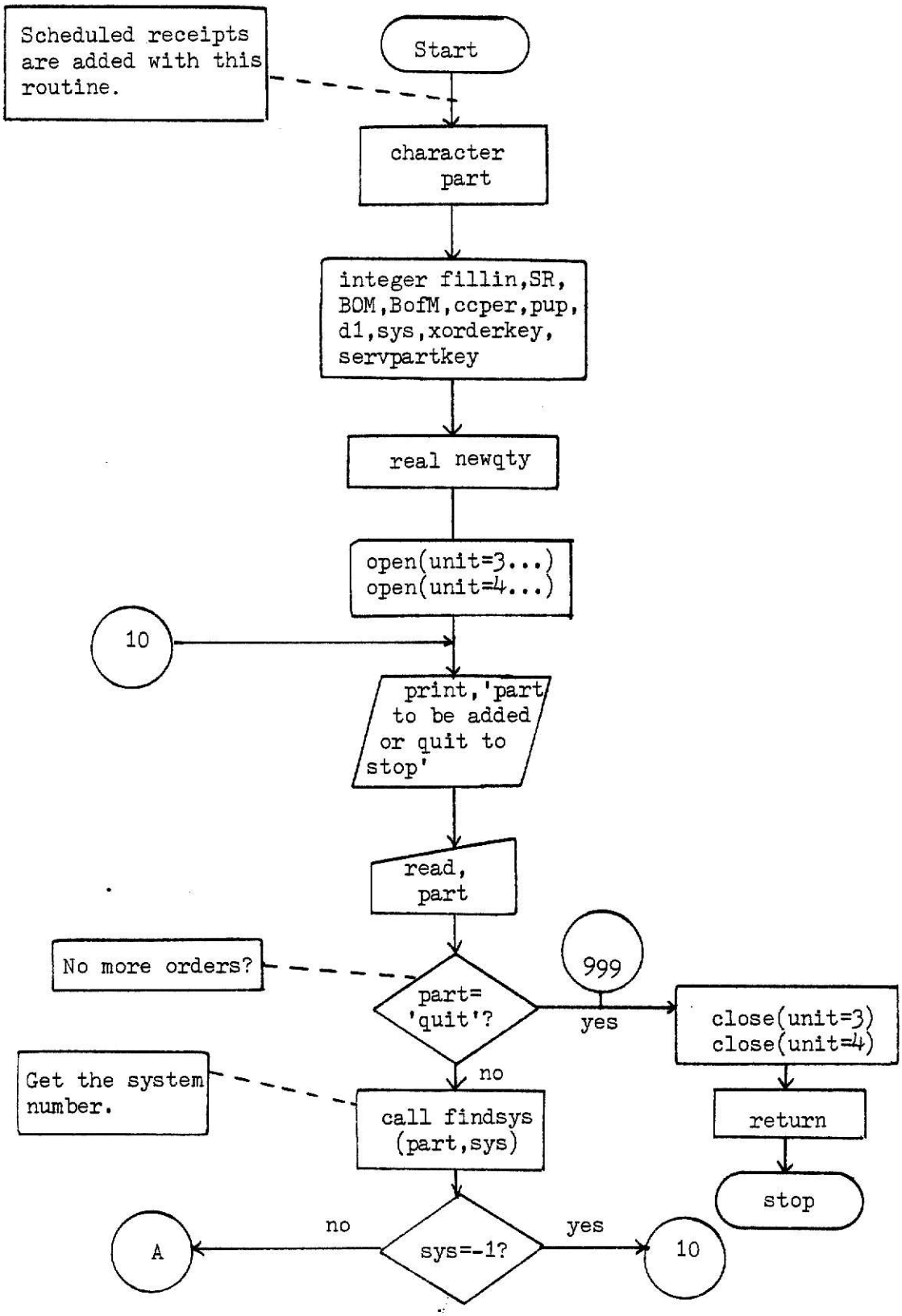

B
k=SR
Find old SR.
20
4
k
read(...)
day,qty,
key
day=oldday
.and.
qty=oldqty?
no
E
yes
4
k
write(...)
0,0,-1
Adjust the pointers
and the keys.
n ∅?
yes
4
k1
write(...)
day1,qty1,
key
no
n=∅?
yes
oh=oh
SR=key
no
3
write(...)
oh,...,
levelkey
Are there any
empty records?
sys
4
∅
read(...)
next,fillin
4
∅
write(...)
next,k
yes
fillin=∅?
no
C
10

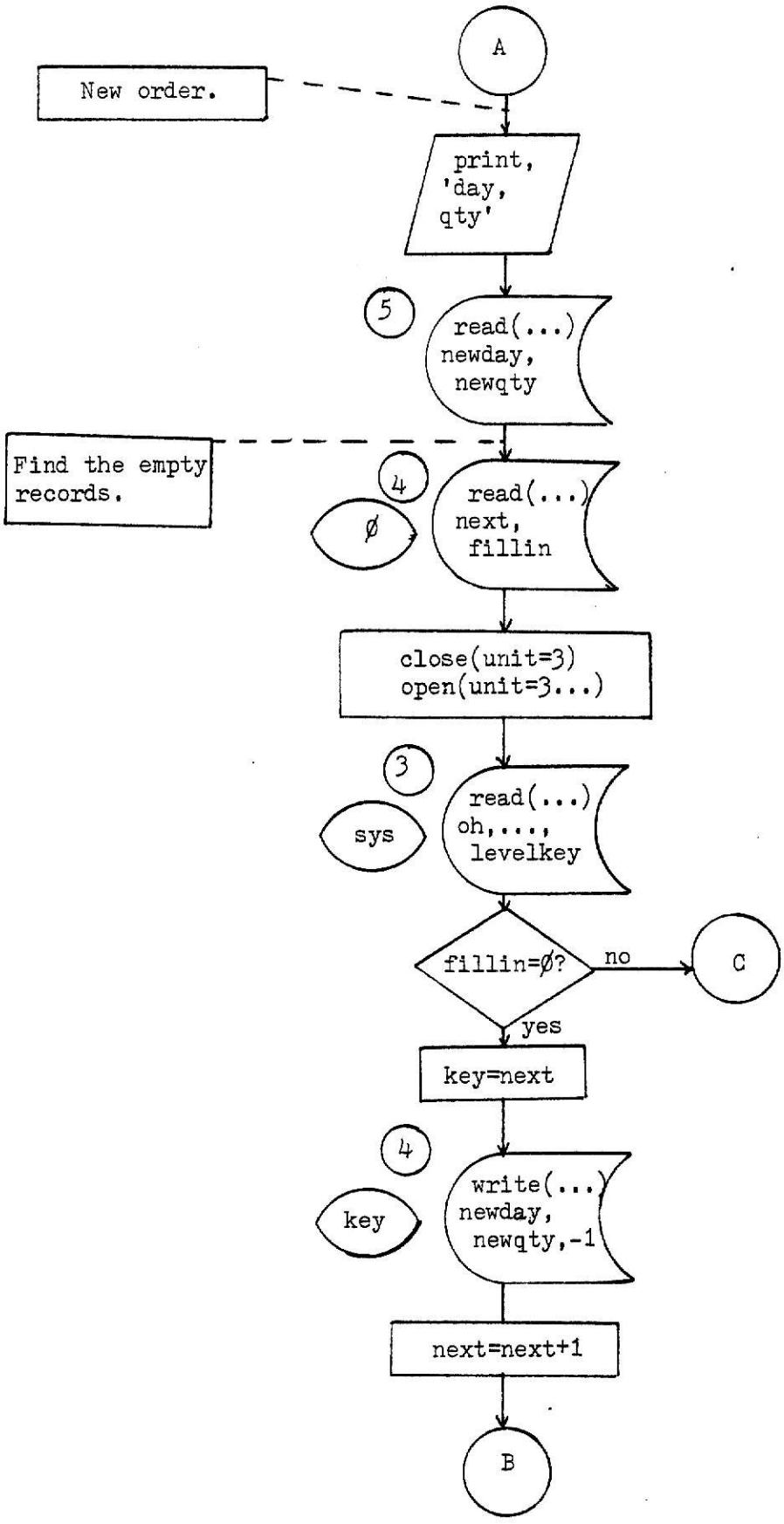

C
Link empty records together.
kf=fillin
30
4
kf
read(...)
d1,q1,k1
4
kf
write(...)
d1,q1,k
yes
k1=-1?
no
kf=k1
10
30

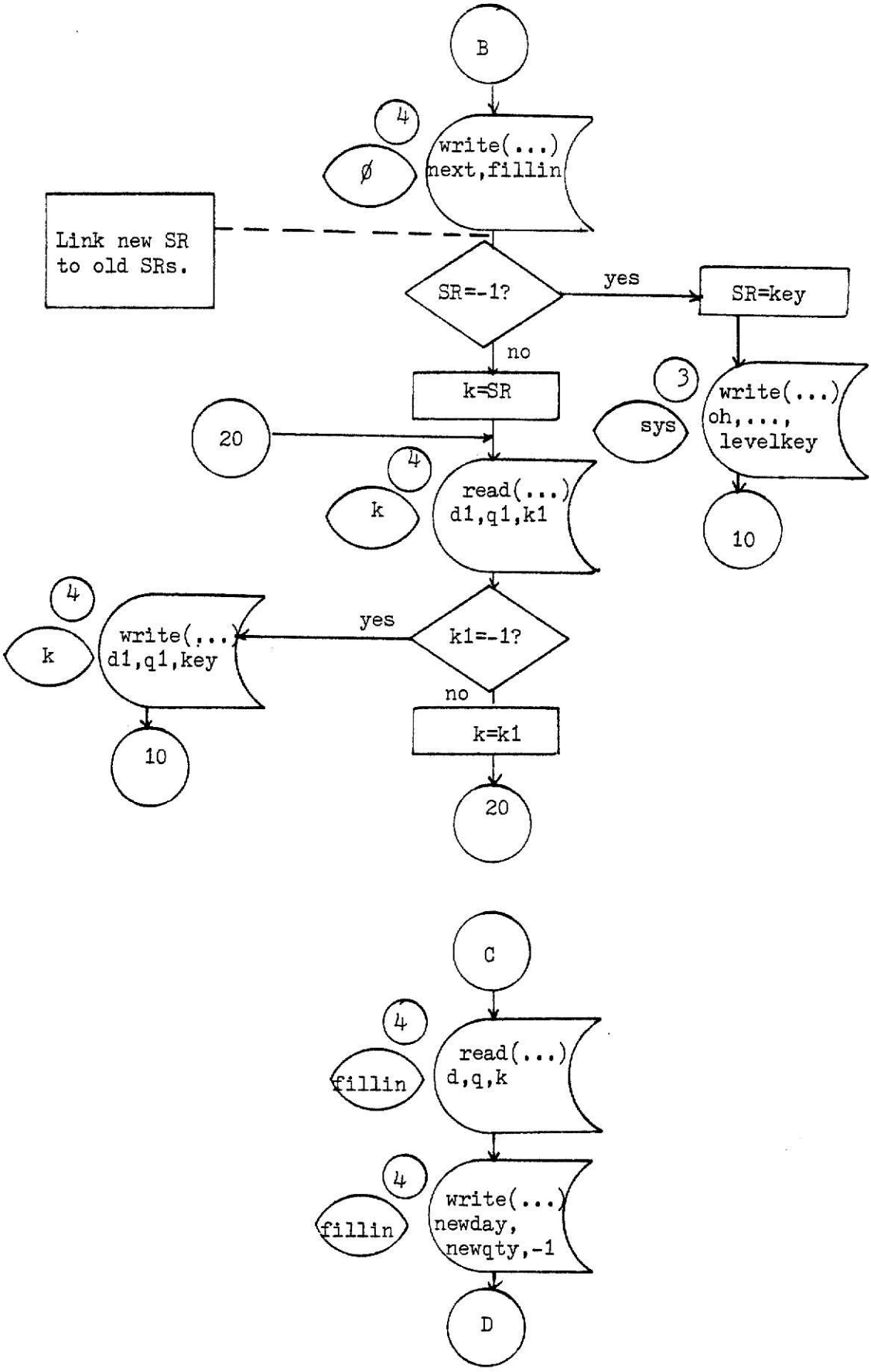

E
What record to check next?
key=-1?
yes
print,'old SR not in file'
no
k1=k
k=key
day1=day
qty1=qty
n=n+1
10
20

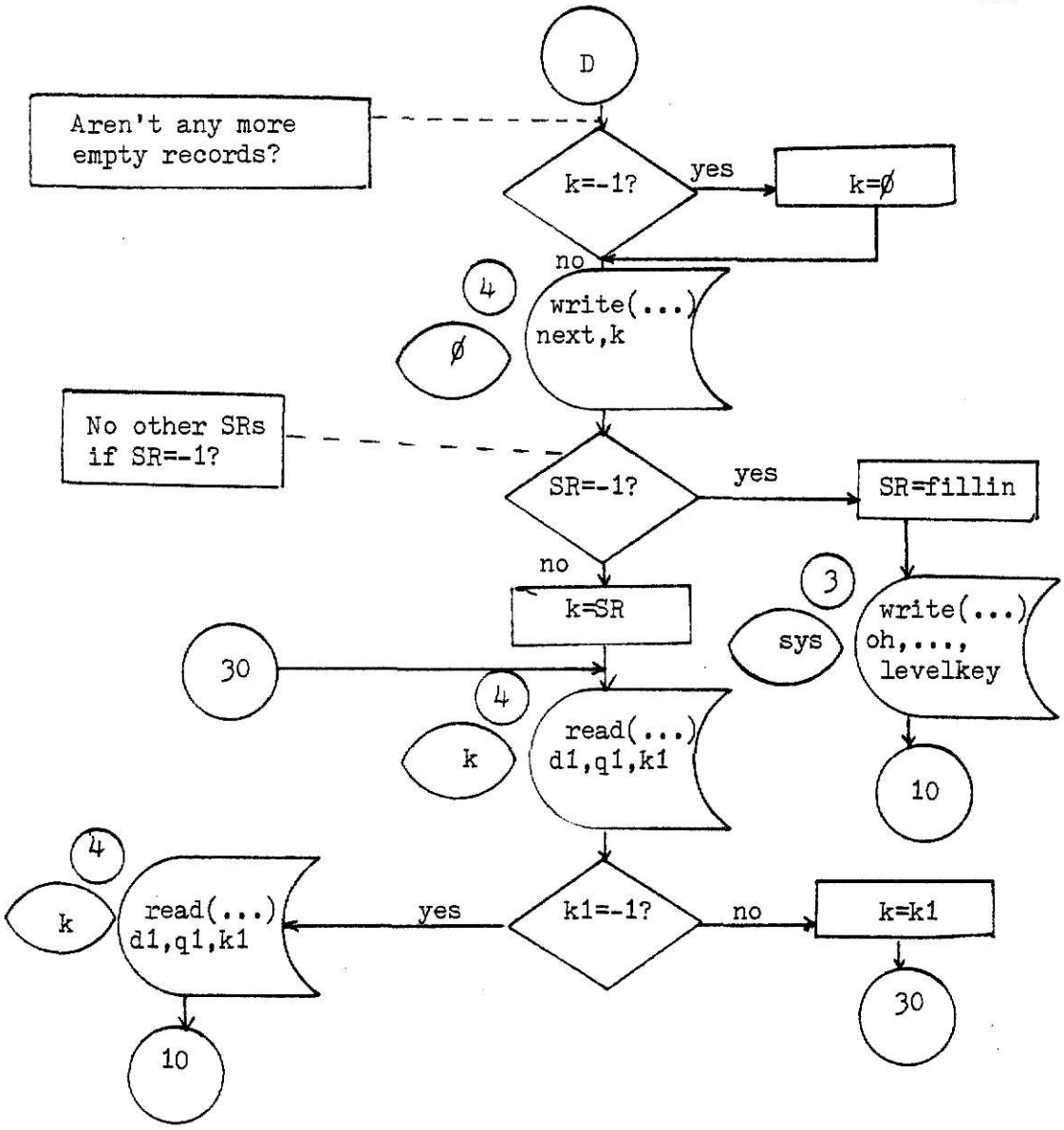
DataPlus

-Subroutine addSR-



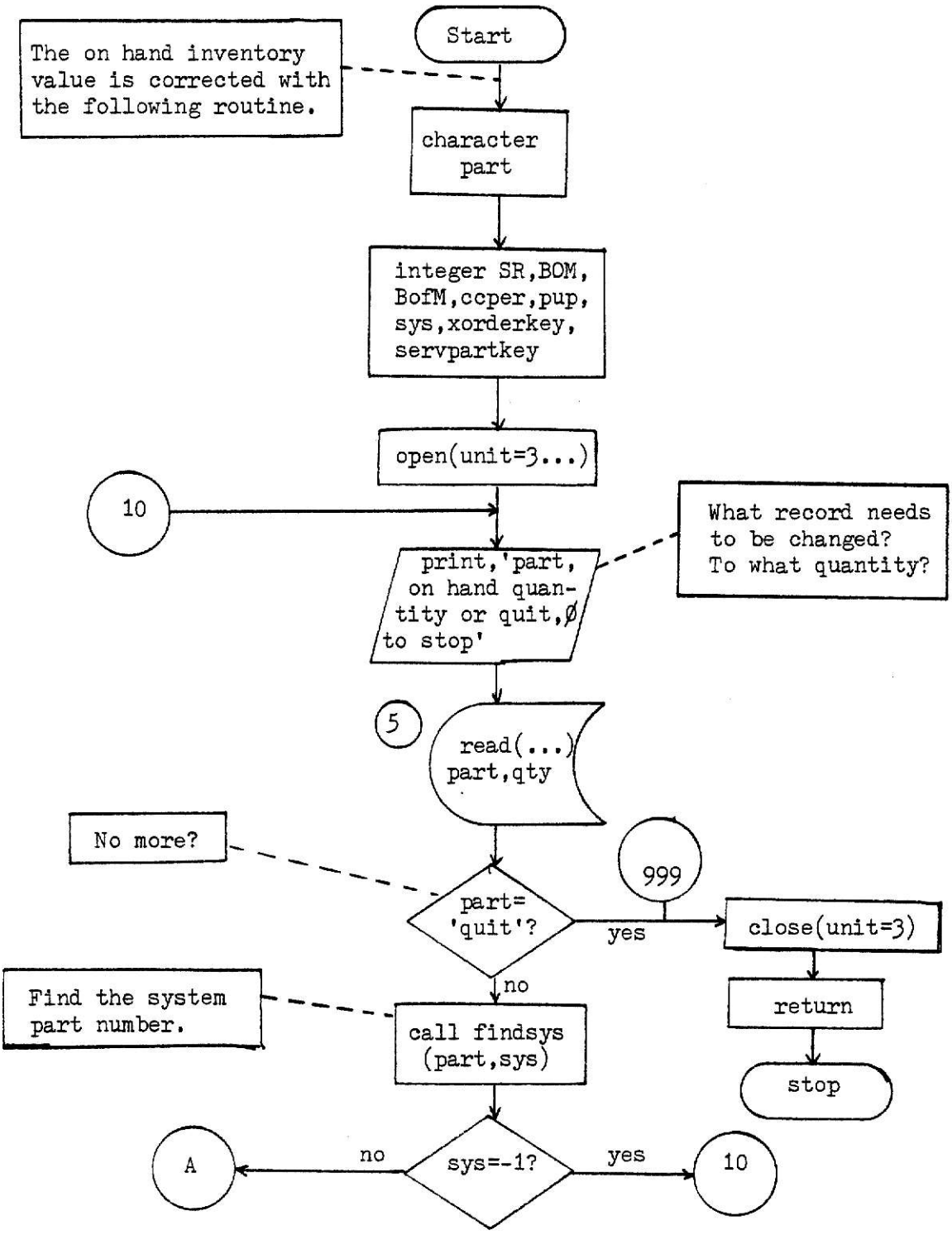

A
New order.
print,
'day,
qty'
5
read(...)
newday,
newqty
Find the empty
records.
4
ø
read(...)
next,
fillin
close(unit=3)
open(unit=3...)
3
sys
read(...)
oh,...,
levelkey
fillin=ø?
no
C
yes
key=next
4
key
write(...)
newday,
newqty,-1
next=next+1
B

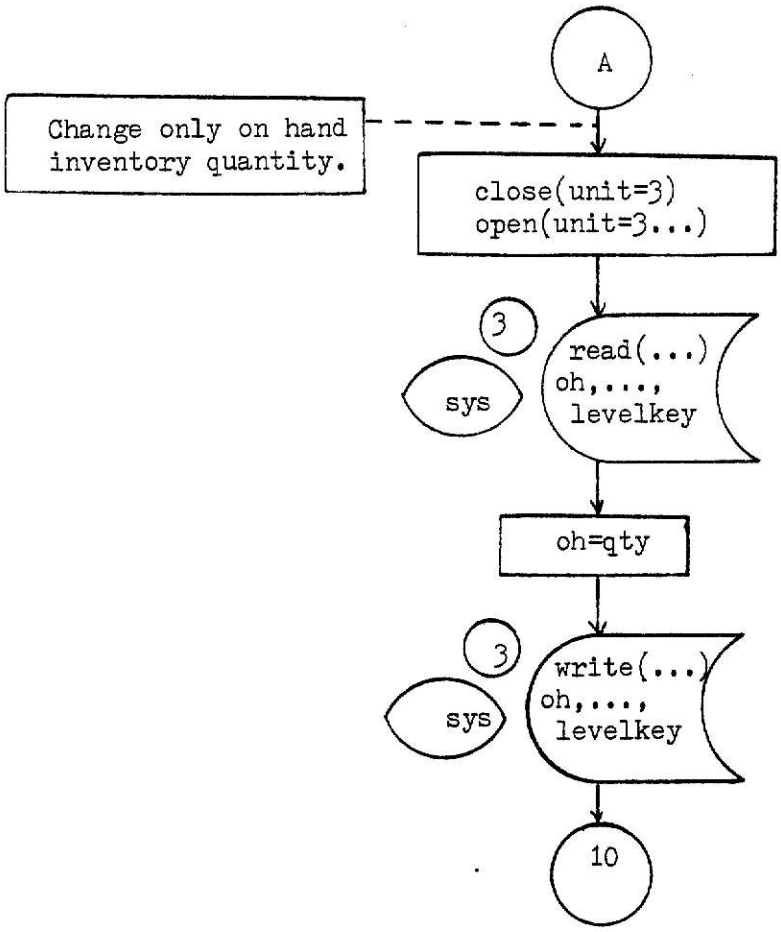

B
4
ø
write(...)
next,fillin
Link new SR
to old SRs.
SR=-1?
yes
SR=key
no
k=SR
3
sys
write(...)
oh,...,
levelkey
20
4
k
read(...)
d1,q1,k1
10
4
k
write(...)
d1,q1,key
yes
k1=-1?
no
k=k1
10
20
C
4
fillin
read(...)
d,q,k
4
fillin
write(...)
newday,
newqty,-1
D


D
Aren't any more empty records?
k=-1?
yes
k=∅
no
4
∅
write(...) next,k
No other SRs if SR=-1?
SR=-1?
yes
SR=fillin
no
k=SR
3
sys
write(...) oh,..., levelkey
30
4
k
read(...) d1,q1,k1
10
4
k
read(...) d1,q1,k1
yes
k1=-1?
no
k=k1
30
10

DataPlus

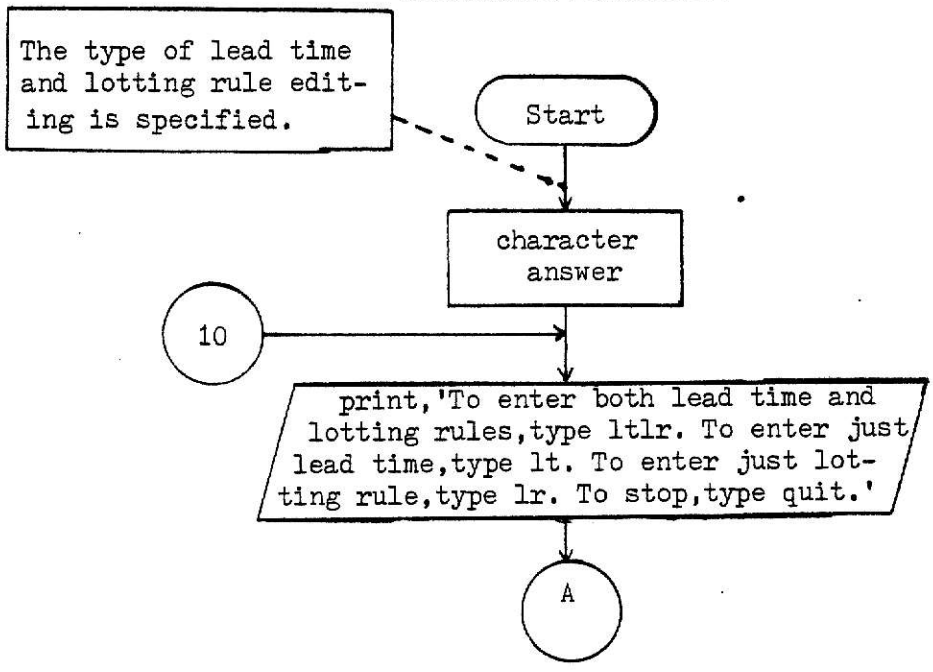
-Subroutine ohchange-

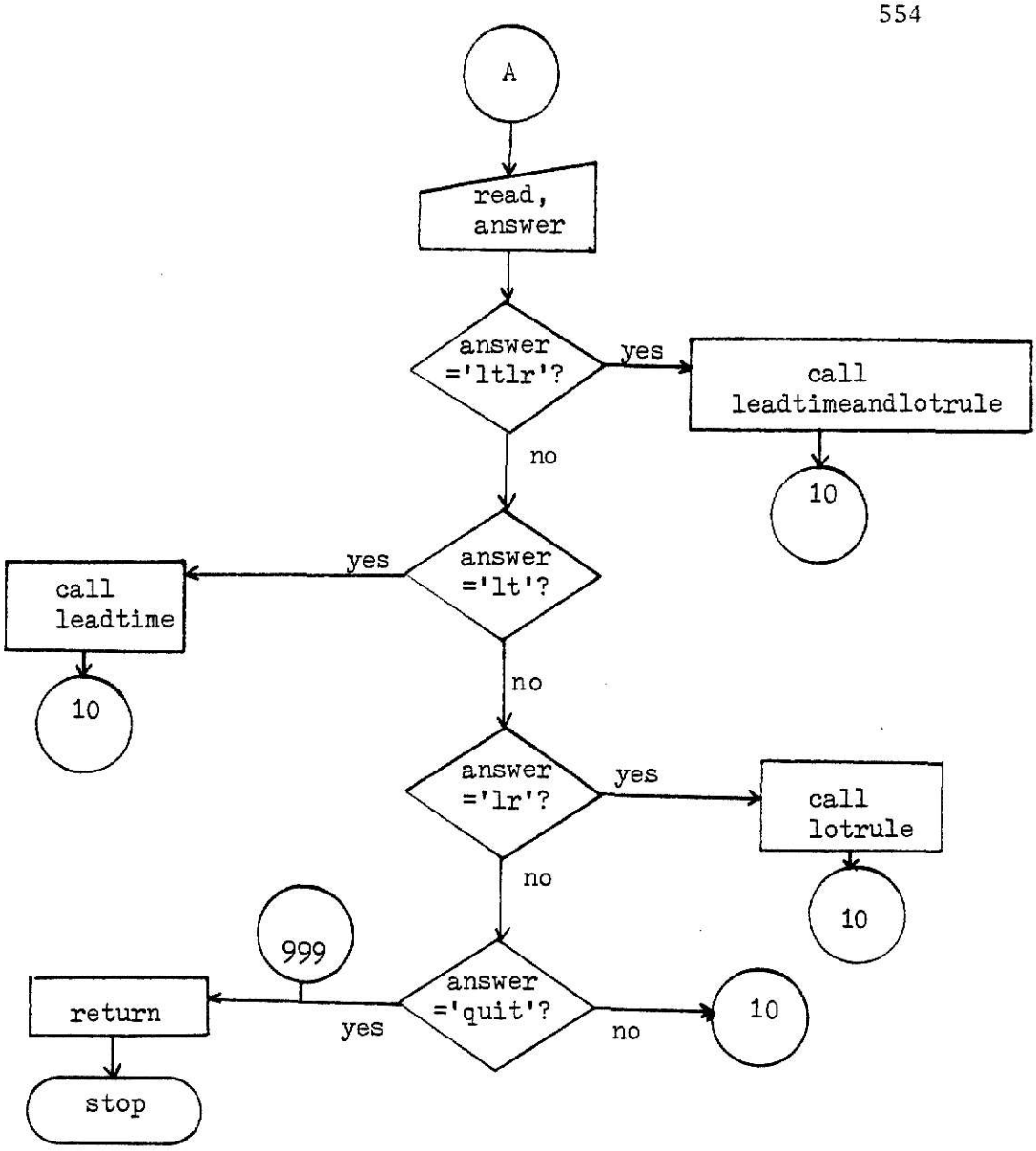




DataPlus

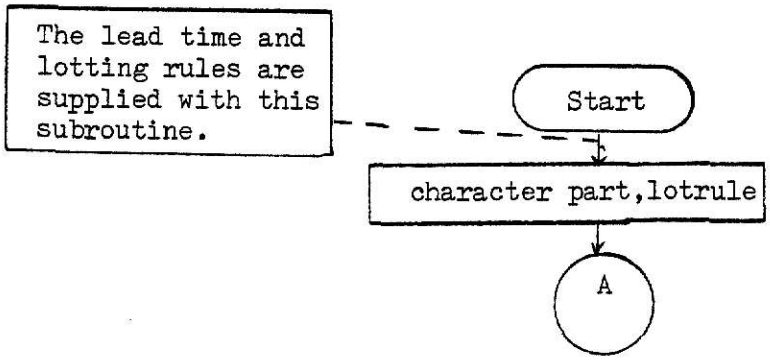
-Subroutine leadandlot-

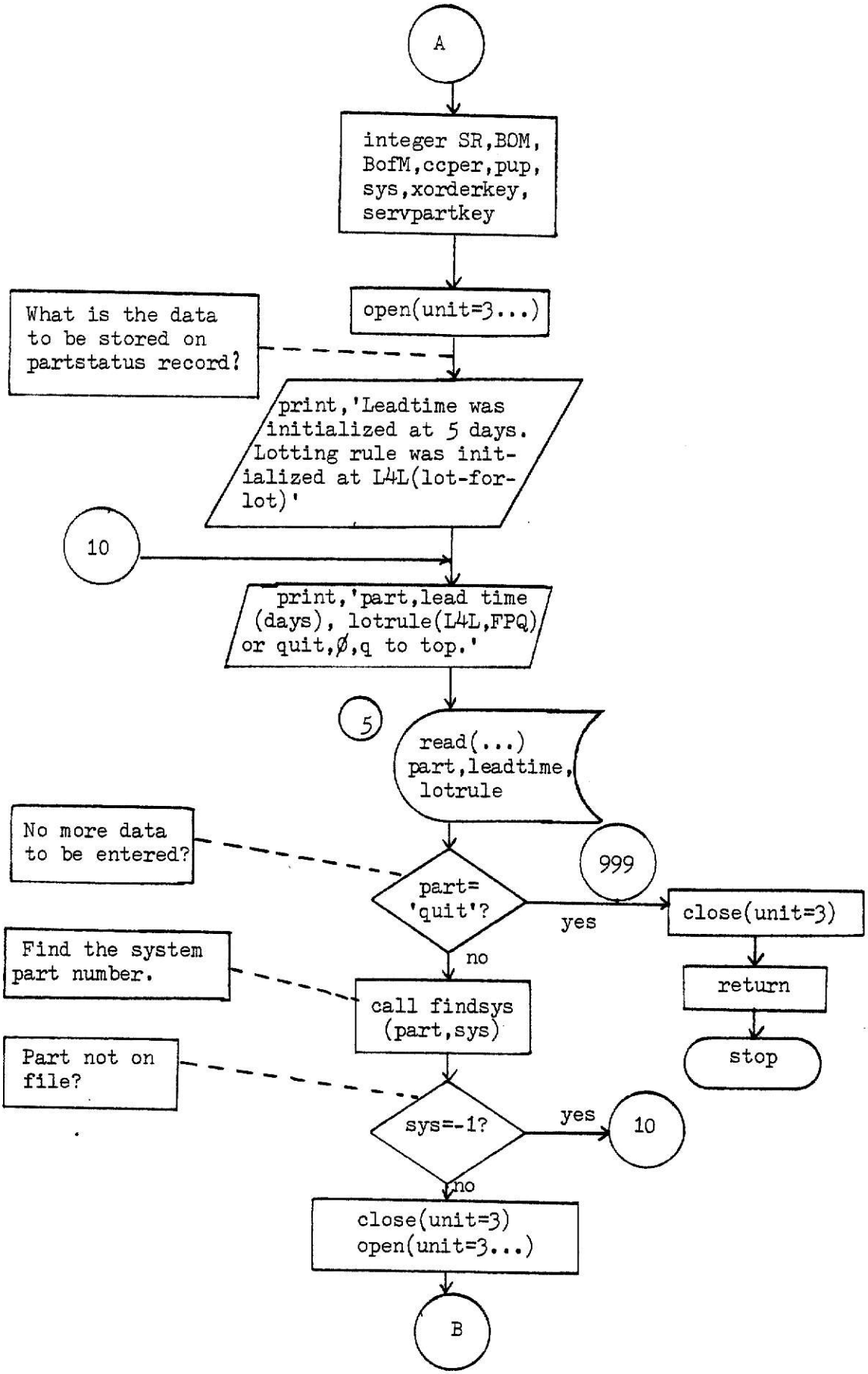


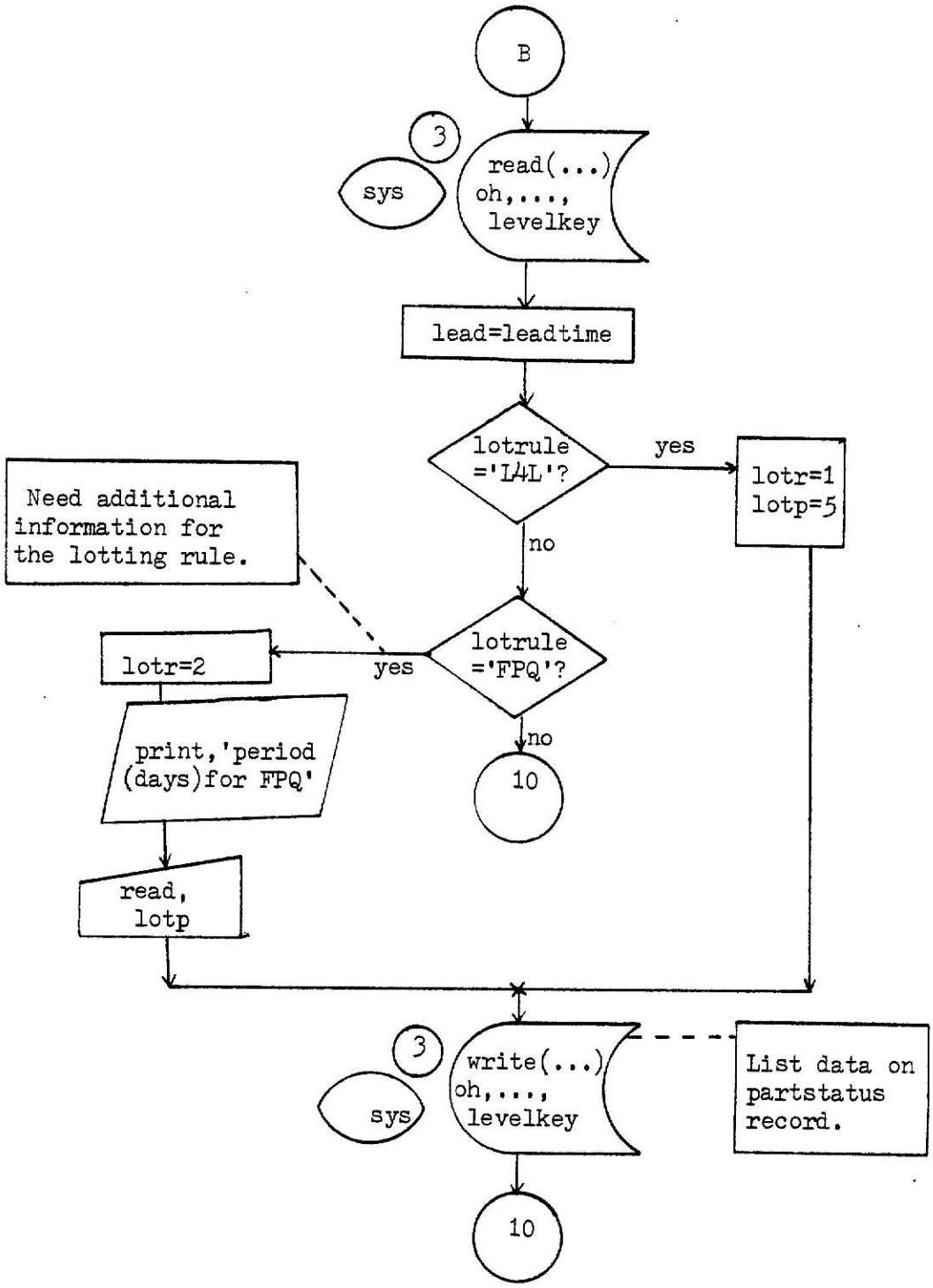


DataPlus

-Subroutine leadtimeandlotrule -

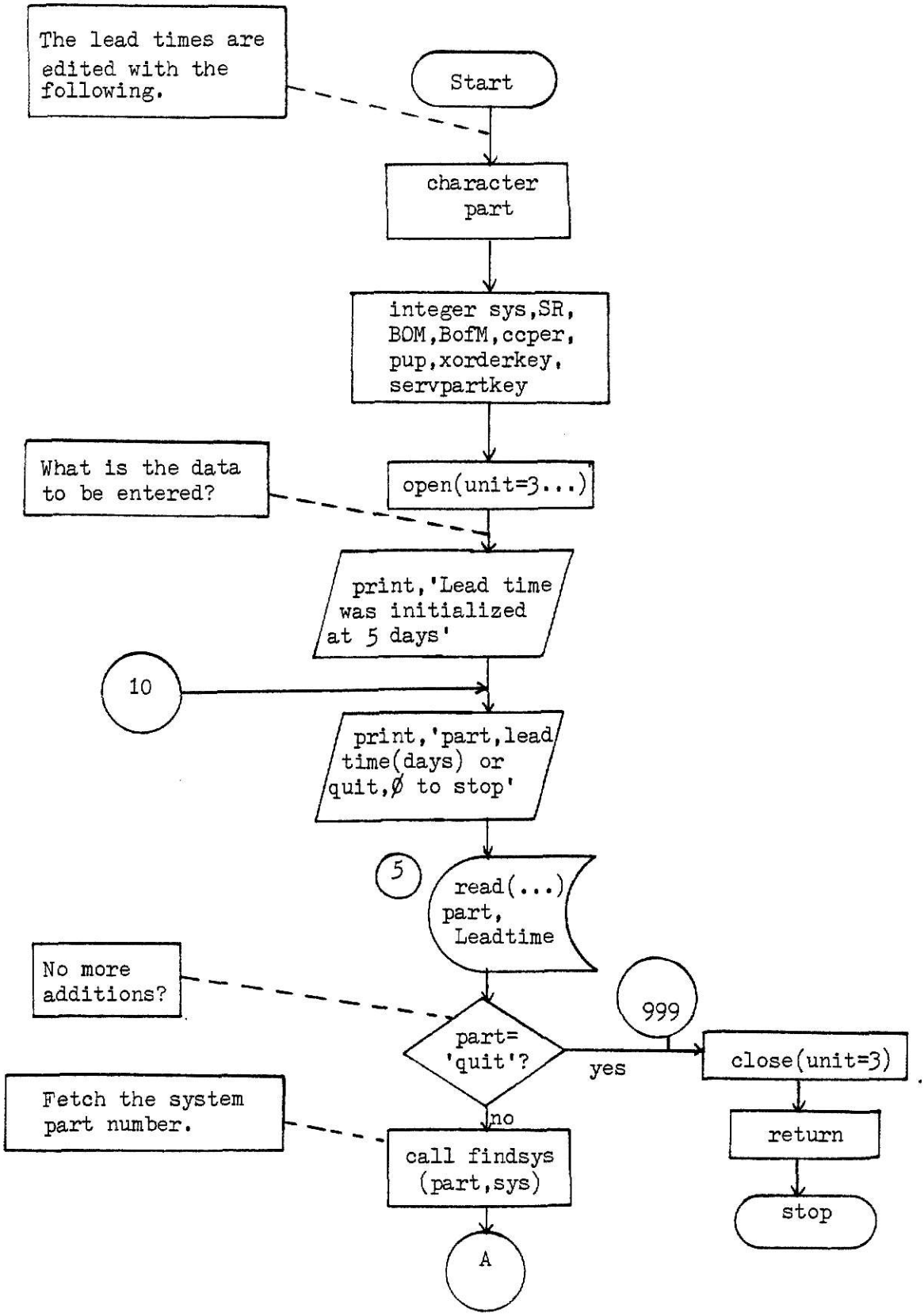


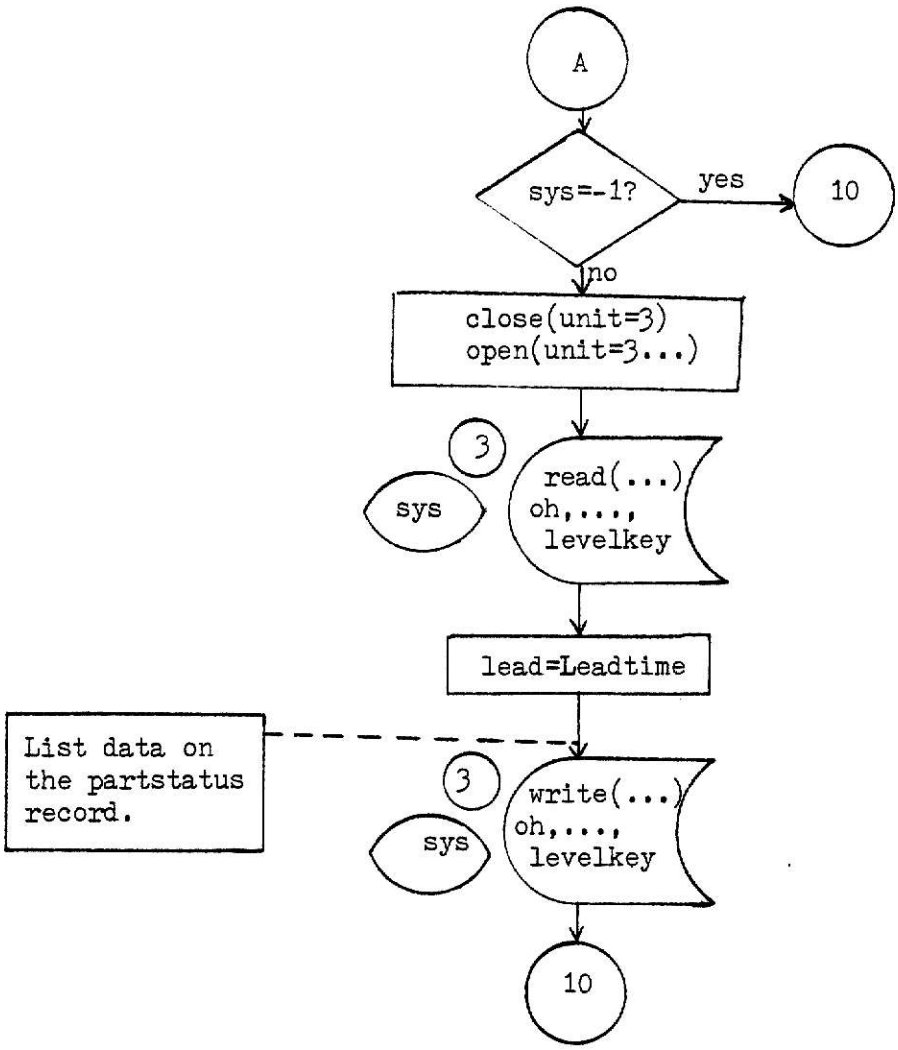

A
integer SR,BOM, BofM,ccper,pup, sys,xorderkey, servpartkey
open(unit=3...)
What is the data to be stored on partstatus record?
print,'Leadtime was initialized at 5 days. Lotting rule was initialized at L4L(lot-for-lot)'
10
print,'part,lead time (days), lotrule(L4L,FPQ) or quit,Ø,q to top.'
5
read(...) part,leadtime, lotrule
No more data to be entered?
part= 'quit'?
999
yes
close(unit=3)
return
stop
no
Find the system part number.
call findsys (part,sys)
Part not on file?
sys=-1?
yes
10
no
close(unit=3) open(unit=3...)
B


B
3
sys
read(...)
oh,...,
levelkey
lead=leadtime
lotrule
='L4L'?
yes
lotr=1
lotp=5
no
Need additional
information for
the lotting rule.
lotrule
='FPQ'?
yes
lotr=2
print,'period
(days)for FPQ'
no
10
read,
lotp
3
sys
write(...)
oh,...,
levelkey
List data on
partstatus
record.
10

DataPlus

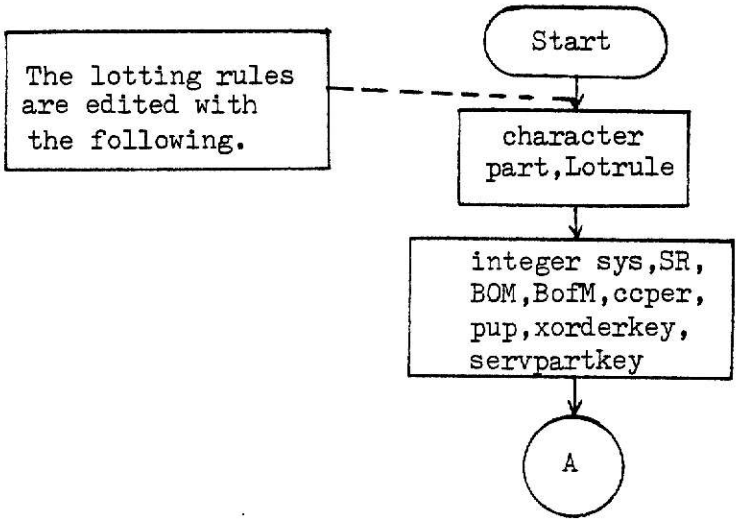
-Subroutine leadtime-

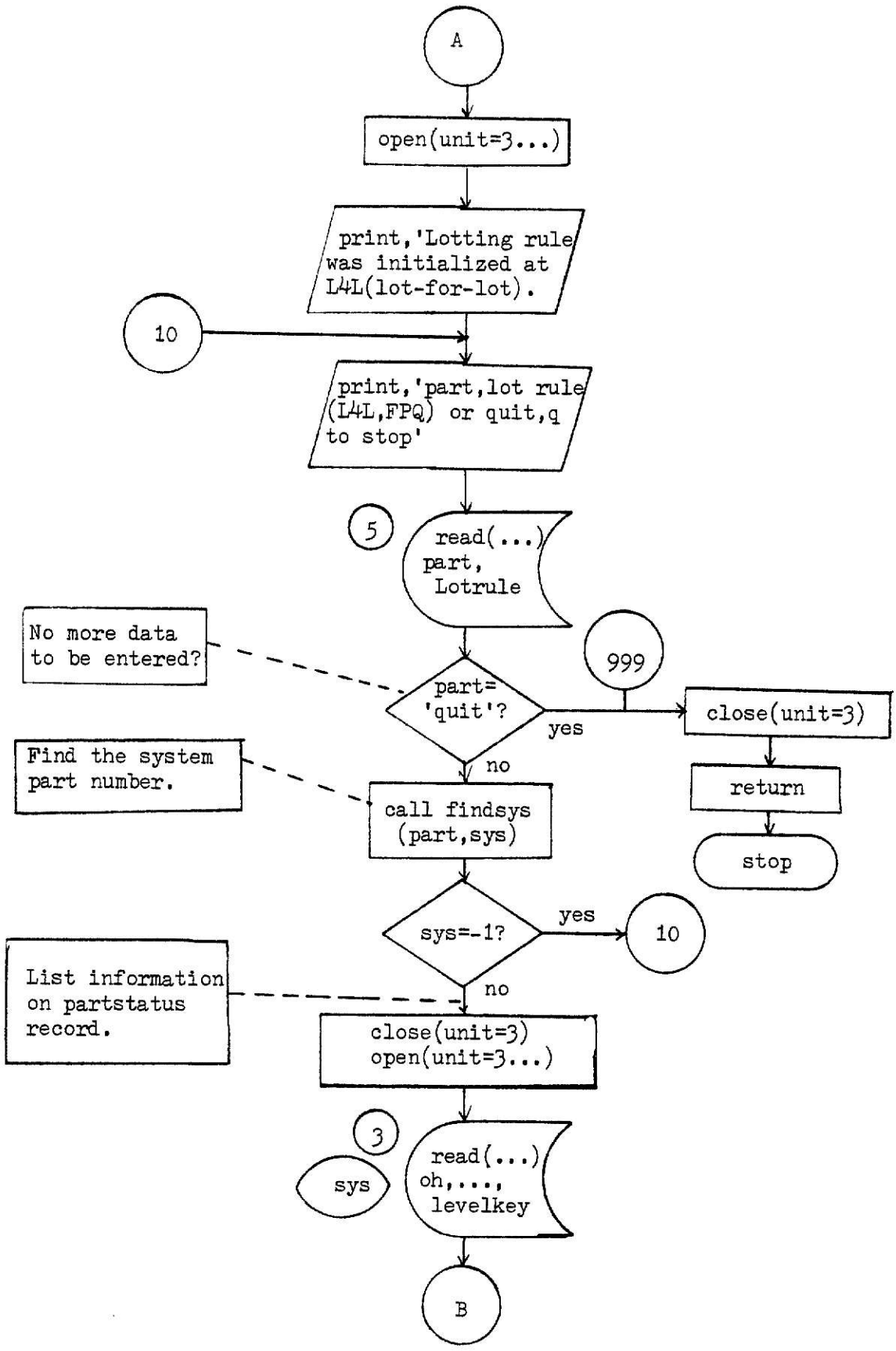


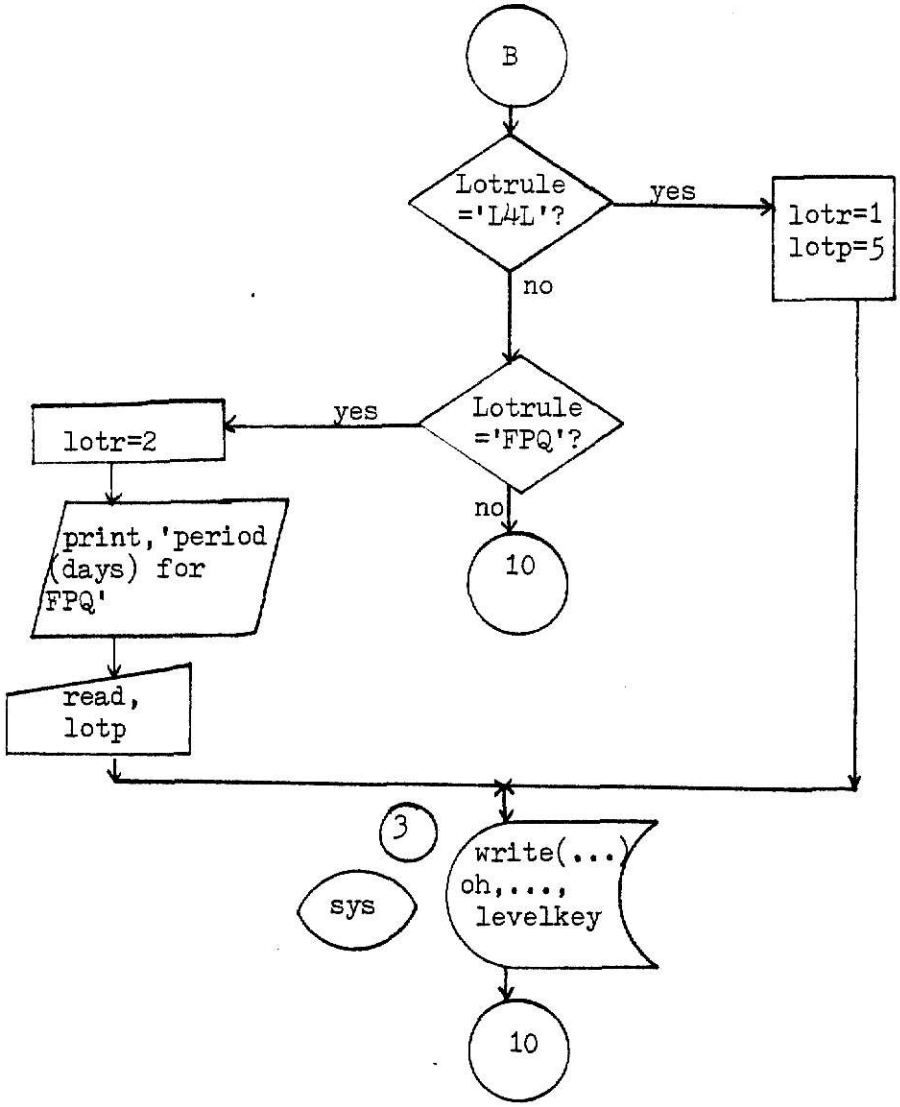


DataPlus

-Subroutine lotrule-

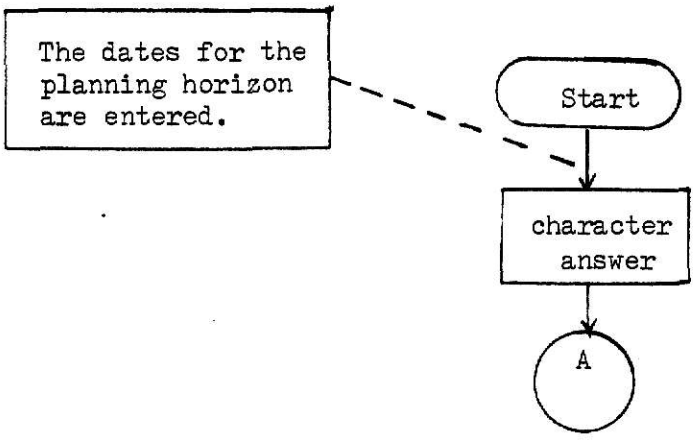


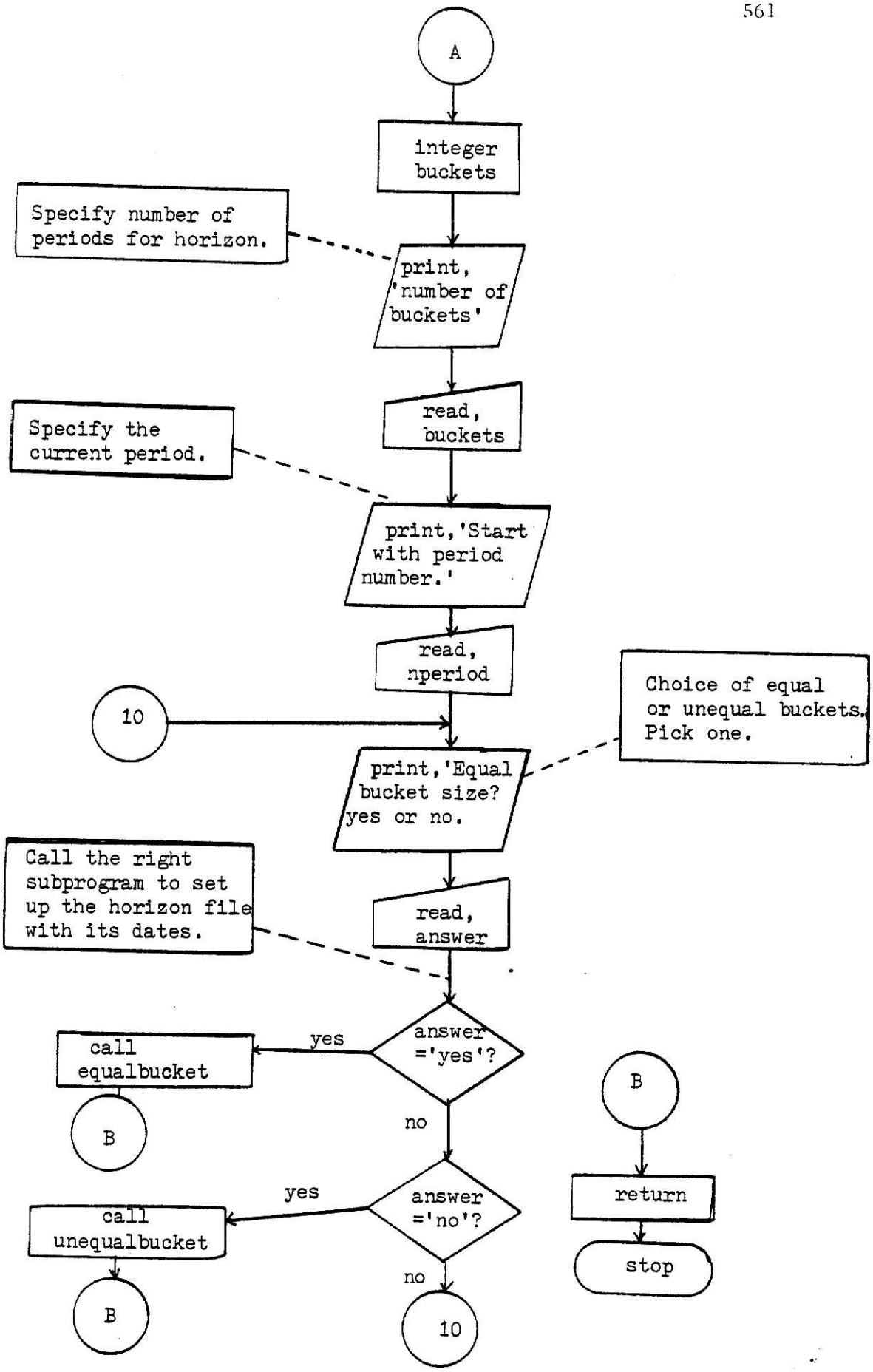

A
open(unit=3...)
print,'Lotting rule was initialized at L4L(lot-for-lot).
10
print,'part,lot rule (L4L,FPQ) or quit,q to stop'
5
read(...) part, Lotrule
No more data to be entered?
part= 'quit'?
999
yes
close(unit=3)
return
stop
Find the system part number.
no
call findsys (part,sys)
sys=-1?
yes
10
List information on partstatus record.
no
close(unit=3)
open(unit=3...)
3
sys
read(...) oh,...., levelkey
B



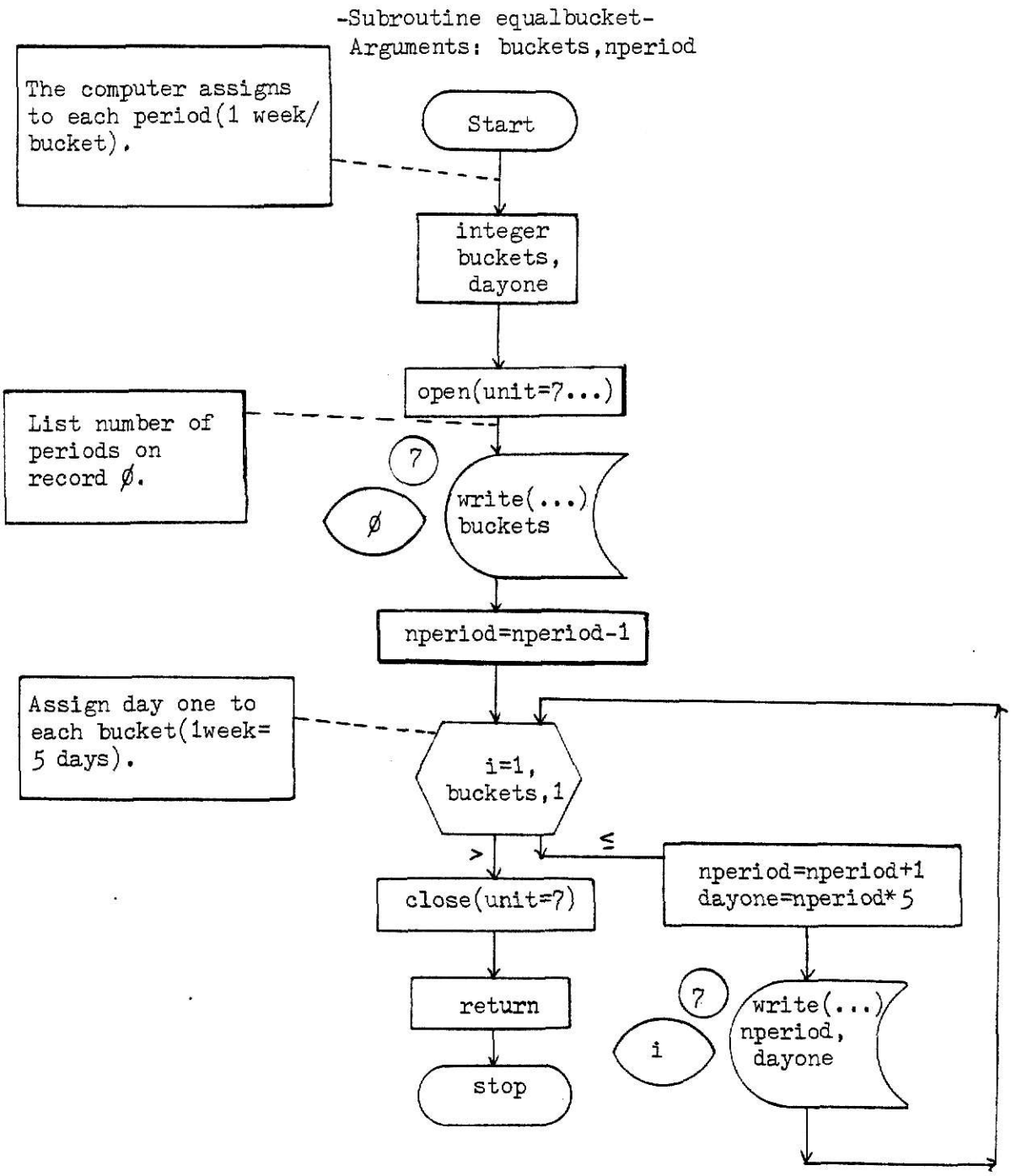
DataPlus

-Subroutine horizon-



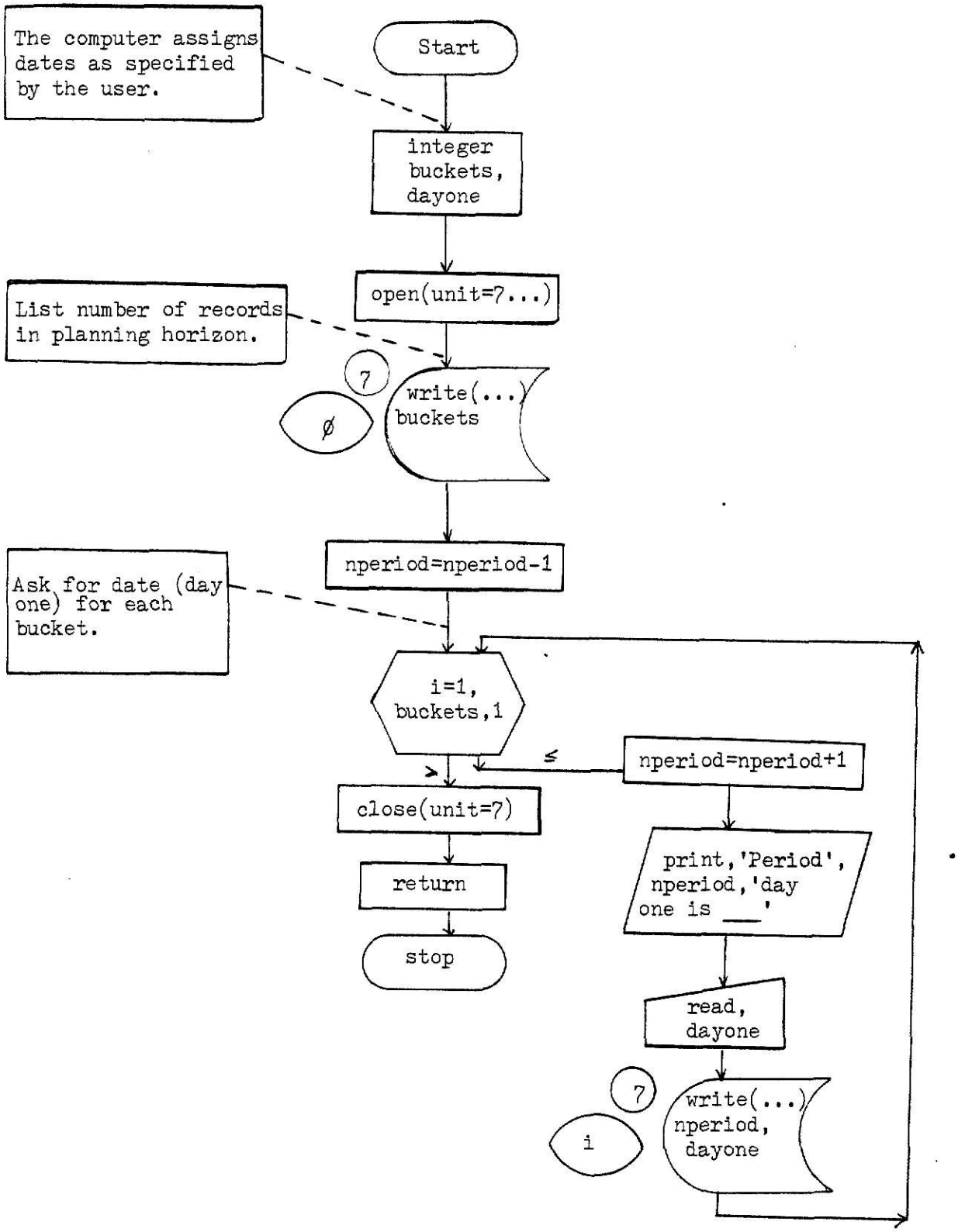

A
integer buckets
Specify number of periods for horizon.
print, 'number of buckets'
read, buckets
Specify the current period.
print,'Start with period number.'
read, nperiod
10
Choice of equal or unequal buckets. Pick one.
print,'Equal bucket size? yes or no.
read, answer
Call the right subprogram to set up the horizon file with its dates.
answer ='yes'?
yes
call equalbucket
B
no
answer ='no'?
yes
call unequalbucket
B
no
10
B
return
stop

DataPlus


-Subroutine equalbucket-
Arguments: buckets,nperiod
The computer assigns to each period(1 week/ bucket).
Start
integer buckets, dayone
open(unit=7...)
List number of periods on record ∅.
7
∅
write(...) buckets
nperiod=nperiod-1
Assign day one to each bucket(1week= 5 days).
i=1, buckets,1
>
≤
close(unit=7)
nperiod=nperiod+1
dayone=nperiod*5
return
7
i
write(...) nperiod, dayone
stop

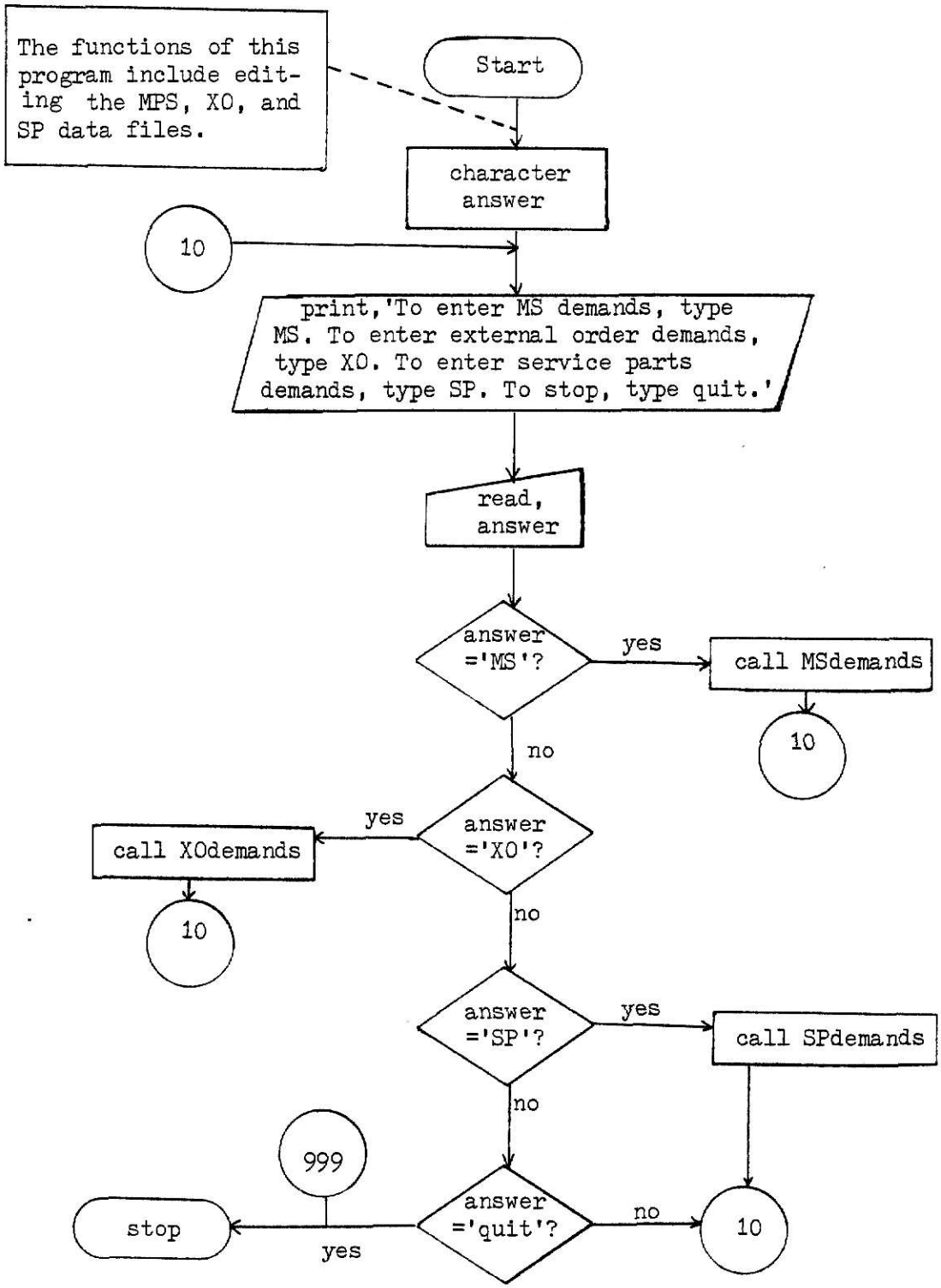
DataPlus

-Subroutine unequalbucket-
Arguments: buckets,nperiod

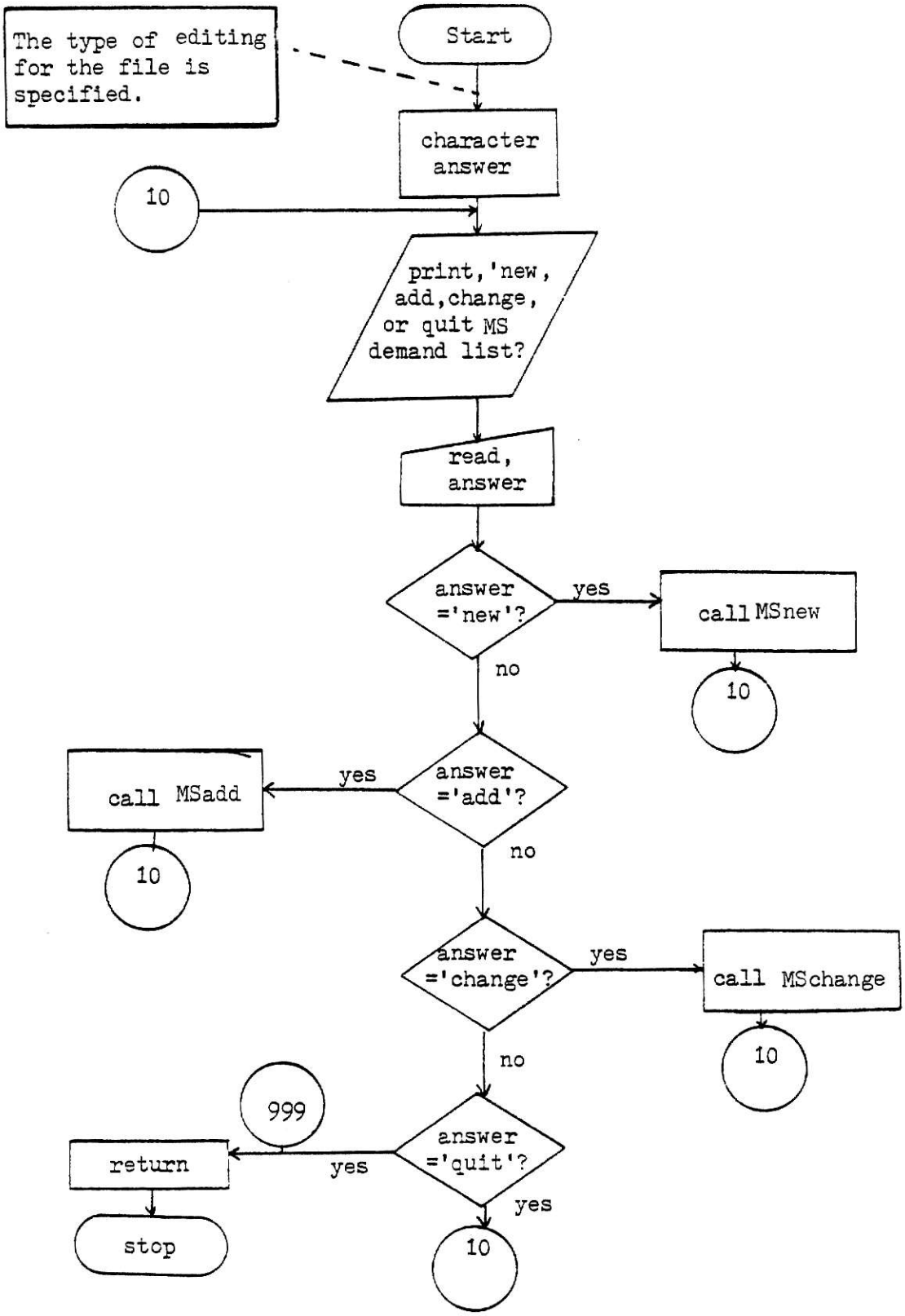


DataPlus2

-Main Program-

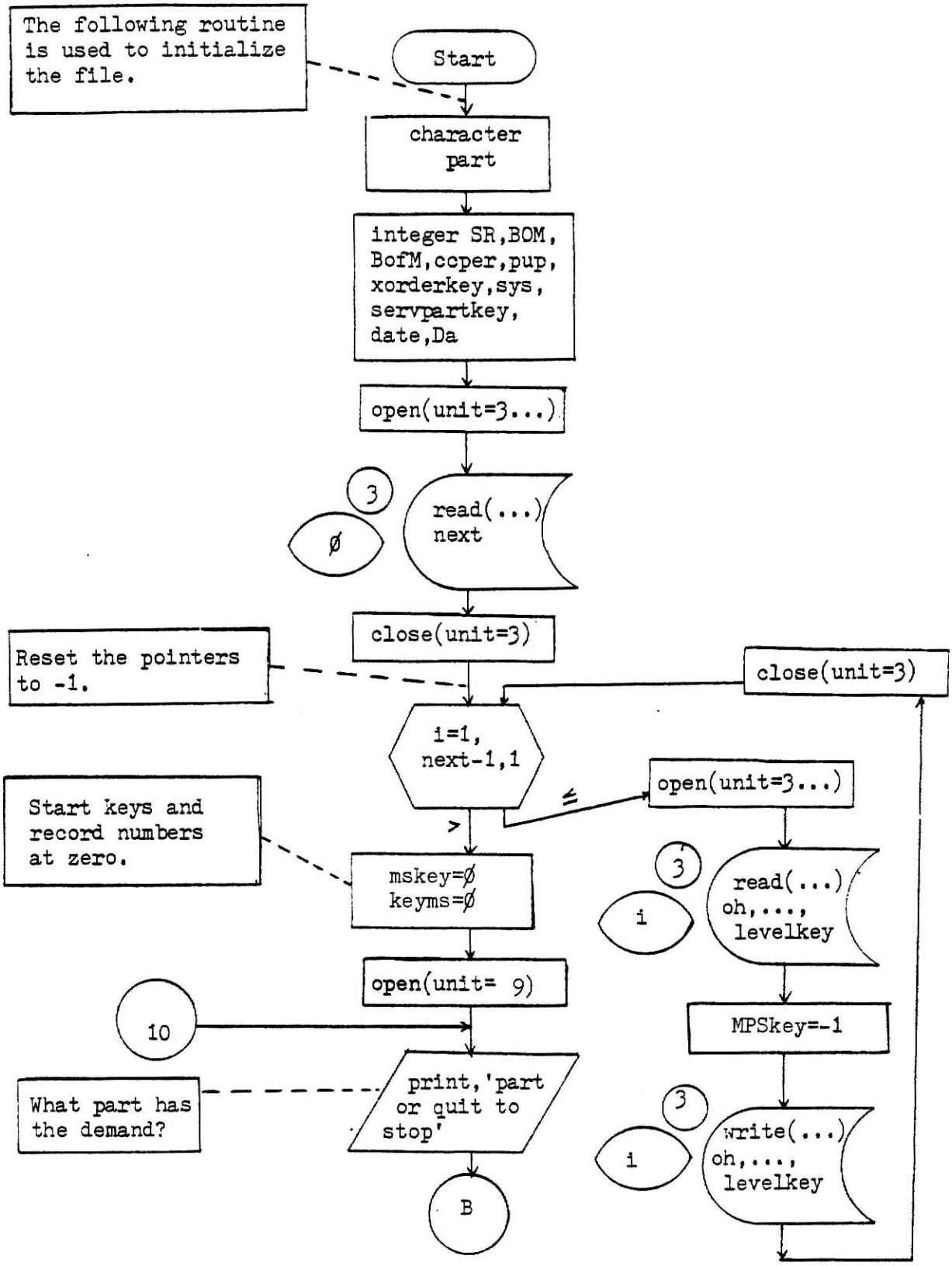


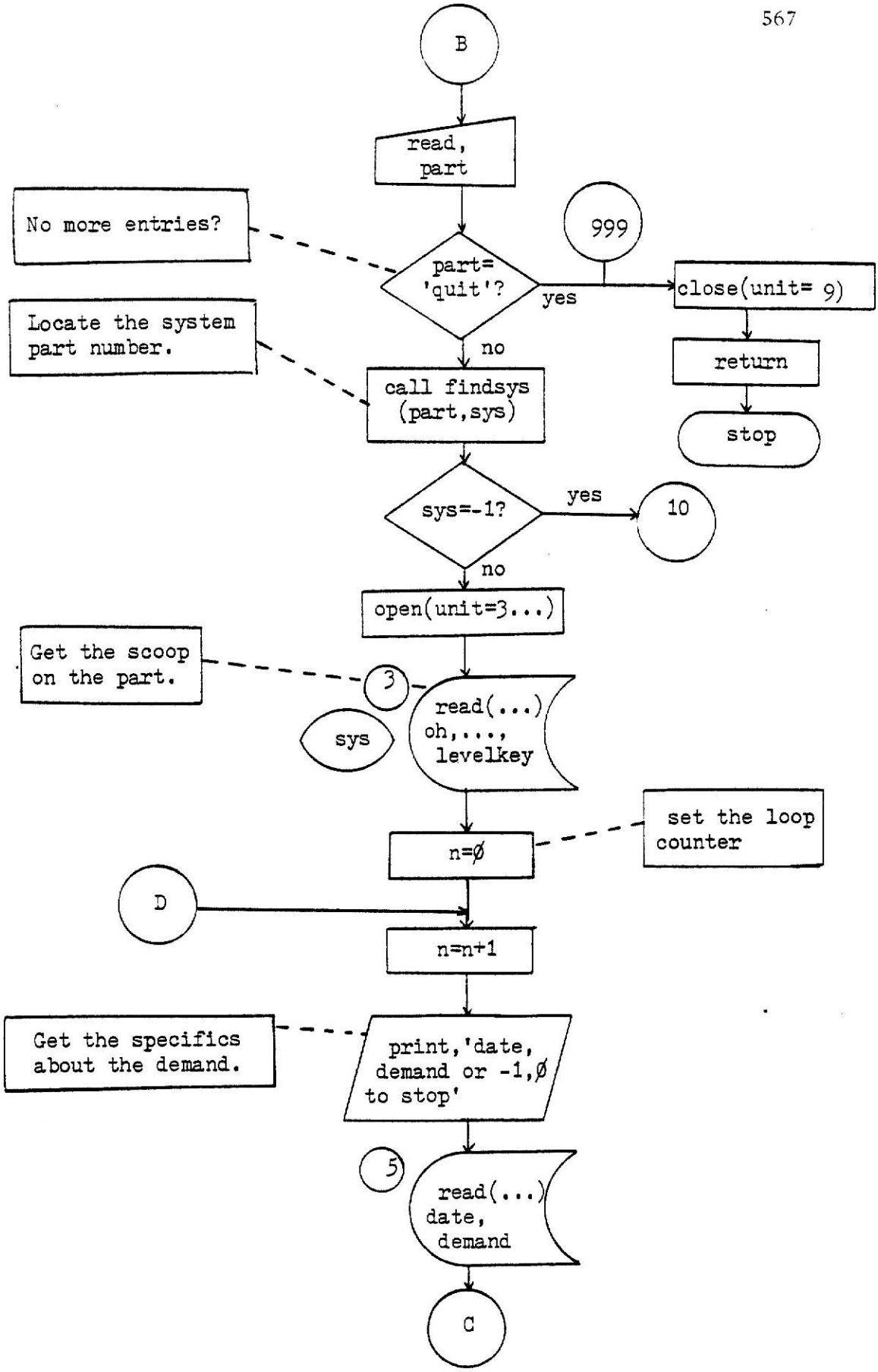
DataPlus2
-Subroutine MSdemands-

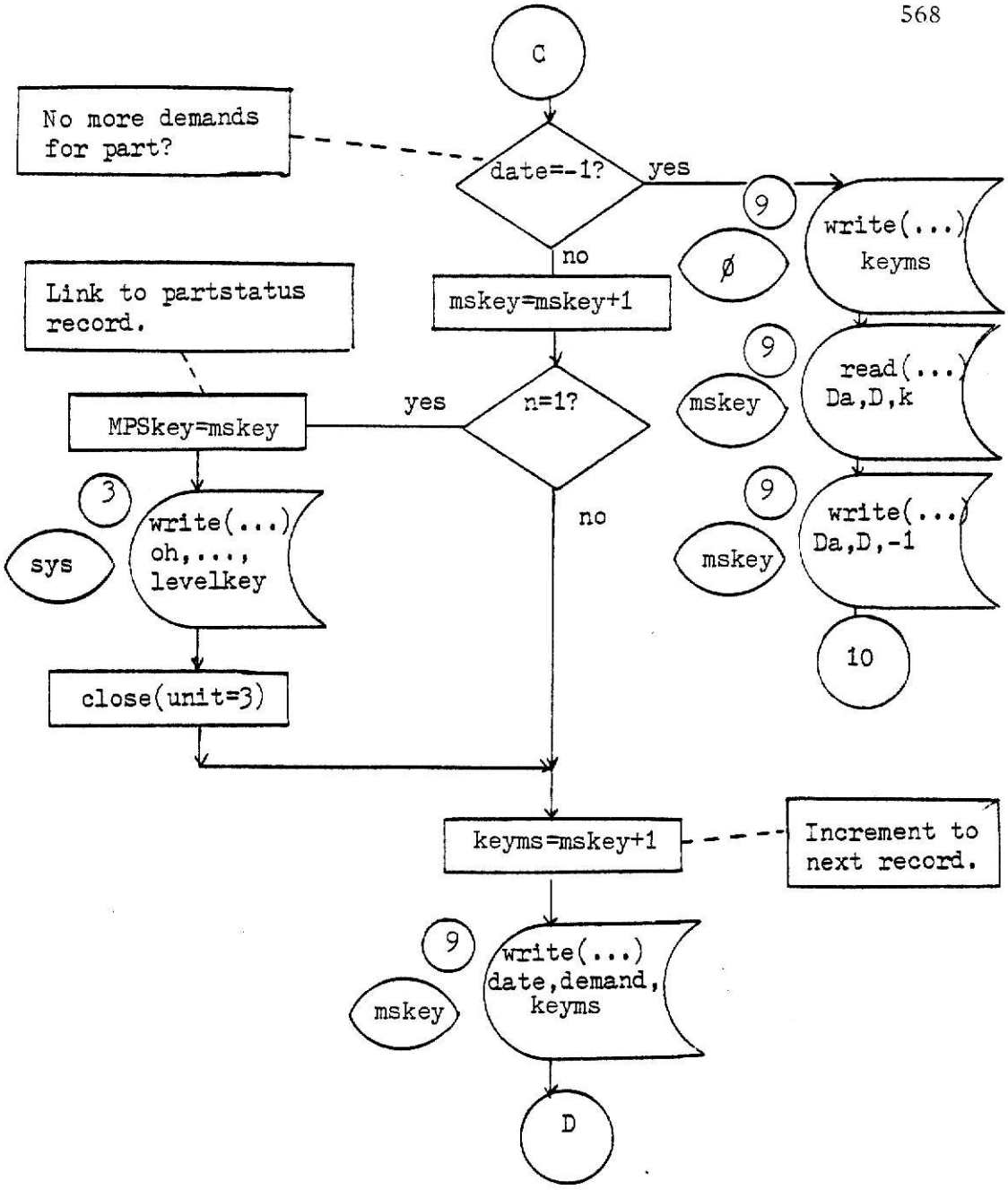


DataPlus2

-Subroutine MSnew-

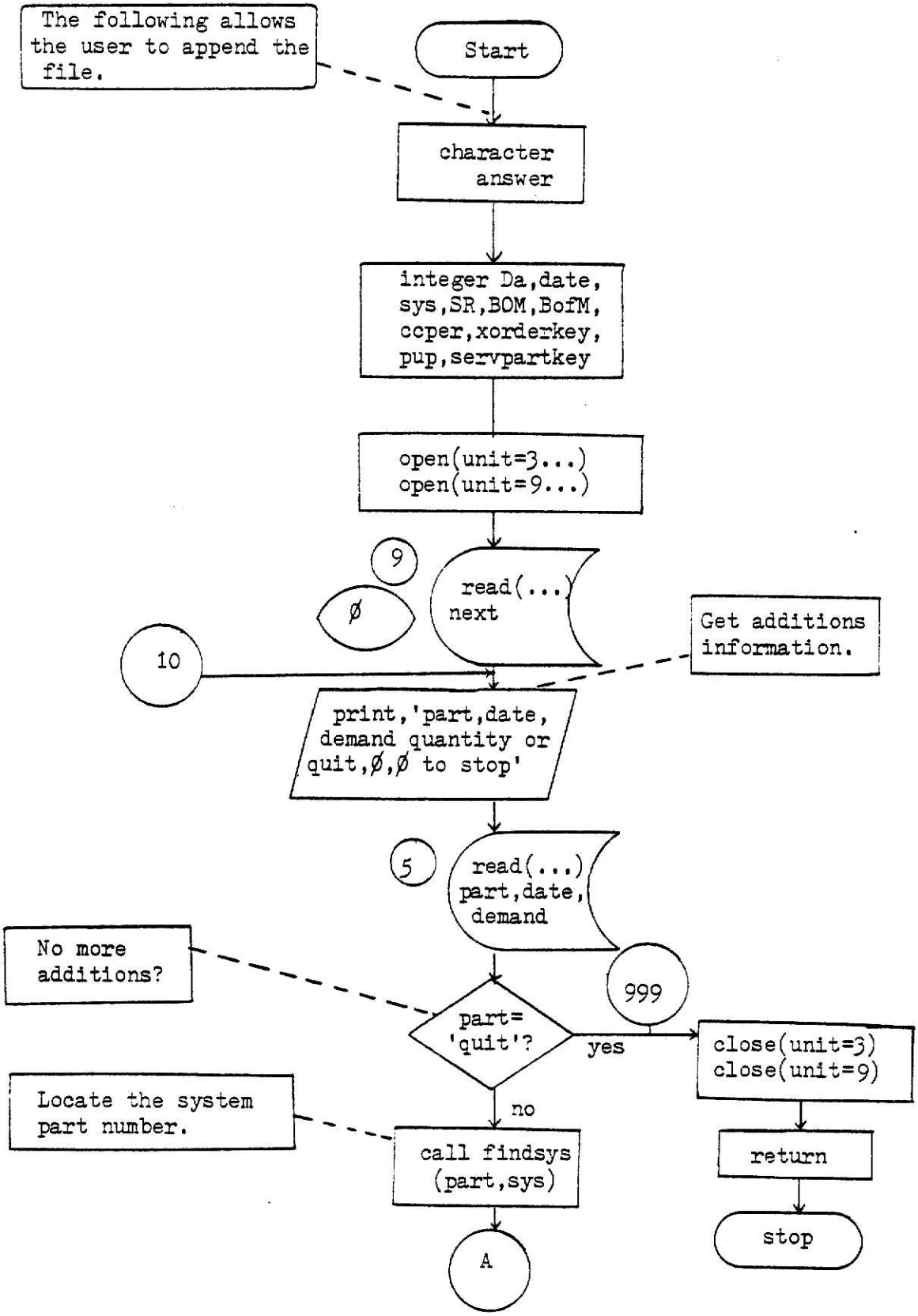


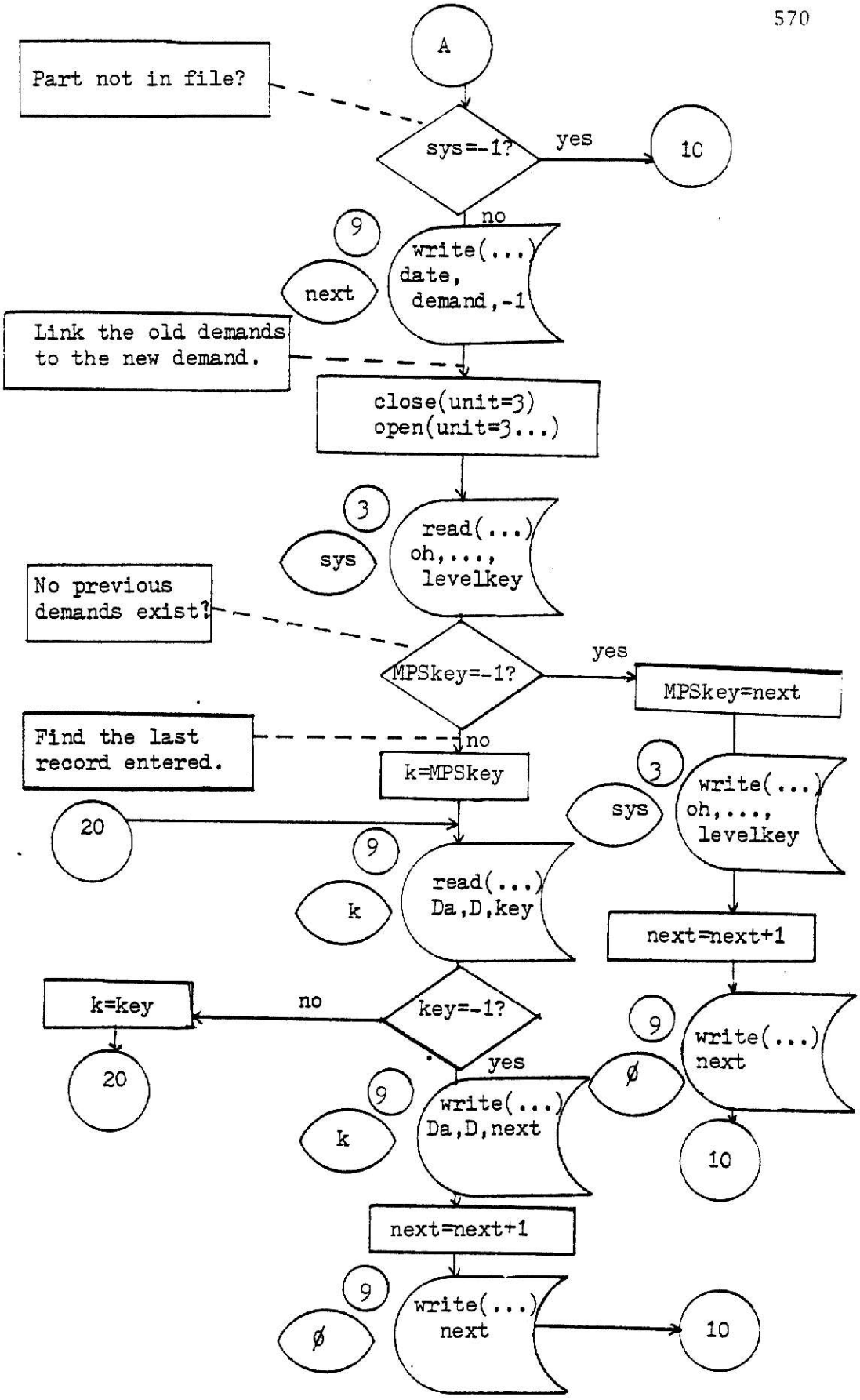

B
read,
part
No more entries?
part=
'quit'?
999
yes
close(unit= 9)
return
stop
Locate the system
part number.
no
call findsys
(part,sys)
sys=-1?
yes
10
no
open(unit=3...)
Get the scoop
on the part.
3
sys
read(...)
oh,...,
levelkey
set the loop
counter
n=ø
D
n=n+1
Get the specifics
about the demand.
print,'date,
demand or -1,ø
to stop'
5
read(...)
date,
demand
C


C
No more demands for part?
date=-1?
yes
no
9
ø
write(...) keyms
Link to partstatus record.
mskey=mskey+1
MPSkey=mskey
yes
n=1?
9
mskey
read(... Da,D,k
9
mskey
write(... Da,D,-1
10
3
sys
write(...) oh,..., levelkey
close(unit=3)
no
keyms=mskey+1
Increment to next record.
9
mskey
write(...) date,demand, keyms
D

DataPlus2

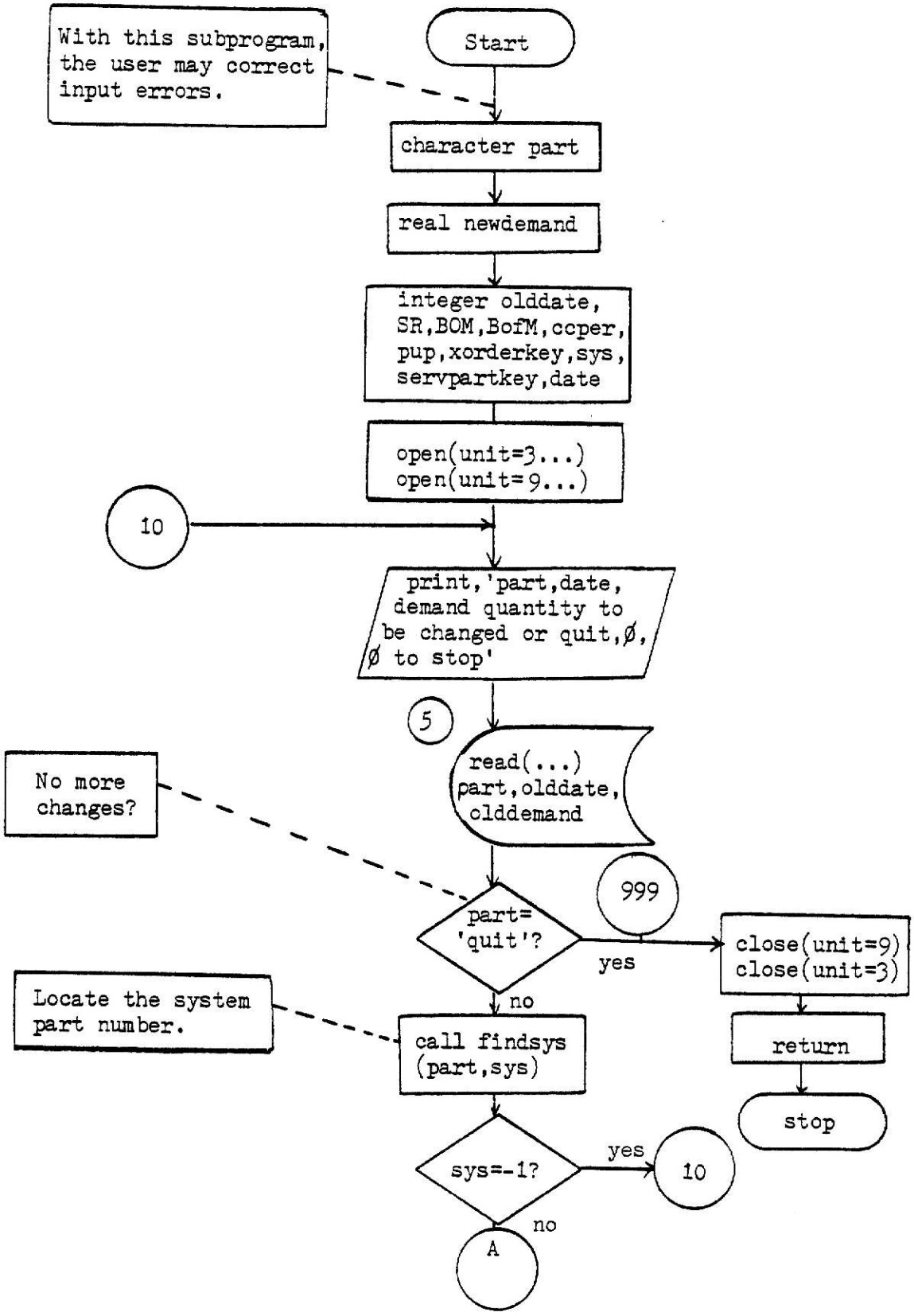
-Subroutine MSadd-

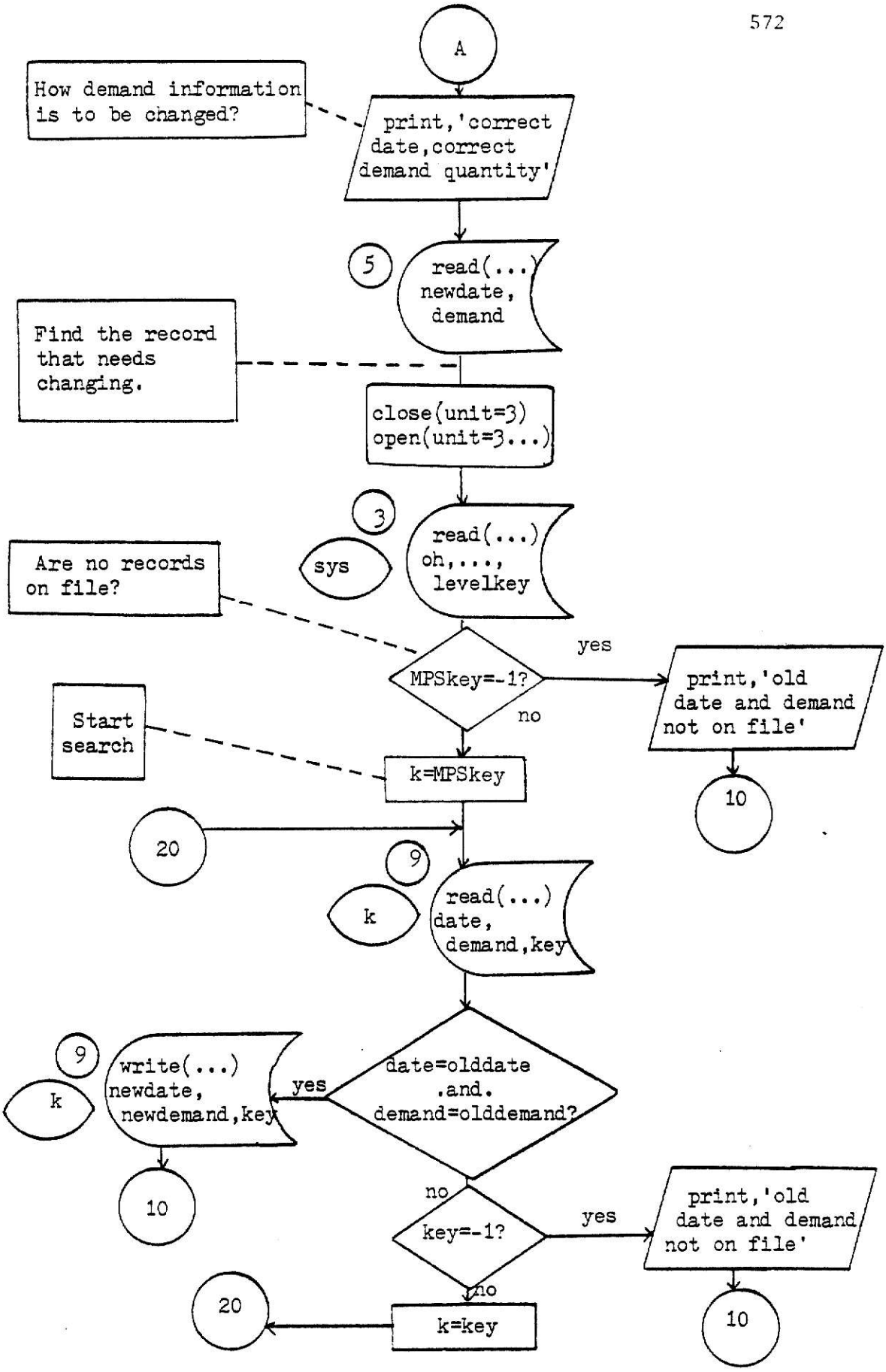



A
Part not in file?
sys=-1?
yes
10
no
9
next
write(...)
date,
demand,-1
Link the old demands to the new demand.
close(unit=3)
open(unit=3...)
3
sys
read(...)
oh,...,
levelkey
No previous demands exist?
MPSkey=-1?
yes
MPSkey=next
Find the last record entered.
no
k=MPSkey
3
sys
write(...)
oh,...,
levelkey
20
9
k
read(...)
Da,D,key
next=next+1
k=key
no
key=-1?
9
ø
write(...)
next
20
yes
9
k
write(...)
Da,D,next
10
next=next+1
9
ø
write(...)
next
10

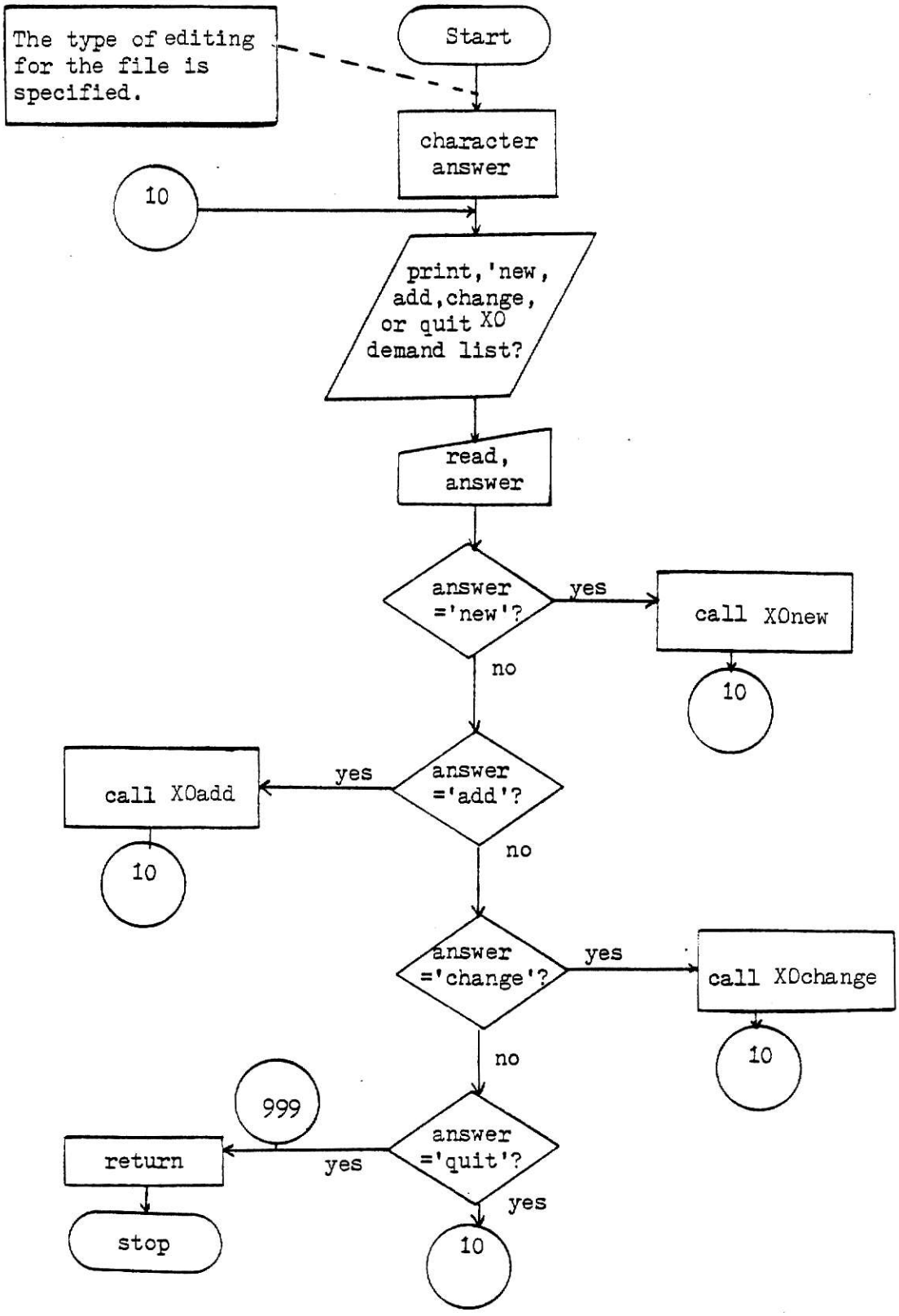
DataPlus2

-Subroutine MSchange-



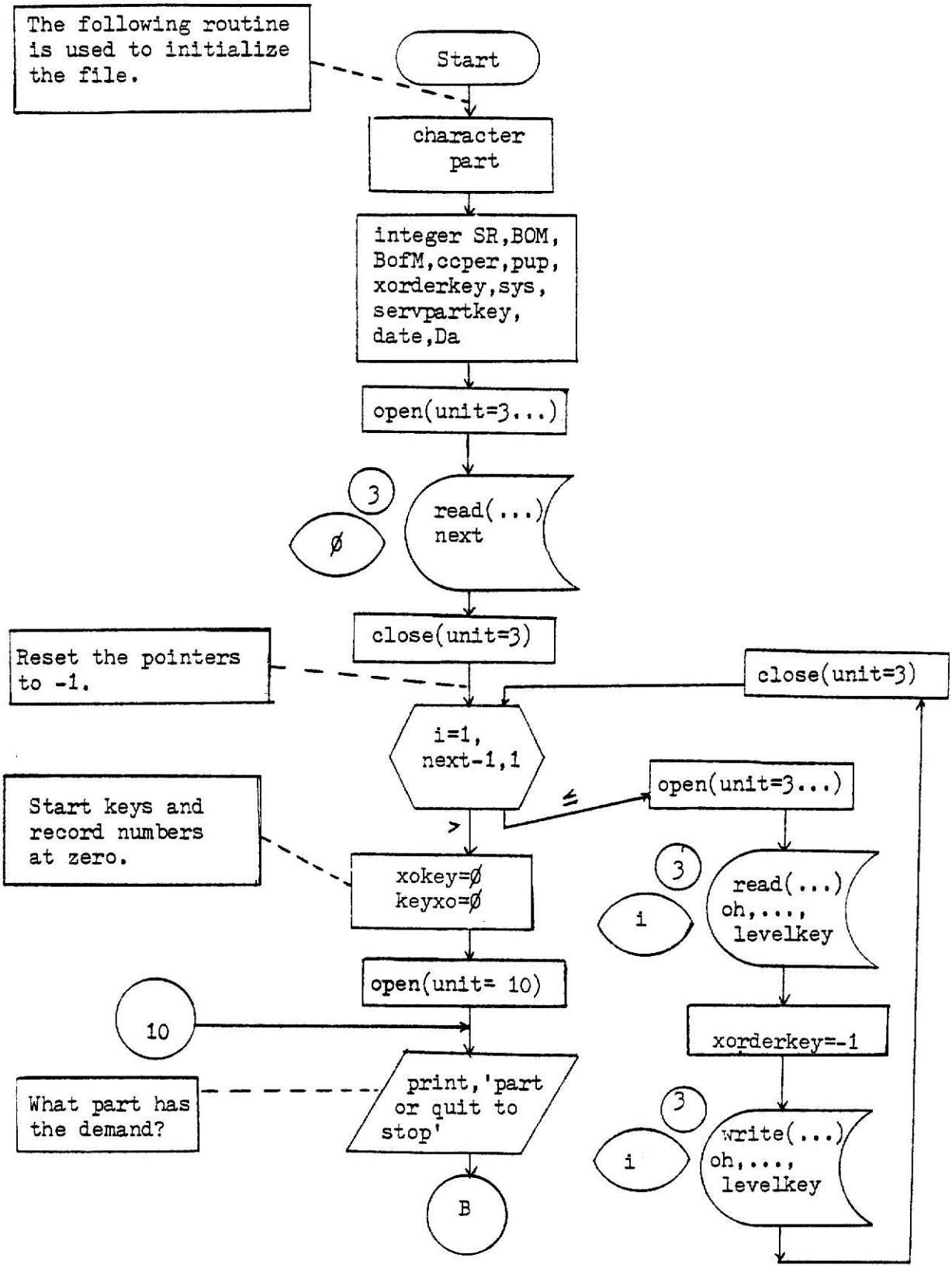

A
How demand information is to be changed?
print,'correct date,correct demand quantity'
5
read(...) newdate, demand
Find the record that needs changing.
close(unit=3) open(unit=3...)
3
sys
read(...) oh,..., levelkey
Are no records on file?
MPSkey=-1?
yes
no
print,'old date and demand not on file'
10
Start search
k=MPSkey
20
9
k
read(...) date, demand,key
date=olddate .and. demand=olddemand?
yes
9
k
write(...) newdate, newdemand,key
10
no
key=-1?
yes
print,'old date and demand not on file'
10
no
k=key
20

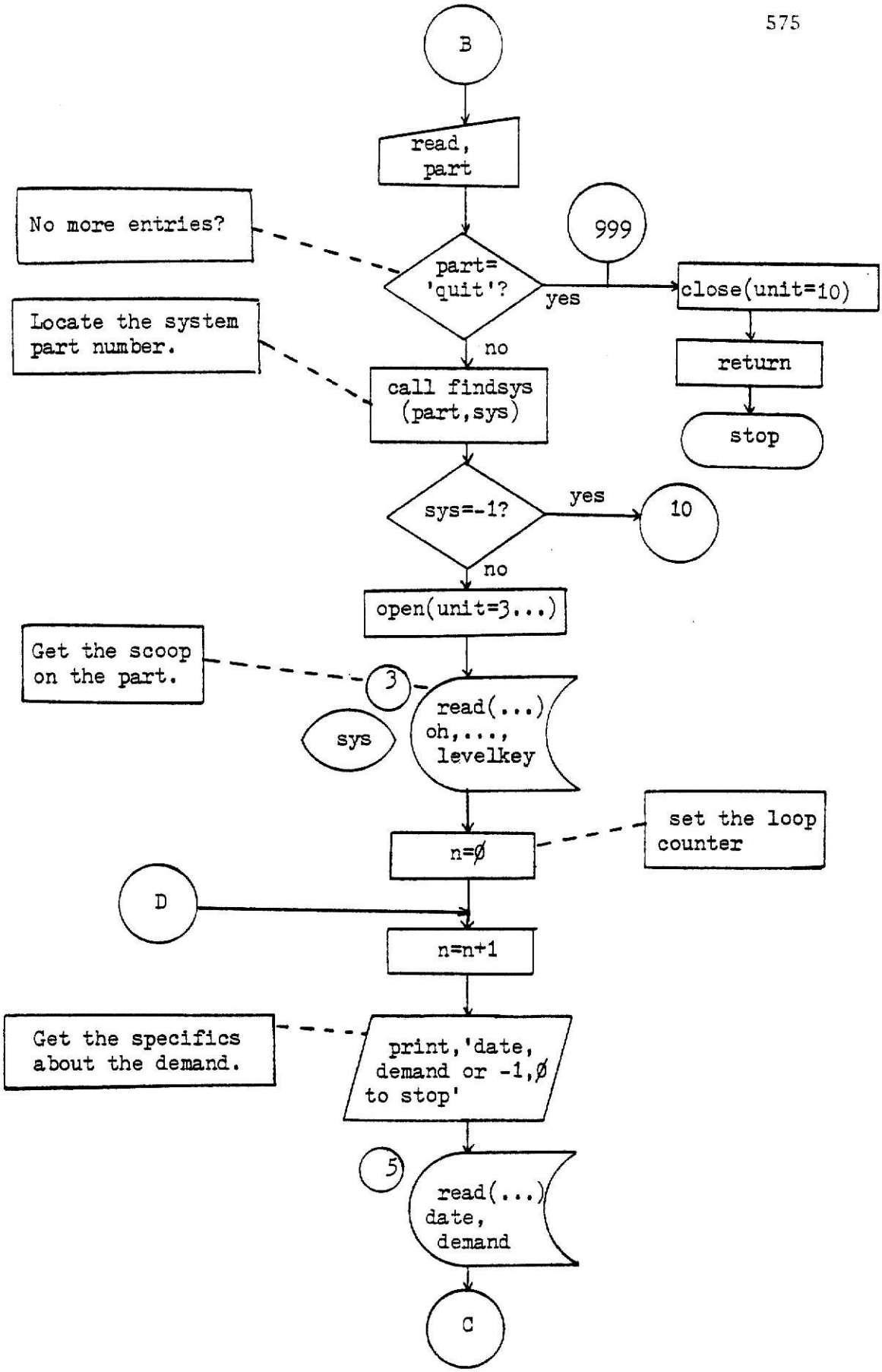
DataPlus2
-Subroutine XOdemands -

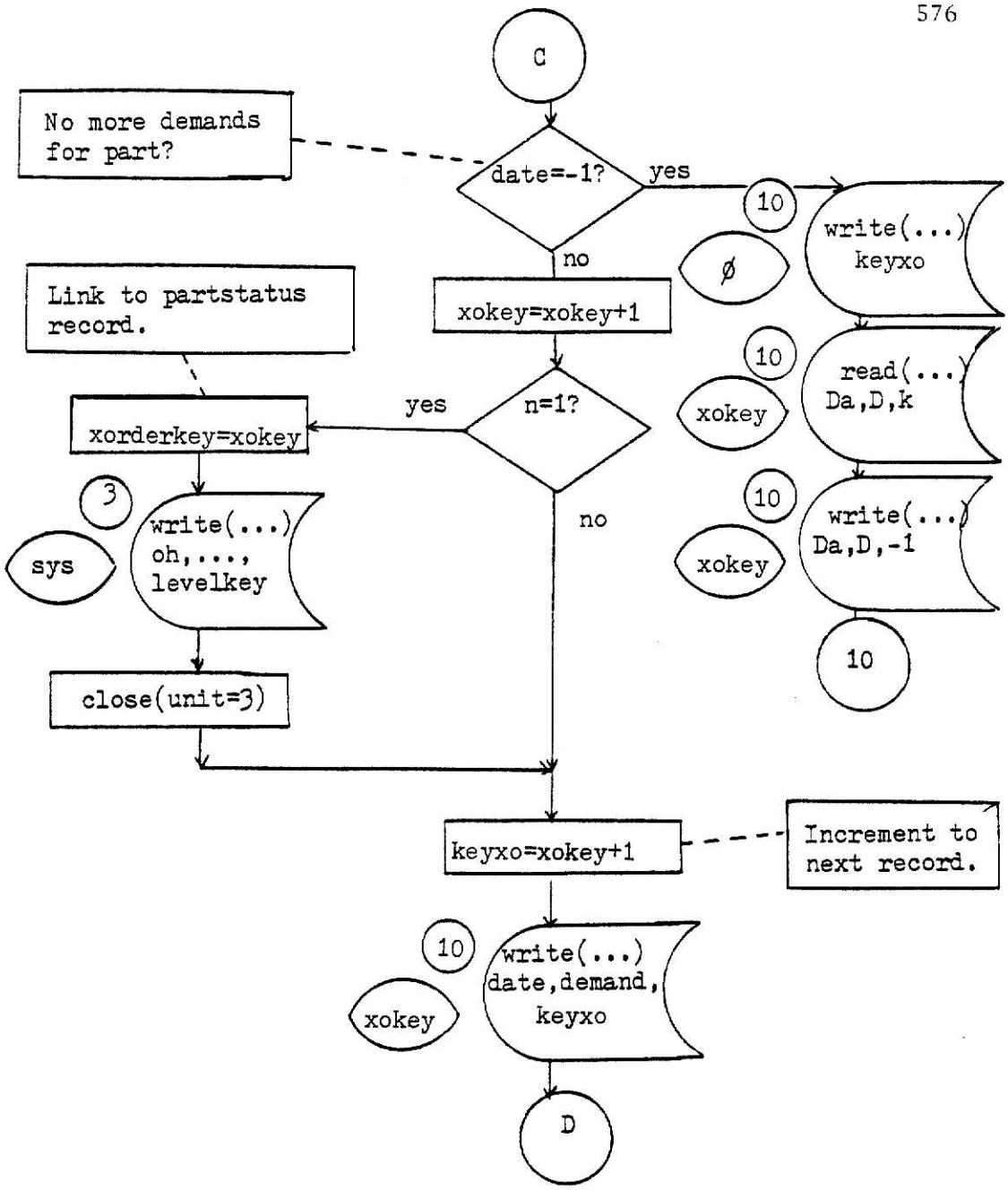


DataPlus2

-Subroutine XOnew-

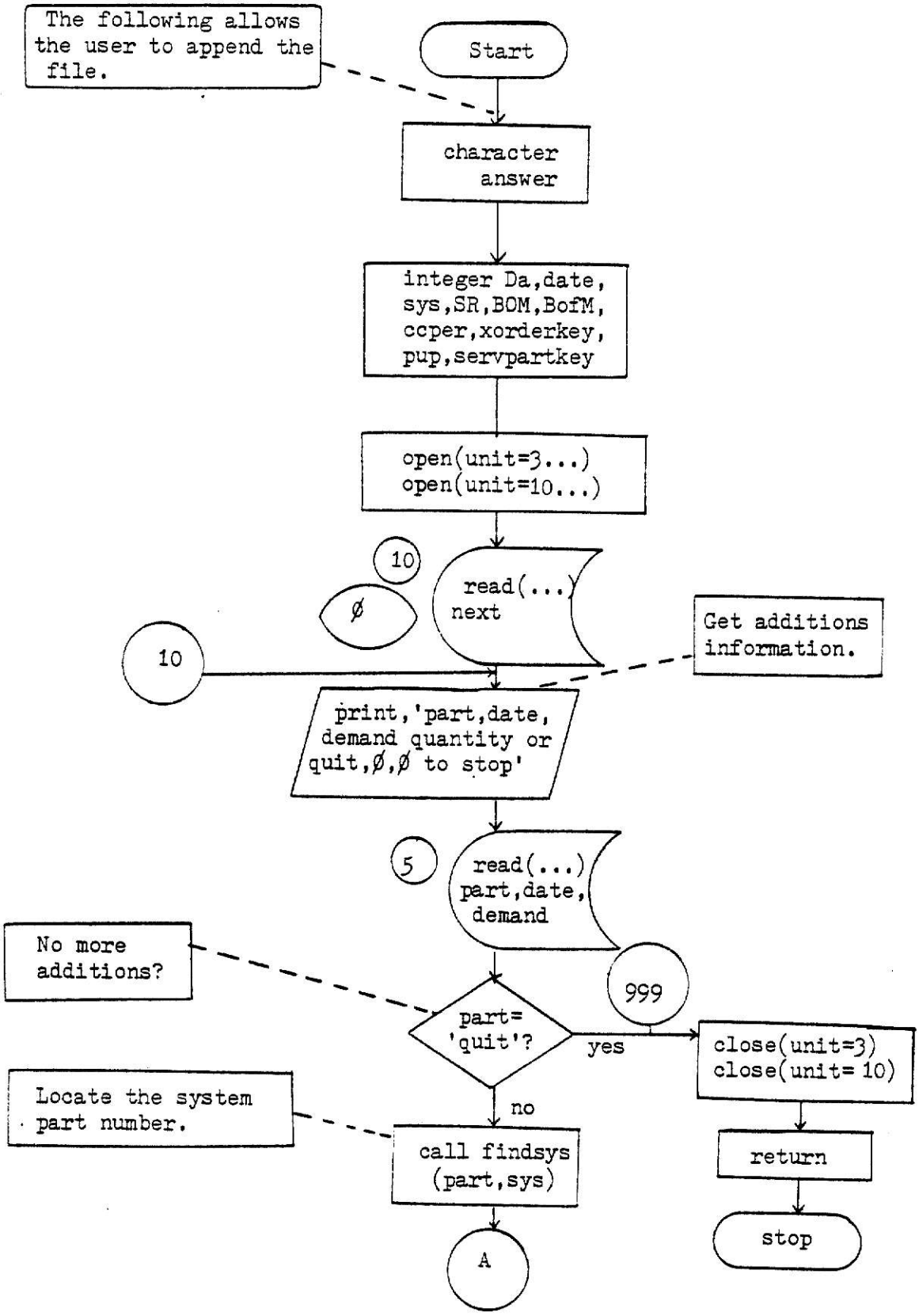


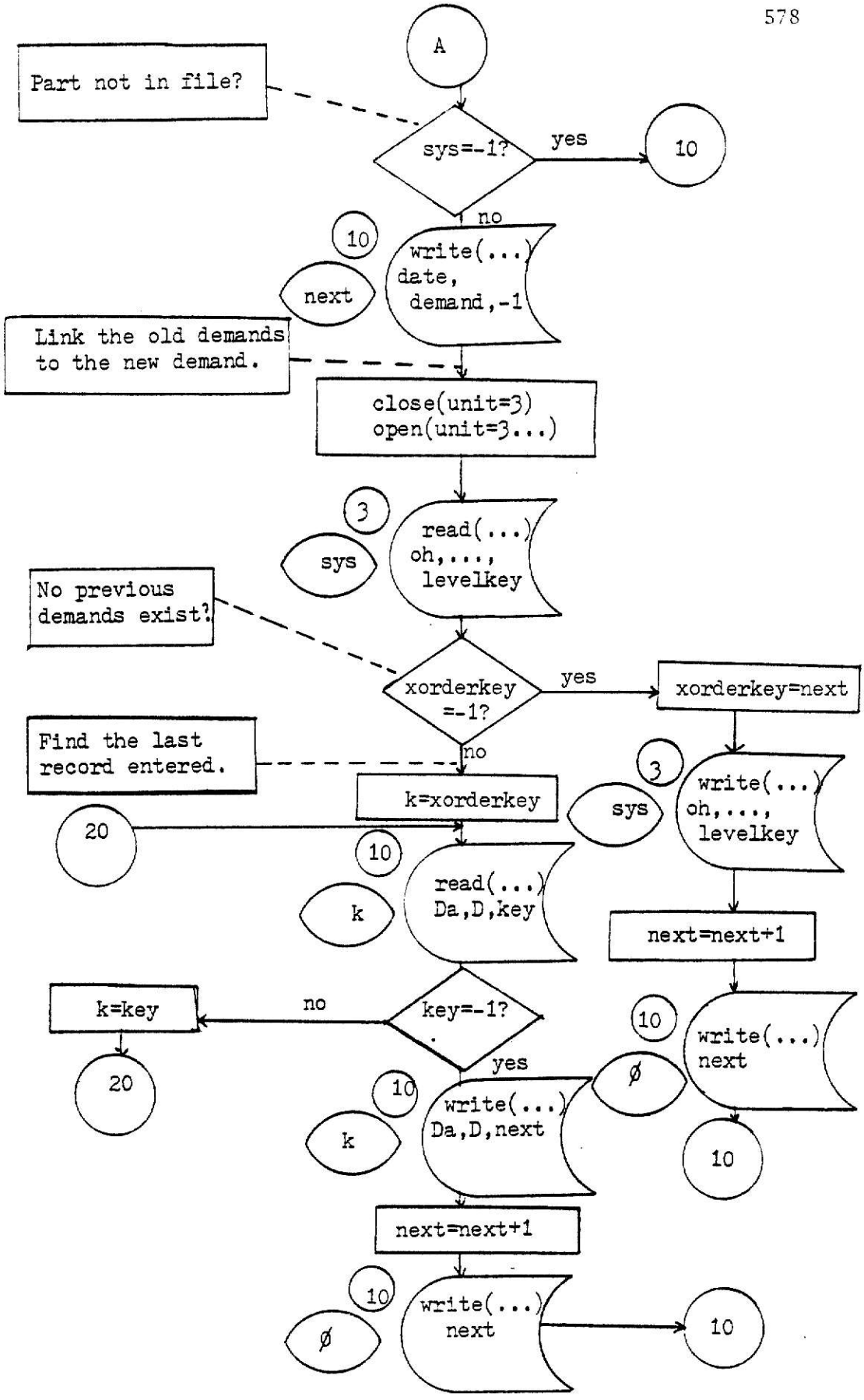

B
read,
part
No more entries?
part=
'quit'?
999
yes
close(unit=10)
return
stop
Locate the system
part number.
no
call findsys
(part,sys)
sys=-1?
yes
10
no
open(unit=3...)
Get the scoop
on the part.
3
read(...)
oh,...,
levelkey
sys
set the loop
counter
n=0
D
n=n+1
Get the specifics
about the demand.
print,'date,
demand or -1,0
to stop'
5
read(...)
date,
demand
C


C
No more demands for part?
date=-1?
yes
no
10
write(...) keyxo
Ø
xokey=xokey+1
Link to partstatus record.
10
read(... Da,D,k
xokey
n=1?
yes
xorderkey=xokey
10
write(... Da,D,-1
xokey
3
write(...) oh,..., levelkey
sys
10
close(unit=3)
keyxo=xokey+1
Increment to next record.
10
write(...) date,demand, keyxo
xokey
D

DataPlus2

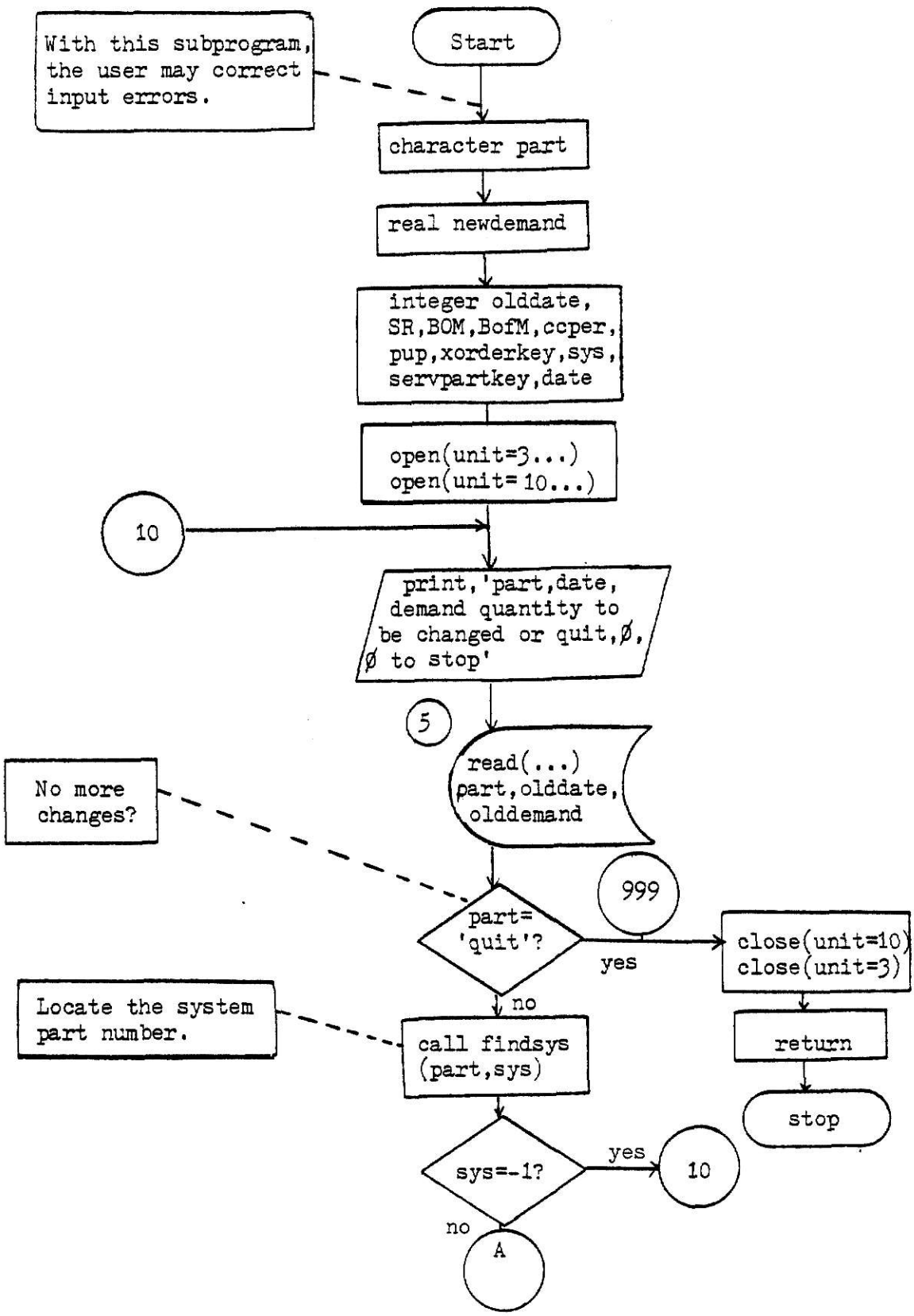
-Subroutine XOadd-

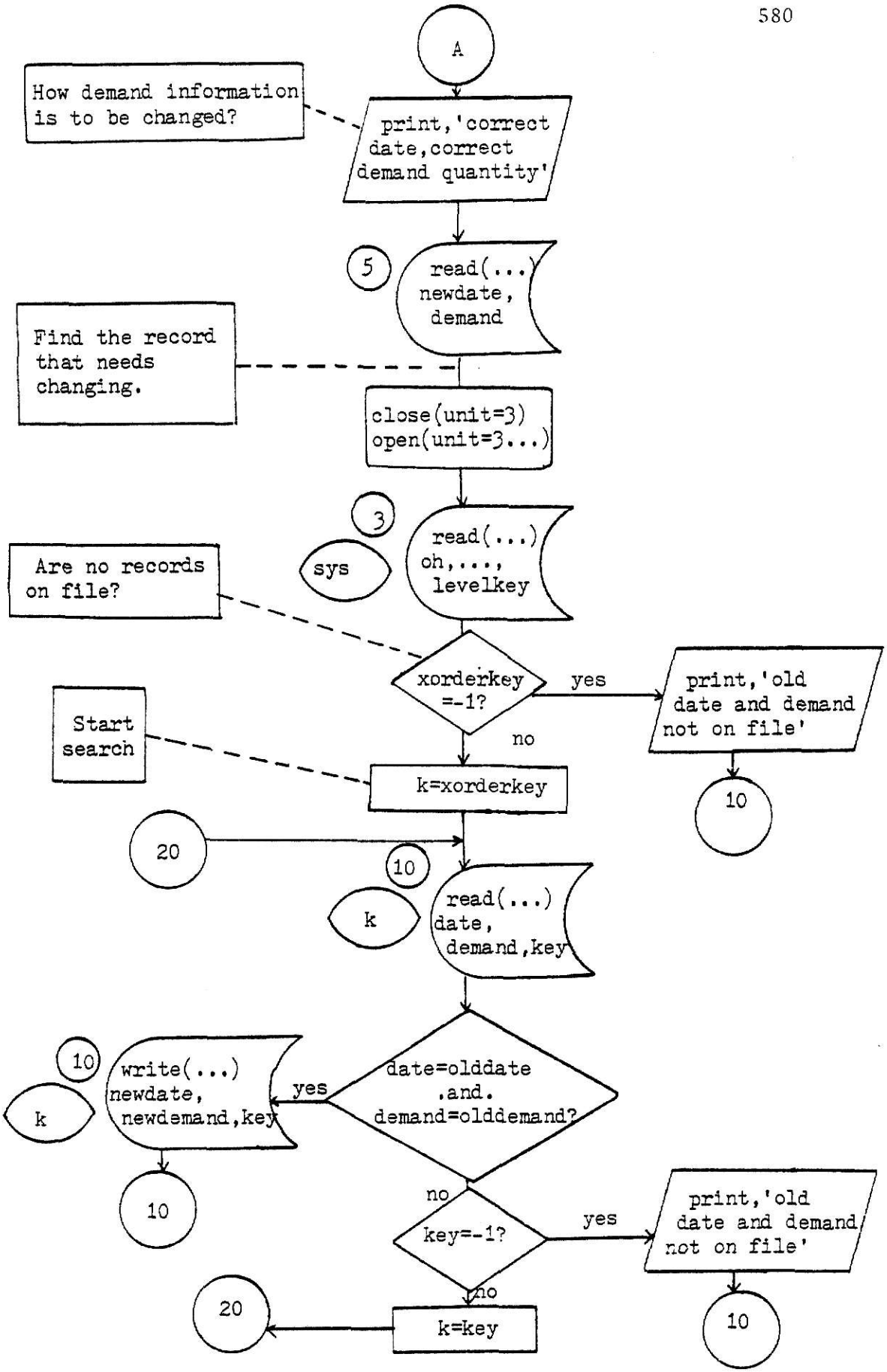



A
Part not in file?
sys=-1?
yes
10
no
10
next
write(...)
date,
demand,-1
Link the old demands to the new demand.
close(unit=3)
open(unit=3...)
3
sys
read(...)
oh,...,
levelkey
No previous demands exist?
xorderkey =-1?
yes
xorderkey=next
Find the last record entered.
no
k=xorderkey
3
sys
write(...)
oh,...,
levelkey
20
10
k
read(...)
Da,D,key
next=next+1
k=key
no
key=-1?
10
write(...)
next
20
yes
10
k
write(...)
Da,D,next
10
next=next+1
10
write(...)
next
10

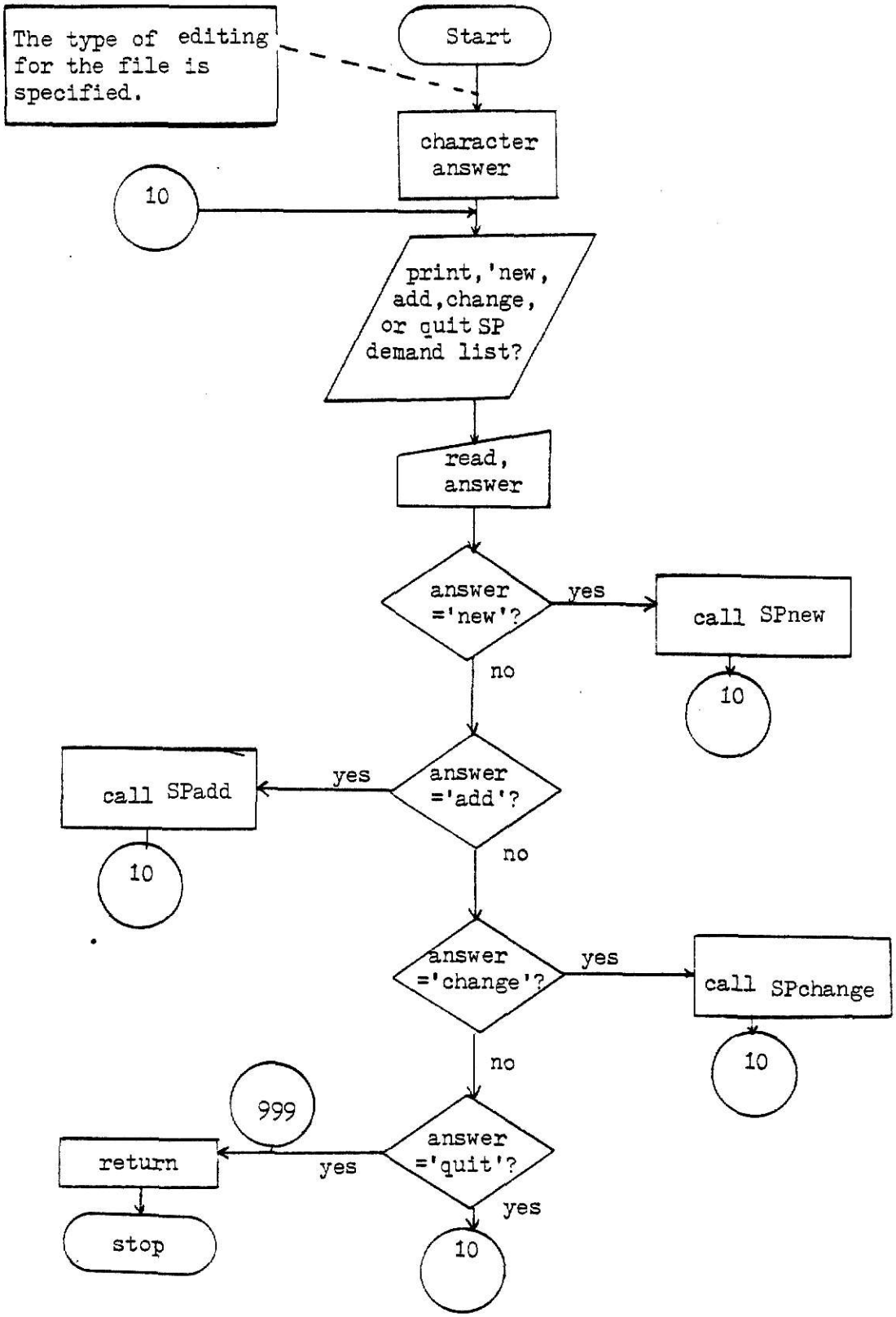
DataPlus2

-Subroutine XOchange-



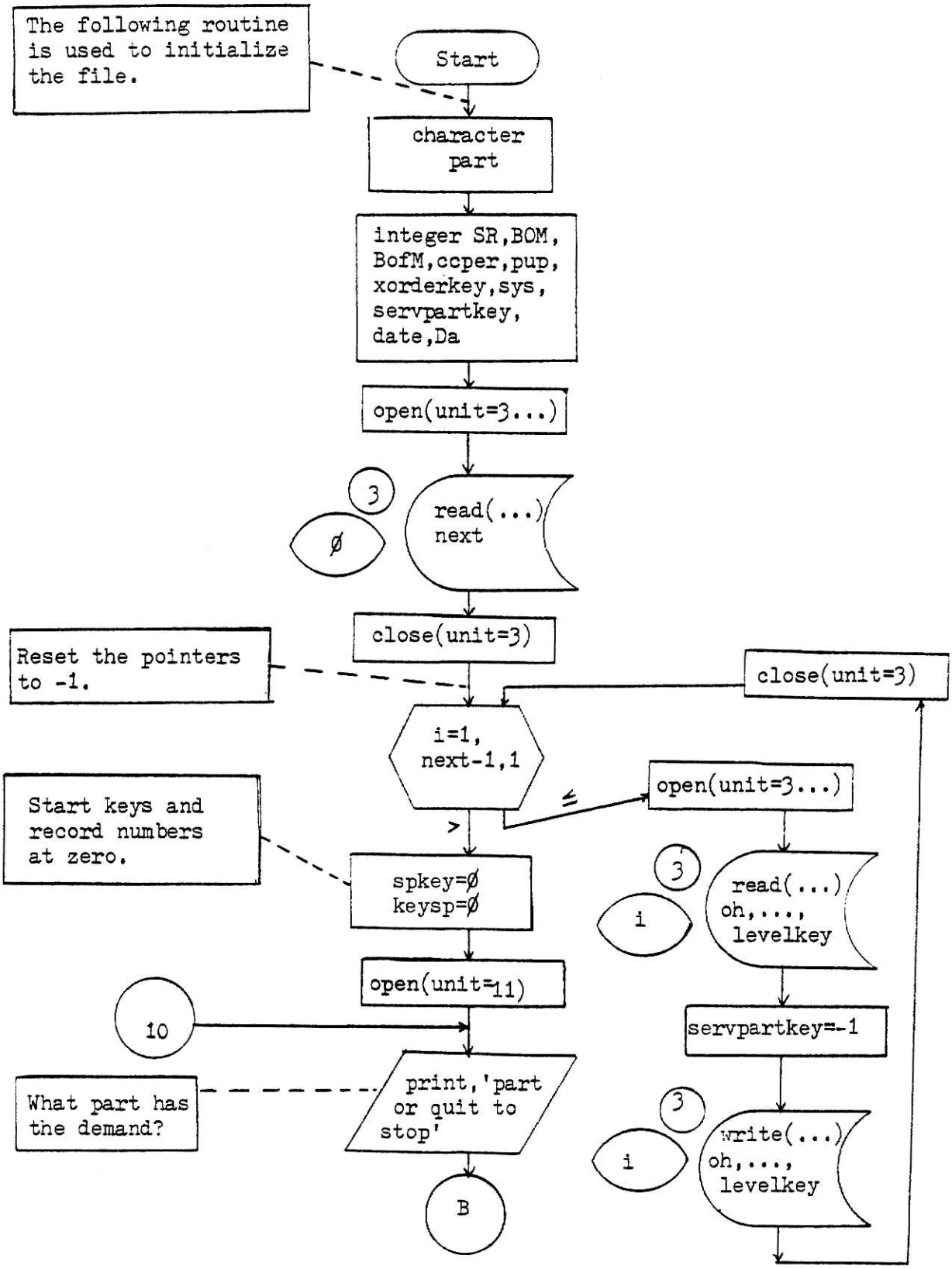

A
How demand information is to be changed?
print,'correct date,correct demand quantity'
5
read(... newdate, demand
Find the record that needs changing.
close(unit=3)
open(unit=3...)
3
sys
read(...) oh,..., levelkey
Are no records on file?
xorderkey =-1?
yes
print,'old date and demand not on file'
10
no
Start search
k=xorderkey
20
10
k
read(...) date, demand,key
date=olddate .and. demand=olddemand?
yes
10
k
write(...) newdate, newdemand,key
10
no
key=-1?
yes
print,'old date and demand not on file'
10
no
20
k=key

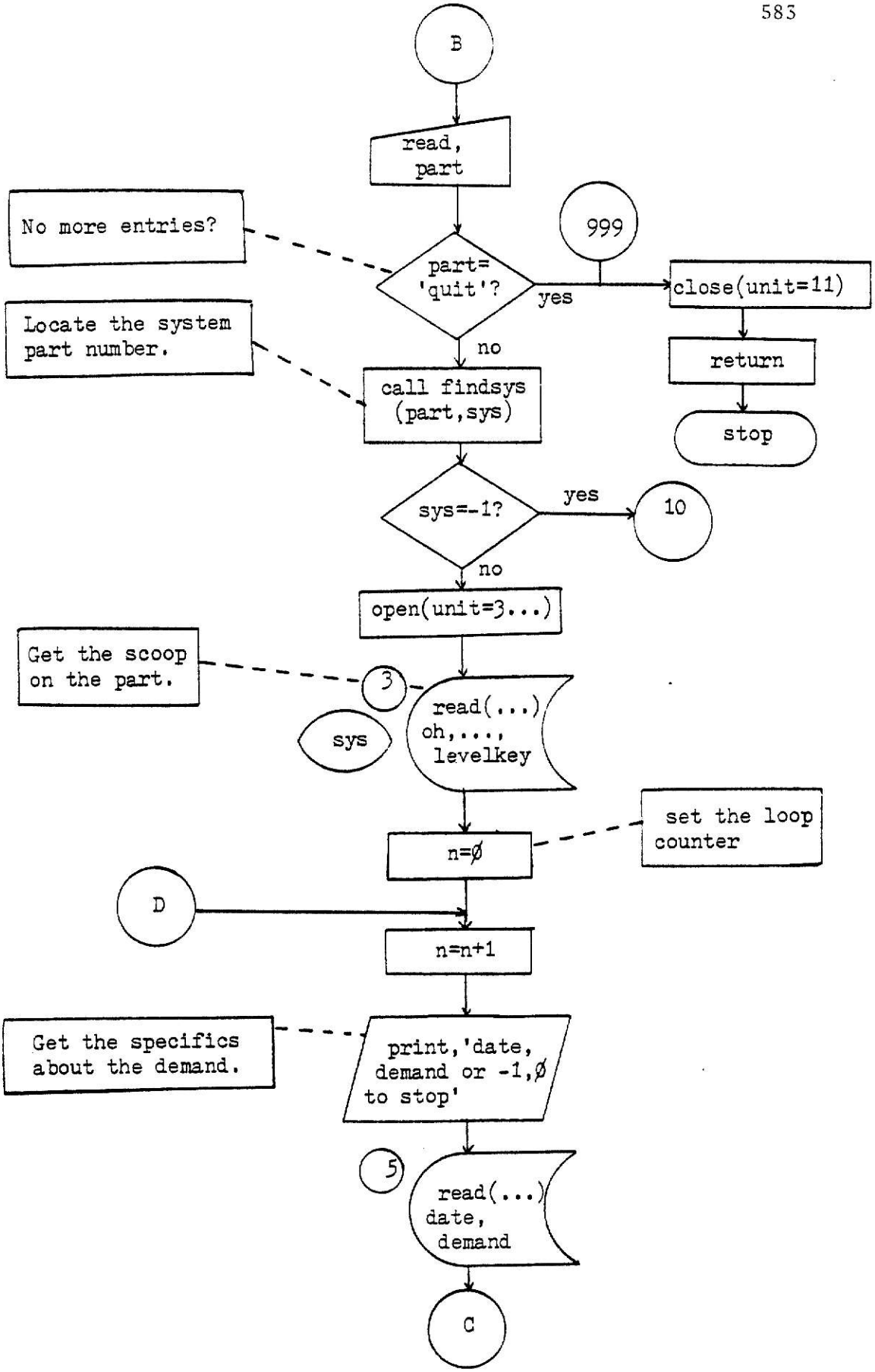
DataPlus2
-Subroutine SPdemands-.

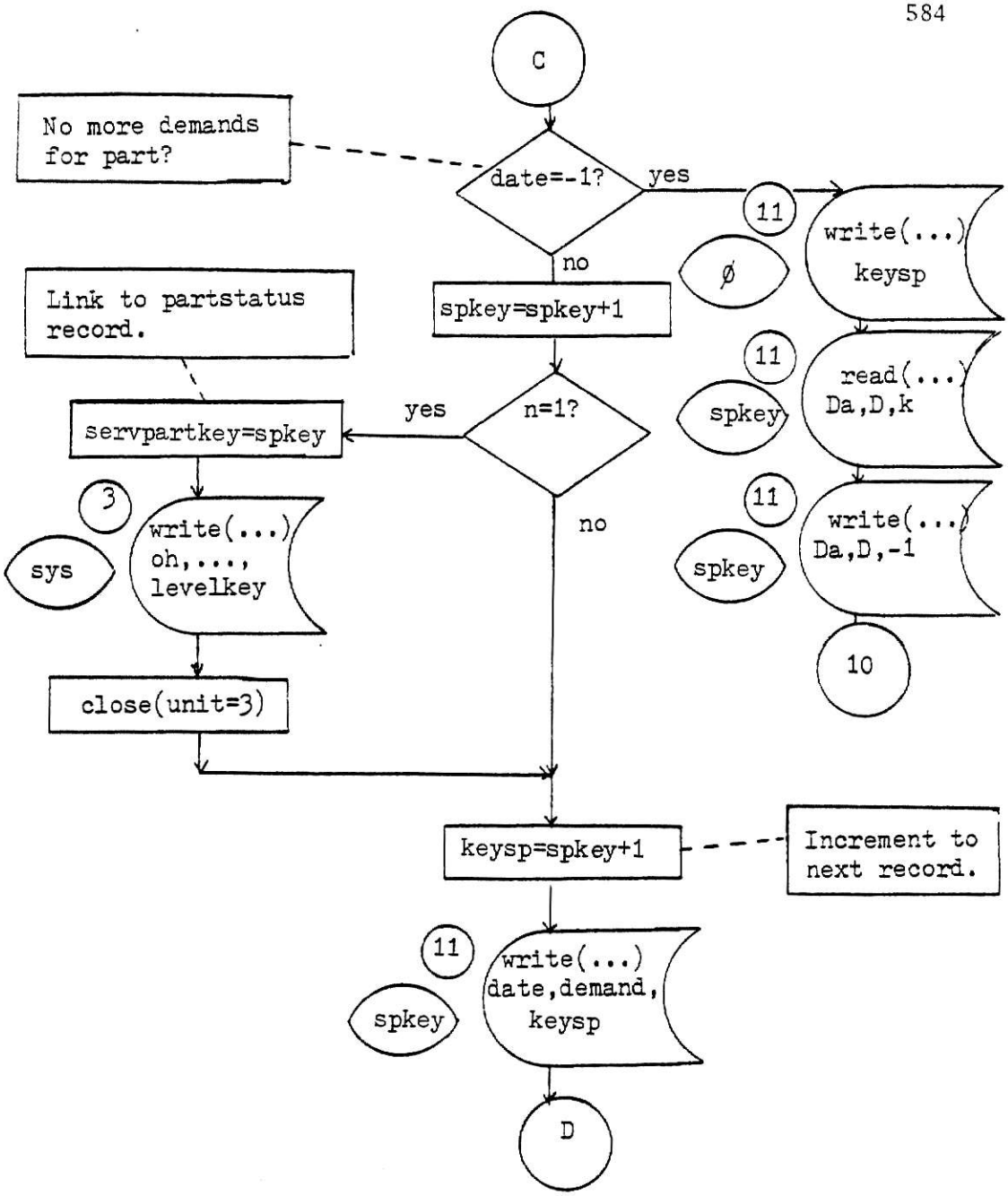


DataPlus2

-Subroutine SPnew-

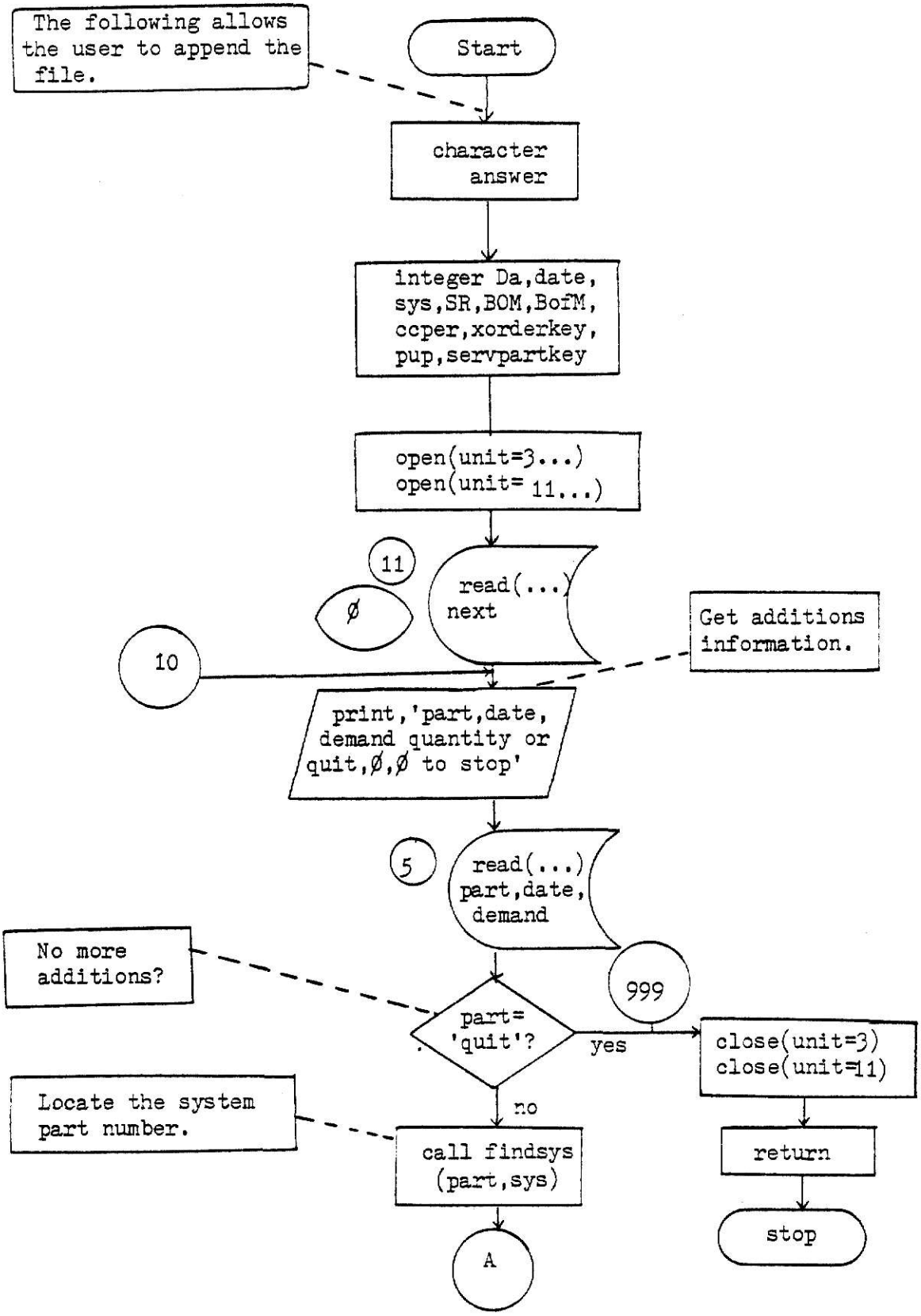


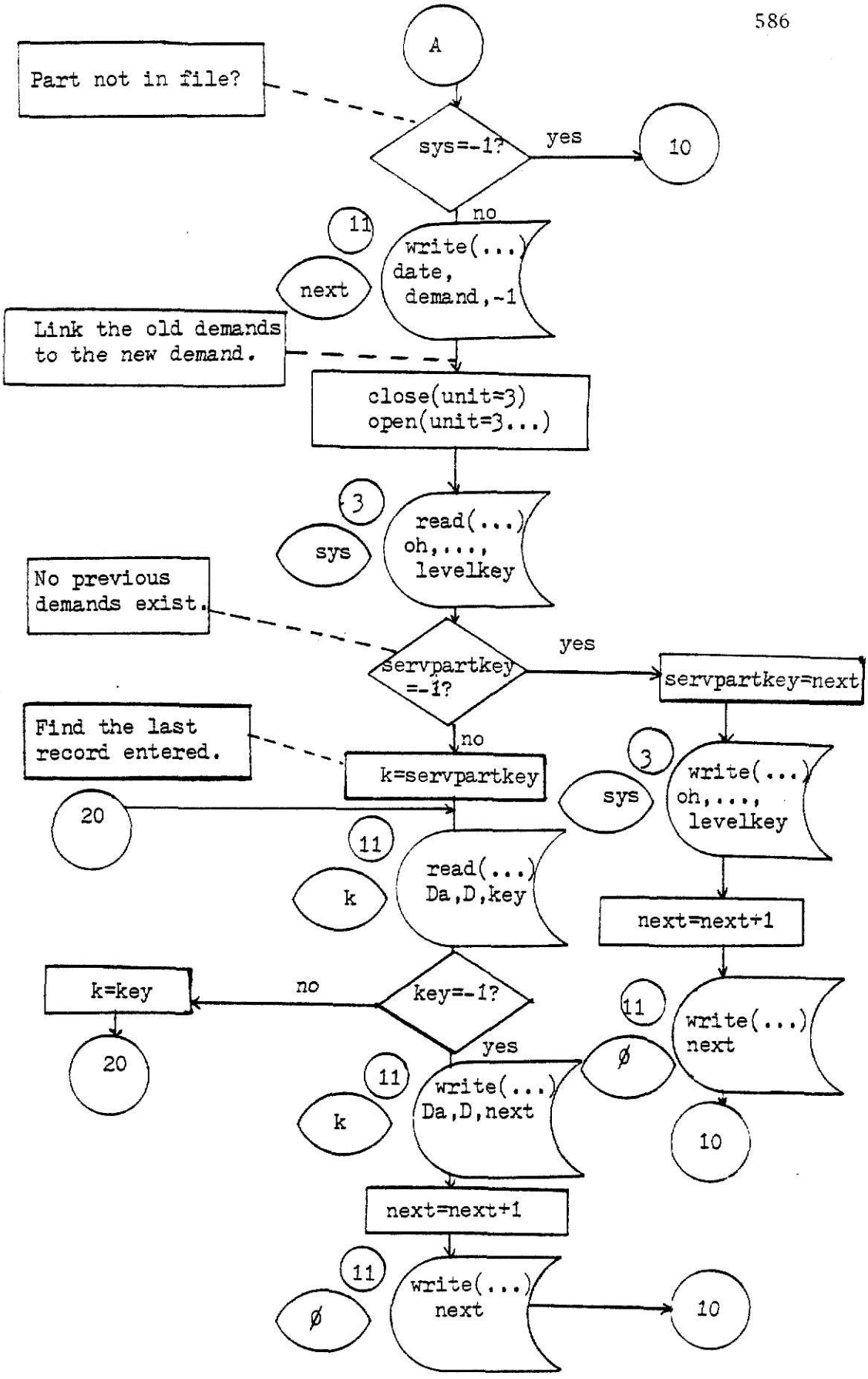

B
read,
part
No more entries?
part=
'quit'?
999
yes
close(unit=11)
return
stop
Locate the system
part number.
call findsys
(part,sys)
no
sys=-1?
yes
10
no
open(unit=3...)
Get the scoop
on the part.
3
read(...)
oh,...,
levelkey
sys
set the loop
counter
n=ø
D
n=n+1
Get the specifics
about the demand.
print,'date,
demand or -1,ø
to stop'
5
read(...)
date,
demand
C


C
No more demands for part?
date=-1?
yes
no
11
ø
write(...) keysp
spkey=spkey+1
Link to partstatus record.
n=1?
yes
no
11
spkey
read(...) Da,D,k
servpartkey=spkey
3
sys
write(...) oh,...,levelkey
11
spkey
write(...) Da,D,-1
10
close(unit=3)
keysp=spkey+1
Increment to next record.
11
spkey
write(...) date,demand, keysp
D

DataPlus2

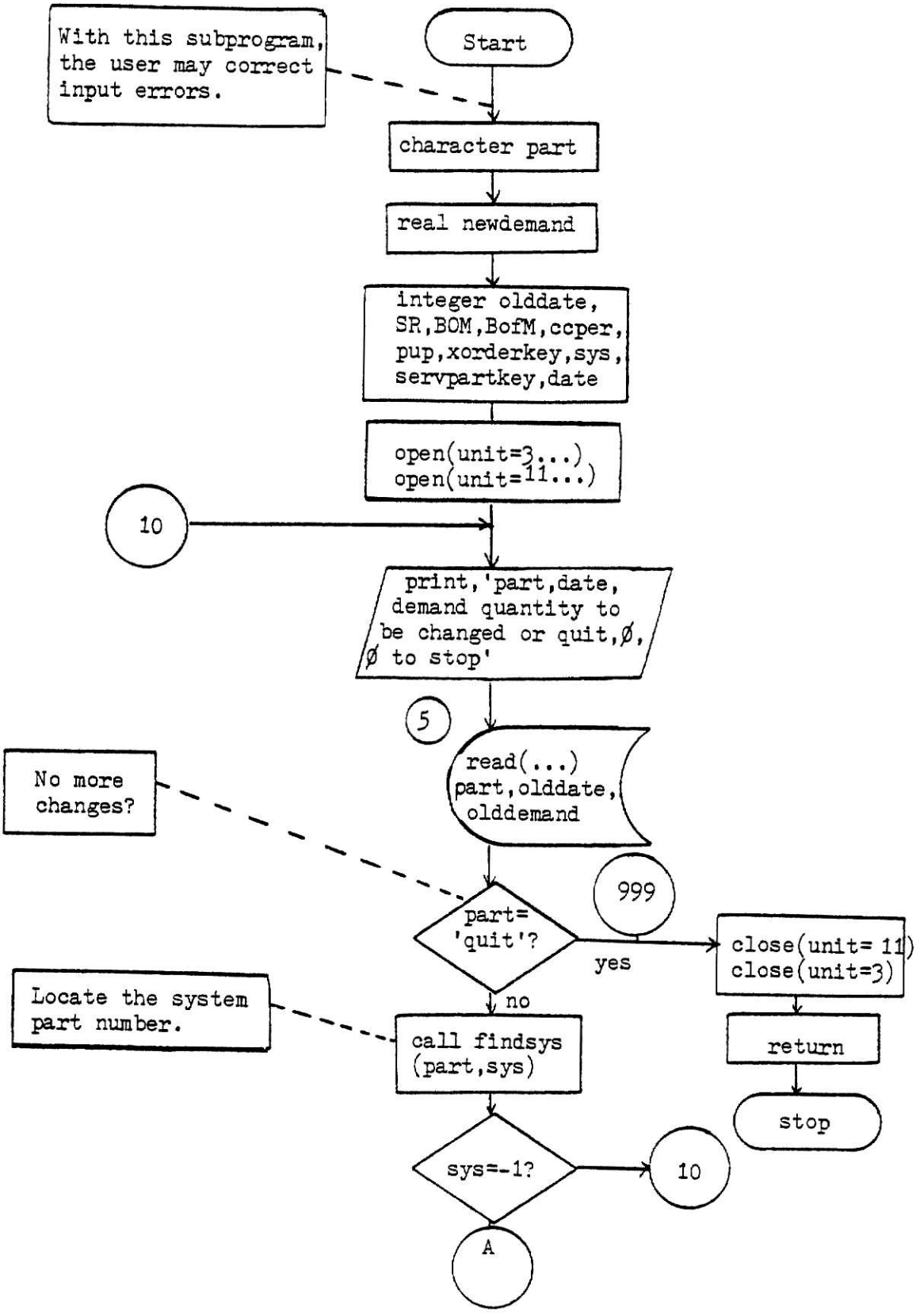
-Subroutine SPadd-

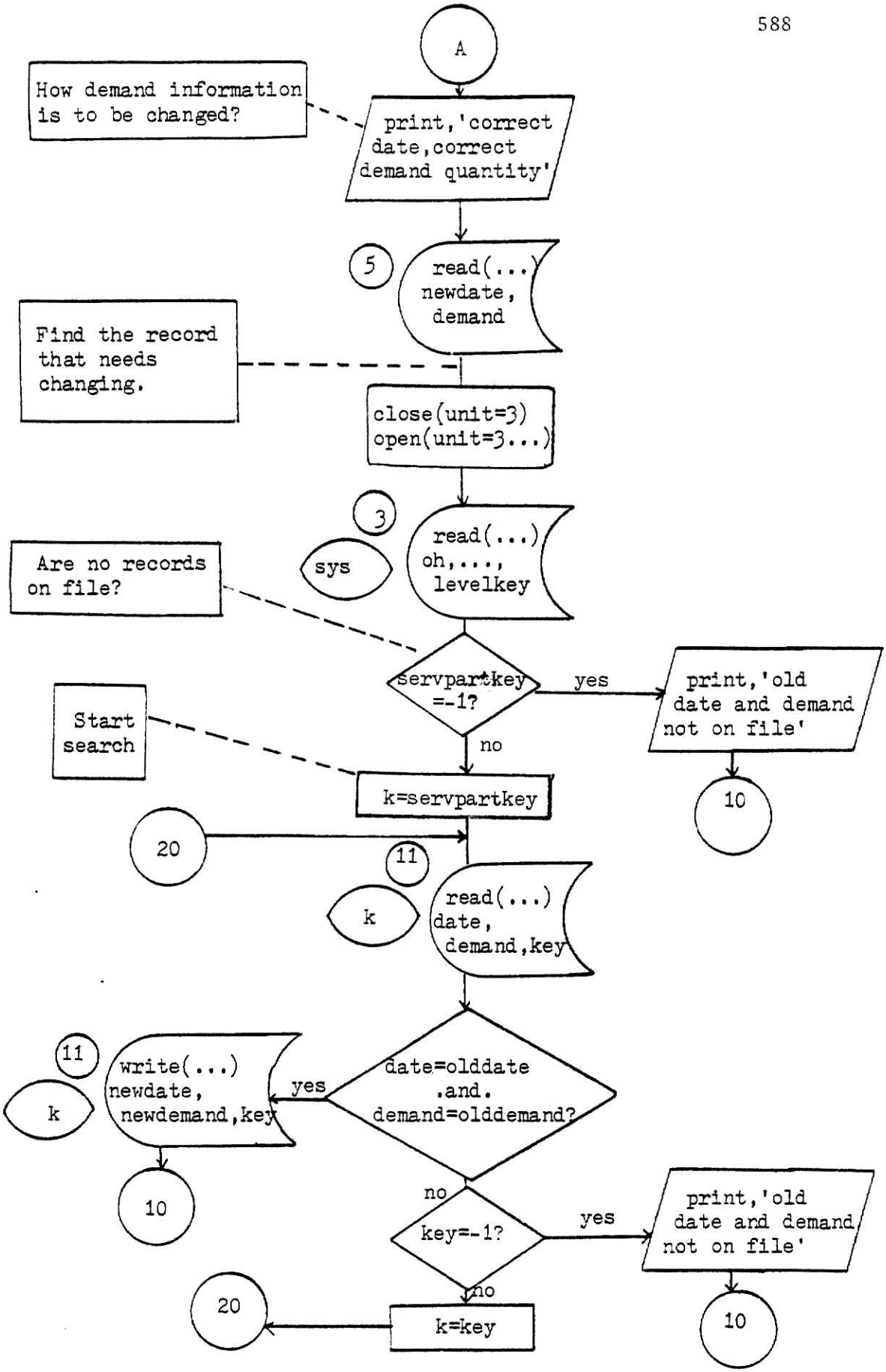



A
Part not in file?
sys=-1?
yes
10
no
11
next
write(...)
date,
demand,-1
Link the old demands to the new demand.
close(unit=3)
open(unit=3...)
3
sys
read(...)
oh,...,
levelkey
No previous demands exist.
servpartkey =-1?
yes
servpartkey=next
Find the last record entered.
no
k=servpartkey
20
3
sys
write(...)
oh,...,
levelkey
11
k
read(...)
Da,D,key
next=next+1
k=key
no
key=-1?
11
write(...)
next
20
yes
11
k
write(...)
Da,D,next
10
next=next+1
11
write(...)
next
10

DataPlus2

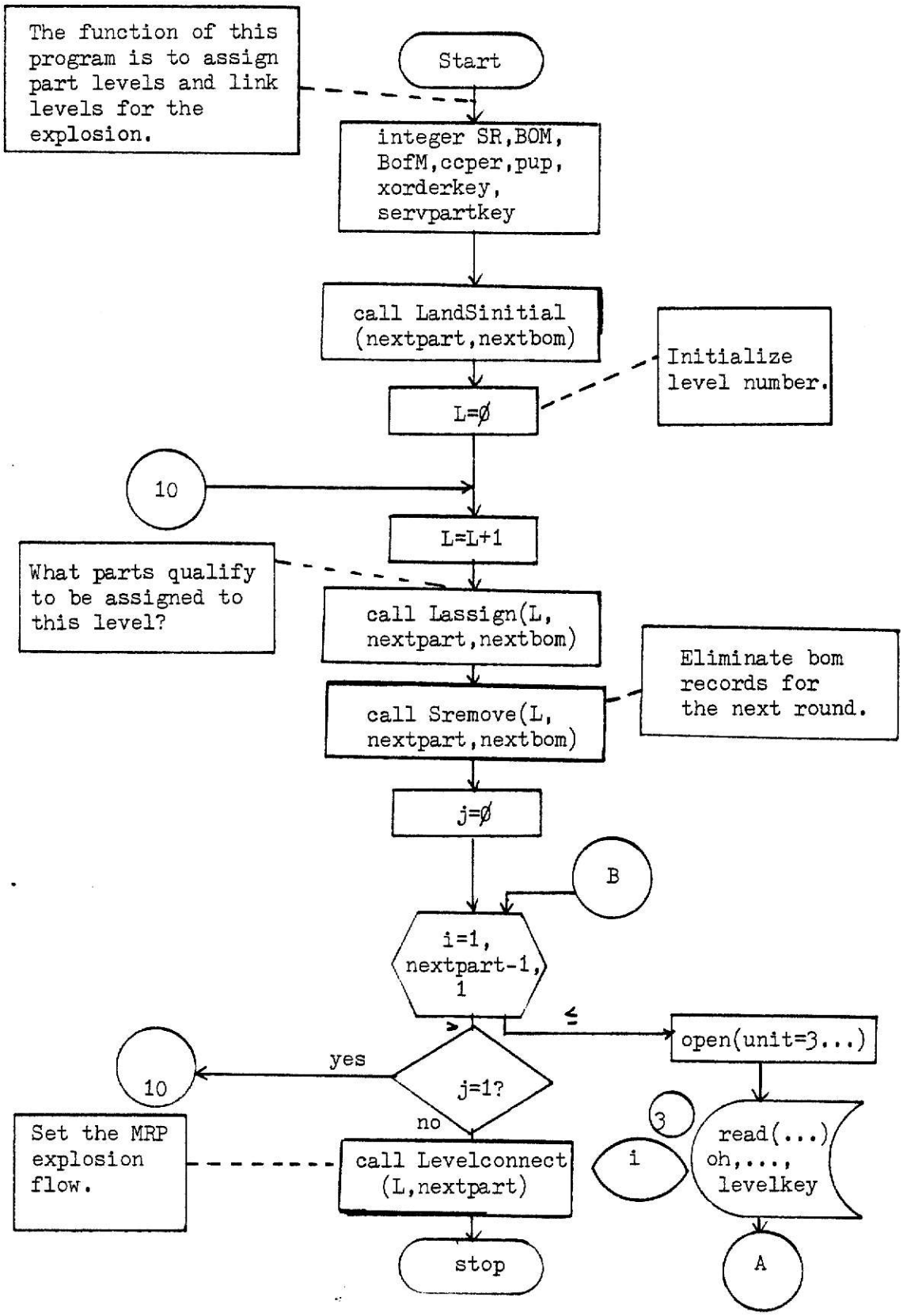
-Subroutine SPchange-

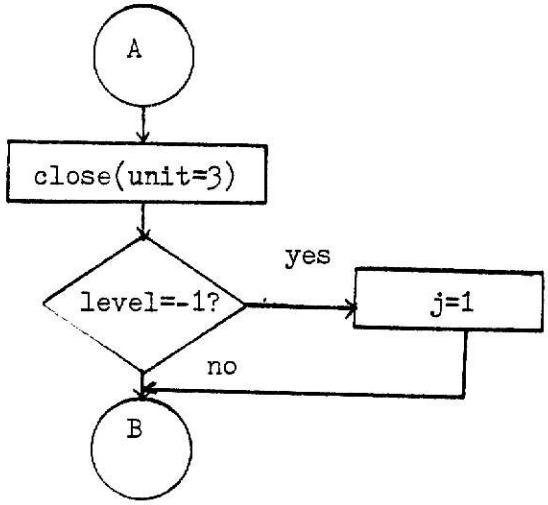


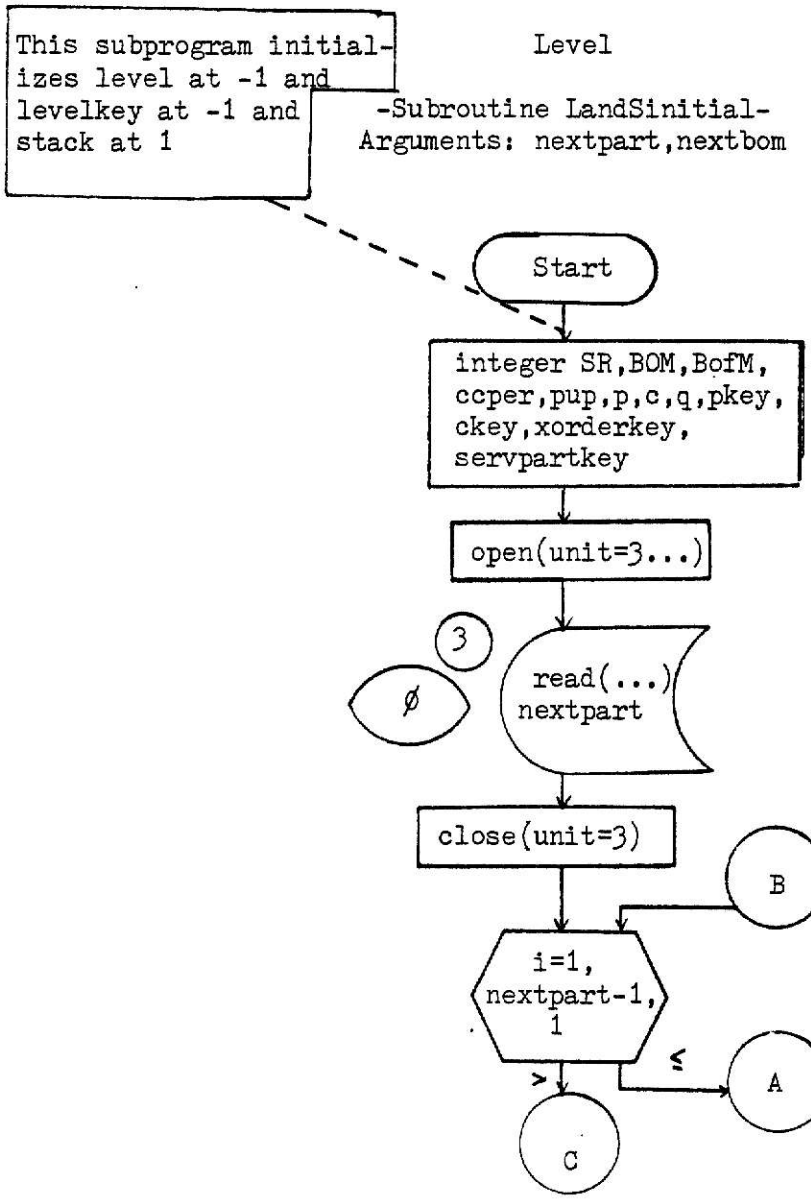

A
How demand information is to be changed?
print,'correct date,correct demand quantity'
5
read(... newdate, demand
Find the record that needs changing.
close(unit=3)
open(unit=3...)
3
sys
read(...) oh,..., levelkey
Are no records on file?
servpartkey =-1?
yes
print,'old date and demand not on file'
10
Start search
no
k=servpartkey
20
11
k
read(...) date, demand,key
date=olddate .and. demand=olddemand?
yes
11
k
write(...) newdate, newdemand,key
10
no
key=-1?
yes
print,'old date and demand not on file'
10
no
k=key
20

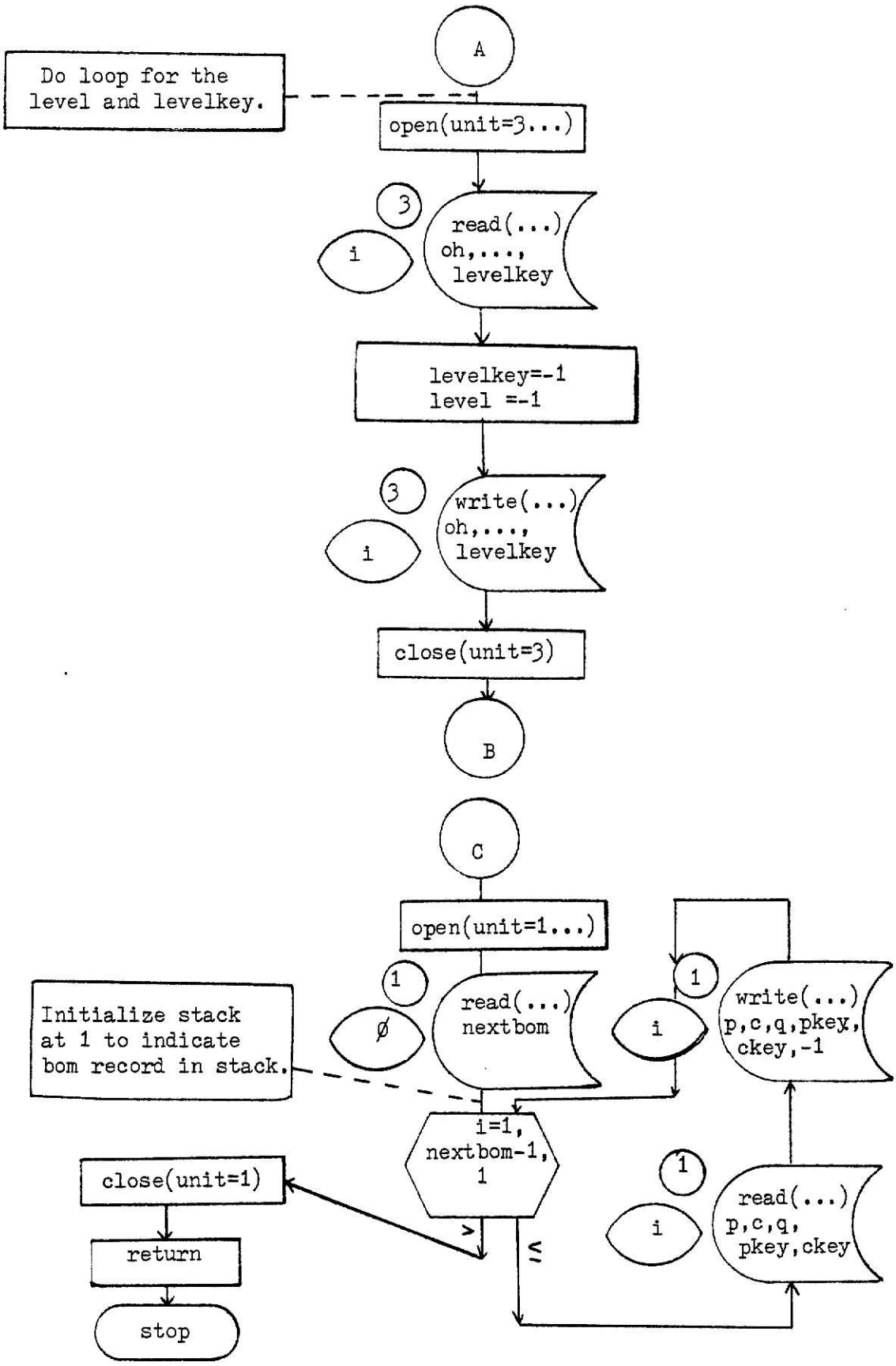
Level

-Main Program-



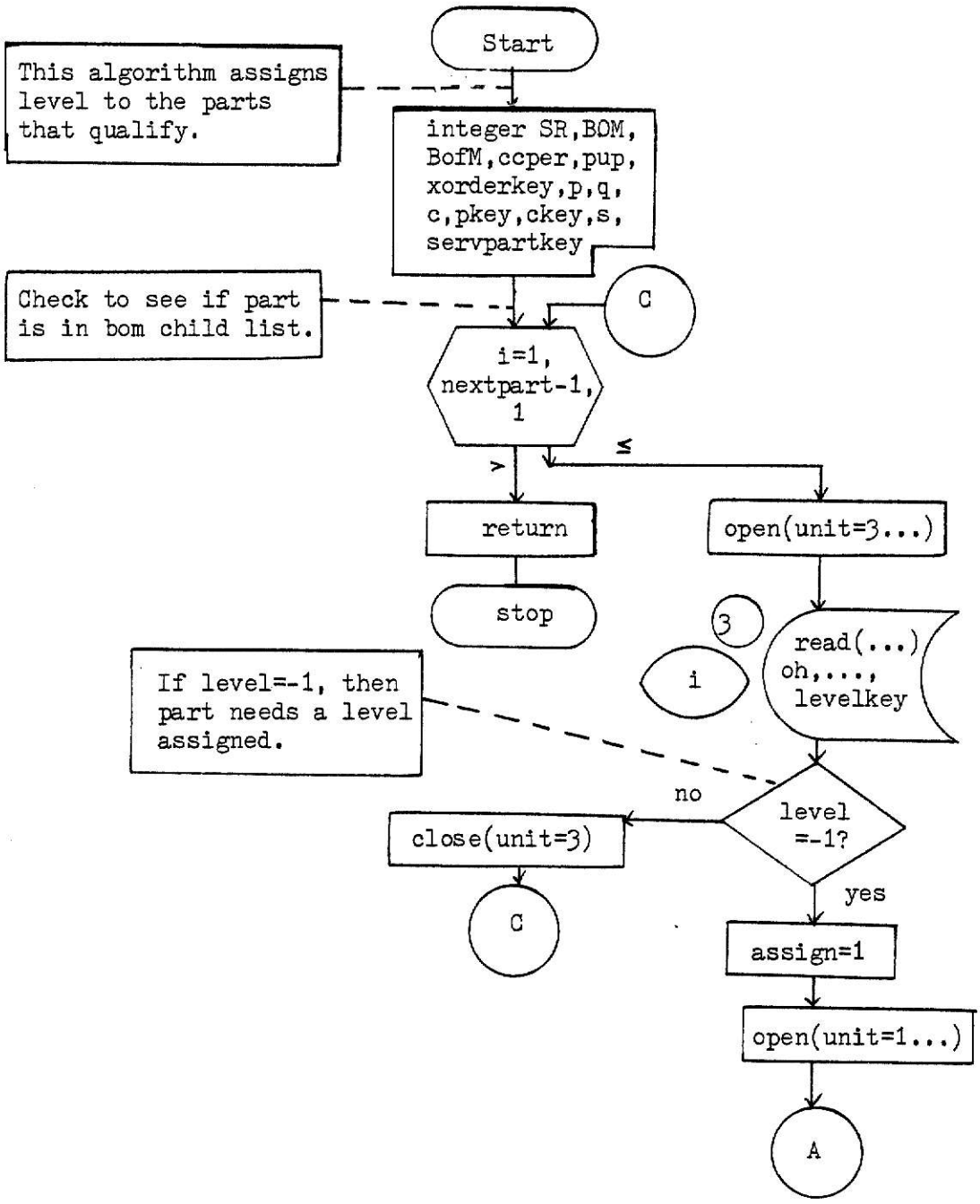

A
close(unit=3)
level=-1?
yes
j=1
no
B

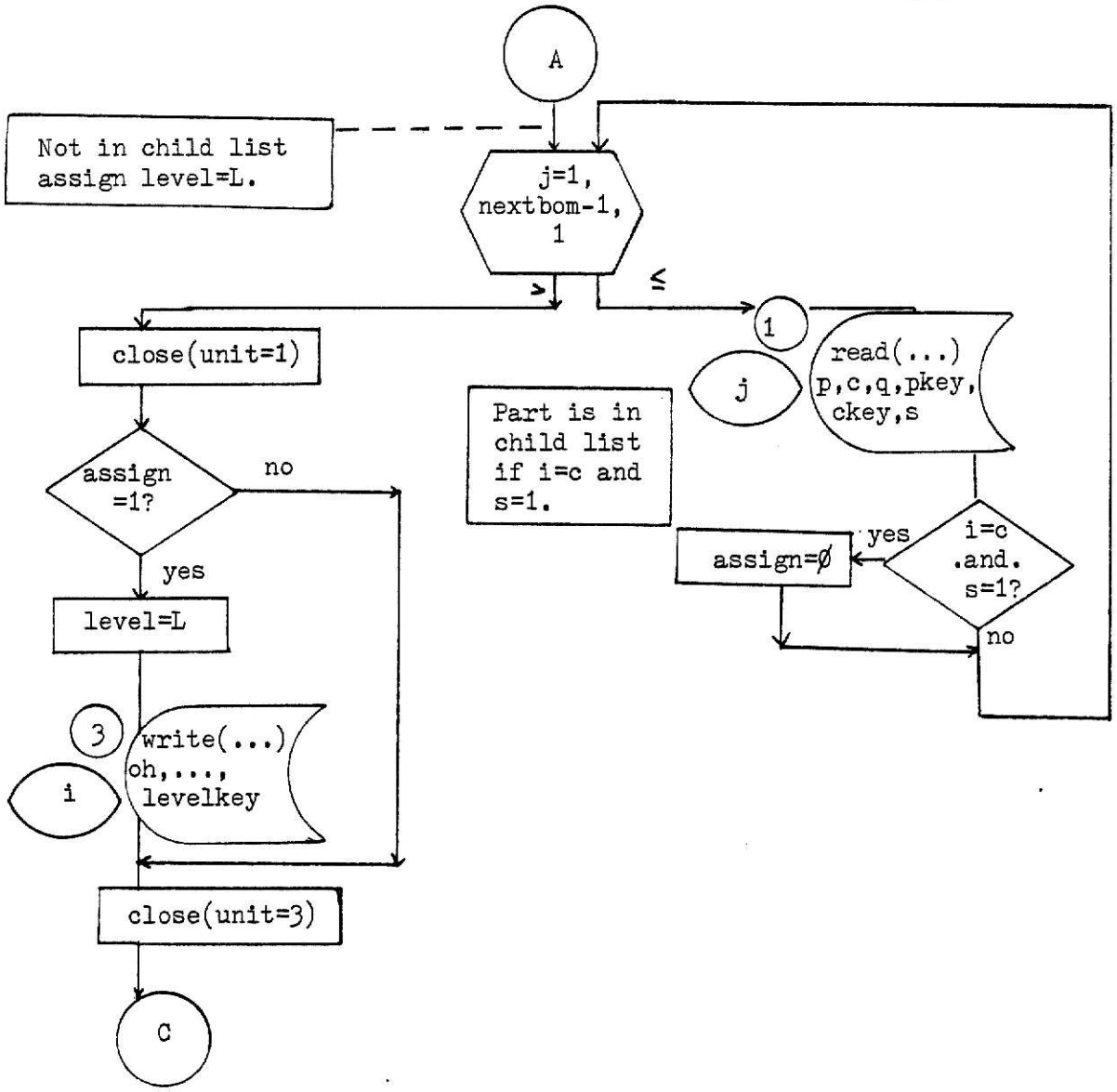

This subprogram initializes level at -1 and levelkey at -1 and stack at 1
Level
-Subroutine LandSinitial-
Arguments: nextpart,nextbom
Start
integer SR,BOM,BofM, ccper,pup,p,c,q,pkey, ckey,xorderkey, servpartkey
open(unit=3...)
3
ø
read(...) nextpart
close(unit=3)
B
i=1, nextpart-1, 1
>
≤
A
C


A
Do loop for the level and levelkey.
open(unit=3...)
3
i
read(...)
oh,...,
levelkey
levelkey=-1
level =-1
3
i
write(...)
oh,...,
levelkey
close(unit=3)
B
C
open(unit=1...)
1
ø
read(...)
nextbom
Initialize stack at 1 to indicate bom record in stack.
1
i
write(...)
p,c,q,pkey,
ckey,-1
i=1,
nextbom-1,
1
>
≤
close(unit=1)
return
stop
1
i
read(...)
p,c,q,
pkey,ckey

Level

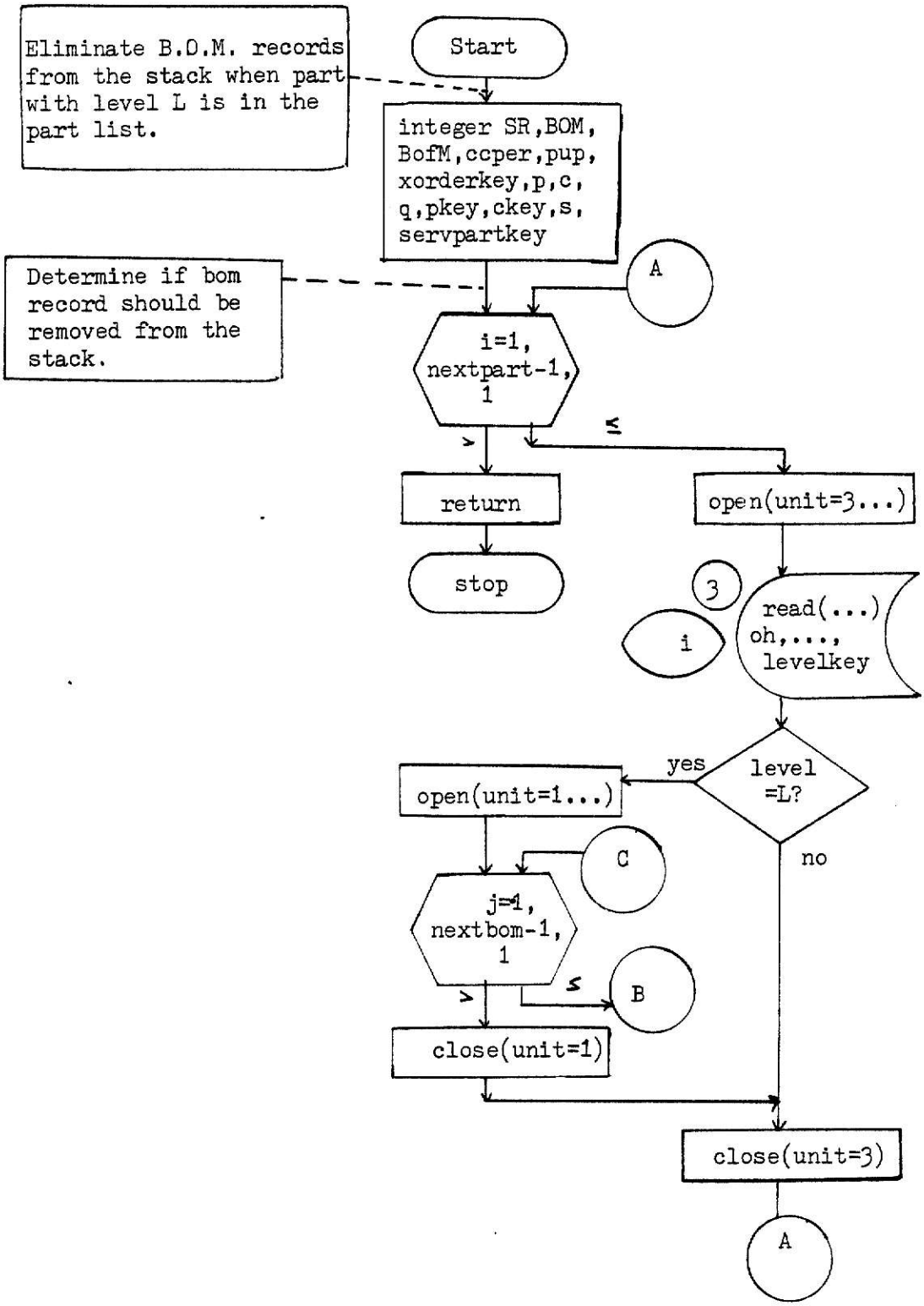
-Subroutine Lassign-
Arguments: L,nextpart,
nextbom

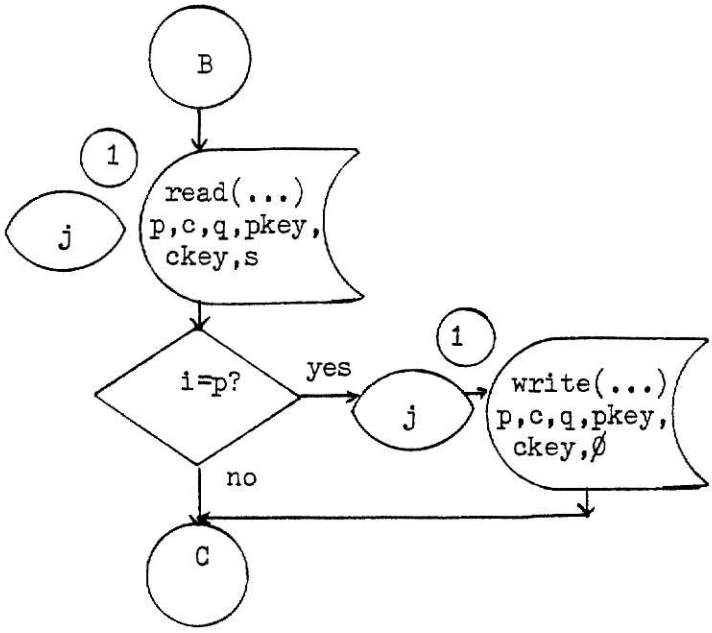



A
Not in child list
assign level=L.
j=1,
nextbom-1,
1
>
≤
close(unit=1)
assign
=1?
no
yes
level=L
3
write(...)
oh,...,
levelkey
i
close(unit=3)
C
1
j
read(...)
p,c,q,pkey,
ckey,s
Part is in
child list
if i=c and
s=1.
i=c
.and.
s=1?
yes
assign=ø
no

Level

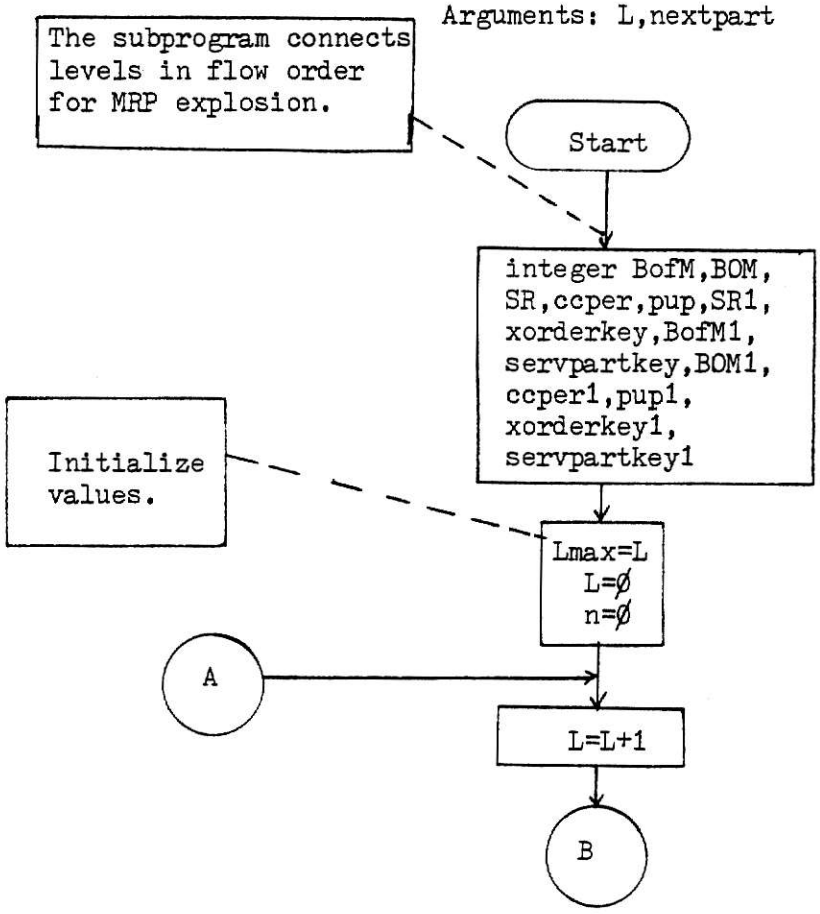
-Subroutine Sremove-
Arguments: L,nextpart,
nextbom

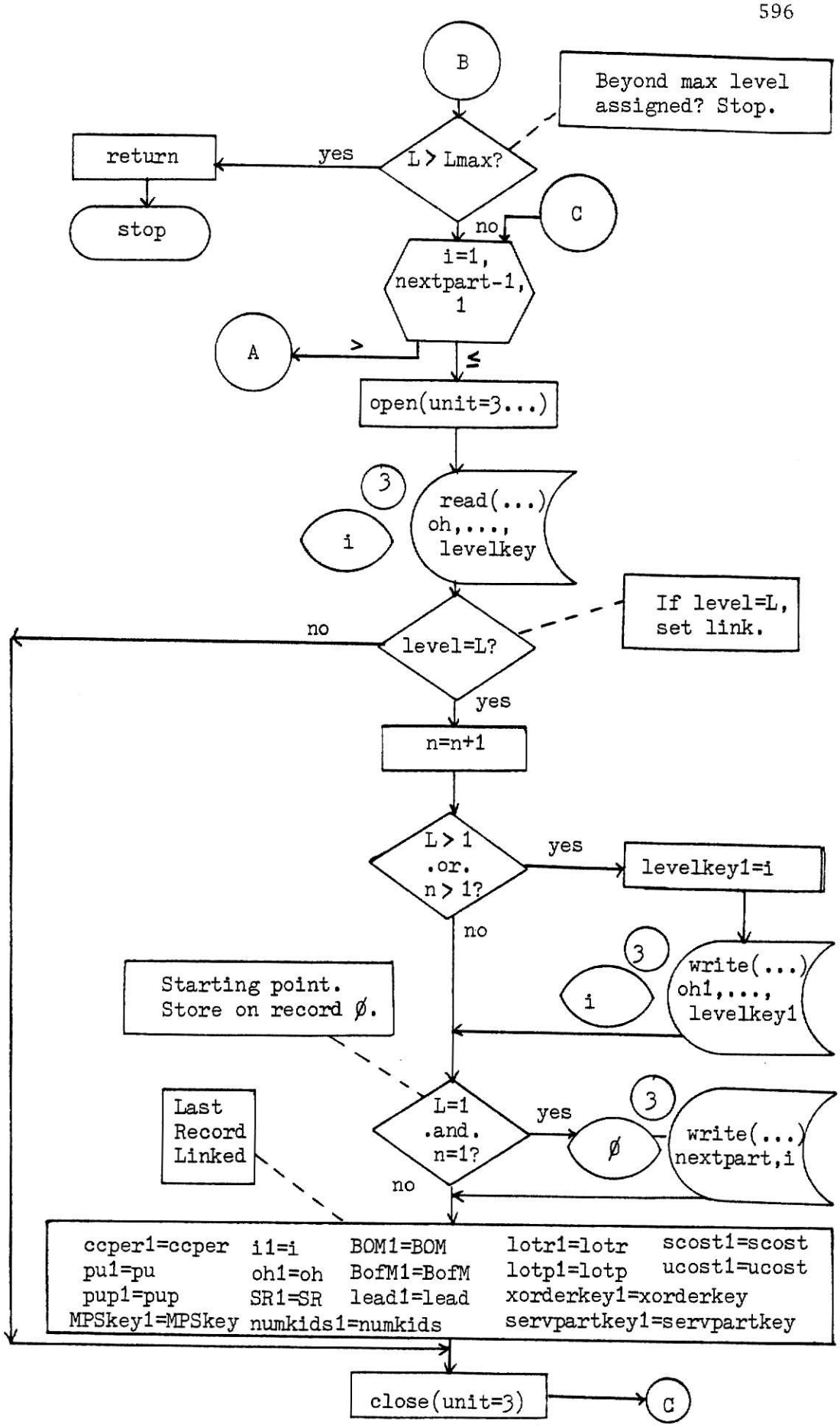




Level

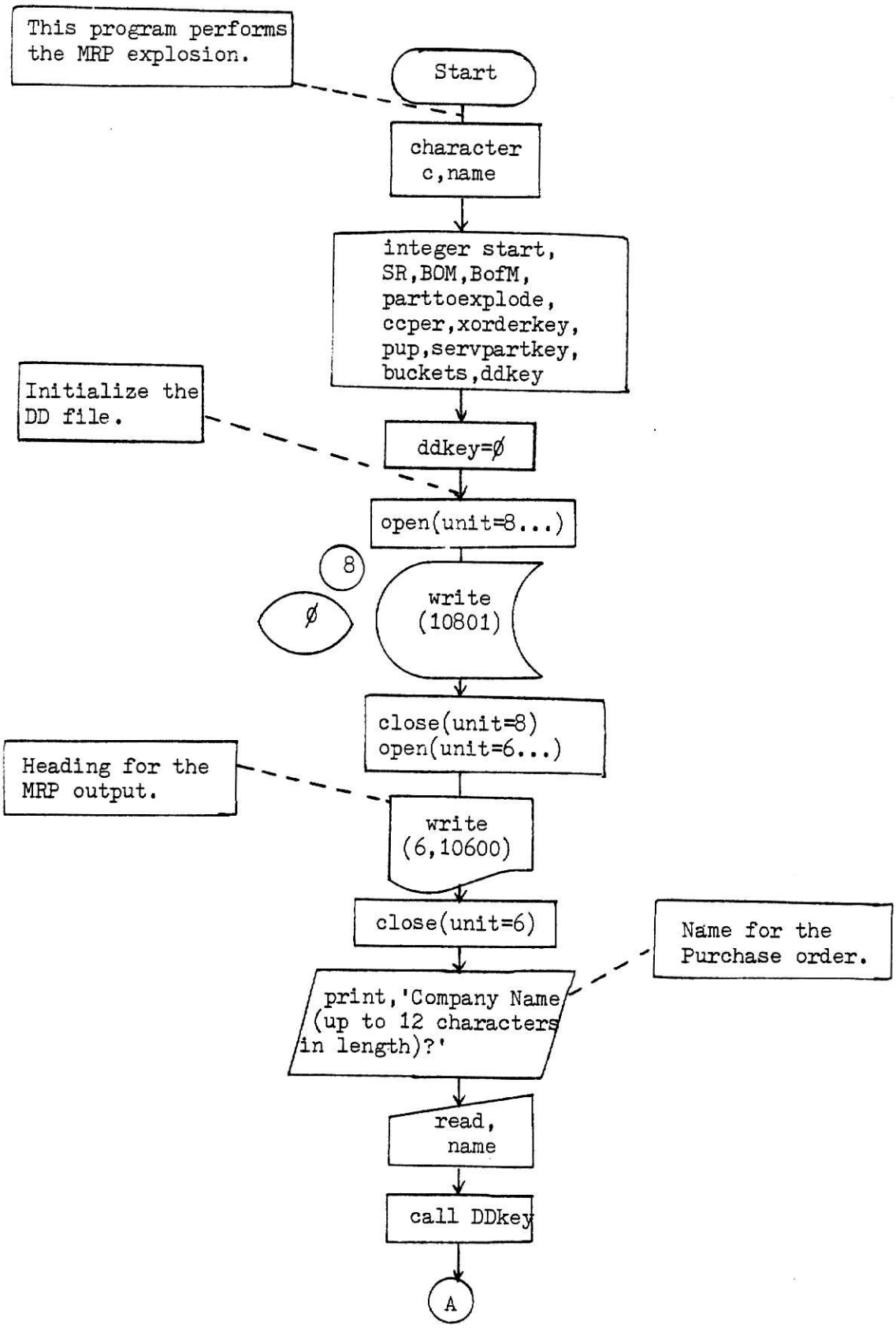
-Subroutine Levelconnect-
Arguments: L,nextpart

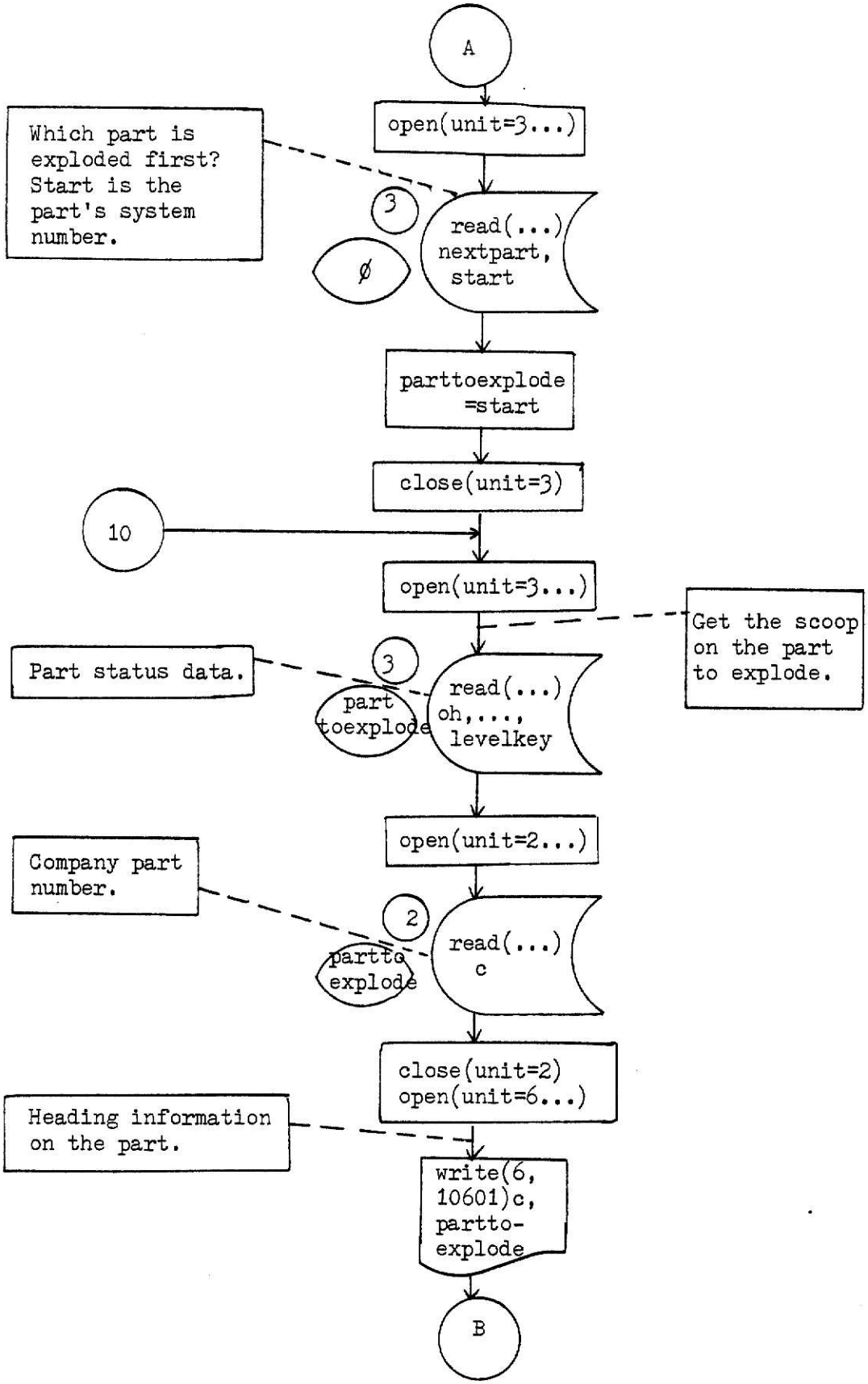


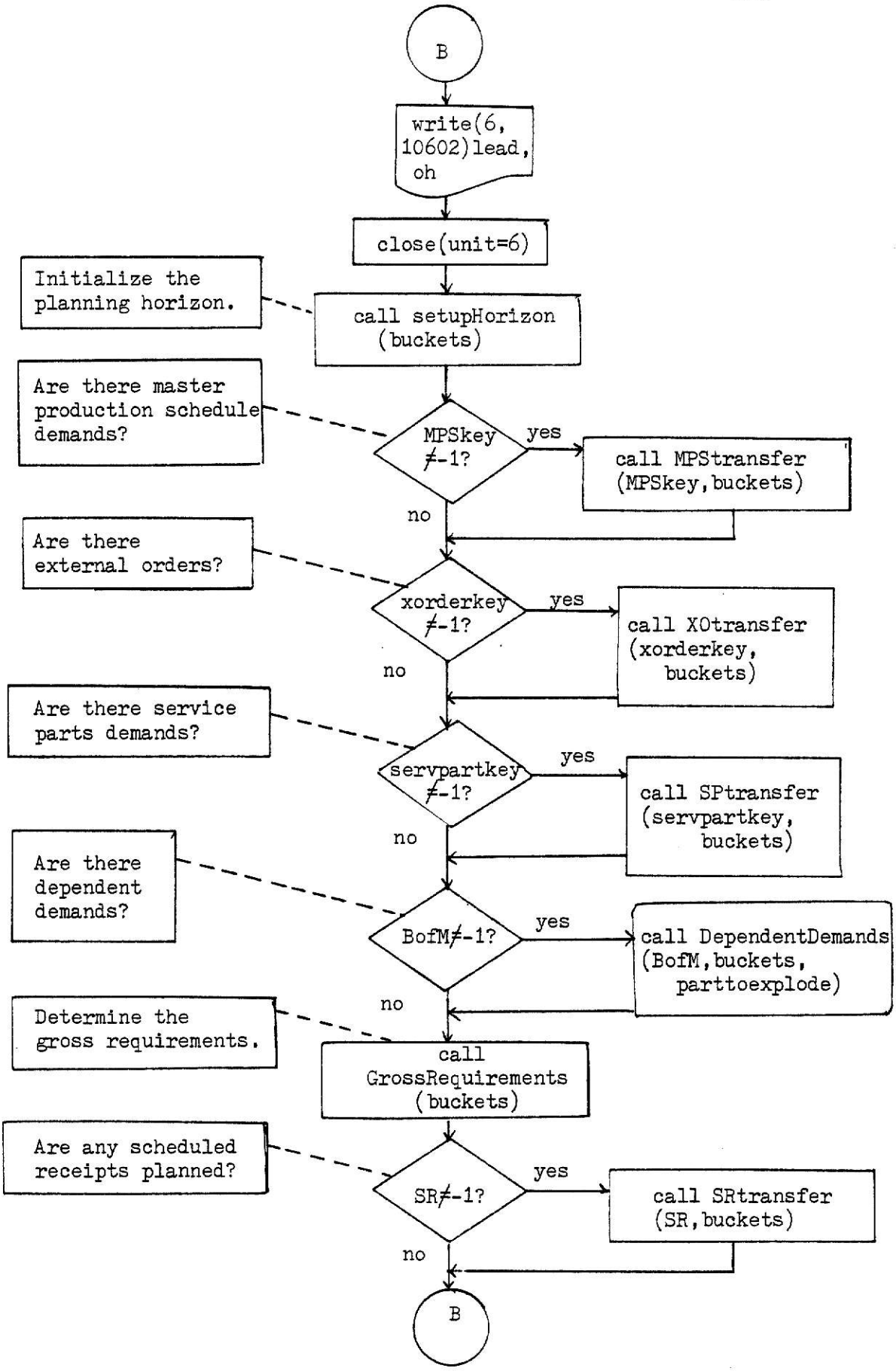

B
Beyond max level assigned? Stop.
L > Lmax?
yes
return
stop
no
C
i=1, nextpart-1, 1
A
>
≤
open(unit=3...)
3
i
read(...) oh,..., levelkey
level=L?
If level=L, set link.
no
yes
n=n+1
L > 1 .or. n > 1?
yes
levelkey1=i
no
3
i
write(...) oh1,..., levelkey1
Starting point. Store on record ∅.
Last Record Linked
L=1 .and. n=1?
yes
3
∅
write(...) nextpart,i
no
ccper1=ccper i1=i BOM1=BOM lotr1=lotr scost1=scost
pu1=pu oh1=oh BofM1=BofM lotp1=lotp ucost1=ucost
pup1=pup SR1=SR lead1=lead xorderkey1=xorderkey
MPSkey1=MPSkey numkids1=numkids servpartkey1=servpartkey
close(unit=3)
C

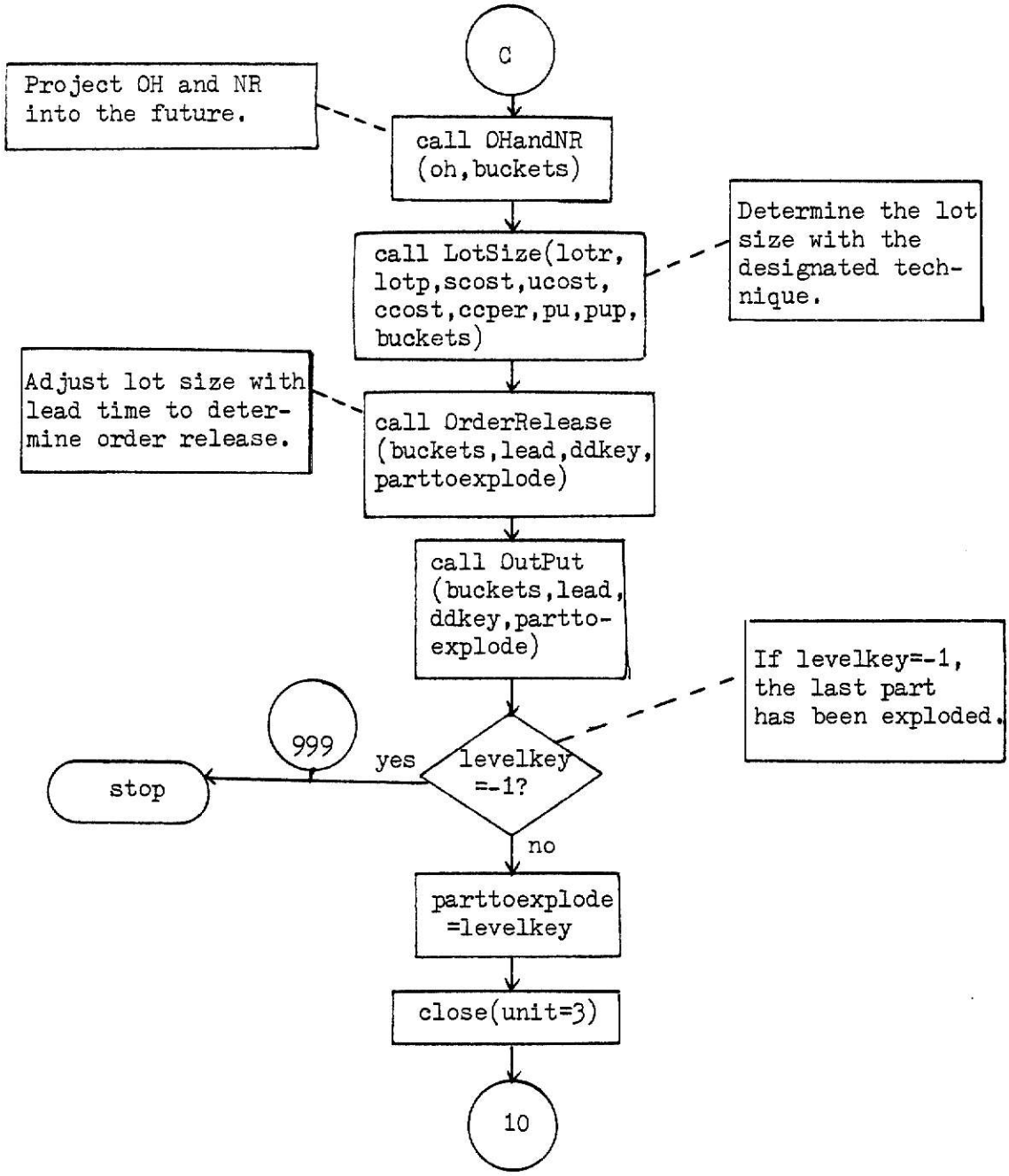
MRP

-Main Program-



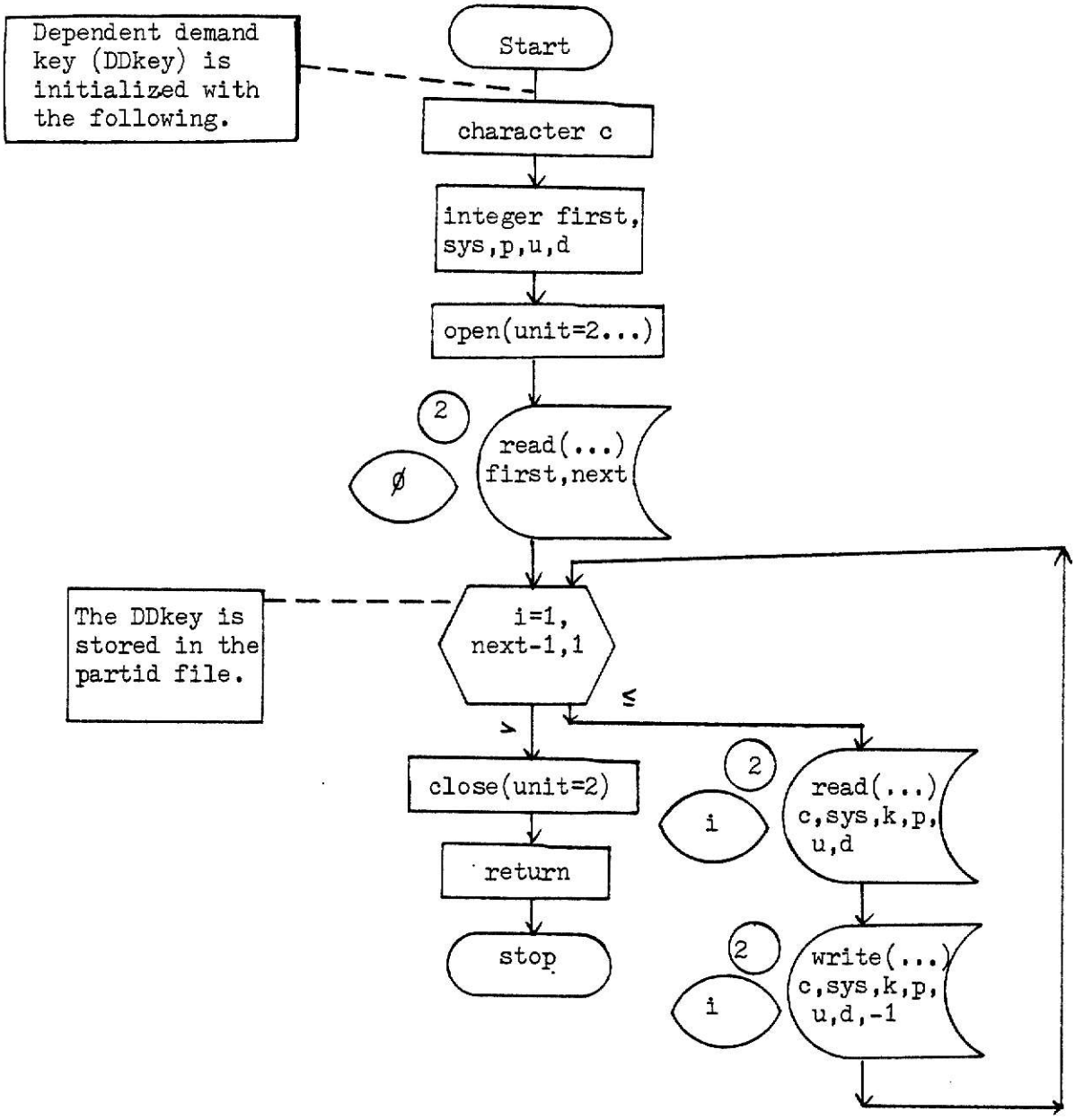

A
open(unit=3...)
Which part is exploded first? Start is the part's system number.
3
ø
read(...) nextpart, start
parttoexplode =start
close(unit=3)
10
open(unit=3...)
Get the scoop on the part to explode.
Part status data.
3
part toexplode
read(...) oh,..., levelkey
open(unit=2...)
Company part number.
2
partto explode
read(...) c
close(unit=2) open(unit=6...)
Heading information on the part.
write(6, 10601)c, partto-explode
B


B
write(6, 10602)lead, oh
close(unit=6)
Initialize the planning horizon.
call setupHorizon (buckets)
Are there master production schedule demands?
MPSkey ≠-1?
yes
call MPStransfer (MPSkey,buckets)
no
Are there external orders?
xorderkey ≠-1?
yes
call XOtransfer (xorderkey, buckets)
no
Are there service parts demands?
servpartkey ≠-1?
yes
call SPtransfer (servpartkey, buckets)
no
Are there dependent demands?
BofM≠-1?
yes
call DependentDemands (BofM,buckets, parttoexplode)
no
Determine the gross requirements.
call GrossRequirements (buckets)
Are any scheduled receipts planned?
SR≠-1?
yes
call SRtransfer (SR,buckets)
no
B


C
call OHandNR (oh,buckets)
Project OH and NR into the future.
call LotSize(lotr, lotp,scost,ucost, ccost,ccper,pu,pup, buckets)
Determine the lot size with the designated technique.
call OrderRelease (buckets,lead,ddkey, parttoexplode)
Adjust lot size with lead time to determine order release.
call OutPut (buckets,lead, ddkey,partto- explode)
levelkey =-1?
yes
999
stop
If levelkey=-1, the last part has been exploded.
no
parttoexplode =levelkey
close(unit=3)
10

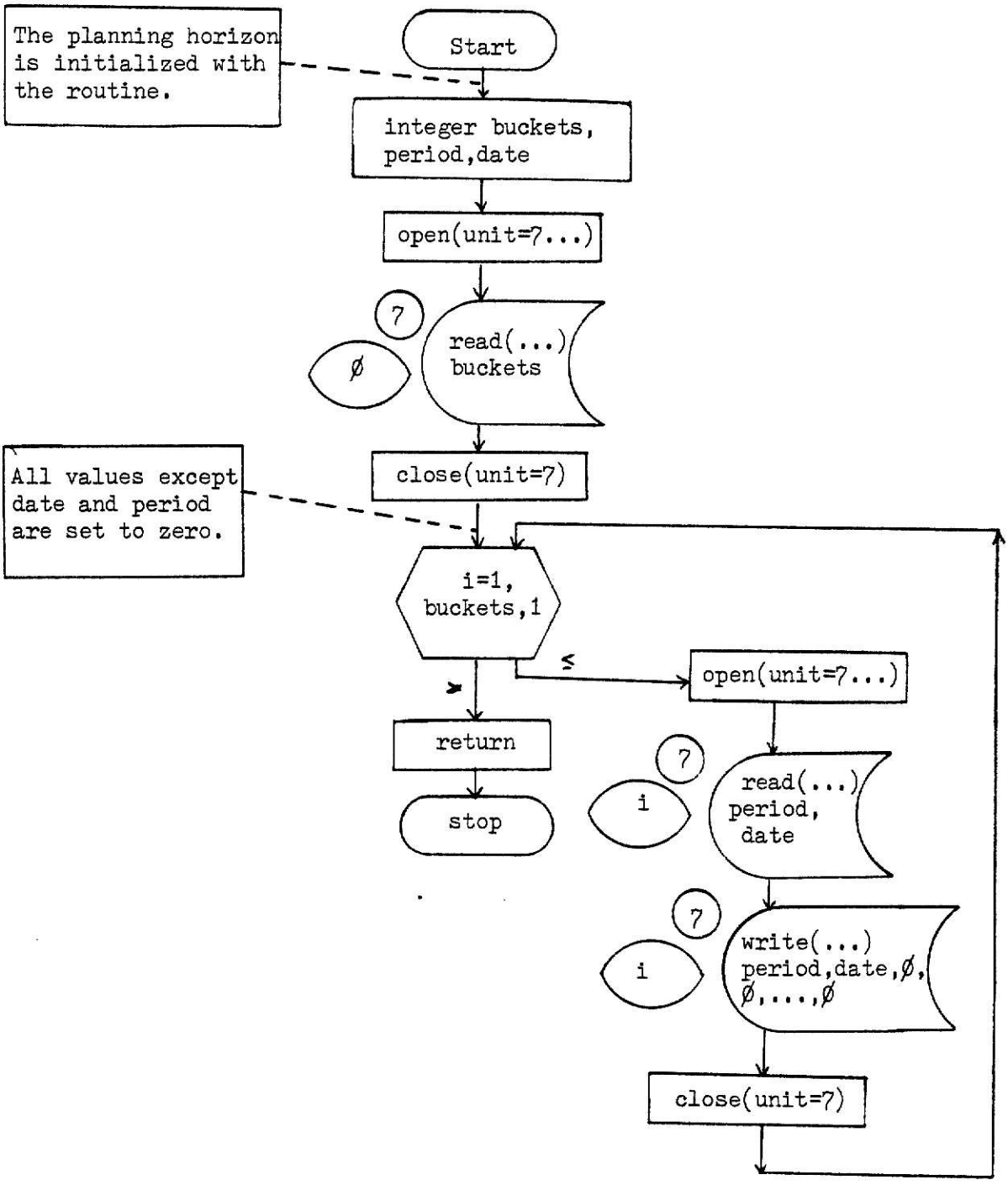
MRP

-Subroutine DDkey-



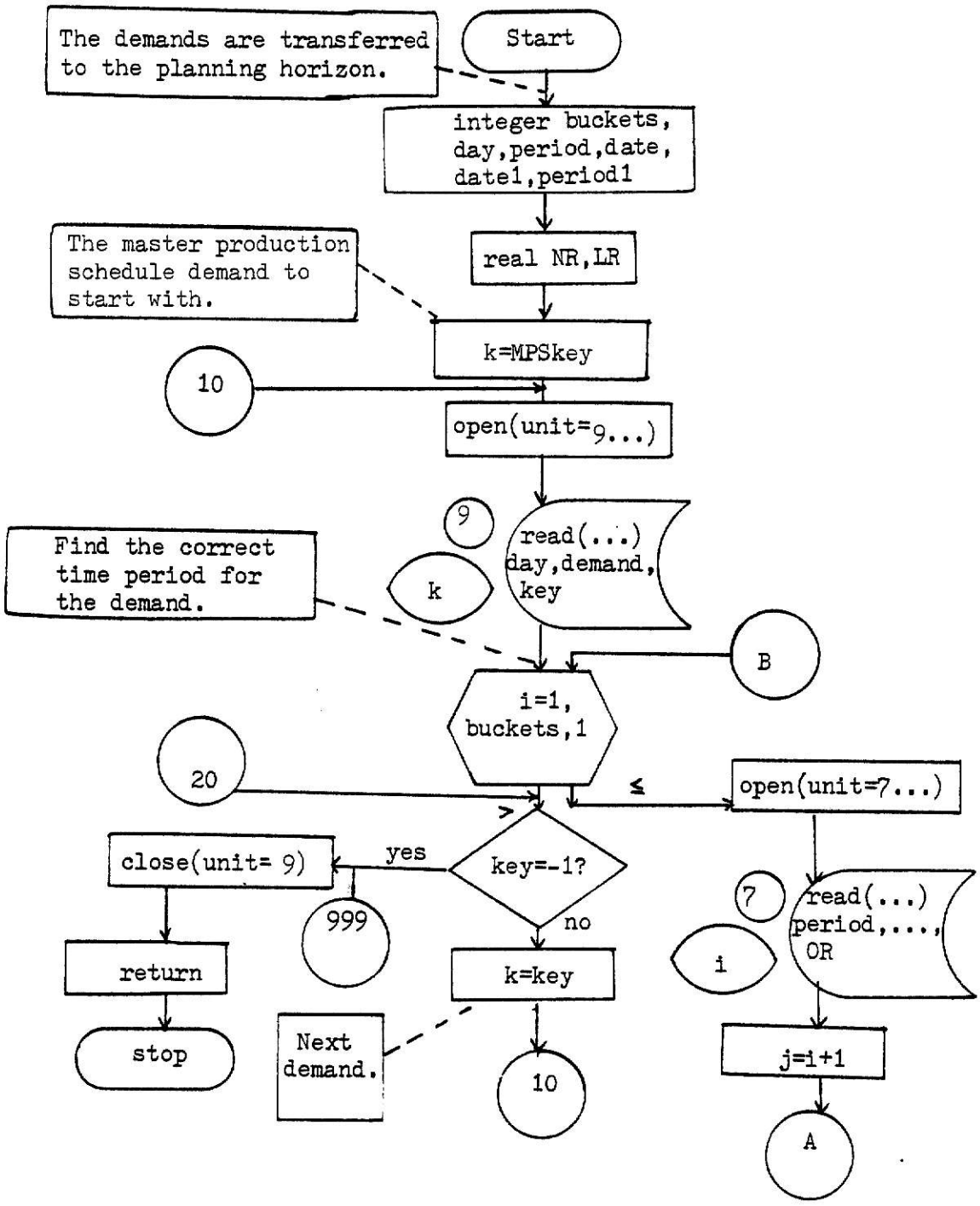
MRP

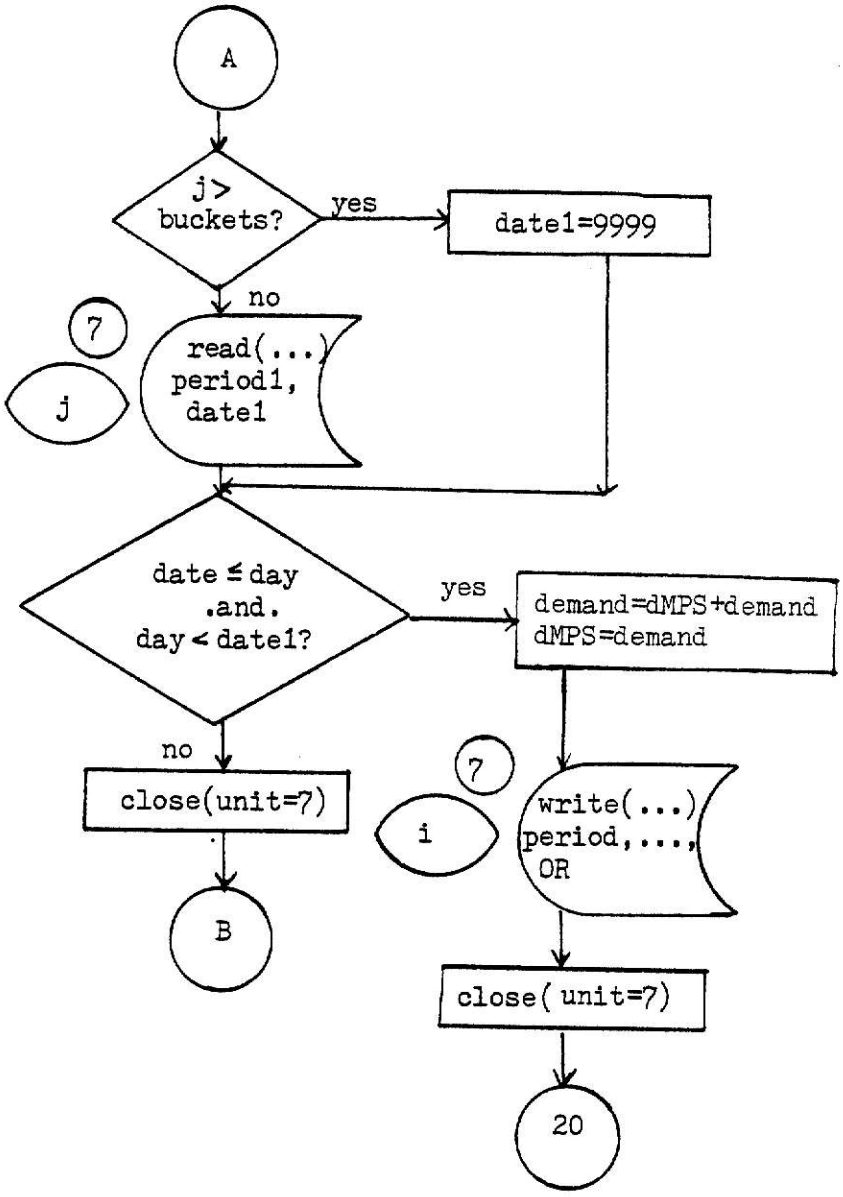
-Subroutine setupHorizon-
Arguments: buckets



MRP

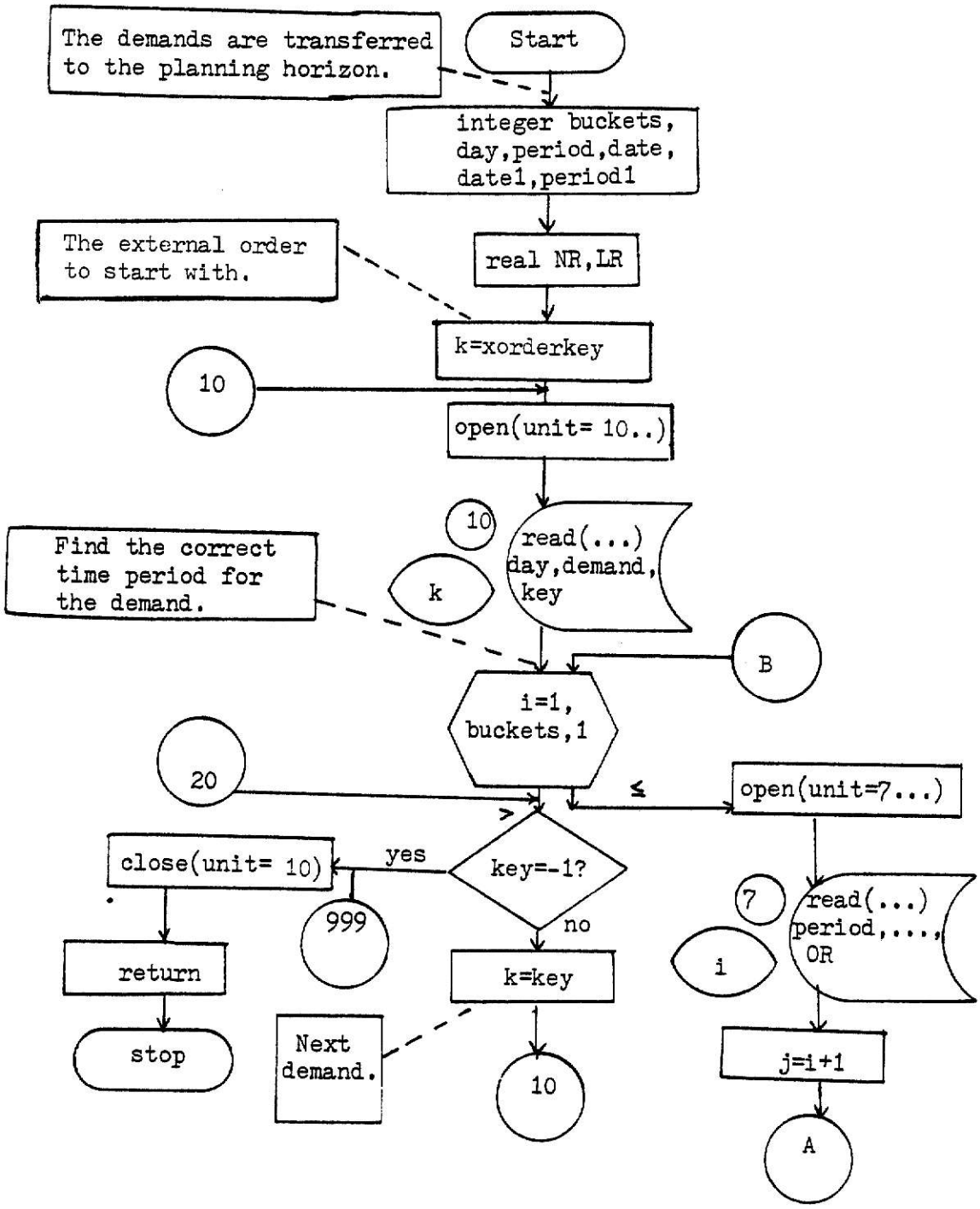
-Subroutine MPStransfer-
Arguments: MPSkey,
buckets

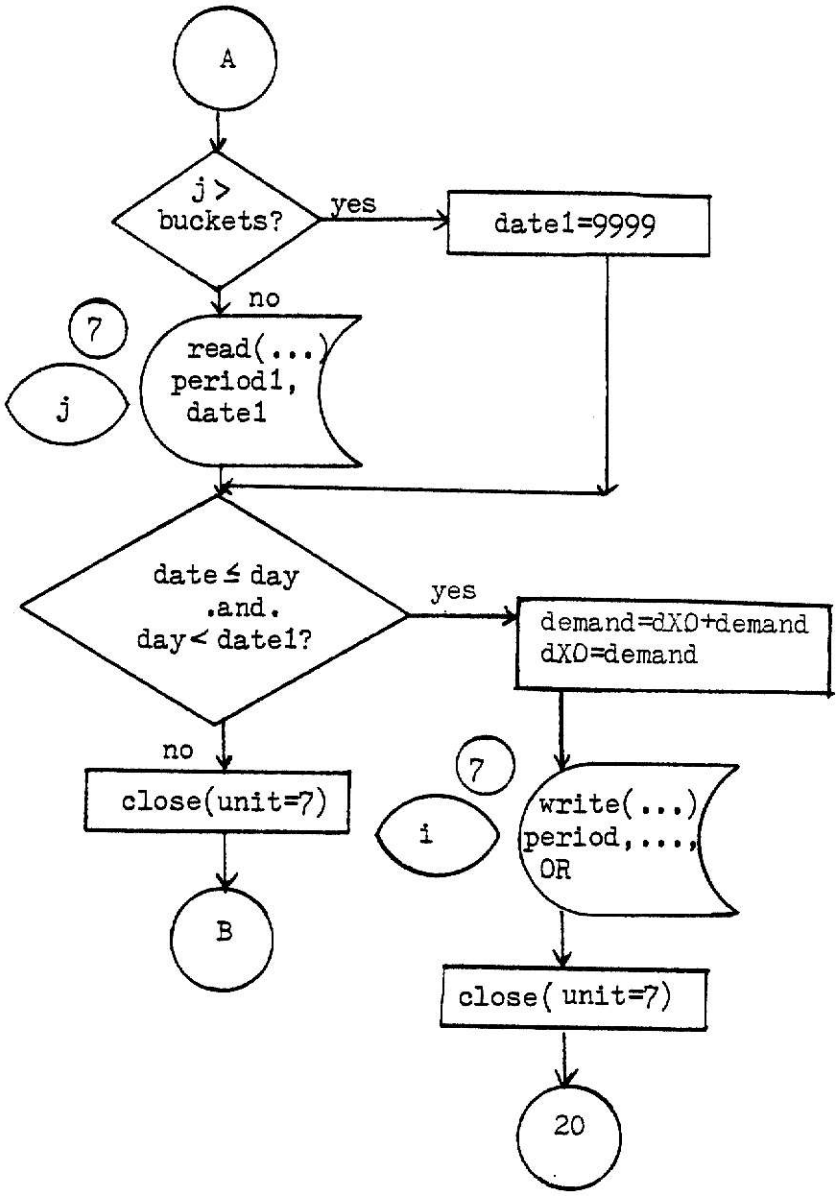



A
j>
buckets?
yes
date1=9999
no
7
j
read(...)
period1,
date1
date ≤ day
.and.
day < date1?
yes
demand=dMPS+demand
dMPS=demand
no
close(unit=7)
B
7
i
write(...)
period,...,
OR
close(unit=7)
20

MRP

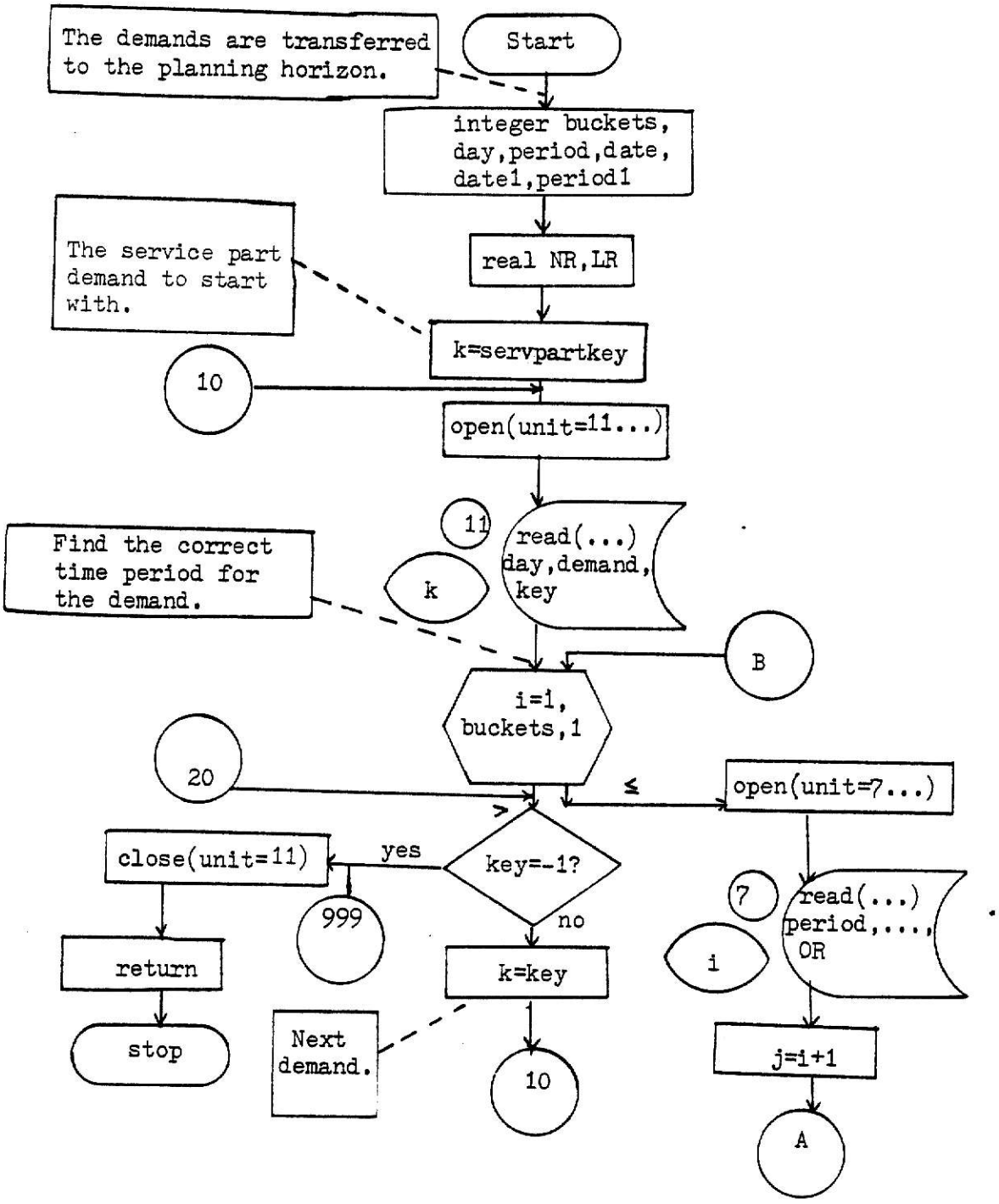
-Subroutine XOtransfer-
Arguments: xorderkey,
buckets

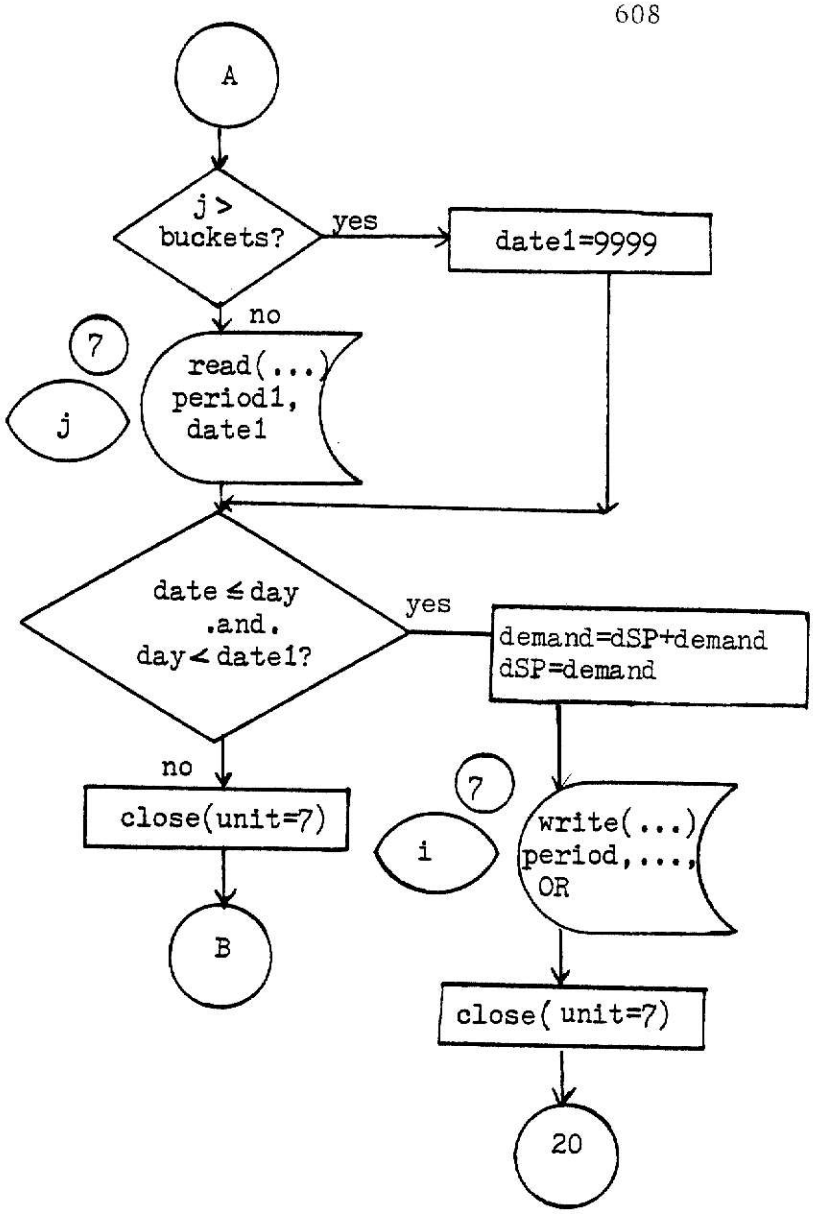



A
j > buckets?
yes
date1=9999
no
7
j
read(...) period1, date1
date ≤ day .and. day < date1?
yes
demand=dXO+demand
dXO=demand
no
close(unit=7)
B
7
i
write(...) period,...,OR
close(unit=7)
20

MRP

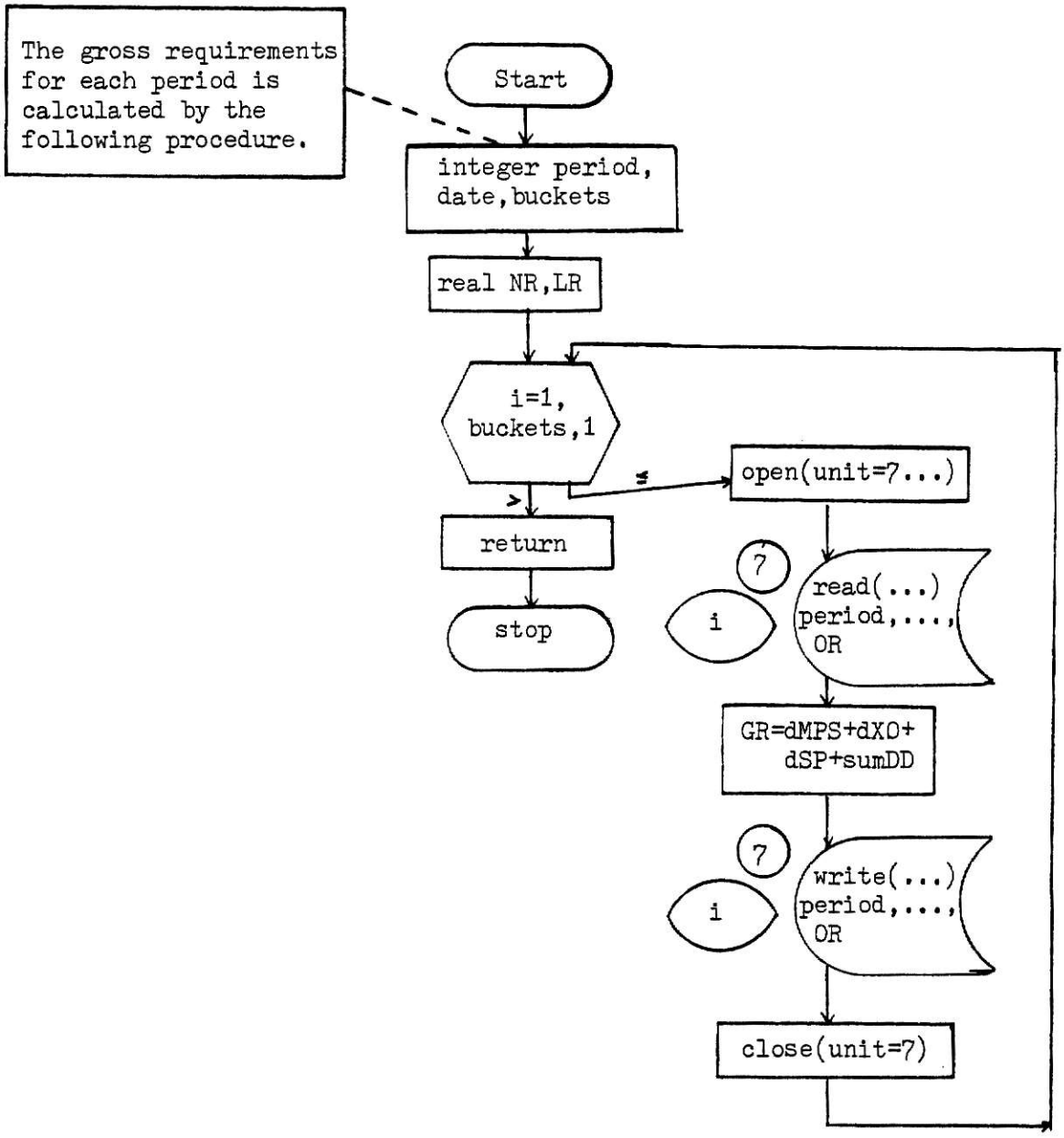
-Subroutine SPtransfer-
Arguments: servpartkey, buckets




A
j > buckets?
yes
date1=9999
no
7
j
read(...) period1, date1
date ≤ day .and. day < date1?
yes
demand=dSP+demand
dSP=demand
no
close(unit=7)
B
7
i
write(...) period,..., OR
close(unit=7)
20

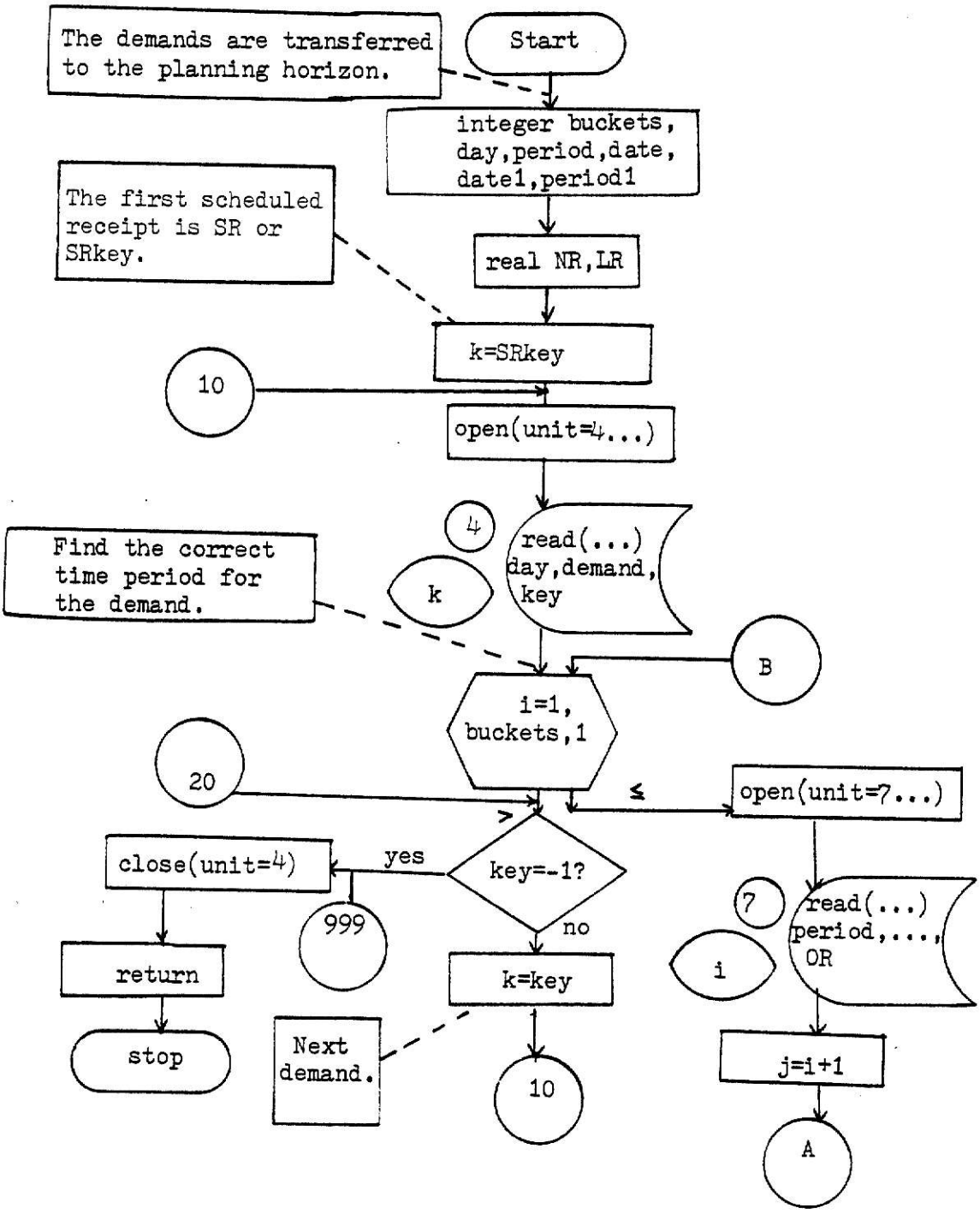
MRP

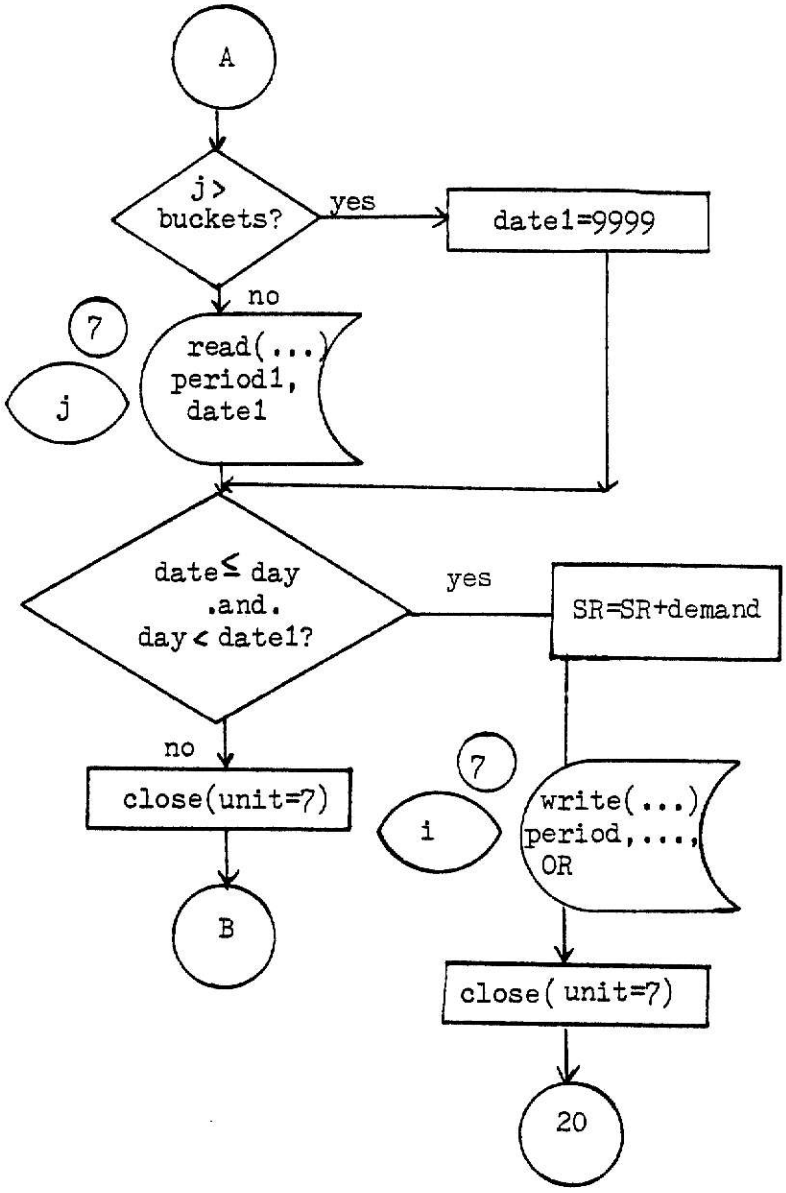
-Subroutine GrossRequirements-
Arguments: buckets



MRP

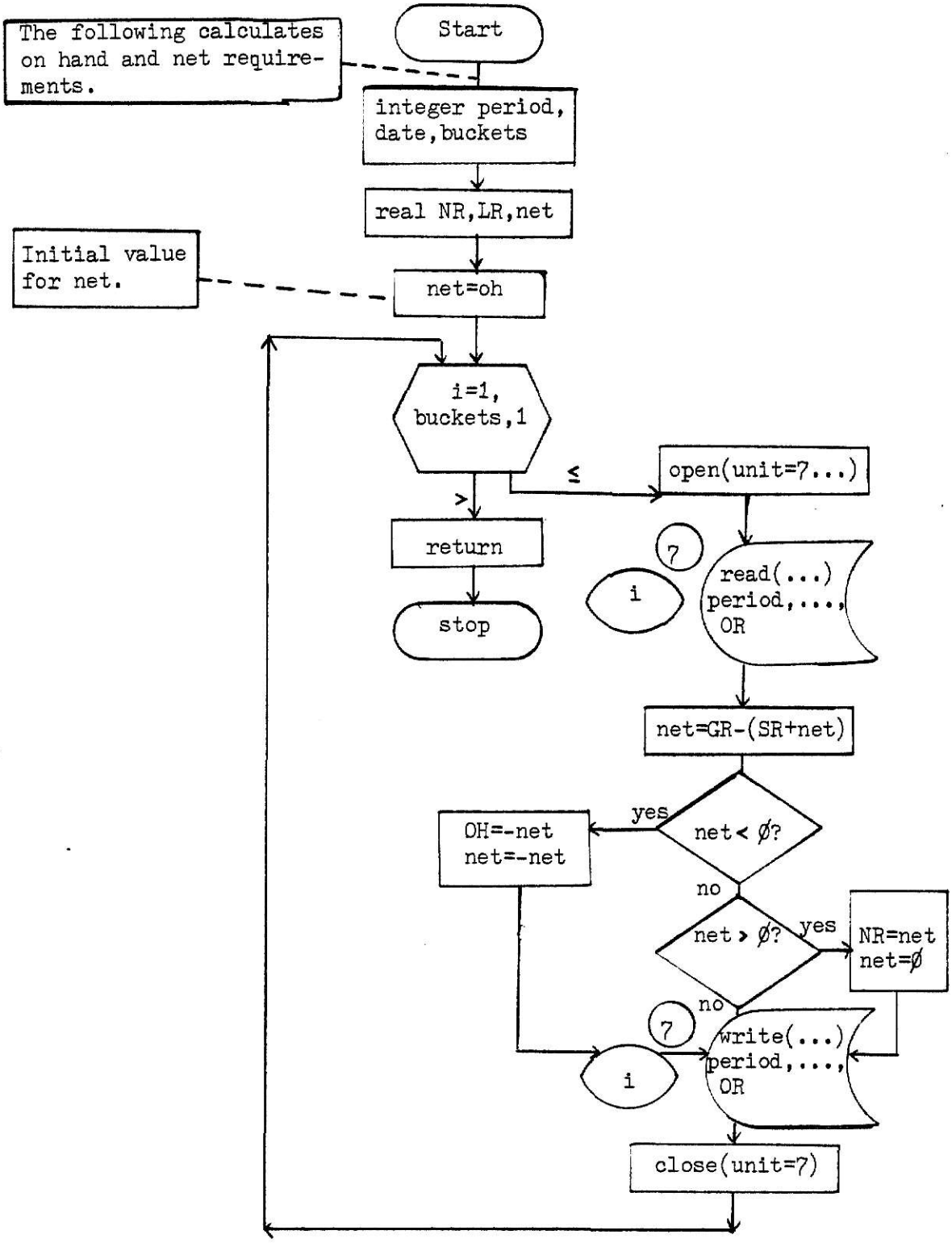
-Subroutine SRtransfer-
Arguments: SRkey,
buckets




A
j>
buckets?
yes
date1=9999
no
7
j
read(...)
period1,
date1
date≤ day
.and.
day < date1?
yes
SR=SR+demand
no
close(unit=7)
B
7
i
write(...)
period,...,
OR
close(unit=7)
20

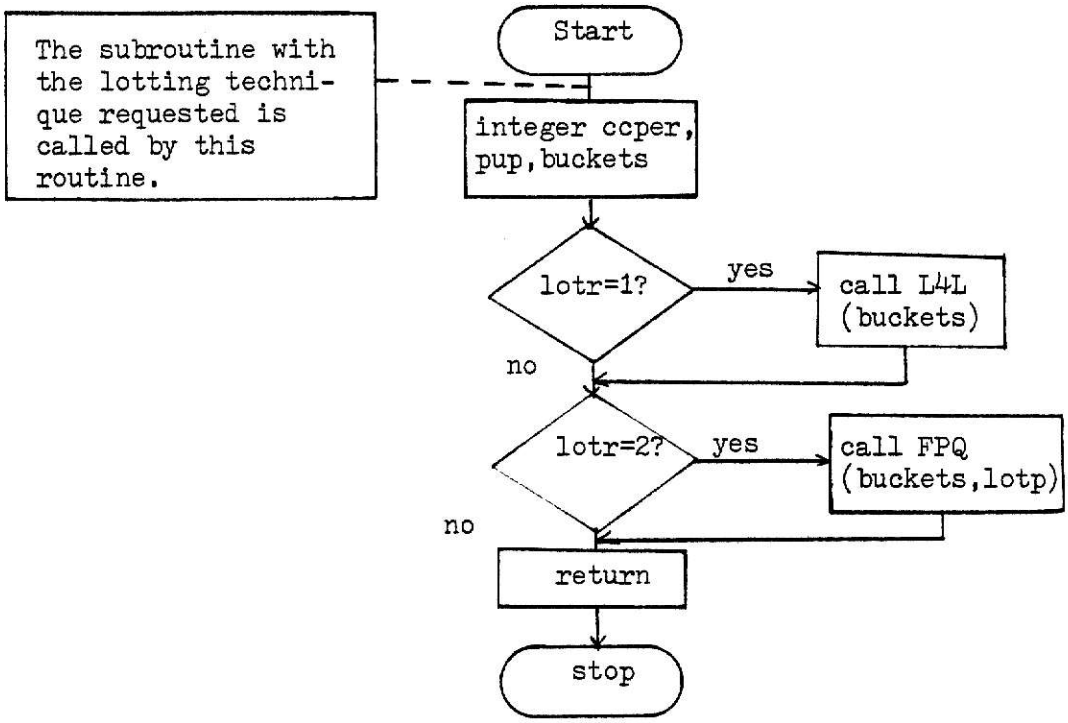
MRP

-Subroutine OHandNR-
Arguments: oh,buckets



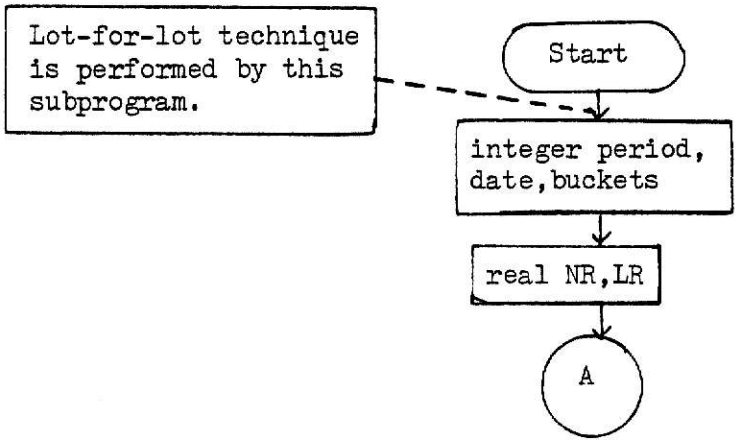
MRP

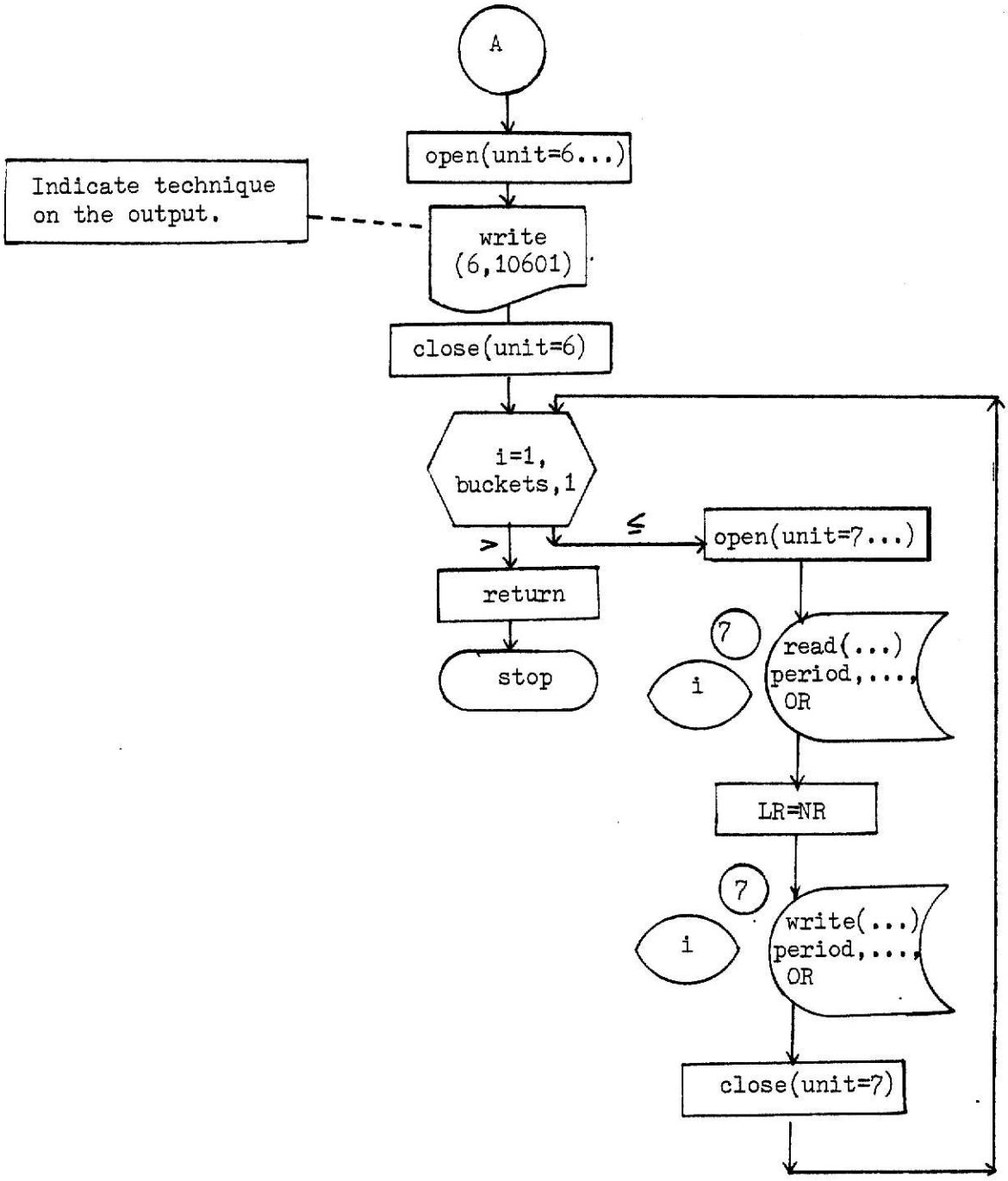
-Subroutine LotSize-
Arguments: lotr,lotp,
scost,ucost,ccost,
ccper,pu,pup,buckets



MRP

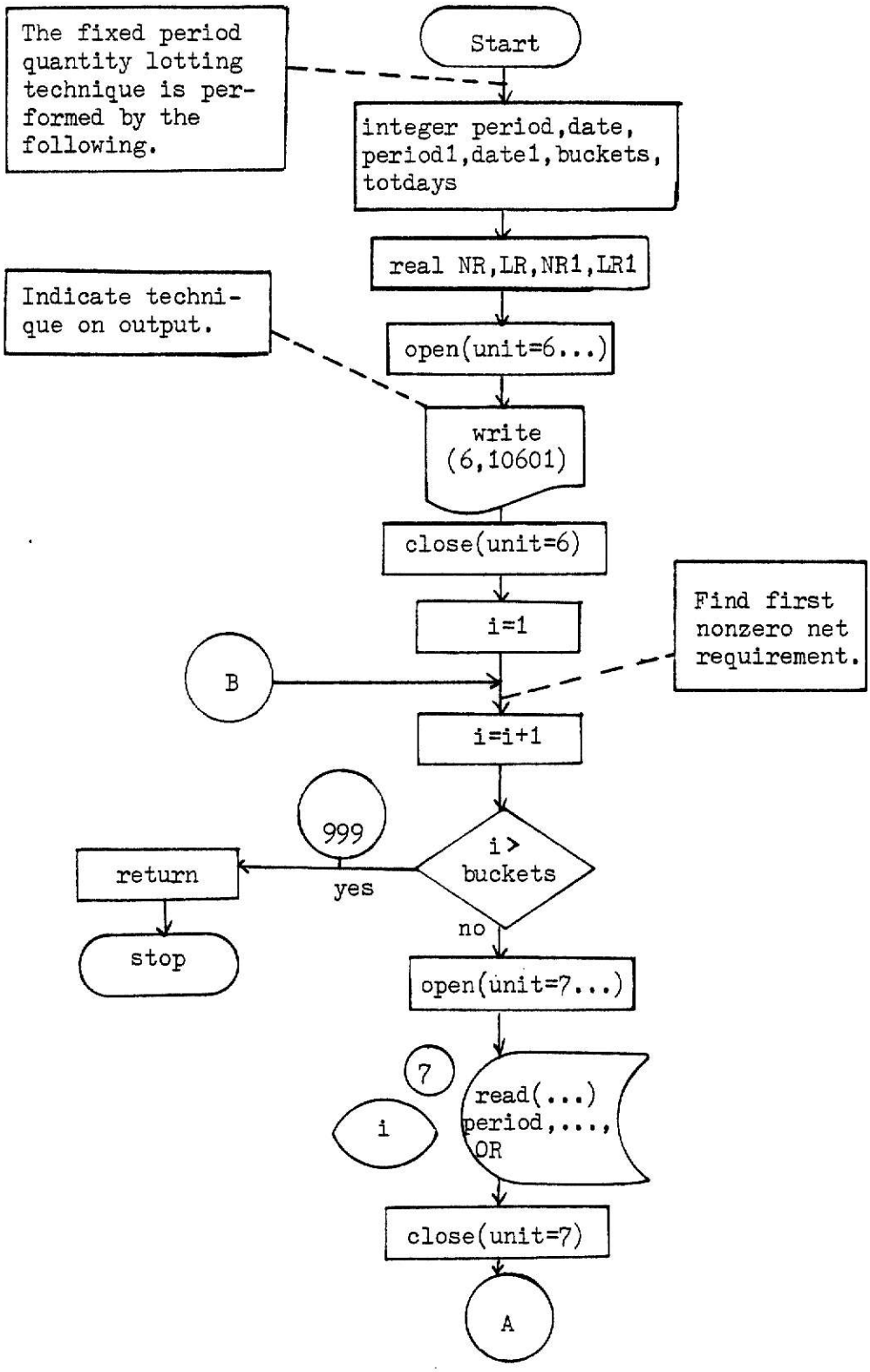
-Subroutine L4L-
Argument: buckets

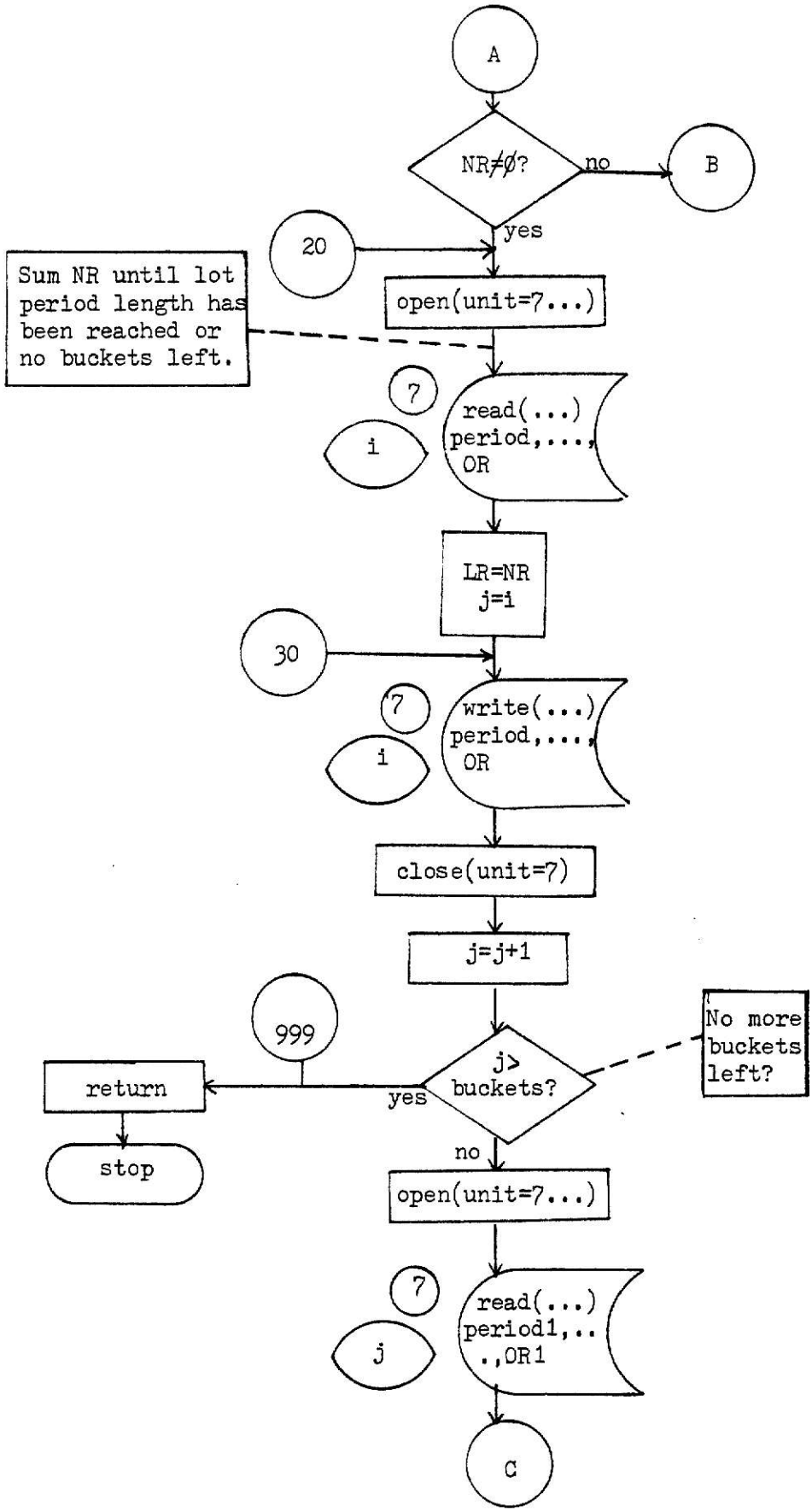


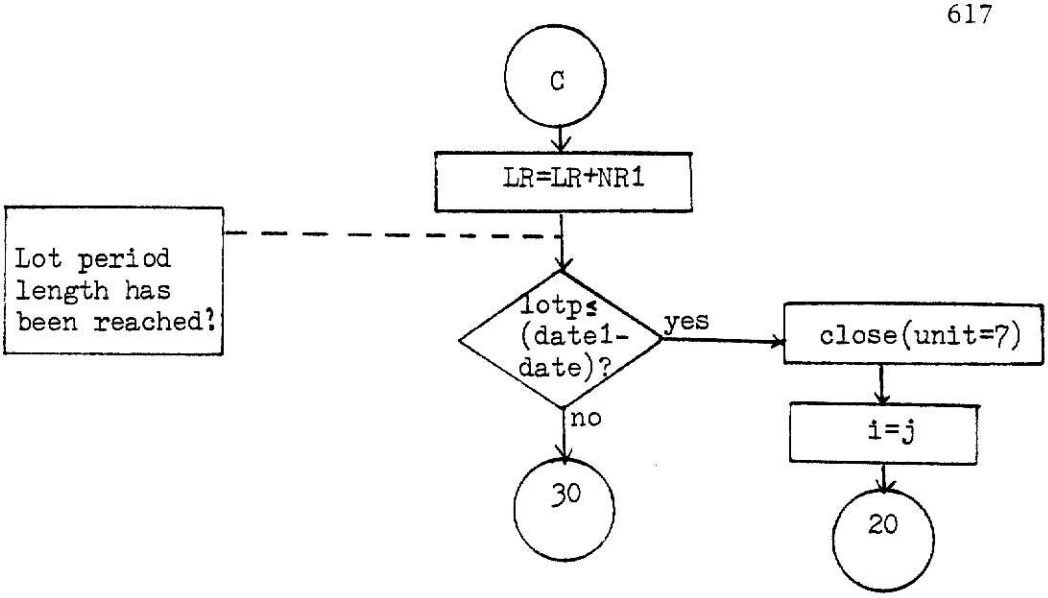

A
open(unit=6...)
Indicate technique on the output.
write (6,10601)
close(unit=6)
i=1, buckets,1
≤
>
open(unit=7...)
return
stop
7
i
read(...) period,..., OR
LR=NR
7
i
write(...) period,..., OR
close(unit=7)

MRP

-Subroutine FPQ-
Arguments: buckets,lotp

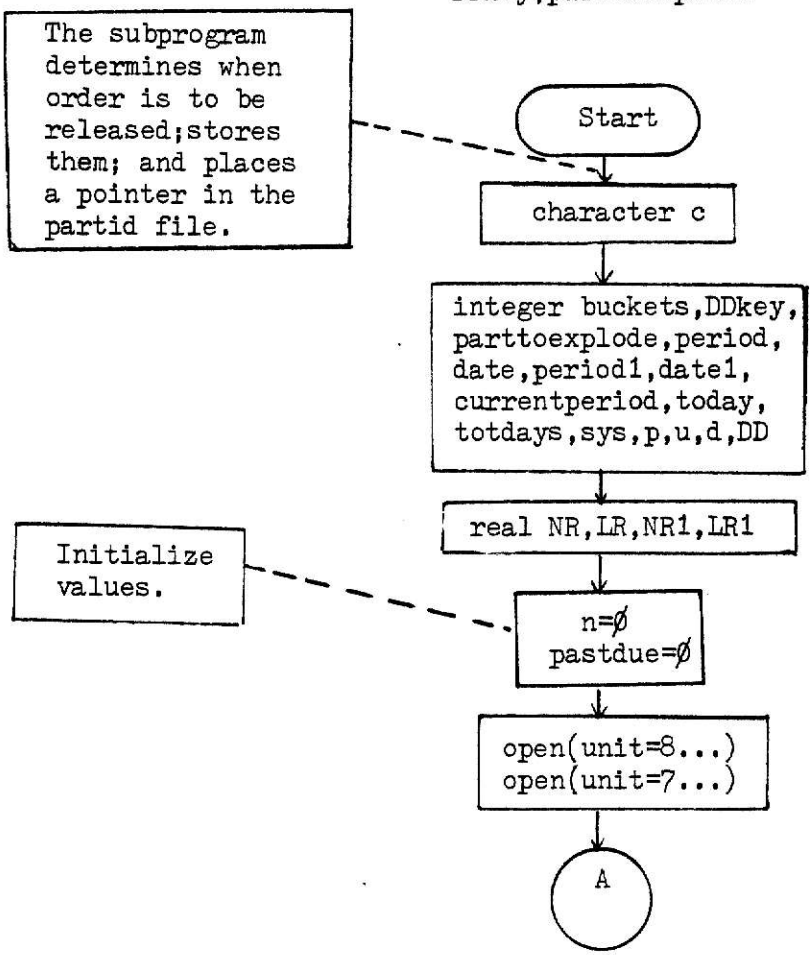


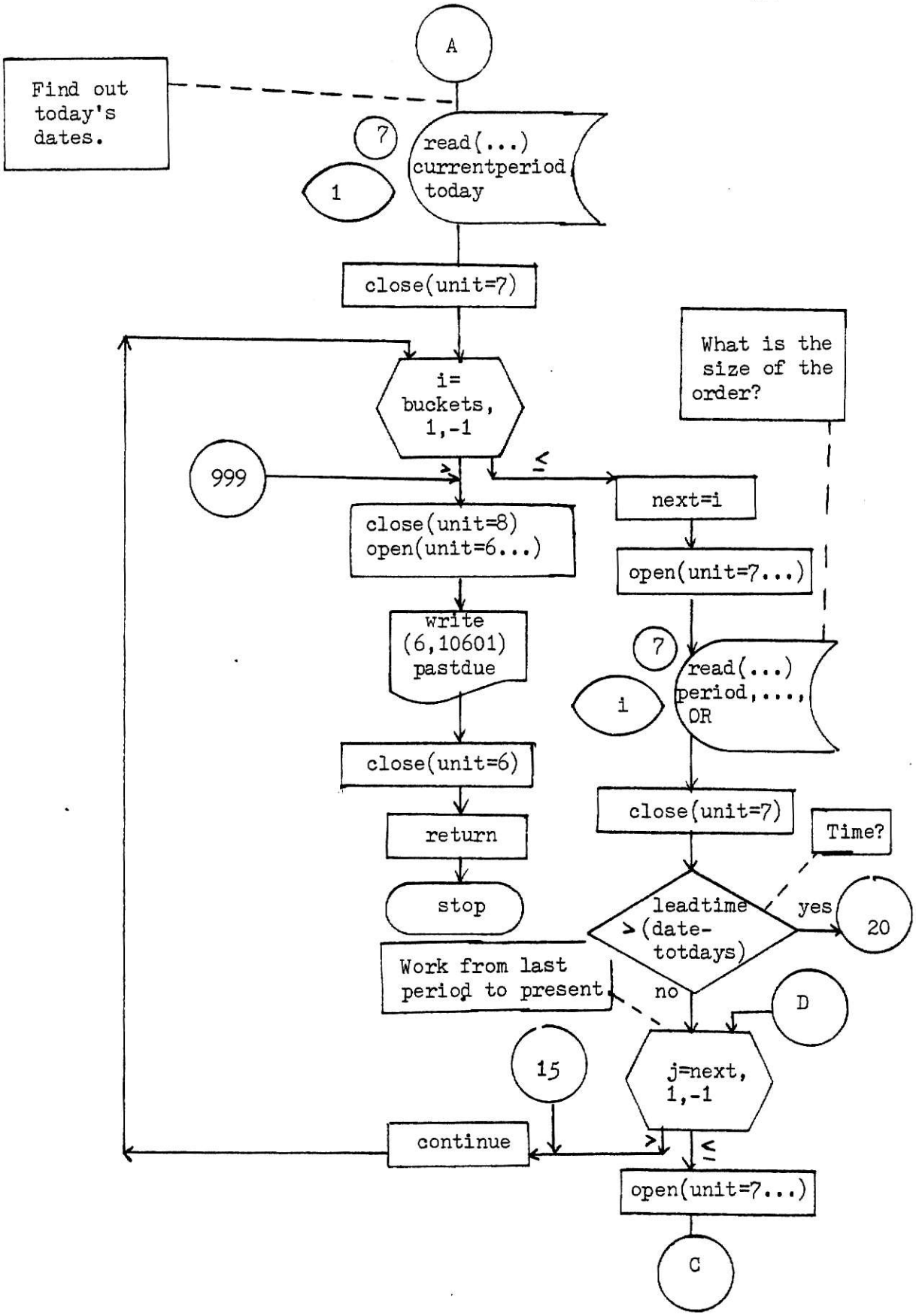

A
NR≠Ø?
no
B
yes
20
Sum NR until lot period length has been reached or no buckets left.
open(unit=7...)
7
i
read(...) period,..., OR
LR=NR
j=i
30
7
i
write(...) period,..., OR
close(unit=7)
j=j+1
999
j> buckets?
No more buckets left?
yes
return
stop
no
open(unit=7...)
7
j
read(...) period1,.. .,OR1
C

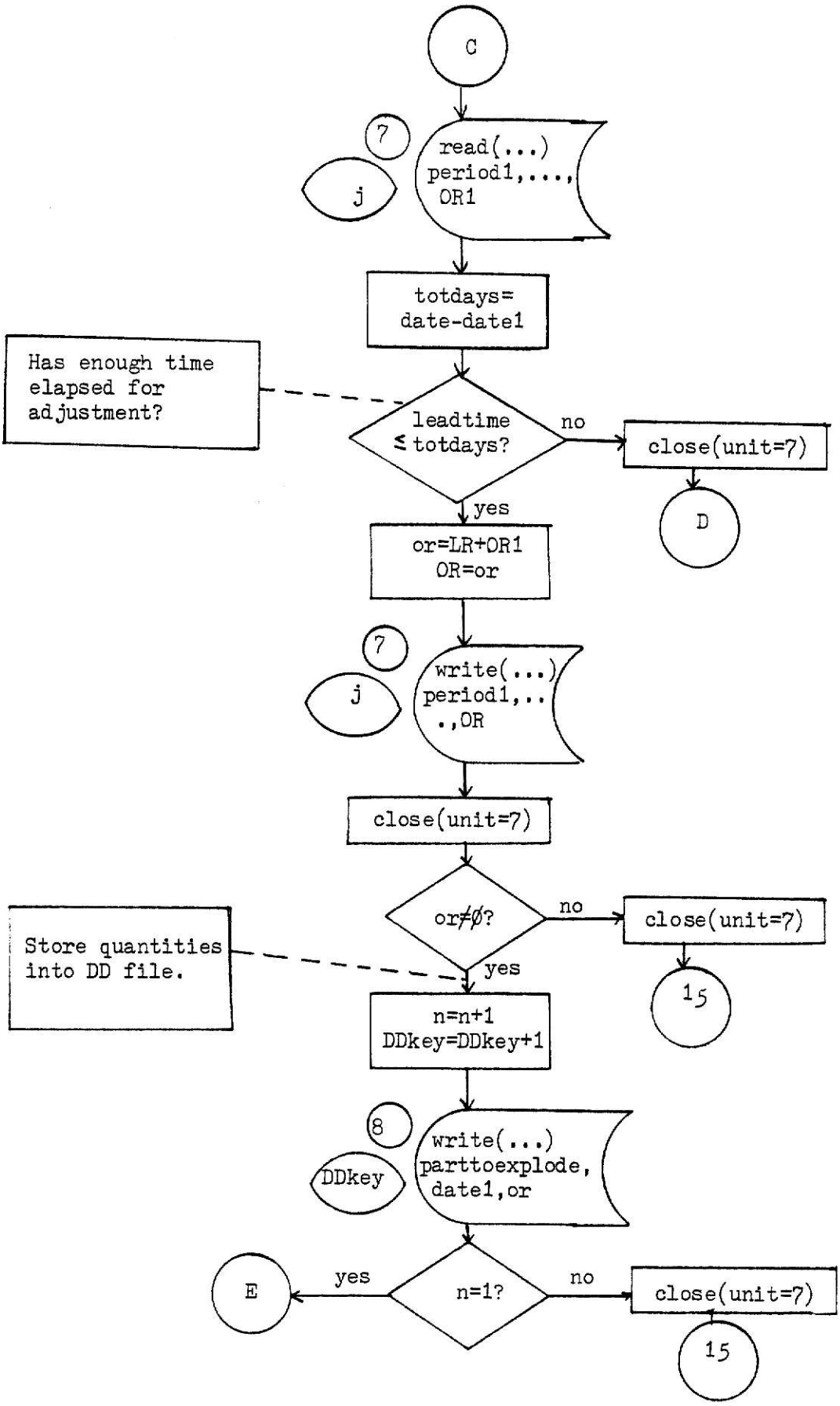


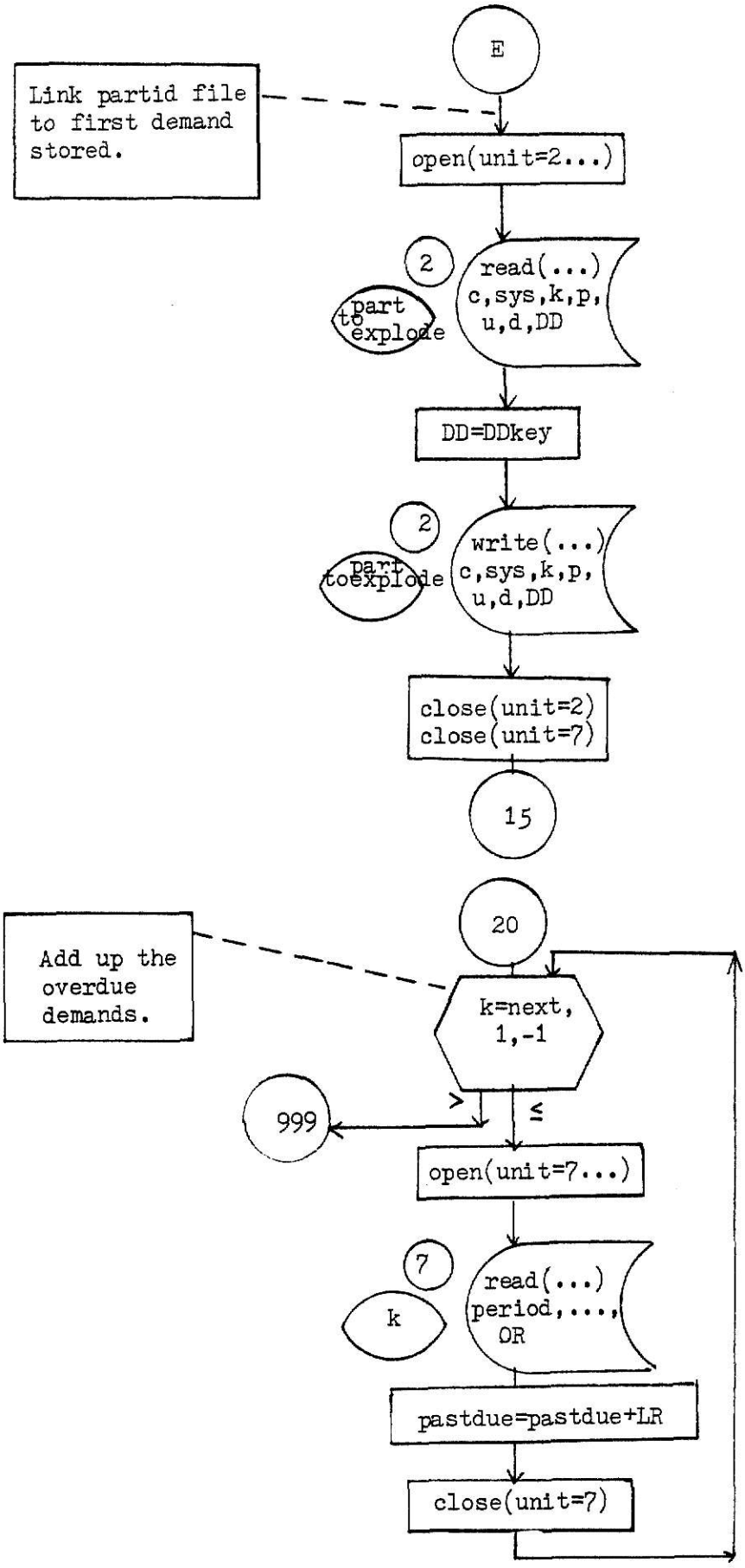
MRP

-Subroutine OrderRelease-
Arguments: buckets,leadtime,
DDkey,parttoexplode



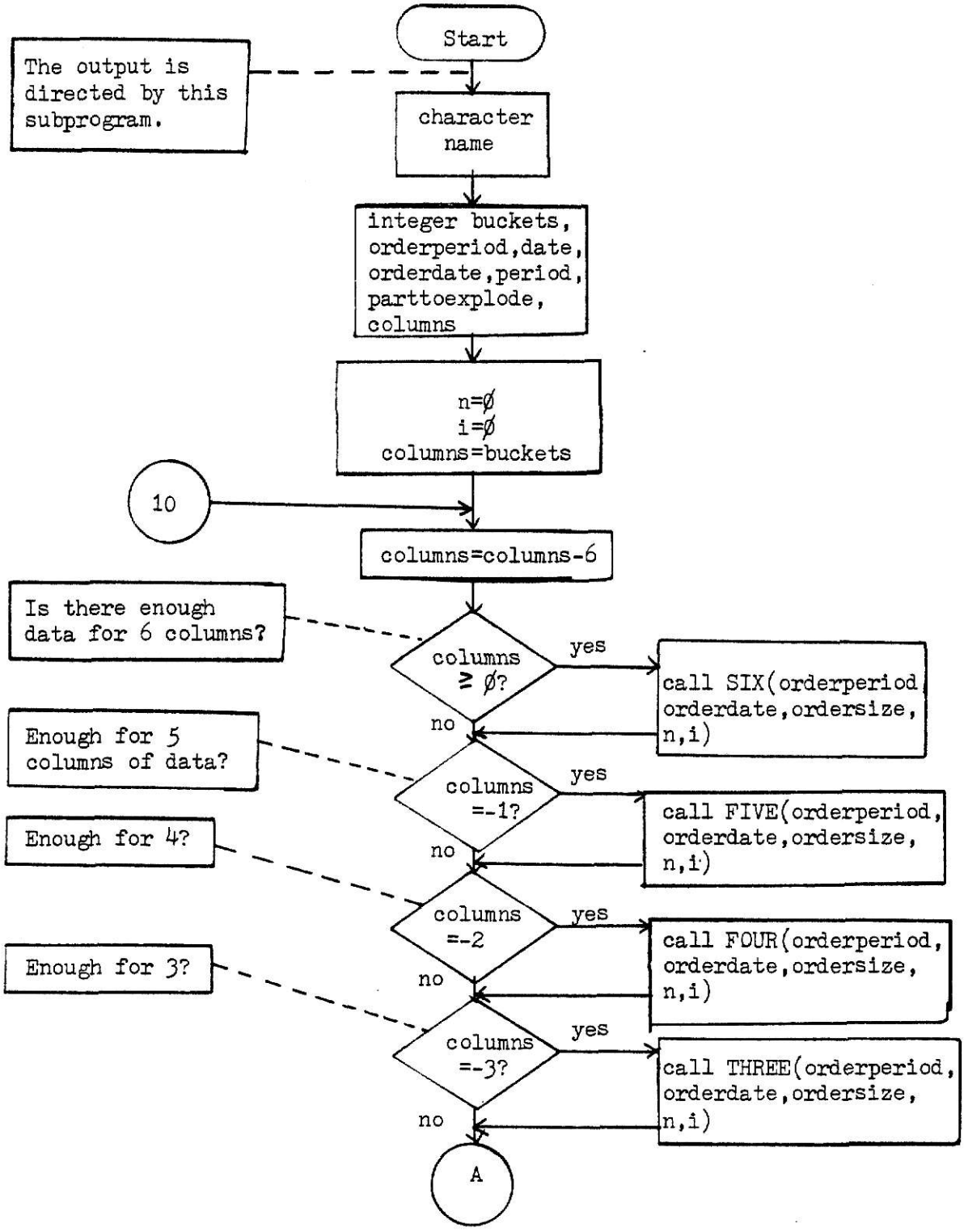

A
Find out today's dates.
7
1
read(...) currentperiod, today
close(unit=7)
What is the size of the order?
i= buckets, 1,-1
999
>
≤
next=i
close(unit=8) open(unit=6...)
open(unit=7...)
write (6,10601) pastdue
7
i
read(...) period,..., OR
close(unit=6)
close(unit=7)
Time?
return
stop
leadtime > (date- totdays)
yes
20
Work from last period to present
no
D
15
j=next, 1,-1
continue
>
≤
open(unit=7...)
C

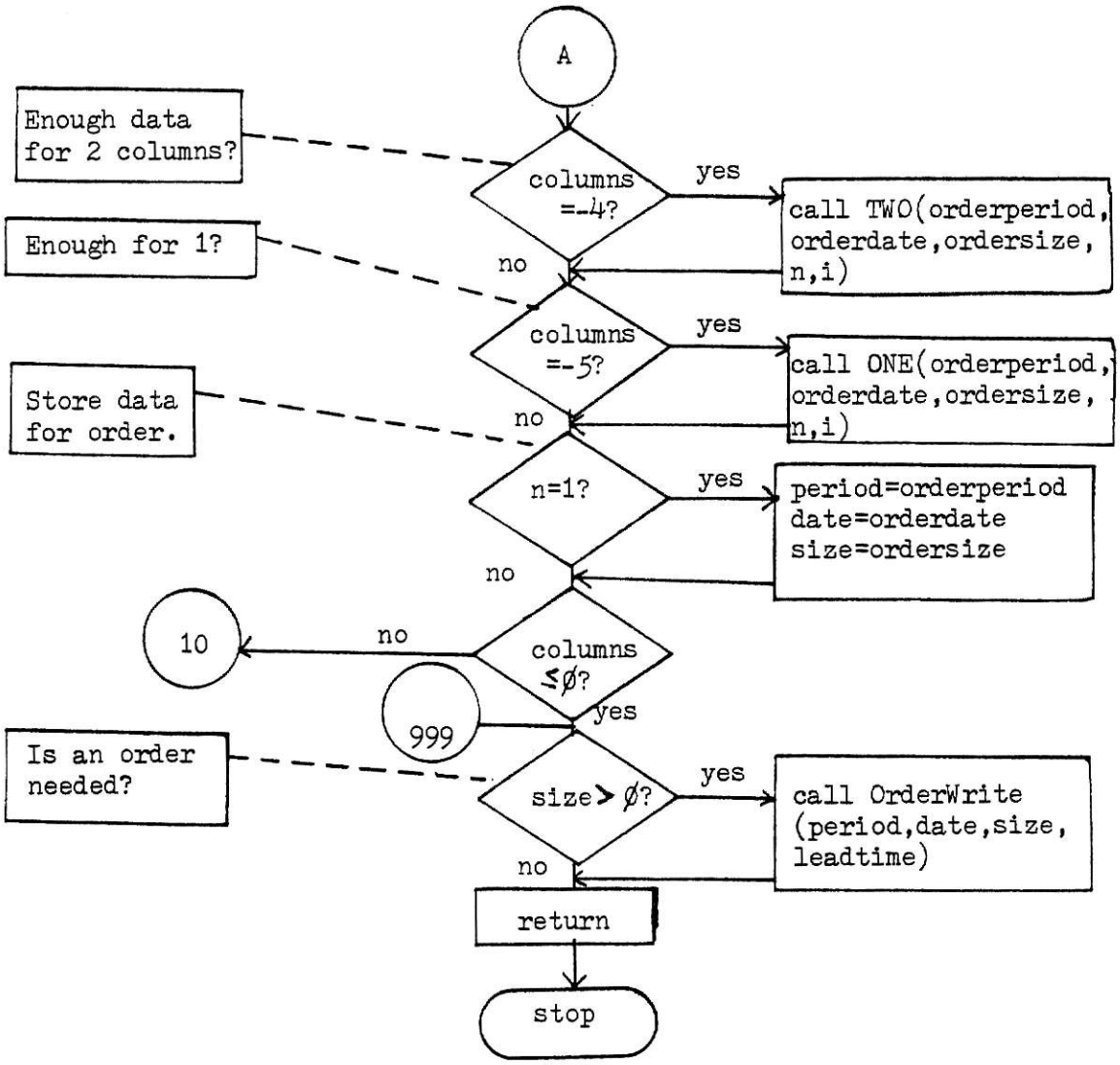

C
7
j
read(...)
period1,...,
OR1
totdays=
date-date1
Has enough time
elapsed for
adjustment?
leadtime
≤ totdays?
no
close(unit=7)
D
yes
or=LR+OR1
OR=or
7
j
write(...)
period1,..
.,OR
close(unit=7)
or≠0?
no
close(unit=7)
15
Store quantities
into DD file.
yes
n=n+1
DDkey=DDkey+1
8
DDkey
write(...)
parttoexplode,
date1,or
E
yes
n=1?
no
close(unit=7)
15


E
Link partid file to first demand stored.
open(unit=2...)
2
part to explode
read(...) c,sys,k,p, u,d,DD
DD=DDkey
2
part toexplode
write(...) c,sys,k,p, u,d,DD
close(unit=2)
close(unit=7)
15
20
Add up the overdue demands.
k=next, 1,-1
>
≤
999
open(unit=7...)
7
k
read(...) period,...., OR
pastdue=pastdue+LR
close(unit=7)

MRP

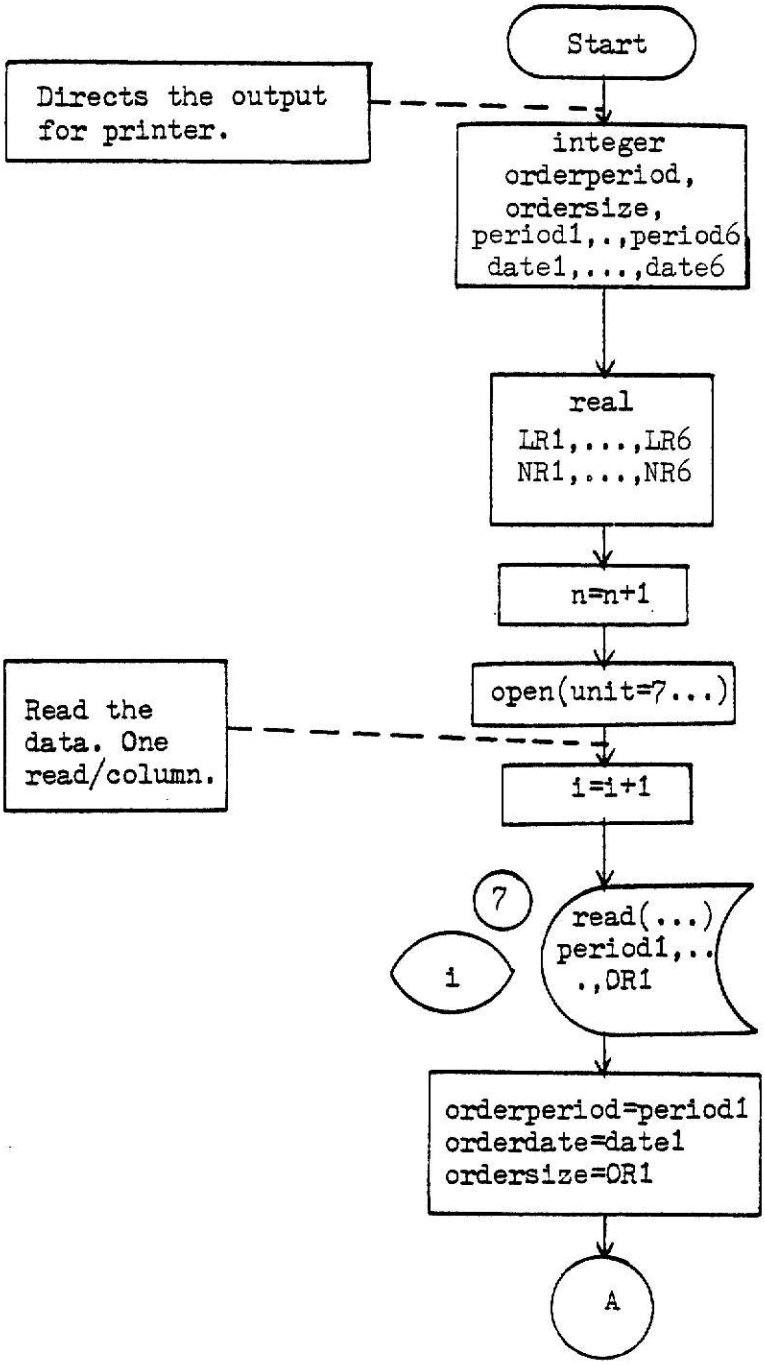
-Subroutine OutPut-
Arguments: buckets,leadtime,
name,parttoexplode

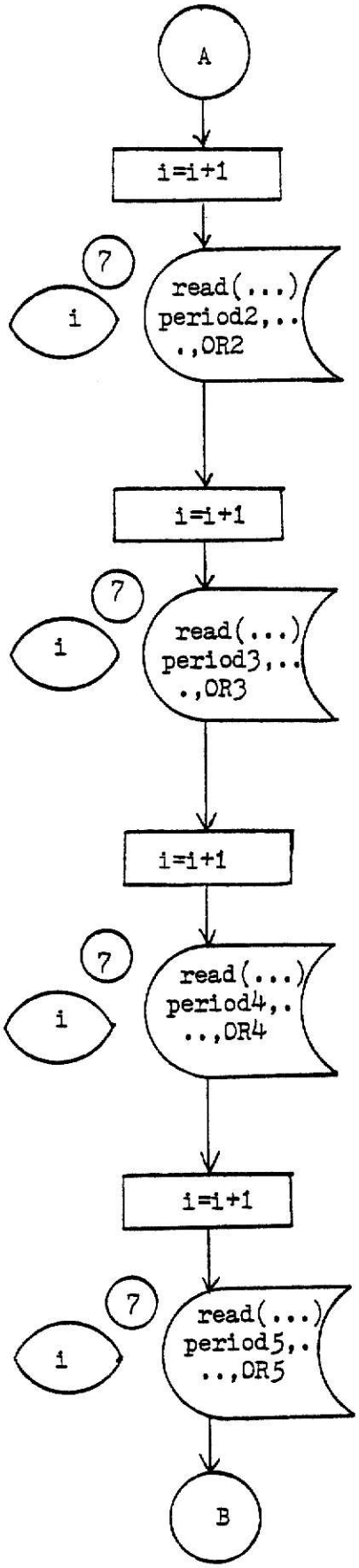


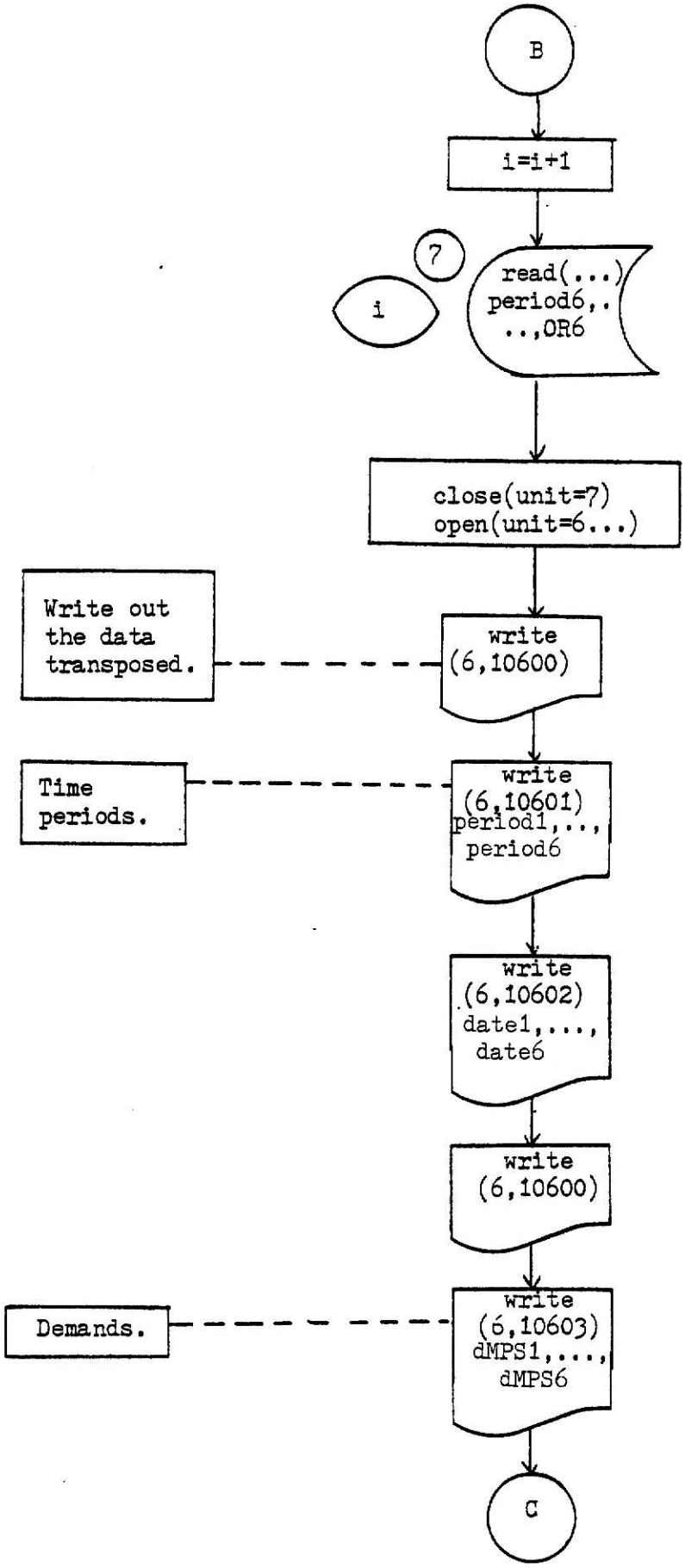

A
Enough data for 2 columns?
columns =-4?
yes
call TWO(orderperiod, orderdate,ordersize, n,i)
Enough for 1?
no
columns =-5?
yes
call ONE(orderperiod, orderdate,ordersize, n,i)
Store data for order.
no
n=1?
yes
period=orderperiod
date=orderdate
size=ordersize
no
10
no
columns ≤0?
999
yes
Is an order needed?
size > 0?
yes
call OrderWrite (period,date,size, leadtime)
no
return
stop

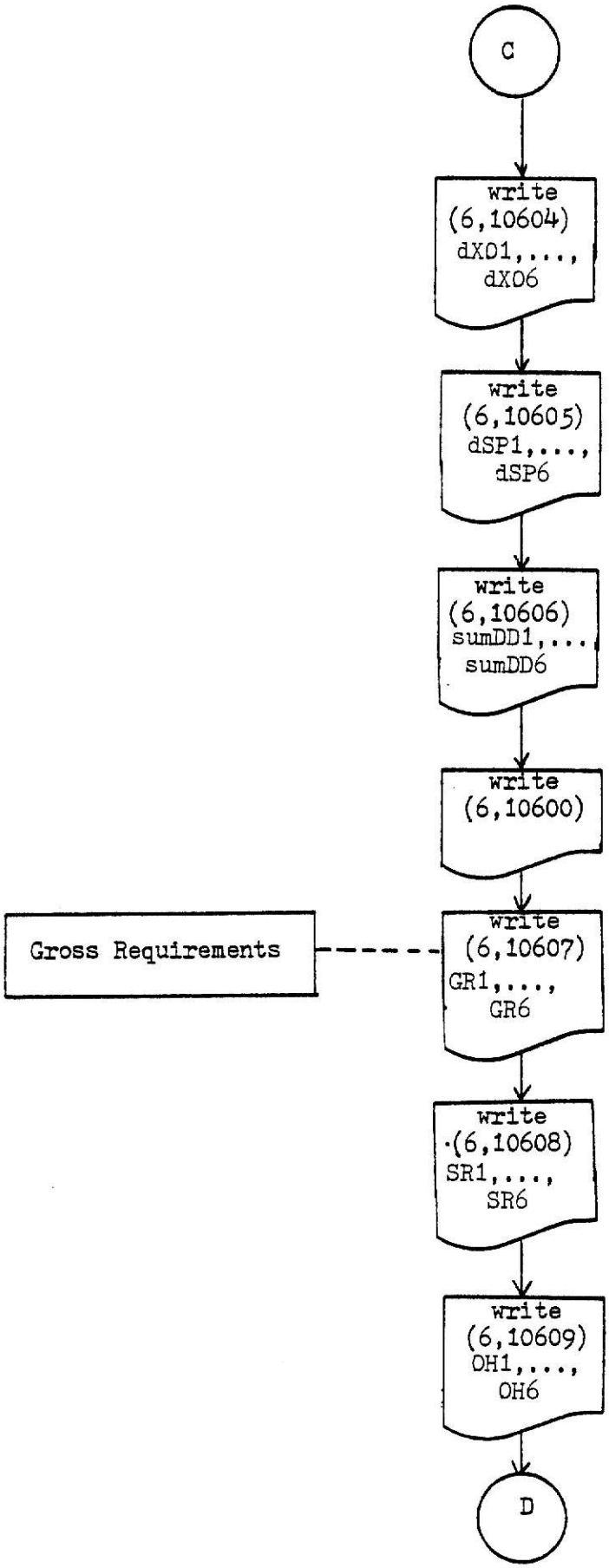
MRP

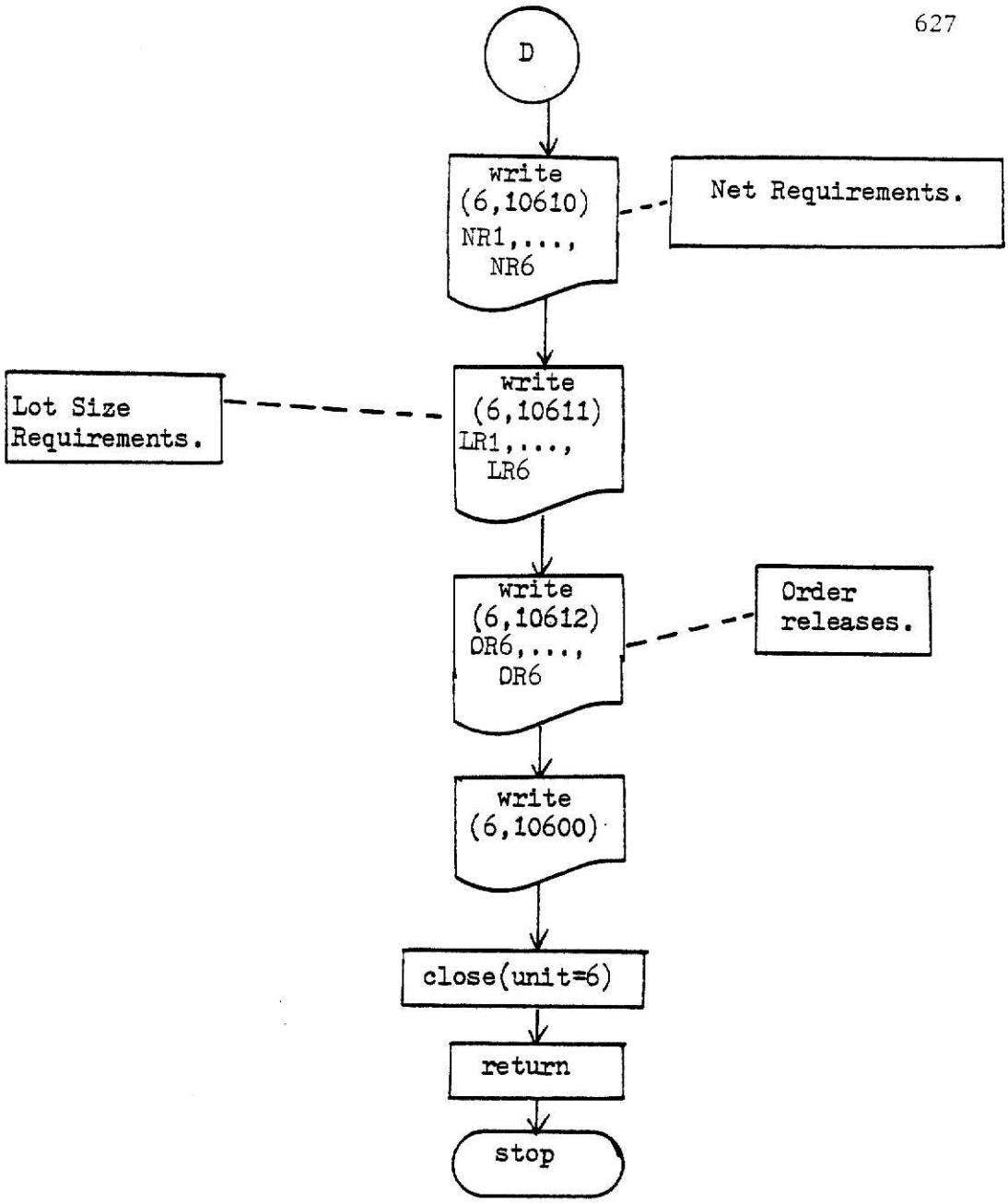
-Subroutine SIX-
Arguments: orderperiod,
orderdate,ordersize,n,i




A
i=i+1
7
i
read(...)
period2,..
.,OR2
i=i+1
7
i
read(...)
period3,..
.,OR3
i=i+1
7
i
read(...)
period4,.
..,OR4
i=i+1
7
i
read(...)
period5,.
..,OR5
B

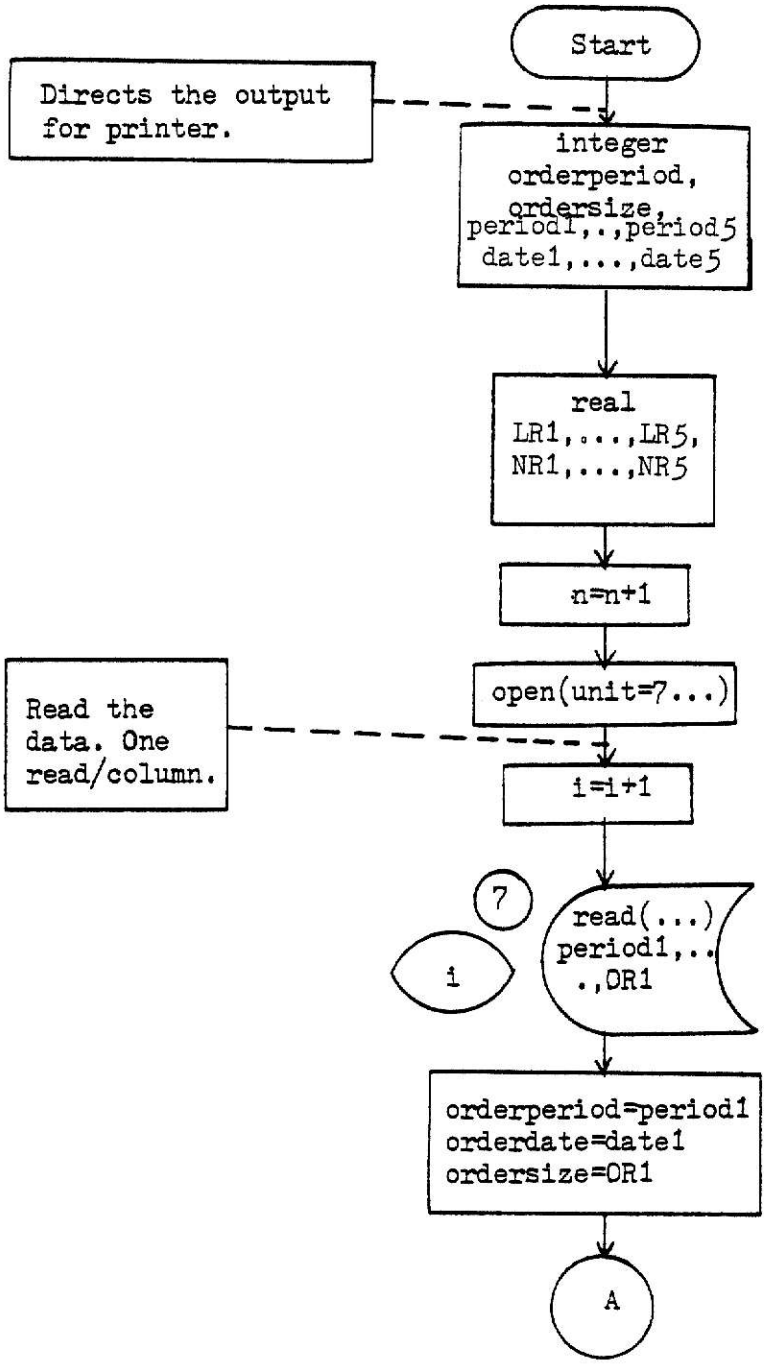

B
i=i+1
7
i
read(...)
period6,..
..,OR6
close(unit=7)
open(unit=6...)
Write out the data transposed.
write (6,10600)
Time periods.
write (6,10601) period1,.., period6
write (6,10602) date1,..., date6
write (6,10600)
Demands.
write (6,10603) dMPS1,..., dMPS6
C

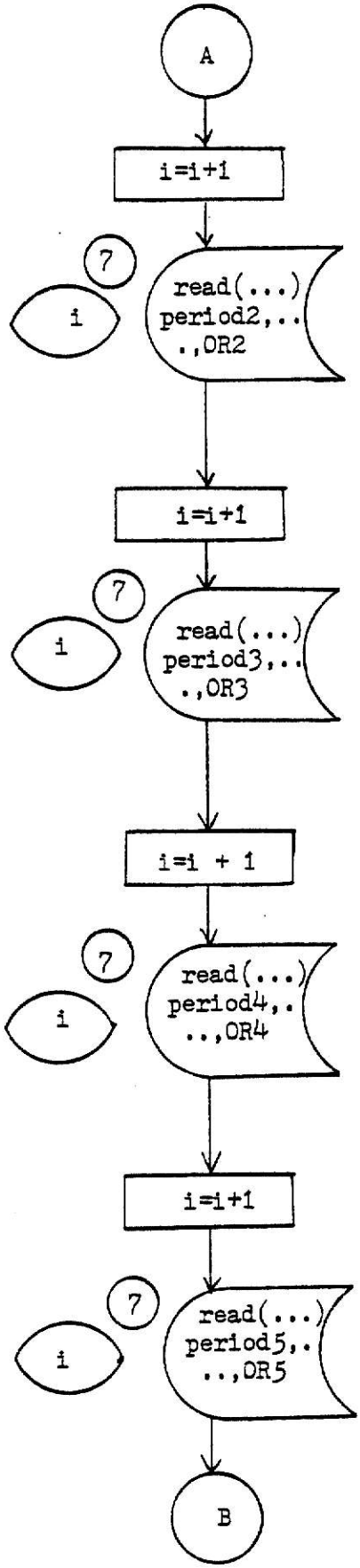

C
write (6,10604) dX01,..., dX06
write (6,10605) dSP1,..., dSP6
write (6,10606) sumDD1,... sumDD6
write (6,10600)
Gross Requirements
write (6,10607) GR1,..., GR6
write (6,10608) SR1,..., SR6
write (6,10609) OH1,..., OH6
D

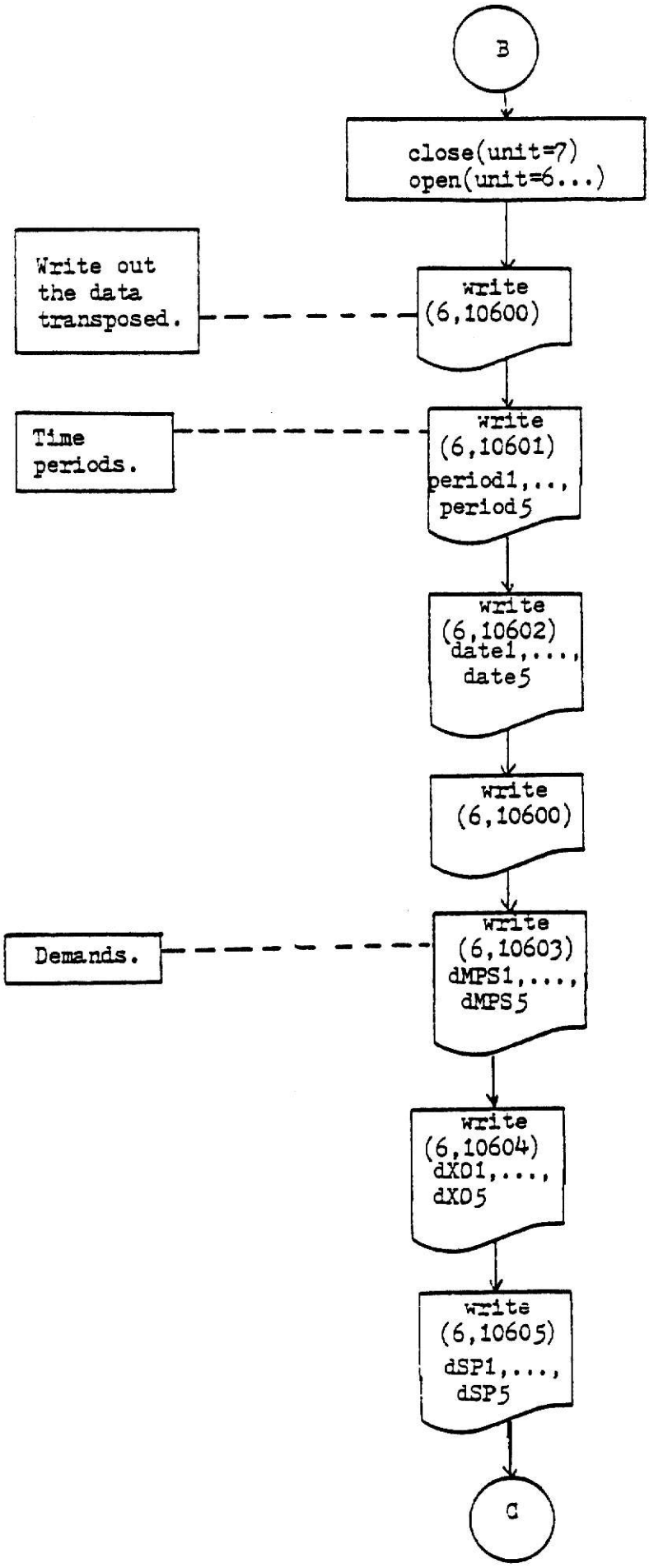

D
write
(6,10610)
NR1,...,
NR6
Net Requirements.
write
(6,10611)
LR1,...,
LR6
Lot Size
Requirements.
write
(6,10612)
OR6,...,
OR6
Order
releases.
write
(6,10600)
close(unit=6)
return
stop

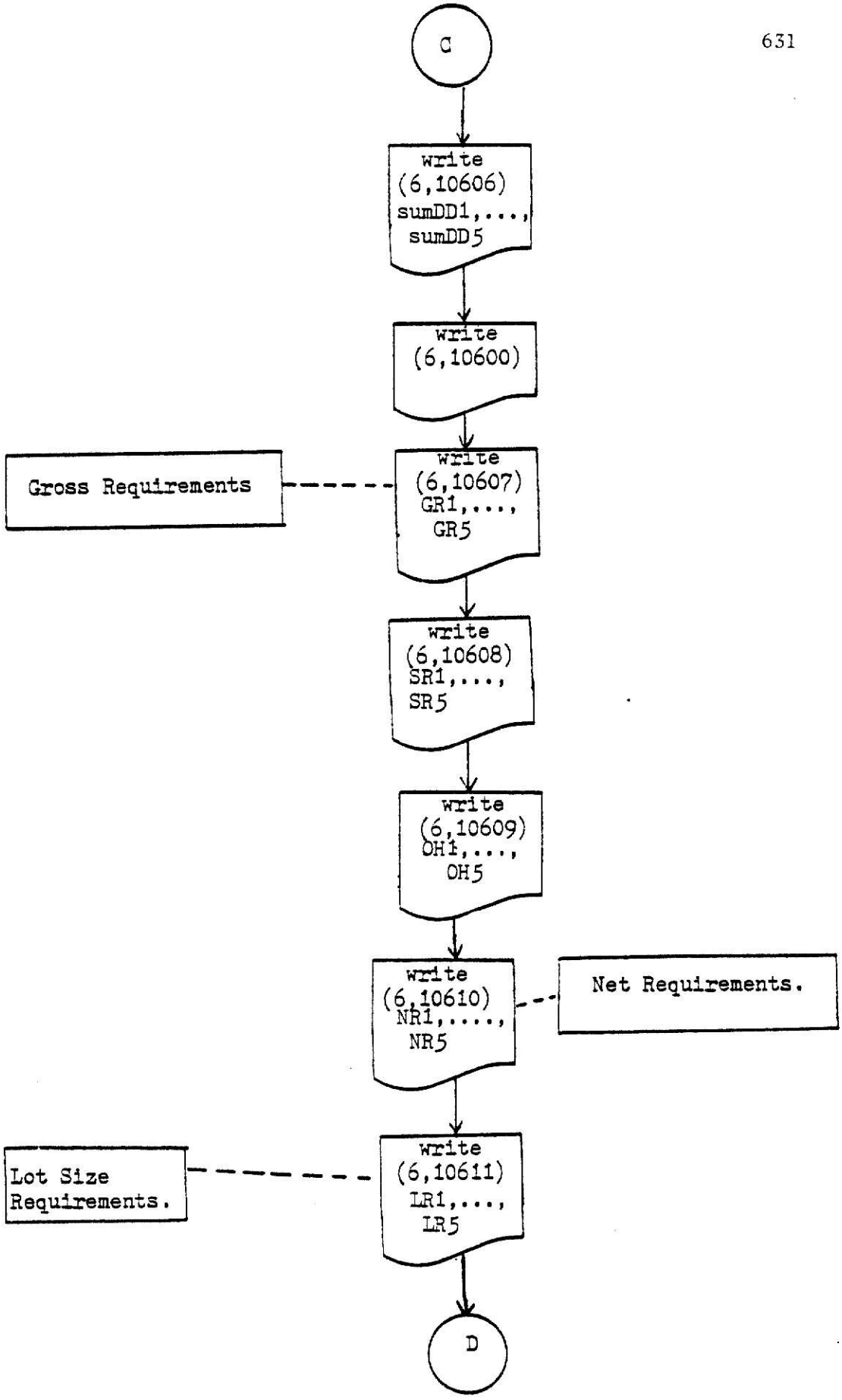
MRP

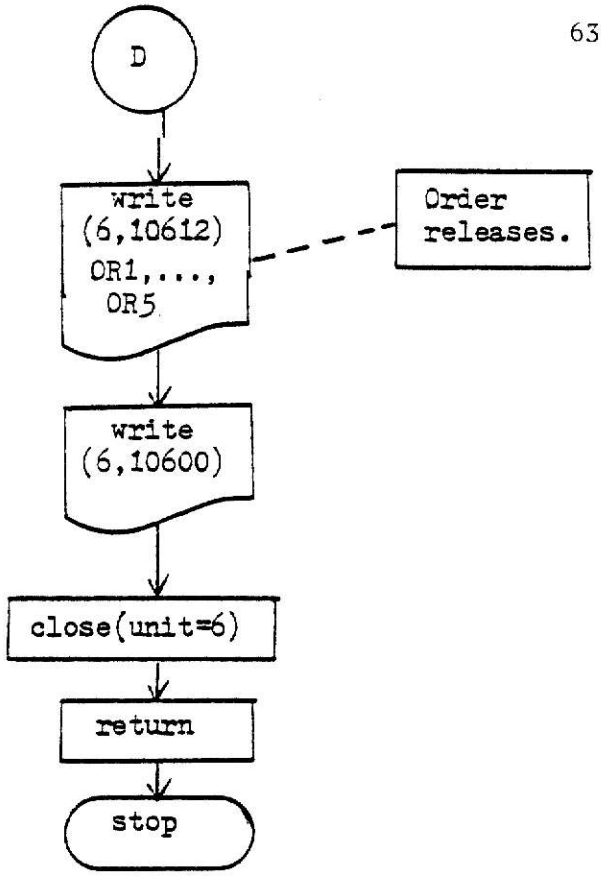
-Subroutine FIVE-
Arguments: orderperiod, orderdate,ordersize,n,i




A
i=i+1
7
i
read(...)
period2,..
.,OR2
i=i+1
7
i
read(...)
period3,..
.,OR3
i=i + 1
7
i
read(...)
period4,.
..,OR4
i=i+1
7
i
read(...)
period5,.
..,OR5
B

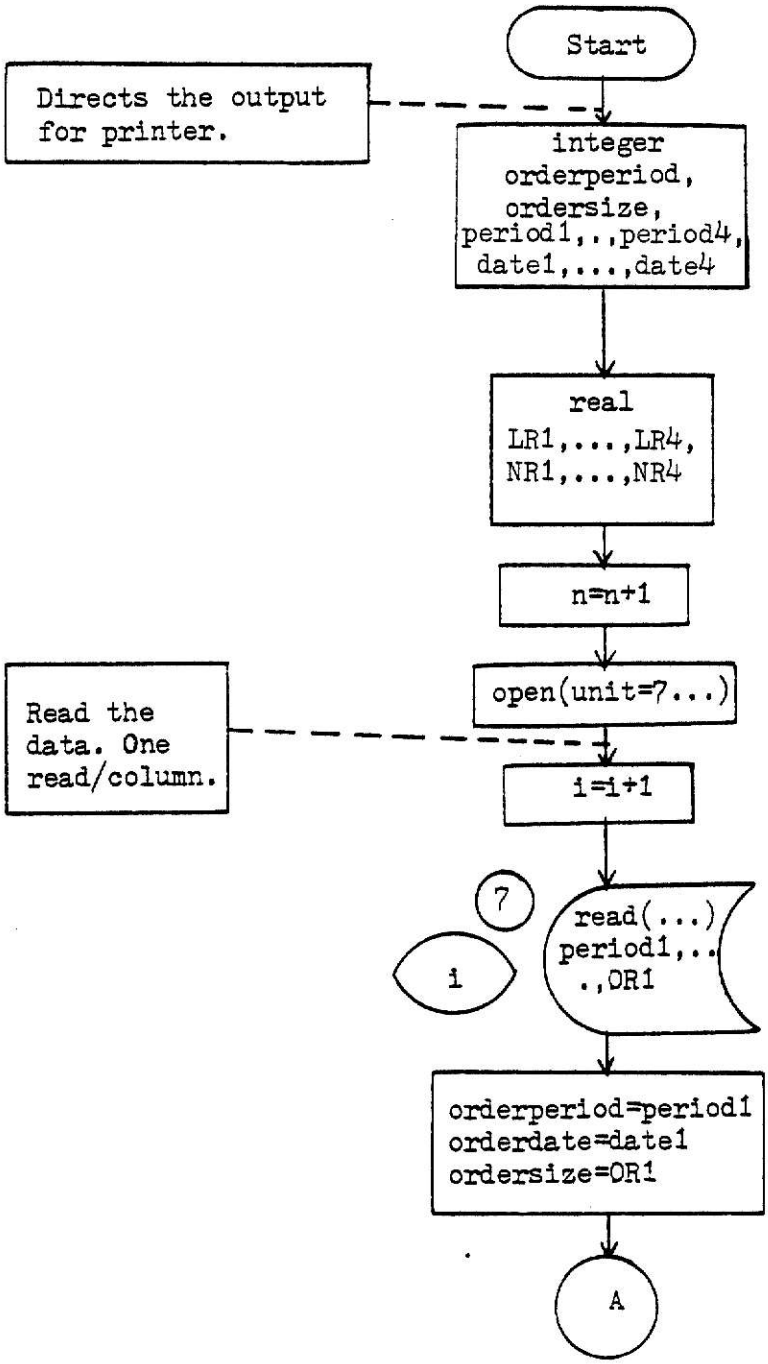

B
close(unit=7)
open(unit=6...)
Write out the data transposed.
write (6,10600)
Time periods.
write (6,10601) period1,.., period5
write (6,10602) date1,..., date5
write (6,10600)
Demands.
write (6,10603) dMPS1,..., dMPS5
write (6,10604) dX01,..., dX05
write (6,10605) dSP1,..., dSP5
C

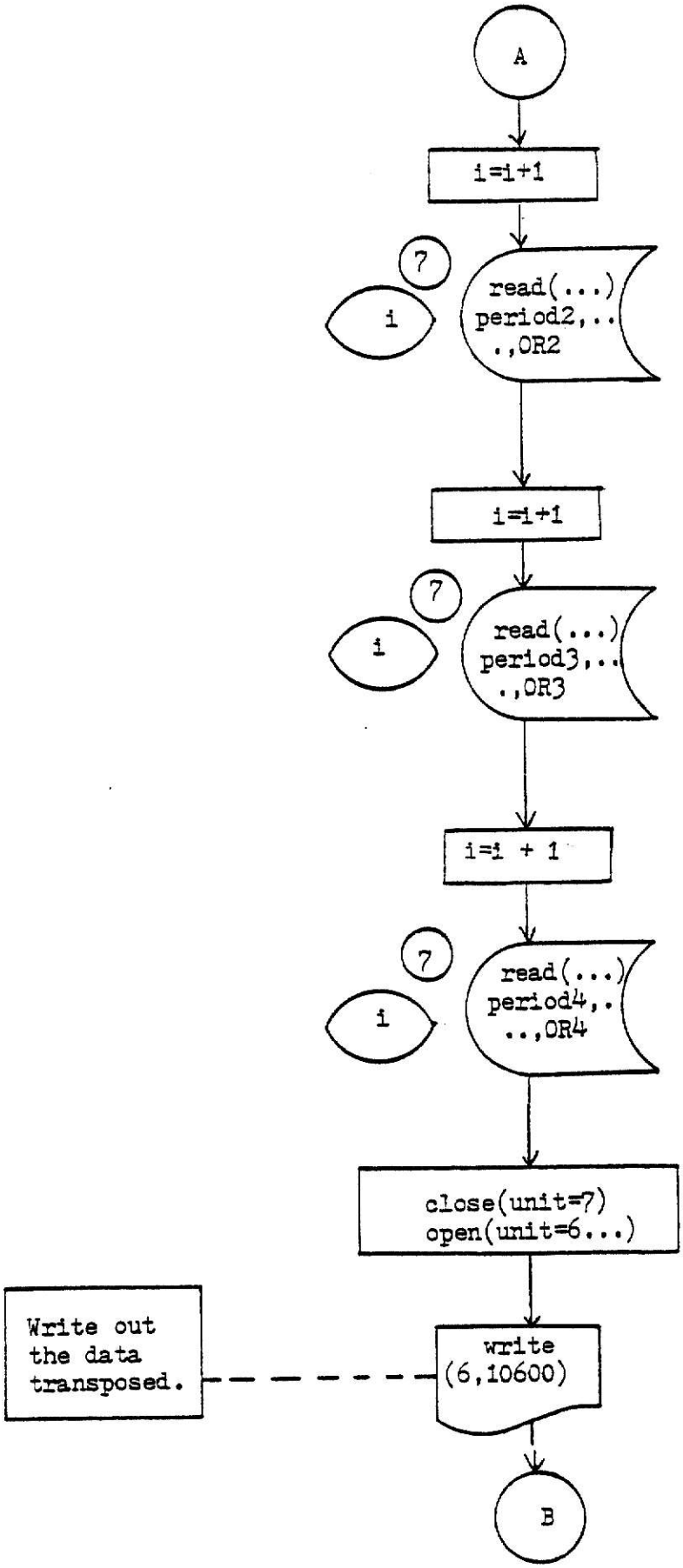

C
write (6,10606) sumDD1,..., sumDD5
write (6,10600)
write (6,10607) GR1,..., GR5
Gross Requirements
write (6,10608) SR1,..., SR5
write (6,10609) OH1,..., OH5
write (6,10610) NR1,...., NR5
Net Requirements.
write (6,10611) LR1,..., LR5
Lot Size Requirements.
D

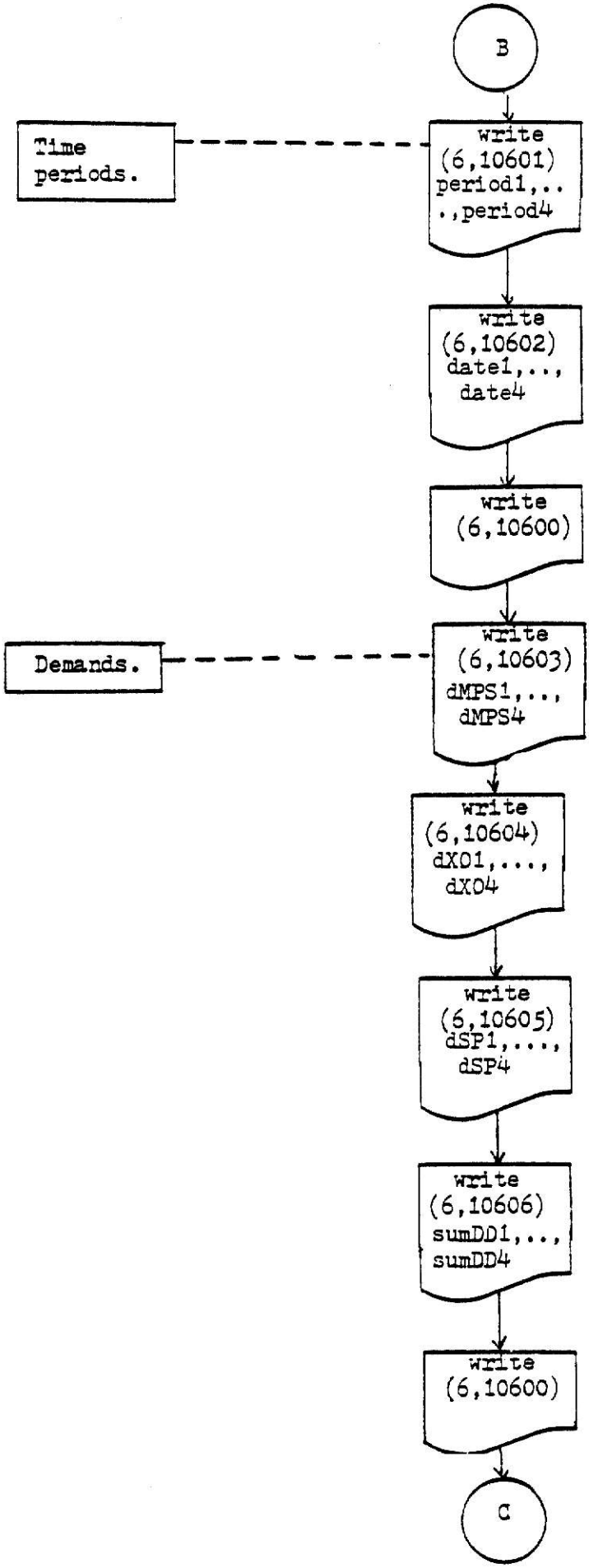

D
write
(6,10612)
OR1,...,
OR5
Order
releases.
write
(6,10600)
close(unit=6)
return
stop

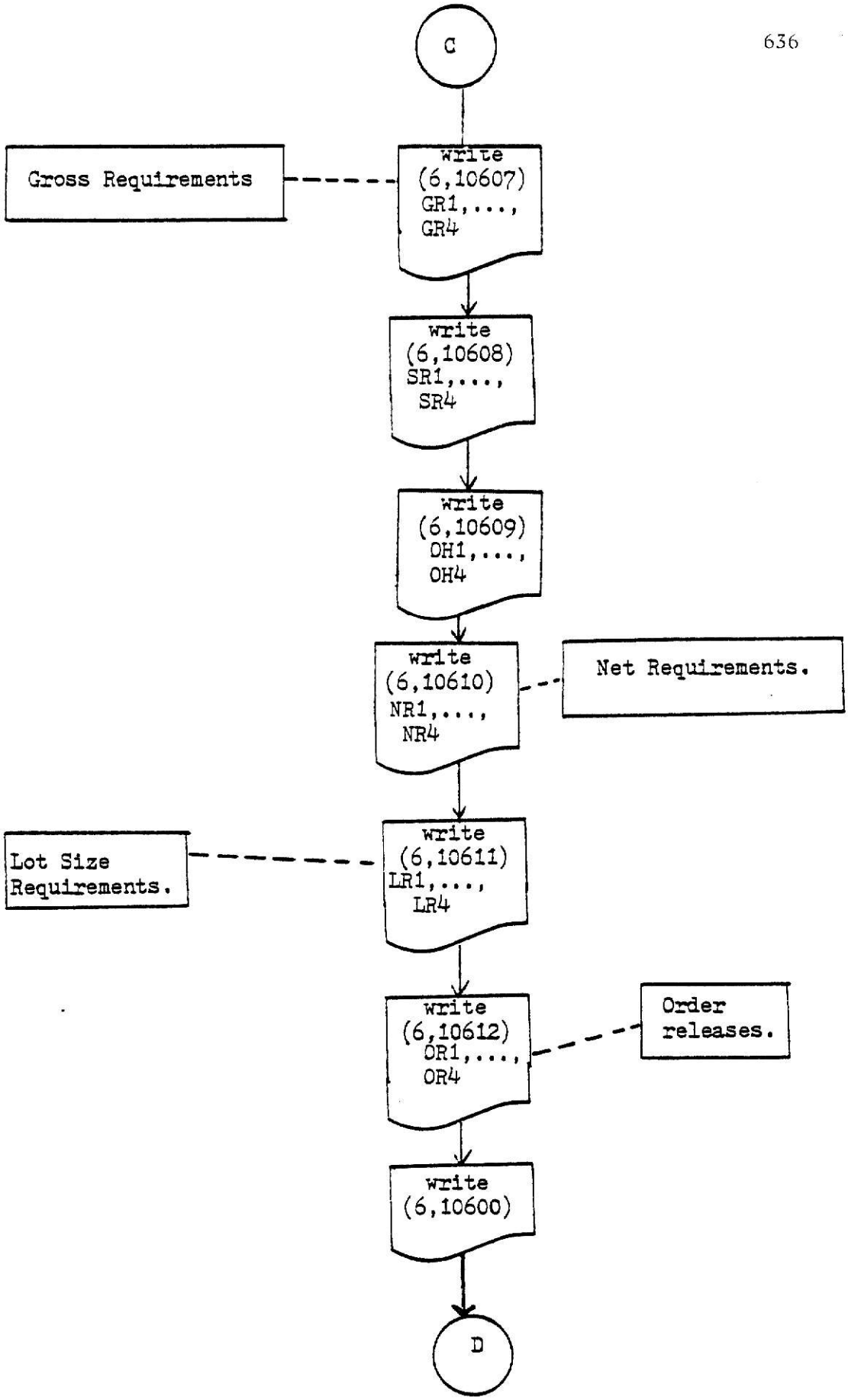
MRP

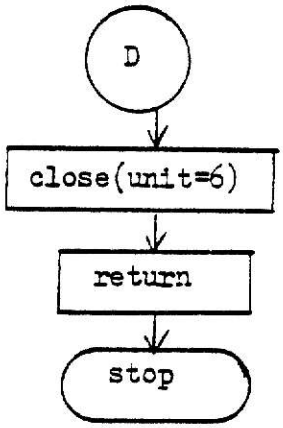
-Subroutine FOUR-
Arguments: orderperiod, orderdate,ordersize,n,i




A
i=i+1
7
i
read(...)
period2,..
.,OR2
i=i+1
7
i
read(...)
period3,..
.,OR3
i=i + 1
7
i
read(...)
period4,.
..,OR4
close(unit=7)
open(unit=6...)
Write out
the data
transposed.
write
(6,10600)
B

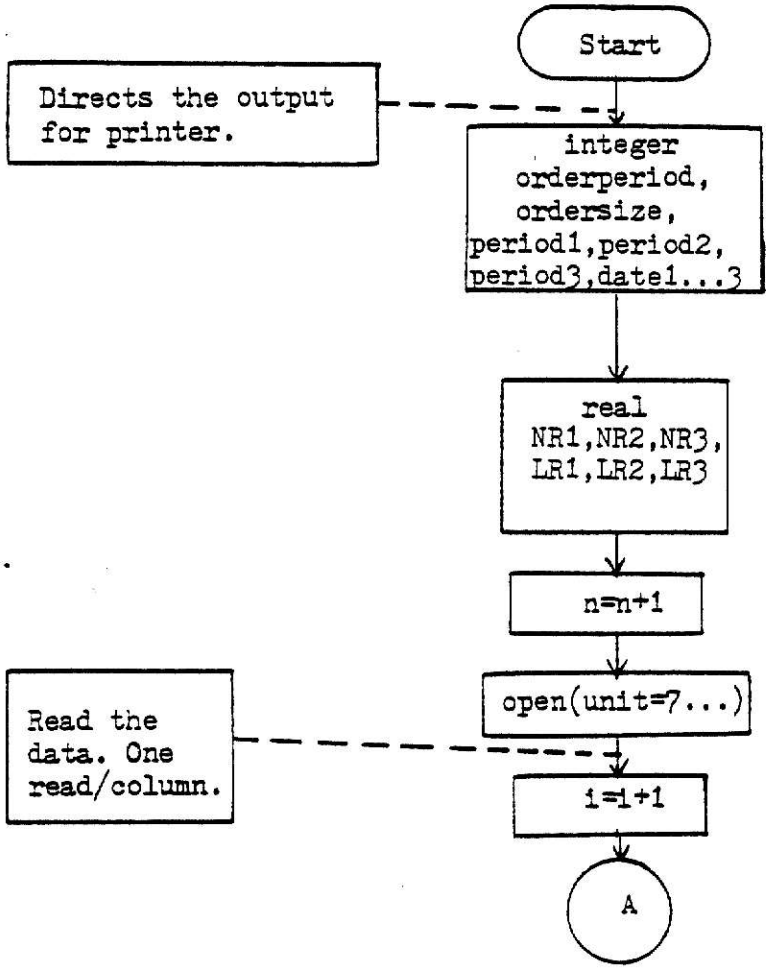

B
Time periods.
write (6,10601) period1,.. .,period4
write (6,10602) date1,.., date4
write (6,10600)
Demands.
write (6,10603) dMPS1,.., dMPS4
write (6,10604) dX01,...., dX04
write (6,10605) dSP1,...., dSP4
write (6,10606) sumDD1,.., sumDD4
write (6,10600)
C

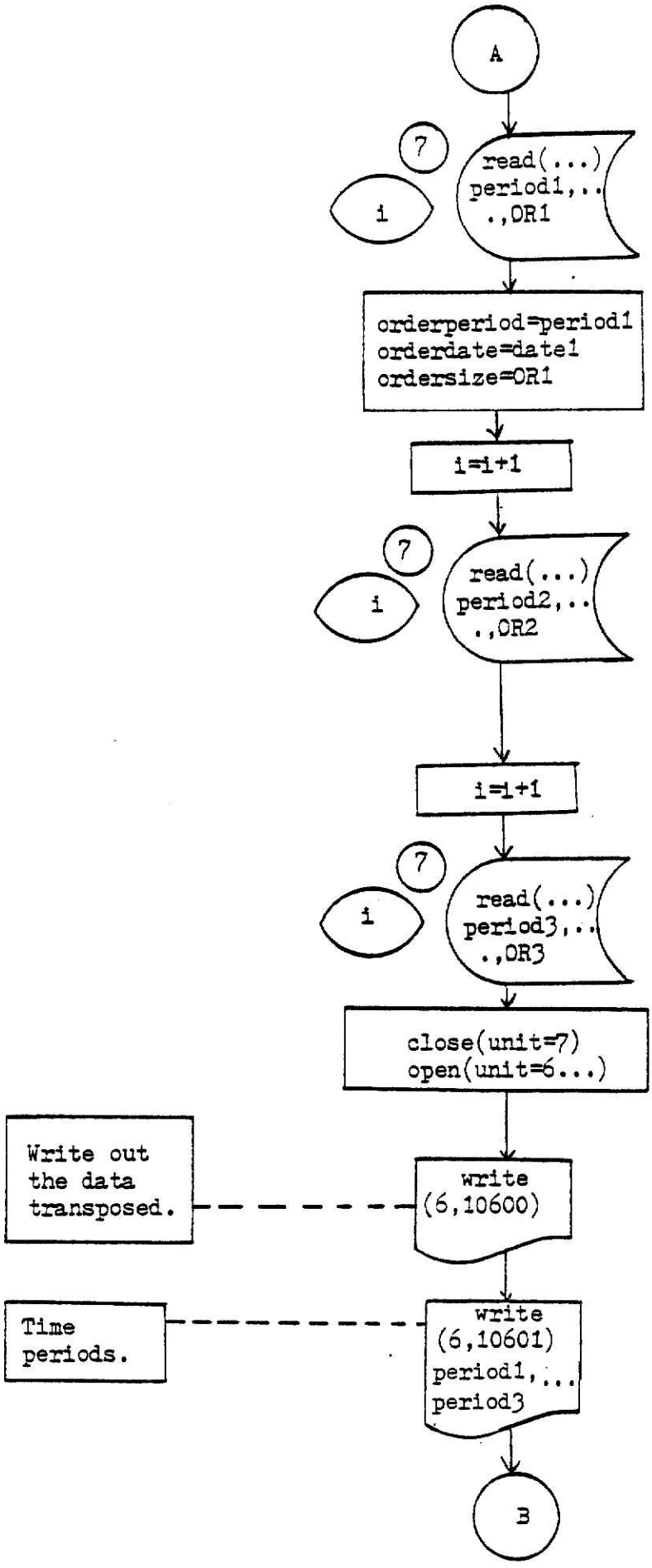

C
write (6,10607) GR1,..., GR4
Gross Requirements
write (6,10608) SR1,..., SR4
write (6,10609) OH1,..., OH4
write (6,10610) NR1,..., NR4
Net Requirements.
write (6,10611) LR1,..., LR4
Lot Size Requirements.
write (6,10612) OR1,..., OR4
Order releases.
write (6,10600)
D

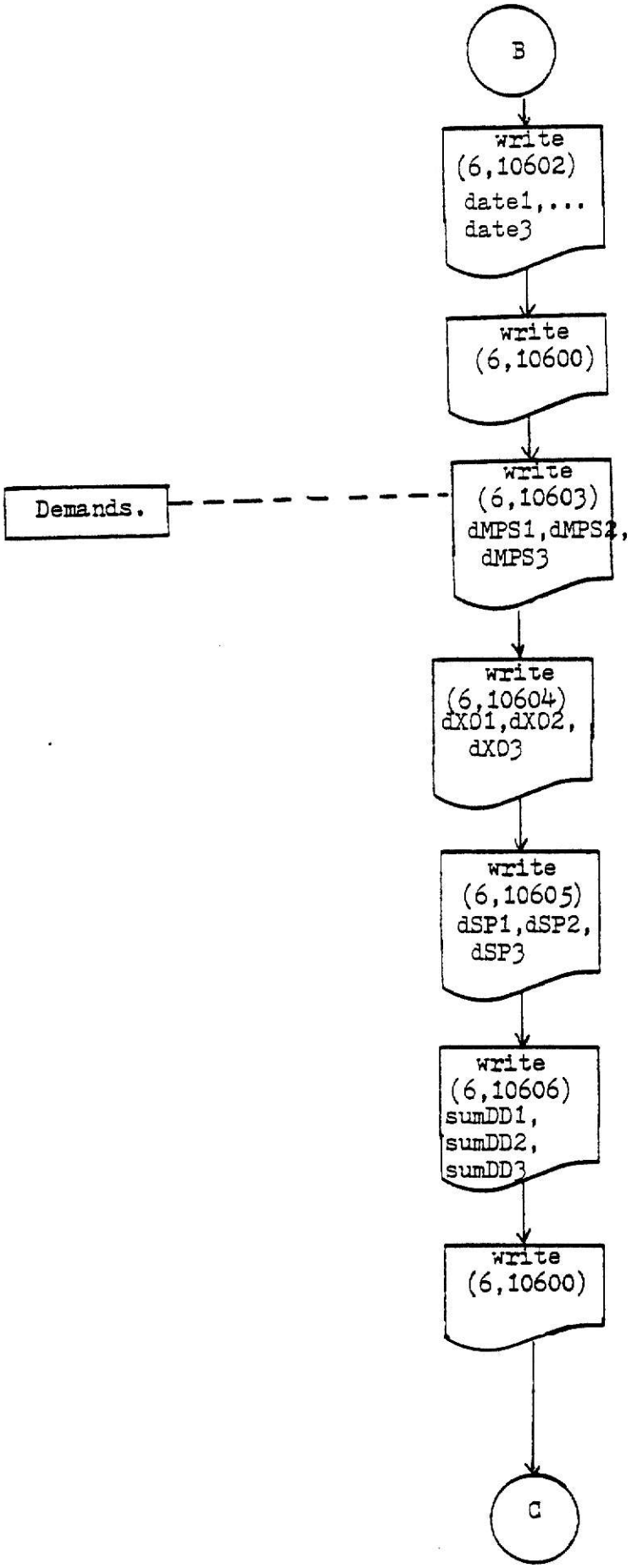


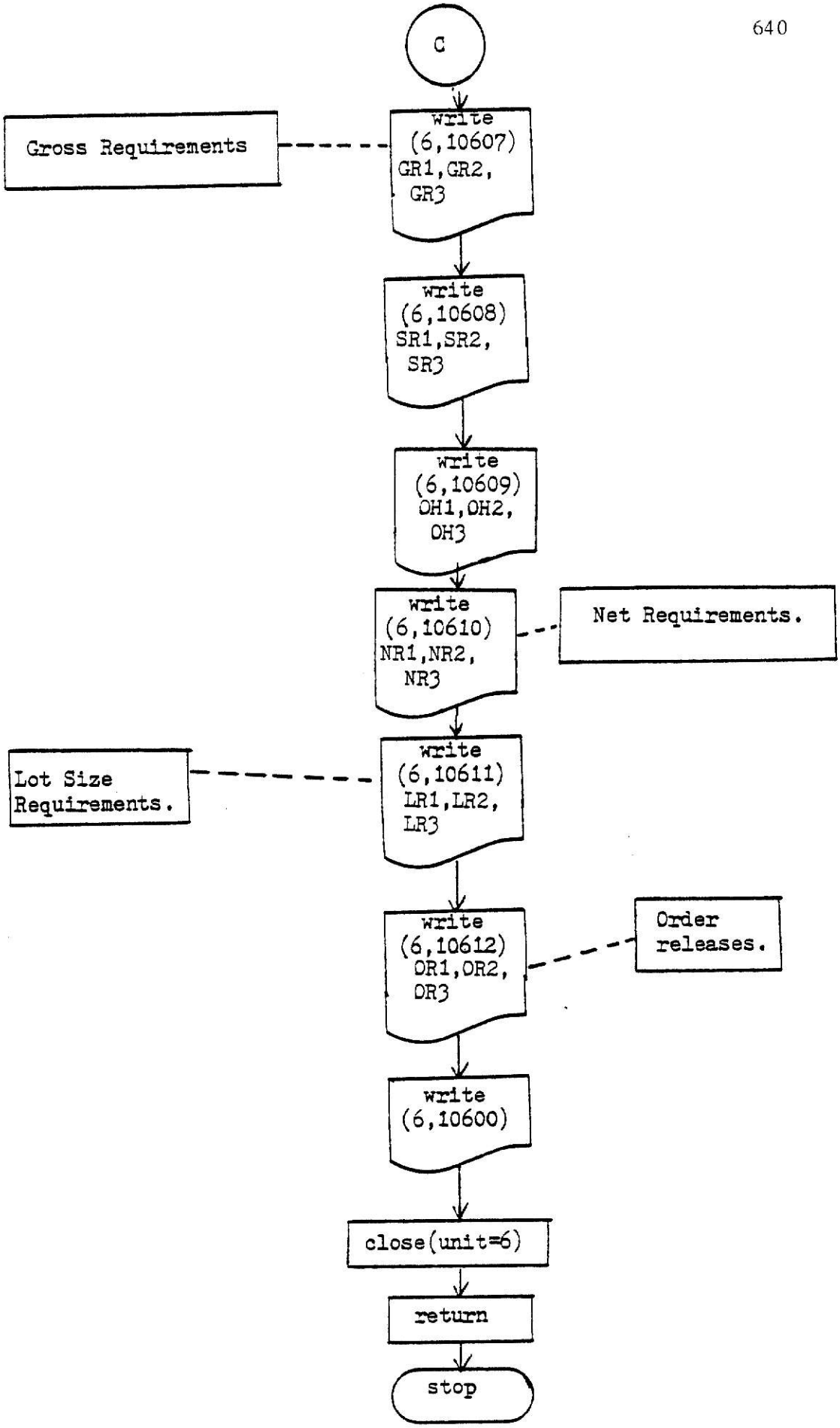
MRP

-Subroutine THREE-
Arguments: orderperiod, orderdate,ordersize,n,i



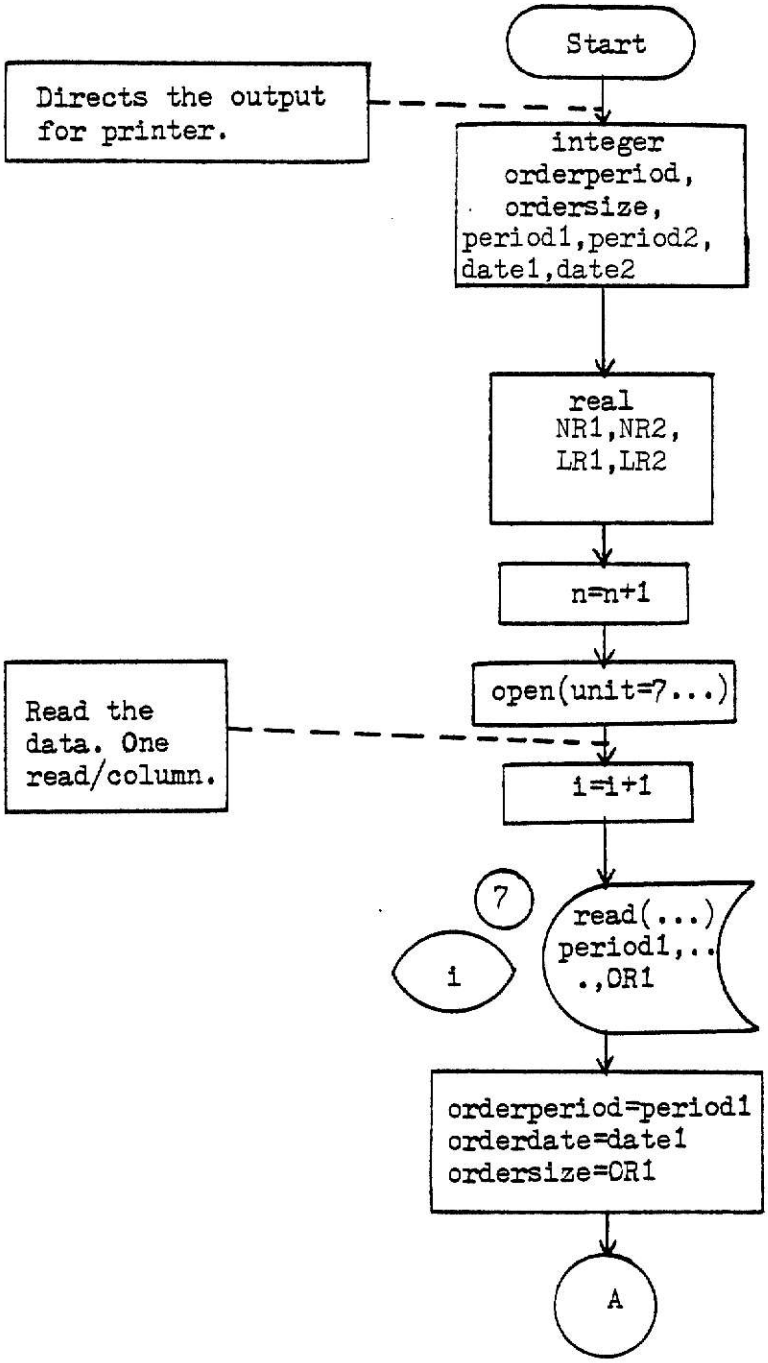

A
7
i
read(...)
period1,..
.,OR1
orderperiod=period1
orderdate=date1
ordersize=OR1
i=i+1
7
i
read(...)
period2,..
.,OR2
i=i+1
7
i
read(...)
period3,..
.,OR3
close(unit=7)
open(unit=6...)
Write out the data transposed.
write (6,10600)
Time periods.
write (6,10601) period1,... period3
B


B
write (6,10602) date1,... date3
write (6,10600)
write (6,10603) dMPS1,dMPS2, dMPS3
Demands.
write (6,10604) dX01,dX02, dX03
write (6,10605) dSP1,dSP2, dSP3
write (6,10606) sumDD1, sumDD2, sumDD3
write (6,10600)
C


C
write (6,10607) GR1,GR2, GR3
Gross Requirements
write (6,10608) SR1,SR2, SR3
write (6,10609) OH1,OH2, OH3
write (6,10610) NR1,NR2, NR3
Net Requirements.
write (6,10611) LR1,LR2, LR3
Lot Size Requirements.
write (6,10612) OR1,OR2, OR3
Order releases.
write (6,10600)
close(unit=6)
return
stop

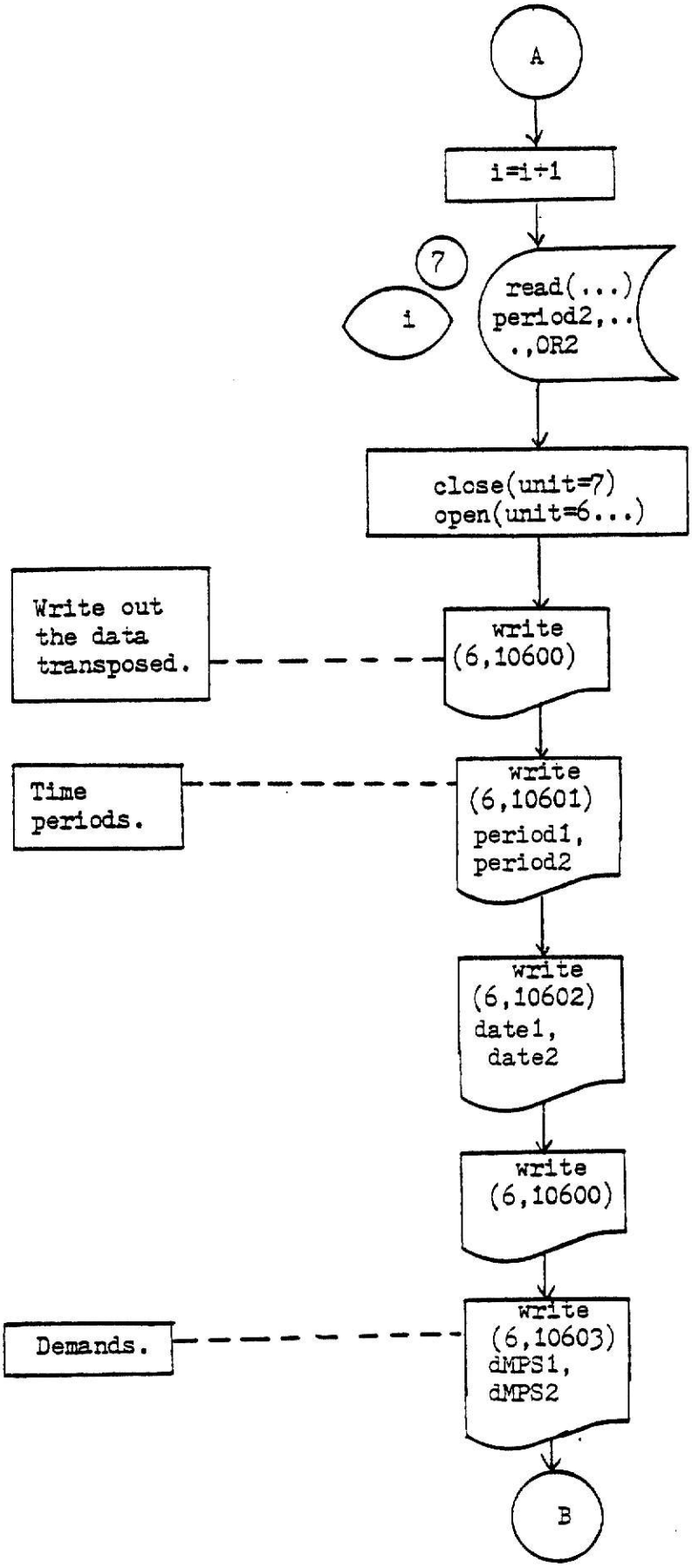
MRP

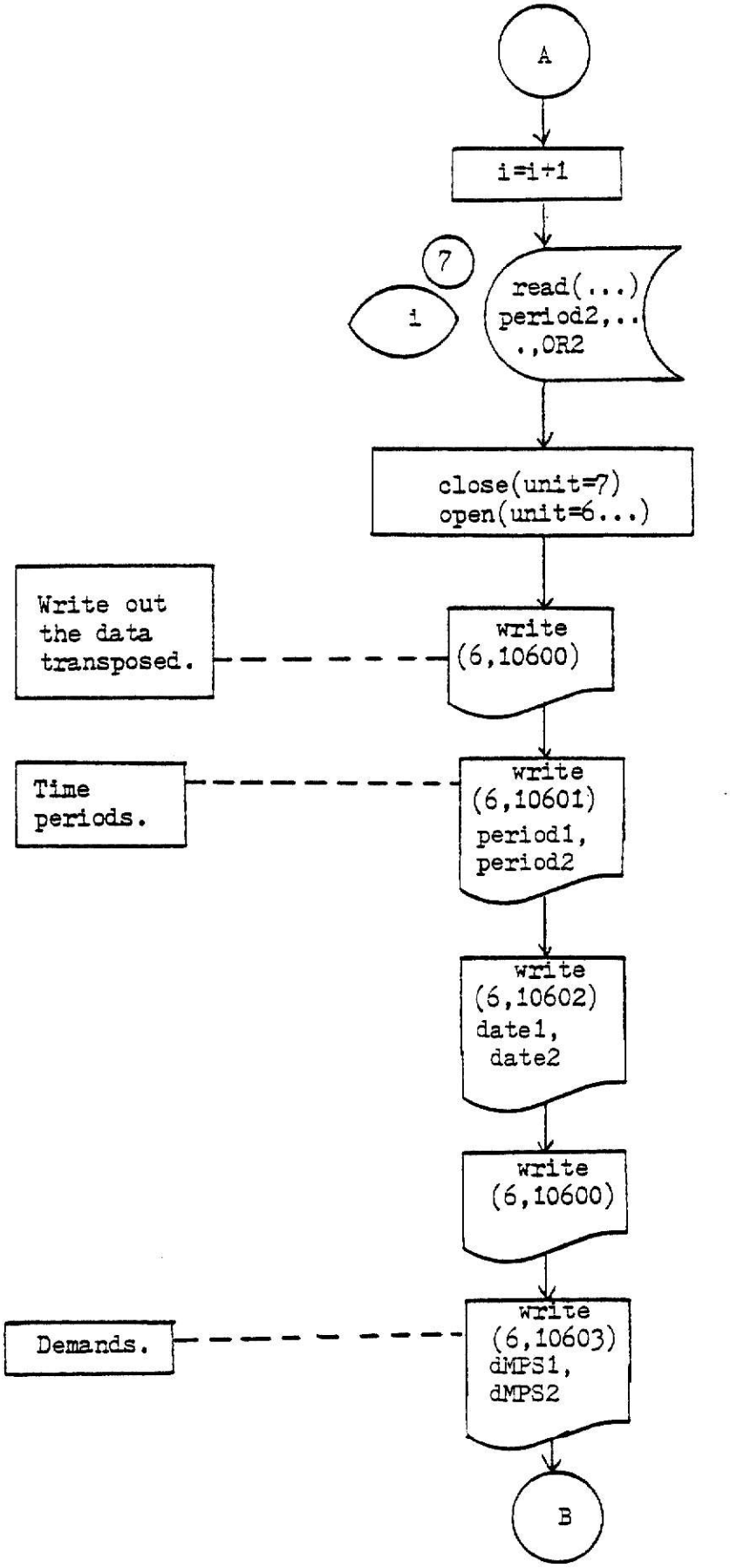
-Subroutine TWO-
Arguments: orderperiod,
orderdate,ordersize,n,i

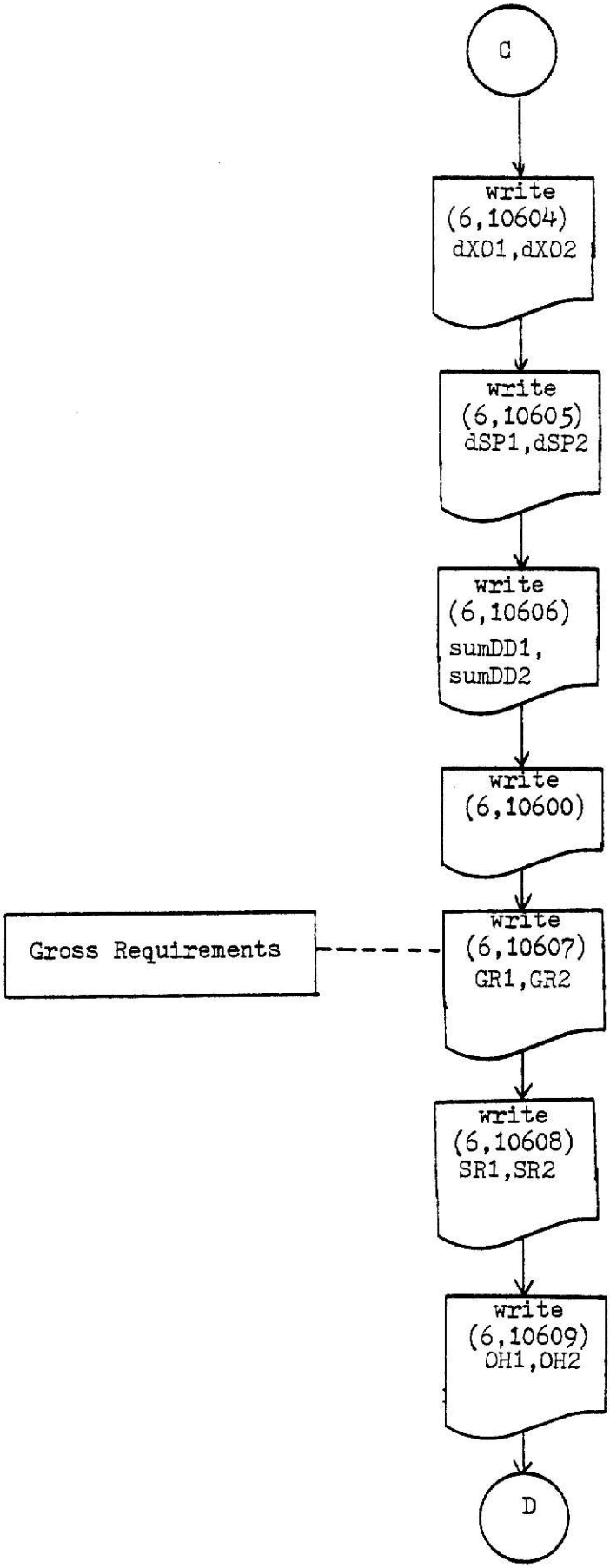


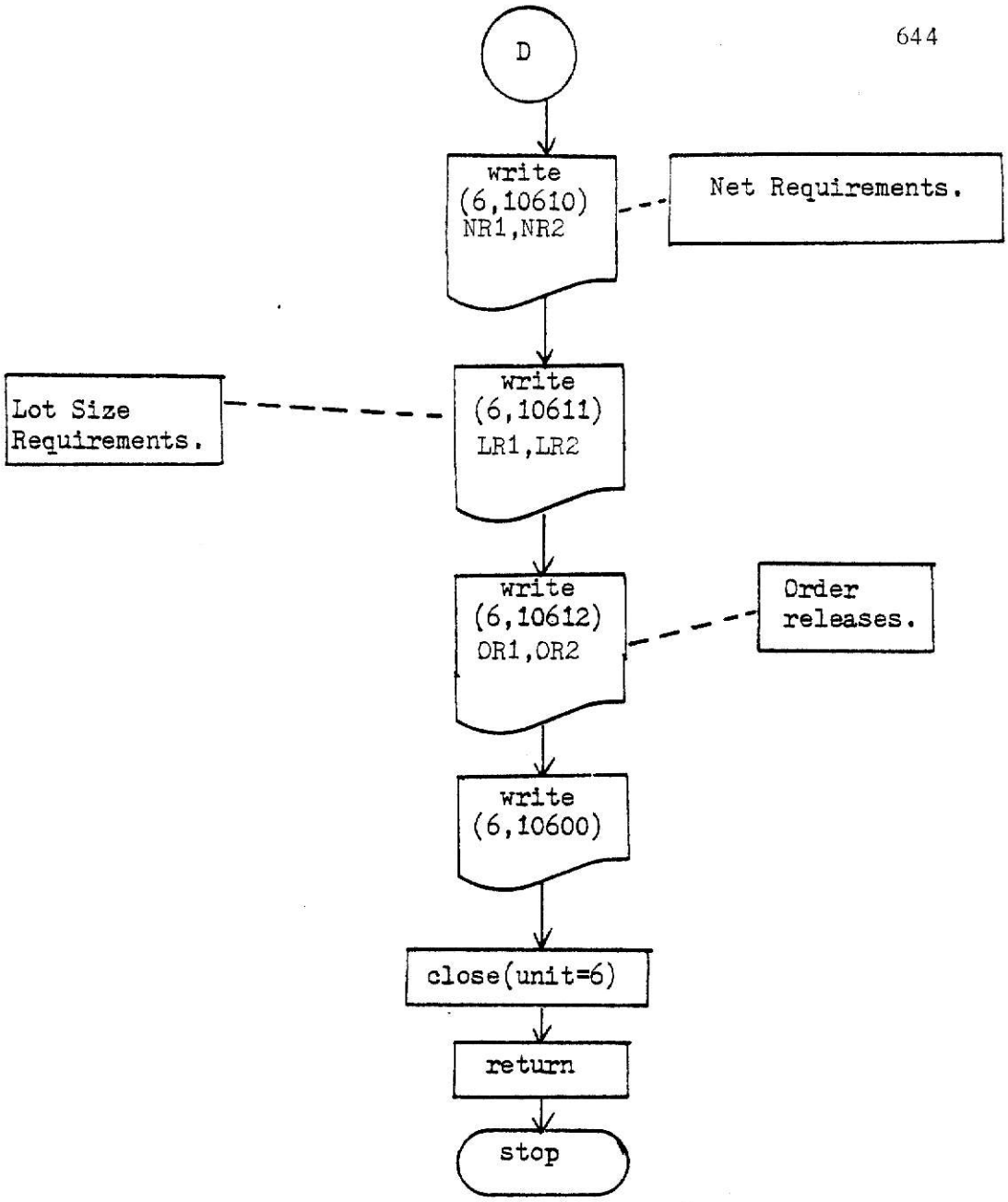
THIS BOOK WAS BOUND WITH TWO PAGES NUMBERED 642. THESE PAGES ARE THE SAME.

THIS IS AS RECEIVED FROM CUSTOMER.


A
i=i+1
7
i
read(...)
period2,..
.,OR2
close(unit=7)
open(unit=6...)
Write out the data transposed.
write (6,10600)
Time periods.
write (6,10601) period1, period2
write (6,10602) date1, date2
write (6,10600)
Demands.
write (6,10603) dMPS1, dMPS2
B

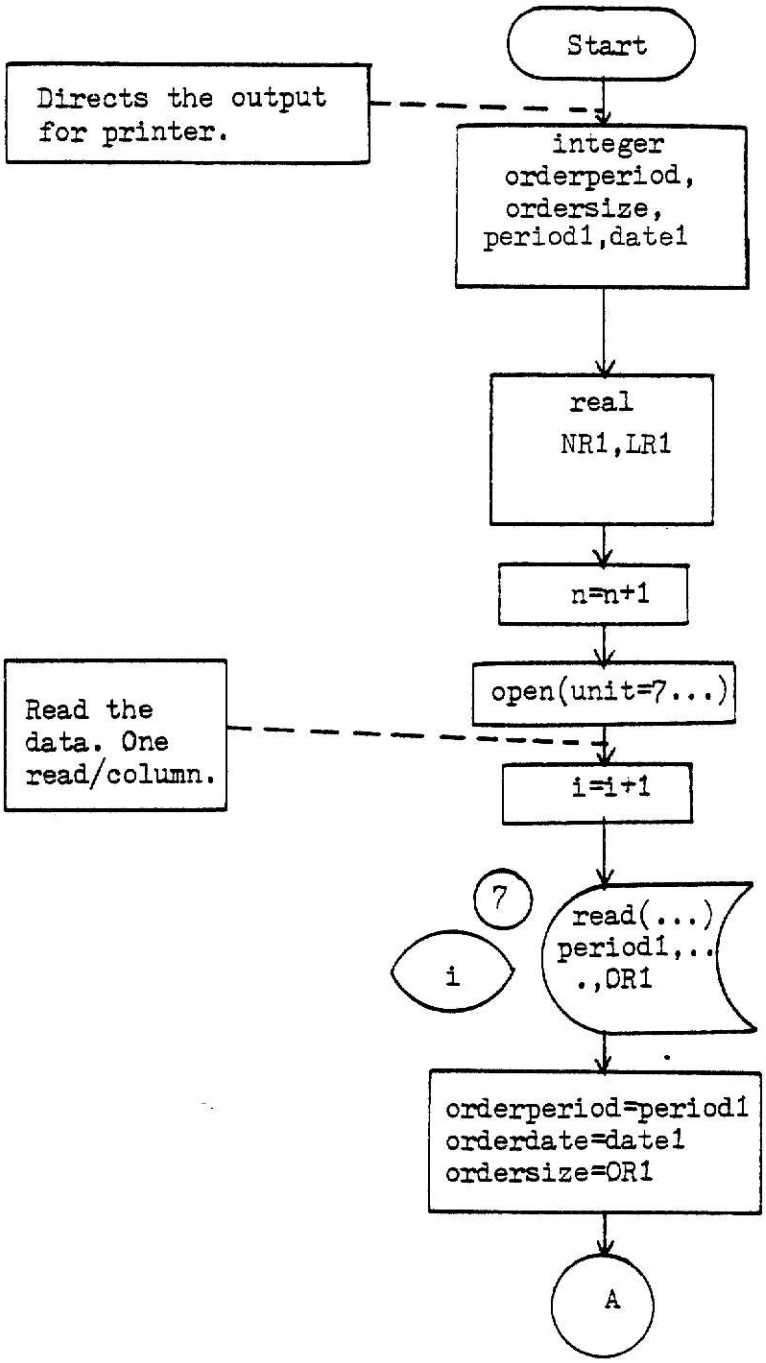

A
i=i+1
7
i
read(...)
period2,..
.,OR2
close(unit=7)
open(unit=6...)
Write out
the data
transposed.
write
(6,10600)
Time
periods.
write
(6,10601)
period1,
period2
write
(6,10602)
date1,
date2
write
(6,10600)
Demands.
write
(6,10603)
dMPS1,
dMPS2
B

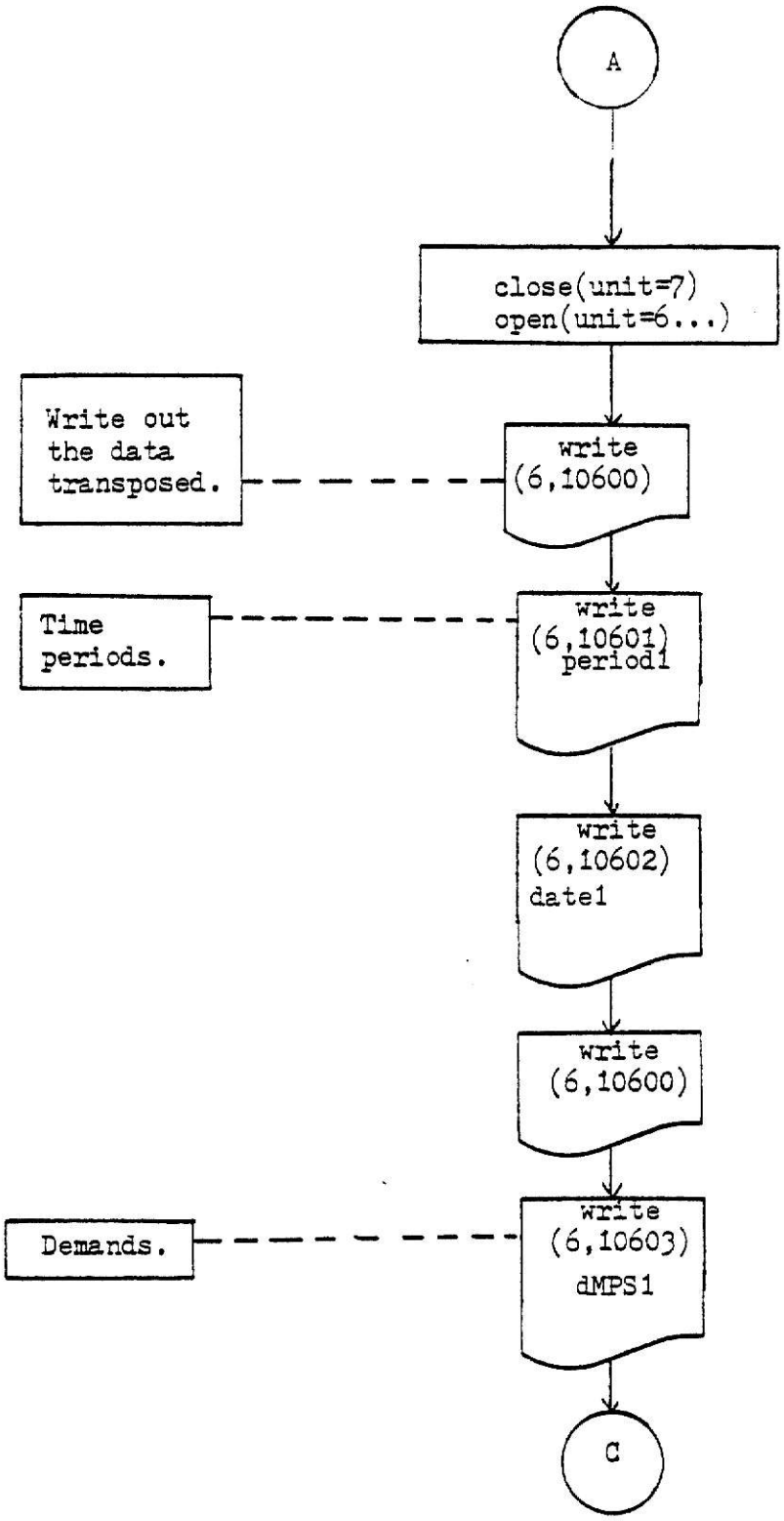

C
write
(6,10604)
dXO1,dXO2
write
(6,10605)
dSP1,dSP2
write
(6,10606)
sumDD1,
sumDD2
write
(6,10600)
Gross Requirements
write
(6,10607)
GR1,GR2
write
(6,10608)
SR1,SR2
write
(6,10609)
OH1,OH2
D

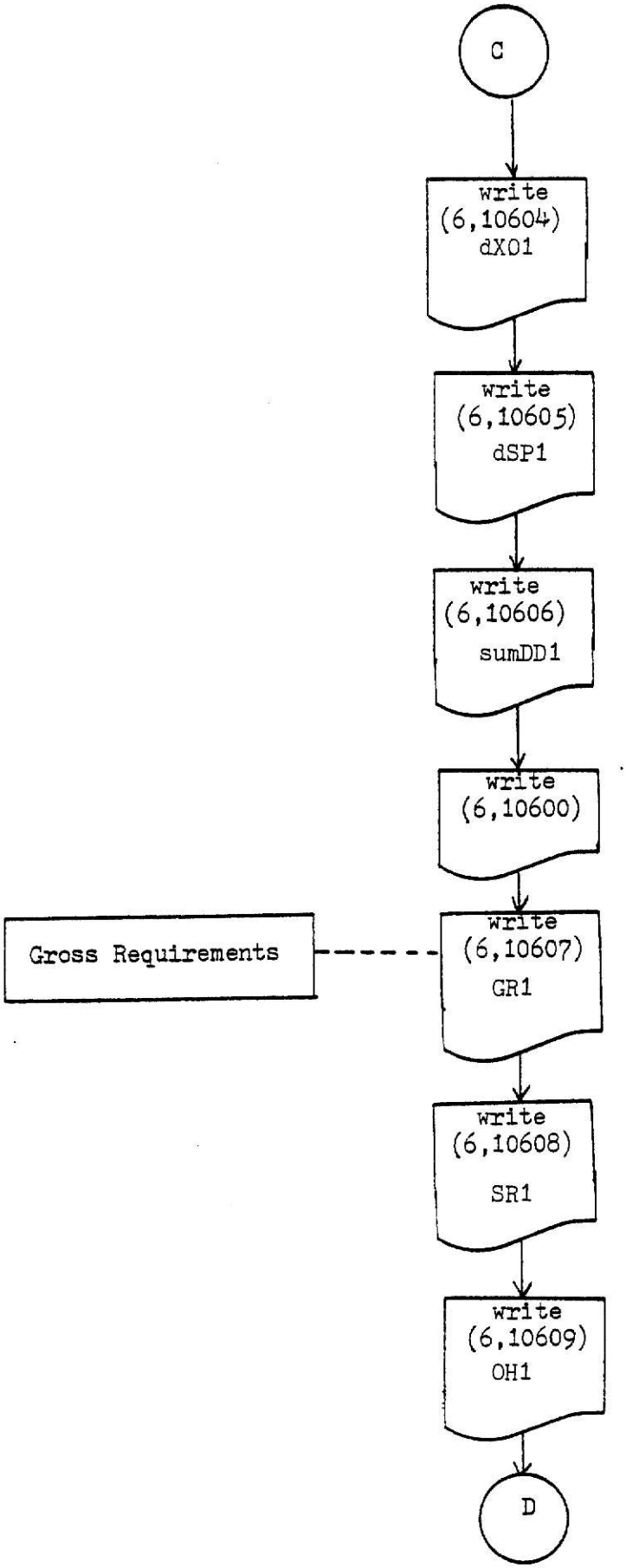

D
write
(6,10610)
NR1,NR2
Net Requirements.
write
(6,10611)
LR1,LR2
Lot Size
Requirements.
write
(6,10612)
OR1,OR2
Order
releases.
write
(6,10600)
close(unit=6)
return
stop

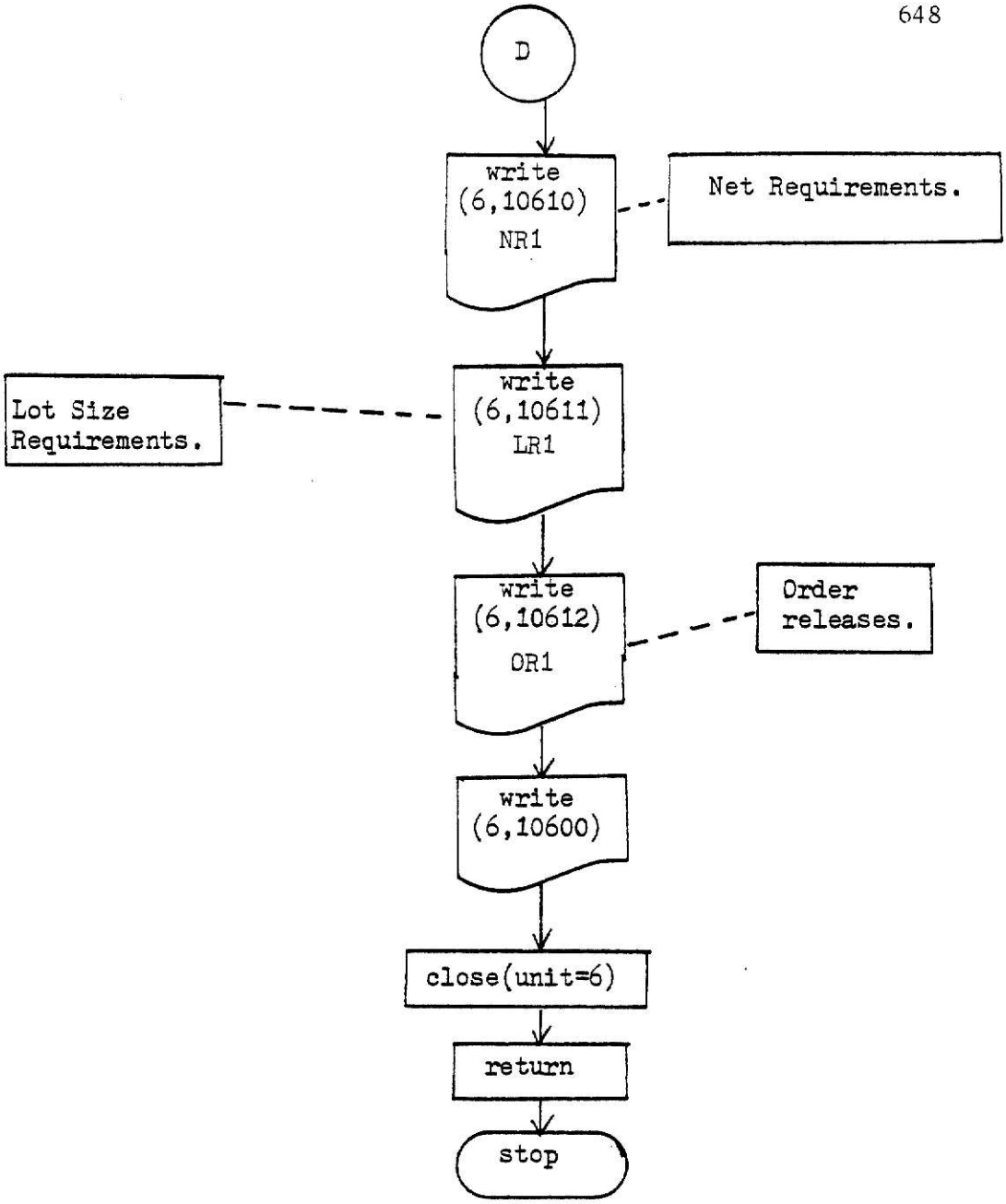
MRP

-Subroutine ONE-
Arguments: orderperiod, orderdate,ordersize,n,i



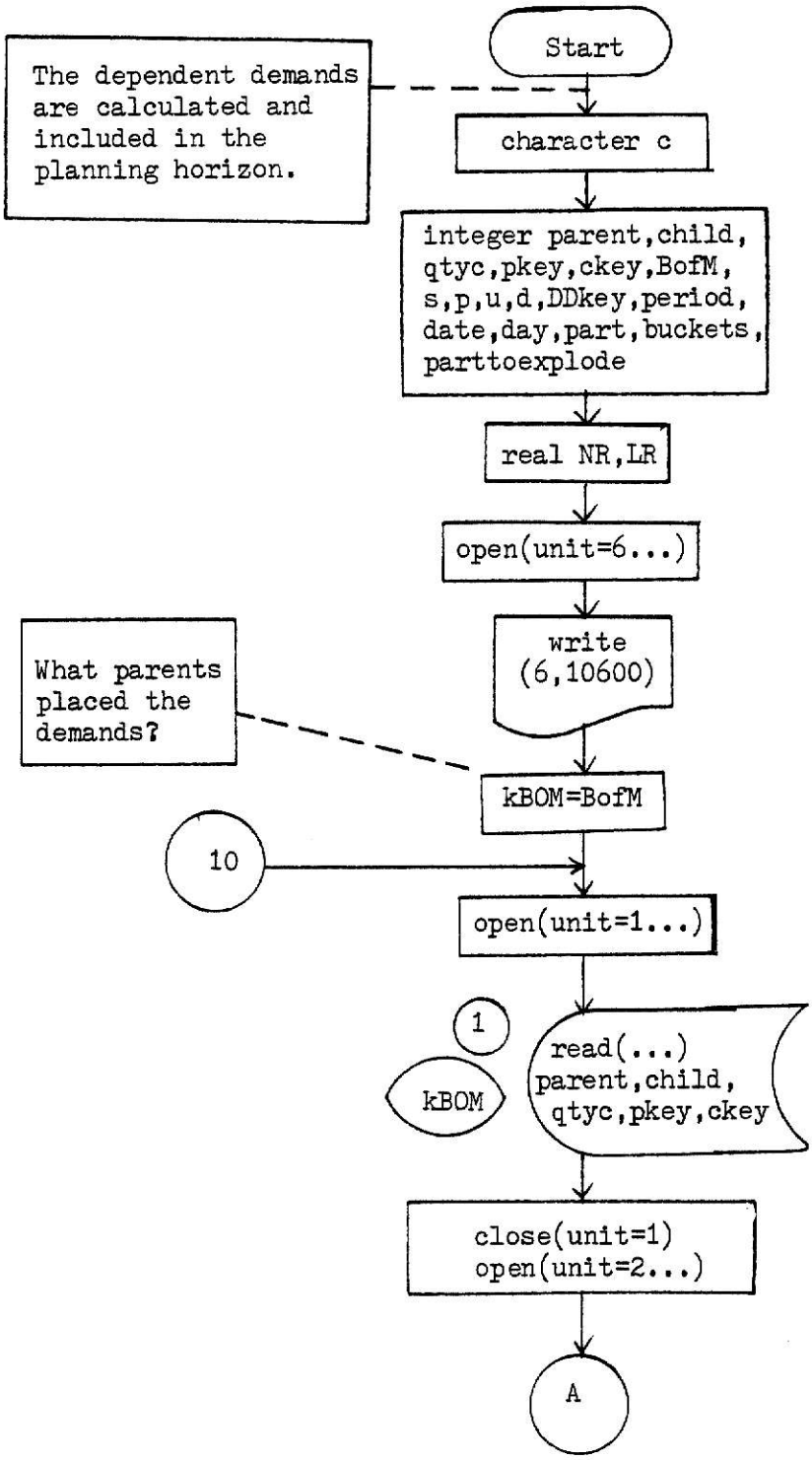

A
close(unit=7)
open(unit=6...)
Write out the data transposed.
write (6,10600)
Time periods.
write (6,10601) period1
write (6,10602) date1
write (6,10600)
Demands.
write (6,10603) dMPS1
C

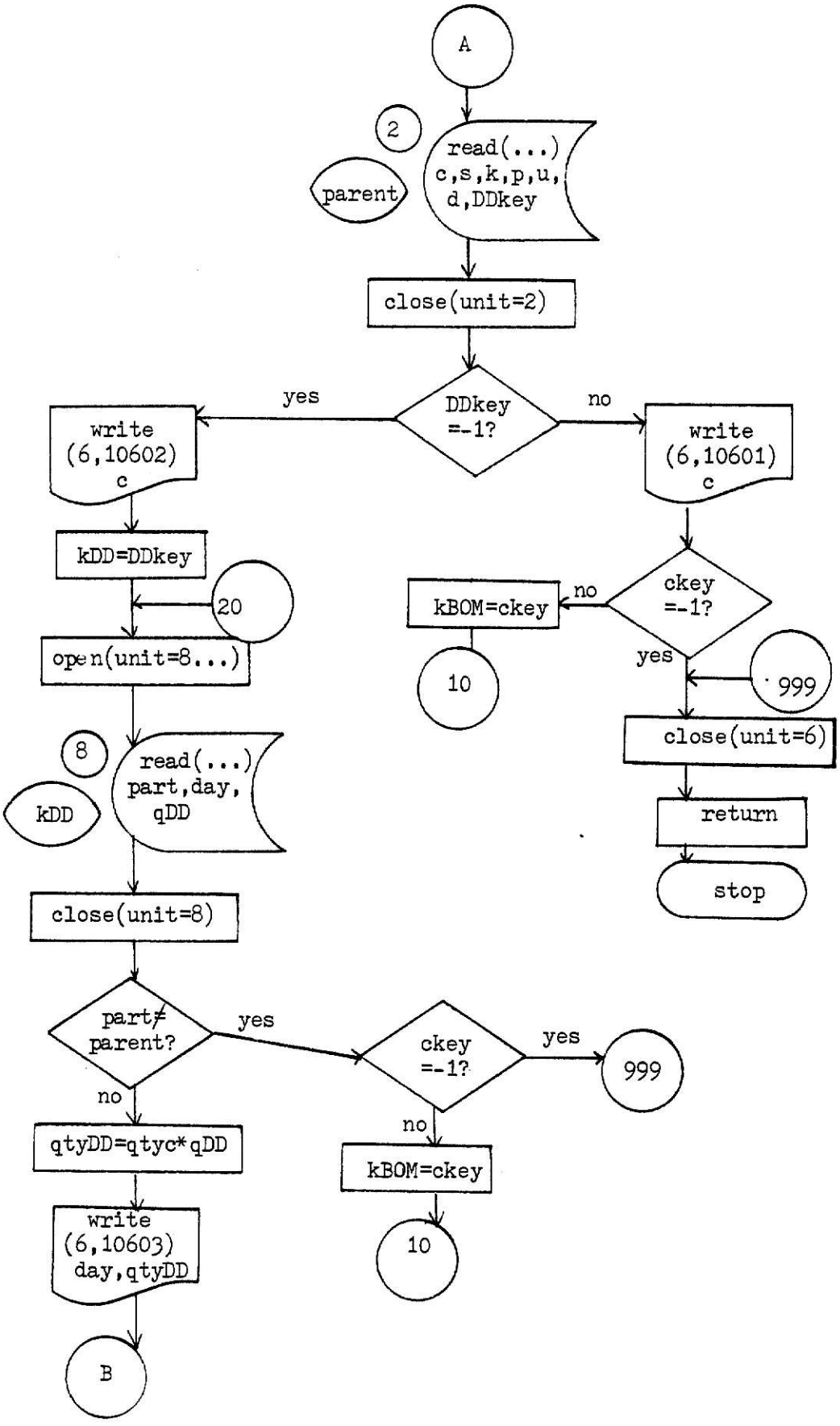

C
write
(6,10604)
dXO1
write
(6,10605)
dSP1
write
(6,10606)
sumDD1
write
(6,10600)
Gross Requirements
write
(6,10607)
GR1
write
(6,10608)
SR1
write
(6,10609)
OH1
D

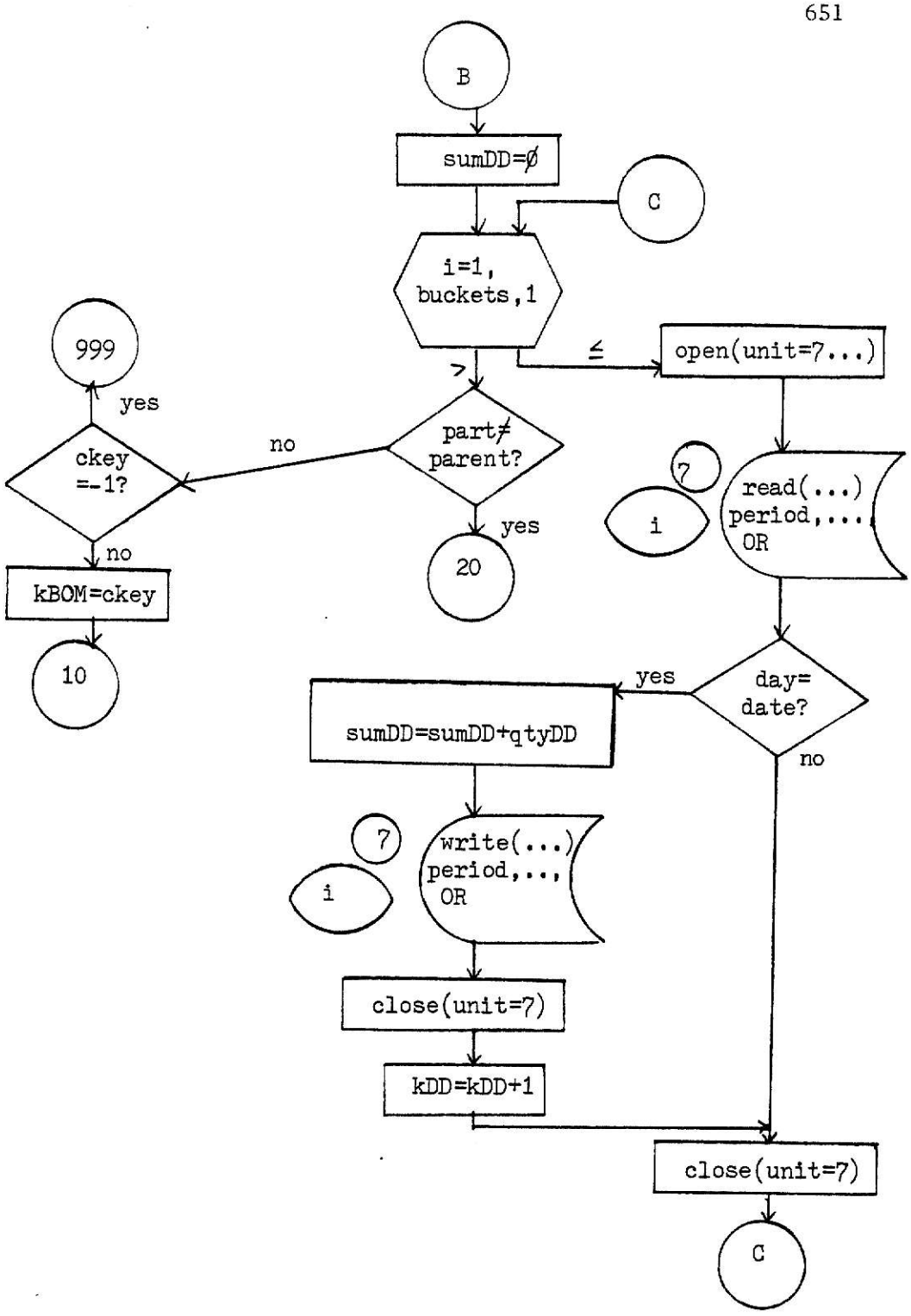

D
write
(6,10610)
NR1
Net Requirements.
write
(6,10611)
LR1
Lot Size
Requirements.
write
(6,10612)
OR1
Order
releases.
write
(6,10600)
close(unit=6)
return
stop

MRP

-Subroutine DependentDemands-
Arguments: BofM,buckets,
parttoexplode

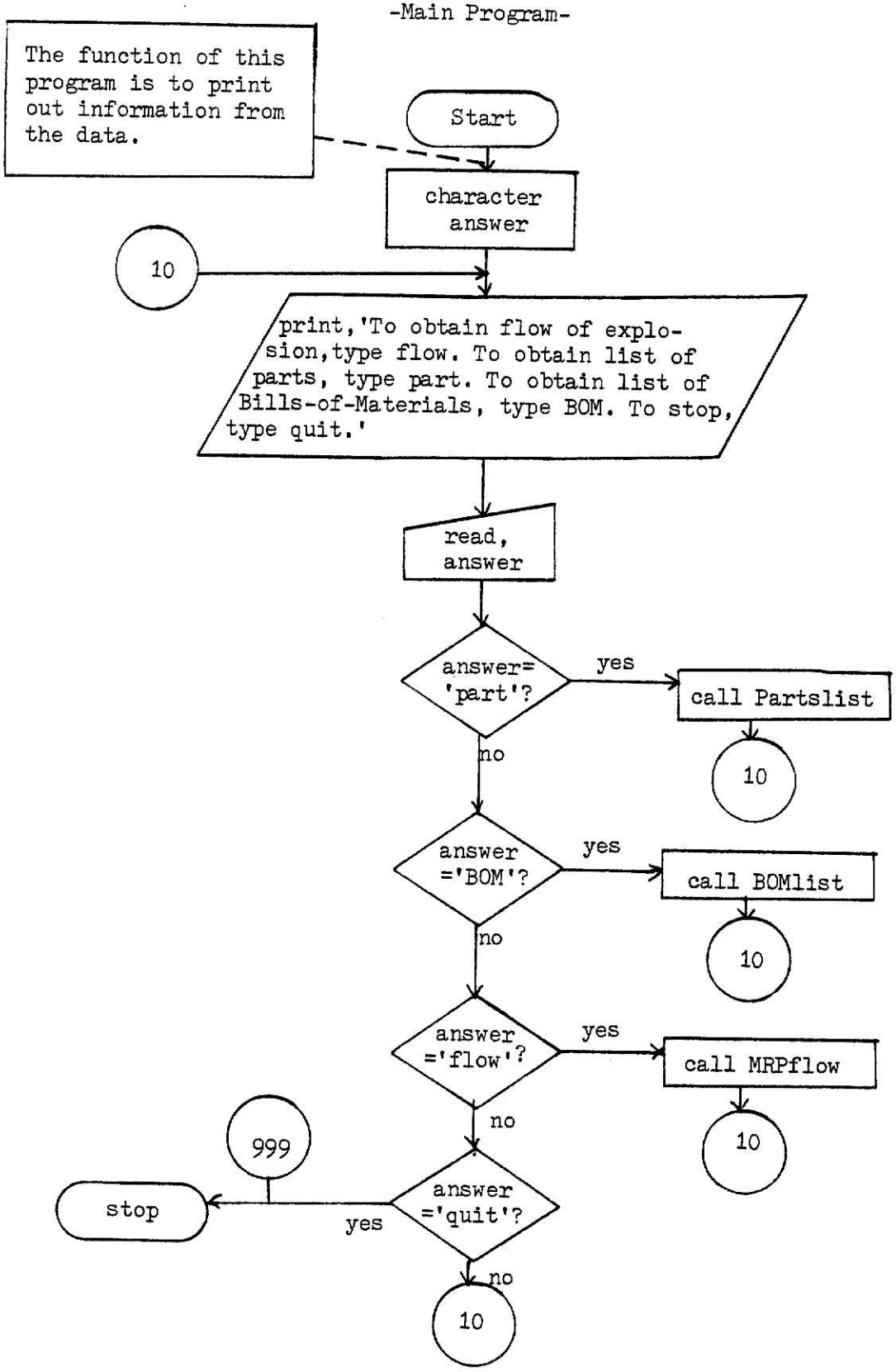



A
2
parent
read(...)
c,s,k,p,u,
d,DDkey
close(unit=2)
DDkey
=-1?
yes
no
write
(6,10602)
c
write
(6,10601)
c
kDD=DDkey
20
open(unit=8...)
kBOM=ckey
no
10
ckey
=-1?
yes
999
close(unit=6)
return
stop
8
kDD
read(...)
part,day,
qDD
close(unit=8)
part≠
parent?
yes
ckey
=-1?
yes
999
no
qtyDD=qtyc*qDD
write
(6,10603)
day,qtyDD
no
kBOM=ckey
10
B


B
sumDD=ø
C
i=1,
buckets,1
≤
open(unit=7...)
>
part≠
parent?
no
yes
20
999
yes
ckey
=-1?
no
kBOM=ckey
10
7
i
read(...)
period,...
OR
day=
date?
yes
no
sumDD=sumDD+qtyDD
7
i
write(...)
period,..,
OR
close(unit=7)
kDD=kDD+1
close(unit=7)
C

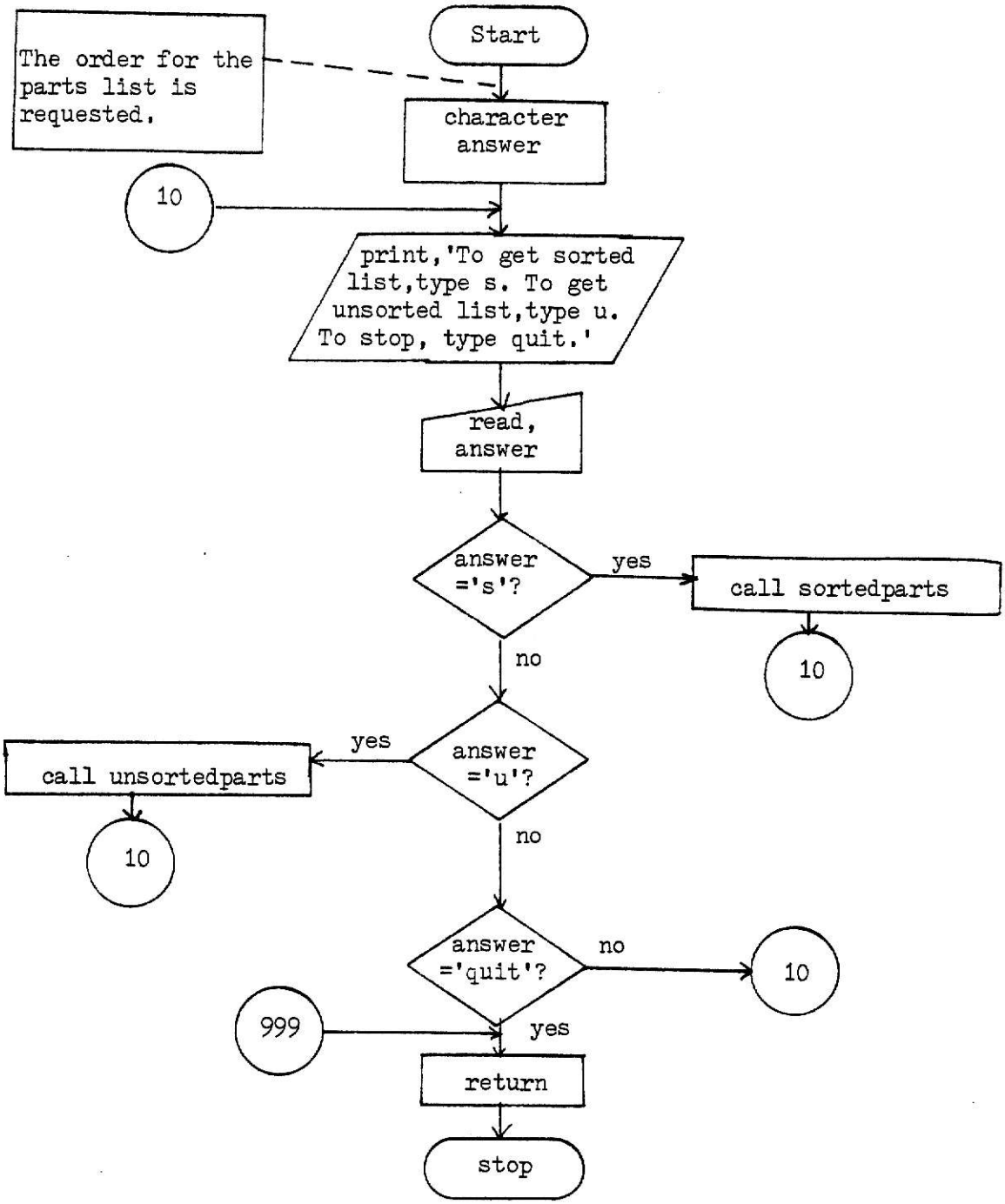
SystemStatus

-Main Program-



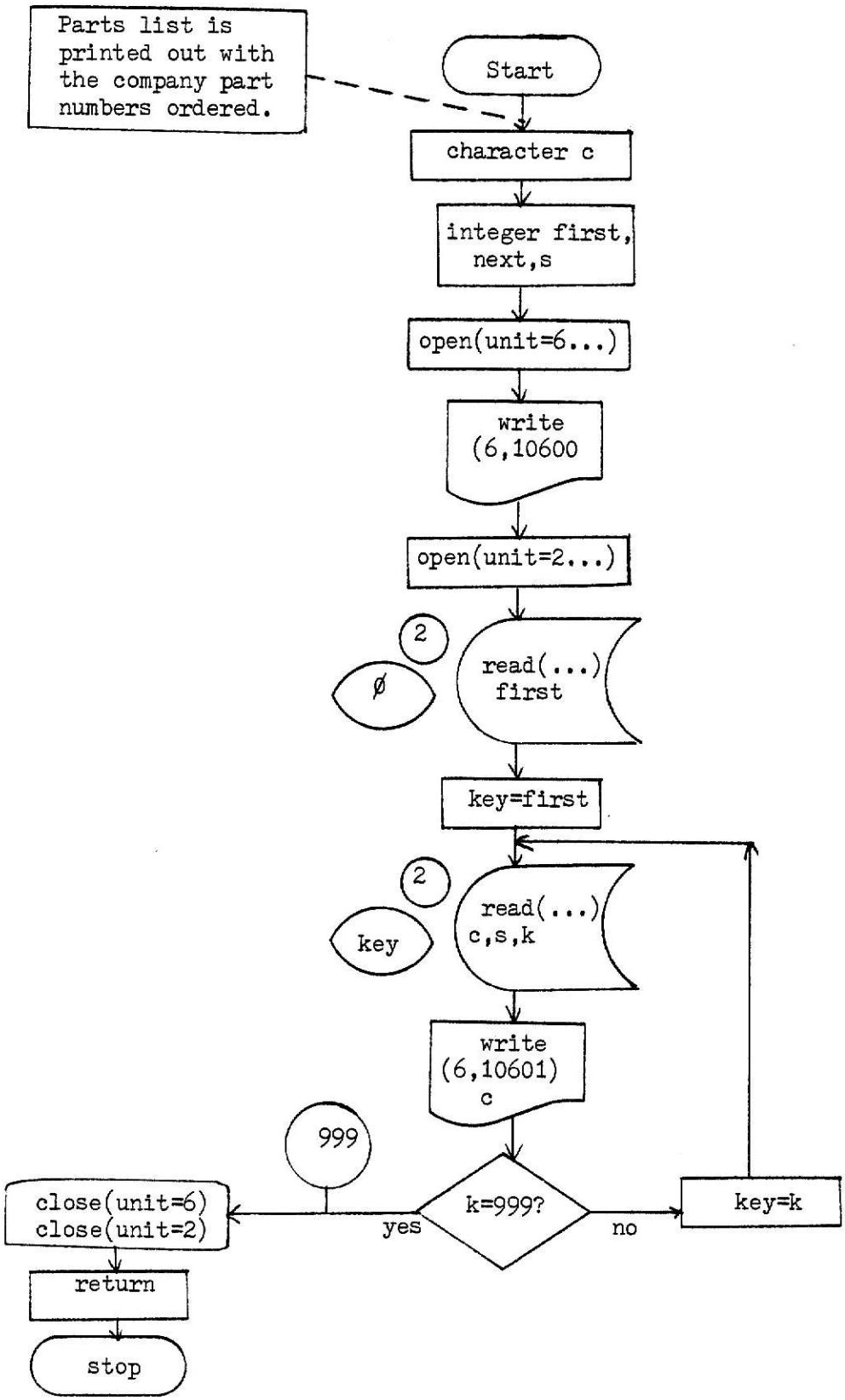
System Status

-Subroutine Partslist-



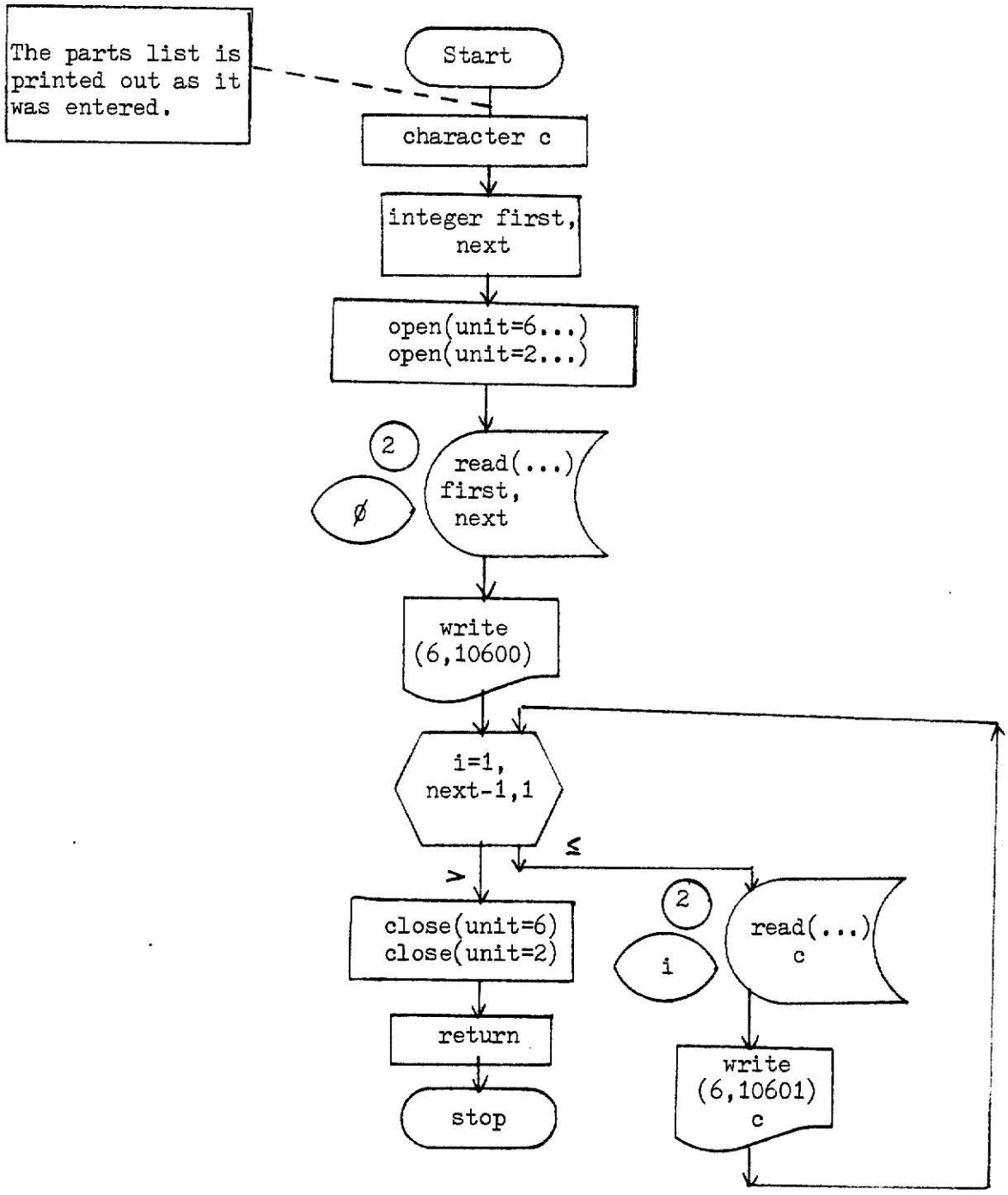
SystemStatus

-Subroutine sortedparts-



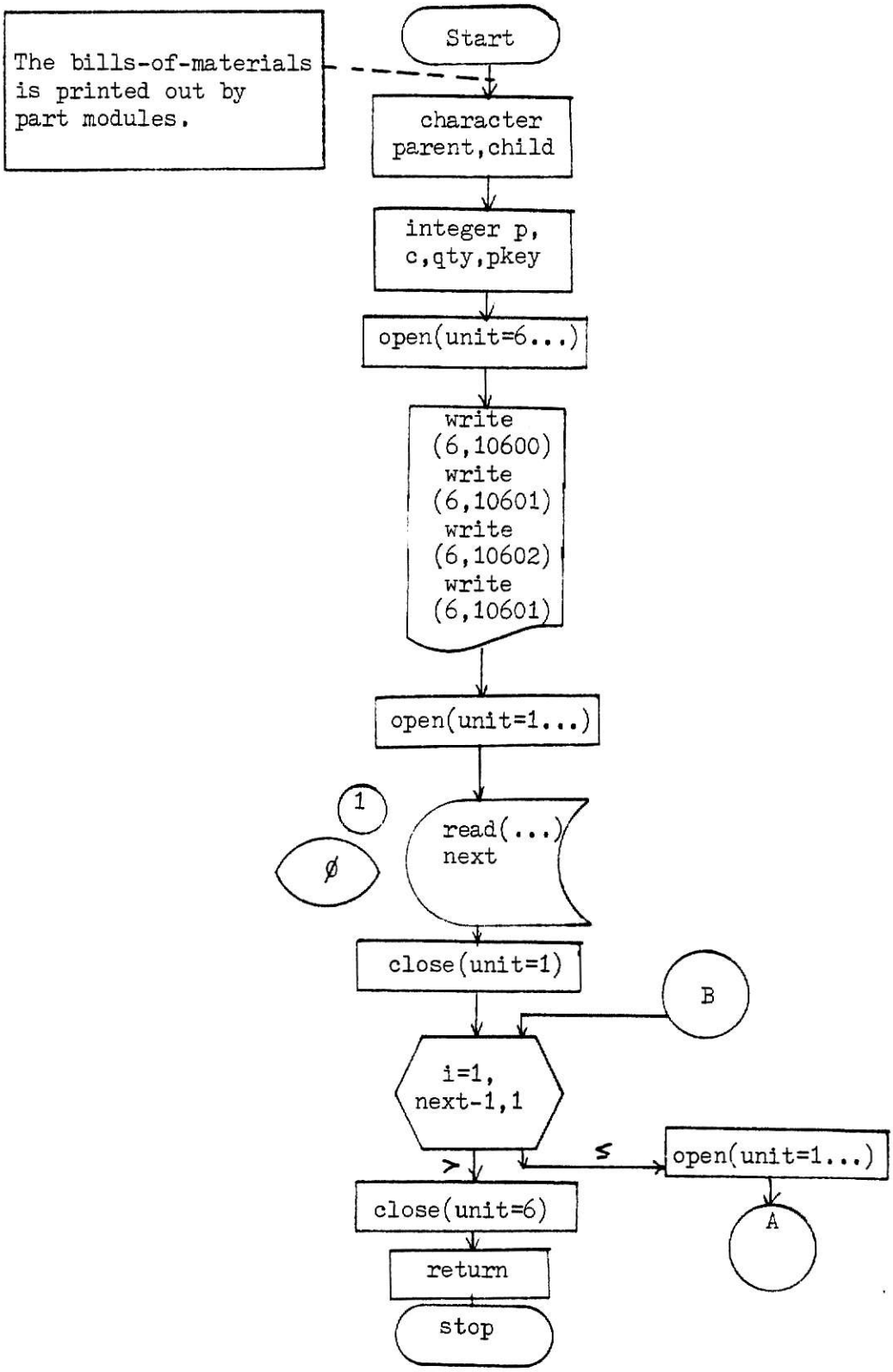
SystemStatus

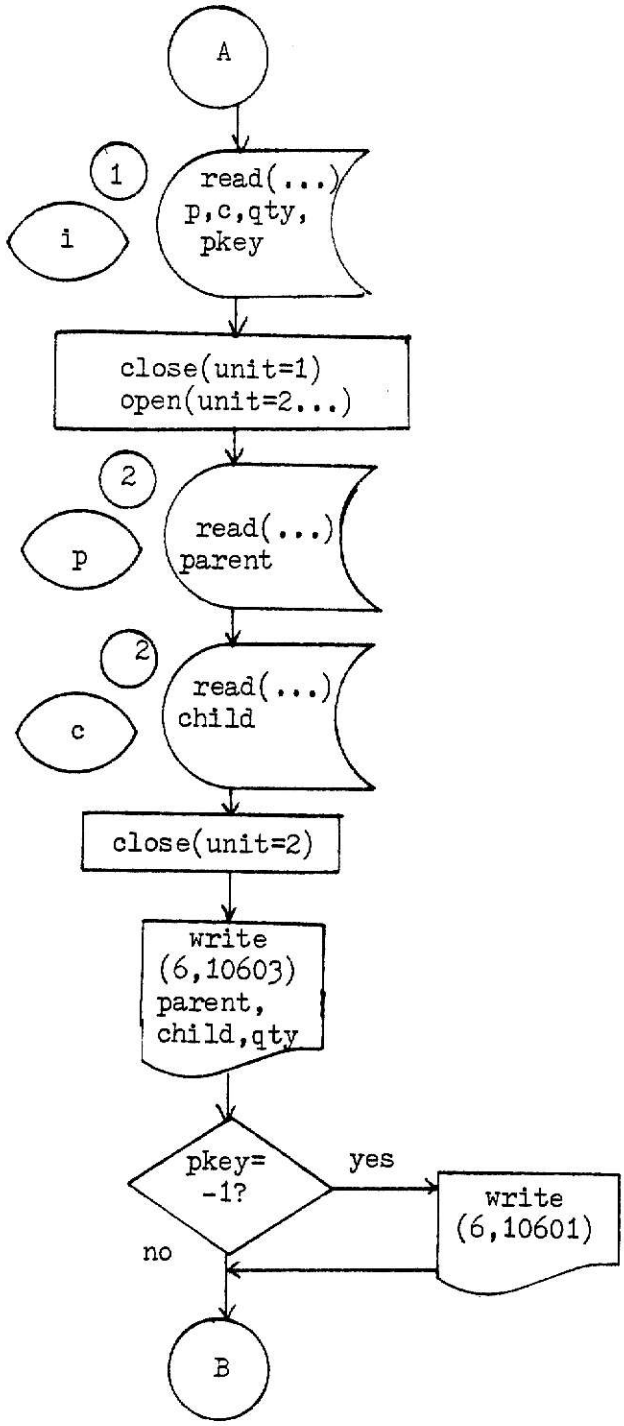
-Subroutine unsortedparts-



SystemStatus

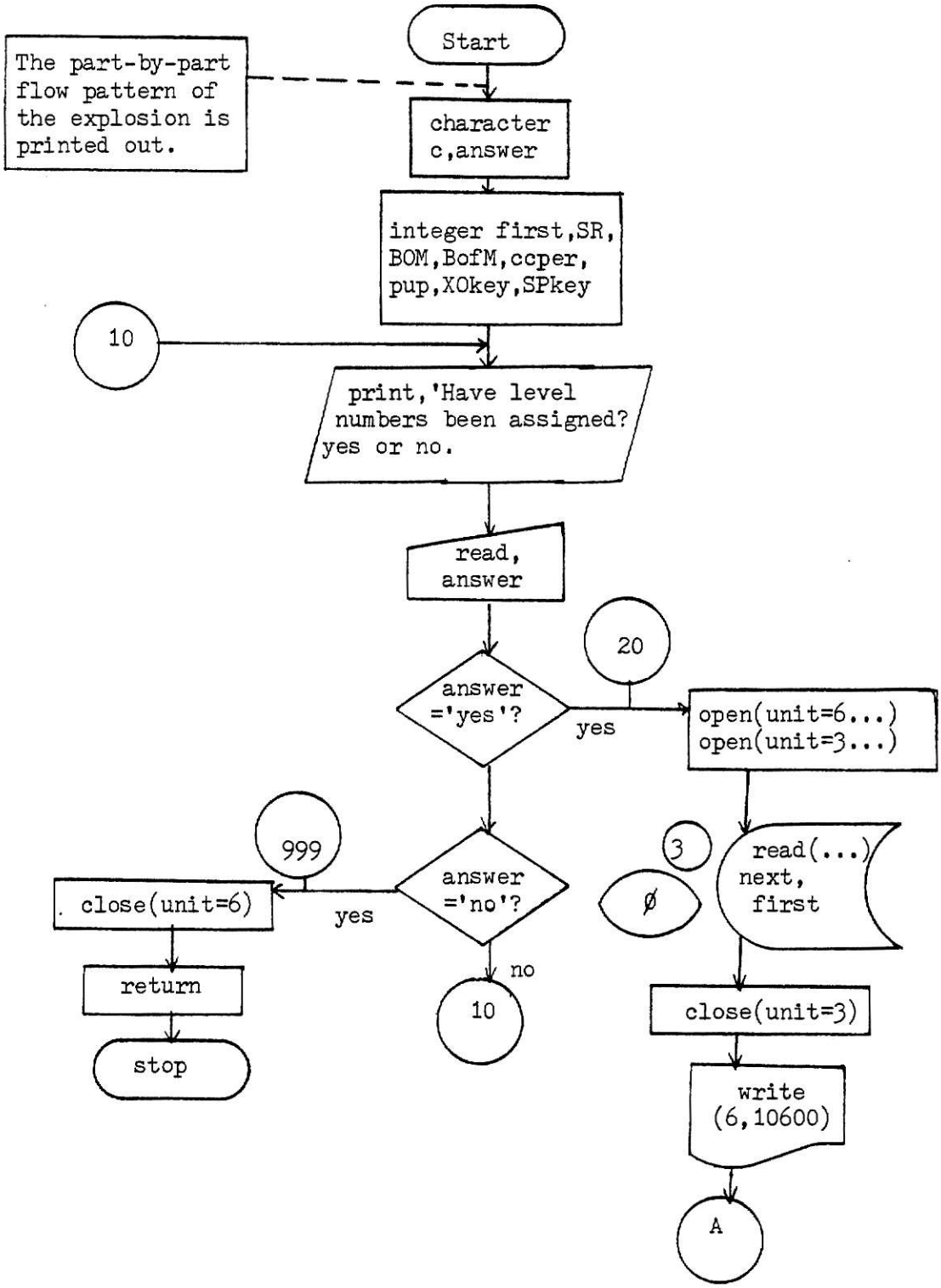
-Subroutine BOMlist-

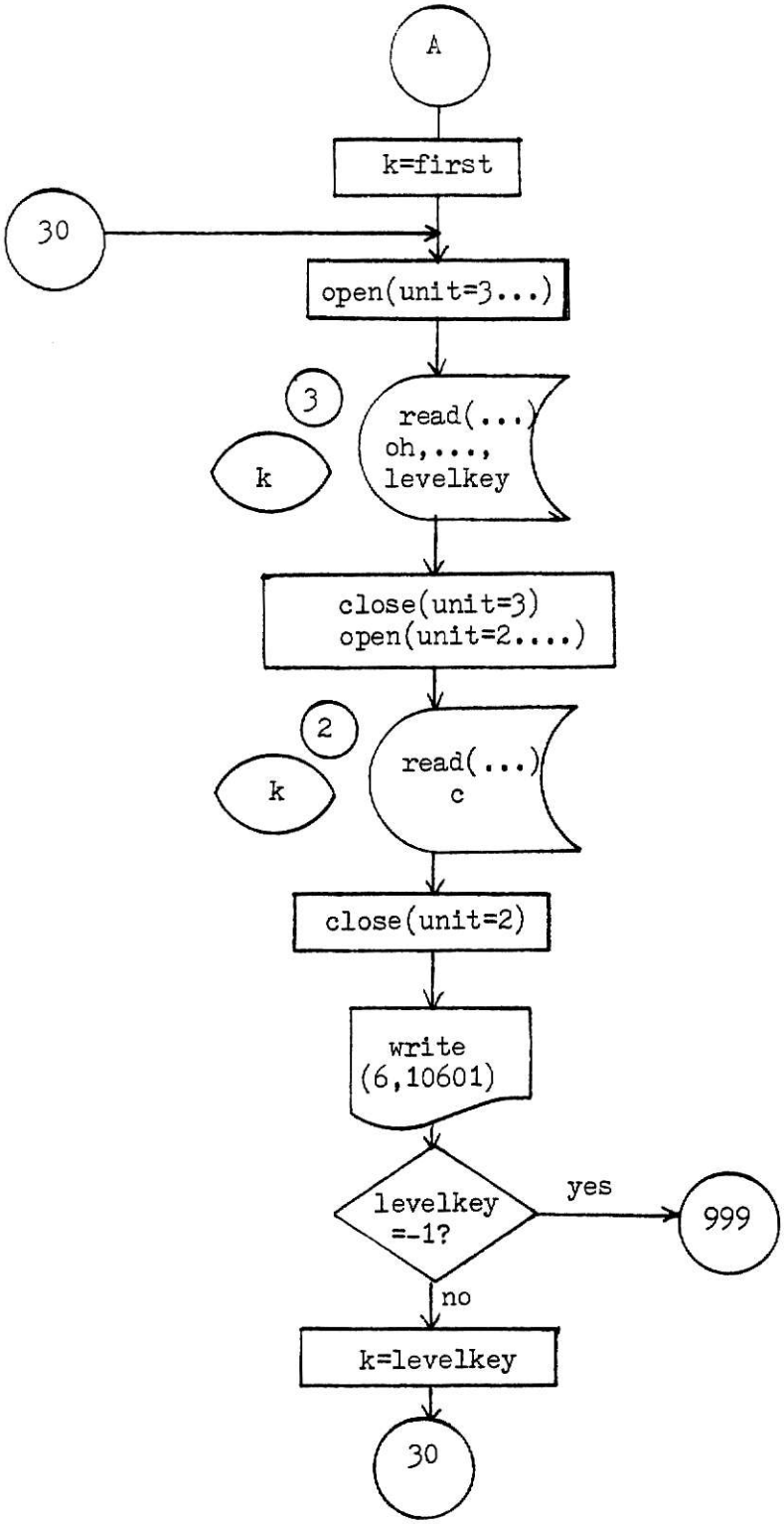



A
1
i
read(...)
p,c,qty,
pkey
close(unit=1)
open(unit=2...)
2
p
read(...)
parent
2
c
read(...)
child
close(unit=2)
write
(6,10603)
parent,
child,qty
pkey=
-1?
yes
write
(6,10601)
no
B

SystemStatus

-Subroutine MRPflow-




A
k=first
30
open(unit=3...)
3
k
read(...)
oh,...,
levelkey
close(unit=3)
open(unit=2....)
2
k
read(...)
c
close(unit=2)
write
(6,10601)
levelkey
=-1?
yes
999
no
k=levelkey
30

APPENDIX D

MISCELLANEOUS

Data File Names

Abbreviated Name	Complete Name
bom	(f:27)bom,rel
partid	(f:35)partid,rel
partstatus	(f:100)partstatus,rel
SR	(f:18)SR,rel
horizon	(f:98)horizon,rel
DD	(f:17)DD,rel
MPS	(f:18)MPS,rel
XO	(f:18)XO,rel
SP	(f:18)SP,rel

The complete name is required for the SuperPET editing commands, such as get and copy.

Open Commands

Unit Number	File Name	Command
1	bom	open(unit=1,file='(f:27)bom,rel', access='DIRECT',recl=27)
2	partid	open(unit=2,file='(f:35)partid, rel',access='DIRECT',recl=35)
3	partstatus	open(unit=3,file='(f:100)partstatus, rel',access='DIRECT',recl=100)
4	SR	open(unit=4,file='(f:18)SR,rel', access='DIRECT',recl=18)
5	CRT	
6	printer	open(unit=6,file='printer')
7	horizon	open(unit=7,file='(f:98)horizon, rel',access='DIRECT',recl=98)
8	DD	open(unit=8,file='(f:17)DD,rel', access='DIRECT',recl=17)
9	MPS	open(unit=9,file='(f:18)MPS,rel', access='DIRECT',recl=18)
10	XO	open(unit=10,file='(f:18)XO,rel', access='DIRECT',recl=18)
11	SP	open(unit=11,file='(f:18)SP,rel', access='DIRECT')

Scheme for

Format Statements

The basic scheme for most of the read and write statements is as follows:

the format number is ABCDE,

where A=Ø if the format statement is for a write to record number Ø;

A=1 if the format statement is for a read to record number Ø;

A=2 if the format statement is for a write to record number 1 to n;

BC= the unit number assigned to the file location for the read or write;

DE= the number of variables to be read or written.

List of

Format Statements

(Using Scheme)

Format Number	Format Specifications
101	i4,5x,'Bill-of-Material'
203	3i3,8x,'Partid Data File'
204	4i3,5x,'Partid Data File'
301	i3,8x,'Partstatus Data File'
302	2i3,5x,'Partstatus Data File'
402	2i4,5x,'SR'
701	i3,5x,'Planning Horizon'
901	i4,5x,'MPS'
1001	i4,5x,'XO'
1101	i4,5x,'SP'
10101	i4
10201	i3
10202	2i3
10203	3i3
10204	4i3
10301	i3
10302	2i3
10402	2i4
10701	i3
10801	'DD file'
10901	i4

List of

Format Statements

(Using Scheme)

-cont.-

Format Number	Format Specifications
11001	i4
11101	i4
20105	2(i3,',',),i4,2(',',i4)
20106	2(i3,',',),i4,3(',',i4)
20203	a10,2(',',i3)
20204	a10,3(',',i3)
20206	a10,5(',',i3)
20207	a10,5(',',i3),',',i4
20319	f8.0,5(',',i4),',',i1,',', i4,',',f8.2,2(',',f8.4), ',',i1,',',f8.0,',',i1, 5(',',i3)
20403	i4,',',f8.0,',',i4
20702	i3,',',i4
20712	i3,',',i4,10(',',f8.0)
20803	i3,',',i4,',',f8.0
20903	i4,',',f8.0,',',i4
21003	i4,',',f8.0,',',i4
21103	i4,',',f8.0,',',i4

**********Materials Requirement Planning**********

Part Number a ;System Number 1

Lead Time = 10 days; On Hand Inventory is 10.

Lot Size is determined with Lot-for-Lot Technique

The amount of parts overdue is 0.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	10.	12.	14.	14.	16.	16.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	0.	0.	0.	0.	0.	0.
___GR___	10.	12.	14.	14.	16.	16.
___SR___	0.	12.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.	0.
___NR___	0.	0.	14.	14.	16.	16.
___LR___	0.	0.	14.	14.	16.	16.
___OR___	14.	14.	16.	16.	18.	18.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	18.	18.	20.	20.	20.
___XO___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	0.	0.	0.	0.	0.
___GR___	18.	18.	20.	20.	20.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	18.	18.	20.	20.	20.
___LR___	18.	18.	20.	20.	20.
___OR___	20.	20.	20.	0.	0.

Sample Prob.
PURCHASE_ORDER

Period_placed_ 19
Date_placed___ 95

Part_Number_a

Quantity_____ 14.

Due_Date_____ 105

Part Number j ;System Number 10
Lead Time = 5 days; On Hand Inventory is 3.
Lot Size is determined with Fixed Period
Algorithm.
The amount of parts overdue is 0.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	3.	4.	4.	5.	3.	10.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	0.	0.	0.	0.	0.	0.
___GR___	3.	4.	4.	5.	3.	10.
___SR___	0.	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.	0.
___NR___	0.	4.	4.	5.	3.	10.
___LR___	0.	8.	0.	8.	0.	12.
___OR___	8.	0.	8.	0.	12.	0.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	2.	3.	4.	4.	3.
___XO___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	0.	0.	0.	0.	0.
___GR___	2.	3.	4.	4.	3.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	2.	3.	4.	4.	3.
___LR___	0.	7.	0.	7.	0.
___OR___	7.	0.	7.	0.	0.

Sample Prob.
PURCHASE_ORDER

Period_placed_ 19
Date_placed___ 95

Part_Number_j

Quantity____ 8.

Due_Date____ 100

Part Number g ;System Number 7
Lead Time = 10 days; On Hand Inventory is 10.
Lot Size is determined with Lot-for-Lot Technique
The amount of parts overdue is 0.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	9.	1.	10.	12.	11.	12.
___XD___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	0.	0.	0.	0.	0.	0.
___GR___	9.	1.	10.	12.	11.	12.
___SR___	0.	0.	0.	0.	0.	0.
___OH___	1.	0.	0.	0.	0.	0.
___NR___	0.	0.	10.	12.	11.	12.
___LR___	0.	0.	10.	12.	11.	12.
___OR___	10.	12.	11.	12.	13.	12.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	13.	12.	11.	14.	10.
___XD___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	0.	0.	0.	0.	0.
___GR___	13.	12.	11.	14.	10.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	13.	12.	11.	14.	10.
___LR___	13.	12.	11.	14.	10.
___OR___	11.	14.	10.	0.	0.

Sample Prob.
PURCHASE_ORDER

Period_placed_ 19
Date_placed___ 95

Part_Number_g

Quantity_____ 10.

Due_Date_____ 105

Part Number e ;System Number 5
Lead Time = 15 days; On Hand Inventory is 66.
Dependent Demand sources are as follows:

Part_a	_Date_	_Quantity_
	135	40.
	130	40.
	125	40.
	120	36.
	115	36.
	110	32.
	105	32.
	100	28.
	95	28.

Lot Size is determined with Fixed Period Algorithm.
The amount of parts overdue is 0.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	0.	5.	0.	0.	10.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	28.	28.	32.	32.	36.	36.
___GR___	28.	28.	37.	32.	36.	46.
___SR___	0.	67.	0.	0.	0.	0.
___OH___	38.	77.	40.	8.	0.	0.
___NR___	0.	0.	0.	0.	28.	46.
___LR___	0.	0.	0.	0.	74.	0.
___OR___	0.	74.	0.	82.	0.	46.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	0.	2.	0.	6.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	40.	40.	40.	0.	0.
___GR___	40.	42.	40.	6.	0.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	40.	42.	40.	6.	0.
___LR___	82.	0.	46.	0.	0.
___OR___	0.	0.	0.	0.	0.

Part Number h ;System Number 8
Lead Time = 10 days; On Hand Inventory is 10.
Dependent Demand sources are as follows:

Part_g	_Date_	_Quantity_
	135	10.
	130	14.
	125	11.
	120	12.
	115	13.
	110	12.
	105	11.
	100	12.
	95	10.

Lot Size is determined with Fixed Period Algorithm.
The amount of parts overdue is 0.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	10.	12.	11.	12.	13.	12.
___GR___	10.	12.	11.	12.	13.	12.
___SR___	0.	23.	0.	0.	0.	0.
___OH___	0.	11.	0.	0.	0.	0.
___NR___	0.	0.	0.	12.	13.	12.
___LR___	0.	0.	0.	25.	0.	23.
___OR___	0.	25.	0.	23.	0.	24.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	11.	14.	10.	0.	0.
___GR___	11.	14.	10.	0.	0.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	11.	14.	10.	0.	0.
___LR___	0.	24.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.

Part Number c ;System Number 3
Lead Time = 15 days; On Hand Inventory is 0.
Dependent Demand sources are as follows:

Part_a	_Date_	_Quantity_
	135	20.
	130	20.
	125	20.
	120	18.
	115	18.
	110	16.
	105	16.
	100	14.
	95	14.

Part_e	_Date_	_Quantity_
	120	92.
	110	164.
	100	148.

Part_g	_Date_	_Quantity_
	135	20.
	130	28.
	125	22.
	120	24.
	115	26.
	110	24.
	105	22.
	100	24.
	95	20.

Lot Size is determined with Fixed Period Algorithm.
The amount of parts overdue is 14.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	2.	2.	3.	2.	2.	3.
_sum_DD_	34.	186.	38.	204.	44.	134.
___GR___	36.	188.	41.	206.	46.	137.
___SR___	210.	0.	250.	0.	0.	0.
___OH___	174.	0.	209.	3.	0.	0.
___NR___	0.	14.	0.	0.	43.	137.
___LR___	0.	14.	0.	43.	0.	181.
___OR___	43.	0.	181.	0.	93.	0.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.
___SP___	2.	2.	3.	2.	2.
_sum_DD_	42.	48.	40.	0.	0.
___GR___	44.	50.	43.	2.	2.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	44.	50.	43.	2.	2.
___LR___	0.	93.	0.	4.	0.
___OR___	4.	0.	0.	0.	0.

```
*****************************************************************
***                                                           ***
***                                                           ***
***                                                           ***
***                                                           ***
***                      Sample Probl                         ***
***                     PURCHASE_ORDER                        ***
***                                                           ***
***                                                           ***
***                           Period_placed_ 19               ***
***                           Date_placed___  95              ***
***                                                           ***
***                                                           ***
***       Part_Number_c                                       ***
***                                                           ***
***       Quantity____      43.                               ***
***                                                           ***
***       Due_Date____ 110                                    ***
***                                                           ***
***                                                           ***
***                                                           ***
***                                                           ***
*****************************************************************
Part Number i          ;System Number  9
Lead Time =   15 days; On Hand Inventory is       75.
Dependent Demand sources are as follows:
Part_h                _Date_     _Quantity_
                        120            24.
                        110            23.
                        100            25.
Part_j                _Date_     _Quantity_
                        135            14.
                        125            14.
                        115            24.
                        105            16.
                         95            16.
Lot Size is determined with Fixed Period
Algorithm.
The amount of parts overdue is        0.
```

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	4.	0.	0.	3.	0.
___SP___	1.	0.	1.	0.	1.	0.
_sum_DD_	16.	25.	16.	23.	24.	24.
___GR___	17.	29.	17.	23.	28.	24.
___SR___	0.	0.	0.	0.	0.	0.
___OH___	58.	29.	12.	0.	0.	0.
___NR___	0.	0.	0.	11.	28.	24.
___LR___	0.	0.	0.	80.	0.	0.
___OR___	80.	0.	0.	0.	25.	0.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	2.	0.	0.	0.	9.
___SP___	1.	0.	1.	0.	1.
_sum_DD_	14.	0.	14.	0.	0.
___GR___	17.	0.	15.	0.	10.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	17.	0.	15.	0.	10.
___LR___	0.	25.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.

Sample Prob1
PURCHASE_ORDER

Period_placed_ 19
Date_placed___ 95

Part_Number_i

Quantity_____ 80.

Due_Date_____ 110

Part Number b ;System Number 2
Lead Time = 20 days; On Hand Inventory is 0.
Dependent Demand sources are as follows:

Part_a	_Date_	_Quantity_
	135	40.
	130	40.
	125	40.
	120	36.
	115	36.
	110	32.
	105	32.
	100	28.
	95	28.

Part_c	_Date_	_Quantity_
	125	4.
	115	93.
	105	181.
	95	43.

Part_g	_Date_	_Quantity_
	135	40.
	130	56.
	125	44.
	120	48.
	115	52.
	110	48.
	105	44.
	100	48.
	95	40.

Part_j	_Date_	_Quantity_
	135	7.
	125	7.
	115	12.
	105	8.
	95	8.

Lot Size is determined with Fixed Period Algorithm.
The amount of parts overdue is 0.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	119.	76.	265.	80.	193.	84.
___GR___	119.	76.	265.	80.	193.	84.
___SR___	825.	0.	270.	0.	0.	0.
___OH___	706.	630.	635.	555.	362.	278.
___NR___	0.	0.	0.	0.	0.	0.
___LR___	0.	0.	0.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.	0.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	95.	96.	87.	0.	0.
___GR___	95.	96.	87.	0.	0.
___SR___	0.	0.	0.	0.	0.
___OH___	183.	87.	0.	0.	0.
___NR___	0.	0.	0.	0.	0.
___LR___	0.	0.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.

Part Number d ;System Number 4
Lead Time = 20 days; On Hand Inventory is 320.
Dependent Demand sources are as follows:

Part_c	_Date_	_Quantity_
	125	4.
	115	93.
	105	181.
	95	43.

Part_i	_Date_	_Quantity_
	115	100.
	95	320.

Lot Size is determined with Fixed Period Algorithm.
The amount of parts overdue is 421.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	363.	0.	181.	0.	193.	0.
___GR___	363.	0.	181.	0.	193.	0.
___SR___	0.	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.	0.
___NR___	43.	0.	181.	0.	193.	0.
___LR___	421.	0.	0.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.	0.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	4.	0.	0.	0.	0.
___GR___	4.	0.	0.	0.	0.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	4.	0.	0.	0.	0.
___LR___	0.	0.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.

Part Number # ;System Number 6
Lead Time = 20 days; On Hand Inventory is 100.
Dependent Demand sources are as follows:

Part_e	_Date_	_Quantity_
	120	46.
	110	82.
	100	74.
Part_g	_Date_	_Quantity_
	135	10.
	130	14.
	125	11.
	120	12.
	115	13.
	110	12.
	105	11.
	100	12.
	95	10.
Part_i	_Date_	_Quantity_
	115	75.
	95	240.

Lot Size is determined with Fixed Period Algorithm.
The amount of parts overdue is 298.

Period	19	20	21	22	23	24
_1st_day	95	100	105	110	115	120
__MPS___	0.	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.	0.
_sum_DD_	250.	86.	11.	94.	88.	58.
___GR___	250.	86.	11.	94.	88.	58.
___SR___	0.	0.	200.	0.	0.	0.
___OH___	0.	0.	189.	95.	7.	0.
___NR___	150.	86.	0.	0.	0.	51.
___LR___	298.	0.	0.	0.	0.	0.
___OR___	0.	0.	0.	24.	0.	0.

Period	25	26	27	28	29
_1st_day	125	130	135	140	145
__MPS___	0.	0.	0.	0.	0.
___XO___	0.	0.	0.	0.	0.
___SP___	0.	0.	0.	0.	0.
_sum_DD_	11.	14.	10.	0.	0.
___GR___	11.	14.	10.	0.	0.
___SR___	0.	0.	0.	0.	0.
___OH___	0.	0.	0.	0.	0.
___NR___	11.	14.	10.	0.	0.
___LR___	0.	24.	0.	0.	0.
___OR___	0.	0.	0.	0.	0.

Additional Editing Information

To Edit Lead Times Only

To enter or edit the lead times only, perform the following steps:

Step 1. Do Steps 1 and 2 in Section 6.2.22.

Step 2. When 'To enter both lead times and lotting rules...' appears on the screen, type 'lt' and press the return key.

Step 3. When 'part, lead time (days), or quit, Ø to stop' appears, type the name of the part and its lead time based on the number of shop calendar days. Press the return key.

Note: Each time the message in Step 3 appears, additional lead time data can be entered. When no more data are to be entered, type 'quit, Ø' and press the return key.

Step 4. After 'quit, Ø' is typed, type 'quit' and press the return key when 'To enter both lead times and lotting rules...' appears on the screen.

Step 5. Do Step 10 to 12 in Section 6.2.6.

To Edit Lotting Rules Only

To enter or edit the lotting rules only, perform the following steps:

Step 1. Do Steps 1 and 2 in Section 6.2.22.

Step 2. When 'To enter both lead times and lotting rules...' appears on the screen, type 'lr' and press the return key.

Step 3. When 'part, lot rule (L4L, FPQ) or quit, q to stop' appears on the CRT, type the part's name and either 'L4L' or 'FPQ' for its lotting rule. Press the return key.

Note 1: When 'FPQ' is typed, the message 'period (days) for FPQ' will appear on the screen. Then, type in the number of days for the fixed

period length and press the return key.

Note 2: Each time the message in Step 3 appears, additional lotting rules can be entered. When no more data are to be entered, type 'quit, q' and press the return key.

Step 4. After 'quit, q' is typed, type 'quit' and press the return key when 'To enter both lead times and lotting rules...' appears on the screen.

Step 5. Do Steps 10 to 12 in Section 6.2.6.

Error Messages

Conversion Error

The format for a con error is:

*** Error: Conversion Error read, variable name

Error while executing 'program name'

Debug?

When the error occurs, check the type of data entered. If alphanumeric instead of a numeric (integer or real) was typed, type 'c' (for continue), press the return key and reenter the data. If that doesn't work, type 'q' (for quit) and press the return key twice. Then, call the program doctor.

Numeric Overflow

The format for a numeric overflow is:

*** Error: numeric overflow read, variable name

Error while executing 'program name'

Debug?

The data entered was too large. Type 'c' (for continue), press the return key, and reenter a smaller value.

To Remove Files From Disk

To remove or scratch a file off the disk, type either 'scratch file name' or 'scr file name' and press the return key. See Appendix D for the table of the complete names for the data files.

To Copy Files to Printer

To obtain a copy of a file stored on the disk, type 'copy file name to printer' and press the return key. See Appendix D for the complete name of the data file.

To Load File into Computer

To load any file into the computer's memory, first type '*d' and press the return key to clear out the computer's memory. Then, type 'get file name' and press the return key. [Or, type 'g file name' and press the return key.] See Appendix D for the complete name of the data file.

To Copy File From One Disk to Another

To transfer a copy of a file from one disk to another is desired, perform the following steps:

Step 1. Do Steps 1 to 7 in Section 6.1.

Step 2. Remove system floppy disk out of disk drive #1.

Step 3. Insert formatted disk into disk drive #1. This disk will receive the file being transferred.

Step 4. Type 'disk/Ø. filename1 to disk/1. filename2' and press the return key. Filename1 is the file being copied. Filename2 is the name of the file when it appears on the disk in disk drive #1. To avoid confusion, set filename2 = filename1.

To Store File Onto the Disk From Memory

To store file onto the disk from memory, first type 'name file name' and press the return key. Then, type 'put' or 'p' and press the return key.

Note: This is for storing a new file onto the disk. If the file has already been stored on the disk, this step is not necessary. To remove a file from the computer's memory, type '*d' and press the return key.

To Format a Disk

To format a new disk, perform the following steps:

Step 1. Do Steps 1 to 5 in Section 6.1 if they haven't been done.

Step 2. If Step 6 in Section 6.1 has been done, type 'bye' and press the return key.

Step 3. Load the Waterloo basic into the computer's memory by typing 'b' and pressing the return key.

Step 4. When the word 'Ready' appears on the CRT screen, type 'open #15, 'disk', output' and press the return key.

Step 5. When the word 'Ready' appears again, type 'print #15, 'NO: diskname, XX'' where diskname is the variable name for the disk and XX is a digit number. Press the return key.

Note: The disk is now being formatted. Be patient, it will take a few minutes.

Step 6. When 'Ready' appears again, type 'close #15' and press the return key.

Step 7. When 'Ready' appears, type 'bye' and press the return key.

MRP EXPLOSION FOR THE COMMODORE COMPUTER

by

Karen Marie May

B.S., Florida State University, 1975

B.S. I.E., Kansas State University, 1982

An Abstract of a Master's Thesis

submitted in partial fulfillment

of the requirements for

the degree

MASTER OF SCIENCE

Department of Industrial Engineering

Kansas State University

Manhattan, Kansas

1983

The focus of this Master of Science thesis is the development of software for the Commodore SuperPET computer which is capable of performing the basic MRP explosion. MRP stands for Material Requirements Planning which is an inventory planning method. The MRP explosion uses inputs from the master production schedule, the bills-of-materials, and the part status records to generate the gross requirements, the net requirements, the lot sizes, and data on order release for each part.

The files - PartFile, BOMfile, DataPlus, DataPlus2, Level, MRP, and SystemStatus - are written in the programming language known as Waterloo Fortran. With PartFile, BOMfile, DataPlus, and DataPlus2, the computer works interactively with the MRP User to store the input information into direct access files. The Level file assigns a level number to each part in inventory and determines the order for the parts to be exploded. The MRP file performs the explosion, prints out the results, and produces an order when it is needed. The SystemStatus file prints out information from the data files. The thesis contains a program logic manual as well as a user's guide.

Since companies are selling MRP software for profit, the literature on how to write the programs tends to be scarce. IBM has some publications available for the larger mainframe computers. But, most of the printed material revolves around explaining how to implement and maintain a successful MRP system. O. Wight, J. Orlicky, and G. Plossl are a few of the recognized authorities and have written several journal articles and books. Articles are presented regularly in Production and Inventory Management, published by APICS.

The MRP explosion has been developed in this thesis for two reasons. One reason is for its use as a teaching tool. The second reason is to investigate the possibility of using a microcomputer instead of a large mainframe computer. The microcomputer appears capable of handling small parts lists, but the execution time is long.