1976

TEXT EDITOR IMPLEMENTATION FOR THE THIRD NORMAL FORM SYNTHESIS SYSTEM

BY

T. J. STEVENS

B.S., UNIVERSITY OF NEBRASKA, OMAHA, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

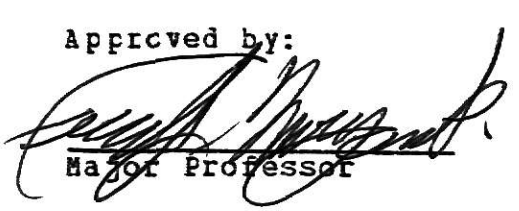
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978

Approved by:



Major Professor

ACKNOWLEDGEMENTS

For their guidance and assistance in the preparation of this report, I would like to thank Doctor Fred Maryanski, Doctor Paul Fisher, and Ed Basham.

TJS

ILLEGIBLE DOCUMENT

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

THIS IS THE BEST COPY AVAILABLE

TABLE OF CONTENTS

ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

1.1 Defining the Project Problem

This report deals with the implementation of a Text Editor for the THIRD NORMAL FORM SYNTHESIS SYSTEM. The goal of this report is to describe in a top down structured format how the Text Editor was conceived and developed.

The need of a Text Editor was created in the initial implementation of a Data Base Management System (DBMS) based upon a relational model. To process data in a relational environment requires the attributes of a relationship to be consistent and non-duplicative[2]. To obtain this state, one must maintain proper representation of the data in a Third Normal Form (3NF). Several methods for reducing data to 3NF have been developed. One of these is Bernstein's algorithm which automatically synthesizes 3NF relations from functional dependencies of attributes. A decision was made to use Berstein's algorithm and to establish a basic system called the THIRD NORMAL FORM SYNTHESIS SYSTEM.

As more detailed study went into the development of this system, it was decided to set up four distinct sections as delineated by function. These are as follows:

1. User Interface
2. Functional Dependency Generator
3. Functional Dependency Analyzer
4. Text Editor

1-2 Defining The Text Editor Problem

The Text Editor function was defined as to have the ability to alter input statements from the User Interface phase for reprocessing of that input. The development of the problem statement led to defining the function of a Text Editor and specifying its capabilities for the 3NF Synthesis System.

A Text Editor is an interactive software tool that uses pattern matching techniques in allowing the on-line user to create and modify symbolic text for any purpose[6]. User capabilities should include inserting, deleting and changing lines of texts, and a symbolic search feature with a program listing function[1].

With this definition in mind, and the problem statement requirement, the Text Editor for the 3NF Synthesis was evaluated. The use of the system editor of the NCR 8250 System was considered because of its availability. One major argument against using the system editor was to make the 3NF Synthesis System portable for commercialization. Additionally, the system editor was slow and required more than twice the storage space of the proposed editor. So a decision was made to construct a completely new Text Editor in NCR interactive COBOL language[4].

CHAPTER 2

BACKGROUND

2-1 General Overview of 3NF Synthesis System

To fully understand why the 3NF Synthesis System was develcped, requires a basic understanding of Data Base Management Systems (DBMS). Currently, there are three recognized data models in DBMS that are used and they are as follows:

1. Network
2. Hierarchical
3. Relational

The most common commercial systems used are the Network and Hierarchical data models. The relaticnal mcdel is not yet commonly used because of its extra memory and processor requirements. However, the Relaticnal data model is the only data model that has a sound workable mathematical basis.

Therefore to develop a workable Relational Data Base Model requires the initial set up of data and its relationships into third normal form (3NF). By representing data in 3NF allows the use of the mathematical tools of Relaticnal Algebra and Calculus[5] in constructing and using data in a relational environment. The representing of data in 3NF is not a trivial task. Therefore, the need for a 3NF synthesis algorithm arises. To use Berstein's algorithm requires the input to be in a valid form of functional dependencies. To take basic data off a business report of a commerical user and synthesized functional dependencies requires considerable programming effort.

2-2. Structure of the System

The 3NF Synthesis system accepts a description of the user data in a form that can easily be realized from a typical report. This input is collected interactivly by the User Interface program which produces a hierarchical representation of the data. This output is known as the Data Description Source File. The Functional Dependencies Generator program (FDGEN) reads the hierarchical data representation and interacts with the user to produce a set of functional dependencies. Then the Functional Dependencies Analyzer (FDA) which implements the Bernstein's algorithm generates relations in 3NF.

It is possible that the user might make a mistake during the User Interface program or that an error may be detected during the Functional Dependencies Generator program. In order to prevent the user from having to reenter all the data from the Data Description Source File, the Text Editor was developed. The editor operates on the output of the User Interface Program and permits the user to change any erroneous information. The editor then produces a file which is used as input to a special version of the User Interface program which checks for additional errors and reconstructs the hierarchical data representation. The flow of the system is depicted in Figure 1. Additional information on the 3NF Synthesis system can be found in reference 2.

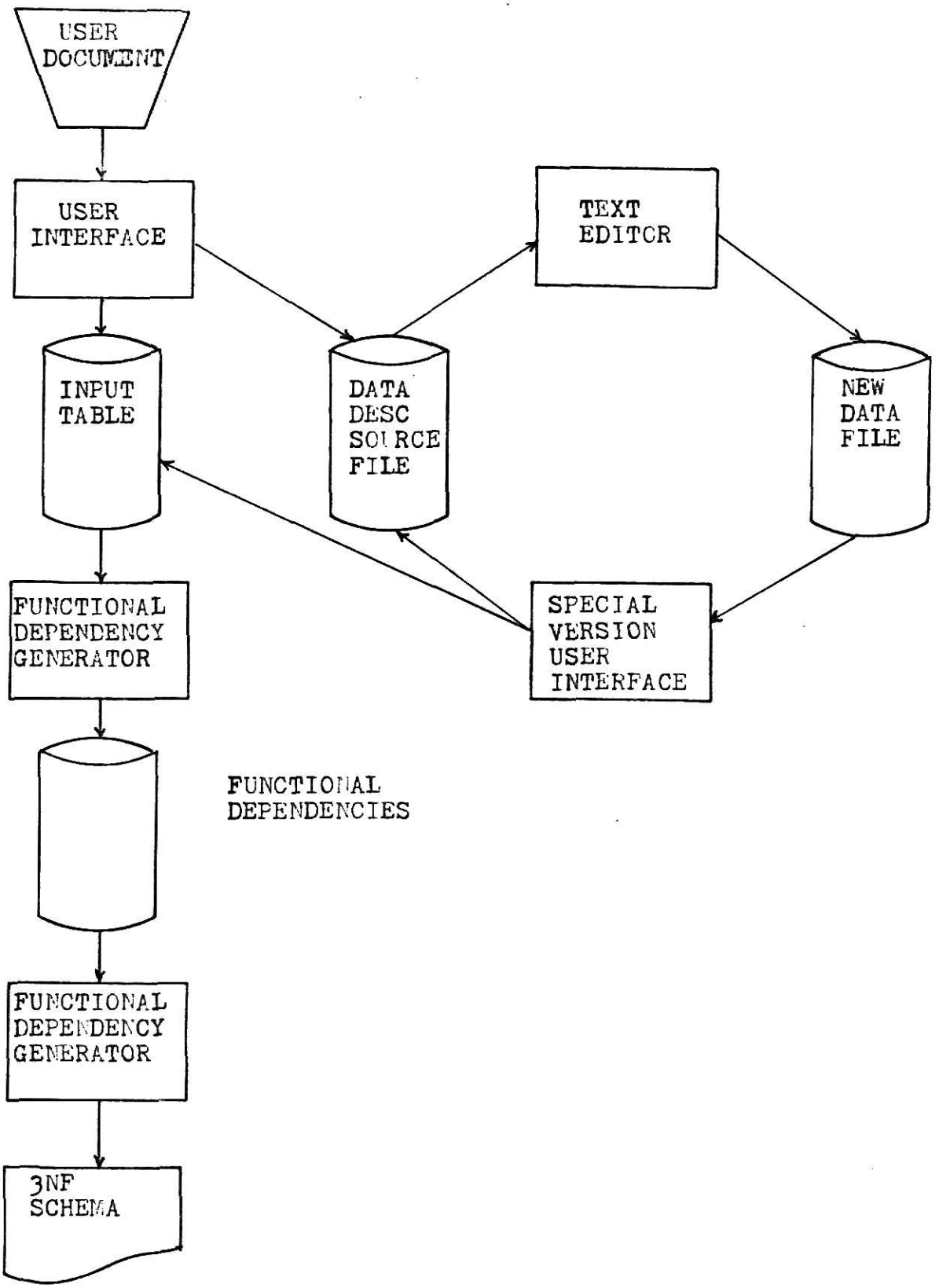


Figure 1: DATA FLOW OF THE THIRD NORMAL FORM SYNTHESIS SYSTEM.

2-3 General Features of Other Editors

All interactive computing systems have some form of editing facility but it is often primitive. Basically, the editor when initiated, copies a file into an internal buffer. The text is modified in the buffer and eventually written back to some external file. The text is never modified except by explicit command. If a file is ruined in the editor routine, it can be cancelled out and a fresh copy read into the buffer and editing start anew. Most editors are "line oriented" in that most editing commands operate on one or more lines. This is a natural organization since most text intrinsically comes in lines. Most editors are not specific about the structure on lines. It is usually presumed that the user of an editor program knows what he is doing and does not require detailed information. The common features are:

Add lines

Delete lines

Print lines

Change or modify characters

Find character(s)

Concatenate strings

Merge lines

Sort lines by keys

One of the major consideration in designing an editor is to structurally engineer the interactive commands to be concise and clear. Error recovery is the second major influence in

the design. The editor maintains precious files and so must be cautious in that when a user enters erroneous commands, it must recovery gracefully and not lose the files[1].

CHAPTER 3

TEXT EDITOR DESCRIPTION

3-1 Design Requirements

The Text Editor designed for the Third Normal Form (3NF) Synthesis System had to satisfy the initial following conditions.

1. Operate upon the Data Description Source File produced by the User Interface program.

2. Provide basic deletion, insertion and resequencing capabilities.

3. Output a modified file using the same data structure used by the input file. This modified file must be sequentially processed through a modified User Interface program which checks for additional errors and reconstructs the hierachial data representation.

4. Implement the editor on the NCR 8250 System using the NCR Interactive COBOL language.

3-2 Design Characteristics

In addition to the basic requirements the following design characterstics were included to allow for a more complete and useful Text Editor. These characteristics are as follows:

1. Make the editor a powerful software tool in creating and modifying the Data Description Source File.

2. Stress error recovery in assisting the user.

3. Human engineer the interactive CRT display of questions and commands for conciseness and simplicity for the unsophisticated user.

4. Design the editor in a top down fashion.

5. Modularize the different function of the editor.

3-3 Overview of Lifecycle

From the basic requirements and design considerations the overall functions and routines of the Text Editor evolved. The editor was implemented using the NCR version II Interactive COBOL language. The editor was structured so that each function would be modular and independent. This allowed the modules to be in different stages of construction and facilitated debugging and testing of the editor program. The editor was envisioned to perform the following operation on the Data Description Source File:

1. Display the file.
2. Delete lines of the file.
3. Add lines to the file.
4. Resequence the lines of the file.
5. Find a character string in the file.
6. Change a character string in the file.
7. Provide a tutorial on the use of the editor.
8. Be able to exit the program gracefully and save the modified file or return the original Data Description Source File.

The first four functions, Display, Add, Delete, and Resequence, are line oriented and were envisioned to operate on a single line, groups of contiguous lines, or all the lines in the file. The two functions, Find and Change, were

character oriented thus permitting the user to locate and change the first occurrence of a specific character string in each line of the file. The HELP function provides a tutorial on the syntax and semantics of the other functions.

3-4 Design Philosophy

In accordance with top down structuring techniques [3] and from the definition of the editor requirement, data structures (Appendix A) and data access graphs (Figure 2) were completed. These software tools gave an overview of the editor program and indicated possible trouble areas. A high level algorithm (Appendix A) was constructed. At the completion of the algorithm, a walk through was conducted for each module to see if the program would fulfill the basic and design requirements. Error recovery was stressed and test cases were developed. As each module was coded, testing and debugging took place. Approximately sixty hours of machine time was used for the complete program. Documentation of each module was kept in a historical file to preclude the same error being made in another module.

3-5 Features

The finialized form of the Text Editor has the following features that provide editing capabilities of the

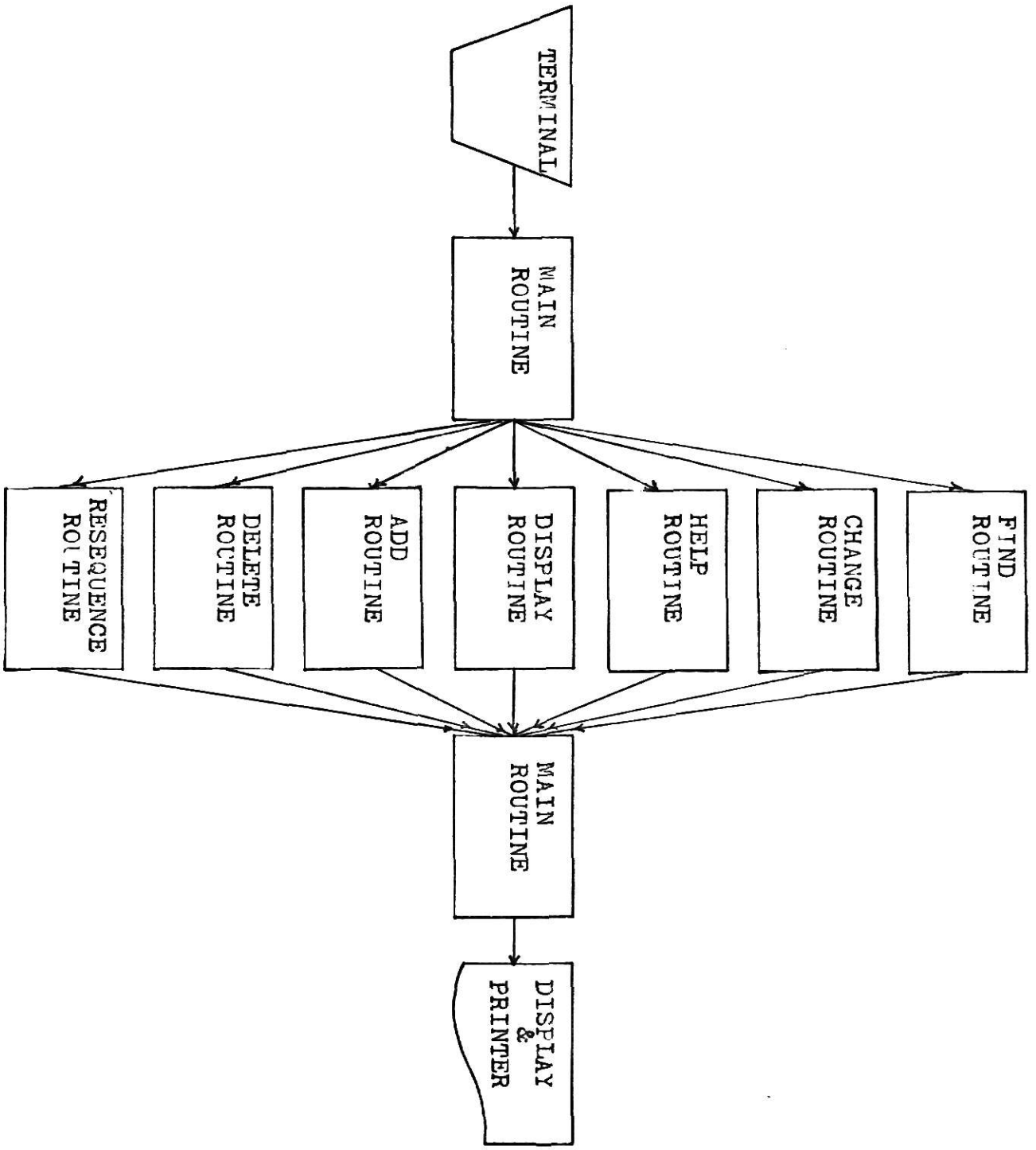


Figure 2: TEXT EDITOR DATA ACCESS GRAPH

Data Description Source File that is created during the User Interface program:

DISPLAY ON CRT

ADD

DELETE

RESEQUENCE

FIND

CHANGE

HELP

DISPLAY FUNCTION

The DISPLAY function displays on the CRT the Data Description Source File. The complete file, selected portions of the file or a single line can be displayed. The format is "D aaaa ,bbbb ", where aaaa is the beginning line number and bbbb is the ending line number to display. To display the complete file "D ALL" is the format. General format:

$$\text{DISPLAY or D} \left[\begin{array}{l} \text{,aaaa} \\ \text{,aaaa,bbbb} \\ \text{ALL or A} \\ \text{LAST or L} \end{array}\right]$$

Blanks, stars and commas are used as delimiters in the command. In "DISPLAY ALL", the screen displays 22 records then request new/line to be typed in to roll the screen.

DELETE FUNCTION

The DELETE function deletes one to n lines of existing records from the Data Description Source File. Format is

"DEL aaaa ,bbbb ", where aaaa is the beginning line number and bbbb is the ending line number. Presence of aaaa indicates deletion of a single line. General format:

```
DELETE [,aaaa        ]
or   D [aaaa,bbbb    ]
```

Blanks, stars and commas are used as delimiters in the command. If the complete file is deleted, no records can be added from the editor routine.

ADD ROUTINE

The ADD function inserts one or more lines to the existing Data Description Source File. The format is "ADD AFTR nnnn", where nnnn is a valid line number of an existing record. One or more lines can be added. Additionally a previous record can be recopyed by typing in the line number. To stop the ADD function, a blackslash "\" must be typed in position 1 of the last line of input. General format:

```
   ADD [AFTR nnnn          ]
or  A  [,AFTR,nnnn         ]
    \  [(to stop function) ]
```

Blanks, stars and commas are used as delimiters in the command. No more than 100 lines are allowed in the Data Description Source File.

RESEQUENCE FUNCTION

The RESEQUENCE function allows the line numbers of the Data Description Source File to be changed to new sequential

line numbers. Format is "RESEQ cccc ,dddd", where RESEQ will resequence the complete file starting at line number 0000, by default, by 10. The use of cccc indicates the starting line number and dddd indicates the amount the lines are tc be resequence. General format:

```
RESEQ  [                ]
or RE  [ ,cccc          ]
       [  cccc,dddd     ]
```

Blanks, stars and commas are used as delimiters for the command. Although the lines can be resequenced to any amcunt while in the editor routine, they are resequenced by 10 when the editor is terminated. This resequence is necessary as a result of the structure of the User Interface program[2].

FIND FUNCTION

The FIND function allcws for a string eeee to be found in the Data Description Source File. The search area can be a single line, a group cf contiguous lines or the complete file. Format is "FIND/eeee/ALL", where the maximum length of eeee is 40 characters. General format:

```
FIND[/eeee/  ALL or A   ]
or  [/eeee/  aaaa       ]
  FI[/eeee/  aaaa,bbbb  ]
```

String eeee must be enclosed by slashes. Commas and stars are delimiters. If a string eeee is not found in the search area then a message to indicate string not found is displayed.

CHANGE FUNCTION

The CHANGE function allows a string eeee to be changed to string ffff in the first occurrence of a single line, a group of contiguous lines, or the complete file. Format is "CH/eeee/ffff/ALL,V" where "V" means the old and modified lines are displayed for user's verification. General format:

```
CHANGE [/eeee/ffff/ ALL or A                ]  [,V]
 or CH [/eeee/ffff/ aaaa                    ]
                    aaaa,bbbb               ]
```

Slashes are used to enclose strings eeee and ffff. Blanks and commas are used as delimiters.

HELP FUNCTION

The HELP function provides a visual display on the CRT of the general formats of the six other functions of the editor routine. Two pages are used in the visual display. The first page contains the formats of DISPLAY, DELETE and ADD. When the viewer finishes with the page new/line is typed in which displays the second page. The formats of FIND, CHANGE and RESEQUENCE are displayed. The format is "HELP".

CONCLUSION

To terminate the Text Editor, the format "QUIT" or "QU" is used. Once this is entered, the question whether the changed file (Edit File) is to be saved is displayed. If the answer is "yes", the Edit file replaces the old Data

Description Source File. If the answer is "no", the old Data Description Source File is returned instead of the modified file.

3-6 OVERVIEW OF THE TEXT EDITOR

In order to use the Text Editor, job control language statements are required to assign the Data Description Source File to the editor's Old-Data-File. This file is read into a working storage table called WS-Table. The format of the input is shown in Figure 3. The size of the WS-Table is 101 lines with each line containing a maximum of 77 characters. Line 101 of the WS-Table is used for the Find and Change routines. As the file is read into the table the lines are doubly linked by pointers WS-NEXT and WS-LAST. If the file being read does not take all 100 lines, then the remainder are doubly linked in a free list that is used in the Add and Delete routines (Fig 4). Once the file is read in, a question on the use of the line printer is asked. If the answer is yes to the question, then the line printer is set up to produce a printed output when the editor routine is terminated.

Next a statement appears on the CRT asking the user to enter the appropriate command for the needed edit function. To facilitate the discussion of the editor functions and user interactions, each command will be discussed in the order displayed on the CRT.

```
POS  1 - 4 5 6 7                                                    76
     XXXX  C b XX----------------------------------------------------XX
```

User Data

COLUMN 1-4 : SEQUENCE NUMBER
COLUMN 5 : STATUS
COLUMN 6 : LEVEL
COLUMN 7-76 : USER DATA

Figure 3: INPUT DATA STRUCTURE FROM USER'S DATA DESCRIPTION SOURCE FILE

WS-TABLE

WS-LAST 1 - 3	LINE NO	SEQ NO 1 4567	USER DATA	77	WS-NEXT 1 - 3
0	1	0000	REPORT ORGANIZATION		2
1	2	0010	DEPT DBUDGET MANAGER	Y	3
2	3	0020	EMP PROJ OFFICE PHONE	Y	4
3	4	0030	PROJ PBUDGET	S	5
4	5	0040	OFFICE AREA	S	6
5	6	0050	END		0
0	7				8
7	8				9
•					
•					
•					
97	98				99
98	99				100
99	100				0
	101				

FREE LIST POINTER = 7

Figure 4: DATA STRUCTURE OF WS-TABLE

If the user requests the DISPLAY function, then a single line, multiple lines, or the complete file can be displayed. The DISPLAY function is line oriented on the sequence number of each line. If "DISPLAY ALL" is the command typed in, then the routine sets pointers at the start of the file (WS-START) and searches until the end of the file and sets the ending pointer (WS-END). The Write submodule is called and displays line by line of the doubly linked table until the end pointer is reached. If more than 22 lines are to be displayed, a roll feature allows the first 22 lines to be displayed. The screen is rolled when new/line is typed in which allows the rest of the lines to be displayed. If the command is to "DISPLAY nnnn" for a single line then the table is searched until nnnn is found. A pointer WS-START is set and the write submodule is called and the line nnnn is displayed. If the command is "DISPLAY nnnn,mmmm" then the table is searched for nnnn. Once found the pointer WS-START is set and the the table is searched for mmmm. Once found the pointer WS-END is set. The write submodule is called and all lines that are linked from WS-START to WS-END are displayed. The command "DISPLAY LAST" is handled like a single line. Error routines include the standard sequence not found, sequence out of order and syntax error in formatting.

The DELETE routine deletes lines from the WS-Table and adds the deleted lines to the free list. The command "DELETE nnnn" will delete only one line. When the command

is entered the format is checked and the line nnnn is searched for in the table. Once found the pointer WS-START is set and the write submodule is called to display the line nnnn. The statement that the following lines are to be deleted is displayed with a request for verification. This prevents the erroneous deletion of pertinent information. If the deletion is verified then the pointer WS-START is moved to the free list while the index pointers WS-LAST and WS-Next are modified for the WS-Table and free list. This is the same as deleting a node from a doubly linked list and adding the deleted node to space available list. See Figure 5 for an example. If more than one line is to be deleted then the pointer WS-END is set up to point to the last line to be deleted. No change is made in the deletion of the node submodule. The standard error routine for formatting and sequencing are the same.

WS-TABLE

WS-LAST	LINE NO	SEQ NO	USER DATA		WS-NEXT
1 - 3		1 4567		77	1 - 3
0	0	0000	REPORT ORGANIZATION		2
1	2	0010	DEPT DBUDGET MANAGER	Y	3
2	3	0020	EMP PROJ OFFICE PHONE	Y	4
3	4	0030	PRCJ PBUDGET	S	5
4	5	0040	OFFICE AREA	S	6
5	6	0050	END		0
0	7				8
7	8				9
•					
•					

FREE LIST POINTER = 7

BEFORE DELETION OF LINES 10-20

AFTER DELETION OF LINES 10-20

WS-LAST	LINE NC	SEQ NO	USER DATA		WS-NEXT
1 - 3		1 4567		77	1 - 3
0	1	0000	REPCRT ORGANIZATICN		4
0	2				3
2	3				7
1	4	0030	PROJ PBUDGET	S	5
4	5	0040	CFFICE AREA	S	6
5	6	0050	END		0
3	7				8
7	8				9
•					
•					

FREE LIST POINTER = 2

Figure 5: DELETION OF LINES FROM WS-TABLE

The ADD routine inserts lines by modifying the existing WS-LAST and WS-NEXT pointers in the free list of the WS-Table. This gives the appearance of the lines being added even though the lines are already in the WS-Table. When the command is of the form "ADD AFTR nnnn", the format is checked and then the WS-Table is searched for nnnn and the pointer WS-START is set. "ADD AFTER THIS LINE" and line nnnn are displayed on the CRT. The routine then waits for the data to be entered by the user. For each line entered, a question concerning the level of the data is asked [2]. When the user gives the correct response, the line pointed to by the WS-FREE-LIST pointer is inserted after line number nnnn. This is accomplished by the same technique of inserting a node in a doubly linked list. Figure 6 illustrates the use of the ADD function. Pointers WS-LAST and WS-NEXT are modified in the WS-TABLE. If the data entered is numeric, a line previously entered is to be repeated and the line number is searched for in the table. If not found a statement of invalid line number is given and the user is told to reenter the data for that line again. If the line is a beginning or ending of a report, the question of level is not asked. After the data has been entered the new line is displayed on the CRT. The add routine is terminated when the backslash "\" is typed in as data. The standard error routine for formatting and sequencing are the same. The ADD routine will allow no more than 100 lines of data to be in the WS-Table. If this

amount is exceeded an error message is displayed indicating maximum lines in the table.

The RESEQUENCE function reorders the line numbers by a requested amount from a starting line number until the end of file is reached. The command "RESEQ" resequences the complete file from the starting line to the end of the file. The amount that is resequenced defaults to 10. The command is unstrung into TEXT1, TEXT2 and TEXT3. If TEXT3 is blank then the amount is 10. If the command is "RE nnnn", which indicates that TEXT2 is numeric and indicates a starting point for the resequencing. A statement is displayed on the CRT indicating that resequencing has taken place from line nnnn by 10. The command "RE nnnn,mmmm" indicates that from starting point nnnn, the resequence will be by amount mmmm. A statement is displayed that indicates all lines are sequenced by mmmm from nnnn. If data is entered that makes TEXT3 not numeric or a not positive number, then an error message indicates that all lines are resequenced from nnnn by 10 and that the requested amount, mmmm, is not possible. When the routine is finished, control is returned to the main module. See Figure 6 for an example of resequencing.

Addition of 2 New Lines Added After Line 0030

```
ADD AFTER 0030

    CUSTOMER                                          N
    EMP  PROJ  OFFICE  PHONE                          Y

                              WS-TABLE

WS-LAST LINE NO SEQ NO          USER DATA                  WS-NEXT
 1 - 3          1   4567                              77   1 - 3
   0       1    0000   REPORT ORGANIZATION                   4
   4       2           CUSTOMER                        M     3
   2       3           EMP  PROJ  OFFICE  PHONE        Y     5
   1       4    0030   PRCJ  PBUDGET                  S     2
   3       5    0040   OFFICE  AREA                    S     6
   5       6    0050   END                                   0
   0       7                                                 8
   7       8                                                 9
   .
   .

FREE LIST POINTER = 7

              AFTER RESEQ OF WS-TABLE FROM 0000 BY 10

WS-LAST LINE NO SEQ NO          USER DATA                  WS-NEXT
 1 - 3          1   4567                              77   1 - 3
   0       1    0000   REPORT ORGANIZATION                   4
   4       2    0020   CUSTOMER                        N     3
   2       3    0030   EMP  PROJ  OFFICE  PHONE        Y     5
   1       4    0010   PRCJ  PBUDGET                  S     2
   3       5    0040   OFFICE  AREA                    S     6
   5       6    0050   END                                   0
   0       7                                                 8
   7       8                                                 9
   .
   .

FREE LIST POINTER = 7
```

Figure 6: ADDITION OF LINES AND RESEQUENCING THE WS-TABLE