A SEQUENTIAL PASCAL MANUAL FOR FORTRAN PROGRAMMERS

by

JERRY DEAN RAWLINSON

B. S., University of Illinois, 1964

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1977

Approved by:

[Signature]

William J. Hankley
Major Professor
ILLEGIBLE DOCUMENT

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

THIS IS THE BEST COPY AVAILABLE
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.
TABLE OF FIGURES

As the main body of this report consists of nothing but slides, the table of contents and the first slide in each slide set form a tree structure for locating figures depicting particular subjects.

EXAMPLE

Locate a slide showing operations that can be performed on arrays:

1. The table of contents indicates the array slide set begins on page 4.1.

2. Page 4.1 indicates that slide(s) showing operations on arrays begin on page 4.9.
INTRODUCTION

1. PASCAL, SEQUENTIAL PASCAL, AND CONCURRENT PASCAL.

The programming language Pascal was developed by Niklaus Wirth and was specifically designed as a general purpose language which could be used for systematic programming. As a result, Pascal is very readable and can be used in top-down program design in which the final source language program still reflects the structured design and lends itself to systematic verification.\(^5\)

Sequential Pascal (SPASCAL) refers to a specific implementation of Pascal defined by Per Brinch Hansen and Alfred Hartmann of California Institute of Technology for a PDP-11/45 computer. SPASCAL differs from Wirth's definition of Pascal in several areas because of restrictions and extensions required for the implementation.

Concurrent Pascal (CPASCAL) has also been defined by Brinch Hansen and was designed for structured programming of computer operating systems. CPASCAL is an extension of SPASCAL which allows concurrent processes, monitors and classes and was used by Brinch Hansen to write a single user operating system called SOLO for the PDP-11/45 computer.

2. KANSAS STATE UNIVERSITY IMPLEMENTATION OF PASCAL.

Kansas State University Department of Computer Science has implemented SPASCAL, CPASCAL, and SOLO as defined by Brinch Hansen for the INTERDATA 8/32 computer. This implementation was made as part of a research project investigating computer networks. Pascal was chosen because it enforces structured programming, contains very powerful data structures, is very readable, and the SPASCAL compiler can detect many errors which would not be found at compile time using other languages.

3. OBJECTIVE OF THIS REPORT.

This report is designed to serve as an instructional aid for introducing persons who can at least read FORTRAN programs to the Kansas State University implementation of SPASCAL. The design concept is to expand on the student's existing knowledge of FORTRAN so that
he may begin programming in SPASCAL as quickly as possible. This report
is not designed to be either a user guide or a self-contained primer
for SPASCAL. Rather, the report attempts to provide a set of primer
examples which may be used in conjunction with a short course or series
of lectures. The format of the examples should provide the student
with an overview of SPASCAL in an effective and efficient manner.

4. REPORT DESIGN.

This report has been designed to provide a top-down learning process
for the student. The report consists of a series of slides for which the
instructor must provide his own narrative. The slides are divided into
logical learning sets, and the sets are ordered in the sequence in which they
should be presented. Early examples in the slide sequence intentionally conceal
some details of SPASCAL so that specific teaching points may be emphasized.
Later examples in the sequence enumerate upon the basic concepts introduced
earlier.

Each set of slides has been designed to provide the student with a
quick overview of a particular subject while requiring a minimum of reading.
Examples are used to allow the student to compare FORTRAN and SPASCAL
so that he may quickly grasp similarities and differences between the two
languages.

5. INSTRUCTOR GUIDELINES.

As stated above, the slides in this report have been organized for
sequential presentation. No narrative has been provided as it is assumed
each instructor will desire to develop his own narrative to fit
his particular teaching style.

It is recommended that each student be provided with a copy of this
report so that he may make notes on individual slides and use the report
for future reference when programming.

The following references should be available for supplemental study:

Pascal User Manual and Report (3)
Sequential Pascal Report (1)
KSU Pascal Editor (PEDIT) (4)
The Solo Operating System Job Interface (2)
WARNING: Students must be made aware of the differences which exist between SPASKAL and Pascal as defined in Pascal User Manual and Report. These differences have been summarized in Slide Set 11 and are also annotated in the Cross Reference Table in this section (page 0.1.5). All "PASCAL" examples shown in this report are based on the Brinch Hansen definition of SPASKAL as implemented at Kansas State University.

All FORTRAN examples in this report are based on ASCII FORTRAN. Students should be made aware of this as some of the FORTRAN examples may differ from what can be done on the particular FORTRAN implementation they have been working with.

Italic characters in all examples indicate either an error condition or a facility that is not included in the language.

Students may notice that all PASCAL program examples have been indented to emphasize program structure while the FORTRAN program examples have not been indented. This was done based on an assumption that most persons familiar with FORTRAN are accustomed to seeing FORTRAN programs which have not been indented. It should be pointed out that while SPASKAL does not require indentation, it is a good programming practice which improves program readability.

The Cross Reference Table in this section is designed to assist instructors in narrative preparation. The table matches each slide set with the supplemental reference which best describes elements within the slide set. The table also indicates differences between the supplemental reference and SPASKAL.

Most of the SPASKAL examples in this report have been compiled on the Kansas State University implementation of SPASKAL. A card deck containing all of the examples compiled is available through the Department of Computer Science, Kansas State University.

6. CONCLUSION.

After compiling the examples on the INTERDATA 8/32 computer, numerous changes were made based on differences discovered between the written manuals on which the report was based and the actual implementation. All examples in the report reflect the actual implementation on the INTERDATA 8/32. Any additional discrepancies noted by users of the report should be brought to the attention of the KSU department of Computer Science.
Individuals familiar with FORTRAN were provided draft copies of the report and asked to comment on it. Those who reviewed the report felt that it met the stated objective by providing a quick introduction to SPASCAL. As expected, the report does not stand alone. Individuals who knew only FORTRAN experienced some difficulty with new concepts such as enumeration types and records when reading through the report on their own. However, these individuals were able to understand the concepts very quickly when they were talked through the examples in the same manner as an instructor would. The comments provided by these individuals were very helpful and resulted in several changes and additions in the final report to improve the clarity of examples.

Individuals familiar with FORTRAN will find SPASCAL to be a very versatile language due to the increased number of control structures and type declaration facilities available in the language. By using these facilities, SPASCAL can provide for a more direct solution to many problems than is possible in FORTRAN.

The primary disadvantages of the current implementation of SPASCAL are its I/O limitations and the lack of many predefined functions that FORTRAN users are accustomed to. FORTRAN users may also find it inconvenient not to be able to assign, operate, or use as parameters mixed integer and real numbers/variables. However, this is an excellent software engineering feature as many otherwise difficult to find errors are avoided by forcing the individual writing a program to declare his intention to change type.

In summary, SPASCAL is a very flexible general purpose language which is outstanding for use in structured programming. This report will provide instructors with an aid which can significantly reduce the time required to introduce SPASCAL to individuals familiar with FORTRAN and should be applicable to situations in which users merely wish to teach a new language or a more academic situation such as within a formal structured programming course.
REFERENCES


4. Neal, D. KSU Pascal Editor, Department of Computer Science, Kansas State University, 1976.

# CROSS REFERENCE TABLE

<table>
<thead>
<tr>
<th>SLIDE GROUP</th>
<th>REFERENCES*</th>
<th>EXCEPTIONS**</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 CHAP 1, PP 9-11</td>
<td>NO PACKED, LABEL, FILE</td>
</tr>
<tr>
<td></td>
<td>3 CHAP 0, PP 3-8</td>
<td>NO NESTED DECLARATIONS</td>
</tr>
<tr>
<td></td>
<td>3 CHAP 2, PP 12-15</td>
<td>NO SIN(x), COS(x), ARCTAN(x), LN(x)</td>
</tr>
<tr>
<td></td>
<td>1 CHAP 1-5, PP 1-7</td>
<td>EXP(x), SQRT(x), SQR(x), ADD(x), COLN(x) EOF(x), OP</td>
</tr>
<tr>
<td>2</td>
<td>3 CHAP 4, PP 21-33</td>
<td>NO I/O CALLS (WRITELN, READ, ETC.)</td>
</tr>
<tr>
<td></td>
<td>1 CHAP 7, PP 11-16; CHAP 9, PP 26</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 CHAP 3, PP 16-19</td>
<td>NO LABELS</td>
</tr>
<tr>
<td></td>
<td>3 CHAP 5, PP 34-35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 CHAP 7, PP 9-16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3 CHAP 6, PP 36-41</td>
<td>INDEX ,(...); NO PACK, UNPACK</td>
</tr>
<tr>
<td></td>
<td>1 CHAP 7, PP 17-18</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3 CHAP 7, PP 92-99</td>
<td>NO ENUMERATION WITHIN RECORDS</td>
</tr>
<tr>
<td></td>
<td>1 CHAP 7.4, PP 19-20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3 CHAP 11, PP 67-83</td>
<td>NO NESTED PROCEDURES</td>
</tr>
<tr>
<td></td>
<td>1 CHAP 11-12, PP 29-35</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 CHAP, PP 31-32</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2 CHAP 4, PP 5-7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 CHAP 8, PP 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>pp 1-18</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

*1 SEQUENTIAL PASCAL REPORT
2 THE SOLO OPERATING SYSTEM JOB INTERFACE
3 PASCAL USER MANUAL AND REPORT
4 KSU PASCAL EDITOR

**DIFFERENCES BETWEEN REFERENCE AND KSU IMPLEMENTATION OF SPASCAL

0.1.6
PRELIMINARIES

PASCAL (or SPASCAL)
    means SEQUENTIAL PASCAL
    as distinct from CONCURRENT PASCAL

KSU IMPLEMENTATION:
- ported from Brinch Hansen's
  PDP-11/45 implementation at California Institute of
  Technology (which differs slightly from the PASCAL
  report.)

- SPASCAL programs run as a job process under SOLO (single
  user operating system written in CPASCAL, from Brinch
  Hansen, et. al.)

- SOLO runs as a task under OS-32/MT (80K partition)
  on Interdata 8/32.

- currently the PASCAL source is interpreted. The
  interpreter is written in Interdata Common mode CAL.
  (A compiler is being developed.)

- maximum program size is 36K bytes of source text
  (it can be changed)
WHY PASCAL?

IMMEDIATE

- PASCAL ENFORCES STRUCTURE.
  - IN PROGRAM FLOW
  - IN DATA TYPES
  - IN DATA STRUCTURES
  - IN INTRA-PROCEDURE COMMUNICATION

- THE COMPILER WILL CHECK THE STRUCTURE AND DETECT ERRORS WHICH WOULD NOT BE FOUND AT COMPILATION TIME USING OTHER LANGUAGES.

- PASCAL DATA STRUCTURES ARE VERY POWERFUL (TYPES CAN BE DEFINED BY THE PROGRAMMER)

- PASCAL IS VERY READABLE.

FUTURE

- RESEARCHERS ARE BUILDING FORMAL VERIFIERS FOR PASCAL PROGRAMS

- PASCAL HAS A FORMAL SEMANTIC DEFINITION - PROGRAMS SHOULD PRODUCE THE SAME RESULTS UNDER ALL COMPILERS.
**COMPARISONS WITH FORTRAN**

<table>
<thead>
<tr>
<th>FORTRAN VERSION</th>
<th>PASCAL VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALL PASCAL PROGRAMS HAVE BEEN INDENTED TO SHOW DESIRED PROGRAMMING STYLE. INDENTION MUST BE DONE BY THE PROGRAMMER, IT IS NOT DONE AUTOMATICALLY.</td>
</tr>
</tbody>
</table>

Many of the slides in this presentation contain comparisons of Fortran and Pascal program elements. If the comparison programs fit on the same slide, the Fortran example will be on the left and the Pascal example will be on the right, the programs are not all identical, but they will illustrate related features.

*Italic characters such as this indicate an error condition or a facility that does not exist in the language.*
GROUPS OF SLIDES

(A) 1 Simple Features
     2 Control Structures (no go tos)
     3 Simple Data Types (defining new types)
     4 Arrays (generalize indexing)
     5 Records (non-uniform arrays)
     6 Program Structure
     7 Passing Parameters (compiler checks)
     8 Minimal I/O
     9 Standard Prefix
    10 Running the Program
    11 Differences from Jensen/Wirth

(B) 12 Sets
    13 Pointers
    14 Files
    15 Other Things to Read

0.5
GROUP 1: SIMPLE FEATURES

- COMMENTS (1.2)
- BLANKS (1.2)
- END-OF-LINE AND CONTINUATIONS (1.2)
- RESERVED WORDS (1.4)
- SEPARATORS (1.7)
- IDENTIFIERS AND VARIABLES (1.8)
- INTEGERS AND REALS (1.10)
- ASSIGNMENT (1.11)
- ARITHMETIC OPERATORS (1.12)
- RELATIONAL OPERATORS (1.13)
- LOGICAL OPERATORS (1.14)
- OPERATOR PRECEDENCE (1.15)

1.1
C THIS IS A FORTRAN COMMENT
C
C FORTRAN ALLOWS ONE STATEMENT PER LINE
C STATEMENT MUST BE WRITTEN IN COLUMNS 7 TO 72
C A LINE DELIMITS A STATEMENT:

A=B
C=D
E=A+B+C+D

C
C BLANKS GENERALLY HAVE NO EFFECT WITHIN A
C STATEMENT UNLESS USED FOR DATA:

C
A = B
E = A + B + C + D
   D= G + H
C STATEMENTS ARE CONTINUED BY A CHARACTER IN
C COLUMN SIX:

C
A=B+C+D+
*E+F+G
"THIS IS A PASCAL COMMENT"

"PASCAL ALLOW PARTIAL, WHOLE, OR
SEVERAL STATEMENTS ON EACH LINE"

"STATEMENTS ARE CODED IN COLUMNS 1 TO 80
AND ARE SEPARATED BY A SEMICOLON (;) OR DELIMITED BY A
RESERVED WORD"

A:=B;
C:=D;E:=A+B+C+D;

"BLANKS AND COMMENTS HAVE NO EFFECT"

A := B ; "COMMENT" E := A + B + C + D ;

"CONTINUATION IS AUTOMATIC IF A DELIMITER OR SEPARATOR
IS NOT INSERTED"

A:=B+C+D
+ E+F+G;
RESERVED WORDS

FORTRAN

NONE

PASCAL

31 IN KSU IMPLEMENTATION

MAY NOT BE USED AS IDENTIFIERS

ARE UNDERLINED IN THIS PRESENTATION:
BEGIN END

ARE NOT UNDERLINED IN AN ACTUAL PROGRAM

IF(IF.EQ.3)DO=27
(MAY NOT BE LEGAL ON SOME COMPILERS)
### PASCAL RESERVED WORDS
(KSU IMPLEMENTATION)

<table>
<thead>
<tr>
<th>OPERATORS</th>
<th>CONTROL WORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV</td>
<td>BEGIN</td>
</tr>
<tr>
<td>MOD</td>
<td>END</td>
</tr>
<tr>
<td>NOT</td>
<td>CASE</td>
</tr>
<tr>
<td>WHILE</td>
<td>OF</td>
</tr>
<tr>
<td>OR</td>
<td>BEGIN</td>
</tr>
<tr>
<td>REPEAT</td>
<td>CASE</td>
</tr>
<tr>
<td>IF</td>
<td>OF</td>
</tr>
<tr>
<td>FOR</td>
<td>WHILE</td>
</tr>
<tr>
<td>TO</td>
<td>DO</td>
</tr>
<tr>
<td>WITH</td>
<td>REPEAT</td>
</tr>
<tr>
<td>FORWARD</td>
<td>USING</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DECLARATIONS</th>
<th>TYPE IDENTIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST</td>
<td>ARRAY</td>
</tr>
<tr>
<td>TYPE</td>
<td>RECORD</td>
</tr>
<tr>
<td>VAR</td>
<td>SET</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>SET</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>UNIV</td>
</tr>
<tr>
<td>PROGRAM</td>
<td></td>
</tr>
</tbody>
</table>

1.5
RESERVED WORDS

CANNOT BE USED AS IDENTIFIERS--
COMPILER WILL NOT ALLOW IT
EG. BEGIN, VAR, TYPE

KEY WORDS

GOOD IDEA NOT TO USE THESE AS VARIABLES--
BUT COMPILER DOES NOT CHECK
EG. BOOLEAN, REAL, INTEGER, LINE, PAGE, PARAM,

BASIC DATA TYPES
USED IN PREFIX
(DEFINED IN GROUP 9)
SEPARATORS: ; , . :

THESE SEPARATE SYNTACTICAL UNITS

(THEY ALSO DELIMIT TOKENS DURING LEXICAL ANALYSIS)

PAIRED DELIMITERS

" --- "
' --- '
RECORD ---- END
BEGIN ---- END
( --- )
( , --- , )
( : --- : )
CASE --- OF --- END

THESE (AND OTHERS) MARK
THE BEGINNING AND END OF
A SYNTACTICAL UNIT OR
SEQUENCE OF UNITS

NON-PAIRED DELIMITERS

IF --- THEN ---
IF --- THEN --- ELSE ---
REPEAT --- UNTIL ---
VAR ---
TYPE --- = ---
ARRAY --- OF ---

THESE (AND OTHERS) DELIMIT
THE PARTS OF A SYNTACTIC UNIT,
BUT NOT NECESSARILY THE END
OF THE UNIT.

(EVERYTHING HERE WILL BE EXPLAINED LATER IN CONTEXT)
### Forming Identifiers

<table>
<thead>
<tr>
<th></th>
<th><strong>FORTRAN</strong></th>
<th><strong>PASCAL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Characters in Letter Set</strong></td>
<td>A-Z AND $</td>
<td>A-Z AND _ (underscore)</td>
</tr>
<tr>
<td><strong>First Character</strong></td>
<td>MUST BE IN LETTER SET</td>
<td>MUST BE IN LETTER SET</td>
</tr>
<tr>
<td><strong>Following Characters</strong></td>
<td>LETTER OR DIGIT</td>
<td>LETTER OR DIGIT</td>
</tr>
</tbody>
</table>

### Examples

<table>
<thead>
<tr>
<th></th>
<th><strong>FORTRAN</strong></th>
<th><strong>PASCAL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>TOTAL</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>A136</td>
<td>A136</td>
<td></td>
</tr>
<tr>
<td>$787</td>
<td>$787 (ERROR)</td>
<td></td>
</tr>
<tr>
<td>_ABLE (ERROR)</td>
<td>_ABLE</td>
<td></td>
</tr>
<tr>
<td>2 HITS (ERROR)</td>
<td>2 HITS (ERROR)</td>
<td></td>
</tr>
<tr>
<td>CAN,DO (ERROR)</td>
<td>CAN,DO (ERROR)</td>
<td></td>
</tr>
<tr>
<td>CAN.DO (ERROR)</td>
<td>CAN.DO</td>
<td></td>
</tr>
</tbody>
</table>

### Spaces Within Identifiers

<table>
<thead>
<tr>
<th></th>
<th><strong>LEGAL IN FORTRAN</strong></th>
<th><strong>ILLEGAL IN PASCAL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examples</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIT IT</td>
<td>HIT IT (ERROR)</td>
<td></td>
</tr>
<tr>
<td>MISS IT</td>
<td>MISS IT (ERROR)</td>
<td></td>
</tr>
</tbody>
</table>

### Maximum Significant Length

- **FORTRAN**: 6 characters
- **PASCAL**: 80 characters

<table>
<thead>
<tr>
<th></th>
<th><strong>SAME</strong></th>
<th><strong>DIFFERENT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD COST (7 CHARACTERS)</td>
<td>ADD_COST</td>
<td></td>
</tr>
<tr>
<td>ADD COSTE (8 CHARACTERS)</td>
<td>ADD_COSTE</td>
<td></td>
</tr>
</tbody>
</table>

1.8
VARIABLES

FORTRAN

DEFAULT
I THRU N INTEGER
A THRU H
O THRU Z REAL
$

DECLARATION OPTIMAL

EXAMPLE

C PROGRAM ONE
IMPLICIT INTEGER (C-F)
INTEGER A,B,SAM
REAL I,J
LOGICAL R

A=2
I=3.0
K=4
B=K+A

BEGIN
A:=2;
I:=3.0;
K:=4; "ERROR AS K NOT DECLARED"
B:=K+A "ERROR AS K NOT DECLARED"
END

PASCAL

PROGRAM ONE
NO IMPLICIT IN PASCAL
VAR A,B,SAM: INTEGER;
I,J: REAL;
R: BOOLEAN;

BEGIN
A:=2;
I:=3.0;
K:=4; "ERROR AS K NOT DECLARED"
B:=K+A "ERROR AS K NOT DECLARED"
END.
### NUMERICAL TYPES

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTEGER</strong></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>350</td>
<td>350</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-475</td>
<td>-475</td>
</tr>
<tr>
<td>25E6</td>
<td>25E6</td>
</tr>
<tr>
<td>27E-4</td>
<td>27E-4</td>
</tr>
<tr>
<td>-16E-3</td>
<td>-16E-3</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>2,346</td>
<td>2,346</td>
</tr>
<tr>
<td><strong>REAL</strong></td>
<td></td>
</tr>
<tr>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>.25</td>
<td>.25</td>
</tr>
<tr>
<td>27.493</td>
<td>27.493</td>
</tr>
<tr>
<td>542</td>
<td>542</td>
</tr>
<tr>
<td>-2.47</td>
<td>-2.47</td>
</tr>
<tr>
<td>25.0E6</td>
<td>25.0E6</td>
</tr>
<tr>
<td>-432.15E2</td>
<td>-432.15E2</td>
</tr>
<tr>
<td>637.2E-4</td>
<td>637.2E-4</td>
</tr>
<tr>
<td>-234.34E-5</td>
<td>-234.34E-5</td>
</tr>
<tr>
<td>5.2E1,3</td>
<td>5.2E1,3</td>
</tr>
</tbody>
</table>

MUST HAVE A DIGIT BEFORE AND AFTER THE DECIMAL POINT
ASSIGNMENT

FORTRAN

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>=</th>
</tr>
</thead>
</table>

PASCAL

<table>
<thead>
<tr>
<th></th>
<th>:=</th>
</tr>
</thead>
</table>

TYPE RULES

INTEGER AND REAL MAY BE MIXED.
BOOLEAN MUST BE ASSIGNED TO BOOLEAN.

IN ALL CASES THE TYPE MUST AGREE.

INTEGER = INTEGER
REAL = REAL
REAL = INTEGER
BOOLEAN = BOOLEAN
INTEGER = REAL
REAL = BOOLEAN (ERROR)
BOOLEAN = INTEGER (ERROR)

( THERE ARE OTHERS )
ARITHMETIC OPERATORS

**FORTRAN**

+  
-  
*  

**PASCAL**

OPERANDS MUST BE SAME TYPE

+  
-  
*  

INTEGER OR REAL RESULTS

INTEGER OR REAL

NONE IN PASCAL

MUST WRITE PROCEDURE

**INTEGER DIVISION**

/  

**REAL DIVISION**

/  

**INTEGER REMAINDER**

NOT AN OPERATOR USE

FUNCTION MOD(I, J)

**I MOD J**

(OPERANDS INTEGER ONLY)

I DIV J

(OPERANDS INTEGER ONLY)

(A/B

(OPERANDS REAL)

(RESULT INTEGER)

(RESULT REAL)
### RELATIONAL OPERATORS

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EQ.</td>
<td>=</td>
</tr>
<tr>
<td>.NE.</td>
<td>&lt;&gt;</td>
</tr>
<tr>
<td>.LT.</td>
<td>&lt;</td>
</tr>
<tr>
<td>.GT.</td>
<td>&gt;</td>
</tr>
<tr>
<td>.LE.</td>
<td>&lt;=</td>
</tr>
<tr>
<td>.GE.</td>
<td>&gt;=</td>
</tr>
<tr>
<td><strong>NONE</strong></td>
<td><strong>IN</strong> (SET MEMBERSHIP)</td>
</tr>
</tbody>
</table>

#### EXAMPLES

- A.LT.B  \( \Rightarrow \) A\(<\)B
- A.GE.B  \( \Rightarrow \) A\(\geq\)B
- A.EQ.B  \( \Rightarrow \) A=B
## LOGICAL OPERATORS

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NOT.</td>
<td>NOT</td>
</tr>
<tr>
<td>.AND.</td>
<td>&amp;</td>
</tr>
<tr>
<td>.OR.</td>
<td>OR</td>
</tr>
<tr>
<td>OPERATOR PRECEDENCE</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td><strong>FORTRAN</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FUNCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>SUBPROGRAMS</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FUNCTION, PROCEDURE, NOT</strong></td>
<td></td>
</tr>
<tr>
<td>****</td>
<td>2</td>
</tr>
<tr>
<td><strong>NOT IN PASCAL-USE PROCEDURE</strong></td>
<td></td>
</tr>
<tr>
<td>*<strong>,/</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>+, -</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>.EQ., .NE., .LT.,</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>.LE., .GE., .GT.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>.NOT.</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>.AND.</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>.OR.</strong></td>
<td>8</td>
</tr>
</tbody>
</table>

*NOTE: MAJOR DIFFERENCE IS IN LOGICAL OPERATOR PRECEDENCE*
LIKE FORTRAN - EVALUATE EQUAL PRECEDENCE LEFT TO RIGHT. EXPRESSIONS IN ( ) ARE EVALUATED INDEPENDENT OF PRECEDING AND SUCCEEDING OPERATORS.
EXAMPLE OF LOGICAL OPERATOR PRECEDENCE DIFFERENCE

FORTRAN

PASCAL

1.17
GROUP 2: CONTROL STRUCTURES

IF THEN (2.2)

IF THEN ELSE (2.4)

CASE (2.5)

DO, WHILE, & REPEAT FLOW CHARTS (2.8)

LOOP EXAMPLES (2.13)

EXIT FROM THE MIDDLE OF A LOOP (2.19)
IF B THEN

B = BOOLEAN EXPRESSION
S = STATEMENT

EXAMPLES

FORTRAN                      PASCAL

SINGLE STATEMENT             IF A=B THEN J:=1;
IF(A.EQ.B) J=1

MULTIPLE STATEMENT           IF A=B THEN
IF(A.NE.B) GO TO 100        BEGIN  J:=1;
J=1                          K:=1;
K=1                          L:=1
L=1
100 CONTINUE                END;

NOTE: NO SEMICOLON AFTER
STATEMENT PRECEDING END
NEVER PUT A SEMICOLON AFTER THEN

\[
\text{IF } A > B \text{ THEN; } J := 1;
\]

\[
J := 1 \text{ WILL ALWAYS EXECUTE.}
\]

\[
\text{IF } A > B \text{ THEN;}
\]
\[
\begin{align*}
\text{BEGIN} & \quad J := 1; \\
& \quad K := 1; \\
& \quad L := 1 \\
\text{END};
\end{align*}
\]

THESE STATEMENTS WILL ALWAYS EXECUTE.

MUST USE BEGIN END IF MORE THAN ONE STATEMENT IS TO BE EFFECTED.

\[
\text{IF } A > B \text{ THEN}
\]
\[
\begin{align*}
J := 1; \\
K := 1; \\
L := 1;
\end{align*}
\]

THESE STATEMENTS ALWAYS EXECUTE
IF B THEN $S_1$ ELSE $S_2$

B=BOOLEAN EXPR.
$S_1$=STATEMENT
$S_2$=STATEMENT

FORTRAN

SINGLE STATEMENTS
IF(A. GT. B) J=1
IF(A. LE. B) J=2

MULTIPLE STATEMENTS
IF(A. LE. B) GO TO 100
J=1
K=1
GO TO 200
100 J=2
K=2
200 CONTINUE

PASCAL

IF A > B THEN J:=1
ELSE J:=2;
NOTE: NEVER PLACE A SEMICOLON BEFORE OR AFTER AN ELSE.
IF A > B THEN
BEGIN J:=1;
K:=1 END
ELSE
BEGIN J:=2;
K:=2 END;
CASE

E

= L_1

= L_2

= L_3

\ldots

= L_N

S_1

S_2

S_3

S_N

RULES:
1. EXPRESSION TYPE AND LABEL TYPE MUST AGREE
2. EXPRESSION MAY NOT BE TYPE REAL
3. LABELS MUST BE CONSTANTS

WARNING: AN INTEGER VARIABLE = INTEGER CONSTANT, THESE ARE DEFINED IN GROUP 3.

FORMAT

CASE E OF

L_1:S_1;
L_2:S_2;
L_3:S_3;
\ldots
\ldots
L_N:S_N

END

2.5
### CASE EXAMPLE

**FORTRAN**

```fortran
C ASSUME I,J,K ARE INTEGERS
C AND I.GE.1, I.LE.3

: ...

GO TO(100,200), I

100 J=1
    K=1
    GO TO 400

200 J=2
    GO TO 400

300 J=3
    K=3

400 CONTINUE
```

**PASCAL**

```
"ASSUME I,J,K DECLARED INTEGER
AND I>=1, I<=3"

: ...

CASE I OF

1: BEGIN J:=1
    K:=1 END;

2: J:=2;

3: BEGIN J:=3;
    K:=3 END END;
```

**NOTE:** LABELS ARE GLOBAL TO THE PROGRAM MODULE

**NOTE:** LABELS ARE LOCAL TO THE CASE STATEMENT (NOT GLOBAL TO THE WHOLE MODULE)
FORTRAN

C ASSUME I, J ARE INTEGER
C I .GE. 1, I .LE. 6

: 
IF(I.EQ.1.OR.I.EQ.2) J=1
IF(I.EQ.3.OR.I.EQ.4) J=2
IF(I.EQ.5.OR.I.EQ.6) J=3

: 

PASCAL

"ASSUME I, J ARE INTEGER AND I \geq 1, I \leq 6"

: 

CASE I OF
1,2: J:=1;
3,4: J:=2;
5,6: J:=3
END;

C ASSUME I, J, K, L, FLAG
C ARE INTEGER

: 

IF(I.EQ.J) FLAG=1
IF(I.EQ.K) FLAG=2
IF(I.EQ.L) FLAG=3

: 

PASCAL

"ASSUME I, FLAG DECLARED INTEGER J,K,L ARE INTEGER CONSTANTS"

CASE I OR
J: FLAG:=1;
K: FLAG:=2;
L: FLAG:=3
END;

WARNING: INTEGER CONSTANTS ARE DIFFERENT FROM INTEGER VARIABLES!
PASCAL
CASE EXAMPLE

"ASSUME I, J ARE INTEGER AND I = 1, I = 6"
I := 3
CASE I OF
1, 2: J := 6;
5, 6: J := 8
END;

"ASSUME I, J ARE INTEGERS AND I = 1, I = 6"

I := 7
RUN TIME ERROR!!
CASE I OF
1, 2: J := 6;
5, 6: J := 8
END;

2.7.1
THE FORTRAN DO

1. $S_1$ thru $S_N$ execute at least once.

2. Counter and control must be integer and greater than zero.

3. The counter may only be increment by a positive integer.
PASCAL

FOR \( V := E_1 \) TO \( E_2 \) DO \( S \);

\( V := E_1 \)

\( V := \text{succ}(v) \)

\( V <= E_2 \)

\( S \)

\( V = \text{variable} \)
\( E = \text{expression} \)
\( S = \text{statement} \)

1. \( V, E_1, \) AND \( E_2 \) MUST BE OF SAME TYPE

2. TYPE MAY BE: CHARACTER
   BOOLEAN
   INTEGER

   MAY NOT BE: CONSTANT
   RECORD FIELD
   FUNCTION IDENTIFIER
   ARRAY ELEMENT
   REAL

3. \( V \) MAY NOT BE CHANGED IN \( S \).

4. \( V \) IS UNDEFINED AFTER COMPLETION OF \text{FOR} \ STMTATION

5. \( V \) IS ALWAYS INCREMENTED BY ONE FOR INTEGERS.
1. \( V, E_1 \) AND \( E_2 \) MUST BE OF COMPATABLE TYPE

2. \( V \) TYPE MAY BE: CHARACTER
   BOOLEAN
   INTEGER
   MAY NOT BE: REAL
   CONSTANT
   RECORD FIELD
   FUNCTION IDENTIFIER
   ARRAY ELEMENT

3. \( V \) MAY NOT BE CHANGED IN \( S \).

4. \( V \) IS UNDEFINED AFTER COMPLETION OF FOR STATEMENT.

5. \( V \) IS ALWAYS INCREMENTED BY ONE FOR INTEGERS.
2. Normally used when number of repetitions required

3. S will not be executed if B is false.

4. B is any legal Boolean expression.
1. **Body may be a sequence of statements.** (Note: **begin**, **end** not required)

2. **Normally used when number of repetitions required is not known.**

3. **$S_1;...;S_N$ is executed at least once.**

4. **B is any legal Boolean expression.**
FORTRAN

C I INTEGER, A,B REAL
C BEGIN PROGRAM
B=13.0
A=1.0
DO 100 I=1,10
A = A + A/B
100 CONTINUE
END

PASCAL

"ASSUME I DECLARED INTEGER
A,B DECLARED REAL"
BEGIN
B:=13.0
A:=1.0;
FOR I:=1 TO 10 DO A:=A+A/B;
END.
FORTRAN

C ASSUME ALL VARIABLES DECLARED
C AND INITIALIZED
C BEGIN PROGRAM

. ...

DO 100 I=1,10
A=A*B
C=C+C

100 CONTINUE

. ...

END

PASCAL

"ASSUME VARIABLES DECLARED
AND INITIALIZED"
BEGIN

. ...

FOR I:=1 TO 10 DO
BEGIN
A:=A*B;
C:=C+C
END;

. ...

END.
FOR DOWNTO DO EXAMPLE

FORTRAN

C ASSUME VARIABLES DECLARED
C AND INITIALIZED
C BEGIN PROGRAM

: : :
DO 100 J=1,10
I=11-J
B=B*B
A(I)=B
100 CONTINUE

: : :
END

PASCAL

"ASSUME VARIABLES DECLARED AND INITIALIZED"
BEGIN

: : :
FOR I:=10 DOWNTO 1 DO
BEGIN
B:=B*B
A(.I.):=B
END;

: : :
END.
WHILE DO EXAMPLES

FORTRAN

C     I,K INTEGER
C     A,B REAL

C     BEGIN PROGRAM
    :
    :
C     ASSUME I AND K ASSIGNED
C     INTEGER VALUE
    :
    B:=2.0
    A:=1.0
100    IF(I.LE.K)GO TO 200
    A:=A+A/B
    I:=I-1
    GO TO 100
200    CONTINUE
    :
    :
END

PASCAL

"ASSUME I,K DECLARED INTEGER
A,B DECLARED REAL"

BEGIN
    :
    :
"ASSUME I AND K ASSIGNED SOME
INTEGER VALUE"
    :
    B:=2.0;
    A:=1.0;
WHILE I > K DO
    BEGIN A:=A+A/B;
    I:=I-1
    END;
    :
    :
END.

2.16
**REPEAT UNTIL EXAMPLES**

**INCREMENT EXAMPLE**

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>C ASSUME J,B ARE INTEGERS</td>
<td>&quot;ASSUME B,I DECLARED INTEGERS</td>
</tr>
<tr>
<td>C A IS INTEGER ARRAY INDEXED</td>
<td>A IS INTEGER ARRAY INDEXED</td>
</tr>
<tr>
<td>C FROM 1 TO 100</td>
<td>FROM 1 TO 100&quot;</td>
</tr>
<tr>
<td>C BEGIN PROGRAM</td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>B=2 DO 100 I=2,100 A(I)=B*I</td>
<td>B:=2; I:=0; REPEAT</td>
</tr>
<tr>
<td>100 CONTINUE</td>
<td>I:=I+2; A(.I.,):=B*I UNTIL I=100;</td>
</tr>
<tr>
<td>END</td>
<td>END.</td>
</tr>
</tbody>
</table>

2.17
**FORTRAN**

```fortran
C ASSUME ALL VARIABLES DECLARED
C BEGIN PROGRAM

QUOTNT=0
IF(X.GT.Y)GO TO 200
REMAIN=Y
100 REMAIN=REMAIN-X
QUOTNT=QUOTNT+1
IF(X.LE.REMAIN)GO TO 100
200 CONTINUE

END
```

**PASCAL**

```pascal
"ASSUME ALL VARIABLES DECLARED INTEGER"
BEGIN

IF X > Y THEN QUOTIENT := 0
ELSE BEGIN
  REMAINDER := Y;
  QUOTIENT := 0;
  REPEAT
    REMAINDER := REMAINDER - X;
    QUOTIENT := QUOTIENT + 1
  UNTIL X > REMAINDER
END;

END.
```
C BEGIN PROGRAM

OK = TRUE
DO 100 I = 1, 100
COST = A(I)
TOTAL = TOTAL + COST
IF (TOTAL .GT. BALANCE) GO TO 200
A(I) = 0.0
100 CONTINUE
GO TO 300
300 IF (OK .NE. TRUE) GO TO 400

END

BEGIN

OK = TRUE;
I := 0;
REPEAT
I := I + 1;
COST := A(I);
TOTAL := TOTAL + COST;
IF (TOTAL = BALANCE)
THEN A(I) := 0.0
ELSE OK := FALSE
UNTIL ((I = 100) OR (OK = FALSE));
IF OK = TRUE THEN BEGIN

END;

END.
GROUP 3  SIMPLE DATA TYPES

DECLARATIONS  (3.2)

KINDS OF TYPES  (3.4)

CONSTANTS  (3.5)

NUMBERS (INTEGER AND REAL)  (3.6)

CHARACTERS  (3.8)

BOOLEANS  (3.9)

ENUMERATIONS  (3.10)

STRINGS  (3.13)
DECLARATIONS

FORTRAN

INTEGER I,Q
REAL R,L
LOGICAL B
C DEFAULT K IS INTEGER
C DEFAULT X IS REAL

PASCAL

VAR I,Q: INTEGER;
VAR R,L: REAL;
VAR B: BOOLEAN;
"NO DEFAULTS"

OR

VAR I,Q: INTEGER;
R,L: REAL;
B: BOOLEAN:

OR

TYPE INT= INTEGER;
RE= REAL;
LOG= BOOLEAN;

VAR I,Q: INT;
R,L: RE;
B: LOG;

3.2
NOTE

> THE TYPE STATEMENT DEFINES
  - A TEMPLATE FOR THE INTERNAL LAYOUT OF VARIABLES WHICH
    WILL BE DECLARED LATER; (IE. IT DECLARES TYPE IDENTIFIERS)

  - IT DOES NOT DECLARE OR ALLOCATE SPACE FOR RUN TIME VARIABLES.

> THE VAR STATEMENT DECLARES
  - RUN TIME VARIABLES AND SETS THE AMOUNT AND LAYOUT OF THE
    SPACE THAT THE VARIABLES WILL USE AT RUN TIME.
  - IT DOES NOT INITIALIZE THE VALUE OF ANY VARIABLE.

> THE CONST STATEMENT
  - INITIALIZE A VALUE FOR AN IDENTIFIER, WHICH
    CANNOT BE CHANGED.
KINDS OF TYPES IN PASCAL

ORDERED SCALAR
PRIMITIVES ARE

INTEGER
BOOLEAN 
REAL
CHAR........SINGLE CHARACTER

OTHER PRIMITIVES ARE

POINTER...EFFECTIVELY, AN ADDRESS

SCALAR ENUMERATIONS ARE
USER DEFINED TO BE A LEGAL RANGE
OF VALUES WITH A DEFINED ORDERING

COMPOSITE STRUCTURES
INCLUDE ARRAY,
  RECORD
  SET

★ NEW USER DEFINED TYPES
ARE COMBINATIONS OF ALL OF THE ABOVE

CONSTANTS ARE
NOT REALLY TYPES (INTEGER, REAL,
  STRING)

3.4
CONSTANTS

An identifier which is
- initialized at compile time
- has integer, real, char, string, or boolean value
- cannot be assigned to at run time

C MAIN PROGRAM

INTEGER A, B, D, F(2)
REAL E, G
LOGICAL C
EQUIVALENCE (A), (B)
DATA B, C, D, E, F

'/Y_ _ _', TRUE., 12, 12.6,
'MABL','E_ _ '/

PROGRAM 'MAIN';
CONST B = 'Y';
C = TRUE;
D = 12;
E = 12.6;
F = 'MABLE_';
A = B;
VAR G: REAL;

These are not really constants!

WARNING: NOT AN :=

3.5
**PASCAL**

**NUMBERS (INTEGER AND REAL)**

MAXIMUM INTEGER = 32767
MAXIMUM REAL = 10^{38}
EPS \approx 10^{-6}
ZERO \approx 10^{-38}

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>OPERAND 1</th>
<th>OPERAND 2</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ - *</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>R</td>
<td>ERROR</td>
</tr>
<tr>
<td>/</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>R</td>
<td>ERROR</td>
</tr>
<tr>
<td>DIV</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>RELATIONAL</td>
<td>I</td>
<td>I</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>R</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>R</td>
<td>ERROR</td>
</tr>
</tbody>
</table>
PASCAL

FUNCTIONS ON INTEGERS

PRED (X)..................YIELDS X-1
SUCC(X)..................YIELDS X+1
ABS(X)...................YIELDS ABSOLUTE VALUE OF X
    (A NON-NEGATIVE INTEGER)
CHR(X)...................YIELDS A CHARACTER WITH ORDINAL
    VALUE X, FOR 0 ≤ X ≤ 127
    EG: CHR(65) IS 'A'
CONV(X)..................REAL VALUE X

PASCAL

FUNCTIONS ON REALS

ABS(X)....................ABSOLUTE VALUE
TRUNC(X)..................TRUNCATED INTEGER VALUE OF X
CHAR

SINGLE CHARACTER

PROGRAM MAIN;
VAR A : CHAR;
B : BOOLEAN;
BEGIN
ILLEGAL IN FORTRAN
A : = 'Z';
ILLEGAL IN FORTRAN
B : = A > 'C'
END.

OPERATIONS

RELATIONAL OPERATORS ARE ALL DEFINED FOR CHAR

FUNCTIONS

ORD(X).................YIELDS INTEGER ORDINAL
EG: ORD('B') = ORD ('A') + 1
ORD('A') = 65

SUCCEX(X)..............SUCCESSOR
SUCC('A') is 'B'

PRED(X)..............PREDECESSOR
PRED('B') is 'A'

3.8
<table>
<thead>
<tr>
<th>ASCII CHARACTER SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 nul</td>
</tr>
<tr>
<td>1 soh</td>
</tr>
<tr>
<td>2 stx</td>
</tr>
<tr>
<td>3 etx</td>
</tr>
<tr>
<td>4 eot</td>
</tr>
<tr>
<td>5 enq</td>
</tr>
<tr>
<td>6 ack</td>
</tr>
<tr>
<td>7 bell</td>
</tr>
<tr>
<td>8 bs</td>
</tr>
<tr>
<td>9 ht</td>
</tr>
<tr>
<td>10 lf</td>
</tr>
<tr>
<td>11 vt</td>
</tr>
<tr>
<td>12 ff</td>
</tr>
<tr>
<td>13 cr</td>
</tr>
<tr>
<td>14 so</td>
</tr>
<tr>
<td>15 si</td>
</tr>
<tr>
<td>16 dle</td>
</tr>
<tr>
<td>17 dc1</td>
</tr>
<tr>
<td>18 dc2</td>
</tr>
<tr>
<td>19 dc3</td>
</tr>
<tr>
<td>20 dc4</td>
</tr>
<tr>
<td>21 nak</td>
</tr>
<tr>
<td>22 syn</td>
</tr>
<tr>
<td>23 etb</td>
</tr>
<tr>
<td>24 can</td>
</tr>
<tr>
<td>25 em</td>
</tr>
<tr>
<td>26 sub</td>
</tr>
<tr>
<td>27 esc</td>
</tr>
<tr>
<td>28 fs</td>
</tr>
<tr>
<td>29 gs</td>
</tr>
<tr>
<td>30 rms</td>
</tr>
<tr>
<td>31 us</td>
</tr>
</tbody>
</table>

3.8.1
BOOLEANS

FORTран   ПАСКАЛ

\_TRUE.\_   TRUE
\_FALSE.\_   FALSE

ПАСКАЛ
ОПЕРАЦИИ НА ТИПЕ ИСТИННОСТИ

\&, \_OR, \_NOT \_для \_образования \_в 
и \_перечных \_выражений 

РЕЛЯЦИОНАЛЬНОЕ \_выражение \_в \_форме 

프로그램 "MAIN";
VAR  BOY : BOOLEAN;
    HT : INTEGER;
BEGIN

\_...

    BOY: = HT > 190 ;
\_...

END.

3.9
ENUMERATIONS

CAN BE - AN ORDERED SEQUENCE OF INTEGERS
- ORDERED SEQUENCE FALSE, TRUE
- ORDERED SEQUENCE OF CHARACTERS
- A SEQUENCE OF IDENTIFIERS (UP TO 120)
  (THE SEQUENCE DEFINES AN ORDERING)
- A SUBSEQUENCE OF ANY OTHER ENUMERATION
  (CALLED A SUBRANGE)
PROGRAM MAIN;
TYPE PEOPLE = (BILL, RICH, RUS, RON, JAN);
VAR E1: 0..10;
   E2: (FALSE, TRUE);
   E3: 'A'..'Z';
   E5: PEOPLE;
   E4: BILL..RON;
BEGIN
   E1: = 9;
   E2: = RUS > RICH;
   E3: = SUCC('W');
   E5: = RON;
   E4: = E5
END.
FUNCTIONS ON ENUMERATIONS

ALL RELATIONAL OPERATIONS ARE DEFINED
SUCCE, PRED ARE DEFINED, EG.
SUCCE(RON) IS RUS , BUT SUCCE(JAN) IS NOT DEFINED

ENUMERATIONS ARE USED IN CASE STATEMENTS, IN VARIANT RECORDS.
EG; CONTINUING LAST EXAMPLE

GO TO (101,101,102,101,102),E5  CASE E5 OF
101 E2 = .TRUE.
    GO TO 103
102 E3 = .FALSE.
102 CONTINUE
    RICH, BILL, RON: E2:= TRUE;
    RUS, JAN : E2: = FALSE
END;

3.12
PROGRAM MAIN:

VAR NAME1, NAME2: STRING6;
E2: BOOLEAN;
BEGIN
NAME1 = 'JOHNNY';
NAME2 = 'BOBBY';
IF NAME2 < NAME1 THEN E2 := TRUE
ELSE E2 := FALSE;
END.

NAME1 := 'JOHN';
NAME1(3) := 'A';

3.13
1. A STRING MUST CONTAIN AN EVEN NUMBER OF CHARACTERS.

2. FOR STRING ASSIGNMENT (WITH EXPLICIT INDEXING), THE WHILE STRING MUST BE ASSIGNED. EG. IN THE LAST EXAMPLE

   NAME1: 'JOHN' IS ILLEGAL

OPERATIONS ON STRINGS

RELATIONAL OPERATORS - THE STRINGS MUST BE THE SAME LENGTH.

EG: 'JOHNNY' 'BILL' IS ILLEGAL
GROUP 4 ARRAYS

ARRAYS (4.2)

EXAMPLES - ENUMERATION INDEXING (4.3)

NON-NUMERIC COMPONENTS (4.7.2)

OPERATIONS IN ARRAYS (4.9)

4.1
THE ARRAY TYPE

FORTRAN

FIXED NUMBER OF
COMPONENTS ALL OF
SAME TYPE

COMPONENT TYPES:
  INTEGER
  REAL
  DOUBLE PRECISION
  LOGICAL
  COMPLEX
  (LITERAL POSSIBLE)

SUBSCRIPTS:
  NONZERO INTEGER
  USE "(I,J)"

DECLARATION:

  WITH DIMENSION, INTEGER,
  REAL, ETC., STATEMENTS

STATIC ALLOCATION OF
VARIABLES (EXCEPT
FOR DUMMY ARGUMENTS)

PASCAL

FIXED NUMBER OF
COMPONENTS ALL OF
SAME TYPE

COMPONENT TYPES:
  MAY BE ANY TYPE!

SUBSCRIPTS:
  ANY ENUMERATION!
  USE "(.I,J.)"

DECLARATION:

  MAY BE DECLARED
  IN TYPE OR VAR STATEMENT

STATIC ALLOCATION OF
VARIABLES IN MAIN
PROGRAM
ARRAY EXAMPLES

FORTRAN

C MAIN PROGRAM

: 
: 
DIMENSION A(10), I(10,20)

: 
: 
C BEGIN PROGRAM

: 
J = 2
A(J) = 2.5
I(J,J) = 3

: 
END

PASCAL

PROGRAM MAIN;
VAR

J: INTEGER;
A: ARRAY(1..10.) OF REAL;
I: ARRAY(1..10,1..20.) OF INTEGER;

BEGIN

J: = 2;
A(J): = 2.5;
I(J,J): = 3;

END.
ARRAY EXAMPLES

FORTRAN

C MAIN PROGRAM
.
.
REAL B(20),C(20)
.
.
C I MUST BE IN RANGE 1 TO 20
B(1) = 10.
.
END

PASCAL

PROGRAM MAIN;
TYPE IROW = 1..20;
VEC = ARRAY (.IROW.) OF REAL;
VAR INDEX:IROW;
B,C : VEC;

BEGIN
INDEX:= 20
B(INDEX.): = 10.0;
INDEX:= 21;

END.

PROGRAM "MAIN";
VAR A: ARRAY (.'A'..'Z'.) OF REAL;
BEGIN
.
.
A(.'W'.): = 3.0;
.
.
END.
PROGRAM MAIN;

VAR A: ARRAY (.1..10.) OF ARRAY (.11..20.) OF REAL;
B: REAL;

BEGIN

B = A(.4.)(.15.);

END.
PROGRAM MAIN;

VAR A: ARRAY (.1..10.) OF ARRAY (.11..20.) OF REAL;
    B: REAL;

BEGIN
    B := A(.4)(.15.);
    B := A(.4,15.);

END.

NOTE: INDEXING OF MULTIPLE DIMENSION ON ARRAYS CAN BE DONE IN EITHER FORM
C       MAIN

RED = 1, WHITE = 2,  BLUE = 3
INTEGER  FLAG (3,2), X(6)
DATA X/'BRIG', 'HT_ _', 'PURE', '_ _ _',
 *       'COOL', '_ _ _' /

FLAG(1,1) = X(1)
FLAG(1,2) = X(2)
FLAG(2,1) = X(3)
FLAG(2,2) = X(4)
FLAG(3,1) = X(5)
FLAG(3,2) = X(6)

END
PASCAL

ARRAY EXAMPLE

PROGRAM "MAIN";

TYPE COLOR = (RED, WHITE, BLUE);
  STRING6 = ARRAY [.1..6.] OF CHAR;
VAR FLAG: ARRAY (.COLOR.) OF STRING6;

BEGIN

  FLAG(.RED.): = 'BRIGHT';
  FLAG(.WHITE.): = 'PURE__';
  FLAG(.BLUE.): = 'COOL__';

END.
NOTE ARRAYS REQUIRE STATIC ALLOCATION

EG.

VAR MAX: 100..500;

VAR BILL: ARRAY (1..MAX) OF INTEGER;

; ;

IS ILLEGAL FOR DECLARING A NEW ARRAY BECAUSE
1..MAX IS AN ILLEGAL ENUMERATION SINCE MAX IS
NOT A CONSTANT.
PASCAL

OPERATIONS ON WHOLE ARRAYS

:= }
Provided that the arrays are the same type and size.
RESULT IS TRUE OR FALSE

= }

<> }

REMEMBER: STRINGS ARE AN EXCEPTION:
ALL THE RELATIONAL OPERATORS WORK FOR STRINGS OF THE SAME SIZE.

OPERATIONS ON SINGLE COMPONENTS

ANY OPERATION THAT IS LEGAL FOR THAT TYPE OF COMPONENT.
GROUP 5: RECORDS

GENERALIZATIONS (5.2)

EXAMPLES: MIXED COMPONENTS (5.3.1)

REFERENCING COMPONENTS ( WITH DO ) (5.3.2)

NESTED RECORDS (5.4)

ARRAYS OF RECORDS (5.5)

CONSTRAINTS (5.6)

VARIANT RECORD EXAMPLE (5.8)
**RECORD**

**IS A GENERALIZATION OF ARRAY**

<table>
<thead>
<tr>
<th>ARRAYS</th>
<th>RECORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL COMPONENTS ARE SAME TYPE</strong></td>
<td><strong>COMPONENTS CAN BE DIFFERENT TYPES</strong></td>
</tr>
<tr>
<td><strong>ALL COMPONENTS SAME SIZE (IE. ARRAY IS RECTANGULAR)</strong></td>
<td><strong>COMPONENTS CAN BE DIFFERENT SIZES</strong></td>
</tr>
<tr>
<td><strong>TYPES OF COMPONENTS FIXED AT COMPILE TIME (STATIC)</strong></td>
<td><strong>POSSIBLE TYPES OF COMPONENTS (CALL VARIANTS) ENUMERATED AT COMPILE TIME (STATIC); ACTUAL TYPE OF COMPONENT CAN BE DETERMINED AT RUN TIME (DYNAMIC).</strong></td>
</tr>
</tbody>
</table>
| COMPONENTS ARE GIVEN INDICES; EG.  
  1,2,3,...  
  'A','B','C'...  
  RED, WHITE, BLUE,... | COMPONENTS ARE GIVEN NAMES; EG.  
  FIELD1,FIELD2,FIELD3  
  NAME, AGE, SSNUM |
| COMPONENTS ARE ACCESSED BY INDEXING; EG.  
  BILL(.1,)  
  BOB(.A)  
  HUE(.RED,) | COMPONENTS ARE ACCESSED BY NAME QUALIFICATION, EG.  
  LINE,FIELD2  
  PERSON,SSNUM |

*WARNING*: SPACE FOR THE MAXIMUM SIZE VARIANT IS ALLOCATED STATICALLY.
FORTRAN

REAL FIELD4

INTEGER FIELD1, FIELD2, LINE(13)

LOGICAL FIELD3(10)

EQUIVALENCE (LINE(1), FIELD1), (LINE(2), FIELD2),
          (LINE(3), FIELD3&C), (LINE(13), FIELD4)

DATA I1 = 'A_ _ _'

FIELD1 = I1

FIELD2 = 3

FIELD3(7) = .TRUE.

FIELD4 = 13.31

C THESE ARE NOT REALLY THE SAME AS

C RECORD FIELD NAMES

5.3.1
PASCAL

EXAMPLE 1

VAR: LINE: RECORD
FIELD1: CHAR;
FIELD2: INTEGER;
FIELD3: ARRAY[1..10] OF BOOLEAN;
FIELD4: REAL
END;

BEGIN
LINE,FIELD1: = 'A';
LINE,FIELD2: = 3;
LINE,FIELD3[7]: = TRUE;
LINE,FIELD4: = 13.31
END.

OR

BEGIN
WITH LINE DO
BEGIN FIELD1: = 'A';
FIELD2: = 3;
FIELD3[7]: = TRUE;
FIELD4: = 13.31
END;
END.

5.3.2
EXAMPLE:

"RECORDS CAN BE NESTED"

TYPE STRING12 = ARRAY(1..12) OF CHAR;
STRING8 = ARRAY(1..8) OF CHAR;

VAR MAN: RECORD
   NAME: STRING12
   AGE: INTEGER;
   WIFE: RECORD
      AGE: INTEGER
      NAME: STRING8;
      CHILDREN: BOOLEAN
   END;

OR

"RECORDS CAN BE NESTED THIS WAY"

TYPE STRING8 = ARRAY(1..8) OF CHAR;
TYPE WREC = RECORD
   AGE: INTEGER;
   NAME: STRING8;
   CHILDREN: BOOLEAN
END;

VAR MAN: RECORD
   AGE: INTEGER;
   NAME: STRING8;
   WIFE: WREC
END;

BEGIN
   WITH MAN DO BEGIN
      NAME := 'BILLY_ _ _';
      AGE := 30;
      WIFE.NAME := 'WALLY_ _ _';
      WIFE.CHILDREN := TRUE
   END;

   5.4
EXAMPLE

"AN ARRAY OF RECORDS"

TYPE STRING8 = ARRAY (.1..8.) OF CHAR;
TYPE LINE = RECORD
    FIELD1: CHAR;
    FIELD2: INTEGER;
    FIELD3: STRING8;
    FIELD4: REAL
END;

VAR TABLE: ARRAY (.1..100.) OF LINE;

BEGIN

    TABLE(.13.).FIELD3 := 'IT WORKS';
    TABLE(.13.).FIELD2 := 87;
    TABLE(.13.).FIELD4 := 13.31;

    .
    .
    .

END.
VARIABLES MUST BE DECLARED BEFORE THEY ARE USED

TYPE STRING8 = ARRAY(1..8) OF CHAR;
TYPE MAN = RECORD
  NAME: STRING8;
  AGE: INTEGER;
  WIFE: WOMAN ← ILLEGAL
END;

TYPE WOMAN = RECORD
  NAME: STRING8;
  AGE: INTEGER
END;

...

ALSO, STRUCTURES CANNOT BE RECURSIVELY DEFINED.
"LEGAL"

VAR. LINE: RECORD
    FIELD1: INTEGER;
    FIELD2: 1..10
END;

"LEGAL"

TYPE F2 = 1..10;
VAR LINE : RECORD
    FIELD1: INTEGER;
    FIELD2: F2
END;

ENUMERATION MAY BE DIRECTLY DEFINED WITHIN A RECORD
DEFINITION
PASCAL

SUPPOSE WE WANT AN ARRAY OF RECORDS

TYPE STRING8 = ARRAY (1..8) OF CHAR;
TYPE SEXTYP = (MALE, FEMALE);
TYPE PERSON = RECORD
  "SOMETHING HERE"
END;

VAR TABLE: ARRAY (.1..100.) OF PERSON;

WHERE

FOR MEN, WE WANT:

TYPE WINDEX = 1..100;
TYPE PERSON = RECORD
  NAME: STRING8;
  SEX: SEXTYP;
  AGE: INTEGER;
  WIFE: WINDEX
END;

FOR WOMEN WE WANT:

TYPE PERSON = RECORD
  NAME: STRING8;
  SEX: SEXTYP;
  AGE: INTEGER;
  NUMBER: INTEGER;
  CHILDREN: ARRAY(.1..3.)
    OF STRING8
END;

THIS KIND OF PROGRAMMING IS VERY USEFUL; IT OCCURS COMMONLY IN
PASCAL PROGRAMS
THAT IS:

- THE RECORD DOES NOT ALWAYS HAVE THE SAME FORM
  (i.e., DIFFERENT INSTANCES OF THE RECORD TYPE DO NOT ALL
   HAVE THE SAME FORM)

- THE VARYING PART IS THE LAST COMPONENT OF THE RECORD

- THE TYPE OF THE VARIABLE COMPONENT IS ONE OF AT MOST 16
  DIFFERENT SPECIFIED ALTERNATIVES

- FOR ANY INSTANCE OF THE RECORD,
  THE ACTUAL TYPE OF THE VARIABLE COMPONENT
  DEPENDS UPON THE VALUE OF A PRIOR COMPONENT OF
  THE RECORD WHICH IS AN ENUMERATION TYPE OF AT
  MOST 16 DIFFERENT VALUES
  (EQ. 0..15 OR ID0, ..., ID15)

- WE CAN AFFORD TO ALLOCATE SPACE WITH EVERY INSTANCE OF THE
  RECORD FOR THE LARGEST ALTERNATIVE COMPONENT.
  (HENCE THE RECORD HAS A FIXED ALLOCATION SIZE EVEN
   THOUGH ONE COMPONENT IS VARIABLE.)
PASCAL

TYPE STRING8 = ARRAY (1..8) OF CHAR;
TYPE SEXTYP = (MALE, FEMALE);
TYPE WINDEX = 1..10;
TYPE PERSON = RECORD
  NAME: STRING8;
  AGE: INTEGER;
  CASE SEX: SEXTYP OF
    MALE: (WIFE: WINDEX);
    FEMALE: (NUMBER: INTEGER;
             CHILDREN: ARRAY (1..3) OF STRING8)
  END;

VAR TABLE: ARRAY (1..100) OF PERSON;
  I: 1..10;
  SEXUAL: SEXTYP;

BEGIN
  I := 8,
  SEXUAL := MALE;
  WITH TABLE(I) DO BEGIN
    NAME := 'JONES_ _ _';
    AGE := 38;
    SEX := SEXUAL;
    CASE SEXUAL OF
      MALE: WIFE := 8;
      FEMALE: BEGIN NUMBER := 1;
               CHILDREN(1) := 'HAPPY_ _ _'
               END "BEGIN"
    END "CASE"
  END "WITH";

END.
NOTES ABOUT VARIANT RECORDS

> USE OF END TO CLOSE BOTH THE VARIANT PART OF THE RECORD AND THE RECORD ITSELF

> USE OF "(" ")" IN THE VARIANT SYNTAX - DIFFERENT THAN CASE_STATEMENT SYNTAX

> USE OF THE TAG VARIABLE (SEX) AND THE TAG VARIABLE DECLARATION (SEX:SEX_TYP) AFTER CASE - DIFFERENT THAN CASE_STATEMENT SYNTAX

> THE TAG VARIABLE MUST BE AN ENUMERATION WITH 2 TO 16 VALUES (Eg. 0..15)

> IT IS POSSIBLE THAT FOR SOME OF THE TAG VALUES, THE VARIANT COMPONENT IS NULL (IE. NO COMPONENT)
SET 6  PROGRAM & PROCEDURE STRUCTURE

GENERAL FEATURES (6.2)

EXAMPLES - LOCAL/GLOBAL VARIABLES (6.3)
  - DYNAMIC ALLOCATION (6.4)

CONSTRAINTS (6.5)

EXAMPLES - NESTED PROCEDURES (ILLEGAL) (6.6)
  - FORWARD REFERENCE (RESTRICTED) (6.7)
  - RECURSIVE PROCEDURE (6.8)
  - INDIRECT RECUSION (WITH FORWARD) (6.9)
<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>- SUBPROGRAMS ARE CALLED SUBROUTINES (OR FUNCTIONS)</td>
<td>- SUBPROGRAMS ARE CALLED PROCEDURES (OR FUNCTIONS)</td>
</tr>
<tr>
<td>- SUBPROGRAMS ARE SEPARATE BLOCKS, NOT WITHIN THE MAIN PROGRAM</td>
<td>- EACH SUBPROGRAM MUST BE A DECLARATIONS STATEMENT BEFORE THE BODY OF THE MAIN PROGRAM (EACH WHOLE PROCEDURE IS ONE DECLARATION)</td>
</tr>
<tr>
<td>- MAIN AND SUBS CAN BE COMPILED AS A GROUP OR COMPILED SEPARATELY AND THEN &quot;LINKED&quot;.</td>
<td>- MAIN AND SUBS ARE COMPILED TOGETHER (NO &quot;LINKING&quot; PROCESS)</td>
</tr>
<tr>
<td></td>
<td>HOWEVER= PROGRAMS CAN CALL OTHER PROGRAMS AND EVEN PASS SOME PARAMETERS.</td>
</tr>
<tr>
<td>- MUST USE AN EXPLICIT &quot;CALL&quot; FOR SUBROUTINES</td>
<td>- SUBPROGRAMS ARE INVOICED BY NAME.</td>
</tr>
<tr>
<td>- EXPLICIT &quot;RETURN&quot;</td>
<td>- RETURN IS IMPLICIT</td>
</tr>
<tr>
<td>- ALL SUBPROGRAM VARIABLES ARE LOCAL UNLESS DECLARED &quot;COMMON&quot;</td>
<td>- ALL SUBPROGRAM VARIABLES (WHICH MUST BE DECLARED) ARE LOCAL. ANY VARIABLE NOT DECLARED IN THE SUBPROGRAM IS GLOBAL TO THE MAIN PROGRAM. (EXAMPLE 1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>- ALL VARIABLES ARE ALLOCATED STATICALLY</td>
<td>- SUBPROGRAM VARIABLES ARE ALLOCATED DYNAMICALLY (SEE EXAMPLE 2)</td>
</tr>
</tbody>
</table>
EXAMPLE 1

C MAIN
   COMMON NCAT
   INTEGER NCAT
   :
   NCAT = 1000
   :
   CALL SUB1

C NCAT STILL 1000
   CALL SUB2
C NOW NCAT IS 2
   END

SUBROUTINE SUB2
   COMMON NCAT
   INTEGER NCAT
   NCAT = 2
   RETURN
   END

SUBROUTINE SUB1
   INTEGER NCAT
   NCAT = 2
   RETURN
   END

PROGRAM "MAIN"
   VAR NCAT: INTEGER;

PROCEDURE SUB2
   BEGIN
   NCAT := 2
   END "SUB2";

PROCEDURE SUB1
   VAR NCAT: INTEGER;
   BEGIN
   NCAT := 2
   END "SUB1";

   NCAT := 1000;
   SUB1; "NCAT STILL 1000"
   SUB2; "NOW NCAT IS 2"
   END.

NCAT IS GLOBAL FOR MAIN AND SUB1
NCAT IS LOCAL IN SUB2

6.3
EXAMPLE 2

C

MAIN
COMMON/COM/NCAT, CAT(1000)
NCAT = 1000
.
.
CALL SUB1
.
.
CALL SUB2
.
.
END

SUBROUTINE SUB1
COMMON/COM/NCAT,CAT(1000)
DIMENSION CAT1(1000)
.
.
CAT(I) = CAT1(I)
.
RETURN
END

SUBROUTINE SUB2
COMMON/COM/NCAT,CAT(1000)
.
.
CAT(I) = CAT2(I)
.
RETURN
END

PROGRAM "MAIN"
CONST NCAT = 1000
TYPE ATYPE=ARRAY(1..NCAT).
OF REAL;
VAR CAT:ATYPE; I:INTEGER;

PROCEDURE SUB1;
VAR CAT1: ATYPE; I:1..NCAT;
BEGIN
.
.
CAT(I):=CAT1(I);
.
END "SUB1";

PROCEDURE SUB2;
VAR CAT2:ATYPE;
I:1..NCAT;
BEGIN
.
.
CAT(I):=CAT2(I);
.
END "SUB2";

BEGIN

SUB1;
.
.
SUB2;
.
.
END.

STATIC ALLOCATION:
APPROX. 3003 CELLS
ALLOCATED FOR DATA

DYNAMIC ALLOCATION:
APPROX. 2002 CELLS ALLOCATED
FOR DATA. (CAT1 AND CAT2 BOTH
USE SAME CELLS ON RUN-TIME STACK,
BUT AT DIFFERENT TIMES)
### Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subroutines cannot be nested</strong></td>
<td>Subprogram declarations cannot be nested (this implementation) (see example 3)</td>
</tr>
<tr>
<td><strong>No ordering of subprograms is required</strong></td>
<td>Subprograms must be ordered so that the declaration precedes the reference to the subprogram. (example 4)</td>
</tr>
<tr>
<td><strong>No recursion allowed</strong></td>
<td>Recursive calls are allowed (example 5); indirect recursion requires the forward declaration so as to avoid the reference-before-declaration constraint (example 6)</td>
</tr>
</tbody>
</table>
EXAMPLE 3

PROGRAM MAIN;
VAR A,B: INTEGER;
PROCEDURE SUB1;
VAR C,D: INTEGER;

PROCEDURE SUB2;
BEGIN

ILLEGAL

BEGIN

SUB2;

END "SUB1";

BEGIN

END.

NESTED PROCEDURES ARE ILLEGAL
EXAMPLE 4

ILLEGAL PASCAL

PROGRAM MAIN;

;

PROCEDURE A;

;

BEGIN

;

B;

;

END "A";

PROCEDURE B;

;

;

;

;

;

;

;

;

;

LEGAL PASCAL

PROGRAM MAIN;

;

PROCEDURE B;

;

BEGIN

;

END "B";

PROCEDURE A;

;

BEGIN

;

B;

;

END "A";

BEGIN

;

A;

;

END.

REFERENCE TO A PROCEDURE BEFORE ITS DECLARATION IS ILLEGAL
EXAMPLE 5

PROGRAM MAIN
VAR I: INTEGER;
PROCEDURE FACTORIAL;

BEGIN
...  
I:=I-1;
IF I>=2 THEN FACTORIAL;
...

END "FACTORIAL";
BEGIN
I:=5;
FACTORIAL;
...

END.

RECURSIVE PROCEDURE
(CALLS ITSELF)
EXAMPLE 6 INDIRECT RECURSION

PROGRAM MAIN;
VAR I: INTEGER;
PROCEDURE B(PARAMETER LIST); FORWARD
PROCEDURE A (PARAMETER LIST);
   VAR J: INTEGER;
   BEGIN
      ..
      ..
      IF I > J THEN B(I-1);
      ..
      END "A";
PROCEDURE B;
   VAR K: INTEGER;
   BEGIN
      ..
      ..
      IF I > K THEN A(I-1);
      ..
      END "B";
BEGIN
   ..
   ..
   I:=5;
   A(I);
   ..
   ..
END.

NOTES:
- PARAMETERS ARE LISTED WITH THE "FORWARD" DECLARATION
- PARAMETERS ARE NOT LISTED WITH THE SECOND DECLARATION
- COMPILER CANNOT DETERMINE (STATICALLY) IF THIS WILL CAUSE RUN-TIME STACK OVERFLOW. (AVOID THIS TYPE OF PROGRAMMING IF POSSIBLE!)

NOTE:
PROCEDURE CALL! NOT A(I,I.)
PASSING PARAMETERS

GENERAL (7.2)
PARAMETER LINKAGE (7.3)
VARIABLE PARAMETERS (7.4)
CONSTANT (VALUE) PARAMETERS (7.5)
PARAMETER SYNTAX (7.6)
PARAMETER SPECIFICATIONS (7.7)
TYPE CHECKING EXAMPLES (7.8)
UNIVERSAL LINKAGE (7.13)
FUNCTIONS VS PROCEDURES (7.15)
CHECK ON MOVING (7.16)

7.1
PASSING PARAMETERS TO SUBPROGRAMS/PROCEDURES
(ALSO CALLED ARGUMENTS)

FORMAL PARAMETERS = THOSE DECLARED IN THE SUBPROGRAM BODY

ACTUAL PARAMETERS = THOSE PASSED TO (OR FROM) THE SUBPROGRAM
BY THE CALLING PROGRAM

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST AGREE IN NUMBER</td>
<td>MUST AGREE IN NUMBER</td>
</tr>
<tr>
<td>ARGUMENTS AND FORMAL PARAMETERS MUST BE IN SAME ORDER.</td>
<td>ARGUMENTS AND FORMAL PARAMETERS MUST BE IN SAME ORDER.</td>
</tr>
<tr>
<td>NOT CHECKED BY THE COMPILER!</td>
<td>MUST AGREE IN TYPE!!!!</td>
</tr>
<tr>
<td>NOT CHECKED AT RUN-TIME!</td>
<td>(THE COMPILER CHECK THIS!)</td>
</tr>
<tr>
<td>NOT CHECKED BY THE COMPILER!</td>
<td>STRUCTURES PASSED AS PARAMETERS MUST ALSO AGREE IN SIZE!!</td>
</tr>
<tr>
<td></td>
<td>(PART OF TYPE CHECKING)</td>
</tr>
<tr>
<td>CAN BE DECLARED OR JUST USE TYPE DEFAULTS</td>
<td>PARAMETERS MUST BE DECLARED</td>
</tr>
<tr>
<td></td>
<td>(CALL THIE &quot;SPECIFIED&quot;)</td>
</tr>
<tr>
<td>SPECIFICATIONS MIXED IN WITH DECLARATIONS OF LOCAL</td>
<td>SPECIFICATIONS INCLUDED IN THE PROCEDURE LINE</td>
</tr>
<tr>
<td>VARIABLES</td>
<td></td>
</tr>
</tbody>
</table>

7.2
KINDS OF PARAMETER LINKAGE

FORTRAN

BY LOCATION:
AN ADDRESS IS CALCULATED FOR EACH ACTUAL PARAMETER AND THESE ADDRESSES ARE LINKED INTO THE SUBPROGRAM (EXPRESSIONS ARE EVALUATED AND A LOCATION IS ASSIGNED TO STORE EACH EXPRESSION VALUE)

PASCAL

OPTION:
EITHER: BY "ACCESS VALUE".

NOTE:
-THUS IT IS IMPOSSIBLE FOR THE PROCEDURE TO CHANGE THE VALUE OF THE ACTUAL PARAMETER.
-THOSE ARE CALLED VALUE PARAMETERS

OR: BY REFERENCE (USING VAR):
THE ACTUAL PARAMETER MUST BE EITHER:
-A VARIABLE
-AN ARRAY COMPONENT
-A RECORD COMPONENT

FOR EACH PARAMETER, THE VARIABLE OR COMPONENT ADDRESS IS CALCULATE AND LINKED INTO THE PROCEDURE AT THE TIME OF THE CALL.

NOTE: THESE ARE CALLED VARIABLE PARAMETERS

7.3
KINDS OF PARAMETER LINKAGES (CONT.)

BY FUNCTION-VALUE

THE VALUE OF A FUNCTION
SUBPROGRAM IS RETURNED
AS THE RESULT OF A
FUNCTION REFERENCE.

BY FUNCTION-VALUE

THE VALUE OF A FUNCTION
SUBPROGRAM IS RETURNED
AS THE RESULT OF A
FUNCTION REFERENCE.

7.3.1
PASCAL

VARIABLE PARAMETERS

PROGRAM MAIN
VAR I,J: INTEGER;

PROCEDURE SUB1 (VAR K: INTEGER);

BEGIN "SUB1"

END; "SUB1"

BEGIN "MAIN"

SUB1(I);

SUB1(J);

END. "MAIN"

RESULTS OF PROCEDURES MUST BE VARIABLE PARAMETERS.

VARIABLE PARAMETERS IN THE PROCEDURE HEADING ARE NOT
ALLOCATED STORAGE SPACE, THEY USE THE STORAGE LOCATION ALLOCATED
FOR THE CALLING PARAMETER. (EX. K POINTS TO THE STORAGE LOCATION
OF I OR J WHEN SUB2 IS CALLED ABOVE).
PASCAL
CONSTANT (VALUE) PARAMETERS

PROGRAM MAIN;
VAR I, J: INTEGER;
PROCEDURE SUB1 (K: INTEGER);
VAR L: INTEGER;
BEGIN "SUB1"
  ...
  "ILLEGAL AS K IS CONSTANT"
  K := K + 2;
  L := K * 4;
  ...
END "SUB1";
BEGIN "MAIN"
  ...
  I := 2;
  J := I * 8;
  ...
  SUB1 (I);
  ...
  SUB1 (J);
  ...
END "MAIN"

1. CONSTANT PARAMETERS ARE ASSIGNED STORAGE SPACE.
2. VALUE OF CALLING PARAMETER IS NEVER CHANGED BY PROCEDURE.
PASCAL
SYNTAX VARIATIONS

PROCEDURE JIM (I: INTEGER;
               J: INTEGER;
               W: REAL;
               VAR X: REAL;
               VAR Y: REAL;
               VAR Z: INTEGER);

SAME AS: PROCEDURE JIM (I,J: INTEGER; W: REAL;
                         VAR X, Y: REAL;
                         VAR Z: INTEGER);

(OR IT COULD ALL BE WRITTEN ON ONE LINE)

NOTE: IT IS A GOOD HABIT TO GROUP ALL THE CONSTANT
PARAMETERS BEFORE THE VARIABLE PARAMETERS
PARAMETER SPECIFICATIONS

MUST BE TYPE IDENTIFIER

ILLEGAL

PROGRAM MAIN;
VAR X: ARRAY (.1..10.) OF INTEGER;
PROCEDURE A (Y: ARRAY (.1..10.) OF INTEGER);
   :
   :
   :

LEGAL

PROGRAM MAIN;
TYPE ARR = ARRAY (.1..10.) OF INTEGER;
VAR X: ARR;
PROCEDURE A(Y:ARR);
   :
   :

7.7
TYPE CHECKING

FORTRAN

C MAIN PROGRAM

INTEGER I, J

I = 10

CALL A(I, J)

C WOW, THIS IS LEGAL

C ACTUALLY USEFULL SOMETIMES

END

SUBROUTINE A(X, Y)

INTEGER X

REAL Y

IF (X.LE.20) Y = 5.6

RETURN

END

PASCAL

PROGRAM MAIN;

VAR I, J: INTEGER;

PROCEDURE A(X: INTEGER;

VAR Y: REAL);

BEGIN

IF X<=20 THEN Y := 5.6;

END "A";

BEGIN "MAIN"

I := 10;

A(I, J);

"THIS IS ILLEGAL AND WON'T COMPIL AS J AND Y ARE DIFFERENT TYPES"

RETURN

END.

7.8
TYPE CHECKING

FORTRAN          PASCAL

C    MAIN
REAL Z,Y
ONE= 1.0
Z = 1.0
Y = 1.0
CALL A (ONE)
CALL A(Y+Z)

C    WOW
C    ALL LEGAL
END

SUBROUTINE A(X)
X= X + 1.0
RETURN
END

PROGRAM MAIN;
CONST ONE = 1.0;
VAR Z,Y : REAL;
PROCEDURE A (VAR X: REAL);
BEGIN X:= X+1.0
END "A";
BEGIN
A(ONE);
Z:=1.0;
Y:=1.0;
A(Z+Y)
"WONT COMPILE"
END.

NOTE: THESE WOULD BE LEGAL IF THE PARAMETER WERE DECLARED AS A CONSTANT PARAMETER (IE NO VAR IN THE DECLARATION)
TYPE CHECKING

C    MAIN
PROGRAM "MAIN"
DIMENSION Y(10)
VAR Y: ARRAY (.1..10.) OF REAL;
PROCEDURE A(VAR X: B);
BEGIN
  x(4):=1.0
END "A";
BEGIN
SUBROUTINE A(X)
DIMENSION X(4)
A(X(4.))
"WONT COMPILE - TYPE ERROR:
REAL \neq ARRAY OF REAL"
RETURN
END

7.10
TYPE CHECKING

PROGRAM "MAIN"

TYPE B = ARRAY (.1..20.) OF REAL;
VAR Y: ARRAY (.1..10.) OF REAL;
PROCEDURE A(X: B);

BEGIN

;

END "A";

BEGIN

;

;

A (Y);
"WONT COMPILE"

;

END.

SIZE ERROR:

ARRAY (.1..10.) ≠ ARRAY (.1..20.)
AN EXCEPTION!!!

The size check is relaxed for strings:

The actual string may be less than or equal to the formal string.

PROGRAM MAIN;
TYPE STRING8 = ARRAY (.1..8.) OF CHAR
VAR X: ARRAY (.1..10.) OF CHAR;
   Y: ARRAY (.1..6.) OF CHAR;
PROCEDURE A(Z: STRING8);
   VAR I: INTEGER;
   BEGIN
   : END "A";
BEGIN
   : A(Y); "THIS IS LEGAL!"
   A(X); "THIS IS ILLEGAL"
   : END.

WARNING: Z(.7,) IS SOME VALUE BEYOND Y
WARNING: IF Z WERE A VAR PARAMETER, THEN ASSIGNMENT TO Z(.7,) OR Z(.8,) WOULD OVERWRITE SOMETHING IN MEMORY BEYOND Y!!!awanful!!!!
**UNIVERSAL LINKAGE**

Sometimes it is necessary to cheat on parameter type checking!

(Usually for I/O procedures)

The additional linking mechanism Universal (UNIV) removes the type check part of the linkage, but still enforces the size check.

(Does not work for pointer variables, which are introduced later)

Hence you need to know storage requirements of different types:

<table>
<thead>
<tr>
<th>DATA TYPES</th>
<th>#WORDS OF STORAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>LOWER..UPPER IDENTIFIERS</td>
<td>1</td>
</tr>
<tr>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>4</td>
</tr>
<tr>
<td>SET</td>
<td>8</td>
</tr>
<tr>
<td>STRING (M CHAR)</td>
<td>M/2</td>
</tr>
</tbody>
</table>

7.13
LEGAL

PROGRAM MAIN;
TYPE STRINGX = ARRAY (.1..16.) OF CHAR;
VAR A: ARRAY (.1..2.) OF REAL;
PROCEDURE STORE (X: UNIV STRINGX);
BEGIN

"USEFUL BECAUSE I/O PROCEDURES HANDLE ONLY CHARACTERS OR ARRAYS OF CHARACTERS"

END "STORE";

BEGIN

STORE (A);

END.

COMMON UNIVERSAL TYPES ARE "LINE" AND "PAGE" WHICH ARE DEFINED IN THE STANDARD PREFIX.
PASCAL

FUNCTION PARAMETERS MUST BE: CONSTANT

PROCEDURE PARAMETERS MAY BE: CONSTANT OR VARIABLE
PROCEDURE AND FUNCTION CALLS

FORTRAN

SUBROUTINES ARE CALLED BY A STATEMENT (CALL SUB1)

FUNCTIONS ARE ALWAYS CALLED AS A FACTOR IN AN EXPRESSION

PASCAL

A ROUTINE CALLED AS A STATEMENT MUST BE A PROCEDURE CALL.

A ROUTINE CALLED AS A FACTOR IN AN EXPRESSION MUST BE A FUNCTION CALL.

C

MAIN PROGRAM

.
.
CALL SUB1
CALL SUB2(A,B)

SUBROUTINE CALLS

.
.
F= FUNC1(C)
IF (FUNC1(B).LT.1000)I=1

FUNCTION CALLS

.
.
END

PROGRAM MAIN;

.
.
BEGIN "MAIN"

.
.
SUB1
SUB2(A,B);
"PROCEDURE CALLS"
.
.
F:= ’FUNC1(C);
IF FUNC1(B) > 1000 THEN I=1;
"FUNCTION CALLS"
.
.
END. "MAIN"

7.16
FORTRAN FUNCTIONS

1. A value is assigned and returned as the function name.
2. Value returned must be assigned to function identifier within the function.
3. The function and its assigned value may be of different type (ex. REAL = INTEGER).
4. Parameters may return a value to the calling program!!

C
MULTIPLICATION OF POSITIVE INTEGERS
C
MAIN
INTEGER I,J,K
C
BEGIN MAIN

I= 2
J= 3
K= MULINT (I,J)

END

FUNCTION MULINT (L,M)
INTEGER L,M,N
N=0
100 IF(L.LE.0) GO TO 200
 N= N + M
 L= L - 1
 GO TO 100
200 MULINT= N
RETURN
END

NOTE: I.EQ.0 AFTER FUNCTION COMPLETES

7.17
PASCAL FUNCTIONS

1. A SINGLE VALUE IS RETURNED UNDER FUNCTION NAME.
2. THE VALUE RETURNED MUST BE ASSIGNED TO THE FUNCTION IDENTIFIER WITHIN THE FUNCTION BLOCK.
3. THE FUNCTION AND ITS ASSIGNED VALUE MUST BE OF COMPATIBLE TYPE
4. PARAMETERS CANNOT RETURN A VALUE (PASSED AS CONSTANTS)

"MULTIPLICATION OF POSITIVE INTEGERS"

PROGRAM MAIN;
VAR I, J, K: INTEGER;

FUNCTION MULINT (L, M:INTEGER): INTEGER;
  "L > 0, M > 0"
  VAR L1, N: INTEGER;
  BEGIN
    L1 := L;
    N := 0;
    WHILE L1 > 0 DO
      BEGIN N := N + M;
        L1 := L1 - 1 END;
    MULINT := N
  END "FUNCTION MULINT";

BEGIN "MAIN"
  I := 2;
  J := 3;
  K := MULINT(I, J);
END. "MAIN" "NOTE I, J ARE NOT CHANGED AFTER FUNCTION COMPLETES"
RESTRICTION

KSU (CIT) IMPLEMENTATION WILL NOT ALLOW

FUNCTION NAMES OR PROCEDURE NAMES TO BE PASSED AS AN ACTUAL PARAMETER TO A SUBPROGRAM

NOTE: THIS CAN BE DONE IN THE WINSEN/WIRTH PASCAL REPORT
Group 8: Simple Input/Output

I/O Primitives

Line Oriented Routines

Four Conversion Routines

8.1
PRIMITIVES PROVIDED IN THE STANDARD PREFIX

FOR THE PASCAL CONSOLE

IDENTIFY (HEADER: LINE) ............INITIALIZATION TO IDENTIFY
   THE CALLING PROGRAM; LINE IS
   DEFINED IN THE STANDARD PREFIX
   AS A TYPE OF ARRAY OF 132
   CHARACTERS

ACCEPT (VAR C: CHAR).............READ AND PRINT, RESPECTIVELY,
DISPLAY ( C: CHAR).................A SINGLE CHARACTER TO THE
                                   PASCAL CONSOLE; THE ASCII LINE
                                   FEED CHARACTER (i.e. CHR (10))
                                   IS USED TO TERMINATE EACH LINE
                                   OF INPUT AND OUTPUT

FOR CHARACTER ORIENTED DEVICES (SEE ALSO GROUP 9)

WRITEARG ..................................USED TO IDENTIFY THE FILE OR
READARG ..................................USED TO CHECK THE SUCCESS OR
                                   FAILURE OF THE FILE OR DEVICE
                                   ACCESS

READ (VAR C: CHAR) .......... INPUT AND OUTPUT, RESPECTIVELY,
WRITE ( C: CHAR) ............. OF A SINGLE CHARACTER FROM/TO
                                   THE FILE OR DEVICE DRIVER
                                   SPECIFIED USING WRITEARG.

8.2
Extended I/O Routines (not in Standard Prefix)

RDLINE (NAME; IDENTIFIER; VAR INLINE: LINE; VAR R: BOOLEAN)
WRITLE (NAME: IDENTIFIER; OUTLINE: LINE; VAR R: BOOLEAN)

Used to read/write a line of characters
from/to a file or device driver; where

NAME----------a string of length 12 which
is the name of a file or an
I/O driver, such as listed
on page 10.4

INLINE, OUTLINE----a string of length LINELENGTH
(132); terminated with the
character EM (defined as CHR
(25) in the prefix).

R----------result: false indicts an I/O error.

CONRDLN (VAR INLINE: LINE)------similar to RDLINE and WRTLN,
CONWRTLN (OUTLINE: LINE)
except these read/write from/to
the PASCAL console, and the
line is terminated by a NL
character.
CONVERSION Routines (not in standard prefix)

INT_TO_STR (INT, WIDTH: INTEGER;
VAR INDEX: INTEGER;
VAR R: CONVTERR;
VAR OUTLINE: LINE)

--- INTEGER—to STRING CONVERSION, WHERE

INT ........ INTEGER TO BE CONVERTED

WIDTH ........ WIDTH OF "WINDOW" IN OUTLINE

INDEX ........ STARTING POSITION OF "WINDOW" IN OUTLINE; SET TO STARTING POSITION
OF NEXT POSSIBLE WINDOW

R ............ RETURN PARAMETER:

CONVTERR...... Enumeration TYPE OF:

OK............. CONVERSION DONE
INDEXERR..... ILLEGAL WINDOW POSITION
WIDTHERR..... ILLEGAL WINDOW SIZE
LPRSN
RPRSN
NOTINT
OVERFLOW
NOTFIXED
TRANS..... I/O DEVICE ERROR
FIXED_TO_STR (NUMB: REAL;
    WIDTH, LPRECISION, RPRECISION: INTEGER;
    VAR INDEX: INTEGER
    VAR R: CONVTERM;
    VAR OUTLINE; LINE)

---REAL NUMBER TO STRING CONVERSION, USING FIXED POINT NOTATION; WHERE

    NUMB........REAL NUMBER TO BE CONVERTED
    LPRECISION....NUMBER OF SPACES ALLOWED TO LEFT
                  OF DECIMAL POINT, INCLUDING THE SIGN
    RPRECISION....NUMBER OF SPACES ALLOWED TO RIGHT
                  OF DECIMAL POINT

    NOTE:  INDEX ≥ LPRECISION + RPRECISION + 1

    CONVTERM......ADDITION VALUES ARE:
    LPRSN......ILLEGAL LPRECISION
    RPRSN......ILLEGAL RPRECISION
STR_TO_INT (VAR INT: INTEGER;
    WIDTH: INTEGER;
    VAR INDEX: INTEGER;
    VAR R: CONVTERM;
    VAR INLINE; LINE)

---STRING TO INTEGER CONVERSION; WHERE

INT...........INTEGER VALUE DETERMINED

WIDTH.......WIDTH OF WINDOW TO BE PROCESSED;
    HOWEVER, IF THE WINDOW IS SET TO ZERO,
    THEN THE WINDOW IS EXPANDED UNTIL AN
    INTEGER IS CONVERTED OR UNTIL AN ERROR
    IS DETECTED.
CONVTERM......ADDITIONAL VALUES ARE:
    NOTINT.....SCANNING SYNTAX ERROR
    OVERFLOW...INTEGER TOO LARGE TO CONVERT
STR_TO_FIXED (VAR NUMB; REAL;
    WIDTH: INTEGER;
    VAR INDEX: INTEGER,
    VAR R: CONVTERR;
    VAR INLINE: LINE)

---STRING TO REAL NUMBER CONVERSION, ASSUMING A FIXED POINT
NOTATION, WHERE

    NUMBR........REAL VALUE TO BE DETERMINED
    CONVTERR.....ADDITION VALUES ARE:
    NOTFIXED.....SCANNING SYNTAX ERROR

8.7
GROUP 9: STANDARD PREFIX
PROGRAM FORMAT (9.2)
PURPOSE OF PREFIX (9.3)
PREFIX CONSTANTS (9.4)
PREFIX PROCEDURES (9.5)
FILE PROCEDURES (9.6)
ANNOTATED PREFIX (9.7)
"HEADER COMMENTS"

(OPTION, OPTION . . .)

← SKIP IF NO OPTIONS SELECTED

STANDARD_PREFIX_
DECLARATIONS
(WITHOUT COMMENTS)
(About 120 LINES)
INCLUDES PROGRAM STATEMENT
PROGRAM P(VAR PARAM:ARGLIST);

YOUR_DECLARATIONS_HERE

BEGIN

"BODY OF PROGRAM HERE"

END. DATA_STARTS_HERE
(IF ANY)

#

NOTE: NAME OF PROGRAM IS "P"
(WHICH CAN BE CHANGED, USING
THE EDITOR); HOWEVER, THE
PROGRAM IS INVOKED BY FILE
NAME, WHICH MAY NOT BE "P".
PURPOSES FOR PREFIX

. DEFINES COMMONLY USED CONSTANTS

. DEFINES COMMONLY USED TYPES

. LISTS NAMES OF PROCEDURES WHICH ARE ENTRY POINTS TO SOLO SYSTEM

. IDENTIFIES PARAM, WHICH IS USED TO PASS PARAMETERS TO AND FROM THE PROGRAM
PREFIX CONSTANTS

NL = "NEW_LINE" OR "LINE_FEED"

FF = "FORM_FEED"

CR = "CARRIAGE RETURN"

EM = "END_OF_MEDIUM_MARK" FOR STREAM I/O.

PAGELENGTH = 512 BYTES

LINELENGTH = 132 CHARS

IDLENGH = 12 = MAX LENGTH FOR IDENTIFIERS USED AS PARAMETERS

MAXARG = 10 = MAX NUMBER OF ARGUMENT TO/FROM THE PROGRAM
PREFIX PROCEDURES

HEAP PROCEDURES
MARK, RELEASE

I/O PROCEDURES
READ, WRITE __ CHARACTER I/O, READER(PRINTER)
IDENTIFY, ACCEPT, DISPLAY __ CHARACTER I/O, PASCAL CONSOLE
I/O TRANSFER __ PAGE I/O TO DEVICE
I/O MOVE __ DEVICE CONTROL

RELATED TYPES

ARRAYS OF CHAR: PAGE, LINE, IDENTIFIER
I/O DEVICES: TYPEDEVICE, DISKDEVICE, TAPEDEVICE,
            PRINTDEVICE, CARDDEVICE.
I/O OPERATIONS: INPUT, OUTPUT, MOVE, CONTROL
I/O RESULTS: COMPLETE, INTERVENTION, TRANSMISSION,
             FAILURE, ENDFILE, ENDMEDIUM, STARTMEDIUM
OTHER_CONSTANTS: WRITEEOF, REWIND, VPSPACE,
                  BACKSPACE

9.5
FILE PROCEDURES

OPEN, CLOSE

GET, PUT

LOOKUP, LENGTH

RELATED TYPES

FILEKINDS: EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE
OTHER_CONSTANTS: FILE, FILE_ATTRIB

INTRA_PROGRAM_COMMUNICATION

READARG, WRITEARG ____ EXCHANGE PARAMETERS WITH OTHER PROGRAM

TASK ____ CHECK TASK NAME

RUN ____ RUN A PROGRAM

RELATED TYPES

TASKKINDS: INPUTTASK, JOBTASK, OUTPUTTASK
ARGTAGS: NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE
PROGRESULTS: TERMINATED, OVERFLOW, POINTERERROR,
            RANGEERROR, VARIANTERROR, HEAPLIMIT, STACKLIMIT,
            CODELIMIT, TIMELIMIT, CALLERROR
"SUPER BRINCH HANSEN"

INFORMATION SCIENCE
CALIFORNIA INSTITUTE OF TECHNOLOGY

UTILITY PROGRAMS FOR
THE SOLO SYSTEM

18 MAY 1975"

*************
* ANNOTATED PRE* *
** PREFIX **
*************

(CHECK, NUMBER, TEST, XREF)
"4 A PROGRAM MAY BE PRECEDED BY COMPILER OPTIONS
ENCLOSED IN PARENTHESES. ONLY THE FIRST CHARACTER
OF AN OPTION IS EXAMINED. THE OPTIONS HAVE THE FOLLOWING
EFFECT:

CHECK - THE GENERATED CODE WILL NOT MAKE THE
FOLLOWING CHECKS:
A) CONSTANT ENUMERATION RANGE CHECKS
(CONCURRENT AND SEQUENTIAL);
B) USE OF NIL VALID POINTERS (SEQUENTIAL
ONLY);
C) ILLEGAL VARIANT FIELD CHECKS (SEQUENTIAL
ONLY).

NUMBER - THE GENERATED CODE WILL CONTAIN LINE
NUMBERS OF THE PROGRAM'S TEXT FOR THE
BEGINNING OF ROUTINES ONLY (NORMAL
PRODUCTION MODE - SHOULD NOT BE USED
WHILE DEBUGGING A PROGRAM.)

TEST - THE COMPILER WILL PRINT THE INTERMEDIATE
CODE FOR ALL Passes.

XREF - THE COMPILER WILL PRODUCE A CROSS REFERENCE
TABLE. 5"

CONST NL = '(:10:1)'; FF = '(:12:1)'; CR = '(:13:1)'; EN = '(:25:1)';
"4 THE CONSTANT NL (LINE FEED) IS THE LOGICAL
END OF LINE CHARACTER IN PASCAL. IT IS TRANSLATED BY THE
KERNEL INTO A CARRIAGE RETURN (CR) FOR COMPATIBILITY WITH
32/32-LT DRIVERS. THE END OF RECORD CHARACTER (EN) IS USED
FOR TERMINATING A STREAM OF CHARACTER ORIENTED TRANSFERS
(SEE READ AND WRITE.) 5"

CONST PAGELENGTH = 512;
TYPE PAGE = ARRAY [.1..PAGELENGTH] OF CHAR;
"4 A PAGE IS THE STANDARD UNIT OF DATA IN THE SOLO SYSTEM.
DATA IS STORED ON BOTH DISK AND TAPE AS PAGES. IT MAY BE
CHANGED TO CHARACTER BY CHARACTER TRANSFERS BETWEEN PROCESSES
BY THE SOLO DATA BUFFER MONITOR. 5"

CONST LINELength = 137;
TYPE LINE = ARRAY [.1..LINELENGTH] OF CHAR;
"4 A LINE IS THE UNIT OF TRANSFER TO THE LINE
PRINTER. IT IS ALSO COMMONLY USED AS THE PARAMETER FOR

9,7
Routines That Display Error Messages to the Console.

CONST IOLENGTH = 12;
TYPE IDENTIFIER = ARRAY (-1..IOLENGTH) OF Char;
  "4. An IDENTIFIER is used for passing parameters and file names.
   All filenames may be a maximum of twelve (12) characters
   under SOLO."

TYPE FILE = 1..2;
  "4. Up to two (2) files may be accessed directly by the program
      running in the job process at any time. These files are
      manipulated by the interface routines GET, PUT, OPEN, CLOSE, and
      LENGTH. Either of the IOProcesses may access only one file at a
      time."

TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SPOOL, CONSOLE);
  "4. These are the standard file types used by SOLO."

TYPE FILEATTR = RECORD
  KNUM: FILEKIND;
  ANDR: INTEGER;
  PROTECTED: BOOLEAN;
  NOTUSED: ARRAY (-1..5) OF INTEGER
END;
  "4. This type describes a disk file. It is returned by the
      solo lookup interface procedure. KNUM is the address of the
      file's pagemap. Sjoer Brinch-Hansen: Disk Scheduling at Compile
      Time. [x 3]."

TYPE IODEVICE =
  (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE, CARDDEVICE);
  "4. This is an enumeration of the actual kernel physical
      devices. Any enumeration with five (5) elements will suffice
      for device specification. It is the position in the enumeration
      which is important."

TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);
  "4. These are the valid operations for physical devices. The
      control operation deserves special attention. The execution of
      a control operation to the disk device causes the system
      to be reinitialized using the arg field of the toparam as the
      disk address at which to find the concurrent code to be loaded
      as the new system. This facility is used by the solo start
      program, for example. The use of the control operation on
      the console is to make the process wait for the next bell key
      to be entered on the console. This function is not very useful under
      SOLO because SOLO includes a special process (the watchdog process)
      which uses this function to reload SOLO."

TYPE IOARG = (WRITEFOR, REWIND, USPACE, BACKSPACE);
  "4. This enumeration describes the functions that
      may be used with the move operation to the tape device.
      They are self explanatory.

TYPE IORESULT =
  (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
   EOPFILE, EOPORIGIN, EOPLOCATION);
  "4. These are the valid statuses returned by the kernel
      in response to I/O requests. The mapping from
      OS/32-RT DEVICE INDEPENDENT STATUS TO THESE STATUSES IS

9.8
AS FOLLOWS:

<table>
<thead>
<tr>
<th>O2/32-MT STATUS</th>
<th>PASCAL STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>COMPLETE</td>
</tr>
<tr>
<td>An</td>
<td>INTERVENTION</td>
</tr>
<tr>
<td>8p</td>
<td>TRANSMISSION</td>
</tr>
<tr>
<td>Ck (ANY BIT)</td>
<td>FAILURE</td>
</tr>
<tr>
<td>8a</td>
<td>ENQUEULE</td>
</tr>
<tr>
<td>90</td>
<td>ENCODED</td>
</tr>
</tbody>
</table>

THE STARTUP/SHUTDOWN STATUS IS NOT IMPLEMENTED ON ANY CURRENTLY SUPPORTED DEVICE. 5"

TYPE IOPARAM = RECORD
  OPERATION: IOPERATION;
  STATUS: IORESULT;
  ARG: IOARG
END;

"4 THIS RECORD IS THE ACTUAL PARAMETER WHICH IS PASSED TO THE KERNEL I/O ROUTINES TO CONTROL I/O REQUESTS. REQUESTS ARE SENT TO THE KERNEL VIA THE SOLO IOTRANSFER INTERFACE PROCEDURE. 5"

TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);

"4 THIS ENUMERATION LISTS THE THREE SOLO PROCESS PARTITIONS INTO WHICH A SEQUENTIAL PROGRAM MAY BE LOADED. A SEQUENTIAL PROGRAM MAY DETERMINE WHICH PARTITION IT IS IN THROUGH THE USE OF THE TASK INTERFACE FUNCTION. 5"

TYPE ARGTYPE = RECORD
  CASE TAG: ARGTYPE OF
    NILTYPE: Boolean;
    BOOLTYPE: (BOOL: BOOLEAN);
    INTTYPE: (INT: INTEGER);
    IDTYPE: (ID: IDENTIFIER);
    PTRTYPE: (PTR: POINTER)
END;

"4 THIS RECORD DESCRIBES THE ARGUMENTS (AND PARAMETERS) WHICH MAY BE PASSED AMONG SEQUENTIAL PROGRAMS RUNNING UNDER SOLO. ARGUMENTS MAY BE PASSED AMONG THE THREE PROCESS PARTITIONS BY MEANS OF THE READARG AND WRITEARG INTERFACE PROCEDURES. A PROGRAM MAY ALSO BE CALLED AND PASSED A LIST OF PARAMETERS (AN ARGLIST) AND BE EXECUTED IN THE SAME PARTITION BY MEANS OF THE SOLO RUN ROUTINE. 5"

CONST MAXARG = 10;

TYPE ARGLIST = ARRAY [1..MAXARG] OF ARGTYPE;

"4 AS NOTED ABOVE A SEQUENTIAL PROGRAM MAY BE CALLED BY ANOTHER BY MEANS OF THE RUN INTERFACE PROCEDURE.
A Maximum of Ten (10) Parameters May Be Passed To The Called Program in An Arglist. 5"

Type Argsflag = (imp, out):

"4 This Enumeration Specifies The Destination Of An Argument Request (readarg And writearg). The
job process may use either enumeration. The input
process may only use the out and the output process
may only use the imp as argument destinations. 5"

Type progresresult =

(terminated, overflow, pointererror, rangeerror, varianteerror,
heaplimit, stacklimit, conelimit, timelimit, calerror);

"4 This enumeration represents the standard
completion codes generated by the kernel upon
program exit or error. terminated represents
the fact that the program controlled its termination.
all other instances represent errors detected by
the kernel or solo. timelimit is not currently
implemented in solo. overflow is detected on real
(floating point) but not integer arithmetic.
also, the kernel maps &/32 illegal instruction
and memory fault errors into overflow. This is
generally caused by executing a program file which
is labeled as seqcode, but which actually is not
compiler generated (e.g. an ascii file read from
tape.) 5"

Procedure read(var c: char);

Procedure write(c: char):

"4 These routines control character by character
data transfer between the job and the input and
output processes. The synchronization of this
is handled by a solo pagebuffer monitor. The
typical programming sequence used for reading a
text file in the job process is as follows:
writearg(inp, arg):
specify file
repeat
read(c);
... process character
until c = em;
readarg(inp, arg):
if not arg.bool then ... : errors occurred
in transmission.
The programming sequence for writing a text file
character by character to the output process is
as follows:
writearg(out, arg):
specify file
while not done generating do ('next' or type seqcode
begin
...generate character c:
write(c):
end:
c:= em:
write(c):
if file.is_next then
begin
readarg(out, arg):
length_of_fill_in_pages:= arg.int
end:

9.10
PROCEDURE OPENIF (FILE: ID; IDENTIFIER: VAR FOUND: BOOLEAN);  
PROCEDURE CLOSE(F: FILE);  
PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);  
PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);  
FUNCTION LENGTH(F: FILE): INTEGER;  

"4. These routines allow the sequential program access to the solo resident file system. The procedure OPEN reads the FTLF with the given identifier for data transfer. After OPEN has been called the file is referred to by the file enumeration (1 or 2). CLOSE is the inverse of OPEN which purges the accessibility of the given file enumeration. GET and PUT allow for the transfer from or to the file page indexed by P. LENGTH returns the length (in pages) of the given file enumeration (it must currently be opened.)"

PROCEDURE MARK(VAR TOP: INTEGER);  
PROCEDURE RELEASE(TOP: INTEGER);  

"4. These two routines are used to control the dynamic storage capabilities of sequential Pascal. The procedure MARK returns an integer corresponding to the current heap TOP. The procedure RELEASE reverses this and sets the heap TOP to the program supplied value. (See the section on passing pointers.)"

PROCEDURE IDENTIFY(HEADER: LINE);  
PROCEDURE ACCEPT(VAR C: CHAR);  
PROCEDURE DISPLAY(C: CHAR);  

"4. These routines allow the sequential program access to the Pascal console. The procedure IDENTIFY should be executed in the initialization and after calling another program (using RUN.) The SOLO TYPE RESOURCE monitor and terminal class keep track of program identifiers and display them on the console as necessary. The header is a programmer defined string (e.g. the program name.) ACCEPT is used to receive a character from the console and DISPLAY is used to send a character. The ASCII LINE FEED character (DECIMAL CODE 10) is used to terminate a line of transmission to the console, both on input and output."

PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE; VAR EOF: BOOLEAN);  
PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);  

"4. These two routines allow for transfer of data between the input, job, and output processes. Like read and write the synchronization is handled by the SOLO pagebuffer monitor. Here, however, the EOF of NULL (END) character does not signify the end of transmission. Instead, with each call a boolean EOF is included. If this boolean is true, it signifies that this page
IS EMPTY AND THAT THE LAST PAGE TRANSFERRED IS THE LAST. AS IN USING READ AND WRITE.
THE JOB PROCESS MUST FIRST INSTRUCT THE IO PROGRAM TO LOAD THE CORRECT PROGRAMS INTO THE IO PROCESSES BY USING THE WRITELINE ROUTINE AND FIND THE STATUS OF THE COMPLETED TRANSFER AFTER EOF USING THE READARG ROUTINE. 5"

PROCEDURE READLIN( VAR TEXT: UNIV LINE);;
PROCEDURE WRITELIN( TXFIN: UNIV LINE);;
"4. THESE ROUTINES ARE NO-OPS TO THE JOB PROCESS.
IN THE INPUT PROCESS, READLIN MAY BE USED TO
REQUEST A CARD IMAGE FROM THE SOLO CARDS.
PROCESS. SIMILARLY THE OUTPUT PROCESS MAY USE THE
WRITELINE PROCEDURE TO SEND A LINE TO THE PRINTER.
PROCESS. 5"

PROCEDURE READARG(S: ARGSEN; VAR ARG: ARGTYPE);;
PROCEDURE READARG(S: ARGSEN; ARG: ARGTYPE);;
"4. THESE TWO PROCESSES HAVE BEEN REFERRED TO
SEVERAL TIMES ABOVE. THEY ARE THE PRIMARY
SOURCE OF PROCESS SYNCHRONIZATION IN SEQUENTIAL
PASCAL UNDER SOLO. INITIALLY SOLO LOADS THE IO
PROGRAM INTO THE JOB PROCESS PARTITION AND THE IO
PROGRAM INTO BOTH THE INPUT AND OUTPUT
PROCESS PARTITIONS. IO ACCEPTS COMMANDS FROM
THE CONSOL AND LOADS THE APPLICABLE PROGRAMS
USING THE RUN ROUTINE. IT IS THEN THE RESPONSIBILITY
OF THE SEQUENTIAL PASCAL PROGRAM TO NOTIFY THE
IO PROGRAM OF WHICH DRIVER SHOULD BE LOADED IN
THE INPUT OR OUTPUT PROCESS PARTITIONS. THIS
IS DONE USING THE WRITELINE ROUTINE WITH THE NAME
OF THE (SEQCODE) DRIVER TO BE LOADED, OR THE
(ASCII) FILE TO BE TRANSFERRED FROM DISK (USING
THE DISK DRIVER). WHEN TRANSFER IS COMPLETED,
THE JOB PROCESS DETERMINES THE STATUS OF THE
COMPLETE TRANSFER FROM THE IO PROGRAM BY USE OF THE
READARG ROUTINE AS SHOWN IN READ AND WRITE.
PROGRAMS WITTEN TO EXECUTE IN THE INPUT AND
OUTPUT PROCESS PARTITIONS NEED NOT USE THESE
ROUTINES AS ALL COORDINATION IS PERFORMED BY
THE IO PROGRAM. 5"

PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);;
"4. THIS ROUTINE ALLOWS THE SEQUENTIAL PASCAL
PROGRAM TO DETERMINE WHETHER A GIVEN FILE EXISTS
AND IF SO WHAT ITS ATTRIBUTES ARE. IT USES
THE RESIDENT SOLO FILE ROUTINE TO SEARCH THE
DISK CATLOG. 5"

PROCEDURE IOTRANSFER
(DEVICE: IODEVICE; VAR PARA: IOPARA; VAR FLOCK: UNIV PAGE);;
"4. THIS ROUTINE ALLOWS THE SEQUENTIAL PASCAL
PROGRAM TO ACCESS A PHYSICAL DEVICE (E.G. THE
TAPE DRIVE) FOR DATA TRANSFER. THE PARAMETERS
ARE DESCRIBED IN THEIR TYPE DEFINITION ABOVE. 5"

PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARA: IOPARA);;
"4. THIS ROUTINE ALLOWS FOR THE CONTROL OF
POSITIONING OF PHYSICAL DEVICES. IT IS
CURRENTLY ONLY IMPLEMENTED FOR THE TAPE (TAPEDEVICE). 5"

9.12
FUNCTION TASK: TASKKIND:
"4 THIS FUNCTION ALLOWS THE SEQUENTIAL PASCAL
PROGRAM TO DETERMINE INTO WHICH PROCESS PARTITION
IT HAS BEEN LOADED. THIS IS NECESSARY TO DETERMINE
WHAT FACILITIES ARE AVAILABLE THROUGH THESE
INTERFACE ROUTINES. 5"

PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
VAR LINE: INTEGER; VAR RESULT: PROGRESRESULT);
"4 AS NOTED ABOVE, THE RUN PROCEDURE ALLOWS A
SEQUENTIAL PASCAL PROGRAM TO CALL ANOTHER PROGRAM.
THE IDENTIFIER IS THE NAME OF THE SOURCE CODE DISK
FILE CONTAINING THE DESIRED PROGRAM. THE LINE
WILL IDENTIFY THE LINE NUMBER IN ERROR IN THE
EVENT OF UNNORMAL TERMINATION. THE PARAMETERS
NEEDED BY THE PROGRAM SHOULD BE SUPPLIED IN
THE PARAM ARGLIST. THE FIRST OF THESE (PARAM(1))
IS STANDARDLY A BOOLEAN WHICH WILL BE RETURNED
BY THE CALLED PROGRAM TO NOTIFY WHETHER IT
COMPLETED PROCESSING NORMALLY. 5"

PROGRAM P(VAR PARAM: ARGLIST):
"4 THIS IS THE PROGRAM STATEMENT WHICH SIGNIFIES
THE END OF THE PREFIX. IT ALSO CONTAINS THE
PARAMETERS WHICH HAVE BEEN PASSED TO THIS PROGRAM.
THEY MAY BE ACCESSED AS PARAM(1)."

"INSERT YOUR PROGRAM HERE AND YOU'RE READY TO GO. GOOD LUCK!"

9.13
GROUP 10: RUNNING THE PROGRAM

LOADING SOLO (10.2)

CONSOLE COMMANDS (10.3)

SAMPLE RUN (10.5)

SAMPLE PROGRAM (10.6)

COMPILER OPTIONS (10.7)

COMPILER ERROR MESSAGES (10.8)

RUN TIME ERROR MESSAGES (10.12)
LOADING SOLO/PASCAL AS AN OS-32M/T TASK

(Assume all necessary files are initialized on disk, see Pascal Packaging Manual.)

$COPY; OS-32M/T

"AT THE OPERATORS CONSOLE"

PASCAL CAR, SYS2;PASCAL.VD

"BRING UP PASCAL UNDER SOLO AT CARROSEL"

"VIRTUAL DISK = SYS2;PASCAL.VD"

"$COPY IS ON"

"SYSTEM CUES USER WITH 'DO:' AT PASCAL TERMINAL"
GETTING HELP WITH PASCAL OPERATOR COMMANDS

**type**

HELP .... cues user to type LIST

command_name_w/o_parameters .... explains the parameters.

   e.g:

command_name_with_parameters .... executes command

object_program_name_with_any_parameters .... executes program

**DO:**

COPY
COPY:
TRY AGAIN
   COPY(SOURCE, DESTINATION: IDENTIFIER)
DO:

CONCAT
CONCAT:
TRY AGAIN
   CONCAT(SOURCE(.1.), ..., SOURCE(.N.), DESTINATION: IDENTIFIER)
DO:
SELECTED COMMANDS AT THE PASCAL OPERATORS CONSOLE

FILE OPERATIONS

COPY . . . . copies a file
DO . . . . . executes a command or a file of commands
CONCAT . . . . concatenate files
FILE . . . . . stores a file on disk
LIST . . . . . list all files of selected types

these use "devices":

INPUT
CARD = card reader
APPEND = card + append standard prefix
CONSOLE = PASCAL console
DISK = virtual disk

OUTPUT
PRINTER = virtual printer
REMOVE = remove prefix PRINTER

EDITING

EDIT . . . . simple editor from C.I.T.
PEDIT . . . . similar to OS-32M/T editor

RUN

RUN . . . . compile and run
SPASCAL . . . compiles a program
name . . . . executes named PASCAL object program
SAMPLE RUN

"AT THE PASCAL CONSOLE"
KSU PASCAL INTERPRETER ROO-00
DO:
"READ CARDS"
COPY(CARDS,MYFILE)
DO:
"EDIT SAMPLE PROGRAM"
PEDIT(MYFILE)
PEDIT:
RE
  38 END;
EOF
CH;/;
  38 END.
EN
DO:
CONCAT(PREFIX,MYFILE,MYFILE)
DO:
"COMPIL"E
SPASCAL(MYFILE,PRINTER,OBFILE)
DO:
"EXECUTE WITH I/O TO CONSOLE"
OBFILE
THIS IS A STRING
THIS IS A STRING
COMPARE
OBFILE
THIS IS A STRING
THIS IS
NO COMPARE
OBFILE
THIS IS A STRING
THIX IS A STRING
NO COMPARE

10.5
VAR STRING1, STRING2: LINE;
   I: INTEGER; OK: BOOLEAN;

PROCEDURE REALTEXT(VAR TEXT: LINE);
VAR I: INTEGER; C: CHAR;
BEGIN
   I := 0;
   REPEAT
      I := I + 1;
      ACCEPT(C);
      TEXT(.I.) := C;
      UNTIL C = NL;
END;

PROCEDURE WRITETEXT(TEXT: LINE);
VAR I: INTEGER; C: CHAR;
BEGIN
   I := 0;
   REPEAT
      I := I + 1;
      C := TEXT(.I.);
      DISPLAY(C);
      UNTIL C = NL;
END;

BEGIN
   READTEXT(STRING1);
   READTEXT(STRING2);
   I := 0;
   OK := TRUE;
   REPEAT
      I := I + 1;
      IF STRING1(.I.) < STRING2(.I.) THEN OK := FALSE;
      UNTIL (STRING1(.I.) = NL) OR (STRING2(.I.) = NL) OR (OK = FALSE);
      IF OK
         THEN WRITETEXT("COMPARE::10::")
      ELSE WRITETEXT("NO COMPARE::10::")
   END.
COMPILER OPTIONS

NUMBER . . . only procedures declared will be numbered
(reduces amount of code X 25%)

CHECK . . . (really no check) eliminates run time
checks for legal enumeration_values, non_NIL
printer_values.

TEST . . . for debugging the compiler.

XRED . . . produce a cross reference table.

Default: none of the above.
COMPILATION ERROR MESSAGES

Example:

*** ***  PASS 2  LINE 91  CONSTANT DEFN SYNTAX
*** ***  PASS 3  LINE 210  INVALID NAME USAGE
*** ***  PASS 3  LINE 215  INVALID SUBSCRIPTING

PASS1 LEXICAL ANALYSIS

*ENDLESS COMMENT.*
*INVALID NUMBER.*
*TABLE OVERFLOW.*
*INVALID STRING.*
*BAD CHARACTER.*
SEQL PROGRAM.
DECLARATION.
CONSTANT DFN.
TYPE DFN.
TYPE.
ENUMERATION TYP.
SUBRANGE TYPE.
SET TYPE.
ARRAY TYPE.
RECORD TYPE.
STACK LENGTH.
VAR DECLARATION.
ROUTINE.
PROCEDURE.
FUNCTION.
WITH STMT.
PARAMETER.
BODY.
STMT LIST.
STATEMENT.
ID STMT.
ARGUMENT.
COMPOUND STMT.
IF STMT.
CASE STMT.

LABEL LIST.
WHILE STMT.
REPEAT STMT.
FOR STMT.
CYCLE STMT.
EXPRESSION.
VARIABLE.
CONSTANT.
INIT STMT.
TERMINATION.
PREFIX.
INTERFACE.
POINTER TYPE.
PASS 4  SEMATIC CHECKS

*UNRESOLVED ROUTINE,*
*AMBIGUOUS IDENTIFIER,*
*COMPILER ABORT,*
*INVALID CONSTANT,*
*INVALID SUBRANGE,*
*MISSING ARGUMENT,*
*NOT A ROUTINE,*
*TOO MANY ARGUMENTS,*
*LABEL VALUE TOO LARGE,*
*INVALID LABEL,*
*AMBIGUOUS LABEL,*
*INVALID WITH VARIABLE,*
*INVALID INITIALIZATION,*
*NOT A FUNCTION,*
*INVALID NAME USAGE,*
*INVALID SELECTION,*
*INVALID SUBSCRIPTING,*
*INVALID INTERFACE,*
*INVALID CALL,*
*INVALID POINTING,*
*INVALID RESOLUTION.*

PASS 4  MORE SEMATIC CHECKS

*INVALID NESTING,*
*ADDRESS OVERFLOW,*
*ACTIVE VARIABLE,*
*QUEUE VARIABLE,*
*NESTED PROCESS,*
*INVALID ENTRY VARIABLE,*
*INVALID FUNCTION TYPE,*
*RECORD ENUMERATION,*
*LONG ENUMERATION,*
*INVALID INDEX TYPE,*
*INVALID MEMBER TYPE,*
*PROCESS STACK USAGE,*

*INVALID PARAMETER,*
*COMPILER ABORT,*
*ODD LENGTH STRING TYPE,*
*INVALID RESOLUTION,*
*INVALID TAG TYPE,*
*RECORD POINTER TYPE.*

10.10
PASS 5  MORE SEMANTIC ANALYSIS

"COMPILER ABORT."
"OPERAND TYPE."
"NOT A VARIABLE."
"NOT ASSIGNABLE."
"INVALID INITIALIZATION."

PASS 6  CODE GENERATION AND FLOW ANALYSIS

"TOO MUCH STACK."
"TOO MUCH CODE."

10.11
RUN TIME ERROR MESSAGES

FORM1

TASK NAME: SYSTEM LINE "LINE NUMBER"
TASK NAME: "ERROR MESSAGE"
TASK NAME: TASK PAUSED

FORM2

TASK NAME: JOB LINE "LINE NUMBER"
TASK NAME: "ERROR MESSAGE"
TASK NAME: TASK PAUSED

ERROR MESSAGES

1) OVERFLOWERROR . . . arithmetic error
2) POINTERERROR . . . illegal pointer
3) RANGEERROR . . . illegal value for an enumeration
4) VARIANTERROR . . . illegal value for an enumeration in a record
5) HEAPLIMIT . . . exceed size of HEAP
6) STACKLIMIT . . . exceed size of execution stack

Examples

*PASCAL CAR, USR6:A/PASCAL.VD2
*11:28:17 CAR:SYSTEM LINE 280
*11:28:17 CAR:RANGE ERROR
*11:38:17 CAR:TASK PAUSED
*T CAR
*CAN
*11:28:17 CAR:END OF TASK 255
GROUP 11: DIFFERENCES BETWEEN PASCAL

REPORT AND SPASCAL (11.2)
DIFFERENCES

USER MANUAL AND REPORT

{COMMENT}

begin (UPPER AND LOWER CASE)

ARRAY [1..10] OF INTEGER

+ (FOR POINTERS)

PACKED

AND

FILE

FORWARD (NOT RESERVED)

GO TO

LABEL

NIL

NOT HERE

PROGRAM

KSU IMPLEMENTATION

"COMMENT"

BEGIN (UPPER CASE ONLY)

ARRAY (.1..10.) OF INTEGER

@ (FOR POINTERS)

NOT IN KSU IMPLEMENTATION

&

NOT IN KSU IMPLEMENTATION

FORWARD (RESERVED WORD)

NOT IN KSU IMPLEMENTATION

NOT IN KSU IMPLEMENTATION

UNIV

PREFIX + PREFIX ROUTINES.

PROGRAM

11.2
A SEQUENTIAL PASCAL MANUAL FOR FORTRAN PROGRAMMERS

by

JERRY DEAN RAWLINSON
B. S., University of Illinois, 1964

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
KANSAS STATE UNIVERSITY
Manhattan, Kansas

1977
This report is designed to serve as an instructional aid in the introduction of FORTRAN programmers to the Kansas State University (KSU) implementation of the programming language Sequential Pascal.

The report has been structured to take maximum advantage of the FORTRAN programmer's previously acquired knowledge of the FORTRAN language. Basic language terms are not defined as the programmer is assumed to be familiar with them. Based on this assumption, FORTRAN and Sequential Pascal examples are presented in a sequence designed to allow the programmer to quickly grasp the similarities and differences between the two languages. Typical FORTRAN programming problems are presented along with the corresponding Sequential Pascal solution. This approach allows the user to make direct comparisons between the two languages and provides for quick reference when the user wishes to find a Sequential Pascal solution to a typical FORTRAN programming problem.

Significant differences between the two languages are emphasized through the use of illustrations and warning statements. Programming examples are also used to introduce the user to Sequential Pascal capabilities which can not be duplicated in FORTRAN.

The use of this report as an instructional aid should significantly reduce the time required to train a FORTRAN programmer to write Sequential Pascal programs.