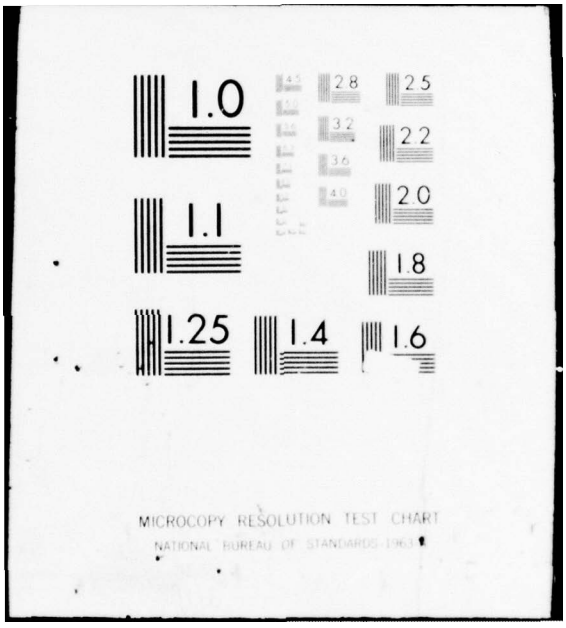
AD-A038 811 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A SHIPBOARD REPORT ORIGINATION SYSTEM UTILIZING A MICROCOMPUTER--ETC(U)
DEC 76 J G HOLYOAK

UNCLASSIFIED NL

1 OF 1
AD A038811

END
DATE FILMED
5-77


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963

(2)
B.S.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A SHIPBOARD REPORT ORIGINATION SYSTEM
UTILIZING A MICROCOMPUTER

by

Joseph Glade Holyoak

December 1976

Thesis Advisor: U. R. Kodres

Approved for public release; distribution unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A Shipboard Report Origination System Utilizing a Microcomputer

5. TYPE OF REPORT & PERIOD COVERED

Master's Thesis, December 1976

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Joseph Glade Holyoak

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE

Dec[redacted]76

13. NUMBER OF PAGES

71

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

report generator
data element
interactive
formated reports
error analysis
required recurring reports

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A Report Origination Systems (ROS) has been implemented, using an inexpensive mircocomputer system, to help ease the administrative burden facing Navy shipboard managers. The system is an interactive line editing system, with optional prompting, which enables a person who is unfamiliar with the report format to respond to queries in order to edit a highly formatted report. Automatic error checking is performed using

DD FORM 1 JAN 73 1473 (Page 1)

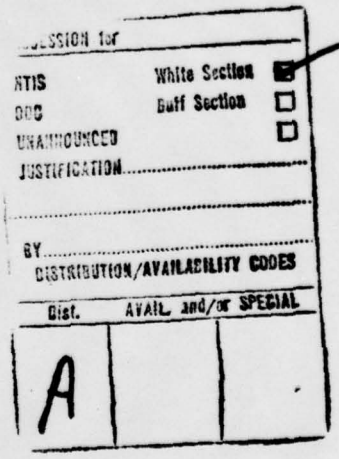
EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED 251 450

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (cont.)

a previous edition of the report as a basis. The system allows creation of new formatted reports. The use of a general purpose microcomputer system makes the editing system affordable to a large number of users and also provides a general computing facility for other uses.



A Shipboard Report Origination System
Utilizing a Microcomputer

by

Joseph Glade Holyoak
Lieutenant, United States Navy
B.S., College of Southern Utah, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
DECEMBER 1976

Author Joseph G. Holyoak

Approved by: Uno R. Kodres
Thesis Advisor

Gary G. Kildall
Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

ABSTRACT

A Report Origination System (ROS) has been implemented, using an inexpensive microcomputer system, to help ease the administrative burden facing Navy shipboard managers. The system is an interactive line editing system, with optional prompting, which enables a person who is unfamiliar with the report format to respond to queries in order to edit a highly formatted report. Automatic error checking has been performed using a previous edition of the report as a basis. The system allows creation of new formatted reports. The use of a general purpose microcomputer system makes the editing system affordable to a large number of users and also provides a general computing facility for other uses.

CONTENTS

I. INTRODUCTION

The ability to process information gathered from deployed military units throughout the world and the ability to develop an accurate picture of the world status at the command and control level has been greatly enhanced with the advent of the computer. The processing time per amount of data has substantially decreased. However, the quantity and the number of required recurring reports has increased. This very trend was noted in a recent article which appeared in the Naval War College Review [1]. The author of the article noticed a 38% increase in the number of required recurring reports during the time he served as Executive Officer aboard a destroyer. This increase in reporting requirements came during a time when paperwork reduction programs were supposedly in effect.

Consider the method by which the reports have been generated in the fleet. Presently, aboard the majority of Naval vessels, report generation follows a pattern similiar to the following. To gain an overview of what information is expected, the originating officer (normally a division officer or department head) refers to previously submitted reports and to the manual which calls for the report. He then proceeds to write out, in long hand, the report content searching the various appendices of the manual for any codes

or other unique features required in the report. The entire report will be written out, even if much of the information remains unchanged from the previous report. This rough copy will then delivered to the Executive Officer for his recommendations and possible changes. After approval by the Executive Officer, the rough report will be delivered to the Commanding Officer for review. If the report has been approved by the Commanding Officer, it will be prepared in the smooth for submission. If the report is being sent out as a message, it will be delivered to Radio Central where a Radioman will type the report while concurrently making a paper tape. If the report is being sent as a letter, a Yoeman will type the report in the smooth. The smooth and rough copies will then be delivered to the originator for proof reading. If no corrections are needed, the smooth report will then be delivered to the Captain for releasing.

With the increasing administrative burden being generated by the demand for more reports together with the archaic methods by which reports have been generated, it is no wonder shipboard managers feel that their operational duties are being threatened by an administrative overload [1].

This thesis proposes the use of an inexpensive, multipurpose, microcomputer system as a tool capable of assisting the shipboard manager in meeting this challenge. The main objective of such a system is to let the system take over the responsibilities of report formatting, encoding,

and error detection, thereby reducing the time that the report originator must spend generating reports.

As a user develops a familiarity with the system execution, he must be given the option of selecting the amount of prompting information he is to receive. A user who is unfamiliar with system execution will have to be prompted as to what information he must enter.

A summary of the events leading to the proposal of using a computer prompted reporting system will be presented in Chapter II. Chapter III outlines the objectives of a Report Origination System (ROS) and discusses the hardware and software considerations of implementing such a system. A summary of recommendations for further development of ROS has been presented in Chapter IV. Some concluding remarks about ROS are contained in Chapter V.

II. BACKGROUND

For some time, concern over the growing amount of paperwork has been receiving increased attention at higher command levels. This chapter summarizes recent proposals which have resulted from this concern.

A. RECENT EFFORTS TO REDUCE REPORTING REQUIREMENTS

1. Composite reporting (COMPREP)

Composite reporting was an attempt to reduce the number of required operational reports by combining common reports into one formatted report. A formatted report is one in which the sequence and position of the individual data elements have meaning. The data in the report has usually been coded to reduce the amount of information transmission and to allow automatic processing at report receiving sites.

In 1971 a COMPREP system was designed and tested by Commander First Fleet. This system appeared to have considerable merit. However, due to limited resources, the system design had not been sufficiently developed to prove it an effective management information system. The main deficiency of the system was a coded output format which was difficult to understand. Also, there was no automated

inherent-error checking or correcting capability.

2. X/C 13 Increment I (COMPREP)

In 1975 an improved COMPREP system [2] was tested and evaluated. The improvements were:

a. to provide operator intervention in order to allow on line error correction capabilities at the report receiving site.

b. to provide formal classroom as well as hands on training to fleet personnel users and receiving site editing personnel.

c. to utilize preprinted forms and a single user's handbook in order to ease the report preparation burden.

Testing and evalutation of the proposed system was conducted by a private contractor using four units of the fleet and the necessary shore supporting activities. The main purpose of the test was to show that shipboard data collection, message formatting, approval, release and distribution to ashore receiving facilities could work effectively.

The recommendation which came out of the evaluation phase was that the developmental COMPREP sytem not be adopted, even though it met the major goals. This recommendation was made because COMPREP was not an integrated reporting system which used easily drafted, readable, and flexibly formatted messages.

3. Composite Operations Reporting System (CORS)

Following X/C 13 Increment I, the basic goals of COMPREP were re-examined. Based on the results of this re-examination the CORS effort was initiated in 1976.

The purpose of CORS was to describe alternatives of implementing a reporting system which would:

a. provide timely and accurate data to all cognizant levels of command.

b. minimize the reporting burden on the originator by integrating the requirements of four independent reporting systems (employment schedule, movement report, casualty report, and Navy force status report) into one simple, non-redundant reporting system.

c. provide significant improvements in the correctness, readability of the information, and communications system loading.

The Naval Electronics System Command (NAVELEX), with guidance from the the Office of the Chief of Naval Operations (OPNAV) CORS Steering Committee, proposed implementation alternatives to meet the objectives. Of twelve original proposals, four were selected by the CORS committee for further study.[3]

The four alternatives as they apply to the report originator were:

a. Alternative 1

This alternative would require the originator to manually draft simple, readable, formatted messages using predefined forms and decision logic tables.

b. Alternative 2

This alternative was identical to Alternative 1 except that the manually drafted messages would be replaced by a Report Origination System (ROS) which would guide the message drafter in generating error-free CORS messages via interactive computer prompting techniques.

c. Alternative 3

This proposal was an implementation of the concepts recommended from the COMPREP tests. It would involve the use of an abbreviated check list to ensure that the information required by the command chain would be provided. The report would be written in narrative or abbreviated narrative English with mimimal formatting required. The editing and formatting functions would be done by data entry clerks working interactively with the incoming messages at the receiving sites.

d. Alternative 4

This alternative was identical to Alternative 3 except that the editing and formatting functions would be done automatically at the receiving site by using a special

purpose front-end text processor.

The CORS steering committee was also asked to make a final recommendation to the Chief of Naval Operations as to which alternative would be feasible for implementation. The committee selected alternative 1 as the method to implement since it provided earliest capability. The committee did point out that aternative 2 provided the best benefits, but due to the cost of the equipment in relation to immediate benefits this cost could not be justified at this time.

It was the purpose of this thesis to demonstrate that alternative 2 can be implemented with a modest and inexpensive ($4000) microcomputer based system. This system can grow to a more complex system when its usefulness has been demonstrated. This system gives immediate feedback to the report originator so that errors can be detected and corrected before they contaminate the data base. The system generates the proper format for the reports and the recepients do not need to spend the effort to edit or format the incoming data.

III. A REPORT ORIGINATION SYSTEM (ROS)

As pointed out in Chapter II the CORS steering committee proposed an automated report origination system to ease the burden of report generation. The main objection to immediate implementation was that the cost of the equipment could not be economically justified for only report generation. This chapter outlines the design objectives and an implementation scheme of a report origination system based on a microcomputer. The system provides a low cost implementation of the reporting system and provides smaller ships with a general purpose computing facility useable for many other applications.

A. ROS DESIGN OBJECTIVES

1. Ease of Use

A shipboard report origination system must not require extensive training to operate the system. The system should be self-helping and tolerant of operator errors.

2. Adaptability

As the user becomes familiar with ROS program execution, he does not require as much prompting as an unfamiliar user. The user is given control of several levels of prompting. The user should also have the option of

displaying the report in the normal coded form, as it would be sent in, or in a decoded interpreted form.

3. Applicability

To serve as a useful tool, ROS must be applicable to a wide spectrum of reporting formats, in spite of the diversity and non-standardization of required reports.

4. Error Detection

The system must conduct error analysis throughout the execution of the program. If errors are detected, the user must have the ability to correct the errors without having previous work destroyed.

5. Life cycle costs

The system should be tailored to affordable hardware. Considerations must be given to purchase or rental costs as well as hardware maintenance costs. The system should also be expandable when its general usefulness is discovered.

B. SYSTEM OVERVIEW

1. Target Report - NAVFORSTAT

Reports submitted by fleet units take on varying structures, from strictly formatted reports with various coding schemes to reports written in the natural language. It would be extremely difficult, if not impossible, to

design and implement an easy to use system which would be applicable to all types of reports. Consequently, the Navy Force Status (NAVFORSTAT) report was chosen as a typical operational report which contains the basic structures which a report origination system must deal with. The data element was such a structure and was widely used in the ROS program.

2. Data Element

A data element within the ROS program consists of four parts:

a. Data Label

A data label is an unique recognizeable identifier for the data element.

b. Code Area

The code area is the reportable information which is pertinent to the data label.

c. Error Command Area

The error commands used to check input information at execution time are stored in the error command area.

d. Prompting Area

This area contains the questions which may be used to solicit the necessary information to generate the report.

3. Program Structure

A brief introduction to the software programs making up the Report Origination System follows:

a. ROS

This is the main program with which the user originates desired reports. The ROS program uses as input a data base (DAT) file and produces as output an updated DAT file and a message (MSG) file. Appendix A contains a user's guide for executing ROS on the Intellec-8 microcomputer system.

b. CREATE

The utility program CREATE was used to create a DAT file. It was assumed that the command requesting a report will create the pertinent DAT file and send a copy of this DAT file to all commands required to submit the report. In this manner the requesting command would be able to specify the report format, the prompting questions, and degree of error analysis. The user interested in creating a DAT file for a report which may be unique to his unit or for a report in which the DAT file doesn't exist, may do so by referring to the user's manual contained in Appendix A.

c. Line Editor

A line editing procedure, LEDIT, was implemented in the programs ROS and CREATE. This line editor allows the

user the capability of editing input information.

4. Files

a. Data Base (DAT)

A DAT file must exist for each report originated with the aid of ROS. The DAT file will either be furnished by the command requiring the report or can be created using the CREATE program.

The first record (128 bytes) of the DAT file has been reserved for the record map. The record map serves as the data element directory and has been used to randomly access data elements within the DAT file. Subsequent records of the DAT file contain the components of the data elements in a rotating pattern of: code and error command area (one record) followed by the prompting information (variable number of records). A portion of a DAT file has been expanded in Figure 1 to show the structure of the file.

b. Message (MSG)

The message file is created during the execution of the ROS program and contains the report which may be submitted. The code area portion of the data element will make up the entries of the MSG file. The carriage return and line feed characters are appended to the end of each entry to allow line by line printing of the data contained within the MSG file.

```
;----------Record Map-----------;----------------------
5 C O M D R 0 1 P E R S N 0 3 ...C O M D R  C D R / W .

----------Code Area---------------;---------------Error
T . H A T C H / 0 1 8 6 8 9 - 2 0 ;binary zeroes... A /

Commands;-------------------------------------------
A / N ¬ W H A T  I S  Y O U R  C O M M A N D I N G  O

--------------------Prompting Area------------------
F F I C E R ' S  R A N K ? / H I S  N A M E ? / H I S

---------------------------------;--Code-------- ...
L I N E A L  N U M B E R ? / ↑ ...P E R S N ...
```

A PORTION OF A DAT FILE

FIGURE 1

C. HARDWARE

The hardware system which was used to implement ROS consisted of:

1. Computer

An Intellec-8 mainframe, which was based on Intel's 8080 microcomputer, with 16K bytes of random access memory.

2. Display Terminal

A Datamedia Elite 2500 cathode ray tube (CRT) display.

3. Auxiliary Memory

A Shugart dual drive floppy disk system.

The motivation behind the selection of this equipment was availability. This system does, however, point the economic benefits of such a system. Currently the cost for a system like this is approximately $6000. Of course, the cost of a system may vary depending upon the capabilities a user desires. It was not the purpose of this thesis to do a cost analysis of available equipment, but rather to demonstrate that a Report Origination System may be implemented on a relatively inexpensive system.

D. ROS SOFTWARE

The ROS software package consists of the main program and a utility program CREATE. The ROS program was designed to make changes to the DAT file and CREATE was designed to create a DAT file if one did not exist. This section deals strictly with the software program ROS whereas CREATE will be discussed in Section E.

The ROS program consists of six modules:

1. Operating System Interface

Certain input, output and utility functions were carried out through system calls to the resident Monitor Control Program (CP/M). The detailed instructions pertaining to operation under CP/M may be found in the CP/M

Interface Guide[7]. Only the actual functions which are used within the ROS software will be discussed here.

a. Console Output

(1) Printchar - Outputs ASCII characters to the display terminal.

b. Disk Input/Output

(1) Search - Search the disk directory for a particular file.

(2) Open - Make a file ready for further operations.

(3) Close - Update the directory entry for the particular file after processing operations are completed.

(4) Diskread - Read the next record (128 bytes) from the referenced file directly to memory area specified by direct memory access (DMA).

(5) Diskwrite - Write a record from the accessed address in memory to the referenced file on the disk.

c. Utility Functions

(1) Set DMA - Set the 128 bytes buffer address at which subsequent disk input and output operations will take place.

(2) Lifthead - Lift the disk read/write head.

2. Initialize Module

The main function of the initialize module was to initialize main memory with the data elements selected by

the user to be included in the report. The submodules that carried out this function were:

a. File Operations

The proper DAT file, if it exists, is opened and a MSG file is made.

b. Selecting the Working Set

To set up the working set, the record map is read into memory and the data labels are displayed, one line at a time, on the CRT display. The user may then select the data elements which are to be included in the report. If a data element is selected, the number of records which the data element occupies on the disk is calculated and stored in the working map along with the disk location information. When all desired data labels have been selected, the user may enter "S" to stop further display of the data labels. At this point, the working map contains the pertinent disk accessing information required to process the report.

c. Initialize Memory

The information pertaining to the selected data elements is read into available memory until either memory is filled or all selected data elements have been read. If memory is filled, a logical variable will be set to true indicating that there are more data elements to be read from the disk. After the read operation is completed, pointers will be establised to allow referencing of the various data

element fields (code,error and prompting). These pointers are aligned such that each area pointer will reference common subfields. In figure 1, for example, the question contained in subfield 1 of the prompt area is the prompting information required to solicit the necessary data to be inserted in subfield 1 of the code area. Likewise, the solicited information is to be checked for errors according to the error commands contained in subfield 1 of the error area. This pointer alignment is maintained throughout the execution of the ROS program.

3. Editing with Instructions

The main function of the edit module was to solicit the necessary information from the operator which may be correctly entered into the report text. This function was carried out by the following submodules:

a. Solicit Information

The prompting question is displayed on the CRT screen. The operator enters a response at the keyboard.

b. Editing

The response is checked for errors according to the error commands pointed to by the error pointer. (The error analysis technique will be discussed in Section 5 as it applies to both editing schemes.) If an error occurs, a diagnostic warning is given and the user may reenter the corrected response. When the solicited information is

deemed correct, it is saved in a code buffer for updating the DAT file and entry into the report.

4. Editing without Instructions

In this environment, the user acts directly upon the coded information. This allows the experienced user quick access to subfields requiring changes without having to go through a prolonged question and answer period. The user uses the features of the line editor [Appendix A] in editing the coded information. Briefly, the line editor uses two buffers: the old and new buffers. Information may be passed between the buffers using the special characters defined to accomplish different functions. These special characters are defined in Table 1 of Appendix A.

a. Information Entry and Editing

A copy of the coded information is duplicated in the old buffer of the line editor. The data label is immediately moved to the new buffer where it is protected from editing by the user. The user may now duplicate any or all information from the old buffer to the new buffer or may enter new data from the keyboard. A carriage return signals the program that the editing function is complete and error analysis may begin. After the error analysis is complete, the new buffer contents (new information) is used to update the code area of the data element in the same manner as in editing without instructions.

5. Error Analysis

Although error analysis is closely connected with the editing process, it actually exists as a separate module in the program. The function of the error analysis module was to prevent contamination of the data base by entry of incorrect data. No attempt has been made to list all possible sources of errors and generate countermeasures against the occurrence of these errors but rather, to develop a system which may be expanded as experience may require. Thus the error analysis module is broken down into submodules of: command recognition and command execution.

a. Command Recognition

The error command (one alphabetic or numeric ASCII character) is compared to entries contained in the Do command (DO$CMD) procedure. When the proper match occurs, an appropriate procedure call will be made. An error command listed in the DAT file for which no execution procedures exist within the ROS program is an error introduced at the time the DAT file was created. If no match occurs during execution, a diagnostic will be given and execution will terminate. In order to maintain the pointer alignment to each subfield, a null error command (0) is used. A match on the null command simply returns program flow to the calling procedure without doing any error analysis.

b. Command Execution

This submodule consists of all error diagnostics which may be called to carry out the error analysis. It is broken down into functional units each of which is a procedure designed to check for certain error conditions. The error analysis capability may be extened by inserting new commands in the DO$CMD procedure and entering the corresponding procedures to check the data for the occurrence of the error. A difference between the error analysis techniques applied to the input and editing modes should be pointed out. In the instructional mode, error analysis is applied to one subfield for each call to the error module, whereas in the non-instructional mode all subfields of the code area are analyzed with one call to the error analysis module.

6. Output

After all the data elements within memory have been processed, an updated version of the coded information will exist in memory. This coded information serves as the basis for updating the DAT file and actual creation of the report. Since the coded portion is the only data to have changed during execution of the ROS program, it is the only information required to be written to the DAT file. Utilizing the working map, which gives the record number within the DAT file where the code record must be written, the proper access may be made and the updated record written to the

proper DAT file. A copy of the coded information with appended carriage return and line feed characters is saved in a temporary buffer until a full record has been accumulated. This record is then written to the MSG file.

After completion of the output phase, if more data elements exist to be processed, the elements will be read into memory and the pointers will be reset. The editing process will continue until all data elements have been processed. After all data elements have been processed, the DAT file and the MSG file will be closed.

E. CREATE

The utility program CREATE allows the user the ability to create a data base file with which a particular report may be generated. The program was designed with the thought that persons familiar with computers at the command requesting the report, would create the appropriate DAT file and forward a copy to all reporting commands. Thus the program execution is somewhat more cryptic and "magical". However, this is not meant to discourage the shipboard manager from using the system. A user's guide is provided [Appendix A] and the system execution may be mastered in a short time.

The DAT file consists of a directory element, the record map, and repetitive entries of code, error commands and prompting information. The entries are organized into records, each record being 128 bytes. The record map occu-

pies the first record of the file. The code area and the error commands for each data element share one record. The code area is located at the first of the record whereas the error commands are located at the end of the record. Any unused space between the code and error areas contains binary zeroes. The prompting area may occupy more than one record. Since the code area begins on a record boundary and together with the error commands will take up no more than one record, the prompt area will always begin on a record boundary. This structure allows random access to the record where each data element begins within the file. To effect this random access, the data element identifier, the data label, is stored within the record map, along with the necessary disk locating information. It then becomes a matter of reading the record map, selecting the particular data label and setting up to read the information of the desired data element.

The CREATE program consists of the following modules:

1. Initialize

The function of the initialize module was to make a DAT file, if one did not already exist. If a DAT file already exists, a diagonstic warning will be given and program execution will cease. The user may then remove the DAT file, if no longer desired, or may use it as input for the ROS program to generate the desired report.

Assuming a previous DAT file does not exist, a DAT file will be made. Specific pointers to available memory (memory between the CREATE program and the resident operating system) will be established with the first 128 bytes of available memory being reserved for the record map. All data to be entered in the DAT file will be entered in a sequential manner between the record map and the operating system.

2. Input

After the DAT file has been opened and the pointers set, the user is free to enter data into the DAT file. Since the DAT file is constructed in memory in a sequential manner, the entries must be in the order of code, error commands, and prompting information. The features of the Line Editor [Appendix A] are used to edit the entered text. When the user is satisfied with the particualar entry, a special character defined in Table of Appendix A is entered and the entered data is stored in the DAT file memory area.

Editing and error analysis is left up to the user. The features of the Line Editor allow ample capabilities to make corrections to entered data. Once the special character, denoting the type of entry, is given, the input data is stored and the user no longer has access to it.

3. Finish

If available memory is filled before all data elements have been entered, the existing memory image of the DAT file will be written to the disk and the pointers reset. After all data elements have been entered, a back slant may be entered to indicate end of file. At this time the memory image of the DAT file is written to the disk file and the DAT file is closed.

IV. RECOMMENDATIONS

The implementation of a Report Origination System (ROS) designed to help ease the administrative burden facing the shipboard manager has been discussed. This has been the first known attempt of implementing such a system and subsequently some arbitary design decisions were made that have become apparent weak points. In addition, complete development of the system has been curtailed due to time constraints. A follow-on thesis aimed at further development is planned and therefore this chapter summarizes some of the areas that could be further developed.

A. DISK DIRECTORY (RECORDMAP)

Currently the record map is limited to 128 bytes which is insufficient space for storage of long or a large number of data labels. Admittedly, this method was an ad hoc procedure and a more efficient method is required. One possible method may be to use a hash coding scheme to reduce the size of the entries.

B. FIXED DATA LABEL LENGTH

Presently all data labels must be of equal length throughout the DAT file. The CREATE user has the responsibility for ensuring that the length of all data labels will be the same. Although this restriction simplifies program

coding and decreases memory usage (important considerations in microcomputer usage) it was considered much too restrictive for general acceptance. A prime consideration for fixed data lengths was for ease of insertion in the record map. If the method of maintaining the disk directory were changed, the emphasis on using a fixed data label length would lessen. Another use of the data label must be considered before removing the restriction. In setting up the editing buffers for subsequent edit operations, the data labels are moved to the appropriate buffers dependent upon the fixed data label size. Thus, to continue to use this method of editing, an alternative approach to moving variable length data labels must be sought. One method to handle the variable length problem might be to mark the end of the data label with a special character. Characters could then be transferred between buffers until this special character is encountered.

C. LACK OF MNEMONICS USE IN THE LINE EDITOR

The line editor commands have no mnemonic value. This resulted as a trade-off for programming efficiency. Rather than use a series of comparing statements to identify the input character, it was decided to use a sequential group of characters and index to the correct procedure call by a case statement. A table which translates mnemonic code to the code used in this program could be added to make the line editor similiar to other editing systems.

D. BACK UP FILES

No provisions have been provided to set up a back up file. This could be accomplished by copying the DAT file to another file (BAK) before the editing of the DAT file begins.

E. OTHER SUPPORTIVE SOFTWARE

1. Headings

Software should be developed to assign heading information (date time group, addressees, classification, etc.). This could be patterned after current techniques (Addressal Indicator Group) in use in the Navy.

2. Optional Display List

As noted in the objectives of ROS, a user should have the option of displaying reported information in coded or interpreted form. This option has not been incorporated, but should be relatively easy to do. By utilizing a pointer alignment scheme, such as previously discussed, the prompting question together with the subsequent response could be displayed for each data element edited during the session.

V. CONCLUSIONS

ROS provided an interactive feed back loop of computer generated prompting instructions, user responses and error analysis. This system permits error detection and correction to be carried out by individuals who generate the report and were therefore most likely to recognize serious errors in report content. Errors in format were unlikely because format design would be generated in the computer. In the present method, the recipient of the report was asked to perform error detection. The recipient was generally able to detect errors in format only and was unable to correct the errors in content without additional information from the report originator.

ROS was an easy to use system with built-in flexibility. As the user becomes familiar with the report structure, he may choose to limit the amount of prompting instructions he will receive.

Although ROS was designed using the NAVFORSTAT report as a pattern, it will be applicable to any report which uses line by line formatting. Since most Navy reports fit into this category, ROS should be applicable to a wide spectrum of reports.

The secondary benefits of having an expandable, multipurpose computing facility aboard naval vessels can not be ignored. As shipboard users become familiar with the computing potential available, the base of application development will be increased by orders of magnitude. Many application programs have already been developed at the Naval Postgraduate School and are ready for further testing.

The Report Origination System (ROS) has been implemented on an inexpensive microcomputer system and shown to be an effective tool in helping the shipboard manager meet the challenge of required recurring reports.

APPENDIX A: User's Guide

ROS Program

1. Introduction

The purpose of this User's Guide is to assist the user in generating required recurring reports through the use of the Report Origination System (ROS). The details of the program logic may be found in the body of the thesis [Chapter III]. This guide is a step by step overview of what the user should expect while executing ROS. This guide describes implementation on an Intellec-8 microcomputer system which uses the resident operating system.

2. Initialization

The program ROS is initiated by typing

ROS <filename> <cr>

<cr> stands for carriage return. The <filename> must be the name of a data base (DAT) file which exists on the diskette. It is assumed that the command requesting a particular report will supply the user with a diskette containing the pertinent DAT file. However, if the user is setting up a file to handle a recurring report unique to his command or if he has not been furnished a DAT file, he may create the DAT file by referencing Section[7] of this guide.

3. Execution

If the proper DAT file exists, it will be opened and the data labels contained within the file will be displayed on the CRT screen. The user may now select (by entering a "Y" or "N" under the data label) the data elements which are to be included in the report. At this point the user will be queried whether or not he wishes to be prompted. A user who is unfamiliar with computer line editing procedures should select to be prompted. After the user becomes familiar with the line editor [Section[8]], he may find it easier to generate a report by using the limited prompting mode. A positive response ("Y") to this query will allow the user to edit the coded area of the report directly using the features of the line editor [Section[8]]. If a negative response ("N") is given, the user will be prompted by appropriate questions to solicit the necessary information for coding into the report.

4. Limited Instruction Mode

If the user selects to edit the coded information directly, the coded information last submitted will be entered into the old buffer and the data label will be entered into the new buffer [Section[8]]. Now, using the features of the line editor, the user may duplicate unchanged information to the new buffer or enter new information from the keyboard. When the editing process for each data element is completed, a carriage return may be entered to signal the ROS program to check for errors. If an error

has occurred, a diagnostic will be given and the old buffer contents (old information) as well as the new buffer contents (new information) up to the point of the error will be displayed. The user may now make the appropriate corrections, duplicate the remaining information and enter a <cr>. When the coded information is correct it will be saved in the coded area for updating the DAT file. Execution in this manner continues for each of the data elements selected at the beginning of the session.

5. Instructional Mode

During the instructional mode of operation, questions will be written on the CRT screen. The user will key responses to these question from the keyboard. As inputs are received by the ROS program, error analysis will be conducted. If an error occurs, a diagnostic will be given and the user will have an opportunity to give another response. Correct responses are retained in the new buffer [Section[8]] until all inputs for the working data element have been gathered. After a data element update is complete the information will be stored in the code area for updating the DAT file. Execution continues in this manner until all previously selected data elements have been processed.

6. End Execution

When all data elements have been processed the ROS program will write the compiled report to the diskette. The

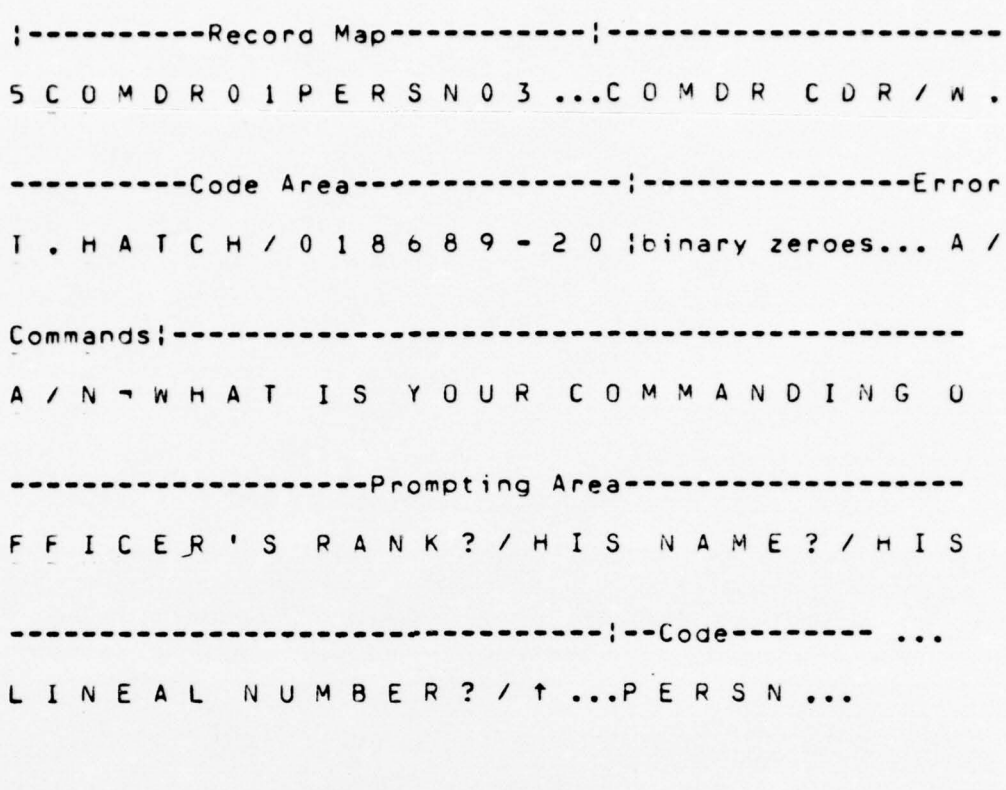
report may be referenced under the file <filename>.MSG.

CREATE

1. Introduction

The utility program CREATE was designed and written to allow the user the ability of building a data base (DAT file). The DAT file contains information specified by the most recent report as well as the prompting questions the operator is asked to respond to in order to collect the necessary information.

A portion of a DAT file has been expanded in Figure 2 to show the structure of this file. The first 128 bytes of the DAT file are reserved for the record map. The record map is a summary of the data labels contained within the DAT file as well as the disk storage location of the first record of each particular data element. Disk locations are maintained by record extent and record number within the file.



A PORTION OF A DAT FILE

FIGURE 2

Information pertinent to the report is contained within the data element. A data element consists of a data label, a code area, error check commands and a prompting area. The data label is a unique recognizable identifier for each line of the proposed report. For example in the Navy Force Status (NAVFORSTAT) report the data label, COMDR, is used to reference information pertaining to the Commanding Officer of the particular reporting activity. The code area contains the reportable information pertaining to the data label. Error check commands are used to check input information at execution time. The questions to be asked to solicit the information needed to generate the report are

contained in the prompting area. With the exception of the record map and the data label PERSN, the example given in Figure 2 represents one data element. Notice that data elements having more than one subfield description are separated by a sub-field delimiter "/". This delimiter is used in the error analysis routines to set up fixed or variable length subfields.

Each subfield of the error analysis command corresponds to the same subfield of the coded portion. For example commands listed in Figure 2 would indicate a check of subfields 1 and 2 of the referenced data label for alphabetic characters only and to check subfield 3 for numerics only.

2. Initiation

The CREATE program is invoked by typing

CREATE <filename> <cr>

The <filename> must be a unique mnemonic (8 or less characters) for the particular report to be generated. For example,

CREATE NFS <cr>

may be used to create a data base for a NAVFORSTAT report. At this point execution begins and the user is asked to specify the length of the data labels to be used within this particular report. The user should note that since the data labels are duplicated in the record map, the lengths should be minimal and in no case should they exceed a length of 9.

3. Execution

The operator is now free to enter text from the console using the features of the line editor [Section [9]] and special characters required to specify the type of input. These special characters are summarized in Table 1. Keep in mind CREATE expects text input in the order of code, error check commands, and questions.

Text is stored in memory until an end of file is encountered or the memory region is filled. At this time the memory image of the DAT file is written to the disk and memory pointers reset or the system reboots in the case of end of file. The end of file is signalled by entering a back slash.

CHARACTER	DESCRIPTION	ASCII	FUNCTION
¦	Bar	7CH	end of code area
¬	Tilde	7EH	end of error commands
↑	Up-arrow	5EH	end of prompt area
\	Back slant	5CH	end of file

TABLE 1

SPECIAL CHARACTER MEANINGS IN PROGRAM CREATE

1. Introduction

ROS and CREATE use the features of a line editor which is incorporated in the programs as the procedure LEDIT and is called when the console is to be read for input.

The procedure LEDIT uses two 90 bytes buffers, the old buffer and the new buffer. As the names imply, the old buffer contains information entered at the last console read operation. The new buffer contains information which is currently being read. The information in the buffers may be transferred between the buffers by entering control characters which are summarized in Table 2.

CONTROL CHARACTER	DEFINITION
A	Acts as a backspace and rub-out command on the new line only. (same as rub-out on many terminals).
B	Replace the old line with the contents of the new line, empties the new line.
C	Copy one character from the old buffer to the new buffer.
D	Copy the remaining characters from the old line to the new line echoing each character. Then terminate the edit.
E	Toggle the insert mode. Begin insert prints "<", end inserts prints ">". Position of the old

pointer does not change during insert.

F Delete the new line without updating the old line. Terminate the edit.

G Display contents of the old and new buffers with control characters interpreted (e.g. "↑I" for TAB, "↑G" for BELL, etc.).

H Copy remaining characters from the old line to the new line, echoing each. Do not terminate the edit.

I Tab. A tab stop is defined every four characters. Same as TAB on many terminals.

J Line feed. Terminate the edit.

K (not used)

L Copy remaining characters of old line to new line without echoing. Do not terminate the edit.

M Carriage return Terminate the edit.

N Backspace old buffer and new buffer one space.

O Copy characters fom the current position of the old pointer to the next character typed.

P Delete characters from the current position of the old pointer to the next occurence of the next character typed. Echoes a "%" for each character deleted.

Q Delete the new line and reset the old pointer to the start of the old buffer.

R Display the remaining contents of the old buffer and all of the new buffer.

S Delete one character from the old buffer. Echo a

"%" for the deleted character.

T Only used in CREATE to transmit information in the new buffer to storage in memory. Used when input from the console exceeds one crt line.

U Copy characters from the old buffer to the new buffer up to the next TAB character.

V Escape character. Turns off any special meaning of character which follows. Enters the character into the new buffer and echoes the characters (e.g. "CTLvCTLm" will echo ↑m).

W (not used)

X Deletes characters from current position of old buffer through next character typed.

Y Copy the remaining characters from the old buffer to the new buffer echoing each, replace the old buffer with the new buffer. Do not terminate the edit.

Z Copy characters from the old buffer to the new buffers through the next occurence of the next character typed.

NOTE: The control character is entered by depressing the CTRL key and then simultaneously depressing the the desired function key.

TABLE 2

LINE EDITOR FEATURES

A Sample Session

This is an example of how a portion of a Naval Force Status (NAVFORSTAT) data base file may be created and then subsequently updated to generate a report. The brackets < > are used to indicate keyboard entries. Comments, as they apply, are enclosed within /* comment */. Text produced during program execution will be as it would appear on the screen.

A. CREATE Execution

```
A> /* system is ready to start */
<CREATE NFS> <cr>
LENGTH OF DATA LABELS TO BE USED?
<5> <cr>
EXPECTING CODE INFO /* prompt message */
<COMDR CDR/J. P. JONES/000111-10¦> <cr> /* the bar (¦) will
not be echoed */
EXPECTING ERROR COMMANDS
<A/A/N¬> <cr>  /* ¬ will not be echoed */
EXPECTING PROMPT INFO
<WHAT RANK IS YOUR COMMANDING OFFICER?/HIS NAME?/HIS LINEAL
NUMBER?↑> <cr> /* ↑ will not be echoed */
EXPECTING CODE INFO
<PERSN NE/0236/0230/0210¦> <cr>
EXPECTING ERROR COMMANDS
```

```
<A/N/N/N¬> <cr>
EXPECTING PROMPT INFO
<TYPE    OF    PERSONNEL?/STRUCTURED    STRENGTH?/AUTHORIZED
STRENGTH?/ASSIGNED  STRENGTH?↑> <cr> /* ↑ will not be echoed
*/
EXPECTING CODE INFO
< > /* that is enough for now */
A>
/* A file (NFS.DAT) now exists */
```

B. ROS Execution

To send out personnel information, a user simply uses ROS.

```
A> /* system is ready */
<ROS NFS> <cr>
SELECT DATA ELEMENTS YOU DESIRE TO WORK WITH
COMDR PERSN
<N> <Y> <cr>
ARE DATA LABELS TO BE INCLUDED IN THE REPORT
<Y> <cr>
DO YOU WISH TO BE PROMPTED
<Y> <cr>
TYPE OF PERSONNEL?
<NE> <cr>
STRUCTURED STRENGTH?
<0256> <cr>
AUTHORIZED STRENGTH?
```

```
<0230> <cr>
ASSIGNED STRENGTH?
<0218> <cr>  /* gained 8 people */
A> /* only one data element was selected */
/* without prompting */
/* execution is the same to question: */
DO YOU WISH TO BE PROMPTED
<N> <cr>
PERSN NE/0236/0230/0210
PERSN <ctlZ> <1> <cr> by a 1 (not echoed) */
PERSN NE/0236/0230/021<8> <cr>  /* no new line created, just
filled in current line */
PERSN NE/0236/0230/0218
A>
/* in either case, message looks like: */
PERSN NE/0236/0230/0218
```

```
/* ************************************************************

   A REPORT ORIGINATION SYSTEM DESIGNED FOR SHIPBOARD
USE IN THE GENERATION OF REQUIRED RECURRING REPORTS.
THE SYSTEM USES AS INPUT A DATA BASE (DAT) FILE AND
PRODUCES AS OUTPUT A MESSAGE (MSG) FILE.  THE
SOFTWARE SYSTEM CONSISTS OF TWO PROGRAMS: ROS AND
CREATE.  CREATE IS USED TO CREATE A DAT FILE AND ROS
IS USED TO UPDATE THE DAT FILE AND CREATE A MESSAGE.

   THE ROS PROGRAM IS MADE UP ON THE FOLLOWING MODULES:

          1.  OPERATING SYSTEM INTERFACE
          2.  INITIALIZE
          3.  EDITING
          4.  ERROR
          5.  OUTPUT

   THE CREATE PROGRAM IS MADE UP OF THE FOLLOWING
MODULES:

          1.  INITIALIZE
          2.  INPUT-EDITING
          3.  FINISH

   BOTH PROGRAMS WERE DESIGNED FOR EXECUTION ON THE
INTELLEC-8 MICROCOMPUTER SYSTEM, WITH CROSS COMPILATION
BEING DONE ON AN IBM 360/65.

************************************************************ */

100H:  /* PROGRAM TO BE LOADED INTO MEMORY STARTING HERE */

  /* ************************************************************

        OPERATING SYSTEM INTERFACE DECLARATIONS.

  ************************************************************ */

DECLARE
    LIT         LITERALLY       'LITERALLY',
    BOOT        LIT     '0',
    ENTRY       LIT     '0005H',  /* ENTRY POINT TO OS */
    TRUE        LIT     '1',
    FALSE       LIT     '0',
    FOREVER     LIT     'WHILE TRUE',
    CR          LIT     '0DH',
    LF          LIT     '0AH',
    CTI         LIT     '0',
    CTS         LIT     '1',
    DCNT              BYTE,
    BDOSA             ADDRESS INITIAL (0006H),
    SBDOS    BASED BDOSA   ADDRESS;

  /* ************************************************************

    INITIALIZE DECLARATIONS

  ************************************************************ */

DECLARE
    PROMPT  BYTE INITIAL(FALSE),
    RM          ADDRESS INITIAL(80H),
    RMPTR       BASED RM   BYTE,
    DL$LEN      BYTE,
    NUM$REC     BYTE,
    WORK$MAP            ADDRESS,
    WMPTR       BASED WORK$MAP  BYTE,
    EXT$RN      BASED WORK$MAP  ADDRESS,
    NUM$ELEMENTS            BYTE,
    E$R$A                   ADDRESS,
    ER        BASED E$R$A   BYTE,
```

```
    NR$READ     BYTE,
    NR      BYTE,
    SAVE$EXT        BYTE,
    SAVE$RN         BYTE,
    DAT$AREA        ADDRESS,
    DAT BASED DAT$AREA   BYTE,
    BASE$DAT$AREA    ADDRESS,
    I$DAT$AREA      ADDRESS,
    CODE$A          ADDRESS,
    CODE BASED CODE$A    BYTE,
    B$CODE$A        ADDRESS,
    TOP$MEM         ADDRESS,
    MSG$AREA        ADDRESS INITIAL (80H),
    MSG  BASED MSG$AREA BYTE,
    MORE     BYTE INITIAL (FALSE),
    HOLD$WM     ADDRESS;

  /* ********************************************************

                      EDITING DECLARATIONS

  ******************************************************** */

DECLARE
    BUFFER (180)            BYTE,
    SIZE$NBUF LIT     '90',
    NEW$BUF                 ADDRESS,
    NBUF   BASED NEW$BUF  BYTE,
    NPTR                    BYTE,
    OLD$BUF                 ADDRESS,
    OBUF   BASED OLD$BUF  BYTE,
    OPTR                    BYTE,
    NB          ADDRESS,
    TN BASED NB BYTE,
    OB          ADDRESS,
    INSERT     BYTE INITIAL (FALSE),
    PERCENT    LIT          '25H',
    BS         LIT          '08H',    /* BACKSPACE  */
    BELL       LIT          '07H',
    TAB        LIT          '09H',
    EOP        LIT          '5EH', /* UP-ARROW; END OF PROMPT */
    EOC        LIT          '7CH', /*BAR; END OF CODE */
    ERR        LIT          '7EH', /* TILDE; END OF ERROR */
    CTLZ       LIT          '1AH',
    RUBOUT     LIT          '7FH',
    END$FILE   LIT          '5CH', /*BACK SLANT */
    CHAR     BYTE,
    PROMPT$AREA     ADDRESS;

  /* ********************************************************

                          ERROR DECLARATIONS

  ******************************************************** */

DECLARE
    ERRA     ADDRESS,
    ECMD     BASED ERRA BYTE,
    BERRA    ADDRESS,
    WARN     BYTE INITIAL (FALSE);

  /* ********************************************************

                          OUTPUT DECLARATIONS

  ******************************************************** */

DECLARE
    DAT$FCB     ADDRESS INITIAL (5CH),
    DFCB   BASED DAT$FCB  BYTE,
    MSG$FCB(33)      BYTE,
    PRINT$LABEL    BYTE INITIAL (FALSE);
```

```
/* ************************************************************

                    OPERATING SYSTEM INTERFACE MODULE

       FUNCTION:  SERVES AS AN INTERFACE TO THE RESIDENT
    OPERATING SYSTEM.  IT ALLOWS INPUT/OUTPUT OPERATIONS
    TO BE HANDLED BY SYSTEM CALLS.

  ************************************************************ */

CRTIN: PROCEDURE BYTE;
    DO WHILE INPUT(CTS);
    END;
    RETURN NOT INPUT(CTI) AND 07FH;
    END CRTIN;

READC: PROCEDURE BYTE;
    DECLARE C BYTE;
    IF (C:=CRTIN) >= 110$0001B  /* LOWER CASE A */
       AND C <= 0111$1010B /* LOWERCASE Z */ THEN
       C = C AND 101$1111B;  /* BECOMES UPPER CASE  */
    RETURN C;
    END READC;

MON1: PROCEDURE (FUNC, INFO);
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON1;

MON2: PROCEDURE (FUNC,INFO) BYTE;
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON2;

PRINTCHAR: PROCEDURE (B);
    DECLARE B BYTE;
    CALL MON1(2,B);
    END PRINTCHAR;

PRINTCHARI: PROCEDURE(C);
    DECLARE C BYTE;
    IF (C AND 0110$0000B) = 0 /* CONTROL CHAR */ THEN
       DO;
         CALL PRINTCHAR(EOP);
         CALL PRINTCHAR(C OR 40H);
       END;
    ELSE
    CALL PRINTCHAR(C);
    END PRINTCHARI;

CRLF: PROCEDURE;
   CALL PRINTCHAR(CR);
   CALL PRINTCHAR(LF);
    END CRLF;

PRINT: PROCEDURE (A);
    DECLARE A ADDRESS;
    CALL MON1(9,A);
    CALL CRLF;
    END PRINT;

SET$DMA: PROCEDURE(A);
    DECLARE A ADDRESS;
    CALL MON1(26,A);
    END SET$DMA;

DISKREAD: PROCEDURE(A) BYTE;
    DECLARE A ADDRESS;
    RETURN MON2(20,A);
    END DISKREAD;
```

```
DISKWRITE: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(21,A);
     END DISKWRITE;

OPEN: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(15,A);
     END OPEN;

CLOSE: PROCEDURE(A) BYTE;
     DECLARE A ADDRESS;
     RETURN MON2(16,A);
     END CLOSE;

SEARCH: PROCEDURE (FCB) BYTE;
     DECLARE FCB ADDRESS;
     RETURN MON2(17,FCB);
     END SEARCH;

MAKE: PROCEDURE (FCB) BYTE;
     DECLARE FCB ADDRESS;
     RETURN MON2(22,FCB);
     END MAKE;

LIFTHEAD: PROCEDURE;
     CALL MON1(12,0);
     END LIFTHEAD;

MOVE: PROCEDURE (SOURCE,DEST,N);
     DECLARE (SOURCE,DEST) ADDRESS,
       (S BASED SOURCE, D BASED DEST, N ) BYTE;
     DO WHILE (N:=N-1) <> 255;
       D=S; SOURCE=SOURCE+1; DEST=DEST+1;
     END;
     END MOVE;

ERROR: PROCEDURE(I);
     DECLARE I BYTE;
          DO CASE I;
          /* CASE 0 OVERWRITING ERROR CODES */
          CALL PRINT(.'OVERWRITING ERROR CODES  $');
          CALL PRINT(.'DISK READ ERROR $');
          CALL PRINT(.'ERROR COMMAND NOT DEFINED  $');
          CALL PRINT(.'A MESSAGE FILE EXISTS    $');
          CALL PRINT(.'DISK WRITE ERROR         $');
          CALL PRINT(.'OUT OF DIRECTORY SPACE   $');
          CALL PRINT(.'DAT FILE NOT PRESENT$');
          CALL PRINT(.'MSG FILE NOT PRESENT$');
     END;
     GO TO BOOT;
     END ERROR;

  /* ************************************************************

                    INITIALIZE MODULE

       FUNCTIONS: TO OPEN THE APPROPRIATE DAT FILE, MAKE
     A MESSAGE FILE AND ALLOW THE USER TO SELECT A SET OF
     DATA ELEMENTS TO WORK WITH.  IT THEN INITIALIZES
     MEMCRY WITH THE SELECTED DATA ELEMENTS.

  ************************************************************ */


MAKE$MSG$FILE: PROCEDURE;
     CALL MOVE(.'MSG',.MSG$FCB + 9,3);
     MSG$FCB,MSG$FCB(12),MSG$FCB(32) = 0;
     IF SEARCH(.MSG$FCB) <> 255 THEN
          CALL ERROR(3);
     IF MAKE(.MSG$FCB) = 255 THEN
```

```
        CALL ERROR(5);
    IF OPEN(.MSG$FCB) = 255 THEN
        CALL ERROR(7);
    END MAKE$MSG$FILE;

INIT: PROCEDURE;
    CALL MOVE(5DH,.MSG$FCB+1,8);
    CALL MAKE$MSG$FILE;
    CALL MOVE(.'DAT',DAT$FCB+9,3);
    DFCB(12),DFCB(32)=0;
    IF OPEN(DAT$FCB)=255 THEN
        CALL ERROR(6);
    IF (DCNT:=DISKREAD(DAT$FCB)) <> 0 THEN
        CALL ERROR(1);
    CALL LIFTHEAD;
    END INIT;


INC$RM: PROCEDURE;
    RM = RM + 1;
    END INC$RM;

INC$WM: PROCEDURE;
    WORK$MAP = WORK$MAP + 1;
    END INC$WM;

PRINT$DATA$E: PROCEDURE;
    DECLARE (I,J) BYTE;
    DO I = 1 TO NUM$ELEMENTS;
       IF RMPTR = EOC THEN /* END OF RECORD MAP */
            DO; RM=101H; NUMELEMENTS=I-1; RETURN; END;
       DO J = 1 TO DL$LEN;
         CALL PRINTCHAR(RMPTR);
         CALL INC$RM;
       END;
       CALL PRINTCHAR(' ');
       RM = RM + 2;
    END;
    END PRINT$DATA$E;

SAVE$EX$RN$NR: PROCEDURE;
    WMPTR = ER; /* EXTENT */
    CALL INC$WM;
    WMPTR = ER(1); /* RN */
    CALL INC$WM;
    WMPTR = ER(DL$LEN + 3) - ER(1); /* NUMBER OF RECORDS */
    CALL INC$WM;
    END SAVE$EX$RN$NR;

PRINTSPACE: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO DL$LEN;
        CALL PRINTCHAR(' ');
    END;
    END PRINTSPACE;

CHK$RESPONSE: PROCEDURE;
    DECLARE (I,C) BYTE;
    DO I = 1 TO NUM$ELEMENTS;
        CALL PRINTCHAR(C:=READC);
        IF C = 'Y' THEN
            CALL SAVE$EX$RN$NR;
        ELSE
        IF C = 'S' THEN
            DO; RM=101H; RETURN; END;
   E$R$A = E$R$A + DL$LEN + 2;
        CALL PRINTSPACE;
    END;
    END CHK$RESPONSE;

SEL$WE: PROCEDURE;
    DL$LEN = RMPTR;
```

```
     CALL INC$RM;
     WORK$MAP = .MEMORY;
     NUM$ELEMENTS=11;
     CALL PRINT(.'SELECT DATA ELEMENTS TO WORK WITH$');
     E$R$A = RM + DL$LEN;
     DO WHILE RM < 100H;
          CALL PRINTDATA$E;
          CALL CRLF;
          CALL CHK$RESPONSE;
     END;
     END SEL$WE;


SET$MEM: PROCEDURE;
     DAT$AREA,BASE$DAT$AREA = WORK$MAP;
     TOP$MEM = SBDOS - 1;
     WORK$MAP = .MEMORY;
     DFCB(32) = WMPTR(1); /* RN TO START READ */
     NR = WMPTR(2);
     NR$READ = 0;
     END SET$MEM;

OPEN$EXT: PROCEDURE;
     DFCB(12) = WMPTR;
     IF OPEN(DAT$FCB) = 255 THEN
          CALL ERROR(1);
     END OPEN$EXT;

READ$D$REC: PROCEDURE;
     IF DFCB(12) <> WMPTR THEN
     CALL OPEN$EXT;
     CALL SET$DMA(DAT$AREA);
     IF (DCNT:=DISKREAD(DAT$FCB)) <> 0 THEN
          CALL ERROR(1);
     NR$READ = NR$READ + 1;
     DAT$AREA = DAT$AREA + 128;
      CALL SET$DMA (80H);
     END READ$D$REC;

READ$DAT: PROCEDURE;
     DO WHILE DAT$AREA+128 < TOP$MEM;
        IF NR$READ = NR THEN
          IF (WORK$MAP:=WORK$MAP+3) >= BASE$DAT$AREA-1 THEN
             DO; /* FINISHED */ MORE = FALSE; RETURN; END;
          ELSE
             DO; DFCB(32) = WMPTR(1); NR$READ = 0;
                 NR = WMPTR(2); END;
        CALL READ$D$REC;
     END;
     MORE = TRUE;
     SAVE$EXT = DFCB(12);
     SAVE$RN = DFCB(32);
     HOLD$WM = WORK$MAP;
     END READ$DAT;

READ$MORE: PROCEDURE;
     DECLARE HOLD ADDRESS;
     HOLD,WORK$MAP = HOLD$WM;
     DFCB(12) = SAVE$EXT;
     DFCB(32) = SAVE$RN;
     DAT$AREA = BASE$DAT$AREA;
     CALL READ$DAT;
     WORK$MAP = HOLD;
     T$DAT$AREA = DAT$AREA;
     DAT$AREA = BASE$DAT$AREA;
     END READ$MORE;

  /* ************************************************************

                              EDITING MODULE

        FUNCTION:  TO ALLOW ENTRY OF DATA AND EDITING OF
```

```
    ENTERED DATA BY USE OF LINE EDITING FUNCTIONS.  THE
    USER MAY SELECT TO ENTER DATA DIRECTLY INTO THE
    CODED AREA OR BE PROMPTED AS TO WHAT INFORMATION IS
    REQUIRED.

  ********************************************************* */

              /*  PROCEDURES OF THE LINE EDITOR  */

BACK$UP: PROCEDURE;
    IF NPTR > 0 THEN
       DO;
         NPTR = NPTR - 1;
         CALL PRINTCHAR(BS);
         CALL PRINTCHAR(' ');
         CALL PRINTCHAR(BS);
       END;
    ELSE
       CALL PRINTCHAR(BELL);
    END BACK$UP;

MOVE$TO$OLD: PROCEDURE;
    CALL MOVE(NEW$BUF+1,OLD$BUF+1,(OBUF:=NPTR));
    OPTR = 0; NPTR = 0;
    END MOVE$TO$OLD;

OLD$TO$NEW: PROCEDURE;
    NBUF(NPTR:=NPTR+1) = OBUF(OPTR:=OPTR+1);
    END OLD$TO$NEW;

ECHO$ON: PROCEDURE;
   CALL PRINTCHAR(NBUF(NPTR:=NPTR+1):=(OBUF(OPTR:=OPTR+1)));
    END ECHO$ON;

COPY$ONE: PROCEDURE;
    IF OPTR <= OBUF THEN
       CALL ECHO$ON;
    ELSE CALL PRINTCHAR(BELL);
    END COPY$ONE;

P$MOVE$ON: PROCEDURE; /* PARTIAL MOVE OLD TO NEW */
    DO WHILE OPTR < OBUF;
    CALL ECHO$ON;
    END;
    END P$MOVE$ON;

ENTER: PROCEDURE;
    IF INSERT THEN
       CALL PRINTCHAR('>');
    ELSE
       CALL PRINTCHAR('<');
    INSERT = NOT(INSERT);
    END ENTER;

PRINT$OLD: PROCEDURE;
    DECLARE I BYTE;
    DO I= 1 TO OBUF;
    CALL PRINTCHARI(OBUF(I));
    END;
    CALL CRLF;
    END PRINT$OLD;

PRINT$NEW: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO NPTR;
    CALL PRINTCHARI(NBUF(I));
    END;
    END PRINT$NEW;

PRINT$BOTH: PROCEDURE;
    CALL PRINT$OLD;
```

```
    CALL PRINT$NEW;
    END PRINT$BOTH;

COPY$RM$O$N: PROCEDURE;
  /* COPIES REMAINING CHARACTERS FOR OLD TO NEW BUFFERS */

    DO WHILE OPTR <= OBUF;
       CALL OLD$TO$NEW;
    END;
    CALL PRINTCHAR('+');  /* INDICATES WHEN DONE  */
    END COPY$RM$O$N;

BS$O$N: PROCEDURE;
    /* BACKSPACE OLD PTR AND NEW PTR 1 CHAR  */
    IF (OPTR > 0) AND (NPTR > 0) THEN
       DO;
        OPTR = OPTR - 1;
        NPTR = NPTR - 1;
         OBUF = OBUF - 1;
       END;
    ELSE
       CALL PRINTCHAR(BELL);
    END BS$O$N;

COPY$ON: PROCEDURE (C);
    DECLARE (C,I) BYTE;
    I=OPTR;
    DO WHILE OBUF(I:=I+1) <> C;
       IF I > OBUF THEN /* NO MATCH */
         DO;
           CALL PRINTCHAR(BELL);
           RETURN;
         END;
    END; /* DO WHILE */
    DO WHILE OPTR < I;
       CALL ECHO$ON;
    END;
    END COPY$CN;

DELETE: PROCEDURE(ECHO);
    DECLARE (I,J,P1,CHAR1,ECHO) BYTE;
    P1=OPTR;
   CHAR1 = READC;
    DO WHILE (OBUF(P1:=P1+1) <> CHAR1);
       IF P1 > OBUF THEN /* NO MATCH */
         DO;
            CALL PRINTCHAR(BELL);
            RETURN;
         END;
    END;  /* DO WHILE  */
    IF ECHO THEN
        DO I = OPTR+1 TO P1;
             CALL PRINTCHAR(PERCENT);
        END;

     /*  NOW CONDENSE THE BUFFER   */
    J=OPTR;
    I=P1;
    DO WHILE I <= OBUF;
       OBUF(J:=J+1) = OBUF(I:=I+1);
    END;
    OBUF = OBUF - (P1-OPTR+1);
    END DELETE;

DEL$N: PROCEDURE;
   OPTR,NPTR = 0;
    OBUF = 0;
    CALL PRINTCHAR(END$FILE);
    CALL CRLF;
    END DEL$N;

DISPLAY$RM$O$N: PROCEDURE;
```

```
    DECLARE I BYTE;
    I = 0;
    CALL CRLF;
    DO WHILE (I:=I+1) <= OBUF;
       IF I <= OPTR THEN /* EVEN LINE */
         CALL PRINTCHAR(' ');
       ELSE CALL PRINTCHAR(OBUF(I));
    END;
    CALL CRLF;
    CALL PRINT$NEW;
    END DISPLAY$RM$O$N;

DEL$O: PROCEDURE;
    IF OPTR > 0 THEN
       DO;
        DECLARE I BYTE;
         I = OPTR-1;
         DO WHILE (I:=I+1) < OBUF;
            OBUF(I) = OBUF(I+1);
         END;
        CALL PRINTCHAR(PERCENT);
        OBUF = OBUF - 1;
       END;
    ELSE CALL PRINTCHAR(BELL);
    END DEL$O;

ESCAPE: PROCEDURE;
    /* TURNS OFF SPECIAL MEANING OF CHARACTER TO FOLLOW
         AND ENTERS CHARACTER IN NEW BUFFER    */

    CALL PRINTCHARI(CHAR:=READC);
    NBUF(NPTR:=NPTR+1) = CHAR;
    END ESCAPE;

PRINT$TAB: PROCEDURE;
    IF (NPTR + 5) > SIZE$NBUF THEN
       CALL PRINTCHAR(BELL);
    ELSE
       NBUF(NPTR:=NPTR+1) = TAB;
       CALL PRINTCHAR(TAB);
    END PRINT$TAB;

       /* END OF PROCEDURES CALLED FROM THE LINE EDITOR */

INC$DA: PROCEDURE;
    DAT$AREA = DAT$AREA +1;
    END INC$DA;

INC$CA: PROCEDURE;
    CODE$A = CODE$A + 1;
    END INC$CA;

INC$ER: PROCEDURE;
    ERRA = ERRA + 1;
    END INC$ER;

MOVE$DL$NEW: PROCEDURE;
    DO WHILE NPTR <= DL$LEN;
         IF PROMPT THEN CALL OLD$TO$NEW;
         ELSE CALL ECHO$ON;
    END;
    NB = NEW$BUF + NPTR + 1;
    OPTR = NPTR;
    END MOVE$DL$NEW;

MOVE$CODE$OLD: PROCEDURE;
    DECLARE DEST ADDRESS, D BASED DEST BYTE;
    DEST = OLD$BUF+1;
    OPTR,NPTR,OBUF = 0;
    DO WHILE DAT <> EOC;
```

```
            D = DAT;
            IF NOT(PROMPT) THEN CALL PRINTCHAR(D);
            CALL INC$DA;
            DEST = DEST +1;
            OBUF = OBUF +1;
        END;
        CALL CRLF;
        CALL MOVE$DL$NEW;
        END MOVE$CODE$OLD;

SET$PTR: PROCEDURE;
        DO WHILE DAT <> ERR;
            CALL INC$DA;
        END;
        CALL INC$DA;
        BERRA,ERRA = DAT$AREA;
        CALL INC$DA;
        DO WHILE DAT <> ERR;
            CALL INC$DA;
        END;
        CALL INC$DA;
        PROMPT$AREA = DAT$AREA;
        END SET$PTR;

NEXT$DE: PROCEDURE;
        CALL MOVE$CODE$OLD;
        CALL INC$DA;
        CALL SET$PTR;
        END NEXT$DE;

UPDATE$DAT: PROCEDURE;
        DECLARE T ADDRESS, (I,A) BYTE;
        INS$INC: PROCEDURE;
            CODE = NBUF(I); I= I + 1;
            CALL INC$CA;
            END INS$INC;

        CODE$A = B$CODE$A + DL$LEN;
        I = DL$LEN+1;
        DO WHILE (A:=I <= NPTR) AND (I <= OBUF);
            CALL INS$INC;
        END;
        IF A THEN /* CODE LINE HAS GROWN */
            DO;
              DO WHILE I <= NPTR+1;
                  IF CODE = ERR THEN /* AT ERROR CMDS */
                    CALL ERROR(0);
                ELSE
                    CALL INS$INC;
              END;
              CODE = EOC;
            END;
        ELSE
            DO;
              CODE = EOC; T = OLD$BUF+OBUF+1;
              DO WHILE (CODE$A :=CODE$A +1) <= T;
                  CODE = 0;
              END;
            END;
        END UPDATE$DAT;

  /* ************************************************************

                        ERROR MODULE
          FUNCTION:  TO CHECK FOR POSSIBLE ERROR CONDITIONS.
            ERROR COMMANDS ARE DEFINED IN DO$CMD PROCEDURE.

    ************************************************************ */

RE$ENTER: PROCEDURE;
        CALL MOVE$TO$OLD;
        CALL PRINT$OLD;
```

```
     NPTR = NB - NEW$BUF;
     CALL PRINT$NEW;
     END RE$ENTER;


WARNING: PROCEDURE (I);
     DECLARE I BYTE;
     WARN = TRUE;
     DO CASE I;
          CALL PRINT(.'WILL DESTROY OLD INFO  $');
          CALL PRINT(.'EXPECTING ALPHABETIC CHAR  $');
          CALL PRINT(.'EXPECTING NUMERIC CHAR  $');
     END; /* CASE  */
     CALL CRLF;
     END WARNING;

INC$NB: PROCEDURE;
     NB = NB + 1;
     END INC$NB;

SP$PD$COM: PROCEDURE BYTE;
     DECLARE SPACE LIT '20H',
          PERIOD LIT '2EH',
          COMMA LIT '2CH';
     RETURN ((TN = SPACE) OR (TN = PERIOD) OR (TN = COMMA));
     END SP$PD$COM;

ALPHA: PROCEDURE BYTE;
     DECLARE LCA LIT '61H', LCZ LIT '7AH';
     RETURN (((TN >= 'A') AND (TN <= 'Z')) OR ((TN >= LCA)
        AND (TN <= LCZ)) OR SP$PD$COM);
     END ALPHA;

CHK$ALPHA: PROCEDURE;
     DO WHILE TN <> '/';
          IF NOT(ALPHA) THEN
             DO;
               CALL WARNING(1);
               RETURN;
             END;
          CALL INC$NB;
     END;
     END CHK$ALPHA;

NUMERIC: PROCEDURE BYTE;
     RETURN (((TN - '0') <= 9) OR (TN = 2DH /* MINUS */)
        OR SP$PD$COM);
     END NUMERIC;

CHK$NUMERIC: PROCEDURE;
     DO WHILE TN <> '/';
          IF NOT(NUMERIC) THEN
          DO; CALL  WARNING(2); RETURN; END;
          CALL INC$NB;
     END;
     END CHK$NUMERIC;

NEXT$SF: PROCEDURE;
     DO WHILE CODE <> '/';
          CALL INC$CA;
     END;
     CALL INC$CA;
     DO WHILE DAT <> '/';
          CALL INC$DA;
     END;
     CALL INC$DA;
     CALL INC$NB;
     CALL INC$ER;
     END NEXT$SF;

DO$CMD: PROCEDURE;
     IF ECMD = '0' THEN RETURN; ELSE
```

```
     IF ECMD = 'A' THEN CALL CHK$ALPHA; ELSE
     IF ECMD = 'N' THEN CALL CHK$NUMERIC; ELSE
     CALL ERROR(2);
     END DO$CMD;

CHK$ERR: PROCEDURE;
     WARN = FALSE;
     DO WHILE ECMD <> ERR;
          IF ECMD = '/' THEN
               CALL NEXT$SF;
          CALL DO$CMD;
     IF WARN THEN RETURN;
     CALL INC$ER;
     END;
     END CHK$ERR;

ASK$QUESTION: PROCEDURE;
     DO WHILE DAT <> '/';
          CALL PRINTCHAR(DAT);
          CALL INC$DA;
     END;
     END ASK$QUESTION;

CHK$ANSWER: PROCEDURE;
     WARN = FALSE;
     DO WHILE ECMD <> '/';
          CALL DO$CMD;
          IF WARN THEN RETURN;
          CALL INC$ER;
     END;
     END CHK$ANSWER;

END$IP: PROCEDURE;
     OB = OLD$BUF + DL$LEN + 1;
     IF (NB:=NEW$BUF+DL$LEN+1) > NEW$BUF + NPTR THEN
        DO;
          CALL WARNING(0);
          RETURN;
        END;
     NBUF(NPTR+1) = '/';
     CALL CHK$ERR;
     END END$IP;

LEDIT: PROCEDURE;

     DO WHILE NPTR < SIZE$NBUF;
     IF (CHAR:=READC) <= CTLZ THEN /* CONTROL CHAR */
          DO CASE CHAR;
               /* CAS 0 NULL */
               ;

               /* CASE 1 CONTROL A  */
               CALL BACKUP;

               /* CASE 2 CONTROL B  */
               CALL MOVE$TO$OLD;

               /* CASE 3 CONTROL C  */
               CALL COPY$ONE;

               /* CASE 4 CONTROL D  */
               DO;
                CALL P$MOVE$ON;
                GO TO ENDEDIT1;
              END;

               /* CASE 5 CONTROL E  */
               CALL ENTER;

               /*  CASE 6 CONTROL F  */
               GO TO ENDEDIT2;
```

```
            /* CASE 7 CONTROL G   */
            CALL PRINT$BOTH;

           /* CASE 8 CONTROL H   */
            CALL P$MOVE$ON;

           /* CASE 9 CONTROL I   */
            CALL PRINT$TAB;

            /* CASE 10 CONTROL J   */
            GO TO ENDEDIT1;

            /* CASE 11 CONTROL K   */
            ;

            /* CASE 12 CONTOL L   */
            CALL COPY$RM$O$N;

            /* CASE 13 CONTROL M   */
            GO TO ENDEDIT1;

            /* CASE 14 CONTROL N   */
            CALL BS$O$N;

            /* CASE 15 CONTROL O   */
             CALL COPY$ON(READC);

            /* CASE 16 CONTROL P   */
            CALL DELETE(TRUE);

            /* CASE 17 CONTROL Q   */
            CALL DEL$N;

            /* CASE 18 CONTROL R   */
            CALL DISPLAY$RM$O$N;

            /* CASE 19 CONTROL S   */
            CALL DEL$O;

            /* CASE 20 CONTROL T   */
            ;

            /* CASE 21 CONTROL U */
            CALL COPY$ON(TAB);

            /* CASE 22 CONTROL V   */
            CALL ESCAPE;

            /* CASE 23 CONTROL W   */
            ; /* LATER */

            /* CASE 24 CONTROL X   */
            CALL DELETE(FALSE);

            /*CASE 25 CONTROL Y */
            DO;
               CALL P$MOVE$ON;
               CALL MOVE$TO$OLD;
            END;

            /* CASE 26 CONTROL Z   */
             CALL COPY$ON(READC);


        END;
    ELSE  /* CHECK SPECIAL CASES */
    IF CHAR = RUBOUT THEN
       CALL BACKUP;
    ELSE
        DO;
           CALL PRINTCHAR(CHAR);
           NBUF(NPTR:=NPTR+1)=CHAR;
```

```
            IF NPTR = 72 THEN CALL PRINTCHAR(BELL);
        IF NOT(INSERT) THEN OPTR = OPTR + 1;
        END;
    END;  /* DO WHILE */

    /* ARRIVE HERE IF BUFFER FULL */

    CALL PRINTCHAR(BELL);
    ENDEDIT1:
    ENDEDIT2: CALL CRLF;
    END LEDIT;

  /* ************************************************************

                                OUTPUT MODULE

            FUNCTION: TO UPDATE THE DAT FILE AND THE
        INFORMATION JUST EDITED TO THE MESSAGE FILE.

  ************************************************************ */

INC$MSG: PROCEDURE;
    IF (MSG$AREA:=MSG$AREA + 1) < 100H THEN
        RETURN;
    IF DISKWRITE(.MSG$FCB) <> 0 THEN
        CALL ERROR(4);
    MSG$AREA = 80H;
    END INC$MSG;

MOVE$MSG: PROCEDURE;
    IF PRINT$LABEL THEN DAT$AREA = B$CODE$A;
    ELSE DAT$AREA = B$CODE$A + DL$LEN + 1;
    DO WHILE DAT <> EOC;
        MSG = DAT;
        CALL INC$MSG;
        CALL INC$DA;
    END;
    MSG = CR;
    CALL INC$MSG;
    MSG = LF;
    CALL INC$MSG;
    END MOVE$MSG;

WRITE$MSG: PROCEDURE;
    MSG = CTLZ;
    IF DISKWRITE(.MSG$FCB) <> 0 THEN
        CALL ERROR(4);
    END WRITE$MSG;

CLOSE$FILES: PROCEDURE;
    IF CLOSE(.MSG$FCB) = 255 THEN
        CALL ERROR(7);
    DFCB(12) = SAVE$EXT;
    DFCB(32) = SAVE$RN;
    IF CLOSE(DAT$FCB) = 255 THEN
        CALL ERROR(6);
    END CLOSE$FILES;

BLANK$BUF: PROCEDURE;
    DECLARE A ADDRESS, (B BASED A,I) BYTE;
    A = .BUFFER;
    DO I = 1 TO 180;
        B = 0; A = A + 1;
    END;
    END BLANK$BUF;

BASE$NEXT$DE: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO WMPTR(2);
        B$CODE$A = B$CODE$A + 128;
    END;
    WORK$MAP = WORK$MAP + 3;
```

```
    END BASE$NEXT$DE;

UPDATE$DAT$FILE: PROCEDURE;
    WORK$MAP = .MEMORY;
    B$CODE$A = BASE$DAT$AREA;
    DO WHILE B$CODE$A < T$DAT$AREA;
        CALL MOVE$MSG;
        DFCB(12) = WMPTR;
        DFCB(32) = WMPTR(1);
        CALL SETDMA(B$CODE$A);
        IF DISKWRITE(DAT$FCB) <> 0 THEN
            CALL ERROR(4);
        CALL BASE$NEXT$DE;
    END;
    CALL SET$DMA(80H);
    END UPDATE$DAT$FILE;


EDIT: PROCEDURE;
    CONTINUE:
    DO WHILE (DAT$AREA < T$DAT$AREA);
    CALL NEXT$DE;
        IF PROMPT THEN
          DO WHILE DAT <> EOP;
                CALL ASK$QUESTION;
                WARN = TRUE;
                DO WHILE WARN;
                   CALL LEDIT;
                   NBUF(NPTR:=NPTR+1) = '/';
                   CALL CHK$ANSWER;
                END;
                CALL NEXT$SF;
          END;
        ELSE
          DO;
          CALL LEDIT;
          CALL END$IP;
          END;
        IF WARN THEN CALL RE$ENTER;
        ELSE
        DO;
          CALL UPDATE$DAT;
          CALL BLANK$BUF;
          CALL BASE$NEXT$DE;
          DAT$AREA,CODE$A = B$CODE$A;
        END;
    END; /* DO WHILE */

    CALL UPDATE$DAT$FILE;
    IF MORE THEN
        DO;
          CALL READ$MORE;
          GO TO CONTINUE;
        END;
    CALL WRITE$MSG;
    CALL CLOSE$FILES;
    GO TO BOOT;
    END EDIT;


     /******* START MAIN PROGRAM HERE   ****** */

    OLD$BUF = (NEW$BUF := .BUFFER)+90;
    OBUF = 0;
CALL INIT;
CALL SEL$WE;
CALL SET$MEM;
CALL READ$DAT;
CALL LIFTHEAD;
T$DAT$AREA = DAT$AREA;
B$CODE$A,CODE$A,DAT$AREA = BASE$DAT$AREA;
WORK$MAP = .MEMORY;
```

```
CALL CRLF;
CALL PRINT(.'DATA LABELS TO BE IN THE REPORT?$');
CALL PRINTCHAR(CHAR := READC);
CALL CRLF;
IF CHAR = 'Y' THEN PRINT$LABEL = TRUE;
CALL PRINT(.'DO YOU WISH TO BE PROMPTED?$');
CALL PRINTCHAR(CHAR:=READC);
CALL CRLF;
IF CHAR = 'Y' THEN PROMPT = TRUE;
CALL EDIT;
EOF
```

```
  /* ************************************************************

       PROGRAM DESIGNED TO CREATE DAT EXECUTABLE FILES USED
     IN CONJUNCTION WITH REPORT ORIGINATION SYSTEM (ROS).
        ROS IS DESIGNED TO GENERATE FORMATTED REPORTS.

  ************************************************************ */
100H:

  /* ************************************************************

                              INITIALIZE DECLARATIONS

  ************************************************************ */
DECLARE
    LIT         LITERALLY       'LITERALLY',
    BOOT        LIT   '0',
    ENTRY       LIT   '0005H',
    TRUE        LIT   '1',
    FALSE       LIT   '0',
    FOREVER     LIT   'WHILE TRUE',
    CR          LIT   '0DH',
    LF          LIT   '0AH',
    DCNT              BYTE,
    CTI         LIT   '0',
    CTS         LIT   '1';

  /* ************************************************************

                            INPUT AND EDITING DECLARATIONS

  ************************************************************ */
DECLARE
    BS          LIT   '08H',    /* BACKSPACE  */
    PERCENT     LIT   '25H',
    BELL        LIT   '07H',
    TAB         LIT   '09H',
    EOP         LIT   '5EH', /* UP-ARROW; END OF PROMPT */
    END$FILE    LIT   '5CH', /*BACK SLANT */
    EOC         LIT   '7CH', /*BAR; END OF CODE */
    ERR         LIT   '7EH', /* TILDE; END OF ERROR */
    CTLZ        LIT   '1AH',
    RUBOUT      LIT   '7FH',
    DAT$FCB           ADDRESS INITIAL (5CH),
    DFCB  BASED DAT$FCB (33)  BYTE,
    NUM$REC                   BYTE,
    DL$LEN                    BYTE,
    TMEM              ADDRESS,
    RECORD$MAP                    ADDRESS,
    RMPTR BASED RECORD$MAP        BYTE,
    EXT         BYTE  INITIAL(0),
    STORE                         ADDRESS,
    SP    BASED STORE         BYTE,
    BSTORE                    ADDRESS,
    SPTR                      ADDRESS,
    BUFFER  (180)             BYTE,
    SIZE$NBUF LIT             '90',
    NEW$BUF                   ADDRESS,
    NBUF    BASED NEW$BUF     BYTE,
    NPTR              BYTE,
    OLD$BUF                   ADDRESS,
    OBUF    BASED OLD$BUF     BYTE,
    OPTR              BYTE,
    INSERT      BYTE INITIAL (FALSE),
    CHAR        BYTE,
    MOD$128$MASK    LIT   '0FF80H',    /* GIVES MEMORY SIZE
         IN MULIPLES OF 128 BYTE BLOCKS */
    BDOSA       ADDRESS INITIAL (0006H),
    SBDOS       BASED BDOSA   ADDRESS;
```

```
CRTIN: PROCEDURE BYTE;
    DO WHILE INPUT(CTS);
    END;
    RETURN NOT INPUT(CTI) AND 07FH;
    END CRTIN;

READC: PROCEDURE BYTE;

    /* GET A CHARACTER FROM THE CONSOLE AND TRANSLATE TO
         UPPER CASE      */

    DECLARE C BYTE;
    IF (C:=CRTIN) >= 110$0001B  /* LOWER CASE A */
       AND C <= 01111010B  /* LOWER CASE Z */ THEN
       C = C AND 101$1111B;  /* BECOMES UPPER CASE */
    RETURN C;
    END READC;

MON1: PROCEDURE (FUNC, INFO);
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON1;

MON2: PROCEDURE (FUNC,INFO) BYTE;
    DECLARE FUNC BYTE, INFO ADDRESS;
    GO TO ENTRY;
    END MON2;

PRINTCHAR: PROCEDURE (B);
    DECLARE B BYTE;
    CALL MON1(2,B);
    END PRINTCHAR;

PRINTCHARI: PROCEDURE(C);
    DECLARE C BYTE;
    IF (C AND 110$0000B) = 0 /* CONTROL CHAR */ THEN
       DO;
         CALL PRINTCHAR(EOP);
         CALL PRINTCHAR(C OR 40H);
       END;
    ELSE CALL PRINTCHAR(C);
    END PRINTCHARI;

CRLF: PROCEDURE;
   CALL PRINTCHAR(CR);
   CALL PRINTCHAR(LF);
    END CRLF;

PRINT: PROCEDURE (A);
    DECLARE A ADDRESS;
    CALL MON1(9,A);
    CALL CRLF;
    END PRINT;

MOVE: PROCEDURE (SOURCE,DEST,N);
    DECLARE (SOURCE,DEST) ADDRESS,
      (S BASED SOURCE, D BASED DEST, N ) BYTE;
    DO WHILE (N:=N-1) <> 255;
       D=S; SOURCE=SOURCE+1; DEST=DEST+1;
    END;
    END MOVE;

FILL: PROCEDURE(START,DEST,CHAR);
    DECLARE (START,DEST) ADDRESS,
      (S BASED START, CHAR) BYTE;
    DO WHILE START < DEST;
       S = CHAR;
       START = START + 1;
    END;
    END FILL;

ERROR: PROCEDURE(I);
```

```
    DECLARE I BYTE;
         DO CASE I;
         CALL PRINT(.'LACK ERROR COMMAND SPACE  $');
         CALL PRINT(.'DISK WRITE ERROR    $');
         CALL PRINT(.'FILE NOT PRESENT $');
         END;
    CALL CRLF;
    GO TO BOOT;
    END ERROR;

  /* ************************************************************

                          INPUT AND EDITING MODULE

  ************************************************************ */

PROMPT: PROCEDURE(I);
    DECLARE I BYTE;
    CALL MON1(9,.'EXPECTING $');
    DO CASE I;
       CALL PRINT(.'CODE INFO $');
       CALL PRINT(.'ERROR COMDS $');
       CALL PRINT(.'PROMPT INFO $');
    END;
    END PROMPT;

INC$RM: PROCEDURE;
    RECORD$MAP = RECORD$MAP + 1;
    END INC$RM;

GO$NEXT$REC: PROCEDURE;
    /* INCREMENTS STORAGE POINTER TO NEXT EVEN
         RECORD SECTOR */
    DO WHILE SPTR < STORE;
         SPTR = SPTR + 128;
        NUM$REC=NUM$REC+1;
         IF NUM$REC = 123 THEN
           DO;
              EXT = EXT + 1;
              NUM$REC = 0;
           END;
    END;
    STORE = SPTR-1;
    END GO$NEXT$REC;

MOVE$DL: PROCEDURE;
    DECLARE A ADDRESS, I BYTE;
    I=0; A=RECORD$MAP+DL$LEN;
    DO WHILE (RECORD$MAP:=RECORD$MAP+1) <= A;
         RMPTR = OBUF(I:=I+1);
    END;
    RMPTR = EXT;
    CALL INC$RM;
    RMPTR = NUM$REC;
    END MOVE$DL;

WRITE: PROCEDURE;
    DECLARE A ADDRESS;
    A = .MEMORY;
    DO WHILE (A:=A+128) < STORE;
         CALL MOVE(A,80H,128);
    IF (DCNT := MON2(21,DAT$FCB)) <> 0 THEN
         CALL ERROR(1);
    END;
    STORE=BSTORE;SPTR=BSTORE;
    CALL MON1(12,0); /* LIFT READ WRITE HEAD */
    END WRITE;

INC$STORE: PROCEDURE;
    /* CHECKS FOR MEMORY OVERFLOW INCREMENTS STORAGE PTR */
    IF (STORE:=STORE+1) > TMEM THEN
         CALL WRITE;
```

```
    END INC$STORE;

MOVE$STORE: PROCEDURE;
    /* STORES INFORMATION FROM INPUT TO FILE MEMORY AREA */
    DECLARE I BYTE;
     DO I=1 TO OBUF;
          CALL INC$STORE;
          SP = OBUF(I);
    END;
    END MOVE$STORE;

FILL$CODE$ZERO: PROCEDURE;
    DECLARE (A,T) ADDRESS, B BASED A BYTE;
    A = STORE; T = SPTR + 128;
    DO WHILE (A:=A+1) < T;
       B = 0;
    END;
    END FILL$CODE$ZERO;

END$DL: PROCEDURE;
    /* CHARACTER IS A BAR (|) INDICATES END OF CODE AREA */
    CALL GO$NEXT$REC;
    CALL MOVE$DL;
    CALL MOVE$STORE;
    CALL INC$STORE;
    SP = EOC;
    CALL FILL$CODE$ZERO;
    CALL CRLF;
    CALL PROMPT(1);
    END END$DL;

END$REC: PROCEDURE;
    /* CALL WHEN A UP-ARROW  IS ENTERED FROM THE KEYBOARD.
         INDICATES END OF DECODED INFORMATION   */

    CALL MOVE$STORE;
    CALL INC$STORE;
    SP = EOP;
    CALL CRLF;
    CALL PROMPT(0);
    END END$REC;

END$ERR: PROCEDURE;
    /* CALLED WHEN (TILDE) ENTERED AT KEYBOARD
       INDICATES END OF ERROR CHECKS   */
    BACK$STORE: PROCEDURE;
         STORE = SPTR + 127; /* TO NEXT RECORD - 1 */
         SP = ERR;
         STORE = STORE - 1;
         OBUF = OBUF + 1;
         DO WHILE (OBUF:=OBUF-1) <> 0;
              IF SP = EOC THEN
                   CALL ERROR(0);
              SP = OBUF(OBUF);
              STORE = STORE - 1;
         END;
         SP = ERR;
         END BACK$STORE;

    CALL BACK$STORE;
    CALL GO$NEXT$REC;
    CALL CRLF;
    CALL PROMPT(2);
    END END$ERR;

END$F: PROCEDURE;
    /* CALLED WHEN END FILE ( ) BLACKSLASH ENTERED INDICATES
              END OF FILE   */
    DECLARE (EX,NR) BYTE;
    CALL MOVE$STORE;
    CALL INC$STORE;
    SP=END$FILE;
```

```
    CALL WRITE;
    CALL INC$RM;
    RMPTR = EOC; /* MARK END OF RECORD$MAP */
    RECORD$MAP = RECORD$MAP + DL$LEN; /* SAVE EXT AND RN OF
            NEXT RECORD TO BE WRITTEN   */
    RMPTR = DFCB(12);
    CALL INC$RM;
    RMPTR = DFCB(32);
    EX = DFCB(12); NR = DFCB(32);
    DFCB(32),DFCB(12) = 0;
    CALL MOVE(.MEMORY,80H,128);
    IF (DCNT := MON2(21,DAT$FCB)) <> 0 THEN
       CALL ERROR(1);
    DFCB(12) = EX; DFCB(32) = NR;
    IF MON2(16, DAT$FCB) = 255 THEN  /* CLOSE FILE */
       CALL ERROR(2);
    GO TO BOOT;
    END END$F;

BACK$UP: PROCEDURE;
    IF NPTR > 0 THEN
       DO;
         NPTR = NPTR -1;
         CALL PRINTCHAR(BS);
         CALL PRINTCHAR(' ');
         CALL PRINTCHAR(BS);
       END;
    ELSE
       CALL PRINTCHAR(BELL);
    END BACK$UP;

MOVE$TO$OLD: PROCEDURE;
    CALL MOVE(NEW$BUF+1,OLD$BUF+1,(OBUF:=NPTR));
    OPTR = 0; NPTR = 0;
    CALL CRLF;
    END MOVE$TO$OLD;

ECHO$ON: PROCEDURE;
   CALL PRINTCHAR(NBUF(NPTR:=NPTR+1):=(OBUF(OPTR:=OPTR+1)));
    END ECHO$ON;

COPY$ONE: PROCEDURE;
    IF OPTR < OBUF THEN
       CALL ECHO$ON;
    ELSE CALL PRINTCHAR(BELL);
    END COPY$ONE;

P$MOVE$ON: PROCEDURE; /* PARTIAL MOVE OLD TO NEW */
    DO WHILE OPTR <  OBUF;
    CALL ECHO$ON;
    END;
    END P$MOVE$ON;

ENTER: PROCEDURE;
    IF INSERT THEN
       CALL PRINTCHAR('>');
    ELSE
       CALL PRINTCHAR('<');
    INSERT = NOT(INSERT);
    END ENTER;

PRINT$OLD: PROCEDURE;
    DECLARE I BYTE;
    DO I= 1 TO OBUF;
       CALL PRINTCHARI(OBUF(I));
    END;
    CALL CRLF;
    END PRINT$OLD;

PRINT$NEW: PROCEDURE;
    DECLARE I BYTE;
    DO I = 1 TO NPTR;
```

```
      CALL PRINTCHARI(NBUF(I));
   END;
   END PRINT$NEW;

PRINT$BOTH: PROCEDURE;
   CALL PRINT$OLD;
   CALL PRINT$NEW;
   END PRINT$BOTH;

COPY$RM$O$N: PROCEDURE;
  /* COPIES REMAINING CHARACTERS FOR OLD TO NEW BUFFERS */

   DO WHILE OPTR <= OBUF;
      NBUF(NPTR:=NPTR+1) = OBUF(OPTR:=OPTR+1);
   END;
   CALL PRINTCHAR('+');  /* INDICATES WHEN DONE  */
   END COPY$RM$O$N;

BS$O$N: PROCEDURE;
   /* BACKSPACE OLD PTR AND NEW PTR 1 CHAR  */
   IF (OPTR > 0) AND (NPTR > 0) THEN
      DO;
       OPTR = OPTR - 1;
       NPTR = NPTR - 1;
        OBUF = OBUF - 1;
      END;
   ELSE
      CALL PRINTCHAR(BELL);
   END BS$O$N;

COPY$ON: PROCEDURE (C,NEXT);
   DECLARE (C,I,NEXT) BYTE;
   I=OPTR;
   DO WHILE OBUF(I:=I+1) <> C;
      IF I > OBUF THEN /* NO MATCH */
        DO;
          CALL PRINTCHAR(BELL);
          RETURN;
        END;
   END; /* DO WHILE */
   IF NOT(NEXT) THEN I=I-1;
   DO WHILE OPTR < I;
      CALL ECHO$ON;
   END;
   END COPY$ON;

DELETE: PROCEDURE(ECHO);

      /* ECHO TRUE INDICATES TO START FROM THE CURRENT
   POSITION OF OLD BUFFER AND ECHO A % (PERCENT) FOR THE
   DELETED CHARACTER.  ECHO FALSE INDICATES TO START AT
   THE BEGINNING OF THE OLD BUFFER AND DON'T ECHO FOR
   THE DELETED CHARACTERS. */

   DECLARE (I,J,P1,CHAR1,ECHO) BYTE;
   IF ECHO THEN P1 = 0;
   ELSE P1 = OPTR;
   CHAR1 = READC;
   DO WHILE (OBUF(P1:=P1+1) <> CHAR1);
      IF P1 > OBUF THEN /* NO MATCH */
        DO;
           CALL PRINTCHAR(BELL);
           RETURN;
        END;
   END;  /* DO WHILE  */
   IF ECHO THEN
       DO I = OPTR+1 TO P1;
          CALL PRINTCHAR(PERCENT);
       END;

    /*  NOW CONDENSE THE BUFFER   */
   J=OPTR;
```

```
     I=P1;
     DO WHILE I <= OBUF;
          OBUF(J:=J+1) = OBUF(I:=I+1);
     END;
     OBUF = OBUF - (P1-OPTR+1);
     END DELETE;

DEL$N: PROCEDURE;
     NPTR=0; OPTR=0;
     CALL PRINTCHAR(END$FILE);
     CALL CRLF;
     END DEL$N;

DISPLAY$RM$O$N: PROCEDURE;
     DECLARE I BYTE;
     I = 0;
     CALL CRLF;
     DO WHILE (I:=I+1) <= OBUF;
        IF I <= OPTR THEN /* EVEN LINE */
          CALL PRINTCHAR(' ');
        ELSE
          CALL PRINTCHAR(OBUF(I));
     END;
     CALL CRLF;
     CALL PRINT$NEW;
     END DISPLAY$RM$O$N;

DEL$O: PROCEDURE;
     IF OPTR > 0 THEN
        DO;
          DECLARE I BYTE;
          I = OPTR - 1;
          DO WHILE (I:=I+1) < OBUF;
               OBUF(I) = OBUF(I+1);
        END;
        CALL PRINTCHAR(PERCENT);
        OBUF = OBUF - 1;
        END;
     ELSE CALL PRINTCHAR(BELL);
     END DEL$O;

ESCAPE: PROCEDURE;
     /* TURNS OFF SPECIAL MEANING OF CHARACTER TO FOLLOW
          AND ENTERS CHARACTER IN NEW BUFFER    */

     CALL PRINTCHARI(CHAR:=READC);
     NBUF(NPTR:=NPTR+1) = CHAR;
     END ESCAPE;

CONT$FILL: PROCEDURE;
     CALL MOVE$STORE;
     CALL CRLF;
     END CONT$FILL;

PRINT$TAB: PROCEDURE;
     IF (NPTR + 5) > SIZE$NBUF THEN
        CALL PRINTCHAR(BELL);
     ELSE
        NBUF(NPTR:=NPTR+1) = TAB;
        CALL PRINTCHAR(TAB);
     END PRINT$TAB;

LEDIT: PROCEDURE;
     /* READS CHARACTERS FROM THE CONSOLE AND ALLOWS EDITING
     USING THE PROCEDURES OF A LINE EDITOR  */

     OPTR = 0; NPTR = 0;
     DO WHILE NPTR < SIZE$NBUF;
     IF (CHAR:=READC) <= CTLZ THEN /* CONTROL CHAR */
          DO CASE CHAR;
               /* CAS 0 NULL */
               ;
```

```
 /* CASE 1 CONTROL A  */
 CALL BACKUP;

 /* CASE 2 CONTROL B  */
 CALL MOVE$TO$OLD;

 /* CASE 3 CONTROL C  */
       CALL COPY$ONE;

 /* CASE 4 CONTROL D  */
 DO;
  CALL P$MOVE$ON;
  GO TO ENDEDIT1;
END;

 /* CASE 5 CONTROL E  */
 CALL ENTER;

 /*  CASE 6 CONTROL F  */
 GO TO ENDEDIT2;

 /* CASE 7 CONTROL G  */
 CALL PRINT$BOTH;

/* CASE 8 CONTROL H  */
 CALL P$MOVE$ON;

/* CASE 9 CONTROL I  */
 CALL PRINT$TAB;

 /* CASE 10 CONTROL J  */
 GO TO ENDEDIT1;

 /* CASE 11 CONTROL K  */
 ;

 /* CASE 12 CONTOL L  */
 CALL COPY$RM$O$N;

 /* CASE 13 CONTROL M  */
 GO TO ENDEDIT1;

 /* CASE 14 CONTROL N  */
 CALL BS$O$N;

 /* CASE 15 CONTROL O  */
 CALL COPY$ON(READC,FALSE);

 /* CASE 16 CONTROL P  */
 CALL DELETE(TRUE);

 /* CASE 17 CONTROL Q  */
 CALL DEL$N;

 /* CASE 18 CONTROL R  */
 CALL DISPLAY$RM$O$N;

 /* CASE 19 CONTROL S  */
 CALL DEL$O;

 /* CASE 20 CONTROL T  */
 CALL CONT$FILL;

 /* CASE 21 CONTROL U */
 CALL COPY$ON(TAB,FALSE);

 /* CASE 22 CONTROL V  */
 CALL ESCAPE;

 /* CASE 23 CONTROL W  */
 ; /* LATER */
```

```
            /* CASE 24 CONTROL X  */
            CALL DELETE(FALSE);

            /*CASE 25 CONTROL Y */
            DO;
               CALL P$MOVE$ON;
               CALL MOVE$TO$OLD;
            END;

            /* CASE 26 CONTROL Z  */
            CALL COPY$ON(READC,TRUE);


     END;
ELSE  /* CHECK SPECIAL CASES */
IF CHAR = RUBOUT THEN
   CALL BACKUP;
ELSE
IF CHAR = EOC THEN     /* INDICATES END OF CODED INFO */
      DO; CALL MOVE$TO$OLD; CALL END$DL; END;
ELSE
IF CHAR = ERR THEN
      DO; CALL MOVE$TO$OLD; CALL END$ERR; END;
ELSE
IF CHAR = EOP THEN /* END OF PROMPT INFORMATION */
      DO; CALL MOVE$TO$OLD; CALL END$REC; END;
      ELSE
IF CHAR = END$FILE THEN /* END OF FILE */
      DO; CALL MOVE$TO$OLD; CALL END$F; END;
      ELSE
     DO;
        CALL PRINTCHAR(CHAR);
        NBUF(NPTR:=NPTR+1)=CHAR;
        IF NOT(INSERT) THEN OPTR = OPTR + 1;
        IF NPTR = 72 THEN CALL PRINTCHAR(BELL);
      END;
END;  /* DO WHILE */

/* ARRIVE HERE IF BUFFER FULL */

CALL PRINTCHAR(BELL);
ENDEDIT1: CALL MOVE$TO$OLD;
ENDEDIT2:
END LEDIT;

 /*    START MAIN PROGRAM HERE    */

OLD$BUF = (NEW$BUF := .BUFFER)+90;
OBUF = 0;
CALL MOVE(.'DAT', DAT$FCB+9,3);
DFCB,DFCB(12),DFCB(32) = 0;
IF MON2(17, DAT$FCB) <> 255 THEN /*FILE EXISTS */
  DO;
   CALL PRINT(.' FILE ALREADY EXISTS  $');
   GO TO BOOT;
  END;
IF MON2(22, DAT$FCB) = 255 THEN
   DO;
     CALL PRINT(.' OUT OF DIRECTORY SPACE  $');
     GO TO BOOT;
   END;
IF (DCNT:=MON2(15,DAT$FCB)) = 255 THEN /* CAN'T OPEN */
     CALL ERROR(2);
CALL MON1(12,0);    /* LIFT READ WRITE HEAD  */
/*  ARRIVE HERE WITH NEW FILE CREATED   */
DFCB(32) = 1; /* RESERVE FIRST RECORD FOR RECORD MAP */
TMEM = (SBDOS - 1) AND MOD$128$MASK;
CALL FILL(.MEMORY,TMEM,0);
SPTR,RECORD$MAP = .MEMORY;
BSTORE = .MEMORY + 128;
STORE = BSTORE;
```

```
      NUM$REC = 0;
      CALL PRINT(.'LENGTH OF DATA LABELS TO BE USED?$');
      CALL PRINTCHAR(CHAR:=READC);
      CALL CRLF;
      MEMORY,DL$LEN = CHAR AND 0FH;
      CALL PROMPT(0);
      DO FOREVER;
         CALL LEDIT;
      END;
EOF
```

BIBLIOGRAPHY

1. Tollefsen, T.S., LCDR, USN, "Reports or Readiness", NAVAL WAR COLLEGE REVIEW, Vol.26, pp.74-82

2. Naval Electronic Systems Command, TEST AND EVALUATION REPORT X/C 13 INCREMENT I (COMPREP) REPORT 27437-W006-RU-00, June 30, 1975

3. Naval Electronic System Command, FLEET COMMAND CENTER COMPOSTITE OPERATIONS REPORTING SYSTEM, REPORT PME-108-P00011, August 20, 1976

4. Office of the Chief of Naval Operations, STEERING GROUP PRESENTATION, September 13, 1976

5. CP/M Interface Guide, DIGITAL RESEARCH, 1975

6. Office of the Chief of Naval Operations Instruction C3501.66A, January 5, 1976

7. PL/M Programming Manual, INTEL CORPORATION, 1975

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
4.	Assoc Professor U. R. Kodres, Code 52Kr Computer Science Department Naval Postgraduate School Monterey, California 93940	1
5.	Assoc Professor G. A. Kildall, Code 52Kd Computer Science Department Naval Postgraduate School Monterey, California 93940	1
6.	LT Joseph G. Holyoak Supervisor of Shipbuilding Conversion and Repair 574 Washington St. Bath, Maine 04530	1


FILM
5