CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

TEXT FORMATTING PROGRAM

A graduate project submitted in partial satisfaction of the requirements for the degree of Master of Science in Computer Science

by

Gerald Donald Cahill

August, 1979
The Graduate Project of Gerald Donald Cahill is approved:

Professor Russell J. Abbott

Professor Brent W. Perner

Professor Philip Gilbert, Chairman
ACKNOWLEDGMENTS

I wish to express my sincere gratitude to Professors Philip Gilbert, Russell Abbott, and Brent Hoerner for their guidance, contributions, and encouragement during my work on this project. I very much appreciate the time they shared with me, especially the time occurring during summer vacation.

I would also like to thank the Director and staff of the Air Force Rocket Propulsion Laboratory for recognizing my educational needs and providing the support to enable me to attend this University.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. PROGRAM CAPABILITIES</td>
<td>9</td>
</tr>
<tr>
<td>3. OVERVIEW OF PROGRAM</td>
<td>23</td>
</tr>
<tr>
<td>4. DESCRIPTION OF ROUTINES</td>
<td>29</td>
</tr>
<tr>
<td>5. MACHINE DEPENDENT CODE</td>
<td>42</td>
</tr>
<tr>
<td>6. CONCLUSIONS</td>
<td>45</td>
</tr>
<tr>
<td>7. RECOMMENDATIONS</td>
<td>46</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>47</td>
</tr>
<tr>
<td>APPENDICES</td>
<td></td>
</tr>
<tr>
<td>A. PROGRAM LISTING</td>
<td>49</td>
</tr>
</tbody>
</table>
This formatting program enables a user to describe the form of a document and to automate many formatting tasks. Input is a stream of characters which are the text of the document interspersed with formatting commands. One input source may be interrupted to insert characters from another source. The program is flexible enough to support experimentation with different types of formatting commands. Several primitive commands are provided to control the formatting process. Users may define macro commands tailored for a specific type of document. Traps may be defined which will automatically invoke a macro each time a specified line is reached on each output page. Text is formatted for output on a line printer. Optional line justification is accomplished by inserting extra blanks between words and over-printing is used to bold face characters. Extra blanks between words on the input file may be eliminated by the program. Selected variables may be manipulated by
the user and their values inserted into the text. The program is written in Fortran and every effort has been made to make it portable. It is currently running on a Control Data 3170 computer.
Section 1

INTRODUCTION

Document creation consists of composing, revising, and formatting text. The usual approach is to set the words to paper, make corrections in pencil, and rearrange segments with scissors before typing the final draft. This process can be automated by using a digital computer. Computer programs for word processing applications consist of two tasks, text editing of the input and formatting of the output. Text editors allow the user to enter new text into storage and to modify that text by inserting, deleting, correcting, and rearranging it. Text formatters enable the user to describe the form of a document and to automate many formatting tasks.

This program formats text for output on a line printer. A separate editor must be used to create and to modify the input files to this program.

Capabilities of Current Programs

The capabilities of some current programs were determined by surveying various user manuals (4,5,6,7,9), articles describing formatting programs (1,2,3,8,11,12), and sales brochures on word processing systems (13,14,15,16,17,18,19,20,21). Both the methods of user input and the types of formatting commands were noted.
Every system accepts both characters of text and formatting commands as input. Each program allows embedded commands within the document's text and some programs also require special control lines placed in a separate section of the input stream (2, 4, 9). Most programs signal the start of a command with a user specified character while others make use of a special non-printing control character. All of these programs view the input as a sequence of words. Some embedded commands cause word or line breaks while other commands do not delimit words or lines. Each system contains formatting commands for at least one class of document. A few systems (6, 7, 8, 9, 12) allow the user to construct his own macro commands and to access special program variables whose values may be inserted into the document as text. A composite list of the features available in different programs is given below. Every command is available in at least two programs but no single program includes all features listed below. The formatting functions listed are grouped by the effect they have on the final output.

**Forms Control**

Set margins.
Set tabs.
Set indentation (left and right).
Set number of columns of output.

**Text Position Control**

Center text on line (or on the next N lines).
Justify text between the left and right margins.
Invoke indentation.
Invoke tabulation:
   Move to next tab.
Move to next tab and leave a trail of dots.
Place text to the right (or left) of tab.
Center text on tab.
Place specific character of text on tab.
Save space (for inserting drawings):
  Skip lines over the page boundary.
  Skip lines only on the current page.
Signal end of current line.
Signal end of current page.
Skip forward (or backward) N spaces.

**Divert Text**

Footnotes:
  Collect at end of document.
  Place on bottom of page.
Place entries into the index.
Place entries into the table of contents.
Accumulate a list of manual corrections necessary.
Reprint table heading when table crosses a page boundary.

**Text Modification**

Underline.
Bold face (over print).
Overstrike (to form special characters).
Automatic hyphenation:
  Optional and required hyphens.
  Position within word determined by algorithm (10).
  Position obtained from stored dictionary.
Upper/lower case commands.
Character set code conversion.
Automatic sentence separation by N spaces.

**Text Generation**

Page numbers.
Paragraph numbers automatically generated.
Reference to numbered paragraph formed for insertion.
Page header and footer.
Current date.
Indicator of revised lines (vertical bar).
Text Insertion

Boilerplate from secondary input files.
Commands from secondary input files.
Control the position of input files (search for key).
Wait for text to be input from the operator's console.
Macro command expansion.

Document Disposition

Print extra copies.
Print side by side copies.

Statement of Problem

Current programs fall into two categories, those which format only a specific class of document and those which handle many types of documents. Programs written for one type of document have all formatting commands built in and the effect of each command may be changed only within very narrow planned limits (such as changing the setting of a margin). This type of program is generally very easy to use and it is robust since few input errors are possible and all errors are easily recovered. Unfortunately, the user cannot change the effect of a command when a different type of format is required. For example, the command to start a new paragraph for a letter will simply space down one line and indent while the same command used for a manual or contract would also generate and insert a paragraph number.

Programs which handle many types of documents allow the user to construct macro commands to control the formatting process. Macro commands consist of text with embedded commands that handle the lower level formatting details. These are very versatile programs but they
are hard to use. They require an experienced programmer to design the macro commands (8). In addition, many input errors are possible and error recovery is difficult. The programs are written in assembly language and are not portable.

What is needed is a program that combines the best attributes of both categories. It should handle many types of documents, be easy to use, recover from input errors, and be portable.

Method of Solution

The implementation methods to achieve program versatility, ease of use, robustness, and portability were chosen because they seemed most likely to succeed. The two obvious methods for making a program flexible are to give the user a very large menu of program options or to allow the user to construct commands to fit the needs. The menu technique requires that all formatting requirements be known in advance of program design. It would be very unlikely if the resulting program could handle a new, unforeseen requirement. Depending on the number of options, it may be difficult for the user to select the specific options needed for a given document.

Allowing the user to construct commands insures program versatility and is the method used. Macro commands are constructed by combining text and primitive (built-in) commands. The success of this program depends on selecting a set of primitive commands which will be easily understood and useful to the users. At the same time, the primitive commands must be powerful enough to enable macros to perform the functions mentioned in the list of currently available
features. Not only must the features needed be known before the program is designed, but the likes and dislikes of the users must be considered. This program is therefore designed to be used as a tool to acquire user feedback by experimenting with different primitive commands.

Program portability is obtained by using a prevalent programming language, Fortran. The standard version is used without extensions and many comments appear in the source code. All unavoidable machine dependent code is isolated and no other programming tricks are used. In addition, all variables are of type integer and only one character is stored in each word to simplify the text handling routines and to make the input record format flexible.

The program is made robust by designing it to handle all possible input, good or invalid. Error messages are issued for the anticipated errors and other errors will be obvious when the output is examined. The input is consumed one character at a time without backtracking and each character is processed (correctly or otherwise) to insure program termination.

The general implementation plan outlined above leads to several secondary design decisions. The program is driven by the input, one character at a time. This eliminates the need for a separate section of the input stream to contain commands. Formatting commands may be embedded anywhere within the text. More than one source file may supply the input allowing prewritten sections of text to be merged into the main input stream. The input files form a hierarchical structure with input taken only from the most recently opened file. There is no
loss in generality but the user is forced to have only the necessary
files opened which conserves program storage space. Provisions for the
user to specify the format of input files is also included.

User defined commands are first stored and then are inserted
back into the input stream when the command is invoked. The program
distinguishes between input from files and input from macros so that
the stored commands can be used to affect the input files. In order
to increase the versatility of defined commands, the program is
designed to handle traps (an automatic interruption of normal process­
ing). The user specifies line-number/command-name pairs and each
time the specified line is reached on the formatted output page, the
corresponding command is executed before normal processing is resumed.

Each character of text which is to be over-printed can be
treated separately from normal text or can be temporarily modified and
then formatted along with normal text. The latter method is used so
that common formatting routines can be used for both types of text.
All Fortran compilers allow at least two characters to be stored in
each integer word insuring that an unused bit is always available to
be used as a flag to cause the character to be over-printed.

The final design decision to note is the handling of the
formatted output. The routine which formats each line writes it to a
work file rather than immediately printing it. The work file is copied
to the printer only after all input has been processed. This allows
other work files to be added to accumulate the table of contents and
index and to be output in their proper positions within the document.
The method of solution is summarized as follows:

1. Versatility is gained by allowing users to construct macro commands and to set traps.

2. The program can be made easy to use by incorporating user feedback into the design and selection of the primitive commands.

3. Portability is achieved by careful use of the Fortran programming language.

4. Robustness results from the method of processing input and recovering from input errors.
Section 2

PROGRAM CAPABILITIES

This program enables a user to describe the form of a document and to automate many formatting tasks. Input is a stream of characters comprising the text of the document interspersed with formatting commands. Input is accepted from more than one source. Users may define macro commands tailored to their specific document. Traps may be defined to automatically invoke a macro each time a specified line is reached on the output page. Text is formatted for output to a line printer. Optional line justification is accomplished by inserting extra spaces between words and over-printing is used to bold face characters. Extra blanks between words on the input file can be eliminated by the program. Selected variables may be accessed by the user and their values inserted into the text.

All control of this program is accomplished through the use of built-in primitive commands. When a primitive command is invoked, the program changes the way it operates in some respect. This process is most easily visualized by considering the program as an abstract machine whose next action depends only upon its current state and the next input character. The program's state is determined entirely by the current values of the state variables. From this point of view, the effect of each command is to change the value of one or more state variables.
There are two categories of state variables, static and dynamic. Static state variables include program constants and variables whose values can only be changed by a command (for example, the set left margin primitive). The static state variables in this program are:

- Characters used to delimit words.
- Character used to signal the start of a command.
- Characters used to delimit sentences.
- Position of the margins.
- The user variables (registers).
- Macro command definitions.
- Definitions of traps.
- Selectors of program processing options:
  - Justify words on a line.
  - Compress extra spaces between words.
  - Overprint characters of text.
- Descriptors of input record format.

The values of the dynamic state variables are changed automatically by the program as it processes the input stream in addition to being indirectly affected by some primitive commands (such as the end current line command). Dynamic state variables include:

- Content of the current (incomplete) word.
- Content of the current line.
- Content of the current page.
- Content of the incomplete file of formatted output.
- Pointers to source of input characters:
  - The open file stack.
  - The macro command storage area.

All of the commands begin with a special start of command character. The character used by default is the slash but may be changed by using the /SC command. Each command is made up of one or more fields and each field is delimited by a comma or a blank. By allowing the blank to delimit a field, a command which contains a syntax error will still be ended by the text that follows it. The first field must contain the name of the command (only the first two
characters of the name are used and the name may be preceded by an optional minus sign). The first character of the command name must be non-numeric. Additional fields are required by some commands and contain either names of other commands or integer values.

The primitive commands built into the interpreter are listed below. When a command name is encountered that is not on this list, the interpreter will invoke a user defined macro with that name. The purpose of each command and an example of its use is given. This set of primitive commands represents the starting point for the experimentation necessary to gain user acceptance of this program. The reason for including each command is discussed where necessary to make the intended use clear.

**General Command Format**

```
/name,field1,field2,...,fieldn,continue text...
```

The only exception to the command format consists of two consecutive start of command characters (no command delimiter). This causes one start of command character to be placed into the current word of text. By including this command, the start of command character need not be changed before it can be printed. An input of "AND/OR" will be printed as "AND/OR".

**Input Format Control**

```
/SC,x
```

Change the start of command character to x. A seldom used character of text is a good choice. The slash is used as default only to make this typewritten report readable. To change
the start of command character to a number sign, use /SC,#, and to change it back to a slash use #SC,/.

/RL,n, Change the record length read from the input file to n characters. The value of n must be at least as large as the position of the last character used (see /LC) but not greater than 90. The default value is 80. Input will be accepted from a 90 character input record by using /RL,90, (and followed by an /LC,90, in most cases).

/FC,n, Set the first character position to n. The character in position n of each input record will be the first character used by the program. The default value is 1. This command is used to skip over sequence numbers when they are at the front of the records. If the first 5 characters of each record are to be ignored, use /FC,6,.

/LC,n, Set the last character position to n. The character in position n of each input record will be the final character used from that record. Default is 80. The value of n must be greater than or equal to the value given by the /FC command and less than or equal to the value given by the /RL command. To skip record identification appearing in positions 73 through 80 of an 80 character record, use /LC,72,.

Input Source Selection

/OP,n, Open input file on unit n. The current input buffer and character pointer are saved (stacked) and input is fetched from the first record of the file on unit n. The system input file is opened by default when this program is started. Up to two additional files may
be opened. Input is always obtained from the last file opened. This command is useful for inserting "boiler plate" from a library input file. To interrupt the current input file and insert text and commands from a file on unit 6, use /OP,6,.

/CL, Close the last input file opened. Subsequent input will be obtained from the last file interrupted by an /OP command (if any). The reasons for using files in a hierarchical structure were covered in the Method of Solution section. In the absence of a /CL command, the program will automatically close a file when the end-of-file is reached.

Text Control

The commands in this group affect the program's dynamic state variables. There is an immediate effect in the formatting process. These commands are provided so that the user can invoke some program actions before they would otherwise be automatically invoked. They also allow the user to override the normal formatting procedures and force special actions which never occur during normal processing. All the commands in this group are fundamental in the sense that a text formatting program without them is hard to imagine.

/EL, End the current line. The last incomplete word (if any) and the current line (if it contains any words) is considered complete. This command will never produce a blank line. The line ended will not be justified. For example, the input stream "...text/EL,Next text..." will be output with "Next" as the first word on a new line.
/BL,n,   Output n blank lines. The current line is ended (see description of /EL). Then n blank lines are produced or the remainder of the page is left blank, which ever occurs first. "...line one /BL,2, new line..." will output two blank lines between the line ending with the word "one" and the line beginning with the word "new".

/PA,   Page eject. The current word and line are ended and the rest of the page is left blank down to the optional page footer. The top and bottom margins are specified by setting traps (see the /DT command). The /PA command has the same effect as a /BL,100. Blank lines are output until a trap occurs whose corresponding macro contains the /IP command described next. An example of page header and footer generating macros is given after the /DM command is covered. To place "LAST" as the last word of normal text on the current page and "FIRST" as the first word on the next page, use "LAST/PA,FIRST".

/IP,   Initialize new page. The current line is ended, the page number is incremented, and the line count is reset to one (thereby causing the next line to be printed at the top of the next page). This command should be used at the end of the macro responding to the bottom of page trap. Once the decision is made to use traps and macros to place the header and footer on a page, this command is needed to force the next page. An example of its use follows the discussion of the /DM command.

/PN,n,   Set the page number to n. The page number is incremented each time the /IP command is invoked. This command is used to reset the page number to any integer value. The page number is held in
user register number one so that the insert register command (/IR) can be used to place the page number into the text. The effect of the /PN,n, command is identical to the effect of the /SR,l,n command but should be more convenient to use. To set the current page number to 99, use /PN,99, before the last line on the page is reached.

/CE,n, Center the next n lines after ending the current line. Lines are centered between the left and right margins and are never justified. If a trap occurs before all n lines have been output, any text generated by the trap's macro is not centered (unless the macro also contains a /CE command) and, when the trap completes, centering of the remaining lines will continue. Although the same formatting function can be done with the /-JU,/-CO, commands, the center command is included for user convenience. Note that each line to be centered is usually ended with the /EL command rather than waiting for the program to fill to the right margin. The input stream "last non-centered line./CE,2,First centered line. /EL, Second centered line/EL, Following non-centered lines." will be printed as:

...last non-centered line.
    First centered line.
    Second centered line.
Following non-centered lines...

/TA,n, Tab to column n. The next character of text will be placed in column n (absolute column number, not relative to the left margin). If a word is tabbed to the right of the right margin, it will be placed at the left margin on the next line. If it is tabbed to the left of the current word, the current line is ended and the next character is placed in column n of the following line. In every case, the tabbed word and all words to the left of the tabbed word are not
justified. One possible use: if the left margin is set to column 11 and the first word of a paragraph is to be indented 8 spaces, first end the current line and then tab "/EL,/TA,18," and follow with the first word of the paragraph.

**Format Control**

The commands in the remaining sections change the default values of the static state variables which affect the general format of each page.

/\LM,n, Set the left margin to print column n (absolute). Each new line will automatically begin in position n after the current line has ended. The /TA command may be used to start a line to the left or right of the left margin. The value of n must be greater than one (column one is used for the carriage control character) and less than the value of the right margin. Default value is 2. For example, "/LM,10," will set the left margin to column 10.

/\RM,n, Set the right margin to print column n (absolute). No character will be printed to the right of this column. If the lines are being justified (see /JU), the rightmost character in the line will be placed in column n. The value of n must be greater than the value of the left margin and less than or equal to 132. The default is 80.

/JU, Justify the words on each line by inserting spaces between them until the first word starts in the left margin and the last word ends in the right margin. Blanks are inserted right to left on even numbered lines and from left to right on odd numbered lines. This will distribute the "white space" on the page. Readability is
enhanced by making no more than one change in the word spacing on each line. Default is to justify the lines.

/-JU,  Do not justify the words on each line. Only one space will be placed between words (see also /CO) and the last character on the line will not necessarily be placed on the right margin. All of the text in the input stream between the /-JU, and the /JU, commands will not be justified.

/CO,  Compress extra blanks between words in the input stream. This feature is especially useful for discarding the trailing spaces in each input record making it unnecessary to split the last word in each record. Default is to compress extra blanks.

/-CO,  Do not compress extra blanks. The exact number of spaces between each pair of words in the input stream will appear on the output if the line justification is also turned off. If the justification is turned on, more blanks may be placed between words but never less blanks. All text appearing between the /-CO, and the /CO, commands will not have the extra spaces removed. For example, if the input contains 80 character card image records and the output is to be an exact listing of the cards, the number of columns between the left and right margins must be equal to 80 and both justification and compression must be turned off: "/LM,2,/RM,81,/JU,/-CO,". Make sure that the last comma following the "/-CO," is in column 80 of the input record so that the first character of the next record will be placed on the left margin of the first line of output (otherwise the trailing blanks will be placed into the first line).
/BF, Bold face all characters of text following this command. Bold face is accomplished by over-printing each character three times. Default is to not over-print.

/-BF, Do not bold face characters. All characters between the /BF, and /-BF, commands will be over-printed. For example, if the "b" in "abc" is to be bold face, use "a/BF, b/-BF,c" without a space between any characters which would delimit a word.

/DT,n,xx, Define a trap. An entry is made in the trap table to contain the line number n and a pointer to the macro named xx (only the first two characters of the name are used). The macro xx must have been previously defined by using the /DM command. Each time line number n is started on the output page, the macro named xx will be automatically invoked. The trap is ended when the /EM command stored at the end of macro xx is executed (of course, macro xx may invoke many other macros first). An example of setting traps to output the page header and footer is given after the /DM command is described.

/DT,n,0, Delete trap. When the name of a macro is input as numeric zero, the entry for line n is deleted from the trap table.

/ET, End trap. Users have no use for this command. The end trap command is used by the program to mark the place in the stack of invoked macros where the trap was invoked. Since "/ET" is not in the user's macro, any macro can be used normally or for a trap.

User Variables

Fifty "registers" are available for user access. Register number one is used to hold the current page number by the program.
The remaining registers may be used for any purpose (such as holding chapter, section, or paragraph numbers).

/SR,n,m,  Set register number n to the integer value m. The range of n is from 1 to 50. The range of m depends on the word size of the computer. To set the current page number to 19, use "/SR,1,19," (or use the /PN command).

/AD,i,j,k,  Add registers. Set the value in register i to the sum of the values in registers j and k. No check is made to see that registers j and k contain valid values. To add the value 2 to the value in register 10, the 2 must first be loaded into some register, say 7: /SR,7,2, /AD,10,10,7,.

/SU,i,j,k,  Subtract registers. The value in register i is set to the value in register j minus the value in register k.

/IR,n,k,  Insert register value into the text. The content of register n is converted from an integer value to a string of characters and inserted into the text. A minimum of k characters will be inserted and leading spaces will be added when necessary. If k is zero, only the number of characters necessary to represent the value in register n will be inserted. This command does not delimit a word so the value inserted becomes part of the current word. For example, if registers 2, 3, and 4 contain 6, 5, and 4 respectively, a paragraph number could be generated by "/IR,2,0,/IR,3,0,/IR,4,0," and would print as "6.5.4".
User Defined Macro

/EM,   End macro definition. Every macro command must be ended with this primitive command. While a macro is supplying the input characters, this command causes the macro to be terminated and the stack of invoked macros to be popped. When defining a new macro (with the /DM command), this command is seen as the last four characters to be stored as part of that macro.

/DM,xx,text of macro/EM,   Define macro command. The text of the new macro is stored character by character until the string "/EM," has been stored. Thereafter, a command of "/xx" will interrupt the input stream and characters will be fetched from the storage area holding macro named xx. Only the first two characters of the name are used and they must be different from the first two characters of any other command name. A macro may be redefined at any time but the storage area holding the old definition will not be reused. Up to 20 macro names may be defined and a total of 3000 characters of macro text may be stored. Macros may invoke other macros for a maximum nesting depth of 50.

This section ends with an example showing how a page header and footer may be specified. A macro must be defined to contain the text and embedded commands necessary to print the header. If the header is to be "FORMATTING PROGRAM" centered on line one and followed by 2 blank lines, we can write a macro named "HD" as follows:

/DM,HD,/CE,1,FORMATTING PROGRAM/BL,2,/EM,

In order to get the HD macro invoked at line one on each page, we
must define a trap for line one: /DT,1,HD.

The footer is handled in the same way. If the last line of normal text on each page is to be printed on line 55, lines 56 and 57 are to be blank, and the word "PAGE" followed by the page number is to be centered on line 58, define the footer macro as follows:

/DM,BOTTOM,/BL,2,/CE,1, PAGE /IR,1,0,/IP,/EM,

Several points should be noted. Only the first two letters of the name will be used by the program. The /BL command forces the blank lines. The user has access to the page number through register one. The insert register command is used to place the page number following the word "PAGE". The page number is centered (along with the word "PAGE") by using the /CE command. The last action of this macro is to initialize the next page by invoking the /IP command which ends the line, increments the page number (in register one), and sets the current line number to one (which will cause a trap for the header macro when the footer trap has ended). Finally, a trap must be defined to invoke the BOTTOM macro when line 55 is reached: /DT,55,BOTTOM.

Note that nothing in the macro definition specifies the line number. The macro is relocatable.

A macro must be defined before the trap which references it is defined. The header and footer macros as well as both trap definitions need appear only once at the front of the input file. Either the header or footer may be changed at any time by either redefining the macros or by setting traps to point to different macros.

If a footer is not wanted, a trap must still be defined to invoke a macro which contains the /IP command or the carriage control
character for top of form will never be sent to the line printer. For example, to place 60 lines of text on each page without either a page header or footer, a macro and trap can be defined as:

/DM,EJECT,/IP,/EM, /DT,61,EJECT,

once, at the front of the input file. This, in effect, has set the bottom margin to line 60 and the top margin to one.
Section 3

OVERVIEW OF PROGRAM

The text formatting program is made up of eight routines. Figure 1 shows the general program structure and data flow. The function of each routine is summarized as follows:

DRIVER. The main program. It obtains characters of input, one at a time, and drives the rest of the routines.

INTRP. The command interpreter. This routine is called to process every command encountered in the input stream.

BUILDL. Builder of formatted lines. Called to place one character of text into the current line, to end the current line, or to produce a blank line of output. Writes completely formatted lines to a work file.

CKTRAP. The trap invoker. Called to check the current line number to see if it should invoke a macro command.

GETCH. Returns one character of input to the caller. The character is obtained from either the macro command storage area or from the most recently opened input file. Returns the NIL character when all input files are closed.

GETFLD. Returns the value of the next field within a command. Called only by the command interpreter.

MSTACK. Handles the stack of invoked macro commands.

ERRMSG. Handles the processing of all error messages.
FIGURE 1

PROGRAM STRUCTURE

CONTROL AND DATA FLOW
Input to the program is a stream of characters which are the text of a document interspersed with embedded formatting commands. The DRIVER defines all common data storage areas and initializes all of the program variables. It then loops to obtain and process each character of input one at a time. When all input characters have been processed, the DRIVER will rewind the work file which contains the formatted text, and copies it to the system printer before terminating.

Before the DRIVER obtains each character of input, it checks to see if BUILDL is about to start a new line of output. If so, a call is made to CKTRAP so the new line number can be checked to see if a trap should occur. There is a trap list which contains user defined pairs of line numbers and corresponding macro command pointers. If a line is about to be started whose number is in this list, CKTRAP suspends normal processing and calls MSTACK to invoke the required macro. Page headers and footers are handled by traps. Invoking a macro simply amounts to pushing the currently invoked macro (if any) onto the stack of invoked macros and setting a pointer for GETCH so that the next character of input will be the first character of the new macro. Control is returned to the DRIVER whether or not a trap was executed.

Next, the DRIVER checks to see if the user has requested blank lines be inserted into the text. If so, the NIL character is sent to BUILDL which then decrements the count of blank lines requested and writes one blank line to the work file. Only one line is output before control is returned so that a trap check can be made for each line.

The DRIVER obtains each character of input by calling GETCH. There are two sources of input, the storage area containing macro
commands previously defined by the user and the input files. The input sources are hierarchical. On the bottom level is the system input file (card reader) initialized to open by the DRIVER. The user may interrupt this file at any place by opening another input file. Up to three files may be opened at any one time. Only the most recently opened file may be closed and the input is resumed from the last interrupted file. In addition to one file interrupting another, the current input file can be interrupted by the user invoking a macro command. Input is then obtained from the storage area which contains that macro. When the end-macro command (which is at the end of each macro) is executed by the command interpreter, the input is continued from this file. And finally, a macro may contain a command to invoke another macro which will then be executed before the invoking macro is resumed. Up to 50 macros may be on the invoked macro stack.

No matter where GETCH gets the next character from, the DRIVER will take one of three actions based on that character:

Case 1. The NIL character received. This is the signal sent by GETCH that there are no further input characters. The DRIVER will rewind the work file generated by BUILDL and copy it to the system printer before terminating the program.

Case 2. The start-of-command character is received. The DRIVER calls the command interpreter which will obtain the rest of the characters in the command itself by calling GETCH. If the command is one of the primitive built-in commands, it is executed immediately. If the command is the name of a stored macro, MSTACK is called to invoke it. In either event, control is returned to the DRIVER. Some of the
primitive commands send characters to BUILDL in order to force an end
of the current line, to produce a blank line, or to have the start-of-
command character printed as text. The insert-register command causes
the interpreter to convert the integer value in one of the registers to
a string of characters which are then inserted into the text by calling
BUILDL for each character in the string. This allows for the numbering
of pages as well as for writing macros to automatically number chap-
ters, sections, or paragraphs. No more than one word is passed to
BUILDL from the interpreter before a return is made to the DRIVER
(there must never be more than one word left over from the previous
line when a trap occurs).

Case 3. Any other character is received. The DRIVER will pass
the character to BUILDL as the next character of text.

Subroutine BUILDL takes each character of text passed to it and
first builds words (a sequence of non-blank characters). When the word
just completed will either just fill the current line or must become
the first word on the next line, BUILDL will adjust the position of
each word on the line as required. If any of the characters on the
line are to be over printed, a line containing the bold face characters
is written to the work file. Then the line of normal characters is
written.

When BUILDL receives the NIL character, it terminates the cur-
rent line. If the current line was empty, BUILDL checks the count of
requested blank lines. When the count is greater than zero, a blank
line is written and the count is decremented. If the count was zero,
the call is ignored. When the NIL is received and the current line is
not empty, BUILDL will simply terminate the line and will not check for blank line requests (it must return control to the DRIVER after each line so that a check for a trap may be made).
Section 4

DESCRIPTION OF ROUTINES

The source code for each routine contains many comments to make the purpose of each group of statements clear. Statement numbers are in increasing order within each routine. Control during program execution flows from the top of the listing to the bottom unless it is clearly indicated otherwise.

Main Program, DRIVER

The main program defines all common storage areas and initial values for all variables are set. It then loops until all input has been processed. The loop consists of three separate actions. First a check is made to see if the number of the current line has changed since the last trip through the loop. If so, CKTRAP is called to check the trap list and invoke a trap if the current line number is found. Next, the DRIVER checks to see if the user has requested one or more blank lines to be generated. If a blank line is needed, the DRIVER sends the NIL character to BUILDL which produces one line of output. The last action in the loop is only carried out when a blank line was not generated. The DRIVER obtains the next input character by calling GETCH. The source of this character can be any of the macro commands previously stored or the most recently opened input file. The DRIVER will take one of three actions depending on the character received:
Case 1. The NIL character received. This is the signal from subroutine GETCH that all input files are closed. The DRIVER will branch out of the loop, rewind the work file written by BUILDL, and copy it to the system printer before terminating the program.

Case 2. The start-of-command character received. The DRIVER will call the command interpreter which will get the rest of the command itself. The interpreter executes the command before returning to the DRIVER.

Case 3. Any other character received. The DRIVER passes the character to the line formatter, BUILDL, which places it into the current line of text. When BUILDL fills a line, it will write it to a work file and start a new line before returning.

Note that no more than one line of output can be generated before the DRIVER regains control again. This is necessary to insure each line number is checked to see if a trap should be invoked. Having BUILDL write the formatted lines to a work file (for the DRIVER to copy after the end of the document is reached) facilitates adding a table of contents file and an index file at a latter time.

Subroutine INTRP, Command interpreter

The DRIVER calls this routine when it encounters the start-of-command character. There are two types of commands, primitives and macros. Primitive commands are built into the interpreter. The list of primitive commands is initialized by the DRIVER and each entry in the list contains the first and second characters of the command name. The index of each entry is used to branch to the code corresponding to
that command. Macro commands are segments of the input stream which have been stored. When the user defines a macro, an entry is made in the defined macro table. Each entry contains the user's two character name for the macro and a pointer to the place the first character of the macro is stored. The format of both types of commands is the same. The start-of-command character is followed by one or more fields. The first field must contain the name of the command.

When INTRP is called, it gets the name of the command by calling GETFLD. The list of primitive commands is searched and, if the name is found, a branch is taken to the code which will execute that command. If a match is not found, the list of defined macros is searched. If an entry with the command name is found, the corresponding pointer in the table is pushed onto the invoked macro stack by calling MSTACK. Otherwise, an error message is issued warning the user that a call was made for a macro not defined.

The code to execute each primitive follows the same general pattern. If the primitive has parameters, GETFLD is called to get the value of each field. Error checking is performed and error messages issued when necessary. The appropriate action is performed and one of two exits is made. If the last field of the command is numeric, the field delimiter is checked to see if it was a comma or blank. If not, the non-digit which delimited the field is passed to BUILDL as text. (This is a missing delimiter error recovery trick.) If the last numeric field was ended with a comma or blank, or the last field was not numeric, a normal exit is taken which does not use the delimiter as text.
The code used for each primitive should be clear from reading the description of the command and following the comments in the source listing. Commands to handle traps and macros are covered in the description of the routines which made use of these features.

**Subroutine BUILDL, Builder of Formatted Lines**

This routine is called to place one character of text into the current line, to end the current line, or to produce a blank line. BUILDL is coded in three sections. The first section looks at the input character and builds a table of words. The second section uses this table of words to format one line. The third section writes the formatted line to a work file and initializes to begin a new line.

Section one examines the input character which must fall into one of three classes: a blank, the NIL character, or a character of text (any character except the blank or NIL). When a character of text is received, it is stored in a work buffer, the count of the characters in the current word is incremented, and a check is made to be sure the word has not become longer than the distance between the margins. If the character is bold face, a bit is added to it before it is placed in the work buffer (details of bold face are given in the section on machine dependent code). Then BUILDL returns to the caller.

When a blank is received, a check is made to see if the current word has been terminated. If not, this blank is the word delimiter. Words are terminated by making an entry in the word table. This table is indexed by the word number and contains three columns: a pointer to the location of the first character of the word in the work buffer, the
lowest printer position the first character may be placed in, and a flag indicating whether the position of the word in the line can be changed. Before the first word is entered, the table is initialized so that the first column of the first item in the table points to the first character of the work buffer. The minimum print position of the first item is set to the left margin. When the first character of any word is received, the word count (table index) is incremented and the adjust flag in the table is set. When a word is terminated, a pointer to the next free location in the work buffer is placed in the word table for the next word (if any). The minimum print position column is also set for the next word so that one blank will be left between words (or two blanks if the last word ended with a period). After each word is terminated, a check is made to see if the current line is full. If not, BUILDL returns to the caller. When there are enough words in the word table to fill the current line, a branch is made to section two of BUILDL to format the line.

When a blank is received in section one and it does not end the current word, a check is made to see if the user requested blank compression. If not, the minimum print position for the next word in the table is incremented. If compression of blanks is required, the blank is ignored and BUILDL returns to the caller.

When the NIL character is received, the caller wants the current line ended or a blank line output. If there are any words on the current line, the last word is terminated (if necessary), the adjust flag for the last word is set to false, and BUILDL branches to section two to format the line. If there are no words in the word table, a
check is made to see if the user requested one or more blank lines to
be output. If not, the call is ignored and BUILDL returns. If some
blank lines are required, the count is decremented, the line buffer is
filled with blanks, and the program branches to section three to output
the line.

Section two (beginning with statement 100) formats the line by
making adjustments to the minimum print position column in the word
table and moving each character in the work buffer to its proper posi-
tion in the line buffer. When section two is entered, either all of
the words in the table will fit on the line or, at most, only one too
many words will be in the table. The extra word, if present, will be-
come the first word on the following line. First the number of blanks
necessary to fill the line is calculated. Then the word table is
examined to determine how many gaps precede words which may be adjusted
(search is made from right to left). If the line is to be centered,
the minimum position of each word in the table is increased by half the
number of padding blanks required. Otherwise, the blanks are added to
line by increasing the rightmost word's minimum position by the total
number of blanks, then going up the table making smaller increases to
positions. The position entry of the first word is not changed.
Blanks are distributed from the right on even numbered lines and from
the left on odd numbered lines. Random insertion of blanks was tried
but I found the uneven word spacing very hard to read. This method of
adjusting the line produces no more than one change in the word spacing
per line and still scatters the "white space" over the page.
Each character of each word to be placed on the line is moved from the work buffer to its print position in the formatted line buffer. The positions are determined by using the adjusted minimum position entry in the word table. A second formatted line buffer is used to hold any characters which are to be over-printed (bold face). For each character not to be bold faced, a blank is stored in this second buffer. After all words have been placed in the formatted line buffers, the program branches to section three to output the line and to initialize for the next line.

Section three of BUILDL (beginning with statement 200) writes the contents of the line buffers onto a work file which will be copied to the system printer by the DRIVER at the end of input processing. If the bold face line buffer contains any non-blank characters, it is written to the file first (the carriage control character was set to a plus sign in section two). Otherwise, it is not written. Then a check is made to see if this is line number one on a new page. If so, the carriage control character is set to 1, otherwise it is left as a blank. Then the formatted line buffer is written to the work file. Finally, section three checks to see if the last word in the word table was left off of the line. If it was not output, the last word is moved to the first word position in the table and its characters are moved to the first positions in the work buffer. If all words were output, the word table is initialized to begin a new line. Then BUILDL returns to the caller.

BUILDL has the most complex structure of any routine in this program. This is not only a result of the many varied requirements
placed on the formatting process but was also necessary to make the subroutine robust enough to recover from all types of user input errors. Note that no error messages are issued from this routine, any input is processed.

**Subroutine GETCH, Supply One Input Character**

GETCH is called when any routine needs the next character from the input stream. Characters come from two sources, the macro command storage area or from the most recently opened input file. When a macro command is invoked, a pointer into the current macro (if any) is pushed onto the invoked macro stack and that same pointer is set to point into the new macro (the macro stack is handled by MSTACK). GETCH will fetch one character from the macro area, increment the pointer, and return the character to the caller. Eventually the command interpreter will execute the end-macro command which has been stored at the end of every macro and the macro stack will be popped (by MSTACK). When the last macro is ended and the stack is empty, the pointer is set to zero and GETCH will get characters from the input files.

GETCH maintains a stack of open input files and a buffer save area. Currently up to three files may be open at one time. The first entry in the stack is initialized by the DRIVER to be the system input file. Each entry in the stack contains the file's unit number and a word to save the pointer into the active input buffer in the event another file is opened. The buffer save area is large enough to hold three physical records, one from each possible open file. A file is opened when the interpreter executes the open command. This causes
the current pointer into the buffer area to be placed in the file stack, the active buffer is saved in the area corresponding to the current stack level, and the unit number of the new file is pushed onto the stack. The pointer into the active buffer area is set to a large number thereby forcing a read before the next character is fetched. When a file is closed, the file stack is popped, the saved buffer is moved to the active buffer area, and the pointer is restored by using the value saved on the stack. Input will resume from the point it was interrupted.

As GETCH fetches characters from the active buffer, the buffer pointer is always left pointing to the next character to get. When the pointer points beyond the last character to use in the buffer (user specified), GETCH will attempt to read the next record into the buffer. If a record was read, the pointer is set to point at the first character to use from the record (also user specified) and then that character is fetched and returned to the caller. If an end-of-file was read, GETCH will automatically close the file by popping the stack. When all files are closed, GETCH will return the NIL character.

**Subroutine CKTRAP, Trap Invoker**

The DRIVER calls this routine each time a new line is started (at most, only the first word of the new line remains from the previous line). If a trap is already in progress, the call is ignored. This code is not reentrant and one trap is not allowed to interrupt another trap. If a trap is not being processed, the trap list is searched to see if it contains the current line number. The list contains pairs of
line numbers and pointers into the storage area for macro commands. Entries are made into this table by using the define trap command. If the line number is not in the table, a return is made to the DRIVER. When the line number is found a trap is invoked as follows:

Step 1. A pointer to a special predefined (and unnamed) macro is pushed onto the macro stack by passing MSTACK an argument on one. The first character of the special macro is placed in the first location of the macro storage area by the DRIVER during initialization. The macro contains only two instructions and no text, the end trap and macro commands (/ET,/EM,).

Step 2. The pointer to the macro we are to invoke is obtained from the trap table and passed to MSTACK to push it onto the macro stack.

Step 3. Some key variables used by BUILDL are copied into a save area. These variables include the first entry in the word table and all other information necessary for BUILDL to continue at this point when the trap is over.

Step 4. Finally, the key variables used by BUILDL are initialized to begin a new line and a return is made to the DRIVER.

Input will now be obtained from the macro storage area until the end of macro command is executed by the interpreter causing the macro stack to be popped. Now the special macro we pushed onto the stack comes into play. The end trap command it contains is executed:

Step 1. An end to the current line is forced by calling BUILDL with the NIL character.
Step 2. All the key variables used by BUILDL are restored from the save area and text processing may be resumed from the point at which it was interrupted.

When the end macro command that follows the end trap command in the special macro is executed, the macro stack is popped and we return to the DRIVER. Everything is like it was before the trap occurred except the output page may contain one or more new lines or the trap may have begun a new page.

**Subroutine MSTACK, Invoked Macro Stack Handler**

Any routine wishing to invoke a macro or wishing to end the execution of a macro does so by calling MSTACK with one argument. If the argument is zero, MSTACK will terminate the current macro. If any suspended macros are on the stack, the pointer into the macro storage area which is used by GETCH will be set to the top value on the stack and the stack will be popped. If MSTACK is called with a zero and no macro is on the stack, the pointer will be set to zero for GETCH's information.

A non-zero argument is assumed to be a pointer into the macro storage area for a new macro to be invoked. If there is not a macro currently executing, GETCH's pointer is simply set to the value passed to MSTACK. If a macro is executing (GETCH's pointer is non-zero), the value of that pointer is pushed onto the macro stack before it is set to the value passed to MSTACK. Up to 50 macro pointers may be nested.

**Subroutine GETFLD, Get One Field**

This routine is called by the command interpreter to obtain the value of the next field of the current command. A field may contain an
alphanumeric name (optionally preceded by a sign and beginning with a non-digit), a signed or unsigned integer, or a single start-of-command character. Alphanumeric and numeric fields are delimited by a comma or by a blank. In addition, the numeric field will be delimited by any non-digit character following the optional sign. The single start-of-command character is a special case which requires no delimiter.

GETFLD will return three arguments, IFLG, IAI1, and IAI2. The value these are set to depends on the type of field encountered.

Alphanumeric Field:

IFLG = 1 (if no preceding sign or sign was plus).
IFLG = -1 (if preceding sign was minus).
IAI1 = the first character in the field.
IAI2 = the second character in the field.

Numeric Field:

IFLG = 0
IAI1 = integer value of field.
IAI2 = non-digit character that delimited field.

Start-of-command is first character in field:

IFLG = 1
IAI1 = start-of-command character.
IAI2 = start-of-command character.

First, variable ISIGN is set to one and the variable which accumulates the integer value of a field is set to zero. GETCH is called to get the first character. If it is the start-of-command character, arguments are set as indicated above and the return is taken. If the first character is a sign, ISIGN is set to plus or minus one depending on the sign and the next character in the field is obtained from GETCH. Next, the character is checked to see if it is a digit. If so, the remaining digits in the field are converted to an integer value. This value is then multiplied by ISIGN and the values
of the arguments are set as indicated for a numeric field. When the first character beyond the optional sign is not a digit, it and the following characters are returned as the value of an alphanumeric field. All remaining characters in the alphanumeric field are skipped.

Note that the code used to convert characters to integer is machine dependent on the maximum value an integer may assume.

Subroutine ERRMSG, Error Message Handler

This subroutine provides a common point for all error messages to be processed. It is called with two arguments, the integer number of characters in the message and the message as a Hollerith string. ERRMSG will write the current page and line number to an error file (currently set by the DRIVER to be the system printer). The number of words in the message is calculated by dividing the number of characters by the number of characters per word and rounding upwards. The number of characters per word is machine dependent and is initialized by the DRIVER to be four for this machine. The message is then written to the error file.

Note that, in addition to the number of characters per word, the format statement itself is machine dependent and must be changed when a machine with a different integer word size is used. This routine can be made machine independent by placing all error messages directly in the formats and calling it with an error message number. I prefer to be able to read the messages in the routines which issues them. Since this routine handles all messages, no other routine needs to be changed due to using a different word size.
Section 5

MACHINE DEPENDENT CODE

One design goal when writing this program was to make it as portable as possible. Due to some of the limitations of the programming language and the variety of compilers and machines, some code is machine dependent. The following paragraphs describe the code which may have to be modified when moving from the CDC 3170 computer.

**Input/Output Code**

The system input file and all user files are implicitly opened when the first read is made. If files must be explicitly opened before use, that code must be added to the DRIVER for the system input file, and to INTRP for user input files. The two output files which may also need to be opened are the system print file and the work file used by the DRIVER and BUILDL.

The system supplied function IFEOF(unit) returns a value of minus one if an end-of-file was read or a zero otherwise. This function is called from the DRIVER and from GETCH.

The number of characters stored in one integer word affects the way error messages are stored. The variable CPW is initialized in the DRIVER to be four characters per word. The only routine using this variable is ERRMSG. The format statement in ERRMSG must reflect the value of CPW too.
**Integer Values**

The maximum value an integer variable can assume affects the code which converts characters in a numeric field to an integer value. There is no error check in GETFLD on the size of the integer being converted, the printed document will just not be printed as expected.

The code in INTRP which executes the insert register command must consider the largest integer value to generate the leftmost digit in the output string. Currently it is set for eight decimal digits.

**Bold Face Code**

The technique for flagging a character of text for over-printing depends on the bit representation of characters in an integer variable. The somewhat lengthy description that follows is an example of the kind of problem which sometimes make program conversion so time consuming.

The bold face request flag, BFFLG, is set to zero by the /-BF command and set to a constant, IBFBIT, by the /BF command. In order to save buffer space and to be able to use the same section of code for both bold face and normal characters, some unused bit in the word containing each character is used as a bold face flag bit. The easiest method of doing this in Fortran is outlined below:

\[
\begin{align*}
\text{IBFBIT} & = \text{constant containing only the flag bit.} \\
\text{BFFLG} & = 0 \text{ if not bold face.} \\
\text{BFFLG} & = \text{IBFBIT if bold face.} \\
\text{ICH} & = \text{the current character of text.}
\end{align*}
\]

To merge the bold face flag bit:

\[
\text{ICH} = \text{ICH .OR. BFFLG \ (for all text characters).}
\]
To test for the presence of the bold face flag bit:

\[
\text{IF((ICH.AND.IBFBIT) .NE. 0) Process as bold face.}
\]

To remove the flag bit before printing the character:

\[
\text{ICH = ICH .AND. IMASK where IMASK contains every bit except the one bit in IBFBIT.}
\]

The Fortran compiler used on the CDC 3170 computer at this University does not allow the logical operators AND and OR to be used on integer variables. It also does not contain the built-in functions IAND and IOR. Therefore, the bold face flag bit handling is done with integer arithmetic to avoid having to include assembly language routines to perform the logical operations.

Text is stored left justified, one character per word. The rightmost bit is used for the flag and its state is changed simply by adding one to the word and the bit may be tested by using the MOD function. Unfortunately, the computer uses a one's complement representation for integers. The left most bit (which is part of the character) is the sign bit and the rightmost bit is zero for positive even numbers and one for even negative numbers. Therefore a combination of the MOD and the ISIGN (transfer of sign) functions must be used to test the bold face flag bit. The code used is as follows:

To merge the bold face flag bit:

\[
\text{ICH = ICH + BFFLG}
\]

To test for the presence of the bold face flag bit:

\[
\text{IF((MOD(ICH,2)*(-2)+ISIGN(1,ICH).EQ.IMASK) process bold face.}
\]

To remove the flag bit before printing the character:

\[
\text{ICH = ICH - IBFBIT where IBFBIT = 1 and IMASK = -1}
\]
Section 6

CONCLUSIONS

This program is able to format many types of documents, is relatively easy to use, can recover from input errors, and is easily converted to run on different computer systems. The structure of this program is flexible enough to support experimentation with different primitive commands and many useful macro commands. While the primitive commands provided are not yet powerful enough to construct macro commands to perform every function in the list of features in current programs, they do provide for a majority of those functions.

The program appears to run correctly and is robust. During program development, the problems encountered were always accompanied with a clear indication of their source. Primitive commands were individually added without changing the program design. The same flexibility which allows for experimentation also allows for repair and errors have been easy to fix. The program is portable and was easily converted to run on a CDC 6400 computer system.

The program has not been available to users yet but the commands and operation have been explained to several potential users. They agree the program will be useful but feel that some of the commands are awkward, especially the macros required for the page header and footer. Most feel the commands would be easy to learn and to use.
Section 7

RECOMMENDATIONS

Several features could easily be added to this program.

(1) Commands that test the registers could be used to implement an IF-THEN-ELSE structure within the input stream. This ability would be used for generating paragraph numbers or placing the page number in a different position on odd and even pages.

(2) Commands to accumulate entries in a table of contents and in an index file could be added. Since the text is not copied to the printer until all input has been read, these files could be printed in their correct position within the document.

(3) Footnotes could be collected in a file for printing at the end of a document or a command could append a footnote to the macro which contains the page footer and, at the same time, lower the line number of its trap.

(4) The formatting routine could be changed to do proportional spacing for a specific printer.

The text formatting program that is finally accepted as useful by the users may be different from what has been described in this report. However, the feedback from writers needed to develop a workable system can come only after the users have something to critique and this program can be used for that purpose.
REFERENCES


   Sales Brochure: Wordstream System.

18. Lextron Corp., Chatsworth, CA.
   Sales Brochure: Videotype 900 System.

19. Linolex Systems Inc. (subsidiary of 3M Company), North Billerica, MA.
    Sales Brochure on Word Processing Systems.

20. NBI, Boulder, CO.
    Sales Brochure: System II.

21. Qantel Corp., Hayward, CA.
    Sales Brochure on Word Processing Systems.
Appendix A

PROGRAM LISTING
PROGRAM DRIVER

TEXT FORMATTING PROGRAM
INPUT = STREAM OF TEXT WITH EMBEDDED COMMANDS.
OUTPUT = DOCUMENT PRINTED ON LINE PRINTER.

GERALD D. CAHILL
CALIFORNIA STATE UNIVERSITY, NORTHRIDGE
GRADUATE PROJECT - JUNE 1979
ADVISOR - DR. PHILIP GILBERT

MAIN PROGRAM. GET INPUT CHARACTERS ONE AT A TIME
AND DRIVE THE REST OF THE ROUTINES.

VARIABLES IN COMMON/ERRM/ ERROR MESSAGE INFO.
CPW NUMBER OF CHARACTERS PER INTEGER WORD.
ERFILE UNIT NUMBER OF ERROR MESSAGE FILE.

VARIABLES IN COMMON/ISTUFF/ INPUT FILE INFO.
IBUF(IMAX*MAXREC) ACTIVE AND SAVED INPUT BUFFERS.
IMAX MAX NUMBER OF INPUT FILES OPEN AT ONE TIME.
ISTK(IMAX,2) THE OPEN INPUT FILE STACK.
IOP POINT TO TOP OF OPEN INPUT FILE STACK.
IWORD POINTER TO NEXT CHAR TO FETCH FROM INPUT BUFFER.
LREC LENGTH OF PHYSICAL INPUT RECORD.
MAXPOS LAST CHAR POSITION TO USE FROM INPUT RECORD.
MAXREC MAX LENGTH OF PHYSICAL RECORD.
MINPOS FIRST CHAR POSITION TO USE FROM INPUT RECORD.

VARIABLES IN COMMON/MACH/ SPECIAL CHARACTERS.
ICOM THE START-OF-COMMAND CHARACTER.
NIL CHARACTER TO FLAG SOME ACTION.

VARIABLES IN COMMON/MACROS/ MACRO COMMAND INFO.
LASTMC LAST LOCATION IN USE IN MDEF.
MAXDEF MAX NUMBER OF CHAR S POSSIBLE IN MDEF.
MDEF(MAXDEF) STORAGE AREA FOR MACRO COMMANDS.
MMAX MAX SIZE OF MACRO STACK.
MPLIST(MPMAX,3) TABLE OF MACRO DEFINITIONS.
MPMAX MAX NUMBER OF MACRO DEFINITIONS ALLOWED.
MPTCP THE NUMBER OF MACRO NAMES STORED IN MPLIST.
MSTK(MMAX) THE INVOKED MACRO STACK.
MTOPO POINTER TO TOPOF INVOKED MACRO STACK.
MWORD POINTER TO NEXT CHAR TO FETCH FROM MACRO AREA.

VARIABLES IN COMMON/PRIM/ PRIMITIVE COMMAND INFO.
PLIST(PTOP,2) LIST OF PRIMITIVE COMMAND NAMES.
C PTOP  NUMBER OF PRIMITIVE COMMANDS.
C VARIABIES IN COMMON/PSTUFF/ FORMATTING INFO.
C ADJ (MAXCOL/2+2) ADJUST FLAG IN WORD TABLE.
C ADJFLG USER SET ADJUST WORDS ON LINE FLAG.
C BBFLG USER SET BOLD FACE FLAG.
C BFLINE(MAXCOL) FORMATTED LINE BUFFER FOR BOLDFACE CHARS.
C CENTER NUMBER OF LINES TO CENTER.
C COMFLG USER SET COMPRESS BLANKS FLAG.
C IBFBIT BIT TO MERGE WITH EACH BOLDFACE CHARACTER.
C ICC CARRIAGE CONTROL CHAR FOR SINGLE SPACE.
C ICCBF CARRIAGE CONTROL CHAR FOR BOLDFACE LINE.
C ICCFG CARRIAGE CONTROL CHAR FOR PAGE EJECT.
C IMASK VALUE FOR BOLDFACE CHAR TEST.
C IRM RIGHT MARGIN COLUMN NUMBER.
C ITEXT UNIT NUMBER OF WORK FILE USED BY (BUILDL).
C LASTCH REMEMBER THE LAST CHAR OF TEXT PROCESSED.
C LBUF(MAXCOL*2) WORK BUFFER FOR (BUILDL).
C LINE(MAXCOL) FORMATTED LINE BUFFER, NON-BOLDFACE CHARS.
C LINENO NUMBER OF THE CURRENT LINE.
C LM LEFT MARGIN COLUMN NUMBER.
C LSAVE(MAXCOL+16) SAVE AREA FOR (BUILDL) KEY VARIABLES.
C USED DURING A TRAP.
C LWORD NUMBER OF WORDS IN THE WORD TABLE.
C MAXCOL MAX NUMBER OF PRINT POSITIONS PER LINE.
C MAXWD NUMBER OF CHARS BETWEEN MARGINS (MAX WORD SIZE).
C MINCOL(MAXCOL/2+2) MIN POSITION COLUMN IN THE WORD TABLE.
C NBL NUMBER OF BLANK LINES REQUESTED BY USER.
C NCH NUMBER OF CHARS IN LAST, UNCOMPLETED WORD IN THE WORD TABLE.
C NEXTCH POINTS TO NEXT AVAILABLE LOCATION IN WORK BUFFER.
C NEXTMC NEXT VALUE TO USE IN WORD TABLE FOR MIN POSITION.
C REG (REGTOP) USER REGISTERS.
C REGTOP THE NUMBER OF USER REGISTERS.
C START(MAXCOL/2+2) ENTRY IN WORD TABLE POINTING TO FIRST CHARACTER OF WORD (IN WORK BUFFER).
C TABFLG USER SET TAB FLAG.
C PAGENO (EQU REG(1)) NUMBER OF CURRENT PAGE.
C VARIABIES IN COMMON/TRAPS/ TRAP INFORMATION.
C TLIST(TMAX/2) THE TABLE OF DEFINED TRAPS.
C TMAX MAX NUMBER OF TRAP DEFINITIONS.
C TRAFFL TRAP IN PROGRESS FLAG (IF NOT ZERO).
C TTOP NUMBER OF TRAP DEFINITIONS IN TLIST.
C IMPLICIT INTEGER (A-Z)
INTEGER ERFILE,CPW
INTEGER PLIST,PTOP
INTEGER ADJ,ADJFLG,BFFLGBKLINE,CENTER,COMFLG,REG
INTEGER REGTOP,START,TABFLG,PAGENO)
INTEGER TLIST,TMAX,TRAPFL,TTOP
INTEGER SYSOUT

COMMON/ERRM/ERFILE,CPW

DIMENSION IBUF(IMAX*MAXREC),ISTK(IMAX,2)

COMMON/ISTUFF/IBUF(270),IMAX,ISTK(3,2),ITOP,IWORD,
1 LREC,MAXPOS,MAXREC,MINPOS

COMMON/MACH/ICOM,NIL

DIMENSION MDEF(MAXDEF),MPLIST(MPMAX,3),MSTK(MMAX)

COMMON/MACROS/LASTMC,MAXDEF,MDEF(3000),MMAX,
1 MPLIST(20,3),MPMAX,MTOP,MSTK(50),MTOP,MWORD

DIMENSION PLIST(PTOP,2)

COMMON/PRIM/PLIST(26,2),PTOP

DIMENSION ADJ(MAXCOL/2+2),BFLINE(MAXCOL),
1 LBIF(MAXCOL*2),LINE(MAXCOL),LSAVE(16+MAXCOL),
C MINCOL(MAXCOL/2+2),REG(REGTOP),START(MAXCOL/2+2)

COMMON/PSTUFF/ADJ(68),ADJFLG,BFLINE(MAXCOL),132),CENTER,
1 COMFLG,IBFBIT,ICCCIBF,ICC,ICCPG,IMASK,IR,M,ITEXT,
2 LASTMC,LIBF(264),LINE(132),LINENO,LM,LSAVE(148),
3 LWORD,MAXCOL,MAXWD,MINCOL(68),VBL,NCH,NEXTMC,
4 NEXTMC,REG(50),REGTOP,START(68),TA3FLG

EQUIVALENCE (PAGENO,REG(1))

DIMENSION TLIST(TMAX,2)

COMMON/TRAPS/TLIST(4,2),TMAX,TRAPFL,TTOP

INITIALIZE VARIABLES IN COMMON ERRM.

ERFILE IS THE UNIT NUMBER OF THE ERROR MESSAGE FILE
SET TO SYSOUT.

ERFILE=61
CPW=4

INITIALIZE THE VARIABLES IN COMMON ISTUFF.
START WITH FILE 1 OPEN (THE SYSTEM INPUT FILE)
BUT WITH AN EMPTY INPUT BUFFER.
SET IWORD TO FORCE GETCH TO DO A READ.

IMAX=3

ISTK(1,1) MUST BE INITIALIZED TO THE SYSTEM INPUT
UNIT NUMBER.

ISTK(1,1)=60
ITOP=1
INITIALIZE VARIABLES IN COMMON MACROS

LASTMC=8
MAXDEF=3000
MMA=50
MPMAX=20
MPTOP=0
MTOP=0
MWORD=0
MDEF(1)=ICOM
MDEF(2)=1HE
MDEF(3)=1HT
MDEF(4)=1HL
MDEF(5)=1COM
MDEF(6)=1HE
MDEF(7)=1HM
MDEF(8)=1H,

INITIALIZE COMMON PRIM, THE TABLE OF PRIMITIVE COMMANDS. THE ORDER OF NAMES MUST AGREE WITH THE ORDER OF HANDLERS IN SUBROUTINE INTRP.

PTOP=26
PLIST(1,1)=1HD
PLIST(1,2)=1HM
PLIST(2,1)=1HE
PLIST(2,2)=1HM
PLIST(3,1)=1HO
PLIST(3,2)=1HP
PLIST(4,1)=1HC
PLIST(4,2)=1HL
PLIST(5,1)=1PD
PLIST(5,2)=1HT
PLIST(6,1)=1HE
PLIST(6,2)=1HT
PLIST(7,1)=1HE
PLIST(7,2)=1HL
PLIST(8,1)=1HB
PLIST(8,2)=1HL
PLIST(9,1)=1HP
PLIST(9,2)=1HA
PLIST(10,1)=1HL
PLIST(10,2)=1HM
PLIST(11,1)=1HR
PLIST(11,2)=1HM
PLIST(12,1)=1HI
PLIST(12,2)=1HP
PLIST(13,1)=1HB
PLIST(13,2)=1HF
PLIST(14,1)=1HT
PLIST(14,2)=1HA
PLIST(15,1)=1HC
PLIST(15,2)=1HE
PLIST(16,1)=1HR
PLIST(16,2)=1HL
PLIST(17,1)=1HF
PLIST(17,2)=1HC
PLIST(18,1)=1HL
PLIST(18,2)=1HC
PLIST(19,1)=1HC
PLIST(19,2)=1HO
PLIST(20,1)=1HJ
PLIST(20,2)=1HU
PLIST(21,1)=1HS
PLIST(21,2)=1HC
PLIST(22,1)=1HP
PLIST(22,2)=1HN
PLIST(23,1)=1HS
PLIST(23,2)=1HR
PLIST(24,1)=1HA
PLIST(24,2)=1HD
PLIST(25,1)=1HS
PLIST(25,2)=1HU
PLIST(26,1)=1HI
PLIST(26,2)=1HR

C
C INITIALIZE VARIABLES IN COMMON PSTUFF.
C
ADJFLG=1
BBFLG=0
CENTER=0
COMFLG=1
IBFBIT=1
ICC=1H
ICCBF=1H+
ICCPG=1H1
IMASK=-1
IRM=80
C
C ITEXT IS THE UNIT NUMBER OF A SCRATCH FILE USED
C FOR FORMATTED LINES OF TEXT.
C
ITEXT=4
LASTCH=0
LINENO=1
LM=2
LWORD=0
INITIALIZE THE VARIABLES IN COMMON TRAPS.

TMAX=4
TRAPFL=0
TTOP=0

INITIALIZE SOME LOCAL VARIABLES.

SYSOUT IS THE UNIT NUMBER OF THE SYSTEM PRINTER.

SYSOUT=61
LASTNO=1

DRIVER IS ONE LOOP FOLLOWED BY A WINDUP SECTION ENTERED WHEN ALL OF THE INPUT HAS BEEN CONSUMED.

CHECK FOR A TRAP IF WE ARE STARTING A NEW LINE.

1 IF(LINENO.NE.LASTNO) CALL CKTRAP

CHECK TO SEE IF WE SHOULD FORCE A BLANK LINE.

IF NOT, GO TO 20.

IF(NBL.EQ.0) GO TO 20

IF BUILDL HAD A WORD LEFT OVER AFTER AN END OF LINE COMMAND, IT REQUESTS ANOTHER NIL BY SETTING NBL=-1. SO RESET NBL TO ZERO.

IF(NBL.LT.0) NBL=0

SET FLAG TO FORCE BLANK LINE.

ICH=NIL
GO TO 30

NO BLANK LINE NEEDED. GET NEXT INPUT CHARACTER.

20 CALL GETCH(ICH)

IF NO DATA, GO TO 90 TO WINDUP THIS PROGRAM.

IF(ICH.EQ.NIL) GO TO 90

IF CHARACTER IS A START OF COMMAND, GO TO 40 TO CALL THE INTERPRETER.

IF(ICH.EQ.ICOM) GO TO 40

OTHERWISE, PUT THE CHARACTER INTO THE CURRENT LINE.

30 LASTNO=LINENO
CALL BUILD(I CH)

CALL BUILD(NIL)
END FILE ITEXT

CALL BUILD(L(NIL)

COPY FILE ITEXT TO SYSOUT.
REWIND ITEXT

READ(ITEXT,203) (BFLINE(I),I=1,MAXCOL)

IF EOF, GO TO 220
IF(IEOF(ITEXT).EQ.(-1)) GO TO 220

IF THIS IS NOT A BOLDFACE LINE, OUTPUT IT HERE.
IF(BFLINE(I).EQ.1CCBF) GO TO 210

WRITE(SYSOUT,203) (BFLINE(I),I=1,MAXCOL)
GO TO 201

WE JUST READ A BOLDFACE LINE.
NOW GET THE LINE OF NON-BOLDFACE TEXT.

READ(ITEXT,203) (LINE(I),I=1,MAXCOL)

WRITE THE LINE OF TEXT, THEN OVERPRINT 3 TIMES WITH THE BOLDFACE LINE.
WRITE(SYSOUT,203) (LINE(I),I=1,MAXCOL)
DO 215 K=1,3

WRITE(SYSOUT,203) (BFLINE(I),I=1,MAXCOL)
GO TO 201

WE READ AN EOF FROM FILE ITEXT, EJECT LAST PAGE.

WRITE(SYSOUT,221)

STOP
END
SUBROUTINE INTRP

C COMMAND INTERPRETER. CALLED BY THE DRIVER WHEN IT ENCOUNTERS THE START-OF-COMMAND CHARACTER. GET THE REST OF THE COMMAND AND EXECUTE IT.

C IMPLICIT INTEGER (A-Z)
INTEGER PLIST,PTOP
INTEGER ADJ,ADJFLG,BFFLG,BFLINE,CENTER,COMFLG,REG
INTEGER REGTOP,START,TABFLG,PAGENO
INTEGER TLIST,ITOP,TRAPFL,TTOP
INTEGER SAVEC,SIGN,VAL

COMMON/ISTUFF/IBUF(270),IMAX,ISTK(3,2),ITOP,IWORD
LREC,LMAXPOS,LMAXREC,LMINPOS
COMMON/MACH/ICOM,NIL
COMMON/MACROS/LASTMC,MDEF,MDEF(3000),MMAX,
MPLIST(29,3),MPMAX,MPTOP,MSTK(50),MTOP,MWORD
COMMON/PRIM/PLIST(29,2),PTOP
COMMON/PSTUFF/ADJ(68),ADJFLG,BFFLG,BFLINE(132),CENTER,
COMFLG,BFBIT,ICC,ICCBF,ICCPG,IMASK,IRM,ITEXT,
LASTCH,IBUF(264),LINE(132),LINEQ,LM,LSAVE(148),
LWORD,MAXCOL,MAXWD,MINCOL(63),NBL,NCH,NEXTCH,
NEXTMC,REG(50),REGTOP,START(68),TABFLG
EQUIVALENCE (PAGENO,REG(1))
COMMON/TRAPS/TLIST(4,2),TMAX,TRAPFL,TTOP

INTEGER DIG(10)
DATA DIG/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
DATA IAME, IAMM/1HE,1HM/

C GET FIRST FIELD (CONTAINING THE COMMAND NAME).
CALL GETFD(IFLG,IA1,IA2)

C IF IFLG IS ZERO, WE GOT A NUMBER.
IF(IFLG.NE.0) GO TO 10
CALL ERRMSG(27,27HCOMMAND STARTED WITH NUMBER)
GO TO 999

C IF WE GOT A SECOND START OF COMMAND CHARACTER,
  TREAT IT AS TEXT.
10 IF(IA2.EQ.ICOM) GO TO 998

C IF WE HAVE A PRIMITIVE COMMAND,
  GO TO 100 WITH IP SET TO INDEX.
DO 21 IP=1,PTOP
   IF(PLIST(IP,1).NE.IA1) GO TO 21
   IF(PLIST(IP,2).EQ.IA2) GO TO 100
21 CONTINUE

C COMMAND IS NOT A PRIMITIVE. SEARCH THE MACRO TABLE.
DO 25 IM=1,MPTOP
IF(MPLIST(IM,1).NE.IA1) GO TO 25
IF(MPLIST(IM,2).EQ.IA2) GO TO 26
25 CONTINUE
C
THE COMMAND IS NOT EITHER A PRIMITIVE OR MACRO.
CALL ERRMSG(26,26HCALL FOR MACRO NOT DEFINED)
GO TO 999
C
WE FOUND A MACRO, PUSH THE OPEN MACRO STACK.
26 CALL MSTACK(MPLIST(IM,3))
GO TO 999
C
* * * * * * * * * * * * * * * * * * * * * * * * * *
C EXECUTE THE PRIMITIVE COMMAND POINTED TO BY IP
C (THE INDEX IN THE PLIST ARRAY).
C
DM EM OP CL DT ET EL BL PA LM RM
IP BF TA CE RL FC LC GO JU SC PN
C
100 GO TO (110,140,150,160,170,190,200,210,220,230,240,1
250,260,270,280,290,300,310,320,330,340,350,2
360,370,380,400), IP
C
* * * * * * * * * * * * * * * * * * * * * * * * * *
C DM - DEFINE MACRO (IP=1).
C
GET NAME OF MACRO BEING DEFINED.
110 CALL GETFLD(IFLG,IA1,IA2)
C
DONT ALLOW NUMERIC MACRO NAMES.
IF(IFLG.NE.0) GO TO 112
CALL ERRMSG(29,29HMACRO NAME MAY NOT BE NUMERIC)
GO TO 998
C
SEE IF NAME IS ALREADY DEFINED.
112 DO 113 IM=1,MPTOP
IF(MPLIST(IM,1).NE.IA1) GO TO 113
IF(MPLIST(IM,2).EQ.IA2) GO TO 116
113 CONTINUE
C
MACRO NOT DEFINED, ADD IT TO THE END OF THE LIST.
IF(MPTOP.LE.MPMAX) GO TO 114
CALL ERRMSG(26,26HTOO MANY MACRO DEFINITIONS)
GO TO 999
C
114 MPTOP=MPTOP+1
IM=MPTOP
MPLIST(IM,1)=IA1
MPLIST(IM,2)=IA2
C
ADD TEXT OF MACRO TO MDEF ARRAY.
C SAVE CURRENT POINTER TO LAST CHARACTER
JUST IN CASE THE TEXT DOES NOT FIT IN MDEF.
116 MCSAVE=LASTMC
C
C SET POINTER WORD IN PLIST TO POINT TO THE FIRST
C CHARACTER OF THE MACRO (IN MDEF).
LASTMC=LASTMC+1
MPLIST(IM,3)=LASTMC
C
C SET STATE OF /EM DETECTOR TO ONE.
ISTOP=1
C
C LOOP TO STORE TEXT OF MACRO INTO MDEF ARRAY.
117 IF(LASTMC.LE.MAXDEF) GO TO 124
C
C NOT ENOUGH ROOM LEFT IN MDEF FOR MACRO TEXT.
C Clobber MACRO NAME IF OLD MACRO OR POP FROM LIST
C IF NEW MACRO. THEN RESET POINTER TO LAST GOOD
C CHARACTER IN MDEF ARRAY.
IF(IM.EQ.MPTOP) GO TO 118
MPLIST(IM,1)=NIL
GO TO 122
118 MPTOP=MPTOP-1
122 LASTMC=MCSAVE
CALL ERRMSG(28,28HNO ROOM FOR MACRO DEFINITION)
GO TO 999
C
C GET NEXT CHARACTER OF MACRO AND STORE IT IN MDEF.
124 CALL GETCH(ICH)
MDEF(LASTMC)=ICH
C
C LOOK FOR /EM TO STOP DEFINITION PROCEDURE.
C (CASE OF ISTOP)
GO TO (126,128,130,999),ISTOP
C
C ISTOP=1 - WE ARE LOOKING FOR THE START OF COMMAND
C CHARACTER (ICOM).
126 IF(ICH.NE.ICOM) GO TO 134
ISTOP=2
GO TO 134
C
C ISTOP=2 - WE ARE LOOKING FOR AN E.
C IF WE DON'T HAVE AN E, DONT LOOK FOR A SECOND ICOM,
C IT WOULD BE TEXT.
128 ISTOP =3
IF(ICH.NE.IAME) ISTOP=1
GO TO 134
C
C ISTOP=3 - IF WE HAVE AN M, WE ONLY NEED ONE MORE
C CHARACTER (THE COMMAND DELIMITER).
130 IF(ICH.NE.IAMM) GO TO 132
ISTOP=4
GO TO 134
C NOT AN M. START LOOKING FOR A START OF COMMAND CHARACTER AGAIN.

132 ISTOP=1
    IF(ICH.EQ.ICOM) ISTOP=2
    (END OF CASE ISTOP)

C INCREMENT THE POINTER INTO MDEF.

134 LASTMC=LASTMC+1

C AND GO LOOP ON NEXT CHARACTER OF MACRO TEXT.
69 TO 117

C ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C EM - END MACRO (IP=2).
C POP THE MACRO OFF THE OPEN MACRO STACK.
140 CALL MSTACK(0)
60 TO 999

C ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C OP - OPEN FILE (IP=3).
C GET THE FILE NUMBER.

150 CALL GETFLD(IFLG,IA1,IA2)
C BE SURE FILE NUMBER IS NUMERIC.
IF(IFLG.EQ.0) GO TO 152
CALL ERRMSG(28,28HOPEN FILE NUMBER NOT NUMERIC)
GO TO 999
C PUSH FILE ONTO OPEN FILE STACK IF THERE IS ROOM.
152 IF(ITOP.LT.IMAX) GO TO 154
CALL ERRMSG(19,19HTOO MANY FILES OPEN)
GO TO 998
C SAVE ACTIVE INPUT BUFFER AND WORD POINTER.
C SET IWORD TO FORCE GETCH TO DO A READ.
154 J=ITOP*MAXREC
DO 155 I=1,LREC
K=J+1
155 IBUF(K)=IBUF(I)
istik(ITOP,2)=IWORD
ITOP=ITOP+1
istik(ITOP,1)=IA1
IWORD=9999
GO TO 998
C ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C CL - CLOSE FILE (WHICH IS ON TOP OF THE OPEN FILE STACK). (IP=4)
C BE SURE THERE IS A FILE TO CLOSE.
160 IF(ITOP.LE.0) GO TO 999
C
C POP THE OPEN FILE STACK.
    ITOP=ITOP-1
C
C IF STACK IS NOT EMPTY, RESTORE THE ACTIVE BUFFER
C AND WORD POINTER.
    IF(ITOP.LE.0) GO TO 999
    J=ITOP*MAXREC
    DO 161 I=1,LREC
         K=J+I
161   IBUF(I)=IBUF(K)
    IWORD=ISTK(ITOP,2)
    GO TO 999
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C DT - DEFINE TRAP (IP=5).
C
C GET THE LINE NUMBER.
170 CALL GETFLD(IFLG,LN,IA2)
C
C BE SURE LINE NUMBER IS NUMERIC.
    IF(IFLG.EQ.0) GO TO 172
    CALL ERRMSG(22,22HDT LINE NUMBER MISSING)
    GO TO 999
C
C GET NAME OF MACRO TO CALL.
172 CALL GETFLD(IFLG,IA1,IA2)
C
C FIND PLACE (LINE NUMBER) IN TABLE TLIST.
    NEW=TTOP+1
    IF(TTOP.EQ.0) GO TO 174
    DO 173 I=1,TTOP
173   IF(TLIST(I,1).EQ.LN) NEW=I
C
C CHECK TO SEE IF CALLER IS DELETING A TRAP
C (MACRO NAME OF NUMERIC ZERO).
C IF THIS IS NOT A DELETE, GO TO 180
174   IF(IFLG.NE.0) GO TO 180
C
C MAKE SURE THE ARGUMENT IS ZERO.
    IF(IA1.EQ.0) GO TO 175
    CALL ERRMSG(21,21HDT MACRO NAME MISSING)
    GO TO 998
C
C DELETE THE LINE NUMBER BY COMPRESSING TRAP TABLE.
175   IF(NEW.LE.TTOP) GO TO 176
    CALL ERRMSG(26,26HTRIED TO DELETE UNDEF TRAP)
    GO TO 998
C
C IF WE ARE DELETING THE TOP ELEMENT,
C THERE IS NOTHING TO COMPRESS.
  176 IF(NEW.EQ.TTOP) GO TO 178
C
C COMPRESS THE TRAP TABLE.

  K=TTOP-1
  DO 177 I=NEW,K
       TLIST(I,1)=TLIST(I+1,1)
  177 TLIST(I,2)=TLIST(I+1,2)
  178 TTOP=TTOP-1
  GO TO 998
C
C WE ARE ADDING A TRAP TO THE TABLE.
C FIND THE POINTER TO THE REQUIRED MACRO.
  180 DO 181 IM=1,MPTOP
     IF(MPLIST(IM,1).NE.IA1) GO TO 181
     IF(MPLIST(IM,2).EQ.IA2) GO TO 182
  181 CONTINUE
     CALL ERRMSG(25,25HDT MACRO NAME NOT DEFINED)
  GO TO 999
C
C MAKE SURE THE TRAP WILL FIT IN THE TRAP TABLE.
  182 IF(NEW.LE.TMAX) GO TO 184
     CALL ERRMSG(14,14HTOO MANY TRAPS)
  GO TO 999
C
C STORE THE NEW TRAP.
  184 TLIST(NEW,1)=LN
     TLIST(NEW,2)=MPLIST(IM,3)
     IF(NEW.GT.TTOP) TTOP=NEW
  GO TO 999
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C ET - END TRAP (IP=6).
C
C MAKE SURE THERE IS A TRAP TO END.
  190 IF(TRAPFL.EQ.0) GO TO 999
C
C FORCE END OF CURRENT LINE.
C CALL BUILDL(NIL)
C
C RESTORE STUFF SAVED FOR BUILDL (SAVED IN CKTRAP).

START(1)  =LSAVE( 1)
START(2)  =LSAVE( 2)
ADJ(1)    =LSAVE( 3)
MINCOL(1) =LSAVE( 4)
LWORD     =LSAVE( 5)
NEXTCH    =LSAVE( 6)
NEXTMC    =LSAVE( 7)
MAXWD     =LSAVE( 8)
ADJFLG    =LSAVE( 9)
MAXCOL    =LSAVE(10)
CENTER    =LSAVE(11)
LM = LSAVE(12)
IRM = LSAVE(13)
COMFLG = LSAVE(14)
BFFLG = LSAVE(15)
NBL = LSAVE(16)
J = START(2)
DO 191 I = 1, J

191 LBUF(I) = LSAVE(I + 16)

C
C SET TRAP IN PROGRESS FLAG FALSE.
TRAPIFL = 0
GO TO 999
C
C ******************************************************
C EL - END LINE (IP=7).
C
200 NBL = 0
CALL BUILDL(NIL)
GO TO 999
C
C ******************************************************
C BL - FORCE BLANK LINES (IP=8).
C
210 CALL GETFLD(IFLG, IA1, IA2)
C
C MAKE SURE FIELD IS NUMERIC.
IF(IFLG.EQ.0) GO TO 212
CALL ERRMSG(18, 18HBL ARG NOT NUMERIC)
GO TO 999
C
212 NBL = IA1
CALL BUILDL(NIL)
GO TO 998
C
C ******************************************************
C PA - FORCE END OF CURRENT PAGE (IP=9).
C BY FORCING BLANK LINES UNTIL /IP IS EXECUTED
C (BY A TRAP INVOKED MACRO).
C
220 NBL = 1000
GO TO 999
C
C ******************************************************
C LM - SET LEFT MARGIN (IP=10).
C
230 CALL GETFLD(IFLG, IA1, IA2)
IF(IFLG.EQ.0) GO TO 232
CALL ERRMSG(14, 14HLM MISSING ARG)
GO TO 999
C CHECK FOR VALID RANGE OF COLUMN NUMBER.
232 IF((IA1.GE.1).AND.(IA1.LT.IRM)) GO TO 234
CALL ERRMSG(15,15HLM OUT OF RANGE)
GO TO 998
C
C RESET NEXTMC FOR BUILDL IF THE LINE DOESN'T HAVE
C A WORD IN IT YET.
234 IF(NEXTMC.EQ.LM) NEXTMC=IA1
C
C RESET THE LEFT MARGIN.
LM=IA1
GO TO 998
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C RM - SET RIGHT MARGIN (IP=11).
C
C GET COLUMN NUMBER.
240 CALL GETFLD(IFLG,IA1,IA2)
IF(IFLG.EQ.0) GO TO 242
CALL ERRMSG(14,14HRM MISSING ARG)
GO TO 999
C
C CHECK FOR VALID RANGE OF COLUMN NUMBER.
242 IF((IA1.GT.LM).AND.(IA1.LE.MAXCOL)) GO TO 244
CALL ERRMSG(15,15HRM OUT OF RANGE)
GO TO 998
C
244 I RM=IA1
GO TO 998
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C IP - INITIALIZE NEW PAGE (IP=12).
C
C STOP BLANK LINE OUTPUTING IF NECESSARY AND
C MAKE SURE CURRENT LINE HAS ENDED.
250 IF(NBL.GT.0) NBL=0
IF(LSAVE(16).GT.0) LSAVE(C16)=0
CALL BUILDL(NIL)
C
C RESET LINE NUMBER AND INCREMENT PAGE NUMBER.
LINENO=1
PAGENO=PAGENO+1
GO TO 999
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C BF - TURN BOLD FACE FLAG ON OR OFF (IP=13).
C
C SET BFFLG ON IF NO-LEADING MINUS SIGN ON COMMAND.
C NOTE - BFFLG IS NOT BOOLEAN. IT IS AN INTEGER THAT
C IS ADDED TO EVERY NON-BLANK CHARACTER OF TEXT AND
C IS REMOVED BEFORE PRINTING.
260 IF(IFLG.EQ.1) BFFLG=IBFBIT
IF(FLG.EQ.(-1)) BFLAG=0
GO TO 999

TA–N – TAB TO COLUMN N (IP=14)

GET N.
270 CALL GETFLD(IFLG,N,IA2)
   IF(IFLG.EQ.0) GO TO 272
   CALL ERRMSG(14,14HTA MISSING ARG)
   GO TO 999

IF TAB IS TO LEFT OF CURRENT WORD,
OUTPUT CURRENT LINE BEFORE SETTING.
272 IF(N.LT.NEXTMC) CALL BUILDL(NIL)

CHECK COLUMN NUMBER BEFORE SETTING TAB.
IF(N.GE.2.AND.N.LE.MAXCOL) GO TO 274
   CALL ERRMSG(19,19HTA ARG OUT OF RANGE)
   GO TO 998

274 NEXTMC=N
   TABFLG=1
   GO TO 998


GET N, THE NUMBER OF LINES TO BLANK.
280 CALL GETFLD(IFLG,N,IA2)
   IF(IFLG.EQ.0) GO TO 282
   CALL ERRMSG(14,14HCE MISSING ARG)
   GO TO 999

END THE CURRENT LINE IF NECESSARY.
282 CALL BUILDL(NIL)

MAKE SURE N IS POSITIVE BEFORE GOING ON.
IF(N.LT.0) N=0
   CENTER=N
   GO TO 998

RL–N – SET INPUT RECORD LENGTH TO N (IP=16).

GET THE LENGTH, N.
290 CALL GETFLD(IFLG,N,IA2)
   IF(IFLG.EQ.0) GO TO 292
   CALL ERRMSG(14,14HRL MISSING ARG)
   GO TO 999

CHECK THE RANGE OF LENGTH BEFORE STORING.
292 IF(N.GT.0.AND.N.LE.MAXREC) GO TO 294
CALL ERRMSG(19,19HRL ARG OUT OF RANGE)
GO TO 998
LREC=N
GO TO 998
C
* * * * * * * * * * * * * * * * * * * * *
C FC,N - SET FIRST CHAR OF INPUT RECORD TO BE USED
(1P=17).
C
C GET N.
300 CALL GETFLD(IFLG,N,IA2)
IF(IFLG.EQ.0) GO TO 302
CALL ERRMSG(14,14HFC MISSING ARG)
GO TO 999
C
C CHECK RANGE OF N BEFORE STORING.
302 IF(N.GE.1.AND.N.LT.MAXPOS) GO TO 304
CALL ERRMSG(19,19HFC ARG OUT OF RANGE)
GO TO 998
304 MINPOS=N
GO TO 998
C
* * * * * * * * * * * * * * * * * * * * *
C LC,N - SET LAST CHAR OF INPUT RECORD TO BE USED
(1P=18).
C
C GET N.
310 CALL GETFLD(IFLG,N,IA2)
IF(IFLG.EQ.0) GO TO 312
CALL ERRMSG(14,14HLC MISSING ARG)
GO TO 999
C
C CHECK RANGE OF N BEFORE STORING.
312 IF(N.LE.LREC.AND.N.GT.MINPOS) GO TO 314
CALL ERRMSG(19,19HLC ARG OUT OF RANGE)
GO TO 998
314 MAXPOS=N
GO TO 998
C
* * * * * * * * * * * * * * * * * * * * *
C CO - TURN COMPRESS BLANKS FLAG ON OR OFF (1P=19).
C
C SET FLAG DEPENDING ON THE LEADING SIGN ON COMMAND.
320 IF(IFLG.EQ.1) COMFLG=1
IF(IFLG.EQ.(-1)) COMFLG=0
GO TO 999
C
* * * * * * * * * * * * * * * * * * * * *
C JU - TURN JUSTIFY LINE FLAG ON OR OFF (1P=20).
C
C SET BASED ON THE LEADING SIGN OF THE COMMAND.
330 IF(IFLG.EQ.1) ADJFLG=1
IF(IFLG.EQ.(-1)) ADJFLG=0  
GO TO 999

C  SC,N - SET THE START OF COMMAND CHARACTER (IP=21).
C
340 CALL GETCH(ICh)
       CALL GETCH(IA2)
C  CHANGE THE START-OF-COMMAND CHAR IN EVERY MACRO.
   DO 341 I=1, LASTMC
       IF(MDEF(I).EQ.ICOM) MDEF(I)=ICH
   341 CONTINUE
C  SET THE NEW START-OF-COMMAND CHARACTER.
   ICOM=ICH
   GO TO 998
C  PN,N - SET THE CURRENT PAGE NUMBER TO N (IP=22).
C
350 CALL GETFLD(IFLG,N,IA2)
   IF(IFLG.EQ.O) GO TO 352.
       CALL ERRMSG(14,14HPN ARG \MISSING)
   GO TO 999
352 PAGENO=N
   GO TO 998
C  SR,N,M - SET REGISTER N TO INTEGER VALUE M (IP=23).
C
360 CALL GETFLD(I1,N,IA2)
       CALL GETFLD(I2,M,IA2)
   IF(I1.EQ.O.AND.I2.EQ.O) GO TO 362
       CALL ERRMSG(23,23HSR ARG1 OR ARG2 \MISSING)
   GO TO 999
C  CHECK REGISTER NUMBER BEFORE SETTING IT.
362 IF(N.GE.1.AND.N.LE.REGTOP) GO TO 364
       CALL ERRMSG(24,24HSR ARG1 NOT \VALID REG NO)
   GO TO 998
364 REG(N)=M
   GO TO 998
C
370 CALL GETFLD(I1,I,IA2)
       CALL GETFLD(I2,J,IA2)
CALL GETFLD(I3,K,IA2)
IF(I1.EQ.0.AND.I2.EQ.0.AND.I3.EQ.0) GO TO 372
CALL ERRMSG(14,14HAD MISSING ARG)
GO TO 999
C
C   CHECK FOR VALID REGISTER NUMBER.
372 IF(I.GE.1.AND.I.LE.REGTOP) GO TO 374
CALL ERRMSG(20,20HAD ARG1 OUT OF RANGE)
GO TO 998
374 IF(J.GE.1.AND.J.LE.REGTOP) GO TO 376
CALL ERRMSG(20,20HAD ARG2 OUT OF RANGE)
GO TO 998
376 IF(K.GE.1.AND.K.LE.REGTOP) GO TO 378
CALL ERRMSG(20,20HAD ARG3 OUT OF RANGE)
GO TO 998
C
C   OK, DO THE ADDITION NOW.
378 REG(I)=REG(J)+REG(K)
GO TO 998
C
C   ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
C
C   GET I, J, AND K.
380 CALL GETFLD(I1,I,IA2)
CALL GETFLD(I2,J,IA2)
CALL GETFLD(I3,K,IA2)
IF(I1.EQ.0.AND.I2.EQ.0.AND.I3.EQ.0) GO TO 382
CALL ERRMSG(14,14HSU MISSING ARG)
GO TO 999
C
C   CHECK FOR VALID REGISTER NUMBER.
382 IF(I.GE.1.AND.I.LE.REGTOP) GO TO 384
CALL ERRMSG(20,20HSU ARG1 OUT OF RANGE)
GO TO 998
384 IF(J.GE.1.AND.J.LE.REGTOP) GO TO 386
CALL ERRMSG(20,20HSU ARG2 OUT OF RANGE)
GO TO 998
386 IF(K.GE.1.AND.K.LE.REGTOP) GO TO 388
CALL ERRMSG(20,20HSU ARG3 OUT OF RANGE)
GO TO 998
C
C   OK, DO THE SUBTRACTION NOW.
388 REG(I)=REG(J)-REG(K)
GO TO 998
C
C   ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **
C   IR,N,K - INSERT VALUE OF REGISTER N INTO THE TEXT
C   WITH A MINIMUM FIELD WIDTH OF K
C
C   GET N.
400 CALL GETFLD(IFLG,N,IA2)
    IF(IFLG.EQ.0) GO TO 402
    CALL ERRMSG(14,14H1R MISSING ARG)
    GO TO 999

C CHECK FOR VALID REGISTER NUMBER.
402 IF(N.GE.1.AND.N.LE.REGTOP) GO TO 404
    CALL ERRMSG(20,2DHIR ARG1 OUT OF RANGE)
    GO TO 998

C GET MINIMUM FIELD WIDTH, K.
404 CALL GETFLD(IFLG,K,IA2)
    IF(IFLG.EQ.C) GO TO 406
    CALL ERRMSG(14,14H1R MISSING ARG)
    GO TO 999

C CONVERT VALUE OF REGISTER TO SIGN-MAGNITUDE.
406 VAL=REG(N)
    SIGN=0
    IF(VAL.GE.0) GO TO 408
    SIGN=-1
    VAL=-VAL

C FIND OUT HOW MANY DIGITS WE NEED
    (MAX OF 9-1=8 ON THIS MACHINE).
408 DO 409 I=2,9
    IF(VAL.LT.10**(I-1)) GO TO 410
409 CONTINUE
    I=10
410 NDIG=I-1

C COMPUTE THE NUMBER OF LEADING BLANKS REQUIRED.
    NBLKS=K-NDIG+SIGN
    IF(NBLKS.LE.0) GO TO 412

C SET COMPRESS FALSE AND INSERT THE LEADING BLANKS.
    SAVEC=COMFLG
    COMFLG=0
    DO 411 I=1,NBLKS
        CALL BUILDL(1H )
        COMFLG=SAVEC
411 CONTINUE

C INSERT THE MINUS SIGN IF REGISTER IS NEGITIVE.
412 IF(SIGN.LT.0) CALL BUILDL(1H-)

C INSERT THE DIGITS (FROM LEFT TO RIGHT OF COURSE).
    DO 413 I=1,NDIG
        J=NDIG-I
        K=MOD(VAL/(10**J),10)
413 CALL BUILDL(DIG(K+1))
    GO TO 998
C * * * * * * * * * * * * * * * * * * * *
C RETURN AFTER READING A NUMERIC FIELD. IF NUMERIC
C FIELD WAS NOT DELIMITED BY A BLANK OR COMMA,
C TREAT THE DELIMITER AS TEXT.
  998 IF(IA2.NE.1H .AND. IA2.NE.1H) CALL BUILDL(IA2)
C 999 RETURN
    END
SUBROUTINE BUILDL(ICH)

C LINE BUILDER. CALLED TO PLACE ONE CHARACTER OF
C TEXT IN THE CURRENT LINE, TO END THE CURRENT LINE,
C OR TO PRODUCE A BLANK LINE. COMPLETELY FORMATTED
C LINES ARE WRITTEN TO A WORK FILE.
C
C IMPLICIT INTEGER (A-Z)
INTEGER ADJ, ADJFLG, BFFL5, BFLINE, CENTER, COMFLG, REG
INTEGER REGTOP, START, TABFLG, PAGENO
INTEGER ALL, FROM, GET, GOTBF, PAD, PUT, REM, TRIGER
C
COMMON/MACH/ICOM, NIL
COMMON/PSTUFF/ADJ(68), ADJFLG, BFFLS, BFLINE(132), CENTER,
1 COMFLG, IBFBIT, ICC, ICCBF, ICCPG, IMASK, IRM, ITEXT,
2 LASTCH, LBUF(264), LINE(132), LINENO, LNSAVE(148),
3 LWORD, MAXCOL, MAXWD, MINCOL(68), NBL, NCH, NEXTCH,
4 NEXTMC, REG(50), REGTOP, START(68), TABFLG
EQUIVALENCE (PAGENO, REG(1))
C
DATA IBLK, IPER/1H, 1H./
C
IF ICH=NIL, GO TO 80 TO END CURRENT WORD AND LINE.
IF(ICH.EQ.NIL) GO TO 80
C
IF ICH=BLANK, GO TO 30 TO HANDLE THE BLANK.
IF(ICH.EQ.IBLK) GO TO 30
C
ICH IS A CHARACTER OF TEXT.
GO TO 10 IF THIS IS NOT THE FIRST CHARACTER
OF A NEW WORD.
IF(NCH.NE.0) GO TO 10
C
WE MUST START A NEW WORD OF TEXT.
LWORD=LWORD+1
MINCOL(LWORD)=NEXTMC
ADJ(LWORD)=ADJFLG
IF(TABFLG.EQ.1) ADJ(LWORD)=0
TABFLG=0
C
STORE ICH IN THE LINE BUFFER
(SET THE BOLD FACE FLAG BIT IF NECESSARY).
10 LBUF(NEXTCH)=ICH+BFFL5
NEXTCH=NEXTCH+1
NCH=NCH+1
C
GO TO 40 TO FORCE END OF WORD IF IT FILLS THE LINE.
IF(NCH.GE.MAXWD) GO TO 40
C
OTHERWISE, RETURN TO THE CALLER.
GO TO 99
THIS SECTION HANDLES A BLANK CHARACTER.
IF THIS IS A LEADING BLANK ON THE LINE,
RETURN TO CALLER.
30 IF(NEXTCH.EQ.1) GO TO 99
GO TO 40 IF THIS BLANK DELIMITS THE CURRENT WORD.
IF(NCH.NE.0) GO TO 40
THIS MUST BE AN EXTRA TRAILING BLANK OF THE WORD.
IF(USER WANTS TO COMPRESS BLANKS, RETURN TO CALLER.
IF(COMFLG.EQ.1) GO TO 99
OTHERWISE, COUNT THE BLANK BY GOING TO 50
GO TO 50
THIS SECTION ENDS A CURRENT WORD.
40 START(LWORD+1)=NEXTCH
NEXTMC=NEXTMC+NCH+1
NCH=0
IF THE LAST CHARACTER WAS A PERIOD, GO TO 50 TO
FORCE 2 TRAILING BLANKS. OTHERWISE, GO TO 60.
IF(LASTCH.NE.IPER) GO TO 60
COUNT ONE ADDITIONAL BLANK ON THE LAST WORD.
50 NEXTMC=NEXTMC+1
IF THE NEXT WORD IS GOING TO START TO THE RIGHT
OF THE RIGHT MARGIN, GO TO 100 TO FORCE A NEW LINE.
60 IF(NEXTMC.GT.IRM) GO TO 100
OTHERWISE, RETURN TO THE CALLER.
GO TO 99
THIS SECTION FORCES BOTH THE END OF THE CURRENT
WORD (IF ANY) AND THE END OF THE CURRENT LINE.
WE ENTER HERE ONLY WHEN THE CALLER SENDS US A NIL.
GO TO 90 IF THERE ARE WORDS WAITING TO BE OUTPUT.
80 IF(LWORD.NE.0) GO TO 90
SINCE THERE IS NO WORD WAITING TO BE OUTPUT, WE
MUST CHECK TO SEE IF A BLANK LINE IS REQUIRED.
IF NOT, RETURN.
IF(NBL.EQ.0) GO TO 999
A BLANK LINE IS REQUIRED SO COUNT THE LINE AND
GO TO 100 TO OUTPUT IT.
NBL=NBL-1
GO TO 100
OUTPUT THE WORDS THAT ARE WAITING.
90 ADJ(LWORD)=0
C
C       IF THE LAST WORD HAS ALREADY BEEN ENDED, GO TO 100.
C         IF(NCH.EQ.0) GO TO 100
C
C         OTHERWISE, DELIMIT THE CURRENT WORD FIRST.
C         START(LWORD+1)=NEXTCH
C         NEXTMC=NEXTMC+NCH+1
C         NCH=0
C
C         ALSO CHECK TO SEE IF AN EXTRA BLANK IS NEEDED TO
C         FOLLOW A PERIOD (JUST IN CASE THE LAST WORD BECOMES
C         THE FIRST WORD ON THE NEXT LINE).
C         IF(LASTCH.EQ.IPER) NEXTMC=NEXTMC+1
C         GO TO 100
C
C         KEEP TRACK OF THE LAST CHARACTER OF TEXT.
C         99 LASTCH=ICH
C         GO TO 999
C
C         THIS SECTION IS ENTERED TO OUTPUT ONE LINE.
C         NOTE THAT THERE IS NEVER MORE THAN ONE TOO MANY
C         WORDS IN THE WORD TABLE TO FIT ON THIS LINE.
C
C         INITIALIZE THE WRITE BOLD FACE LINE FLAG TO FALSE.
C         100 COTBF=0
C
C         KEEP TRACK OF THE LAST WORD IN TABLE IN CASE IT
C         DOESN'T FIT ON THE LINE.
C         LASTWD=LWORD
C
C         IF THIS LINE IS NOT BLANK, GO TO 109
C         IF(LWORD.NE.0) GO TO 109
C
C         OTHERWISE, BLANK THE WRITE BUFFER.
C         LINE(1)=ICC
C         DO 101 I=2,MAXCOL
C         101 LINE(I)=IBLK
C
C         AND GO TO 200 TO WRITE THE LINE.
C         GO TO 200
C
C         FIND OUT HOW CLOSE THE NON-BLANK LINE COMES TO
C         FILLING THE SPACE BETWEEN THE MARGINS.
C         109 PAD=IRM-MINCOL(LWORD)-START(LWORD+1)+START(LWORD)+1
C
C         GO TO 120 IF ALL THE WORDS IN THE TABLE WILL FIT
C         ON THE LINE.
C         IF(PAD.GE.0) GO TO 120
C
C         THEY ALL WONT FIT. IF WE ONLY HAVE ONE WORD,
C         OUTPUT IT ANYWAY. (THE MARGINS MUST HAVE CHANGED)
PAD=0
IF(LWORD.EQ.1) GO TO 120
C
C IF WE HAVE ALREADY BYPASSED THE LAST WORD,
C OUTPUT ANYWAY. (MARGINS HAVE BEEN MOVED)
IF(LWORD.NE.LASTWD) GO TO 120
C
C LEAVE THE LAST WORD IN THE TABLE OFF OF THIS LINE.
LWORD=LWORD-1
GO TO 109
C
C COUNT NUMBER OF GAPS WHICH CAN GET EXTRA BLANKS.
120 NGAP=0
I=LWORD
123 IF(I.EQ.1) GO TO 126
IF(ADJ(I).EQ.0) GO TO 126
NGAP=NGAP+1
I=I-1
GO TO 123
C
C GO TO 128 IF LINE IS TO BE CENTERED.
126 IF(CENTER.GT.0) GO TO 128
C
C GO TO 140 IF LINE CANNOT BE PADDED WITH BLANKS.
IF(NGAP.LE.0) GO TO 140
C
C ALSO GO TO 140 IF LINE DOESNT NEED PADDING.
IF(PAD.LE.0) GO TO 140
C
C OTHERWISE, GO TO 130 TO PAD THE LINE.
GO TO 130
C
C CENTER THE LINE AND DECREMENT THE COUNT OF LINES
C WHICH NEED TO BE CENTERED.
128 CENTER=CENTER-1
PAD=PAD/2
DO 129 I=1,LWORD
129 MINCOL(I)=MINCOL(I)+PAD
GO TO 140
C
C WE MUST PAD THE LINE WITH BLANKS.
C PUT PAD BLANKS INTO NGAP SLOTS STARTING WITH
C MINCOL(LWORD) AND WORK BACKWARDS.
130 ALL=PAD/NGAP
REM=PAD-(ALL*NGAP)
I=LWORD
C
C EACH ELIGIBLE GAP WILL GET (ALL) BLANKS ADDED.
C (REM) BLANKS WILL THEN BE DISTRIBUTED FROM THE
C RIGHT (OR LEFT ON ALTERNATE LINES).
TRIGER=I-NGAP+REM
IF(MOD(LINENO/2).EQ.0) TRIGER=I
STUFF BLANKS INTO ONE OF THE GAPS BY MOVING THE
START OF COLUMN OF THE WORD TO THE RIGHT.

133  MINCOL(I)=MINCOL(I)+PAD
     PAD=PAD-ALL
     IF(I.GT.TRIGER) GO TO 134
     IF(REM.LE.0) GO TO 134
     FAD=PAD-1
     REM=REM-1
134  I=I-1
     LOOP UNTIL ALL BLANKS HAVE BEEN INSERTED.
     IF(PAD.GT.0) GO TO 133
     NO (FURTHER) ADJUSTMENT IS NEEDED.
     MOVE LBUF TO LINE (AND TO BFLINE IF NECESSARY).
140  LINE(1)=ICC
     EFLINE(1)=ICCBF
     PUT=2
     GET=1
     DO 149 I=1,LWORD
         LEN=START(I+1)-START(I)
         IF(LEN.LE.0) GO TO 144
         IF(PUT.GE.MINCOL(I)) GO TO 144
         INSERT ONE BLANK.
         LINE(PUT)=IBLK
         EFLINE(PUT)=IBLK
         PUT=PUT+1
         GO TO 143
144  DO 147 J=1,LEN
         JCH=JCH-IBFGIT
         IF(MOD(JCH,2)*(-2)+ISIGN(1,JCH).EQ.IMASK) GO TO 145
         BFLINE(PUT)=IBLK
         GO TO 146
145  JCH=JCH-IBFBIT
         GOBF=1
         BFLINE(PUT)=JCH
146  LINE(PUT)=JCH
         PUT=PUT+1
147  CONTINUE
149  CONTINUE
C FILL OUT THE RIGHT END OF THE LINE WITH BLANKS.
151 IF(PUT.GT.MAXCOL) GO TO 200
LINE(PUT)=IBLK
EFLINE(PUT)=IBLK
PUT=PUT+1
GO TO 151
C
C ENTER THIS SECTION WITH (LINE) READY TO BE PRINTED
C AND WITH (BFLINE) READY TOO IF GOTBF IS TRUE.
C
C OUTPUT ONE LINE.
200 CONTINUE
C
C IF SOME BOLD FACE, OUTPUT TO TEXT FILE FIRST.
C IF(GOTBF.EQ.0) GO TO 202
WRITE(ITEXT,203) (BFLINE(I),I=1,MAXCOL)
C
C IF THIS IS THE FIRST LINE ON THE PAGE,
C SET THE CARRIAGE CONTROL CHARACTER.
C
202 IF(LINENO.EQ.1) LINE(1)=ICCPG
C
C OUTPUT THE LINE OF TEXT.
WRITE(ITEXT,203) (LINE(I),I=1,MAXCOL)
203 FORMAT(132A1)
C
C INCREMENT LINE COUNT.
LINENO=LINENO+1
C
C GO TO 210 IF WE HAVE A WORD LEFT IN THE WORD TABLE.
C IF(LWORD.NE.LASTWD) GO TO 210
C
C NO WORD LEFT OVER. INITIALIZE FOR THE NEXT LINE.
START(1)=1
NEXTMC=LM
LWORD=0
NEXTCH=1
GO TO 290
C
C MOVE THE LEFT OVER WORD TO THE FIRST POSITION
C IN THE WORD TABLE.
210 FROM=START(LASTWD)
LEN=START(LASTWD+1)-START(LASTWD)
DO 213 I=1,LEN
LBUF(I)=LBUF(FROM)
FROM=FROM+1
213
C
C INITIALIZE TO INCLUDE THE WORD JUST MOVED.
START(1)=1
START(2)=LEN+1
MINCOL(1)=LM
NEXTMC=LM+NEXTMC-MINCOL(LASTWD)
LWORD=1
NEXTCH=LEN+1
C
C NOTE. THE NEXT STATEMENT FORCES A WORD THAT WAS
C TABBED TOO FAR TO THE RIGHT TO FIT ON THE LINE TO
C BE PRINTED ON THE LEFT MARGIN OF THE NEXT LINE.
ADJ(1)=ADJFLG
C
C IF END LINE WAS REQUESTED AND WE ARE NOT WRITING
C BLANK LINES, REQUEST ANOTHER NIL FROM THE DRIVER
C TO OUTPUT THE LEFT OVER WORD.
IF(ICH.EQ.NIL.AND.NBL.EQ.0) NBL=-1
C
C RECOMPUTE THE DISTANCE BETWEEN MARGINS.
290 MAXWD=IRM-LM+1
C
C RETURN TO CALLER.
999 RETURN
END
SUBROUTINE CKTRAP

C CHECK FOR TRAP. CALLED BY THE DRIVER TO CHECK IF
C A TRAP SHOULD OCCUR ON THE CURRENT LINE NUMBER-
C IF SO, INVOKE THE TRAP BEFORE RETURNING.

C IMPLICIT INTEGER (A-Z)
INTEGER ADJ, ADJFLG, BFFLG, BFLINE, CENTER, COMFLG, REG
INTEGER REGTOP, START, TABFLG, PAGENO
INTEGER TLIST, TMAX, TRAPFL, TTOP

COMMON/PSTUFF/ADJ(68), ADJFLG, BFFLG, BFLINE(132), CENTER,
1 COMFLG, IBFBIT, ICC, ICCBF, ICCPG, IMASK, IRM, ITEXT,
2 LASTCH, LBUF(264), LINE(132), LINENO, LM, LSAVE(148),
3 LWORD, MAXCOL, MAXWD, MINCOL(68), NBL, NCH, NEXTCH,
4 NEXTMC, REG(50), REGTOP, START(68), TABFLG
EQUIVALENCE (PAGENO, REG(1))
COMMON/TRAPS/TLIST(4, 2), TMAX, TRAPFL, TTOP

C RETURN IF A TRAP IS ALREADY IN PROGRESS.
IF (TRAPFL.EQ.1) GO TO 999

C DONT BOTHER TO CHECK AN EMPTY TRAP LIST.
IF (TTOP.EQ.0) GO TO 999

C LOOK FOR (LINENO) IN THE TRAP LIST.
DO 11 I=1, TTOP
IF (TLIST(I,1).EQ. LINENO) GO TO 20
11 CONTINUE

C RETURN, LINENO IS NOT ON THE TRAP LIST.
GO TO 999

C WE FOUND LINENO ON THE LIST - TRAP.
FIRST, PUSH THE END TRAP MACRO ONTO THE OPEN MACRO
STACK. NOTE, THE END TRAP MACRO IS ASSUMED TO BE
THE FIRST THING STORED IN THE MDEF ARRAY.
20 CALL MSTACK(1)

C NEXT, PUSH THE REQUIRED MACRO ONTO THE STACK.
CALL MSTACK(TLIST(I, 2))

C SET FLAG TO PREVENT ANOTHER TRAP FROM INTERRUPTING.
TRAPFL = 1

C SAVE THE PARTIAL LINE STUFF FOR SUBROUTINE BUILDL
C (END TRAP MACRO HANDLER IN INTRP WILL RESTORE).
LSAVE( 1) = START(1)
LSAVE( 2) = START(2)
LSAVE( 3) = ADJ(1)
LSAVE( 4) = MINCOL(1)
LSAVE( 5) = LWORD

C
LSAVE( 6) = NEXTCH
LSAVE( 7) = NEXTMC
LSAVE( 8) = MAXWD
LSAVE( 9) = ADJFLG
LSAVE(10) = MAXCOL
LSAVE(11) = CENTER
LSAVE(12) = LM
LSAVE(13) = IRM
LSAVE(14) = COMFLG
LSAVE(15) = BFFFLG
LSAVE(16) = NBL
J = START(2)
DO 23 I = 1, J
23 LSAVE(I+16) = LBUF(I)

C
C INITIALIZE FOR STARTING A NEW LINE.
START(1) = 1
LWORD = 0
NEXTCH = 1
NEXTMC = LM
CENTER = 0
BFFFLG = 0
NBL = 0

C
C RETURN TO CALLER.
999 RETURN
END
SUBROUTINE GETCH(ICH)

GET NEXT CHARACTER OF INPUT. THE CHARACTER IS
OBTAINED FROM EITHER THE MACRO COMMAND STORAGE
AREA OR FROM THE MOST RECENTLY OPENED INPUT FILE.
RETURNS THE NIL CHARACTER WHEN NO FILE IS OPEN.

IMPLICIT INTEGER (A-Z)

COMMON/ISTUFF/IBUF(270),IMAX,ISTK(3,2),ITOP,IWORD,
1 LREC,MAXPOS,MSTRK,MINPOS
COMMON/MACH/ICOM,NIL
COMMON/MACROS/LASTMC/MADEF,MDEF(3000),MMAX,
1 MPLIST(20,3),MPMAX,MPTOP,MSTK(50),MTOPO,NWORD

GO TO 50 IF WE MUST INPUT FROM THE MACRO AREA.
1 IF(MWORD.GT.0) GO TO 50
GO TO 10 IF THE OPEN INPUT FILE STACK IS NOT EMPTY.
IF(ITOP.GT.0) GO TO 10
OTHERWISE, WE ARE OUT OF INPUT. RETURN NIL.
ICH=NIL
GO TO 99

GET CHARACTER FROM INPUT FILE.
GO TO 20 IF THE ACTIVE BUFFER IS EMPTY.
10 IF(IWORD.GT.MAXPOS) GO TO 20

OTHERWISE, GET THE NEXT CHARACTER FROM THE BUFFER.
ICH=IBUF(IWORD)
IWORD=IWORD+1
GO TO 99

INPUT BUFFER IS EMPTY. TRY TO READ ANOTHER RECORD.
20 IFECF(IFILE).EQ.0) GO TO 1
READ(IFILE,21) (IBUF(I),I=1,LREC)
21 FORMAT(132A1)
IWORD=MINPOS

BEGIN AGAIN IF MORE DATA WAS READ INTO THE BUFFER.
IF(IFECF(IFILE).EQ.0) GO TO 1

GOT EOF. TRY TO 'POP THE OPEN INPUT FILE STACK.
ITOP=ITOP-1

IF STACK IS EMPTY NOW, BEGIN AGAIN (JUST TO QUIT).
IF(ITOP.EQ.0) GO TO 1

OTHERWISE, MOVE THE STACKED BUFFER TO THE ACTIVE
BUFFER AREA.
J=ITOP*MAXREC
DO 25 I=1,LREC
   K=J+I
25  IBUF(I)=IBUF(K)
  C  AND RESTORE THE WORD POINTER.
     IWORD=ISTK(ITOP,2)
  C  THEN PRETEND WE WERE NEVER INTERRUPTED -
  C  BEGIN AGAIN FROM THE TOP.
     GO TO 1
  C  GET THE CHARACTER FROM A MACRO DEFINITION.
  C  GO TO 60 IF THERE IS NO CODING ERROR.
50  IF(MWORD.LE.LASTMC) GO TO 60
     CALL ERRMSG(17,'MISSING MACRO-END')
  C  RECOVER BY POPPING MACRO STACK AND BEGINNING AGAIN.
     CALL MSTACK(0)
     GO TO 1
  C  GET THE CHARACTER.
60  ICH=MDEF(MWORD)
     MWORD=MWORD+1
  C  RETURN
END
SUBROUTINE GETFLD(IFLG,IA1,IA2)

GET NEXT FIELD OF A COMMAND.

IMPLICIT INTEGER (A-Z)
INTEGER DIG(10)

COMMON/MACH/ICOM,NIL

DATA DIG/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
DATA INEG,IPLUS,IBLK,ICOMMA/1H-,1H+,1H,/

CASE 1 - IF FIELD STARTS WITH A START OF COMMAND, RETURN IFLG=1, IA1=IACOM.
CASE 2 - IF FIELD IS ALPHA, RETURN IFLG=1 IF NO LEADING MINUS SIGN
         IFLG=-1 IF LEADING MINUS SIGN
         IA1=THE FIRST CHARACTER IN THE FIELD
         IA2=THE SECOND CHARACTER IN THE FIELD.
         SKIP ALL FOLLOWING CHARACTERS UNTIL A BLANK OR COMMA IS ENCOUNTERED.
CASE 3 - IF FIELD IS NUMERIC, RETURN IFLG=0
         IA1=INTEGER VALUE
         IA2=CHARACTER WHICH DELIMITED THE NUMERIC FIELD.

ISIGN=1
NUM=0

GET FIRST CHARACTER OF THE FIELD.
CALL GETCH(ICH)

IF NOT START OF COMMAND, GO TO 20
IF(ICH.NE.ICOM) GO TO 20

NOTIFY CALLER WE ENCOUNTERED AN ICOM WHICH SHOULD BE CONSIDERED TEXT.
IFLG=1
IA1=ICOM
IA2=ICOM
GO TO 999

NOT START OF COMMAND CHAR, LOOK FOR LEADING SIGN.
20 IF(ICH.NE.INEG) GO TO 22
ISIGN=-1
GO TO 24
22 IF(ICH.NE.IPLUS) GO TO 30

GET THE CHARACTER WHICH FOLLOWS THE SIGN.
24 CALL GETCH(ICH)
C CHECK FIRST NON-SIGN CHARACTER FOR BEING A DIGIT.
30 DO 31 I=1,10
   IF(ICH.EQ.DIG(I)) GO TO 40
31 CONTINUE
C WE HAVE A NON-NUMERIC FIELD.
   IFLG=ISIGN
   IA1=ICH
C GET THE SECOND CHARACTER OF ALPHA FIELD.
   CALL GETCH(ICH)
   IA2=ICH
C SKIP THE REST OF THE FIELD (CONSUME THE TRAILING
C BLANK OR COMMA).
35 IF(ICH.EQ.ICOMMA) GO TO 999
   IF(ICH.EQ.IBLK) GO TO 999
   IF(ICH.EQ.NIL) GO TO 999
   CALL GETCH(ICH)
   GO TO 35
C WE HAVE A NUMERIC FIELD.
C CONVERT NUMERIC CHARACTERS TO INTEGER AND
C ACCUMULATE THE VALUE.
40 NUM=NUM*10+I-1
C GET NEXT DIGIT.
   CALL GETCH(ICH)
   DO 41 I=1,10
      IF(ICH.EQ.DIG(I)) GO TO 40
41 CONTINUE
C THIS NON-DIGIT DELIMITS THE FIELD.
   IFLG=0
   IA1=NUM*ISIGN
   IA2=ICH
C 999 RETURN
END
SUBROUTINE MSTACK(NWORD)
C
MACRO STACK HANDLER. CALLED TO PUSH OR POP THE
STACK OF INVOKED MACROS.
C
IMPLICIT INTEGER (A-Z)
C
COMMON/MACROS/LASTMC,MAXDEF,MDEF(3000),MMAX,
1 MPLIST(20,3),MPMAX,MPTOP,MSTK(50),MTOP,MWORD
C
IF NWORD IS GT 0, GO TO 20 TO INVOCEx THE MACRO.
IF(NWORD.GT.0) GO TO 20
C
SINCE NWORD=0, TERMINATE THE CURRENT MACRO...
MWORD=0
C
...AND RETURN TO CALLER IF THE MACRO STACK IS EMPTY.
IF(MTOP.EQ.0) GO TO 99
C
OTHERWISE, POP THE MACRO STACK BEFORE RETURNING.
MWORD=MSTK(MTOP)
MTOP=MTOP-1
GO TO 99
C
PUSH THE CURRENT MACRO BEFORE INVOKING THE NEW ONE.
C
IF THERE IS NO CURRENT MACRO OPEN,
C THEREx IS NOTHING TO STACK.
20 IF(MWORD.EQ.0) GO TO 30
C
WE GOT TO PUSH.
IF(MTOP.LT.MMAX) GO TO 22
C
GOOD GRIEF, THERE IS NO ROOM ON THE STACK.
CALL ERRMSG(25,25HOPEN MACRO STACK OVERFLOW)
C
RECOVER BY SKIPPING THE NEW MACRO.
GO TO 99
C
THERE IS ROOM ON THE STACK.
22 MTOP=MTOP+1
MSTK(MTOP)=MWORD
C
TELL SUBROUTINE GETCH WHERE THE NEW MACRO IS.
30 MWORD=NWORD
C
99 RETURN
END
SUBROUTINE ERRMSG(LEN,MES)

C ERROR MESSAGE HANDLER. CALLED WHEN ANY ROUTINE
C WANTS TO OUTPUT AN ERROR MESSAGE. MESSAGES ARE
C WRITTEN ON FILE ERFILE (SYSTEM PRINTER).

C IMPLICIT INTEGER (A-Z)
INTEGER ERFILE,CPW
INTEGER ADJ,ADJFLG,BFFLG,BFLINE,CENTER,COMFLG,REG
INTEGER REGTOP,START,TABFLG,PAGENO

C DIMENSION MES(1)

COMMON/ERRM/ERFILE,CPW
COMMON/PSTUFF/ADJ(68),ADJFLG,BFFLG,BFLINE(132),CENTER,
1 COMFLG,IBFBIT,ICC,ICCMG,ICCPG,lmASK,IRM,ITText,
2 LASTCH,LBUF(264),LINE(132),LINO,LM,LSAVE(148),
3 LWORD,MAXCOL,MAXWD,MINCOL(68),NBL,NCH,NEXTCH,
4 NEXTMC,REG(50),REGTOP,START(68),TABFLG
EQUIVALENCE (PAGENO,REG(1))

C COMPUTE THE NUMBER OF WORDS IN MESSAGE
C (ON THIS MACHINE).
K=(LEN+CPW-1)/CPW

C OUTPUT LOCATION OF ERROR.
WRITE(ERFILE,11) PAGENO,LINENO
11 FORMAT(1X,13HERROR ON PAGE,I6,6H, LINE,I6)

C OUTPUT THE MESSAGE.
C NOTE - FORMAT IS MACHINE DEPENDENT (FOR CPW
C CHARACTERS PER WORD).
WRITE(ERFILE,13) (MES(I),I=1,K)
13 FORMAT(1X,32A4,-A3)

RETURN
END