Special Problem

DEVELOPMENT OF A TEXT FORMATTER UNDER VAX/VMS OPERATING SYSTEM

Presented to

The Department of Computer Sciences

NORTH TEXAS STATE UNIVERSITY

in partial fulfillment

of requirements for a Master of Science degree

by

Perng Chow

March, 1984

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. INTRODUCTION

No matter how extended the use of computer is, the printed document is still the primary medium for the presentation information, and will continue to be for some time. The use of computing facilities for preparation and production of the document is becoming as prevalent as their use for numeric computation. Commercially, document preparation systems are now a standard facility at research institution, and they have become quite common on each computer system. A conventional document preparation system usually contains two parts : a text editor used to create, enter, update, and maintain the text and control words that comprise the document in its "input" form, and a text formatter used to process that input and produce the final document[16].

A text formatter is a program for neatly formatting a document on a suitable printer. It produces output for the devices like terminals and line printers, with automatic right margin justification, pagination (skipping over the fold in the paper), page numbering and titling, centering, underlining, indenting, and multiple line spacing[7,16]. A text formatter is an important tool for anyone who writes, because, once correct, material is never retyped. This has some obvious cost benefits, and ensures that the number of

errors decrease with time. Machine formatting eases the typing job, since margin alignment, centering, underlining and similar tedious operations are handled by the computer. It also permits drastic format changes in a document without altering any text. But perhaps most important, it seems to encourage writers to improve their product, since the overhead of making an improvement is small and there is an esthetic satisfication in having a clear copy.

The original text formatter running under VAX/VMS operating system, called RUNOFF, is a large and complex formatting system[18]. Its full power is realized in the preparation of large "structured" documents such as books, manuals, theses, research, technical, and instructional publications containing footnotes, tables, figures, index, and table of contents. My project is trying to develop a small and simple, but complete text formatter under VAX/VMS operating system. I call it STF for the Small Text Formatter. It includes all the necessary functions to prepare most simple documents, such as office correspondence and small papers. It also has the facilities to prepare the table of contents and index.

This paper is divided into seven sections. A brief synopsis of the sections is included below.

Section I introduces the problem to be studied and the organization of this paper.

Section II discusses the format of control words and

their parameters.

Section III describes the structure of the whole system and its features. The structure of the program and its hierarchy chart are also described.

Section IV discusses the strategies used in several important view points of the program. Those strategies include structure of index, structure of table of contents, structure of titles, structure of user defined macros, boldface and underline effects, centering and right adjustment, a method to determine the line break point, and method to distribute extra spaces on an text output line.

Section V presents the conclusions of the study.

Section VI discusses the possible further extensions.

Section VII is the appendices. Appendix A contains the source listing of the program. Appendix B is the sample input data. Appendix C is the output produced from the sample input data. Appendix D is the STF user's manual, and Appendix E is the bibliography.

II. CONTROL WORDS AND PARAMETERS

II.1 Format of Control Words

The inputs to STF are the text and control words that you enter into the computer using a terminal and a program called "editor". The text comprises the main body of the document, and the control words specify the formatting requirements of the document. Each time STF reads a line from the input file, it looks at the character in column 1. If that character is a period ("."), STF considers the input line to be a control word. Otherwise, the input line is treated as text. Each STF control word is of the form

```
.xx  parameter(s)
```

The period (".") at column 1 is called the control word indicator and "xx" is a 2-character control word name. Some STF control words have one or more parameters. There should be at least one space between the control word and the first parameter. Each parameter must also be separated by at least one space. Both the control words and their parameters may be entered in any combination of uppercase and lowercase characters. In the STF system, control words and parameters are treated as lowercase characters. After finding the input line is a control word, STF first converts the input line into its lowercase equivalent, and then processes it. With this way, STF will never be confused by

the different combination of uppercase and lowercase characters in the control words and their parameters.

II.2 Syntax of Control Words

The following list describes the syntax of all STF control words. These control words show that STF provides all the functions needed to prepare a simple document. Some functions, for example, drawing boxes, multiple input files, multiple columns per page, and interactive input from the terminal, are not included in STF[1,2,3,12,18]. The reason why I do not include these complex functions in STF is that they are not frequently used for preparing a simple document, or they can be done by using another way.

(1). .AD < ALL | EVEN | ODD > < n | +n | -n >

(2). .BD < 1 | n | ON | OFF >

(3). .BI < 1 | n | ON | OFF >

(4). .BM < ALL | EVEN | ODD > < n | +n | -n >

(5). .BR

(6). .BT < ALL | EVEN | ODD > n /s1/s2/s3/

(7). .CE < 1 | n | ON | OFF >

(8). .CM information

(9). .CO < ON | OFF >

(10). .CP n

(11). .DM < name | OFF >

(12). .DS

(13). .EA < n | +n | -n >

(14). .EB n /s1/s2/s3/

(15). .EP

(16). .ET n /s1/s2/s3/

(17). .FB < ON | OFF | PUT >

(18). .FD < ON | OFF >

(19). .FI

(20). .FM < ALL | EVEN | ODD > < n | +n | -n >

(21). .FO < ON | OFF >

(22). .FS < ALL | EVEN | ODD > < n | +n | -n >

(23). .HM < ALL | EVEN | ODD > < n | +n | -n >

(24). .HS < ALL | EVEN | ODD > < n | +n | -n >

(25). .IN < n | +n | -n >

(26). .IR < n | +n | -n >

(27). .IX

(28). .JU < ON | OFF >

(29). .LL < n | +n | -n >

(30). .LS < n | +n | -n >

(31). .NC

(32). .NF

(33). .NJ

(34). .OA < n | +n | -n >

(35). .OB n /s1/s2/s3/

(36). .OP

(37). .OT n /s1/s2/s3/

(38). .PA < %+1 | n | +n | -n | EVEN | ODD >

(39). .PI

(40). .PL < n | +n | -n >

(41). .PN < ARABIC | ROMAN >

(42). .PS character

(43). .PT

(44). .RI < 1 | n | ON | OFF >

(45). .RO < BIG | SMALL >

(46). .SK < 1 | n >

(47). .SP < 1 | n >

(48). .SS

(49). .TC < 1 | n >

(50). .TI < n | +n | -n >

(51). .TM < ALL | EVEN | ODD > < n | +n | -n >

(52). .TT < ALL | EVEN | ODD > n /s1/s2/s3/

(53). .UC < 1 | n | ON | OFF >

(54). .UN < n | +n | -n >

(55). .UP < 1 | n | ON | OFF >

(56). .US < 1 | n | ON | OFF >

According to the format of parameters, STF control words can be divided into eight categories.

1. Control words which do not have any parameter.

   Generally, most of the control words in this category are the short-hand way of another control word. Using another control word, you also can obtain the same result, but they offer an easy way to specify the formatting requirements. The other control words in this

category have an unique function so that they need not have the parameter to specify the result to be produced. For example, ".IX" control word indicates that the next text input line will be put into the index structure as well as used to produce output text.

2. Control words whose parameters look like < n | +n | -n >.

   The parameter of these control words is a number. It may be an unsigned number, a positive number, or a negative number. A parameter with the form "n" will change the current setting value absolutely, and a parameter with the form "+n" or "-n" will change the current setting value relatively. Some control words will only affect the next text input line. Some of them will remain in effect for all the subsequent text input lines until another control word alters the current setting value.

3. Control words whose parameters have the form < 1 | n >.

   The parameters of these control words may be omitted. If the parameter is omitted, then 1 is assumed. Otherwise, the parameter must be an unsigned number. This parameter is treated as a positive number.

4. Control words with < ON | OFF > in parameter field.

   These control words always affect a block of text input lines. This block of text input lines starts from the control word with the "ON" parameter to the control word with the "OFF" parameter.

5. Control words whose parameters have the form

   < 1 | n | ON | OFF >

   These control words will affect next "n" text input lines or a block of text input lines. If no parameter is presented, then 1 is assumed to affect the next text input line.

6. Control words with the parameter of the form

   < ALL | EVEN | ODD > < n | +n | -n >

   These control words will affect the output text of even and/or odd numbered pages. The second parameter is used to change the current setting value absolutely or relatively. The new setting value will be in effect for all the subsequent text output lines until another control word alters it.

7. Control words whose parameters look like n /s1/s2/s3/ or < ALL | EVEN | ODD > n /s1/s2/s3/.

   These control words define three items of the title information to be printed at the top and/or bottom of the even and/or odd numbered pages. The value of "n" indicates the title line number. S1, s2, and s3 are three parts of this title line, and will appear left-adjusted, centered, and right-adjusted on this title line.

8. Control words which do not belong to any category described above.

   These control words have a special format of

parameter, and can not be classified into any category described above. These control words are ".CM" (COMMENT) control word, ".CP" (CONDITIONAL PAGE EJECT) control word, ".DM" (DEFINE MACROS) control word, ".FB" (FLOATING BLOCK) control word, ".PA" (PAGE EJECT) control word, ".PN" (PAGE NUMBER TYPE) control word, ".PS" (PAGE NUMBER SYMBOL) control word, and ".RO" (ROMAN) control word. Each one of them has a special format of parameter, and should be treated individually.

III. STRUCTURE OF THE STF SYSTEM AND ITS FEATURES

III.1 System Data Flow

The inputs of STF are the text and control words that comprise the input form of the document. This input form of the document is stored on the disk as an input save file. The user first inputs the text and control words into the computer by using a terminal and an editor program. STF then reads the data of this input save file and produces the output. The output can be sent directly to a printer to produce the final document or it can be saved on the disk as another save file. After that, this output save file can be used to produce the final document or used for further process. Figure 1 shows the system data flow and the structure of the whole system.

III.2 Structure of the STF system

The STF system has been developed with maximum use of state-of-the-art software development techniques. The development techniques used are top-down software design methodology, modular approach, and structured programming. The development of the program concentrates on the highest level of logic first, and then goes down to the details one level at a time. The whole program is divided into more than one hundred subroutines. Each subroutine only has one

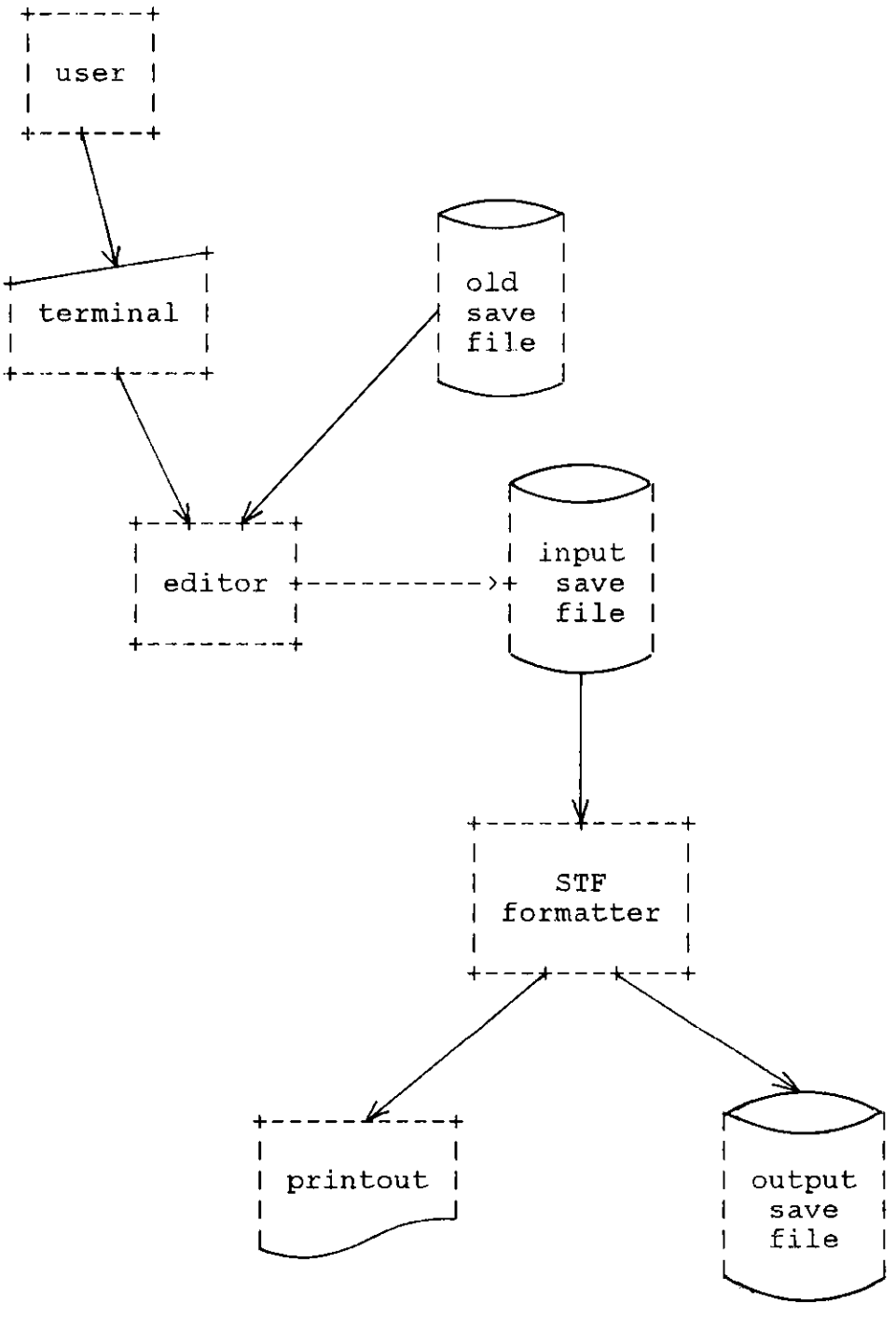


Figure 1. System data flow chart

function and is almost independent to other subroutines.

The control structures used in the program are IF-THEN-ELSE,

WHILE-DO, and REPEAT-UNTIL control structure. No unconditional branching is used. The implementation language used is the VAX/VMS version PASCAL. The reason why I choose this language is that this version of PASCAL has a data type of VARYING OF CHAR which can be used to simplify the implementation of variable-length character strings[4]. It also has some useful built-in functions that make me handle character strings easily.

The whole program consists of three parts : INITIAL, PROCESS, and END. INITIAL and END parts contain some housekeeping operations such as opening files, initiating variables, and terminating process. The main part of the program is the PROCESS part. The PROCESS part can be further divided into two routines : Control Words routine and Text routine. Control Words routine works as a control words dispatcher. When an input line was recognized to be a control word, the Control Words routine sends the input line to an appropriate routine according to its control word name. Each control word has its own processing path. The Control Words routine is a supervisor which matches each control word to its own path.

The Text routine processes the text input lines. It formats the text input lines according to the control words encountered or by default, and produces the output. It decides whether the text input lines should be put into the index and/or table of contents or not. It also decides that

the text input lines are centered, right adjusted, concatenated, or justified. The Control Words routine and Text routine are almost independent of each other, and can be implemented and debugged separately. The advantage is that I can concentrate on a small part of the program at a time, and solve the problem more easily. Figure 2 illustrates the hierarchy chart of the program.

III.3 Features of the STF System

STF is a text formatter primarily used to prepare the simple documents. According to this goal, some complex functions unfrequently used to prepare a simple document are not implemented in STF[1,2,3,12,18]. The following are the functions available in STF.

1. Different adjustment values, margins, and titles on even and odd numbered pages

   In STF, user is allowed to specify different adjustment values, margins, and titles for even and odd numbered pages, and STF can alter them automatically according to the value of the current page number. User need not change these values through the document to obtain the formatting requirements that he wants. All the alternative jobs are done by STF. This advantage is more obvious especially on the documents that are printed on both sides of the paper. It also offers some meaningful benefits on time and efforts of preparing

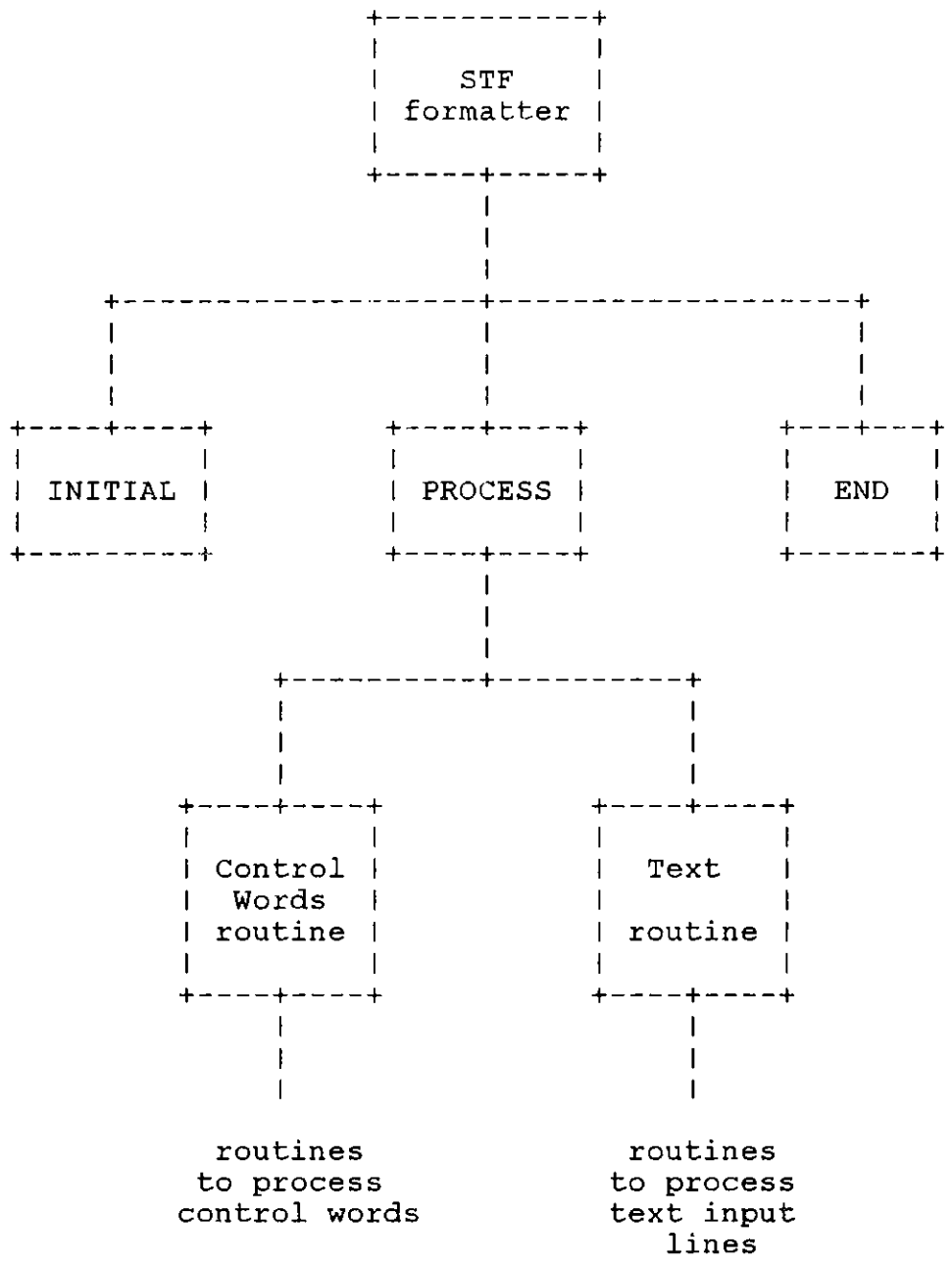


Figure 2. Hierarchy chart of the program

input form of the document.

2. Comments in the input save file

Comments are allowed to exist in the input save file, and STF will ignore them when these comments are encountered. Comments can be anywhere in the input save file and may be seen whenever the input file is edited or printed. Comments may be used to stored unique identifications for use during editing of the file. This advantage is more useful when the document is updated frequently in the few fixed positions. It also can be used to comment the function and use of user defined macros.

3. User defined macros

STF allows user to define his own macros and use them to format the document. The macro defined has a 2-character string as its name, and it should not be the same as that of the system control words, or another user defined macro name. Any character can be used to comprise these two characters. The usage of macros are the same as the system control words except that they can not have the parameters. Text, system control words, and user defined macros can be the contents of a macro. User should take the responsibility to ensure that no iteration is made, or an undefined situation will be happened.

4. Multiple interline spacing

The number of blank lines inserted between the text

output lines of the document is not limited to be 0 and 1 (single spacing and double spacing). User can specify line spacing other than these two numbers to produce other kind of line spacing. This feature is more useful when a special layout of the document is needed.

5. Skipping to the desired numbered page

STF allows user to control the output page ejecting by using ".PA" control word. It also allows user to specify the page number of the next output page. User can control the printing of output page and lets the output text be printed on the top of an even or odd numbered page. STF does this job by comparing the current page number and the parameter of the control word. If they are not matched with each other, STF will produce a blank page and force the page number to the desired one. With this feature, user can specify that the beginnings of the chapters are printed on the top of the desired numbered pages.

6. Floating block of text input lines

STF can keep a block of text input lines encountered at one time and process them at another time. Several blocks of text input lines also can be accumulated together and processed at a time. STF will remember the text input lines in the blocks and produce the output at the place where user specified. This feature is particularly useful to produce the summary of

the document.

7. Keeping a diagram together

In many instances, it is necessary to ensure that a block of text does not be split across a page boundary, for example, the text for diagrams and tables. STF has a facility to keep a block of text input lines be printed together on the current output page or on the next page. This is done by using the ".FD ON/OFF" control word. The block of text input lines, usually comprising a diagram or table, are printed as they are and STF will decide where to print them. If there is sufficient room to accommodate the diagram, it will be printed on the current output page. Otherwise, the diagram will be printed on the next page. With this feature, a diagram will never be split by the fold of the paper, and user need not worry about the position of the diagram.

8. Indent on right margin

STF allows the output to be indented on the right margin without changing the length of the output line. This feature is used together with the indent on the left margin to specify a block of text input lines that have special meaning. It is useful when someone else's statements are referenced in the document. With this feature, user can easily specify the beginning and the end of the text block and print it on the document very

obviously.

9. Indexing

STF has a facility to prepare the index of the document. User only need to specify the text input line he wants it to be indexed, and STF will create the index entity for the text input line and also keep the output page number of this text line printed. If an indexed text line is specified more than once, only one index entity is created in the index structure. STF also trys to eliminate the redundancy of the page numbers of an index entity. Only one occurrence per page number is allowed in an index entity. Index is printed when the ".PI" control word is encountered. User can use this control word anywhere in the input form to produce the index. Index structure is not destroyed after it is printed.

10. Multiple-level table of contents

STF allows that the document has multiple levels in its table of contents. The level of the heading in the table of contents is specified by the user. The current output page number and level number are saved with the heading in the table of contents by STF. When user uses the ".PT" control word to produce table of contents, the level number is used to calculate the first printing position of the heading.

11. Temporary indenting on the left margin

STF allows a single text input line to be temporarily indented on the left margin. ".TI" control word causes next text input line to be indented to the right of a specified column, and does not affect subsequent text input lines. This feature is especially useful when it is used before the first line of a paragraph. It will create an indented paragraph style.

IV. IMPLEMENTATION OF THE STF SYSTEM

IV.1 Structure of Index

The whole index structure is organized as a binary search tree. Each indexed text line comprises a node in this binary search tree. Each node, called Index Entity (IE) node, contains the following information :

1. indexed text line
2. the string form of the indexed text line
3. the location of the first page number node of its page number linked list
4. the location of the root of its left subtree
5. the location of the root of its right subtree

Every time a text line is indexed, this binary search tree is searched to find whether the same text line has been indexed before. If this text line exists in the binary search tree, the current page number is compared to the last inserted page number of this IE node. If these two page numbers are the same, it means a text line is indexed twice on the same output page, and no page number is inserted to this IE node. Otherwise, the current page number is inserted into this IE node. If this text line does not exist in the binary search tree, a new IE node is created and inserted into the leaves of the index binary search tree. The current page number is also inserted into this

new IE node.

The page numbers of an IE node are arranged as a linked list. Each page number comprises a node, called Page Number (PN) node. Each PN node has the following information :

1. page number of output page that indexed text line printed
2. the location of next Page Number node of the same page number linked list

Figure 3 shows the data structure of PN nodes and the structure of the page number linked list.

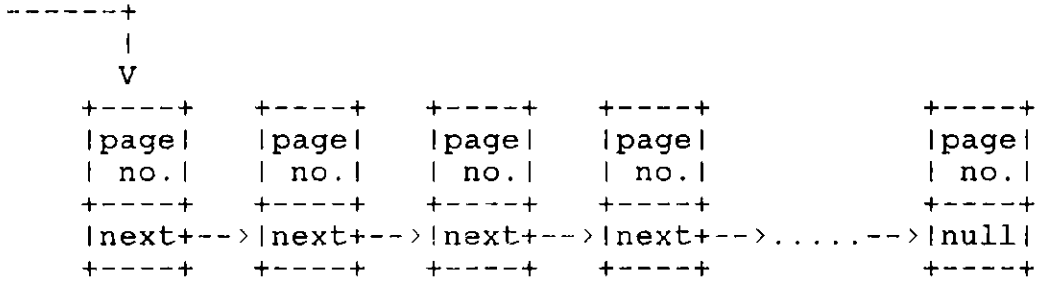


Figure 3. Structure of PN nodes and page number linked list

There are several advantages provides by using a binary search tree to implement the index structure.

1. The size of the index structure is not limited in a fixed range. If the index structure is organized as an array, the size of the array has to be decided in advance by the developer. It is very hard to decide because a large array may cause a lot of memory spaces wasted, and a small one can not accommodate a big index structure.

With binary search tree as its data structure, the index structure can be as large as possible it wishes, and the memory spaces used are just as they are needed.

2. The processing time to index a text line is reduced. With binary search tree, the searching algorithm used is binary search tree searching method. Its average searching time is about O(LOG n) for a particular node of a binary search tree containing n nodes. If the index structure is implemented as an array or as a linked list, the only searching method that can be used is sequential searching method. Its average searching time is O(n/2) for a particular node. It is obvious that the processing time of binary search tree searching method is much less than that of sequential searching method.

Figure 4 illustrates the data structure of the index structure.

IV.2 Structure of Table of Contents

The table of contents is arranged as a linked list. Each node of the linked list contains the following information :

1. heading line
2. page number of output page that the heading line printed
3. level number of the heading line
4. the location of the next node

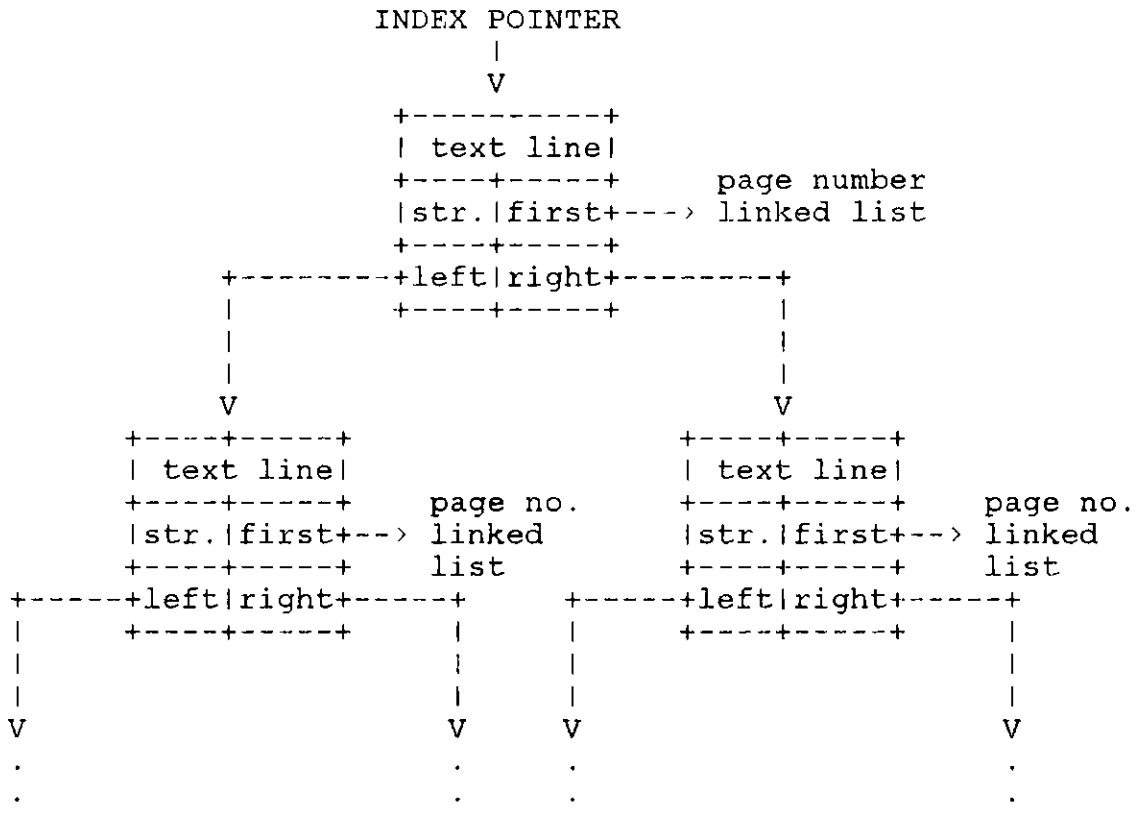


Figure 4. Data structure of index structure

There are two pointers associated with this linked list, that is, Table of Contents Head Pointer (TCHP) and Table of Contents Tail Pointer (TCTP). TCHP points to the first node of the linked list, and is used when table of contents is printed. TCTP points to the last node of the linked list, and is used when a heading line is put into the table of contents. When a heading line needs to be put into the table of contents, a new node is created and inserted after the node pointed by TCTP. And then the TCTP is updated and points to this new node. The page number field stores the page number of the output page where the heading

line is printed, and the level number field is used to store the level number of the heading line. The level number is used to determine the printing position of this heading line when the table of contents is printed. The printing position of the heading line is calculated by using the following formula :

printing position = 3 * (levle-no. - 1) + 1

From the formula we know that the heading line whose level number is 1 will be printed from column 1. The control word used to specify the heading line in input file is the ".TC" control word. The parameter of this control word specifies the level number of the heading line. If the parameter value is 0, no heading line is put into the table of contents. STF will ignore this control word because of its invalid parameter value. Figure 5 shows the structure of the table of contents.

IV.3 Structure of Titles

There are four kinds of titles in STF. They are top and bottom titles of the even and odd numbered pages. Each title has the same data structure. I use a linked list to implement it. Each title linked list has fixed size and contains 30 nodes at most. Each node has four fields containing the following information :

1. left part of the title line
2. middle part of the title line

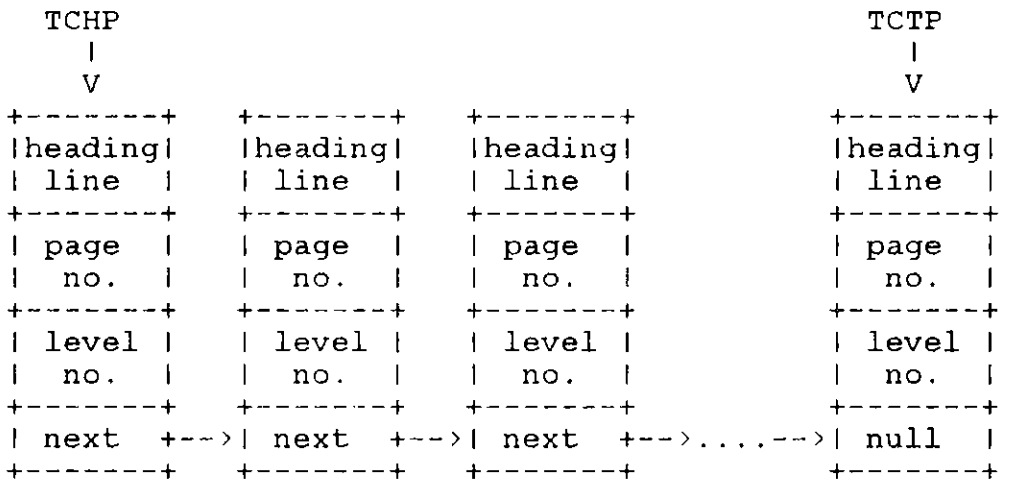


Figure 5. Structure of the table of contents

3. right part of the title line
4. the location of next node in the same title linked list

Each title linked list is created and initiated in the INITIAL part, and the default values of three parts of each title line are the null string. These three parts will produce one title line. The left part is left adjusted on the title line. The middle part is centered, and the right part is right adjusted on the title line. The length of the title line is equal to the length of the text output line. When STF produces the title line, the middle part may overlay the left part if necessary, and the right part may overlay the middle part if necessary. When a title is printed, only the nodes in the range of heading or footing spacing need to be printed rather than all the nodes of the title linked list are printed. The range of heading and

footing spacing are specified by the ".HS" and ".FS" control words. The three parts of a title line is put into the corresponding node by using several changing title control words, such as, ".TT" (TOP TITLE), ".BT" (BOTTOM TITLE), ".ET" (EVEN PAGE TOP TITLE), etc. The numeral parameter of these control words specifies the line number of this title line. Figure 6 illustrates the structure of a title linked list.

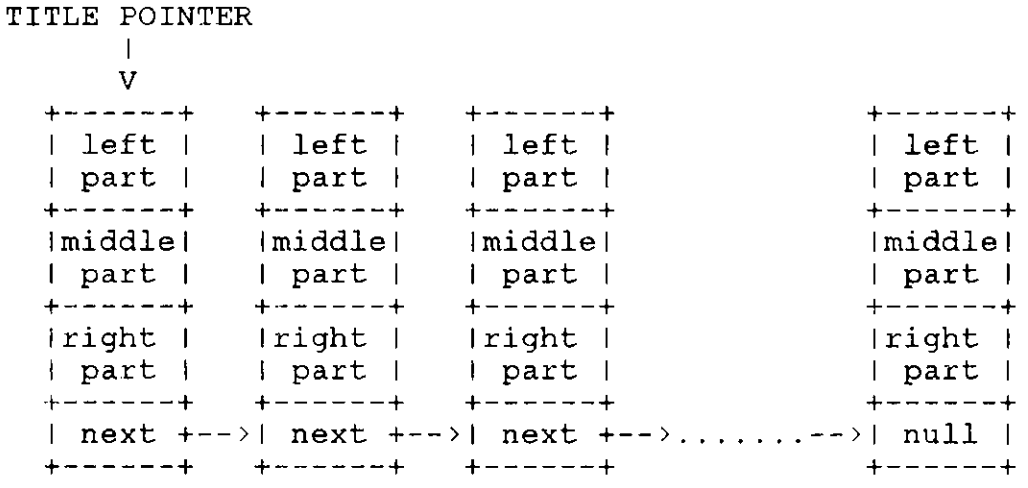


Figure 6. Structure of a title linked list

IV.4 Structure of Macros

User defined macros are organized in a binary search tree. Each macro comprises a node in the binary search tree, and the contents of a macro are arranged in a linked list. Each node of the macros binary search tree, called Macro node, has four fields and contains the following information :

1. name of macro
2. the location of the first node of its contents linked list
3. the location of the root of its left subtree
4. the location of the root of its right subtree

The contents of a macro are organized in a linked list, and there is a field in Macro node to keep the location of the first node of this linked list. Each node of the contents linked list contains two fields and contains the following information :

1. data of the macro
2. the location of next node in the same contents linked list

Figure 7 shows the data structure of the contents linked list of a macro node.

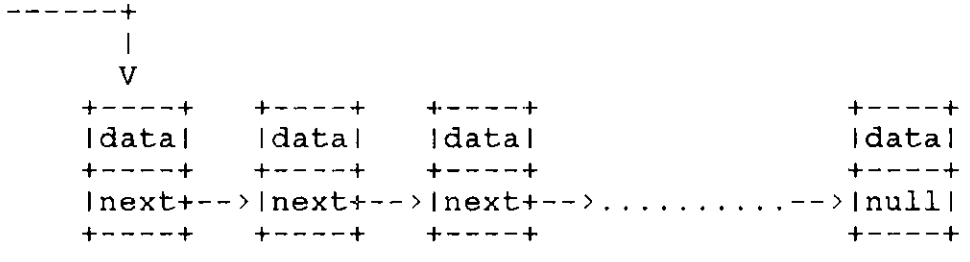


Figure 7. Data structure of the contents linked list

Each time when a ".DM name" control word is encountered, the name parameter is used to compare with the names of the system control words. If they are not matched, the name parameter is compared to the names of the user

defined macros that are already defined. A macro's name should not be the same as either the names of the system control words or the names of user defined macros that are already used, otherwise an error condition is occurred and the formatting process will be terminated. After the name is recognized as a unique one, a Macro node is created and inserted into the appropriate position in the macros binary search tree. The subsequent input lines are read in and linked at the end of the contents linked list until the ".DM OFF" control word is encountered. This control word terminates the macro definition operation by putting the location of the first node of the contents linked list into new Macro node. With this organization, user can define as many as macros he wishes. The contents of a macro can also be as many as they are. There is no limitation for the number of macros that can be defined and the size of a macro.

The macros are used just as the system control words. A period preceding its name indicates that this input line should be treated as a control word. When an input line is recognized as a control word, its name is first compared to the names of the system control words. If it matches a system control word, the input line is processed according to the normal routine. Otherwise, the macros binary search tree will be searched. If the name exists in the macros binary search tree, the datum in the nodes of the contents

linked list are used as they are read from input file. If it does not exist in the macros binary search tree, an error condition is occurred and the formatting process is terminated. Figure 8 illustrates the structure of the user defined macros.

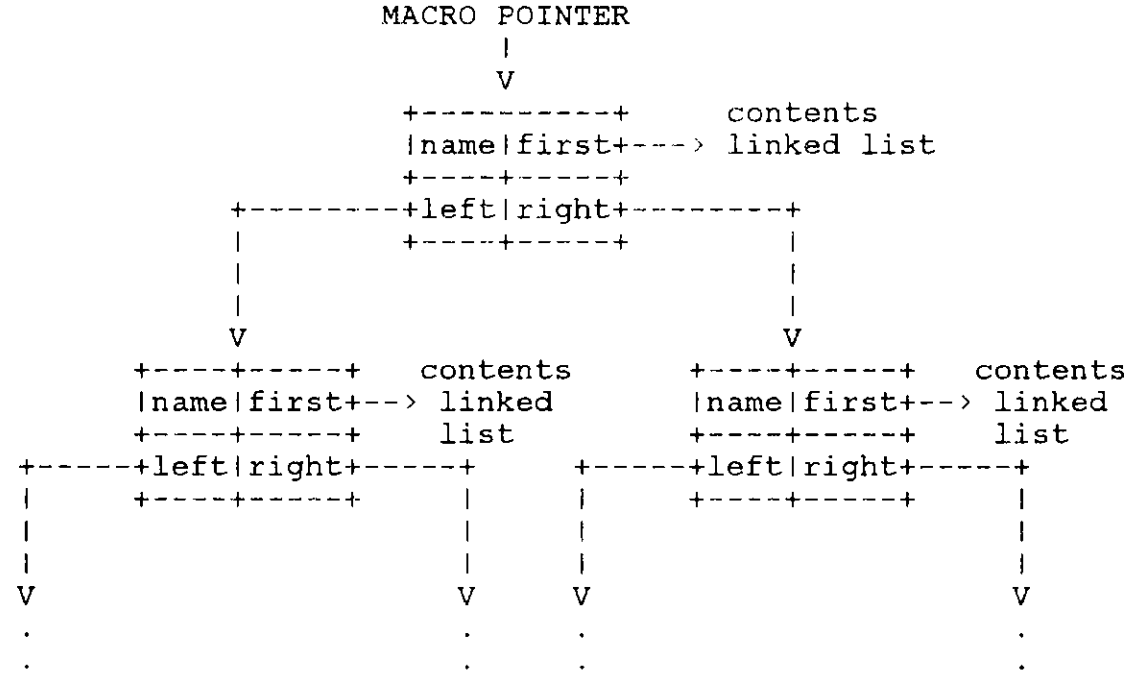


Figure 8. Structure of user defined macros

IV.5 Boldface and Underline Effects

In STF, boldface effect is obtained by printing a character twice at the same position. STF uses BACKSPACE character (ASCII"8") to force the printer head back one space, and prints the same character once again[4,5]. If boldface is in effect, each character in a text input line is converted into

character BACKSPACE character

Though each character is converted into three characters, these three characters only produce one character on the output device. They should be counted as one character in the text input line.

The underline effect is also obtained by using the same method in STF except that only alphanumeric characters are underlined. Each word in the text input line is underlined by converting each alphanumeric character into

_ BACKSPACE character

On many video terminals, underlining a character erases it, so STF prints the underline first. With that way you get to see the character even if the underline is erased.

IV.6 Centralization and Right Adjustment

Centering and right adjusting a text line are obtained by calculating the first printing position that the text line is printed, and padding the extra spaces in the front of the text line. The formula used to calculate the first printing position of centering a text line

printing position = (A - B) / 2 + 1

and the formula used in right adjusting a text line is

printing position = A - B + 1

where A is the length of the text output line, and B is the real length of the text line.

IV.7 Determining the Line Breaking Point

A basic problem in text formatting is that of determining the breaking points to separate a string of words into lines to obtain a formatted paragraph. When formatted text is required to be aligned with both the left and right margins, the choice of breaking points greatly affects the quality of the formatted document. There are several methods to solve this line break problem[1,3,6,11]. The method used in STF is the line-by-line method. With this method, the text output lines are to be formed by shifting words to or from the next text input line so that the resulting text output line will be as close to the line length as possible without exceeding it or splitting a word into two lines. When a text input line is read in, STF examines this text input line and determines the line breaking point according to the current line length. The unused words are stored in a buffer and used to form next text output line when another text input line is read in.

The line-by-line method is simple to implement. It uses less processing time and memory spaces than any other methods[6,11]. Though it does not always produce the best result when text is being justified, it is tolerant and works well in many situations.

IV.8 Distributing the Extra Spaces on a Text Output Line

When the justification is required to format text,

extra spaces are inserted into the words of the output text line so that the resulting text line is aligned with both the left and right margins. There are several methods to distribute these additional spaces into a text line, such as, pseudo-random method and particular-character-first method[1,3,6,16,18]. The method used in STF is the alternation method, that is, distributing the extra spaces for each text line between the words starting from the left and right sides alternately. If line one has additional spaces, the spaces between the first and second words, the second and third words and so on will be increased by one until the extra spaces are used up. If line two also has extra spaces, it is now used up by increasing spaces between the last and the last but one words, the last but one and the preceding word, and so on. The aim of this strategy is to avoid "rivers" of white space running down the length of the page. This method is simple and easy to implement. It also works well in most situations. In STF, a variable, called DIRECTION, is used to keep the direction of distributing the extra spaces. It is a BOOLEAN variable and changes value from TRUE to FALSE or FALSE to TRUE after a text output line is formed.

V. CONCLUSIONS

The text formatter is a tool for people who write. It is also a useful tool in offices and the printing industry. A text formatter should be classified according to its complexity and functions[16]. Generally, a complex text formatter needs more memory space and longer processing time than a simple one. The users also need to select their own text formatter according to their demands. It is not necessary to use a complex text formatter for a simple task.

The primary goal of my project is to design and develop a small text formatter used for preparing simple documents. The final STF system is a program that contains about 3000 lines of code. It has been developed by using several design and programming methodologies. The whole program is composed of a set of modules, and each of them deals with a particular function of the overall formatting process. Each module has been tested alone and the whole system also has been tested by using several different sets of data. The results are just like what I wish.

STF is a medium sized text formatter with simple algorithm. Because of its modularity, it is easy to debug and extend. Only one system dependent feature is used, that is, the VARYING OF CHAR data type[4]. Because many versions PASCAL support this kind of data type, STF is easy to

transfer from one system to another without any change. This feature allows STF to be used on a lot of different systems.

VI. FURTHER POSSIBLE EXTENSIONS

As I said above, STF is a small and simple text formatter. Some complex functions unfrequently used to format a simple document are not implemented in STF. There are lots of things that could be added to STF and make it more useful in formatting complicated document[1,3,17,18].

VI.1 Extra Space after Sentence

Most people prefer an extra space after the period that terminates a sentence, that is, two spaces after that period[1]. This could be done by examining the last character of each word when the text output line is formed. If a period is found, an extra space will be added after the period.

VI.2 Setting Backspace Character

Some documents may have special symbol that needs two or more characters to form it[1,3]. This can be achieved by defining a special punctuation as an user defined backspace character, and changing it to real backspace character (ASCII"8") when the text input line is read in. When a text input line is read in, the text formatter first changes each occurrence of user defined backspace character to the real backspace character. And then this text input line is processed following the normal routine. There should be a

new control word that allows the user to change this user defined backspace character to what he wants.

VI.3 Setting Output Tabs

Output tabs are useful when there are tables in the document. The user needs to specify a single punctuation as the user defined tab character and the positions that each occurrence of user defined tab character stops[1,3,18]. Text formatter first examines the text input line to find whether it contains user defined tab characters or not. If it contains user defined tab characters, the text line is rearranged by aligning each occurrence of user defined tab character to its position. Extra spaces are inserted into the text line to replace the user defined tab characters and adjust words to the appropriate positions. The resulting text line is then processed to produce output text.

VI.4 Multiple Control Words on a Single Input Line

In STF, an input line only can contain one control word. It is possible to allow more than one control word appeared on a single input line by separating each control word with a special character[1]. This special character then is called the control word separator. It may be a default character or a user defined character. In general, an input line that contains multiple control words is entered as follows :

.xx parameters;.yy parameters;.....

where semicolon (";") works as a control word separator. This can be done by examining each input line that the character at column 1 is a period. If this input line contains control word separator, it will be split into several control words. These control words then can be processed one at a time. The buffer used to hold these control words can be a linked list because it is easy to implement and no limit exists on the number of the control words contained on a single input line.

VI.5 Redefining Macros

It is useful if a macro can be redefined. Usually, only few user defined macros are used all over the document. Most of them are used in the different portions of the document. With this facility, the number of user defined macro can be reduced, and the processing time of a macro also can be decreased. This facility can be achieved by continuing process the Define Macro (".DM") control word instead of raising an error condition when the name parameter of the control word is found in the macros binary search tree. When the same macro name is found, the old contents linked list of this Macro node is thrown away. New contents linked list then is formed and linked to this Macro node in the normal way.

VI.6 Footnotes

Footnote is the formatted text that will be printed at

the bottom of the text area. Many documents require the use of footnotes. The formatting of the footnotes should not disturb the formatting of the normal text[1,18]. I think the footnotes should be formatted when they are encountered, and saved in a buffer. A linked list is an appropriate data structure to implement this buffer. When a footnote is processed over, the text formatter goes back to process the normal text, and leaves sufficient room at the bottom of the page to ensure that the footnote text is printed at the same page. If enough room does not exist, the text formatter should print the footnote text immediately, and those footnote text lines that are not printed at this output page will be printed at the bottom of the text area of the next output page. Text formatter should keep the number of output lines that footnote text has and uses it to reserve sufficient room in the output page so that footnote text can be printed in the same page.

VI.7 Conditional Sections

Some times not the total document need be printed, only the selected portions of the document are used. This can be done by associating a conditional section code with selected portion of the document, and unclassified portions should have a special value in their code by default[1,3,18]. When the document is produced, user specifies the output level. Text formatter compares the value of the conditional section

code with the output level to decide whether this portion needs to be formatted and output. This facility is useful when a summary of the document needs to be produced.

VI.8 Multiple Input Files

With any large document, it is an advantage to create a separate input file for each its constituent parts, and then construct a small "driver" file to process each file, in the desired order[1,18]. This needs that the text formatter can read input either from its standard input file or from another input file. The text formatter can keep another logical file for the use of this purpose. When the end of this additional input file is encountered, the text formatter goes back to read in the next input line followed in the standard input file. A special control word will be presented to invoke the text formatter reading input lines from another input file. It is obvious that the save file name of this additional input file should be one of the parameters of this control word. When this control word is encountered, text formatter opens this additional input file and reads the input from it until the end of file is reached. Then text formatter closes this file and goes back to read the input lines from the standard input file.

VI.9 Multiple Columns Per Page

The text in newspaper and magazines is formatted in multiple columns per page. A number of journals also use

this style (some IEEE publications, for example). This facility is hard to implement, and I do not have any ideal about how to implement it simply and which kind of data structure will be used to implement it.

VI.10 Hyphenation

STF fills lines by packing as many words onto a line as possible without exceeding the line length or splitting a word into two lines. Hyphenation increases the number of words in a line on the average, and thus improves the appearance of the output[1]. The problem is to design a reasonably accurate scheme for hyphenating English words by program. It needs particular study on the composition of English words and should be developed by the group containing computer professionals and English linguists.

VI.11 Detailed Error Message

In STF, the error message only tells the user which input line causes the error condition. No more details can be obtained from the error message. It is not a good ideal to leave the user to find the problem by himself. The error message should contain some detailed information about the possible reason of the error condition and how to correct it[1,3,18]. The error message system of STF can be enhanced by numbering each error condition and using a table to store the error messages of these error conditions in their corresponding positions. An one dimensional array is

suitable to implemente this error message table, and the error condition code is the subscript of the element that its corresponding error message stored. Every time an error condition occurs, this error message table is examined, and the error message of that error condition is printed out. With this error message system, more details can be provided to the user when an error condition occurs.

VII. APPENDICES

Appendix A contains the source listing of the program. Appendix B is the sample input data. Appendix C is the output produced from the sample input data. Appendix D is the STF user's manual, and Appendix E is the bibliography.

Appendix A

Source Program Listing

In the source program listing, the "@" is used to substitute the pointer symbol "^" in the PASCAL language. All the "@" symbol should be replaced by "^" in the program when it is typed in.

```
PROGRAM FORMAT (INPUT, OUTPUT, INF, OUTF);
CONST
        MAXSTR = 600;    { MAX. LENGTH OF A STRING }
        COMMA = ',';
        PERIOD = '.';
        BLANK = ' ';
        PLUS = '+';
        MINUS = '-';
        UDLN = '_';

        NULL = CHR(0);   { EMPTY CHAR. }
        BS = CHR(8);     { BACK SPACE }
        LF = CHR(10);    { LINE FEED }
        FF = CHR(12);    { FORM FEED }
        CR = CHR(13);    { CARRIAGE RETURN }

TYPE
                  { STRING TYPE }
        STRING = VARYING [MAXSTR] OF CHAR;
                  { COMMAND TYPE }
        CMDTYPE = PACKED ARRAY [1..2] OF CHAR;
                  { TEXT TYPE }
        TEXTTYPE = PACKED ARRAY [1..80] OF CHAR;

        { TEXT LINKED LIST }
        TX_P = @TEXT_N;
        TEXT_N = RECORD
                    TX_TXT : STRING;
                    TX_NXT : TX_P;
                 END;

        { MACRO BINARY SEARCH TREE }
        DM_P = @MACRO_N;
        MACRO_N = RECORD
                     DM_LFT : DM_P;
                     DM_NAM : CMDTYPE;
                     DM_RIT : DM_P;
                     DM_FST : TX_P;
                  END;

        { PAGE NUMBER LINKED LIST }
        IN_P = @IN_N;
        IN_N = RECORD
                  IN_INT : INTEGER;
                  IN_NXT : IN_P;
               END;
```

```
        { INDEX BINARY SEARCH TREE }
        IX_P = @INDEX_N;
        INDEX_N = RECORD
                      IX_LFT : IX_P;
                      IX_TXT : TEXTTYPE;
                      IX_STR : STRING;
                      IX_PGN : IN_P;
                      IX_RIT : IX_P;
                  END;

        { TABLE OF CONTENTS LINKED LIST }
        TC_P = @TC_N;
        TC_N = RECORD
                   TC_TXT : STRING;
                   TC_PGN : INTEGER;
                   TC_LVN : INTEGER;
                   TC_NXT : TC_P;
               END;

        { TITLE LINKED LIST }
        TT_P = @TITLE_N;
        TITLE_N = RECORD
                      TT_LFT : STRING;
                      TT_MDL : STRING;
                      TT_RIT : STRING;
                      TT_NXT : TT_P;
                  END;

VAR
        INF : TEXT;   { INPUT DATA FILE }
        OUTF : FILE OF STRING;   { OUTPUT DATA FILE }

        INSTR,   { INPUT DATA AREA }
        WKSTR,   { WORKING DATA AREA }
        OUTSTR,  { OUTPUT DATA AREA }
        CRLF     { CARRIAGE RETURN & LINE FEED }
               : STRING;

        PAUSE,         { PAUSE AT EACH PAGE = FALSE }
        START_FLAG,    { START FLAG = TRUE }
        ERROR          { ERROR FLAG = FALSE }
               : BOOLEAN;
        CARDNO   { INPUT CARD NO. = 0 }
               : INTEGER;
```

```
SYSDM    { MACRO ROOT POINTER }
       : DM_P;
SYSIX    { INDEX ROOT POINTER }
       : IX_P;
SYSTCH,  { TABLE OF CONTENTS HEAD POINTER }
SYSTCE   { TABLE OF CONTENTS END POINTER }
       : TC_P;
SYSTTE,  { EVEN PAGE TOP TITLE POINTER }
SYSTTO,  { ODDPAGE TOP TITLE POINTER  }
SYSBTE,  { EVEN PAGE BOTTOM TITLE POINTER }
SYSBTO   { ODDPAGE BOTTOM TITLE POINTER  }
       : TT_P;
SYSFDH,  { FLOATING DIAGRAM HEAD POINTER  }
SYSFDE,  { FLOATING DIAGRAM END POINTER  }
SYSFBH,  { FLOATING BLOCK HEAD POINTER }
SYSFBE   { FLOATING BLOCK END POINTER }
       : TX_P;

SYSPS,     { PAGE NUMBER SYMBOL = '%' }
INDEXCHAR { FIRST CHAR. IN INDEX ENTRY }
      : CHAR;

SYSPN,   { PRINTED PAGE NO. TYPE = TRUE }
ROMBG,   { BIG ROMAN NUMERAL OR NOT = FALSE }
COBOL,   { CONCATENATE = TRUE }
JUBOL,   { JUSTIFY = TRUE }
BDBOL,   { BOLD = FALSE }
CEBOL,   { CENTER = FALSE }
RIBOL,   { RIGHT ADJUST = FALSE }
USBOL,   { UNDERSCORE = FALSE }
UPBOL,   { UPPERCASE = FALSE }
IXBOL,   { INDEX = FALSE }
PAGETYPE, { EVEN PAGE OR ODD PAGE = TRUE }
DIRECTION { DIRECTION TO INSERT BLANKS = TRUE }
          : BOOLEAN;
```

```
SYSLS,   { LINE SPACE = 0 }
SYSTME,  { EVEN TOP MARGIN = 6 }
SYSTMO,  { ODD TOP MARGIN = 6 }
SYSHSE,  { EVEN HEADING SPACE = 1 }
SYSHSO,  { ODD HEADING SPACE = 1 }
SYSHME,  { EVEN HEADING MARGIN = 1 }
SYSHMO,  { ODD HEADING MARGIN = 1 }
SYSBME,  { EVEN BOTTOM MARGIN = 6 }
SYSBMO,  { ODD BOTTOM MARGIN = 6 }
SYSFME,  { EVEN FOOTING MARGIN = 1 }
SYSFMO,  { ODD FOOTING MARGIN = 1 }
SYSFSE,  { EVEN FOOTING SPACE = 1 }
SYSFSO,  { ODD FOOTING SPACE = 1 }
SYSADE,  { EVEN ADJUST = 0 }
SYSADO,  { ODD ADJUST = 0 }
SYSIND,  { INDENT = 0 }
SYSIR,   { INDENT RIGHT = 0 }
SYSPL,   { PAGE LENGTH = 66 }
SYSLL,   { LINE LENGTH = 60 }

SKVAL,   { SKIP = 0 }
SPVAL,   { SPACE = 0 }
LBVAL,   { LEADING BLANKS = 0 }
UNVAL,   { UNDENT = 0 }
TIVAL,   { TEMPORARY INDENT = 0 }
TXVAL,   { TEXT CHAR. PER LINE = 60 }
CEVAL,   { CENTER = 0 }
RIVAL,   { RIGHT ADJUST = 0 }
LINENO,  { CURRENT PRINTED LINE NO. = 0 }
PAGENO,  { CURRENT PAGE NO. = 1 }
CPVAL,   { CONDITIONAL PAGE EJECT = 0 }
USVAL,   { UNDERSCORE = 0 }
UPVAL,   { UPPERCASE = 0 }
BDVAL,   { BOLD = 0 }
TCVAL,   { TABLE OF CONTENTS LEVEL = 0 }
TXBOTE,  { TEXT AREA OF EVEN PAGE = 54 }
TXBOTO,  { TEXT AREA OF ODDPAGE = 54 }
SYSAD,   { ADJUST OF CURRENT PAGE = 0 }
TXBOT    { TEXT AREA OF CURRENT PAGE = 54 }
      : INTEGER;
```

```
{-------------------------------------------------}
{  DEFINITION OF SUBROUTINES                      }
{-------------------------------------------------}

    PROCEDURE ADDINDEXNO_PROC (P : IX_P;
                               INQ : IN_P);
    FORWARD;

    FUNCTION ALLDIGITS_FCTN (S : STRING) : BOOLEAN;
    FORWARD;

    PROCEDURE ARRANGEOUTSTR_PROC (VAR S1, S2 : STRING;
                                  I : INTEGER);
    FORWARD;

    FUNCTION BACKWARD_FCTN (S : STRING;
                            I : INTEGER) : STRING;
    FORWARD;

    FUNCTION BDSTR_FCTN (S : STRING) : STRING;
    FORWARD;

    PROCEDURE BREAK_PROC (B : BOOLEAN);
    FORWARD;

    PROCEDURE BUILDTT_PROC (VAR P : TT_P);
    FORWARD;

    FUNCTION BUILDTTPART_FCTN (S : STRING) : STRING;
    FORWARD;

    PROCEDURE CE_PROC (S : STRING;
                       I : INTEGER);
    FORWARD;

    PROCEDURE CO_PROC (S : STRING);
    FORWARD;

    PROCEDURE COMMAND_PROC (S : STRING);
    FORWARD;

    FUNCTION CONVERSE_FCTN (S : STRING) : CMDTYPE;
    FORWARD;

    PROCEDURE CP_PROC (S : STRING);
    FORWARD;
```

```
PROCEDURE DM_PROC (S : STRING);
FORWARD;

PROCEDURE END_PROC;
FORWARD;

PROCEDURE FB_PROC (S : STRING);
FORWARD;

PROCEDURE FD_PROC (S : STRING);
FORWARD;

PROCEDURE FO_PROC (S : STRING);
FORWARD;

FUNCTION FORWARD_FCTN (S : STRING;
                       I : INTEGER) : STRING;
FORWARD;

PROCEDURE GETPARA_PROC (VAR S : STRING;
                        VAR P : STRING;
                        VAR R : BOOLEAN);
FORWARD;

PROCEDURE GETWORD_PROC (VAR S : STRING;
                        VAR W : STRING;
                        VAR L : INTEGER;
                        VAR R : BOOLEAN);
FORWARD;

PROCEDURE INITFILE_PROC;
FORWARD;

PROCEDURE INITIAL_PROC;
FORWARD;

PROCEDURE INITTT_PROC;
FORWARD;

PROCEDURE INITVAL_PROC;
FORWARD;

FUNCTION INRANGE_FCTN (C, R1, R2 : CHAR) : BOOLEAN;
FORWARD;

FUNCTION INTSTR_FCTN (S : STRING) : INTEGER;
FORWARD;
```

```
FUNCTION INTTOARB_FCTN (I : INTEGER) : STRING;
FORWARD;

PROCEDURE INTTOROM_PROC (I : INTEEGR;
                         VAR S : STRING;
                         VAR E : BOOLEAN);
FORWARD;

FUNCTION ISAEO_FCTN (S : STRING) : INTEGER;
FORWARD;

FUNCTION ISALPHA_FCTN (C : CHAR) : BOOLEAN;
FORWARD;

FUNCTION ISCW_FCTN (A : CMDTYPE) : BOOLEAN;
FORWARD;

FUNCTION ISDIGIT_FCTN (C : CHAR) : BOOLEAN;
FORWARD;

FUNCTION ISEVEN_FCTN (I : INTEGER) : BOOLEAN;
FORWARD;

FUNCTION ISLOWER_FCTN (C : CHAR) : BOOLEAN;
FORWARD;

FUNCTION ISNUMBER_FCTN (S : STRING) : INTEGER;
FORWARD;

FUNCTION ISOO_FCTN (S : STRING) : INTEGER;
FORWARD;

FUNCTION ISPUNCT_FCTN (C : CHAR) : BOOLEAN;
FORWARD;

FUNCTION ISUPPER_FCTN (C : CHAR) : BOOLEAN;
FORWARD;

FUNCTION ISXXOFF_FCTN (XX : CMDTYPE;
                       S : STRING;
                       VAR CMND : CMDTYPE) : INTEGER;
FORWARD;

PROCEDURE IX_PROC (S : STRING);
FORWARD;

PROCEDURE LBVAL_PROC;
FORWARD;
```

```
PROCEDURE LINKTEXT_PROC (VAR HP, EP : TX_P;
                         S : STRING);
FORWARD;

PROCEDURE LISTLINK_PROC (VAR DH, DE : TX_P;
                         XX : CMDTYPE;
                         VAR J : INTEGER);
FORWARD;

PROCEDURE LS_PROC (S : STRING);
FORWARD;

FUNCTION LWSTR_FCTN (S : STRING) : STRING;
FORWARD;

PROCEDURE MACRO_PROC (P : DM_P);
FORWARD;

FUNCTION MAKESTRING_FCTN (I : INTEGER;
                          C : CHAR) : STRING;
FORWARD;

FUNCTION MAX_FCTN (I, J : INTEGER) : INTEGER;
FORWARD;

FUNCTION MIN_FCTN (I, J : INTEGER) : INTEGER;
FORWARD;

PROCEDURE MINUSONE_PROC (VAR I : INTEGER;
                         VAR E : BOOLEAN);
FORWARD;

PROCEDURE NEWPAGENO_PROC;
FORWARD;

PROCEDURE OTHERWISE_PROC (S : STRING;
                          A : CMDTYPE);
FORWARD;

PROCEDURE PA_PROC (S : STRING);
FORWARD;

PROCEDURE PARA0_PROC (S : STRING;
                      VAR E : BOOLEAN);
FORWARD;
```

```
PROCEDURE PARA1_PROC (S : STRING;
                      VAR P1 : STRING;
                      VAR E : BOOLEAN);
FORWARD;

PROCEDURE PARA2_PROC (S : STRING;
                      VAR P1, P2 : STRING;
                      VAR E : BOOLEAN);
FORWARD;

PROCEDURE PARA2NT_PROC (S : STRING;
                        VAR P1, P2 : STRING;
                        VAR E : BOOLEAN);
FORWARD;

PROCEDURE PARA3_PROC (S : STRING;
                      VAR P1, P2, P3 : STRING;
                      VAR E : BOOLEAN);
FORWARD;

FUNCTION PARTDIGITS_FCTN (S : STRING;
                          I, J : INTEGER) : BOOLEAN;
FORWARD;

PROCEDURE PN_PROC (S : STRING);
FORWARD;

PROCEDURE PREPAREINDEXNO_PROC (VAR S : STRING;
                               INP : IN_P);
FORWARD;

PROCEDURE PRINTBLANKPAGE_PROC;
FORWARD;

PROCEDURE PRINTFB_PROC;
FORWARD;

PROCEDURE PRINTFOOTHEAD_PROC;
FORWARD;

PROCEDURE PRINTINDEX_PROC (IP : IX_P);
FORWARD;

PROCEDURE PRINTLINES_PROC (I : INTEGER);
FORWARD;

PROCEDURE PRINTSPACELINE_PROC;
FORWARD;
```

```
PROCEDURE PRINTTEXT_PROC (S : STRING);
FORWARD;

PROCEDURE PRINTTC_PROC (TP : TC_P);
FORWARD;

PROCEDURE PRINTTITLE_PROC (P : TT_P);
FORWARD;

PROCEDURE PROCESS_PROC (S : STRING);
FORWARD;

PROCEDURE PS_PROC (S : STRING);
FORWARD;

PROCEDURE PUTHF_PROC (E : BOOLEAN);
FORWARD;

PROCEDURE PUTINDEX_PROC;
FORWARD;

PROCEDURE PUTTC_PROC;
FORWARD;

PROCEDURE PUTTITLE_PROC (V1, V2, V3 : INTEGER;
                         P : TT_P);
FORWARD;

PROCEDURE RO_PROC (S : STRING);
FORWARD;

FUNCTION REVERSE_FCTN (S : STRING) : STRING;
FORWARD;

PROCEDURE RI_PROC (S : STRING;
                   I : INTEGER);
FORWARD;

FUNCTION RMLB_FCTN (S : STRING) : STRING;
FORWARD;

FUNCTION RMTB_FCTN (S : STRING) : STRING;
FORWARD;

PROCEDURE SEARCHINDEX_PROC (T : TEXTTYPE;
                            VAR P : IX_P;
                            VAR I : INTEGER);
FORWARD;
```

```
PROCEDURE SEARCHMACRO_PROC (A : CMDTYPE;
                            VAR P : DM_P;
                            VAR I : INTEGER);
FORWARD;

FUNCTION SELECT_FCTN (E : BOOLEAN;
                      J, K : INTEGER) : INTEGER;
FORWARD;

PROCEDURE SEPTITLE_PROC (S : STRING;
                         VAR S1, S2, S3 : STRING;
                         VAR E : BOOLEAN);
FORWARD;

PROCEDURE SET1N_PROC (S : STRING;
                      VAR I : INTEGER;
                      E : BOOLEAN);
FORWARD;

PROCEDURE SET1NOO_PROC (S : STRING;
                        VAR I : INTEGER;
                        VAR B : BOOLEAN;
                        E : BOOLEAN);
FORWARD;

PROCEDURE SETAEON_PROC (S : STRING;
                        VAR I, J : INTEGER;
                        E : BOOLEAN);
FORWARD;

PROCEDURE SETAEONT_PROC (S : STRING;
                         P, Q : TT_P;
                         E : BOOLEAN);
FORWARD;

PROCEDURE SETN_PROC (S : STRING;
                     VAR I : INTEGER);
FORWARD;

PROCEDURE SETNT_PROC (S : STRING;
                      P : TT_P;
                      I : INTEGER);
FORWARD;

FUNCTION SETNUM_FCTN (I : INTEGER;
                      S : STRING) : INTEGER;
FORWARD;
```

```
PROCEDURE SETOO_PROC (S : STRING;
                      VAR B : BOOLEAN);
FORWARD;

PROCEDURE SETTITLE_PROC (P : TT_P;
                         I : INTEGER;
                         S1, S2, S3 : STRING);
FORWARD;

PROCEDURE SK_PROC (S : STRING);
FORWARD;

PROCEDURE SP_PROC (S : STRING);
FORWARD;

FUNCTION SPREAD_FCTN (S : STRING;
                      I : INTEGER) : STRING;
FORWARD;

FUNCTION STRCOPY_FCTN (S : STRING) : STRING;
FORWARD;

FUNCTION STRTOTEXT_FCTN (S : STRING) : TEXTTYPE;
FORWARD;

PROCEDURE TC_PROC (S : STRING);
FORWARD;

PROCEDURE TEXT_PROC (S : STRING);
FORWARD;

PROCEDURE TONEWPAGE_PROC;
FORWARD;

PROCEDURE TXBOT_PROC;
FORWARD;

FUNCTION UPSTR_FCTN (S : STRING) : STRING;
FORWARD;

FUNCTION USSTR_FCTN (S : STRING) : STRING;
FORWARD;

FUNCTION VERIFY_FCTN (S : STRING;
                      C : CHAR) : INTEGER;
FORWARD;

FUNCTION WIDTH_FCTN (S : STRING) : INTEGER;
FORWARD;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PUT THE PAGE NUMBER IN THE PAGE NUMBER       }
{    LINKED LIST OF AN INDEX ENTRY.               }
{-------------------------------------------------}

    PROCEDURE ADDINDEXNO_PROC;
    BEGIN
      IF PAGENO <> P@.IX_PGN@.IN_INT THEN
        BEGIN
          INQ@.IN_NXT := P@.IX_PGN;
          P@.IX_PGN := INQ;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CHECK THE CHARACTERS OF A STRING TO FIND     }
{    WHETHER THEY ARE ALL THE DIGITS OR NOT.      }
{-------------------------------------------------}

    FUNCTION ALLDIGITS_FCTN;
    BEGIN
      ALLDIGITS_FCTN := FALSE;
      IF LENGTH(S) > 0 THEN
        IF PARTDIGITS_FCTN(S,1,LENGTH(S)) THEN
          ALLDIGITS_FCTN := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{   USE LINE-BY-LINE METHOD TO ARRANGE THE        }
{   WORDS IN AN OUTPUT TEXT LINE.                 }
{-------------------------------------------------}

    PROCEDURE ARRANGEOUTSTR_PROC;
    VAR
            W : STRING;
            J, K, L : INTEGER;
            E, R : BOOLEAN;
    BEGIN
      S2 := '';
      J := 0;
      K := 0;
      E := FALSE;
      IF S1[1] = BLANK THEN
        BEGIN
          K := VERIFY_FCTN(S1,BLANK) - 1;
          S2 := S2 + SUBSTR(S1,1,K);
        END;
      GETWORD_PROC(S1,W,L,R);
      WHILE R AND (NOT E) DO
        IF I >= K + J + L + 1 THEN
          BEGIN
            IF J = 0 THEN
              BEGIN
                S2 := S2 + W;
                J := L;
              END
            ELSE
              BEGIN
                S2 := S2 + BLANK + W;
                J := J + L + 1;
              END;
            GETWORD_PROC(S1,W,L,R);
          END
        ELSE
          BEGIN
            E := TRUE;
            IF J = 0 THEN
              S2 := S2 + W
            ELSE
              S1 := W + BLANK + S1;
          END;
      IF R THEN
        BEGIN
          S2 := SPREAD_FCTN(S2,I);
          S1 := RMLB_FCTN(S1);
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    INSERT PARTICULAR NUMBER OF BLANKS INTO A    }
{    STRING FROM THE BACKWARD DIRECTION.          }
{-------------------------------------------------}

    FUNCTION BACKWARD_FCTN;
    BEGIN
      S := REVERSE_FCTN(S);
      S := FORWARD_FCTN(S,I);
      S := REVERSE_FCTN(S);
      BACKWARD_FCTN := S;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    MAKE THE STRING BE PRINTED TWICE.            }
{-------------------------------------------------}

    FUNCTION BDSTR_FCTN;
    VAR
            S1 : STRING;
            I : INTEGER;
    BEGIN
      S1 := '';
      FOR I := 1 TO LENGTH(S) DO
        IF (S[I] <> BLANK) AND (S[I] <> BS) THEN
          S1 := S1 + S[I] + BS + S[I]
        ELSE
          S1 := S1 + S[I];
      BDSTR_FCTN := S1;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    MAKE A BREAK IN THE PRINTED TEXT.            }
{-------------------------------------------------}

    PROCEDURE BREAK_PROC;
    BEGIN
      IF B THEN
        WHILE (LENGTH(WKSTR) > 0) DO
          BEGIN
            IF WIDTH_FCTN(WKSTR) <= TXVAL THEN
              BEGIN
                OUTSTR := WKSTR;
                WKSTR := '';
              END
            ELSE
              ARRANGEOUTSTR_PROC(WKSTR,OUTSTR,TXVAL);
            PRINTTEXT_PROC(OUTSTR);
          END;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    INITIATE A TITLE LINKED LIST.                }
{-------------------------------------------------}

    PROCEDURE BUILDTT_PROC;
    VAR
            Q, R : TT_P;
            I : INTEGER;
    BEGIN
      R := NIL;
      FOR I := 1 TO 30 DO
        BEGIN
          NEW(Q);
          Q@.TT_LFT := '';
          Q@.TT_MDL := '';
          Q@.TT_RIT := '';
          Q@.TT_NXT := R;
          R := Q;
        END;
      P := R;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PREPARE ONE OF THREE PARTS OF A TITLE.       }
{    CHANGE THE PAGE NUMBER SYMBOL OF THE TITLE   }
{    TO THE REAL PAGE NUMBER.                     }
{-------------------------------------------------}

    FUNCTION BUILDTTPART_FCTN;
    VAR
            I : INTEGER;
            S1, S2 : STRING;
            R : BOOLEAN;
    BEGIN
      S1 := '';
      WHILE (LENGTH(S) > 0) DO
        BEGIN
          I := INDEX(S,SYSPS);
          IF I = 0 THEN
            BEGIN
              S1 := S1 + S;
              S := '';
            END
          ELSE
            BEGIN
              S1 := S1 + SUBSTR(S,1,I-1);
              IF SYSPN THEN
                S1 := S1 + INTTOARB_FCTN(PAGENO)
              ELSE
                BEGIN
                  INTTOROM_PROC(PAGENO,S2,R);
                  IF R THEN
                    S1 := S1 + S2
                  ELSE
                    S1 := S1 + INTTOARB_FCTN(PAGENO);
                END;
              IF LENGTH(S) = I THEN
                S := ''
              ELSE
                S := SUBSTR(S,I+1,LENGTH(S)-I);
            END;
        END;
      BUILDTTPART_FCTN := S1;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS A PRINTED TEXT LINE THAT NEEDS TO    }
{    BE CENTERED.                                 }
{-------------------------------------------------}

    PROCEDURE CE_PROC;
    VAR
            J, K : INTEGER;
    BEGIN
      K := ((TXVAL - I) DIV 2) + 1;
      K := MAX_FCTN(K,1);
      FOR J := 1 TO (K - 1) DO
        S := BLANK + S;
      PRINTTEXT_PROC(S);
      MINUSONE_PROC(CEVAL,CEBOL);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE TEXT WHEN THE CONCATENATION      }
{    SWITCH IS ON.                                }
{-------------------------------------------------}

    PROCEDURE CO_PROC;
     BEGIN
      IF S[1] = BLANK THEN
        BREAK_PROC(TRUE);
      IF LENGTH(WKSTR) = 0 THEN
        WKSTR := S
      ELSE
        WKSTR := WKSTR + BLANK + S;
      WHILE (WIDTH_FCTN(WKSTR) >= TXVAL) DO
        BEGIN
          ARRANGEOUTSTR_PROC(WKSTR,OUTSTR,TXVAL);
          PRINTTEXT_PROC(OUTSTR);
        END;
    END;
```

```
{-------------------------------------------------------}
{  FUNCTION :                                           }
{    PROCESS SYSTEM CONTROL WORDS AND                   }
{    USER DEFINED MACROS.                               }
{-------------------------------------------------------}

    PROCEDURE COMMAND_PROC;
    VAR
            CMND : CMDTYPE;
            P0 : STRING;
            I : INTEGER;
    BEGIN
      I := LENGTH(S);
      IF I < 3 THEN
        ERROR := TRUE
      ELSE
        BEGIN
          S := LWSTR_FCTN(S);
          P0 := SUBSTR(S,2,2);
          CMND := CONVERSE_FCTN(P0);
          IF I > 3 THEN
            S := SUBSTR(S,4,I-3)
          ELSE
            S := '';

          IF CMND = 'ad' THEN
            BEGIN
              SETAEON_PROC(S,SYSADE,SYSADO,TRUE);
              SYSAD := SELECT_FCTN(PAGETYPE,SYSADE,SYSADO);
            END
          ELSE
          IF CMND = 'bd' THEN
            SET1NOO_PROC(S,BDVAL,BDBOL,FALSE)
          ELSE
          IF CMND = 'bi' THEN
            BEGIN
              SET1NOO_PROC(S,BDVAL,BDBOL,FALSE);
              SET1NOO_PROC(S,USVAL,USBOL,FALSE);
            END
          ELSE


        { SAME ROUTINE WILL CONTINUE ON NEXT PAGE }
```

```
  IF CMND = 'bm' THEN
    BEGIN
      SETAEON_PROC(S,SYSBME,SYSBMO,FALSE);
      IF NOT ERROR THEN
        BEGIN
          TXBOT_PROC;
          IF NOT ERROR THEN
            IF (SYSBME < (SYSFME + SYSFSE)) OR
               (SYSBMO < (SYSFMO + SYSFSO)) THEN
              ERROR := TRUE;
        END;
    END
  ELSE
  IF CMND = 'br' THEN
    BEGIN
      PARA0_PROC(S,ERROR);
      IF NOT ERROR THEN
        BREAK_PROC(TRUE);
    END
  ELSE
  IF CMND = 'bt' THEN
    SETAEONT_PROC(S,SYSBTE,SYSBTO,TRUE)
  ELSE
  IF CMND = 'ce' THEN
    BEGIN
      SETlNOO_PROC(S,CEVAL,CEBOL,TRUE);
      IF (NOT ERROR) AND CEBOL THEN
        RIBOL := FALSE;
    END
  ELSE
  IF CMND = 'cm' THEN
    BEGIN
    END
  ELSE
  IF CMND = 'co' THEN
    SETOO_PROC(S,COBOL)
  ELSE
  IF CMND = 'cp' THEN
    CP_PROC(S)
  ELSE
  IF CMND = 'dm' THEN
    DM_PROC(S)
  ELSE
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
IF CMND = 'ds' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
      BEGIN
        S := 'l';
        LS_PROC(S);
      END;
  END
ELSE
IF CMND = 'ea' THEN
  BEGIN
    SETN_PROC(S,SYSADE);
    SYSAD := SELECT_FCTN(PAGETYPE,SYSADE,SYSADO);
  END
ELSE
IF CMND = 'eb' THEN
  SETNT_PROC(S,SYSBTE,0)
ELSE
IF CMND = 'ep' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
      BEGIN
        S := 'even';
        PA_PROC(S);
      END;
  END
ELSE
IF CMND = 'et' THEN
  SETNT_PROC(S,SYSTTE,1)
ELSE
IF CMND = 'fb' THEN
  FB_PROC(S)
ELSE
IF CMND = 'fd' THEN
  FD_PROC(S)
ELSE
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
  IF CMND = 'fi' THEN
    BEGIN
      PARA0_PROC(S,ERROR);
      IF NOT ERROR THEN
        BEGIN
          S := 'on';
          FO_PROC(S);
        END;
    END
  ELSE
  IF CMND = 'fm' THEN
    BEGIN
      SETAEON_PROC(S,SYSFME,SYSFMO,FALSE);
      IF NOT ERROR THEN
        IF (SYSBME < (SYSFME + SYSFSE)) OR
           (SYSBMO < (SYSFMO + SYSFSO)) THEN
          ERROR := TRUE;
    END
  ELSE
  IF CMND = 'fo' THEN
    FO_PROC(S)
  ELSE
  IF CMND = 'fs' THEN
    BEGIN
      SETAEON_PROC(S,SYSFSE,SYSFSO,FALSE);
      IF NOT ERROR THEN
        IF (SYSBME < (SYSFME + SYSFSE)) OR
           (SYSBMO < (SYSFMO + SYSFSO)) OR
           (SYSFSE > 30) OR
           (SYSFSO > 30) THEN
          ERROR := TRUE;
    END
  ELSE
  IF CMND = 'hm' THEN
    BEGIN
      SETAEON_PROC(S,SYSHME,SYSHMO,FALSE);
      IF NOT ERROR THEN
        IF (SYSTME < (SYSHME + SYSHSE)) OR
           (SYSTMO < (SYSHMO + SYSHSO)) THEN
          ERROR := TRUE;
    END
  ELSE
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
IF CMND = 'hs' THEN
  BEGIN
    SETAEON_PROC(S,SYSHSE,SYSHSO,FALSE);
    IF NOT ERROR THEN
      IF (SYSTME < (SYSHME + SYSHSE)) OR
         (SYSTMO < (SYSHMO + SYSHSO)) OR
         (SYSHSE > 30) OR
         (SYSHSO > 30) THEN
        ERROR := TRUE;
  END
ELSE
IF CMND = 'in' THEN
  BEGIN
    SETN_PROC(S,SYSIND);
    IF NOT ERROR THEN
      LBVAL_PROC;
  END
ELSE
IF CMND = 'ir' THEN
  BEGIN
    SETN_PROC(S,SYSIR);
    IF NOT ERROR THEN
      LBVAL_PROC;
  END
ELSE
IF CMND = 'ix' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
      IXBOL := TRUE;
  END
ELSE
IF CMND = 'ju' THEN
  SETOO_PROC(S,JUBOL)
ELSE
IF CMND = 'll' THEN
  BEGIN
    SETN_PROC(S,SYSLL);
    IF NOT ERROR THEN
      LBVAL_PROC;
  END
ELSE
```

(SAME ROUTINE WILL CONTINUE ON NEXT PAGE)

```
  IF CMND = 'ls' THEN
    LS_PROC(S)
  ELSE
  IF CMND = 'nc' THEN
    BEGIN
      PARA0_PROC(S,ERROR);
      IF NOT ERROR THEN
        BEGIN
          S := 'off';
          SETOO_PROC(S,COBOL);
        END;
    END
  ELSE
  IF CMND = 'nf' THEN
    BEGIN
      PARA0_PROC(S,ERROR);
      IF NOT ERROR THEN
        BEGIN
          S := 'off';
          FO_PROC(S);
        END;
    END
  ELSE
  IF CMND = 'nj' THEN
    BEGIN
      PARA0_PROC(S,ERROR);
      IF NOT ERROR THEN
        BEGIN
          S := 'off';
          SETOO_PROC(S,JUBOL);
        END;
    END
  ELSE
  IF CMND = 'oa' THEN
    BEGIN
      SETN_PROC(S,SYSADO);
      SYSAD := SELECT_FCTN(PAGETYPE,SYSADE,SYSADO);
    END
  ELSE
  IF CMND = 'ob' THEN
    SETNT_PROC(S,SYSBTO,2)
  ELSE
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
IF CMND = 'op' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
       BEGIN
        S := 'odd';
        PA_PROC(S);
      END;
  END
ELSE
IF CMND = 'ot' THEN
  SETNT_PROC(S,SYSTTO,3)
ELSE
IF CMND = 'pa' THEN
  PA_PROC(S)
ELSE
IF CMND = 'pi' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
      PUTINDEX_PROC;
  END
ELSE
IF CMND = 'pl' THEN
  BEGIN
    SETN_PROC(S,SYSPL);
    IF NOT ERROR THEN
      TXBOT_PROC;
  END
ELSE
IF CMND = 'pn' THEN
  PN_PROC(S)
ELSE
IF CMND = 'ps' THEN
  PS_PROC(S)
ELSE
IF CMND = 'pt' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
      PUTTC_PROC;
  END
ELSE
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
IF CMND = 'ri' THEN
  BEGIN
    SETlNOO_PROC(S,RIVAL,RIBOL,TRUE);
    IF (NOT ERROR) AND RIBOL THEN
      CEBOL := FALSE;
  END
ELSE
IF CMND = 'ro' THEN
  RO_PROC(S)
ELSE
IF CMND = 'sk' THEN
  SK_PROC(S)
ELSE
IF CMND = 'sp' THEN
  SP_PROC(S)
ELSE
IF CMND = 'ss' THEN
  BEGIN
    PARA0_PROC(S,ERROR);
    IF NOT ERROR THEN
      BEGIN
        S := '0';
        LS_PROC(S);
      END;
  END
ELSE
IF CMND = 'tc' THEN
  SETlN_PROC(S,TCVAL,FALSE)
ELSE
IF CMND = 'ti' THEN
  BEGIN
    SETN_PROC(S,TIVAL);
    IF NOT ERROR THEN
      LBVAL_PROC;
  END
ELSE
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
      IF CMND = 'tm' THEN
        BEGIN
          SETAEON_PROC(S,SYSTME,SYSTMO,FALSE);
          IF NOT ERROR THEN
            BEGIN
              TXBOT_PROC;
              IF NOT ERROR THEN
                IF (SYSTME < (SYSHME + SYSHSE)) OR
                   (SYSTMO < (SYSHMO + SYSHSO)) THEN
                  ERROR := TRUE;
            END;
        END
      ELSE
      IF CMND = 'tt' THEN
        SETAEONT_PROC(S,SYSTTE,SYSTTO,FALSE)
      ELSE
      IF CMND = 'uc' THEN
        BEGIN
          SETlNOO_PROC(S,UPVAL,UPBOL,FALSE);
          SETlNOO_PROC(S,USVAL,USBOL,FALSE);
        END
      ELSE
      IF CMND = 'un' THEN
        BEGIN
          SETN_PROC(S,UNVAL);
          IF NOT ERROR THEN
            LBVAL_PROC;
        END
      ELSE
      IF CMND = 'up' THEN
        SETlNOO_PROC(S,UPVAL,UPBOL,FALSE)
      ELSE
      IF CMND = 'us' THEN
        SETlNOO_PROC(S,USVAL,USBOL,FALSE)
      ELSE
        OTHERWISE_PROC(S,CMND);
    END;
END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CONVERSE A CONTROL WORD FROM STRING TYPE     }
{    TO COMMAND TYPE.                             }
{-------------------------------------------------}

    FUNCTION CONVERSE_FCTN;
    VAR
            A : CMDTYPE;
            I : INTEGER;
    BEGIN
      FOR I := 1 TO 2 DO
        A[I] := S[I];
      CONVERSE_FCTN := A;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .cp SYSTEM CONTROL WORD.         }
{    .cp n                                        }
{-------------------------------------------------}

    PROCEDURE CP_PROC;
    VAR
            P1 : STRING;
            I : INTEGER;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (NOT ALLDIGITS_FCTN(P1)) THEN
          ERROR := TRUE
        ELSE
          BEGIN
            I := INTSTR_FCTN(P1);
            BREAK_PROC(TRUE);
            IF TXBOT <= (I + LINENO) THEN
              BEGIN
                TONEWPAGE_PROC;
                NEWPAGENO_PROC;
                PUTHF_PROC(TRUE);
              END;
          END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .dm SYSTEM CONTROL WORD.         }
{    .dm < name | off >                           }
{-------------------------------------------------}

    PROCEDURE DM_PROC;
     VAR
            P, Q : DM_P;
            DP, DE : TX_P;
            A : CMDTYPE;
            I : INTEGER;
            P1 : STRING;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF LENGTH(P1) <> 2 THEN
          ERROR := TRUE
        ELSE
          BEGIN
            A := CONVERSE_FCTN(P1);
            ERROR := ISCW_FCTN(A);
            IF NOT ERROR THEN
              BEGIN
                P := SYSDM;
                I := 0;
                SEARCHMACRO_PROC(A,P,I);
                IF I = 3 THEN
                  ERROR := TRUE
                ELSE
                  BEGIN
                    NEW(Q);
                    Q@.DM_LFT := NIL;
                    Q@.DM_NAM := A;
                    Q@.DM_RIT := NIL;
                    Q@.DM_FST := NIL;
                    IF I = 0 THEN
                      SYSDM := Q
                    ELSE
                      IF I = 1 THEN
                        P@.DM_LFT := Q
                      ELSE
                        P@.DM_RIT := Q;
                    DP := NIL;
                    LISTLINK_PROC(DP,DE,'dm',I);
                    IF NOT ERROR THEN
                      Q@.DM_FST := DP;
                  END;
              END;
          END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT THE TEXT REMAINED IN WORKING DATA AREA }
{    WHEN THE INPUT FILE IS END.                  }
{-------------------------------------------------}

    PROCEDURE END_PROC;
    VAR
            S : STRING;
    BEGIN
      IF ERROR THEN
        BEGIN
          S := 'ERROR IN INPUT CARD ' +
               INTTOARB_FCTN(CARDNO) + CRLF;
          WRITE(OUTF,OUTS);
        END
      ELSE
        BEGIN
          BREAK_PROC(TRUE);
          TONEWPAGE_PROC;
          WRITE(OUTF,FF);
        END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .fb SYSTEM CONTROL WORD.         }
{    .fb < on | off | put >                       }
{-------------------------------------------------}

    PROCEDURE FB_PROC;
    VAR
            P1 : STRING;
            I : INTEGER;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (LENGTH(P1) = 1) AND (P1 = NULL) THEN
          ERROR := TRUE
        ELSE
          IF (LENGTH(P1) = 2) AND (P1 = 'on') THEN
            LISTLINK_PROC(SYSFBH,SYSFBE,'fb',I)
          ELSE
            IF (LENGTH(P1) = 3) AND (P1 = 'put') THEN
              PRINTFB_PROC
            ELSE
              ERROR := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .fd SYSTEM CONTROL WORD.         }
{    .fd < on | off >                             }
{-------------------------------------------------}

    PROCEDURE FD_PROC;
    VAR
            P1, OUTS : STRING;
            FDP, FDD : TX_P;
            J : INTEGER;
            E : BOOLEAN;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (LENGTH(P1) <> 2) OR
           ((LENGTH(P1) = 2) AND (P1 <> 'on')) THEN
          ERROR := TRUE
        ELSE
          BEGIN
            BREAK_PROC(TRUE);
            LISTLINK(SYSFDH,SYSFDE,'fd',J);

            { IF NECESSARY, EJECT TO }
            { THE TOP OF NEXT PAGE   }
            IF J + LINENO > TXBOT THEN
              BEGIN
                TONEWPAGE_PROC;
                NEWPAGENO_PROC;
                PUTHF_PROC(TRUE);
              END;

            { PRINT THE BLOCK OF DATA }
            FDP := SYSFDH;
            WHILE (FDP <> NIL) DO
              BEGIN
                PRINTFOOTHEAD_PROC;
                OUTS := MAKESTRING_FCTN(SYSAD,BLANK) +
                        FDP@.TX_TXT + CRLF;
                WRITE(OUTF,OUTS);
                LINENO := LINENO + 1;
                FDD := FDP;
                FDP := FDP@.TX_NXT;
                DISPOSE(FDD);
              END;
            PRINTLINES_PROC(SYSLS);

            SYSFDH := NIL;
            SYSFDE := NIL;
          END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .fo SYSTEM CONTROL WORD.         }
{    .fo < on | off >                             }
{-------------------------------------------------}

    PROCEDURE FO_PROC;
    BEGIN
      SETOO_PROC(S,COBOL);
      SETOO_PROC(S,JUBOL);
     END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    INSERT PARTICULAR NUMBER OF BLANKS INTO      }
{    A STRING FROM THE FORWARD DIRECTION.         }
{-------------------------------------------------}

    FUNCTION FORWARD_FCTN;
    VAR
            S1 : STRING;
            J, K, L : INTEGER;
    BEGIN
      J := 1;
      WHILE (I > 0) DO
        BEGIN
          S1 := SUBSTR(S,J,LENGTH(S)-J+1);
          K := INDEX(S1,BLANK);
          IF K > 0 THEN
            BEGIN
              K := J + K - 1;
              S := SUBSTR(S,1,K) + BLANK +
                   SUBSTR(S,K+1,LENGTH(S)-K);
              S1 := SUBSTR(S,K,LENGTH(S)-K+1);
              L := VERIFY_FCTN(S1,BLANK);
              J := K + L - 1;
            END
          ELSE
            J := 1;
          IF J > 1 THEN
            I := I - 1;
        END;
      FORWARD_FCTN := S;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN FIRST WORD FROM A STRING. IF NO WORD  }
{    IS OBTAINED, THEN A FLAG IS SET TO FALSE.    }
{-------------------------------------------------}

    PROCEDURE GETPARA_PROC;
    VAR
            I, J : INTEGER;
    BEGIN
      R := FALSE;
      S := RMLB_FCTN(S);
      I := LENGTH(S);
      IF I > 0 THEN
        BEGIN
          R := TRUE;
          J := INDEX(S,BLANK);
          IF J = 0 THEN
            BEGIN
              P := S;
              S := '';
            END
          ELSE
            BEGIN
              P := SUBSTR(S,1,J-1);
              S := SUBSTR(S,J+1,I-J);
            END;
        END;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN FIRST WORD FROM A STRING. ITS LENGTH  }
{    IS ALSO RETURNED.                            }
{-------------------------------------------------}

    PROCEDURE GETWORD_PROC;
    BEGIN
      GETPARA_PROC(S,W,R);
      IF R THEN
        L := WIDTH_FCTN(W);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    OPEN TEXT INPUT FILE AND OUTPUT FILE.        }
{    IT ALSO PRINTS THE STARTING MESSAGE.         }
{-------------------------------------------------}

    PROCEDURE INITFILE_PROC;
    VAR
            S : STRING;
            E : BOOLEAN;
    BEGIN
      WRITE('ENTER INPUT FILE NAME : ');
      READLN(S);    { INPUT DATA FILE NAME }
      OPEN(INF,S,HISTORY:=OLD);
      RESET(INF);
      WRITELN('IF ON-LINE PRINTOUTS, THEN PRESS RETURN.');
      WRITE('OTHERWISE, ENTER OUTPUT FILE NAME : ');
      READLN(S);    { OUTPUT DATA FILE NAME }
      S := RMLB_FCTN(S);
      S := RMTB_FCTN(S);
      IF LENGTH(S) = 0 THEN
        S := 'SYS$OUTPUT:';
      OPEN(OUTF,S,CARRIAGE_CONTROL:=NONE);
      REWRITE(OUTF);

      IF (LENGTH(S) = 11) AND
         (S = 'SYS$OUTPUT:') THEN
        BEGIN
          E := FALSE;
          WHILE (NOT E) DO
            BEGIN
              WRITE('PAUSE OR NOT ? (Y/N)  ');
              READLN(S);
              IF LENGTH(S) = 1 THEN
                IF (S = 'y') OR (S = 'Y') THEN
                  BEGIN
                    E := TRUE;
                    PAUSE := TRUE;
                  END
                ELSE
                  IF (S = 'n') OR (S = 'N') THEN
                    BEGIN
                      E := TRUE;
                      PAUSE := FALSE;
                    END;
            END;
          WRITELN('LOAD PAPER AND PRESS RETURN');
          READLN(S);
          WRITE(OUTF,CRLF);
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    INITIATE VARIABLES AND TITLES, OPEN FILES,   }
{    AND PRINT THE STARTING MESSAGE.              }
{-------------------------------------------------}

    PROCEDURE INITIAL_PROC;
    BEGIN
      INITVAL_PROC;
      INITTT_PROC;
      INITFILE_PROC;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    INITIATE TITLES.                             }
{-------------------------------------------------}

    PROCEDURE INITTT_PROC;
    BEGIN
      BUILDTT_PROC(SYSTTE);
      BUILDTT_PROC(SYSTTO);
      BUILDTT_PROC(SYSBTE);
      BUILDTT_PROC(SYSBTO);
    END;
```

```
{-------------------------------------------------------}
{  FUNCTION :                                           }
{    INITIATE VARIABLES.                                }
{-------------------------------------------------------}

    PROCEDURE INITVAL_PROC;
    BEGIN
      INSTR := '';
      WKSTR := '';
      OUTSTR := '';
      CRLF := CR + LF;

      START_FLAG := TRUE;
      ERROR := FALSE;
      CARDNO := 0;

      SYSDM := NIL;
      SYSIX := NIL;
      SYSTCH := NIL;
      SYSTCE := NIL;
      SYSTTE := NIL;
      SYSTTO := NIL;
      SYSBTE := NIL;
      SYSBTO := NIL;
      SYSFDH := NIL;
      SYSFDE := NIL;
      SYSFBH := NIL;
      SYSFBE := NIL;

      SYSLS := 0;
      SYSTME := 6;
      SYSTMO := 6;
      SYSHSE := 1;
      SYSHSO := 1;
      SYSHME := 1;
      SYSHMO := 1;
      SYSBME := 6;
      SYSBMO := 6;
      SYSFME := 1;
      SYSFMO := 1;
      SYSFSE := 1;
      SYSFSO := 1;
      SYSADE := 0;
      SYSADO := 0;
      SYSIND := 0;
      SYSIR := 0;
      SYSPL := 66;
      SYSLL := 60;
```

{ SAME ROUTINE WILL CONTINUE ON NEXT PAGE }

```
    SKVAL := 0;
    SPVAL := 0;
    LBVAL := 0;
    UNVAL := 0;
    TIVAL := 0;
    TXVAL := 60;
    CEVAL := 0;
    RIVAL := 0;
    LINENO := 0;
    PAGENO := 1;
    USVAL := 0;
    UPVAL := 0;
    BDVAL := 0;
    TCVAL := 0;
    TXBOTE := 54;
    TXBOTO := 54;
    TXBOT := 54;
    SYSAD := 0;

    SYSPS := '%';
    INDEXCHAR := ' ';
    SYSPN := TRUE;
    ROMBG := FALSE;
    CEBOL := FALSE;
    RIBOL := FALSE;
    USBOL := FALSE;
    UPBOL := FALSE;
    BDBOL := FALSE;
    COBOL := TRUE;
    JUBOL := TRUE;
    IXBOL := FALSE;
    PAGETYPE := FALSE;
    DIRECTION := TRUE;
END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    WHETHER THIS CHARACTER IS IN A PARTICULAR    }
{    RANGE.                                       }
{-------------------------------------------------}

    FUNCTION INRANGE_FCTN;
    BEGIN
      INRANGE_FCTN := FALSE;
      IF (ORD(C) >= ORD(R1)) AND
         (ORD(C) <= ORD(R2)) THEN
        INRANGE_FCTN := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CONVERSE A STRING INTO AN INTEGER.           }
{-------------------------------------------------}

    FUNCTION INTSTR_FCTN;
    VAR
            I, J : INTEGER;
    BEGIN
      J := 0;
      FOR I := 1 TO LENGTH(S) DO
        J := J * 10 + (ORD(S[I])-48);
      INTSTR_FCTN := J;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CONVERSE AN INTEGER TO AN ARABIC NUMERAL     }
{    STRING.                                      }
{-------------------------------------------------}

    FUNCTION INTTOARB_FCTN;
    VAR
            S : STRING;
            J : INTEGER;
    BEGIN
      IF I = 0 THEN
        INTTOARB_FCTN := '0'
      ELSE
        BEGIN
          S := '';
          WHILE (I > 0) DO
            BEGIN
              J := I MOD 10;
              I := I DIV 10;
              S := CHR(J+48) + S;
            END;
          INTTOARB_FCTN := S;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CONVERSE AN INTEGER THAT IS FROM 1 TO 999    }
{    TO A ROMAN NUMERAL STRING.                   }
{-------------------------------------------------}

    PROCEDURE INTTOROM_PROC;
    VAR
           J : INTEGER;
    BEGIN
      E := FALSE;
      IF (I < 1000) AND (I > 0) THEN
        BEGIN
          E := TRUE;
          S := '';
          J := I MOD 10;
          I := I DIV 10;
          CASE J OF
            1 : S := 'i';
            2 : S := 'ii';
            3 : S := 'iii';
            4 : S := 'iv';
            5 : S := 'v';
            6 : S := 'vi';
            7 : S := 'vii';
            8 : S := 'viii';
            9 : S := 'ix';
          END;
          IF I > 0 THEN
            BEGIN
              J := I MOD 10;
              I := I DIV 10;
              CASE J OF
                1 : S := 'x' + S;
                2 : S := 'xx' + S;
                3 : S := 'xxx' + S;
                4 : S := 'xl' + S;
                5 : S := 'l' + S;
                6 : S := 'lx' + S;
                7 : S := 'lxx' + S;
                8 : S := 'lxxx' + S;
                9 : S := 'xc' + S;
              END;


        { SAME ROUTINE WILL CONTINUE ON NEXT PAGE }
```

```
          IF I > 0 THEN
            CASE I OF
              1 : S := 'c' + S;
              2 : S := 'cc' + S;
              3 : S := 'ccc' + S;
              4 : S := 'cd' + S;
              5 : S := 'd' + S;
              6 : S := 'dc' + S;
              7 : S := 'dcc' + S;
              8 : S := 'dccc' + S;
              9 : S := 'cm' + S;
            END;
        END;
      IF ROMBG THEN
        S := UPSTR_FCTN(S);
    END;
END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A STRING IS 'all', 'odd', OR 'even'.    }
{-------------------------------------------------}

    FUNCTION ISAEO_FCTN;
    BEGIN
      ISAEO_FCTN := 0;              { 0 -- ERROR }
      IF LENGTH(S) = 3 THEN
        BEGIN
          IF S = 'all' THEN
            ISAEO_FCTN := 1         { 1 -- ALL }
          ELSE
            IF S = 'odd' THEN
              ISAEO_FCTN := 2;      { 2 -- ODD}
        END
      ELSE
        IF (LENGTH(S) = 4) AND (S = 'even') THEN
          ISAEO_FCTN := 3;          { 3 -- EVEN }
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A CHARACTER IS ALPHABETIC OR NOT.       }
{-------------------------------------------------}

    FUNCTION ISALPHA_FCTN;
    BEGIN
      ISALPHA_FCTN := FALSE;
      IF INRANGE_FCTN(C,'a','z') OR
         INRANGE_FCTN(C,'A','Z') THEN
        ISALPHA_FCTN := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A STRING IS SYSTEM CONTROL WORDS OR NOT. }
{-------------------------------------------------}

    FUNCTION ISCW_FCTN;
    BEGIN
      ISCW_FCTN := FALSE;
      IF (A = 'ad') OR (A = 'bd') OR (A = 'bi') OR
         (A = 'bm') OR (A = 'br') OR (A = 'bt') OR
         (A = 'ce') OR (A = 'cm') OR (A = 'co') OR
         (A = 'cp') OR (A = 'dm') OR (A = 'ds') OR
         (A = 'ea') OR (A = 'eb') OR (A = 'ep') OR
         (A = 'et') OR (A = 'fb') OR (A = 'fd') OR
         (A = 'fi') OR (A = 'fm') OR (A = 'fo') OR
         (A = 'fs') OR (A = 'hm') OR (A = 'hs') OR
         (A = 'in') OR (A = 'ir') OR (A = 'ix') OR
         (A = 'ju') OR (A = 'll') OR (A = 'ls') OR
         (A = 'nc') OR (A = 'nf') OR (A = 'nj') OR
         (A = 'oa') OR (A = 'ob') OR (A = 'op') OR
         (A = 'ot') OR (A = 'pa') OR (A = 'pi') OR
         (A = 'pl') OR (A = 'pn') OR (A = 'ps') OR
         (A = 'pt') OR (A = 'ri') OR (A = 'ro') OR
         (A = 'sk') OR (A = 'sp') OR (A = 'ss') OR
         (A = 'tc') OR (A = 'ti') OR (A = 'tm') OR
         (A = 'tt') OR (A = 'uc') OR (A = 'un') OR
         (A = 'up') OR (A = 'us') THEN
        ISCW_FCTN := TRUE;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A CHARACTER IS A DIGIT OR NOT.          }
{-------------------------------------------------}

    FUNCTION ISDIGIT_FCTN;
    BEGIN
      ISDIGIT_FCTN := FALSE;
      IF INRANGE_FCTN(C,'0','9') THEN
        ISDIGIT_FCTN := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST AN INTEGER IS EVEN OR ODD NUMBER.       }
{-------------------------------------------------}

    FUNCTION ISEVEN_FCTN;
    VAR
            J : INTEGER;
    BEGIN
      ISEVEN_FCTN := FALSE;    { FALSE -- ODD }
      J := I MOD 2;
      IF J = 0 THEN
        ISEVEN_FCTN := TRUE;    { TRUE -- EVEN }
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A CHARACTER IS LOWERCASE LETTER OR NOT. }
{-------------------------------------------------}

    FUNCTION ISLOWER_FCTN;
    BEGIN
      ISLOWER_FCTN := FALSE;
      IF INRANGE_FCTN(C,'a','z') THEN
        ISLOWER_FCTN := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A STRING IS NUMBER-LOOK OR NOT.         }
{    NUMBERS INCLUDE UNSIGNED NUMBER, POSITIVE    }
{    NUMBER, AND NEGATIVE NUMBER.                 }
{-------------------------------------------------}

    FUNCTION ISNUMBER_FCTN;
    VAR
            C : CHAR;
    BEGIN
      ISNUMBER_FCTN := 0;    { 0 -- NOT NUMBER }
      IF ALLDIGITS_FCTN(S) THEN
        ISNUMBER_FCTN := 1   { 1 -- UNSIGNED }
      ELSE                   {      NUMBER   }
        IF LENGTH(S) >= 2 THEN
          BEGIN
            C := S[1];
            S := SUBSTR(S,2,LENGTH(S)-1);
            IF ALLDIGITS_FCTN(S) THEN
              IF C = PLUS THEN
                ISNUMBER_FCTN := 2  { 2 -- POSITIVE }
              ELSE                  {      NUMBER   }
                IF C = MINUS THEN
                  ISNUMBER_FCTN := 3; { 3 -- NEGATIVE }
          END;                      {      NUMBER   }
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{  TEST A STRING IS 'on' OR 'off'.                }
{-------------------------------------------------}

    FUNCTION ISOO_FCTN;
    BEGIN
      ISOO_FCTN := 0;          { 0 -- ERROR }
      IF (LENGTH(S) = 2) AND
         (S = 'on') THEN
        ISOO_FCTN := 1         { 1 -- on }
      ELSE
        IF (LENGTH(S) = 3) AND
           (S = 'off') THEN
          ISOO_FCTN := 2;      { 2 -- off }
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A CHARACTER IS A PUNCTUATION OR NOT.    }
{-------------------------------------------------}

    FUNCTION ISPUNCT_FCTN;
    BEGIN
      ISPUNCT_FCTN := FALSE;
      IF INRANGE_FCTN(C,CHR(33),CHR(47)) OR
         INRANGE_FCTN(C,CHR(58),CHR(64)) OR
         INRANGE_FCTN(C,CHR(91),CHR(96)) OR
         INRANGE_FCTN(C,CHR(123),CHR(126)) THEN
        ISPUNCT_FCTN := TRUE;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A CHARACTER IS AN UPPERCASE LETTER      }
{    OR NOT.                                      }
{-------------------------------------------------}

    FUNCTION ISUPPER_FCTN;
    BEGIN
      ISUPPER_FCTN := FALSE;
      IF INRANGE_FCTN(C,'A','Z') THEN
        ISUPPER_FCTN := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A STRING LOOKS LIKE .xx off OR NOT.     }
{    THE VALUE OF xx IS PASSED FROM THE CALLING   }
{    ROUTINE.                                     }
{-------------------------------------------------}

    FUNCTION ISXXOFF_FCTN;
    VAR
            S1 : STRING;
    BEGIN
      ISXXOFF_FCTN := 0;    { 0 -- TEXT DATA }
      IF (S[1] = PERIOD) THEN
        BEGIN
          ISXXOFF_FCTN := 2;   { 2 -- CTRL WORD }
          IF LENGTH(S) >= 3 THEN
            BEGIN
              S1 := SUBSTR(S,2,2);
              CMND := CONVERSE_FCTN(S1);
              IF CMND = XX THEN
                BEGIN
                  S := SUBSTR(S,4,LENGTH(S)-3);
                  S := RMLB_FCTN(S);
                  S := RMTB_FCTN(S);
                  IF (LENGTH(S) = 3) AND
                     (S = 'off') THEN
                    ISXXOFF_FCTN := 1; { 1 -- .xx off }
                END;
            END;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS INPUT TEXT WHEN THE INDEX FLAG IS ON. }
{    PUT THE INPUT TEXT INTO THE INDEX LINKED LIST.}
{-------------------------------------------------}

    PROCEDURE IX_PROC;
    VAR
            T : TEXTTYPE;
            IP, IQ : IX_P;
            INQ : IN_P;
            I : INTEGER;
    BEGIN
      T := STRTOTEXT_FCTN(LWSTR_FCTN(S));
      NEW(INQ);
      INQ@.IN_INT := PAGENO;
      INQ@.IN_NXT := NIL;
      IP := SYSIX;
      I := 0;
      SEARCHINDEX_PROC(T,IP,I);
      IF I = 3 THEN    { 3 -- INDEX ENTRY ALREADY EXISTS }
        ADDINDEXNO_PROC(IP,INQ)
      ELSE
        BEGIN
          NEW(IQ);
          IQ@.IX_LFT := NIL;
          IQ@.IX_TXT := T;
          IQ@.IX_STR := S;
          IQ@.IX_PGN := INQ;
          IQ@.IX_RIT := NIL;
          IF I = 0 THEN    { 0 -- EMPTY TREE }
            SYSIX := IQ
          ELSE
            IF I = 1 THEN { 1 -- LEFT SUBTREE }
              IP@.IX_LFT := IQ
            ELSE           { 2 -- RIGHT SUBTREE }
              IP@.IX_RIT := IQ;
        END;
      IXBOL := FALSE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    COMPUTE THE NUMBER OF LEADING BLANKS AND THE }
{    LENGTH OF TEXT AREA OF A PRINTED TEXT LINE.  }
{-------------------------------------------------}

    PROCEDURE LBVAL_PROC;
    BEGIN
      LBVAL := SYSIND - UNVAL + TIVAL;
      TXVAL := SYSLL - LBVAL - SYSIR;
      IF (LBVAL < 0) OR (TXVAL <= 0) THEN
        ERROR := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    LINK A TEXT NODE TO A TEXT LINKED LIST.      }
{-------------------------------------------------}

    PROCEDURE LINKTEXT_PROC;
    VAR
            QP : TX_P;
    BEGIN
      NEW(QP);
      QP@.TX_TXT := S;
      QP@.TX_NXT := NIL;
      IF HP = NIL THEN
        HP := QP
      ELSE
        EP@.TX_NXT := QP;
      EP := QP;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    READ TEXT FROM THE INPUT FILE, LINK THEM INTO }
{    A TEXT LINKED LIST UNTIL A SPECIAL INPUT DATA }
{    IS ENCOUNTERED.                              }
{-------------------------------------------------}

    PROCEDURE LISTLINK_PROC;
    VAR
            S : STRING;
            CMND : CMDTYPE;
            I : INTEGER;
            E : BOOLEAN;
    BEGIN
      J := 0;
      E := FALSE;
      IF NOT EOF(INF) THEN
        REPEAT
          READLN(INF,S);
          CARDNO := CARDNO + 1
          S := RMTB_FCTN(S);
          IF LENGTH(S) = 0 THEN
            LINKTEXT_PROC(DH,DE,'')
          ELSE
            BEGIN
              I := ISXXOFF_FCTN(XX,S,CMND);
              IF I = 1 THEN
                E := TRUE
              ELSE
                IF (((XX = 'dm') OR (XX = 'fb')) AND
                    ((CMND = 'dm') OR (CMND = 'fb') OR
                                      (CMND = 'fd'))) OR
                    ((XX = 'fd') AND (I = 2)) THEN
                  ERROR := TRUE
                ELSE
                  BEGIN
                    LINKTEXT_PROC(DH,DE,S);
                    J := J + 1;
                  END;
            END;
        UNTIL EOF(INF) OR ERROR OR E;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .ls SYSTEM CONTROL WORD.         }
{    .ls n                                        }
{-------------------------------------------------}

    PROCEDURE LS_PROC;
    BEGIN
      SETN_PROC(S,SYSLS);
      IF NOT ERROR THEN
        IF (SYSLS >= TXBOTE) OR
           (SYSLS >= TXBOTO) THEN
          ERROR := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CONVERSE A STRING INTO LOWERCASE LETTER.     }
{-------------------------------------------------}

    FUNCTION LWSTR_FCTN;
    VAR
            I : INTEGER;
    BEGIN
      FOR I := 1 TO LENGTH(S) DO
        IF ISUPPER_FCTN(S[I]) THEN
          S[I] := CHR(ORD(S[I])+32);
      LWSTR_FCTN := S;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS USER DEFINED MACRO CONTROL WORD.     }
{-------------------------------------------------}

    PROCEDURE MACRO_PROC;
    VAR
            DP : TX_P;
            S : STRING;
    BEGIN
      DP := P@.DM_FST;
      WHILE (DP <> NIL) AND (NOT ERROR) DO
        BEGIN
          S := DP@.TX_TXT;
          PROCESS_PROC(S);
          IF NOT ERROR THEN
            DP := DP@.TX_NXT;
        END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    MAKE A STRING WITH SAME CHARACTER AND        }
{    PARTICULAR LENGTH.                           }
{-------------------------------------------------}

    FUNCTION MAKESTRING_FCTN;
    VAR
            S : STRING;
            J : INTEGER;
    BEGIN
      S := '';
      FOR J := 1 TO I DO
        S := S + C;
      MAKESTRING_FCTN := S;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN THE MAXIMUM NUMBER FROM TWO INTEGERS. }
{-------------------------------------------------}

    FUNCTION MAX_FCTN;
    BEGIN
      MAX_FCTN := I;
      IF I < J THEN
        MAX_FCTN := J;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN THE MINUMUM NUMBER FROM TWO INTEGERS. }
{-------------------------------------------------}

    FUNCTION MIN_FCTN;
    BEGIN
      MIN_FCTN := I;
      IF I > J THEN
        MIN_FCTN := J;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    IF AN VARIABLE IS NOT -1, THEN MINUS 1 FROM  }
{    IT .                                         }
{-------------------------------------------------}

    PROCEDURE MINUSONE_PROC;
    BEGIN
      IF I <> -1 THEN
        BEGIN
          I := I - 1;
          IF I = 0 THEN
            E := FALSE;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    IF THE PAGE NUMBER IS UPDATED, THEN UPDATED  }
{    THE ADJUST BLANKS AND TEXT LINES NUMBERS.    }
{-------------------------------------------------}

    PROCEDURE NEWPAGENO_PROC;
    BEGIN
      PAGETYPE := ISEVEN_FCTN(PAGENO);
      SYSAD := SELECT_FCTN(PAGETYPE,SYSADE,SYSADO);
      TXBOT := SELECT_FCTN(PAGETYPE,TXBOTE,TXBOTO);
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    IF THE INPUT CONTROL WORD IS NOT A SYSTEM    }
{    CONTROL WORD, THEN SEARCH THE USER DEFINED   }
{    MACRO CONTROL WORDS.                         }
{-------------------------------------------------}

    PROCEDURE OTHERWISE_PROC;
    VAR
            P : DM_P;
            I : INTEGER;
    BEGIN
      PARA0_PROC(S,ERROR);
      IF NOT ERROR THEN
        BEGIN
          P := SYSDM;
          I := 0;
          SEARCHMACRO_PROC(A,P,I);
          IF I <> 3 THEN { 0,1,2 -- DOES NOT EXIST }
            ERROR := TRUE
          ELSE           { 3 -- EXIST }
            MACRO_PROC(P);
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .pa SYSTEM CONTROL WORD.         }
{    .pa < %+1 | n | +n | -n | even | odd>        }
{-------------------------------------------------}

    PROCEDURE PA_PROC;
    VAR
            P1 : STRING;
            I : INTEGER;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        BEGIN
          IF (LENGTH(P1) = 1) AND (P1 = NULL) THEN
            P1 := '+0';
          I := ISAEO_FCTN(P1);
          IF (ISNUMBER_FCTN(P1) = 0) AND
             (I <> 2) AND
             (I <> 3) THEN
            ERROR := TRUE
          ELSE
            BEGIN
              BREAK_PROC(TRUE);
              TONEWPAGE_PROC;

              { .pa < n | +n | -n > }
              IF ISNUMBER_FCTN(P1) <> 0 THEN
                BEGIN
                  PAGENO := SETNUM_FCTN(PAGENO,P1);
                  NEWPAGENO_PROC;
                END
              ELSE
                BEGIN
                  NEWPAGENO_PROC;

                  { .pa < even | odd> }
                  IF ((I = 2) AND PAGETYPE) OR
                     ((I = 3) AND (NOT PAGETYPE) THEN
                    BEGIN
                      PRINTBLANKPAGE_PROC;
                      NEWPAGENO_PROC;
                    END;
                END;
              PUTHF_PROC(TRUE);
            END;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST A STRING IS AN EMPTY STRING OR NOT.     }
{-------------------------------------------------}

    PROCEDURE PARA0_PROC;
    BEGIN
      S := RMLB_FCTN(S);
      IF LENGTH(S) > 0 THEN
        E := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN THE FIRST PARAMETER FROM A STRING.    }
{    IF THERE IS NOT ONLY ONE PARAMETER, THEN     }
{    ERROR FLAG IS SET TO TRUE.                   }
{-------------------------------------------------}

    PROCEDURE PARA1_PROC;
    VAR
            R : BOOLEAN;
    BEGIN
      GETPARA_PROC(S,P1,R);
      IF NOT R THEN
        P1 := NULL
      ELSE
        IF LENGTH(S) > 0 THEN
          ERROR := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN THE FIRST TWO PARAMETERS FROM A STRING.}
{    IF THERE ARE NOT ONLY TWO PARAMETERS, THEN   }
{    ERROR FLAG IS SET TO TRUE.                   }
{-------------------------------------------------}

    PROCEDURE PARA2_PROC;
    VAR
            R : BOOLEAN;
    BEGIN
      GETPARA_PROC(S,P1,R);
      IF NOT R THEN
        E := TRUE
      ELSE
        BEGIN
          GETPARA_PROC(S,P2,R);
          IF NOT R THEN
            E := TRUE
          ELSE
            IF LENGTH(S) > 0 THEN
              E := TRUE;
        END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN THE FIRST PARAMETER FROM A STRING,    }
{    THEN TREAT REMAINING STRING AS A PARAMETER.  }
{    IF THERE ARE ONLY ONE OR NO PARAMETER, THEN  }
{    ERROR FLAG IS SET TO TRUE.                   }
{-------------------------------------------------}

    PROCEDURE PARA2NT_PROC;
    VAR
            R : BOOLEAN;
    BEGIN
      GETPARA_PROC(S,P1,R);
      IF NOT R THEN
        E := TRUE
      ELSE
        BEGIN
          S := RMLB_FCTN(S);
          IF LENGTH(S) = 0 THEN
            E := TRUE
          ELSE
            P2 := S;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    OBTAIN THE FIRST TWO PARAMETERS FROM A       }
{    STRING, AND THE REMAINING STRING ARE TREATED }
{    AS A PARAMETER. IF THERE ARE NOT ONLY THREE  }
{    PARAMETERS, THEN ERROR FLAG IS SET TO TRUE.  }
{-------------------------------------------------}

    PROCEDURE PARA3_PROC;
    VAR
            R : BOOLEAN;
    BEGIN
      GETPARA_PROC(S,P1,R);
      IF NOT R THEN
        E := TRUE
      ELSE
        PARA2NT_PROC(S,P2,P3,E);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    TEST THE CHARACTERS BETWEEN A PARTICULAR     }
{    RANGE ARE ALL DIGITS OR NOT.                 }
{-------------------------------------------------}

    FUNCTION PARTDIGITS_FCTN;
    VAR
            K, L : INTEGER;
            E : BOOLEAN;
    BEGIN
      PARTDIGITS_FCTN := FALSE;
      K := I + J - 1;
      IF LENGTH(S) >= K THEN
        BEGIN
          L := I;
          E := FALSE;
          WHILE (L <= K) AND (NOT E) DO
            IF ISDIGIT_FCTN(S[I]) THEN
              L := L + 1
            ELSE
              E := TRUE;
          IF NOT E THEN
            PARTDIGITS_FCTN := TRUE;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .pn SYSTEM CONTROL WORD.         }
{    .pn < arabic | roman >                       }
{-------------------------------------------------}

    PROCEDURE PN_PROC;
    VAR
            P1 : STRING;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (LENGTH(P1) = 6) AND (P1 = 'arabic') THEN
          SYSPN := TRUE     { TRUE -- ARABIC }
        ELSE
          IF (LENGTH(P1) = 5) AND (P1 = 'roman') THEN
            SYSPN := FALSE    { FALSE -- ROMAN }
          ELSE
            ERROR := TRUE;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    WHEN PRINTING THE INDEX, PREPARE THE PAGE    }
{    NUMBER OF AN INDEX ENTRY.                    }
{-------------------------------------------------}

    PROCEDURE PREPAREINDEXNO_PROC;
    BEGIN
      S := '';
      WHILE (INP <> NIL) DO
        BEGIN
          IF LENGTH(S) = 0 THEN
            S := INTTOARB_FCTN(INP@.IN_INT)
          ELSE
            S := INTTOARB_FCTN(INP@.IN_INT) + ',' + S;
          INP := INP@.IN_NXT;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT A BLANK PAGE WITH TITLES.              }
{-------------------------------------------------}

    PROCEDURE PRINTBLANKPAGE_PROC;
    VAR
            OUTS : STRING;
    BEGIN
      PUTHF_PROC(TRUE);
      OUTS := MAKESTRING_FCTN(TXBOT,LF) + CR;
      WRITE(OUTF,OUTS);
      PUTHF_PROC(FALSE);
      PAGENO := PAGENO + 1;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .fb put SYSTEM CONTROL WORD.     }
{    PRINT THE CONTENTS OF THE FLOATING BLOCK.    }
{-------------------------------------------------}

    PROCEDURE PRINTFB_PROC;
    VAR
            DP, DQ : TX_P;
            S : STRING;
    BEGIN
      BREAK_PROC(TRUE);
      DP := SYSFBH;
      WHILE (DP <> NIL) DO
        BEGIN
          S := DP@.TX_TXT;
          PROCESS_PROC(S);
          DQ := DP;
          DP := DP@.TX_NXT;
          DISPOSE(DQ);
        END;
      SYSFBH := NIL;
      SYSHBE := NIL;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    IF PRINTED LINE NUMBER IS OVER A PARTICULAR  }
{    NUMBER, THEN PRINT THE BOTTOM TITLE AND TOP  }
{    TITLE.                                       }
{-------------------------------------------------}

    PROCEDURE PRINTFOOTHEAD_PROC;
    BEGIN
      IF LINENO >= TXBOT THEN
        BEGIN
          PUTHF_PROC(FALSE);
          PAGENO := PAGENO + 1;
          NEWPAGENO_PROC;
          PUTHF_PROC(TRUE);
        END;
    END;
```

```
{-------------------------------------------------------}
{  FUNCTION :                                           }
{    PRINT THE CONTENTS OF THE INDEX.                   }
{-------------------------------------------------------}

    PROCEDURE PRINTINDEX_PROC;
    VAR
            S, S1, S2, OUTS : STRING;
            J, K, L : INTEGER;
            E, MO : BOOLEAN;
            C : CHAR;
    BEGIN
      IF IP <> NIL THEN
        BEGIN
          PRINTINDEX_PROC(IP@.IX_LFT);

          C := IP@.IX_TXT[1];

          { IF NECESSARY, PRINT THE FIRST }
          { CHARACTER OF THE INDEX ENTRY. }
          IF C <> INDEXCHAR THEN
            BEGIN
              PRINTSPACELINE_PROC;
              INDEXCHAR := C;
              PRINTFOOTHEAD_PROC;
              OUTS := MAKESTRING_FCTN(SYSAD,BLANK) +
                      '-- ' + INDEXCHAR + ' --' + CRLF;
              WRITE(OUTF,OUTS);
              LINENO := LINENO + 1;
              PRINTSPACELINE_PROC;
            END;


        { SAME ROUTINE WILL CONTINUE ON NEXT PAGE }
```

```
      { PRINT AN INDEX ENTRY }
      PREPAREINDEXNO_PROC(S,IP@.IX_PGN);
      S1 := IP@.IX_STR;
      K := LENGTH(S1);
      WHILE (LENGTH(S) > 0) DO
        BEGIN
          MO := TRUE;
          IF K+4+LENGTH(S) > SYSLL THEN
            BEGIN
              MO := FALSE;
              S2 := SUBSTR(S,1,SYSLL-K-4);
              IF S2[LENGTH(S2)] <> COMMA THEN
                BEGIN
                  E := FALSE;
                  L := LENGTH(S2);
                  WHILE (L > 0) AND (NOT E) DO
                    IF S2[L] = COMMA THEN
                      E := TRUE
                    ELSE
                      L := L - 1;
                  S2 := SUBSTR(S2,1,L);
                END;
              S := SUBSTR(S,LENGTH(S2)+1,
                          LENGTH(S)-LENGTH(S2));
            END;

          IF MO THEN
            BEGIN
              S2 := S;
              S := '';
            END;
          PRINTFOOTHEAD_PROC;
          OUTS := MAKESTRING_FCTN(SYSAD,BLANK) +
                  S1 + '     ' + S2 + CRLF;
          WRITE(OUTF,OUTS);
          IF NOT MO THEN
            BEGIN
              S1 := '';
              FOR J := 1 TO K DO
                S1 := S1 + BLANK;
            END;
          LINENO := LINENO + 1;
        END;

      PRINTINDEX_PROC(IP@.IX_RIT);
    END;
END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT SEVERAL SPACE LINES UNTIL END OF PAGE  }
{    OR PARTICULAR NUMBER OF SPACE LINES.         }
{-------------------------------------------------}

    PROCEDURE PRINTLINES_PROC;
    VAR
            S : STRING;
    BEGIN
      I := MIN_FCTN(TXBOT-LINENO,I);
      IF I > 0 THEN
        BEGIN
          S := MAKESTRING_FCTN(I,LF) + CR;
          WRITE(OUTF,S);
          LINENO := LINENO + I;
        END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    IF NOT THE TOP OF THE PAGE, THEN PRINT A     }
{    BLANK LINE.                                  }
{-------------------------------------------------}

    PROCEDURE PRINTSPACELINE_PROC;
    BEGIN
      PRINTFOOTHEAD_PROC;
      IF LINENO <> 0 THEN
        BEGIN
          WRITE(OUTF,LF);
          LINENO := LINENO + 1;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT A TEXT LINE.                           }
{-------------------------------------------------}

    PROCEDURE PRINTTEXT_PROC;
    VAR
            OUTS : STRING;
    BEGIN
      PRINTFOOTHEAD_PROC;
      OUTS := MAKESTRING_FCTN(SYSAD+LBVAL,BLANK) +
              S + CRLF;
      WRITE(OUTF,OUTS):
      LINENO := LINENO + 1;
      PRINTLINES_PROC(SYSLS);
      TIVAL := 0;
      UNVAL := 0;
      LBVAL_PROC;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT THE TABLE OF CONTENTS.                 }
{-------------------------------------------------}

    PROCEDURE PRINTTC_PROC;
    VAR
            OUTS : STRING;
            I, K : INTEGER;
            S, S1 : STRING;
    BEGIN
      K := 1
      WHILE (TP <> NIL) DO
        BEGIN
          IF (TP@.TC_LVN = 1) AND
             (K > 1) THEN
            PRINTSPACELINE_PROC;
          K := TP@.TC_LVN;
          PRINTFOOTHEAD_PROC;
          S := INTTOARB_FCTN(TP@.TC_PGN);
          IF K = 1 THEN
            S1 := BDSTR_FCTN(TP@.TC_TXT)
          ELSE
            S1 := TP@.TC_TXT;
          I := SYSAD + 3 * (K - 1);
          OUTS := MAKESTRING_FCTN(I,BLANK) +
                  S1 + '     ' + S + CRLF;
          WRITE(OUTF,OUTS);
          LINENO := LINENO + 1;
          IF K = 1 THEN
            PRINTSPACELINE_PROC;
          TP := TP@.TC_NXT;
        END;
    END;
```

```
{-------------------------------------------------------}
{  FUNCTION :                                           }
{    PRINT TITLE.                                       }
{-------------------------------------------------------}

    PROCEDURE PRINTTITLE_PROC;
    VAR
            S, LL, MM, RR, OUTS : STRING;
            I, J : INTEGER;
    BEGIN
      { GET THE THREE PARTS OF THE TITLE }
      LL := BUILDTTPART_FCTN(P@.TT_LFT);
      MM := BUILDTTPART_FCTN(P@.TT_MDL);
      RR := BUILDTTPART_FCTN(P@.TT_RIT);
      S := '';

      IF LENGTH(LL) > 0 THEN  { LEFT PART OF TITLE }
        S := S + LL;
      IF LENGTH(MM) > 0 THEN  { MIDDLE PART OF TITLE }
        BEGIN
          J := ((SYSLL - LENGTH(MM)) DIV 2) + 1;
          J := MAX_FCTN(J,1);
          IF J > LENGTH(S) THEN
            FOR I := 1 TO (J - LENGTH(S) - 1) DO
              S := S + BLANK
          ELSE
            S := SUBSTR(S,1,J-1);
          S := S + MM;
        END;
      IF LENGTH(RR) > 0 THEN  { RIGHT PART OF TITLE }
        BEGIN
          J := SYSLL - LENGTH(RR) + 1;
          J := MAX_FCTN(J,1);
          IF J > LENGTH(S) THEN
            FOR I := 1 TO (J - LENGTH(S) - 1) DO
              S := S + BLANK
          ELSE
            S := SUBSTR(S,1,J-1);
          S := S + RR;
        END;

      IF LENGTH(S) = 0 THEN  { PRINT OUT THE TITLE }
        WRITE(OUT,LF)
      ELSE
        BEGIN
          OUTS := MAKESTRING_FCTN(SYSAD,BLANK) +
                  S + CRLF;
          WRITE(OUTF,OUTS);
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE INPUT DATA.                      }
{-------------------------------------------------}

    PROCEDURE PROCESS_PROC;
    BEGIN
      S := RMTB_FCTN(S);
      IF LENGTH(S) > 0 THEN
        IF S[1] = PERIOD THEN
          COMMAND_PROC(S)
        ELSE
          TEXT_PROC(S)
      ELSE
        BEGIN   { EMPTY LINE }
          BREAK_PROC(TRUE);
          PRINTTEXT_PROC('');
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .ps SYSTEM CONTROL WORD.         }
{    .ps character                                }
{-------------------------------------------------}

    PROCEDURE PS_PROC;
    VAR
            P1 : STRING;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (LENGTH(P1) <> 1) OR
           (NOT ISPUNCT_PROC(P1[1])) THEN
          ERROR := TRUE
        ELSE
          SYSPS := P1[1];
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT THE TITLE. IF THE PARAMETER IS TRUE,   }
{    THEN PRINT TOP TITLE. IF IT IS FALSE, THEN   }
{    PRINT BOTTOM TITLE.                          }
{-------------------------------------------------}

    PROCEDURE PUTHF_PROC;
    VAR
            REPLY : STRING;
            J : INTEGER;
    BEGIN
      IF E THEN
        BEGIN  { TOP TITLE }
          IF NOT PAGETYPE THEN
            BEGIN  { ODD TOP TITLE }
              J := SYSTMO - SYSHSO - SYSHMO;
              PUTTITLE_PROC(J,SYSHSO,SYSHMO,SYSTTO);
            END
          ELSE
            BEGIN  { EVEN TOP TITLE }
              J := SYSTME - SYSHSE - SYSHME;
              PUTTITLE_PROC(J,SYSHSE,SYSHME,SYSTTE);
            END;
          LINENO := 0;
        END
      ELSE  { BOTTOM TITLE }
        BEGIN
          IF NOT PAGETYPE THEN
            BEGIN  { ODD BOTTOM TITLE }
              J := SYSBMO - SYSFMO - SYSFSO;
              PUTTITLE_PROC(SYSFMO,SYSFSO,J,SYSBTO);
            END
          ELSE
            BEGIN  { EVEN BOTTOM TITLE }
              J := SYSBME - SYSFME - SYSFSE;
              PUTTITLE_PROC(SYSFME,SYSFSE,J,SYSBTE);
            END;
          IF PAUSE THEN
            READLN(REPLY);
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT INDEX. THEN SKIP THE THE TOP OF        }
{    NEXT PAGE.                                   }
{-------------------------------------------------}

    PROCEDURE PUTINDEX_PROC;
    BEGIN
      IF SYSIX <> NIL THEN
        BEGIN
          BREAK_PROC(TRUE);
          PRINTINDEX_PROC(SYSIX);
          TONEWPAGE_PROC;
          NEWPAGENO_PROC;
        END;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT THE TABLE OF CONTENTS.                 }
{-------------------------------------------------}

    PROCEDURE PUTTC_PROC;
    BEGIN
      IF SYSTCH <> NIL THEN
        BEGIN
          BREAK_PROC(TRUE);
          PRINTTC_PROC(SYSTCH);
          TONEWPAGE_PROC;
          NEWPAGENO_PROC;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PRINT THE TITLE AND THE BLANK LINES BEFORE   }
{    AND AFTER IT.                                }
{-------------------------------------------------}

    PROCEDURE PUTTITLE_PROC;
    VAR
            OUTS : STRING;
            J : INTEGER;
    BEGIN
      OUTS := MAKESTRING_FCTN(V1,LF) + CR;
      WRITE(OUTF,OUTS);
      J := 0;

      WHILE (P <> NIL) AND (J < V2) DO
        BEGIN
          PRINTTITLE_PROC(P);
          P := P@.TT_NXT;
          J := J + 1;
        END;

      OUTS := MAKESTRING_FCTN(V3,LF) + CR;
      WRITE(OUTF,OUTS);
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    REVERSE A STRING.                            }
{-------------------------------------------------}

    FUNCTION REVERSE_FCTN;
    VAR
            S1 : STRING;
            J : INTEGER;
    BEGIN
      S1 := '';
      J := LENGTH(S);
      WHILE (J > 0) DO
        BEGIN
          S1 := S1 + S[J];
          J := J - 1;
        END;
      REVERSE_FCTN := S1;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE TEXT LINE WHEN THE RIGHT         }
{    ADJUST FLAG IS ON.                           }
{-------------------------------------------------}

    PROCEDURE RI_PROC;
    VAR
            J, K : INTEGER;
    BEGIN
      K := TXVAL - I + 1;
      K := MAX_FCTN(K,1);
      FOR J := 1 TO (K - 1) DO
        S := BLANK + S;
      PRINTTEXT_PROC(S);
      MINUSONE_PROC(RIVAL,RIBOL);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .ro SYSTEM CONTROL WORD.         }
{    .ro < big | small >                          }
{-------------------------------------------------}

    PROCEDURE RO_PROC;
    VAR
            P1 : STRING;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (LENGTH(P1) = 3) AND (P1 = 'big') THEN
          ROMBG := TRUE
        ELSE
          IF (LENGTH(P1) = 5) AND (P1 = 'small') THEN
            ROMBG := FALSE
          ELSE
            ERROR := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    REMOVE THE LEADING BLANKS OF A STRING.       }
{-------------------------------------------------}

    FUNCTION RMLB_FCTN;
    VAR
            I, J : INTEGER;
    BEGIN
      IF LENGTH(S) = 0 THEN
        RMLB_FCTN := S
      ELSE
        BEGIN
          I := VERIFY_FCTN(S,BLANK);
          IF I = 0 THEN
            BEGIN
              I := 1;
              J := 0;
            END
          ELSE
            J := LENGTH(S) - I + 1;
          RMLB_FCTN := SUBSTR(S,I,J);
        END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    REMOVE THE TRAILING BLANKS OF A STRING.      }
{-------------------------------------------------}

    FUNCTION RMTB_PROC;
    VAR
            I : INTEGER;
            E : BOOLEAN;
    BEGIN
      IF LENGTH(S) = 0 THEN
        RMTB_FCTN := S
      ELSE
        BEGIN
          I := LENGTH(S);
          E := FALSE;
          WHILE (I > 0) AND (NOT E) DO
            IF S[I] = BLANK THEN
              I := I - 1
            ELSE
              E := TRUE;
          RMTB_FCTN := SUBSTR(S,1,I);
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SEARCH THE INDEX BINARY SEARCH TREE FOR      }
{    A PARTICULAR INDEX ENTRY.                    }
{-------------------------------------------------}

    PROCEDURE SEARCHINDEX_PROC;
    BEGIN  { 0 -- EMPTY TREE }
      IF (I = 0) AND (P <> NIL) THEN
        IF T = P@.IX_TXT THEN
          I := 3  { 3 -- FOUND }
        ELSE
          IF T > P@.IX_TXT THEN
            IF P@.IX_RIT <> NIL THEN
              BEGIN
                P := P@.IX_RIT;
                SEARCHINDEX_PROC(T,P,I);
              END
            ELSE
              I := 2  { 2 -- RIGHT SUBTREE }
          ELSE
            IF P@.IX_LFT <> NIL THEN
              BEGIN
                P := P@.IX_LFT;
                SEARCHINDEX_PROC(T,P,I);
              END
            ELSE
              I := 1;  { 1 -- LEFT SUBTREE }
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SEARCH THE MACRO BINARY SEARCH TREE FOR      }
{    A PARTICULAR MACRO NAME.                     }
{-------------------------------------------------}

    PROCEDURE SEARCHMACRO_PROC;
    BEGIN  { 0 -- EMPTY TREE }
      IF (I = 0) AND (P <> NIL) THEN
        IF A = P@.DM_NAM THEN
          I := 3  { 3 -- FOUND }
        ELSE
          IF A > P@.DM_NAM THEN
            IF P@.DM_RIT <> NIL THEN
              BEGIN
                P := P@.DM_RIT;
                SEARCHMACRO_PROC(A,P,I);
              END
            ELSE
              I := 2  { 2 -- RIGHT SUBTREE }
          ELSE
            IF P@.DM_LFT <> NIL THEN
              BEGIN
                P := P@.DM_LFT;
                SEARCHMACRO_PROC(A,P,I);
              END
            ELSE
              I := 1;  { 1 -- LEFT SUBTREE }
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    ACCORDING TO THE VALUE OF A BOOLEAN VARIABLE,}
{    SELECT A NUMBER FROM TWO NUMBERS.            }
{-------------------------------------------------}

    FUNCTION SELECT_FCTN;
    BEGIN
      SELECT_FCTN := K;
      IF E THEN
        SELECT_FCTN := J;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SEPARATE THE INPUT TITLE INTO THREE PARTS.   }
{-------------------------------------------------}

    PROCEDURE SEPTITLE_PROC;
    VAR
            C : CHAR;
            I : INTEGER;
    BEGIN
      E := TRUE;
      IF LENGTH(S) >= 4 THEN
        BEGIN
          C := S[1];  { C -- DELIMITER }
          IF ISPUNCT_FCTN(C) THEN
            BEGIN
              S := SUBSTR(S,2,LENGTH(S)-1);
              I := INDEX(S,C);
              IF (I > 0) AND (I < LENGTH(S)) THEN
                BEGIN  { FIRST PART OF TITLE }
                  S1 := SUBSTR(S,1,I-1);
                  S := SUBSTR(S,I+1,LENGTH(S)-I);
                  I := INDEX(S,C);
                  IF (I > 0) AND (I < LENGTH(S)) THEN
                    BEGIN  { SECOND PART OF TITLE }
                      S2 := SUBSTR(S,1,I-1);
                      S := SUBSTR(S,I+1,LENGTH(S)-I);
                      I := INDEX(S,C);
                      IF I = LENGTH(S) THEN
                        BEGIN  { THIRD PART OF TITLE }
                          S3 := SUBSTR(S,1,I-1);
                          E := FALSE;
                        END;
                    END;
                END;
            END;
        END;
    END;
```

```
{-------------------------------------------------------}
{  FUNCTION :                                           }
{    SET THE VALUE OF SYSTEM VARIABLE IF THE            }
{    CONTROL WORD IS IN THE FOLLOWING FORM.             }
{    .xx < 1 | n >                                      }
{-------------------------------------------------------}

    PROCEDURE SET1N_PROC;
    VAR
            P1 : STRING;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        BEGIN
          IF (LENGTH(P1) = 1) AND (P1 = NULL) THEN
            P1 := '1';
          IF NOT ALLDIGITS_FCTN(P1) THEN
            ERROR := TRUE
          ELSE
            BEGIN
              BREAK_PROC(E);
              I := INTSTR_FCTN(P1);
            END;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SET THE VALUE OF SYSTEM VARIABLE IF THE      }
{    CONTROL WORD IS IN THE FOLLOWING FORM.       }
{    .xx < 1 | n | on | off >                     }
{-------------------------------------------------}

    PROCEDURE SET1NOO_PROC;
    VAR
            P1 : STRING;
            J : INTEGER;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        BEGIN
          IF (LENGTH(P1) = 1) AND (P1 = NULL) THEN
            P1 := '1';
          J := ISOO_PROC(P1);
          IF (NOT ALLDIGITS_FCTN(P1)) AND (J = 0) THEN
            ERROR := TRUE
          ELSE
            BEGIN
              BREAK_PROC(E);
              IF J = 0 THEN
                BEGIN  { .xx < 1 | n > }
                  I := INTSTR_FCTN(P1);
                  IF I = 0 THEN
                    B := FALSE
                  ELSE
                    B := TRUE;
                END
              ELSE
                IF J = 1 THEN
                  BEGIN  { .xx on }
                    I := -1;
                    B := TRUE;
                  END
                ELSE  { .xx off }
                  BEGIN
                    B := FALSE;
                    I := 0;
                  END;
            END;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SET THE VALUE OF THE SYSTEM VARIABLE IF THE  }
{    CONTROL WORD IS IN THE FOLLOWING FORM.       }
{    .xx < all | even | odd > < n | +n | -n >     }
{-------------------------------------------------}

    PROCEDURE SETAEON_PROC;
    VAR
            P1, P2 : STRING;
            K : INTEGER;
    BEGIN
      PARA2_PROC(S,P1,P2,ERROR);
      IF NOT ERROR THEN
        BEGIN
          K := ISAEO_FCTN(P1);
          IF (K = 0) OR (ISNUMBER_FCTN(P2) = 0) THEN
            ERROR := TRUE
          ELSE
            BEGIN
              BREAK_PROC(E);

              { 'all' OR 'odd' }
              IF (K = 1) OR (K = 2) THEN
                J := SETNUM_FCTN(J,P2);

              { 'all' OR 'even' }
              IF (K = 1) OR (K = 3) THEN
                J := SETNUM_FCTN(I,P2);
            END;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SET THE VALUE OF THE SYSTEM VARIABLE IF THE  }
{    CONTROL WORD IS IN THE FOLLOWING FORM.       }
{    .xx < all | even | odd > n /s1/s2/s3/         }
{-------------------------------------------------}

    PROCEDURE SETAEONT_PROC;
    VAR
            P1, P2, P3 : STRING;
            S1, S2, S3 : STRING;
            I, J : INTEGER;
    BEGIN
      PARA3_PROC(S,P1,P2,P3,ERROR);
      IF NOT ERROR THEN
        IF (ISAEO_FCTN(P1) = 0) OR
           (NOT ALLDIGITS_FCTN(P2)) THEN
          ERROR := TRUE
        ELSE
          BEGIN
            SEPTITLE_PROC(P3,S1,S2,S3,ERROR);
            IF NOT ERROR THEN
              BEGIN
                I := INTSTR_FCTN(P2);
                IF I > 30 THEN
                  ERROR := TRUE
                ELSE
                  BEGIN
                    J := ISAEO_FCTN(P1);

                    { 'all' OR 'odd' }
                    IF (J = 1) OR (J = 2) THEN
                      IF ((NOT E) AND (I > SYSHSO)) OR
                         (E AND (I > SYSFSO)) THEN
                        ERROR := TRUE
                      ELSE
                        SETTITLE_PROC(Q,I,S1,S2,S3);

                    { 'all' OR 'even' }
                    IF (J = 1) OR (J = 3) THEN
                      IF ((NOT E) AND (I > SYSHSE)) OR
                         (E AND (I > SYSFSE)) THEN
                        ERROR := TRUE
                      ELSE
                        SETTITLE_PROC(P,I,S1,S2,S3);
                  END;
              END;
          END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SET THE VALUE OF THE SYSTEM VARIABLE IF THE  }
{    CONTROL WORD IS IN THE FOLLOWING FORM.       }
{    .xx < n | +n | -n >                          }
{-------------------------------------------------}

    PROCEDURE SETN_PROC;
    VAR
            P1 : STRING;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF ((LENGTH(P1) = 1) AND (P1 = NULL)) OR
           (ISNUMBER_FCTN(P1) = 0) THEN
          ERROR := TRUE
        ELSE
          BEGIN
            BREAK_PROC(TRUE);
            I := SETNUM_FCTN(I,P1);
          END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SET THE VALUE OF THE SYSTEM VARIABLE IF THE  }
{    CONTROL WORD IS IN THE FOLLOWING FORM.       }
{    .xx n /s1/s2/s3/                             }
{-------------------------------------------------}

    PROCEDURE SETNT_PROC;
    VAR
            P1, P2 : STRING;
            S1, S2, S3 : STRING;
            J : INTEGER;
    BEGIN
      PARA2NT_PROC(S,P1,P2,ERROR);
      IF NOT ERROR THEN
        IF NOT ALLDIGITS_FCTN(P1) THEN
          ERROR := TRUE
        ELSE
          BEGIN
            SEPTITLE_PROC(P2,S1,S2,S3,ERROR);
            IF NOT ERROR THEN
              BEGIN
                J := INTSTR_FCTN(P1);
                IF J > 30 THEN
                  ERROR := TRUE
                ELSE
                  BEGIN
                    CASE I OF
                      0 : IF J > SYSFSE THEN
                            ERROR := TRUE;
                      1 : IF J > SYSHSE THEN
                            ERROR := TRUE;
                      2 : IF J > SYSFSO THEN
                            ERROR := TRUE;
                      3 : IF J > SYSHSO THEN
                            ERROR := TRUE;
                    END;
                    IF NOT ERROR THEN
                      SETTITLE_PROC(P,I,S1,S2,S3);
                  END;
              END;
          END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    ACCORDING TO THE KIND OF SECOND PARAMETER,   }
{    CHANGE THE VALUE OF FIRST PARAMETER.         }
{-------------------------------------------------}

    FUNCTION SETNUM_FCTN;
    VAR
            J : INTEGER;
    BEGIN
      J := ISNUMBER_FCTN(S);
      IF J = 1 THEN  { 1 -- UNSIGNED NUMBER }
        I := INTSTR_FCTN(S)
      ELSE
        BEGIN
          S := SUBSTR(S,2,LENGTH(S)-1);
          IF J = 2 THEN  { 2 -- POSITIVE NUMBER }
            I := I + INTSTR_FCTN(S)
          ELSE  { 3 -- NEGATIVE NUMBER }
            I := I - INTSTR_FCTN(S);
        END;
      SETNUM_FCTN := MAX_FCTN(I,0);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SET THE VALUE OF THE SYSTEM VARIABLE IF THE  }
{    CONTROL WORD IS IN THE FOLLOWING FORM.       }
{    .xx < on | off >                             }
{-------------------------------------------------}

    PROCEDURE SETOO_PROC;
    VAR
            P1 : STRING;
            I : INTEGER;
    BEGIN
      PARA1_PROC(S,P1,ERROR);
      IF NOT ERROR THEN
        IF (LENGTH(P1) = 1) AND (P1 = NULL) THEN
          ERROR := TRUE
        ELSE
          BEGIN
            I := ISOO_FCTN(P1);
            IF I = 0 THEN
              ERROR := TRUE
            ELSE
              BEGIN
                BREAK_PROC(TRUE);
                IF I = 1 THEN  { 1 -- on }
                  B := TRUE
                ELSE  { 2 -- off }
                  B := FALSE;
              END;
          END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    CHANGE THE THREE PARTS OF A TITLE ENTRY.     }
{-------------------------------------------------}

    PROCEDURE SETTITLE_PROC;
    VAR
            J : INTEGER;
    BEGIN
      I := I - 1;
      FOR J := 1 TO I DO
        P := P@.TT_NXT;
      P@.TT_LFT := S1;
      P@.TT_MDL := S2;
      P@.TT_RIT := S3;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .sk SYSTEM CONTROL WORD.         }
{    .sk < l | n >                                }
{-------------------------------------------------}

    PROCEDURE SK_PROC;
    VAR
            I : INTEGER;
    BEGIN
      SETlN_PROC(S,SKVAL,TRUE);
      IF NOT ERROR THEN
        WHILE (SKVAL > 0) DO
          BEGIN
            PRINTFOOTHEAD_PROC;
            I := MIN_FCTN(TXBOT-LINENO,SKVAL);
            PRINTLINES_PROC(I);
            SKVAL := SKVAL - I;
          END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE .sp SYSTEM CONTROL WORD.         }
{    .sp < l | n >                                }
{-------------------------------------------------}

    PROCEDURE SP_PROC;
    BEGIN
      SETlN_PROC(S,SPVAL,TRUE);
      IF NOT ERROR THEN
        PRINTLINES_PROC(SPVAL);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    IF NECESSARY, THEN INSERT EXTRA BLANKS INTO  }
{    A STRING UNTIL THE LENGTH OF THIS STRING     }
{    REACHES A PARTICULAR NUMBER.                 }
{-------------------------------------------------}

  FUNCTION SPREAD_FCTN;
  VAR
        J, K : INTEGER;
  BEGIN
    SPREAD_FCTN := S;
    IF JUBOL THEN
      BEGIN
        J := WIDTH_FCTN(S);
        J := I - J;
        IF J > 0 THEN
          BEGIN
            K := INDEX(S,BLANK);
            IF K > 0 THEN
              BEGIN
                IF DIRECTION THEN
                  BEGIN   { FORWARD INSERTION }
                    S := FORWARD_FCTN(S,J);
                    DIRECTION := FALSE;
                  END
                ELSE
                  BEGIN   { BACKWARD INSERTION }
                    S := BACKWARD_FCTN(S,J);
                    DIRECTION := TRUE;
                  END;
                SPREAD_FCTN := S;
              END;
          END;
      END;
  END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    ACCORDING TO THE VALUES OF SEVERAL BOOLEAN   }
{    VARIABLES, CHANGE THE CONTENTS OF A STRING.  }
{-------------------------------------------------}

    FUNCTION STRCOPY_FCTN;
    BEGIN
      IF USBOL THEN
        BEGIN  { UNDERSCORE }
          S := USSTR_FCTN(S);
          MINUSONE_PROC(USVAL,USBOL);
        END;

      IF BDBOL THEN
        BEGIN { BOLD }
          S := BDSTR_FCTN(S);
          MINUSONE_PROC(BDVAL,BDBOL);
        END;

      IF UPBOL THEN
        BEGIN  { UPPERCASE }
          S := UPSTR_FCTN(S);
          MINUSONE_PROC(UPVAL,UPBOL);
        END;
      STRCOPY_FCTN := S;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CONVERSE A STRING FROM STRING TYPE TO        }
{    TEXT TYPE.                                   }
{-------------------------------------------------}

    FUNCTION STRTOTEXT_FCTN;
    VAR
            I : INTEGER;
            T : TEXTTYPE;
    BEGIN
      S := RMLB_FCTN(S);
      S := RMTB_FCTN(S);
      FOR I := 1 TO 80 DO
        T[I] := BLANK;
      IF LENGTH(S) > 0 THEN
        IF LENGTH(S) <= 80 THEN
          FOR I := 1 TO LENGTH(S) DO
            T[I] := S[I]
        ELSE
          FOR I := 1 TO 80 DO
            T[I] := S[I];
      STRTOTEXT_FCTN := T;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    PUT AN INPUT DATA INTO THE TABLE OF CONTENTS }
{    LINKED LIST.                                 }
{-------------------------------------------------}

    PROCEDURE TC_PROC;
    VAR
            TQ : TC_P;
    BEGIN
      NEW(TQ);
      TQ@.TC_TXT := S;
      TQ@.TC_PGN := PAGENO;
      TQ@.TC_LVN := TCVAL;
      TQ@.TC_NXT := NIL;
      IF SYSTCH = NIL THEN
        SYSTCH := TQ
      ELSE
        SYSTCE@.TC_NXT := TQ;
      SYSTCE := TQ;
      TCVAL := 0;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    PROCESS THE INPUT DATA WHEN THE INPUT DATA   }
{    IS TEXT LINE.                                }
{-------------------------------------------------}

    PROCEDURE TEXT_PROC;
    VAR
            I : INTEGER;
    BEGIN
      IF START_FLAG THEN
        BEGIN  { FIRST PASS }
          PUTHF_PROC(TRUE);
          START_FLAG := FALSE;
        END;

      IF UPBOL THEN
        S := UPSTR_FCTN(S);

      IF IXBOL THEN  { INDEX }
        IX_PROC(S);

      IF TCVAL > 0 THEN  { TABLE OF CONTENTS }
        TC_PROC(S);

      S := STRCOPY_FCTN(S);
      I := WIDTH_FCTN(S);

      IF CEBOL THEN  { CENTER }
        CE_PROC(S,I)
      ELSE
        IF RIBOL THEN  { RIGHT ADJUST }
          RI_PROC(S,I)
        ELSE
          IF NOT COBOL THEN { NOT CONCATENATE }
            BEGIN
              S := SPREAD_FCTN(S,TXVAL);
              PRINTTEXT_PROC(S);
            END
          ELSE  { CONCATENATE }
            CO_PROC(S);
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    SKIP TO THE TOP OF NEXT PAGE.                }
{-------------------------------------------------}

    PROCEDURE TONEWPAGE_PROC;
    BEGIN
      IF LINENO <= TXBOT THEN
        BEGIN
          PRINTLINES_PROC(TXBOT);
          LINENO := LINENO + 1;
          PUTHF_PROC(FALSE);
          PAGENO := PAGENO + 1;
        END;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    COMPUTE THE TEXT LINE AREA OF THE ODD PAGE   }
{    AND EVEN PAGE.                               }
{-------------------------------------------------}

    PROCEDURE TXBOT_PROC;
    BEGIN
      TXBOTE := SYSPL - SYSTME - SYSBME;
      TXBOTO := SYSPL - SYSTMO - SYSBMO;
      TXBOT := SELECT_FCTN(PAGETYPE,TXBOTE,TXBOTO);
      IF (TXBOTE <= 0) OR (TXBOTO <= 0) THEN
        ERROR := TRUE;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    CHANGE CHARACTERS OF A STRING TO UPPERCASE.  }
{-------------------------------------------------}

    FUNCTION UPSTR_FCTN;
    VAR
            I : INTEGER;
    BEGIN
      FOR I := 1 TO LENGTH(S) DO
        IF ISLOWER_FCTN(S[I]) THEN
          S[I] := CHR(ORD(S[I])-32);
      UPSTR_FCTN := S;
    END;




{-------------------------------------------------}
{  FUNCTION :                                     }
{    ADD UNDERLINE TO THE CHARACTERS OF A STRING. }
{    PUNCTUATUINS ARE NOT UNDERLINED.             }
{-------------------------------------------------}

    FUNCTION USSTR_PROC;
    VAR
            S1 : STRING;
            I : INTEGER;
    BEGIN
      S1 := '';
      I := 1;
      WHILE (I <= LENGTH(S)) DO
        IF S[I] = BS THEN
          BEGIN
            S1 := S1 + S[I] + S[I+1];
            I := I + 2;
          END
        ELSE
          BEGIN
            IF (S[I] <> BLANK) AND
               (NOT ISPUNCT_FCTN(S[I])) THEN
              S1 := S1 + UDLN + BS + S[I]
            ELSE
              S1 := S1 + S[I];
            I := I + 1;
          END;
      USSTR_FCTN := S1;
    END;
```

```
{-------------------------------------------------}
{  FUNCTION :                                     }
{    FIND THE POSITION OF THE FIRST CHARACTER THAT }
{    IS NOT EQUAL TO A PARTICULAR CHARACTER.      }
{-------------------------------------------------}

    FUNCTION VERIFY_FCTN;
    VAR
            I, J : INTEGER;
            E : BOOLEAN;
    BEGIN
      VERIFY_FCTN := 0;
      J := LENGTH(S);
      IF J > 0 THEN
        BEGIN
          I := 1;
          E := FALSE;
          WHILE (I <= J) AND (NOT E) DO
            IF S[I] <> C THEN
              BEGIN
                E := TRUE;
                VERIFY_FCTN := I;
              END
            ELSE
              I := I + 1;
        END;
    END;



{-------------------------------------------------}
{  FUNCTION :                                     }
{    COMPUTE THE REAL LENGTH OF A STRING.         }
{-------------------------------------------------}

    FUNCTION WIDTH_FCTN;
    VAR
            I, J : INTEGER;
    BEGIN
      I := 1;
      J := 0;
      WHILE (I <= LENGTH(S)) DO
        IF S[I] = BS THEN
          I := I + 2
        ELSE
          BEGIN
            I := I + 1;
            J := J + 1;
          END;
      WIDTH_FCTN := J;
    END;
```

```
{-------------------------------------------------}
{  MAIN ROUTINE                                   }
{                                                 }
{    READ THE INPUT DATA, REMOVE THE TRAILING     }
{    BLANKS, AND PROCESS THEM.                    }
{-------------------------------------------------}

BEGIN
  INITIAL_PROC;
  IF NOT EOF(INF) THEN
    BEGIN
      REPEAT
        READLN(INF,INSTR);
        CARDNO := CARDNO + 1;
        INSTR := RMTB_FCTN(INSTR);
        PROCESS_PROC(INSTR);
      UNTIL EOF(INF) OR ERROR;
      END_PROC;
    END;
END.
```

Appendix B

Sample Input Data

```
.tm all 8
.bm all 7
.ds
.ad all 11
.fo off
.co on
.ti 5
Twenty years ago, digital computer technology in
the form of centralized digital control systems
caused the first fundamental change in industrial
process control.
Unfortunately, these systems, which partly replaced
analog controllers, had a serious disadvantage.
All control functions were dependent on one device,
and back-up controls -- either provision of analog
standby or duplication of the digital computer
systems -- were necessary.
     Today, powerful 16-bit and 32-bit microcomputers
are changing the structure of control systems.
Together with a new way of developing control systems,
this structure is forcing us to redefine the tasks of
control engineers, who must now become real-time
software engineers as well.
     When a computer is controlling a technical process,
its software must interact with the dynamic properties
of the industrial system and must react to
stochastically occurring events within the industrial
system.
It is this interaction between control computer
software and the technical process that leads to two
major real-time requirements :
.sp
.in 6
.ir 6
.un 3
1. Requirements to perform control actions at a certain
point in time. The following example illustrates
this using
.up
peral
(Process and Experiment Automation Real-time Language) :
.sp
.ce 2
.up 2
at 11:45 every 10 sec during 2 min
activate measurement priority 2;
.sp
[Activate the task MEASUREMENT at 11:45 a.m. and repeat
this task activation every 10 seconds during a time
interval of two minutes with priority 2]
.sp
```

.un 3
2. Requirements to perform control actions dependent on stochastically occurring events within the process plant (such as to handle a pressure surge in a chemical rector); for example, in PEARL :
.sp
.ce on
.up on
when pressure acticate valve;
.sp
.ce off
.up off
[If the interrupt signal designated by PRESSURE occurs, activate the task VALVE].
.in -6
.ir 0
.sp
With both types of requirements, we must have a certain control software reaction time as well as the synchronization of parallel tasks within the software system.

Appendix C

Sample Output

Twenty years ago, digital computer technology in the form of centralized digital control systems caused the first fundamental change in industrial process control. Unfortunately, these systems, which partly replaced analog controllers, had a serious disadvantage. All control functions were dependent on one device, and back-up controls -- either provision of analog standby or duplication of the digital computer systems -- were necessary.

Today, powerful 16-bit and 32-bit microcomputers are changing the structure of control systems. Together with a new way of developing control systems, this structure is forcing us to redefining the tasks of control engineers, who must now become real-time software engineers as well.

When a conputer is controlling a technical process, its software must interact with the dynamic properties of the industrial system and must react to stochastically occurring events within the industrial system. It is this interaction between control computer software and the technical process that leads to two major real-time requirements :

1. Requirements to perform control actions at a certain point in time. The following example illustrates this using PEARL (Process and Experiment Automation Real-time Lenguage) :

```
AT 11:45 EVERY 10 SEC DURING 2 MIN
ACTIVATE MEASUREMENT PRIORITY 2;
```

[Activate the task MEASUREMENT at 11:45 a.m. and repeat this task activation every 10 seconds during a time interval of two minutes with priority 2]

2. Requirements to perform control actions dependent on stochastically occurring events within the process plant (such as to handle a pressure surge in a chemical rector); for example, in PEARL :

```
WHEN PRESSURE ACTIVATE VALVE;
```

[If the interrupt signal designated by PRESSURE occurs, activate the task VALVE].

With both types of requirements, we must have a certain control software reaction time as well as the synchronization of parallel tasks within the software system.

Appendix D

STF USER'S MANUAL

This is the user's manual of STF system. In this manual, how to use this system and the function of each control word are described in detail. All the control words are listed in the alphabetic order so that a particular control word can be referenced quickly.

How to use this STF system :

1. When the "$", the VAM/VMS prompt, appears, type "RUN DRB2:[IE68]STF". System will reply the following message :

   "enter input file name : "

2. Type in your input file name. System replys

   "If online printouts, then press return."
   "Otherwise, enter output file name : "

3. If you wish your printout to be printed on the terminal, then press return. Otherwise, type in the output file name.

4. If no output file name is provided, then system replys

   "Pause or not ? (Y/N) "

   If you answer "Y" (yes), then a pause is generated after a page is printed. Otherwise, no pause is generated. Then the following message appears :

   "load paper and press return"

   The formatted text will start to be printed after you press the return.

5. If you type in the output file name, then the "$" appears a moment later. It means that the process is over. You can then type "TYPE output-file-name" to obtain your printout. Using this method, you can save the formatted text in your account for later use.

6. If you want to quite when the system is running, simply type CTRL/Y, that is, press down the "CONTROL" key and press the "Y", to terminate the process.

ADJUST control word

The ADJUST control word causes all output to be moved to the right of the physical left print margin.

ALL effects the adjustment values of even and odd numbered pages.
EVEN effects the adjustment value of even numbered pages only.
ODD effects the adjustment value of odd numbered pages only.

The .AD control word causes the logical left margin of the formatted printout to be moved to the right of the actual left margin of the output device (printer or terminal). This adjustment value remains in effect for all subsequent lines until altered by another ".AD" control word. An parameter of the form "+n" adds the value to the current adjustment value. An parameter of the form "-n" subtracts the value from the current adjustment value.

This control word creates a break. The default for an even or odd adjust is zero.

BOLD control word

The BOLD control word overstrikes an input line with itself for a boldface effect.

n specifies that the next "n" input text records are to be made bold by overstriking each character with itself. If "n" is omitted, a value of 1 is assumed to overstrike the next input text record. If the value of "n" is zero, then it is the same as the "OFF".

ON specifies that all following input text records are to be made bold.

OFF terminates the bold translation after an "ON" was specified. If "n" was given and has not yet been exhausted, an "OFF" parameter will terminate bold also.

The .BD control word overstrikes each printable character of text with itself to produce a bold output effect. The BOLD control word operates independently of other control words that modify text. When more than one of ".BD", ".BI", ".UC", ".UP", ".US" are in effect, the result is the best equivalent of the sum of the effects. Each must be disabled in any order to cancel them all.

This control word does not create a break. If no parameter is presented, 1 is assume to bold the next input text record.

BOLD ITALIC control word

The BOLD ITALIC control word overstrikes an input line with itself and underscores it for a boldface italic effect.

n specifies that the next "n" input text records are to be made bold italic by overstriking each character with itself and underscoring the result. If "n" is omitted, a value of 1 is assumed to overstrike and underscore the next input text record. If the value of "n" is zero, then it is the same as the "OFF".

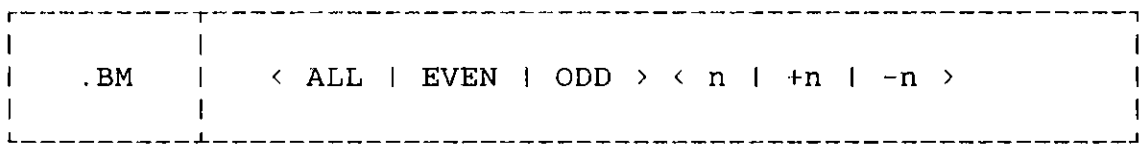
ON specifies that all following input text records are to be made bold italic.

OFF terminates the bold italic translation after an "ON" was specified. If "n" was given and has not yet been exhausted, an "OFF" parameter will terminate bold italic also.

The .BI control word overstrikes each printable character of text with itself to produce a bold effect. Furthermore all alphanumeric character are underscored also.

This control word does not create a break. If no parameter is present, 1 is assumed to bold italicize the next input text record.

BOTTOM MARGIN control word



The BOTTOM MARGIN control word specifies the number of lines which are to appear between the bottom of the output page and the last line of ordinary text.

ALL effects the bottom margin settings of even and odd numbered pages.
EVEN effects the bottom margin setting of even numbered pages only.
ODD effects the bottom margin setting of odd numbered pages only.

At the bottom of all subsequent output pages (including the current page), "n" lines will appear between the bottom of printed text and the physical bottom of the page. An parameter of the form "+n" or "-n" adds this value algebraically to the current bottom margin setting, so long as the resulting value is not negative.

This control word does not create a break. Unless otherwise specified n = 6 will be in effect. At no time may the value set in .BM be smaller than the sum of the .FM and .FS values.

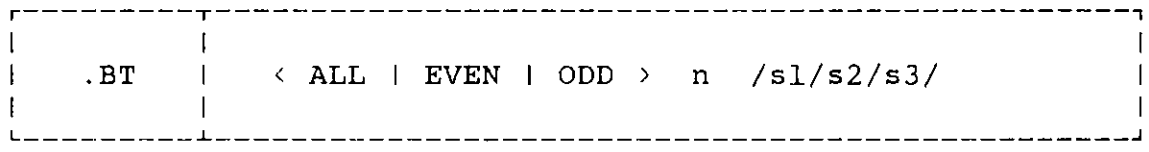
BREAK control word

.BR	

BREAK control word causes the immediately previous line to be printed without filling in with words from the next line. The .BR control word is used to prevent concatenation of lines such as paragraph headings or the last line of a paragraph. It causes the preceding line to be typed as a short line if it is shorter than the current line length.

This control word creates a break. That is its only function. Many of the other control words act as a BREAK. No BREAK is necessary when one of these is presented. A blank in column one of an input line has the effect of a BREAK immediately before the line. If NO CONCATENATE is in effect, all lines appear to be followed by a BREAK.

BOTTOM TITLE control word



The BOTTOM TITLE control word is used to define three items of title information to be printed at the bottom of even and odd numbered pages.

ALL effects the bottom titles of even and odd numbered pages.
EVEN effects the bottom title of even numbered pages only.
ODD effects the bottom title of odd numbered pages only.

The value of "n", from one to the maximum value of the FOOTING SPACE (.FS), gives the footing line number and s1, s2, s3 are character strings not containing the delimiter character "/". The delimiter character can be any punctuation, defined as the first character of the parameter. Any of the fields may be omitted, but the delimiter character must be included to indicate missing fields.

The .BT control word is used in a way similar to the .TT control word. The title items defined with .BT control word will be printed in footing lines near the bottom of even and odd numbered pages. The number of footing lines printed is set by .FS (FOOTING SPACE).

A break is not created by this control. Unless otherwise specified ".BT ////" will be in effect.

CENTER control word

.CE	< 1 | n | ON | OFF >

The line following the CENTER control word will be centered between the margins. The next "n" text lines in the input file will be centered between the left and right margins. If line to be centered is longer than the current line length, it will not be centered. The left and right margins are the value of any indent value (.IN), undent value (.UN), temporary indent value (.TI) and the current line length altered by indent right (.IR), respectively.

This control word causes a break. A numeric parameter will center the following "n" input lines. An "ON" parameter will center all following input lines until "OFF" parameter or a RIGHT ADJUST control word (.RI) is encountered. If no parameter is present, then 1 is assumed to center the next input text record.

COMMENT control word

.CM	information

The COMMENT control word is ignored and may be used to enter comments into a script text file. The .CM control word allows comments to be stored in the script text file. These comments may be seen whenever the input file is edited or printed. Comment lines may be used to store unique identifications for use during editing of the file.

This control word does not cause a break.

CONCATENATE control word

.CO	< ON | OFF >

The CONCATENATE control word enables or disables the formation of output text lines by concatenating input text lines and truncating at the nearest word boundary to fit within the specified line length. The ".CO ON" control word specifies that output text lines are to be formed by shifting words to or from the next input text line. The resulting line will be as close to the line length as possible without exceeding it or splitting a word.

The ".CO OFF" control word prevents the shifting of words from line to line which results in : one line in, one line out. If JUSTIFICATION (.JU) is still enabled then the text will still be filled with blanks to fill the line.

This command creates a break. ".CO ON" is in effect until another ".CO OFF" control word or ".NC" control word alters it.

CONDITIONAL PAGE EJECT control word

.CP	n

The CONDITIONAL PAGE EJECT control word causes a page eject to occur if insufficient lines remain on the current page for text. The .CP control word will cause a page eject to occur if "n" lines do not remain on the current page.

This control word does cause a break. This control word is especially useful by preceding a section heading to insure that the heading will not be left alone at the bottom of the page.

DEFINE MACRO control word

The DEFINE MACRO control word defines a sequence of input lines to be invoked by ".name" as a user defined control word. The .DM control word is used to define a user macro, which is defined with a ".DM name" at the start and ".DM OFF" to terminate. Such user macros may be used for common sequences of control words and text.

The user macro is known by "name", a two-character identifier. The macro defines a sequence of control words and text lines that are invoked by a ".name" macro call. There are three system control words that can not be used in the ".DM name/OFF" sequence, that is, .DM (DEFINE MACRO), .FB (FLOATING BLOCK), and .FD (FLOATING DIAGRAM).

This control word does not create a break when defined. It does not create a break when called either, although the macro may contain control words which will cause a break.

DOUBLE SPACE control word

The DOUBLE SPACE control word causes a line to be skipped between each output text line. The following formatted lines of text will be double spaced. This is done by creating a conditional skip of one line after each text line. The use of a SPACE (.SP) control word when in double spacing will honour the SPACE instead of the DOUBLE SPACE.

This control word is not in effect unless encountered. It does cause a break when specified. The parameter of the SPACE (.SP), SKIP (.SK) and CONDITIONAL PAGE EJECT (.CP) are not doubled when .DS control word is in effect.

EVEN PAGE ADJUST control word

.EA	< n | +n | -n >

The EVEN PAGE ADJUST control word causes all output of the even numbered pages to be moved to the right of the physical left print margin. This control word is similar to the ".AD EVEN" control word, however, it just effects the output of the even numbered pages. An parameter of the form "+n" adds the value to the current even page adjustment value. An parameter of the form "-n" subtracts the value from the current even page adjustment value.

This control word will create a break when encountered. Unless otherwise specifies n = 0 is in effect.

EVEN PAGE BOTTOM TITLE control word

.EB	n /s1/s2/s3/

The EVEN PAGE BOTTOM TITLE control word is used to define three items of title information to be printed at the bottom of even numbered pages.

The value of "n", from one to the maximum value of the EVEN PAGE FOOTING SPACE (.FS EVEN), gives the footing line number and s1, s2, s3 are character strings not containing the delimiter character "/". The delimiter character can be any punctuation, defined as the first character of the parameter. Any of the fields may be omitted, but the delimiter character must be included to indicate missing fields.

The .EB control word is used in a way similar to the ".BT EVEN" control word. The title items defined with .EB control word will be printed in footing lines near the bottom of even numbered pages. The number of footing lines printed is set by .FS (FOOTING SPACE).

A break is not created by this control word. Unless otherwise specified ".EB ////" will be in effect.

EVEN PAGE EJECT control word

.EP	

The EVEN PAGE EJECT control word causes output to continue on an even numbered page. If output is at the top of an even numbered page then no action is performed. It is the short-hand way to specify the ".PA EVEN" control word.

This control word creates a break.

EVEN PAGE TOP TITLE control word

.ET	n /s1/s2/s3/

The EVEN PAGE TOP TITLE control word is used to define three items of the title information to be printed at the top of even numbered pages.

The value of "n", from one to the maximum value of the EVEN PAGE HEADING SPACE (.HS EVEN), gives the heading line number and s1, s2, s3 are character strings not containing the delimiter character "/". The delimiter character can be any punctuation, defined as the first character of the parameter. Any of the fields may be omitted, but the delimiter character must be included to indicate missing fields.

The .ET control word is used in a way similar to the ".TT EVEN" control word. The title items defined with .ET control word will be printed in heading lines near the top of even numbered pages. The number of heading lines printed is set by .HS (HEADING SPACE).

A break is not created by this control word. Unless otherwise specified ".ET ////" will be in effect.

FLOATING BLOCK control word

.FB	< ON | OFF | PUT >

The FLOATING BLOCK control word allows the user to enter a block of text that will print later in the document. The .FB control word defines the beginning and ending of a block of text that is to be printed later in the document. When the ".FB ON" control word is encountered, SCRIPT system prepares to accept the block of text. More than one floating block may be defined; each new block is added internally to the end of the current block. The ".FB PUT" control word is used to cause printing of the block after the "END" parameter is encountered. All blocks are accumulated to this point to be printed.

This control word does not create a break. A .DM (DEFINE MACRO), a .FB (FLOATING BLOCK), or a .FD (FLOATING DIAGRAM) control word is not allowed in the .FB ON/OFF sequence.

FLOATING DIAGRAM control word

.FD	< ON | OFF >

The FLOATING DIAGRAM control word defines a block of input text lines that are treated as a single block. Only text lines are allowed in the ".FD ON/OFF" sequence, and any control word is not allowed to occur. The text lines in this sequence are printed as they are, that is, one line in, one line out. All the text lines are printed in the single space mode, and the blank lines of the line spacing will be printed after all text lines are printed. This control word also has the function of .CP control word. The text lines in the sequence will be printed together. If there is no enough space to print all text lines in the current page, then the text lines will be printed at the top of the next page.

This control word does create a break.

FILL control word

```
┌─────────┬──────────────────────────────────────────────┐
│         │                                              │
│   .FI   │                                              │
│         │                                              │
└─────────┴──────────────────────────────────────────────┘
```

The FILL control word is the short-hand of ".FO ON" control word. This control word specifies that output lines are to be formed by shifting words to or from the next line and padded with extra blanks to produce an even right margin.

This control word creates a break when encountered.

FOOTING MARGIN control word

```
┌─────────┬──────────────────────────────────────────────┐
│         │                                              │
│   .FM   │   < ALL | EVEN | ODD > < n | +n | -n >       │
│         │                                              │
└─────────┴──────────────────────────────────────────────┘
```

The FOOTING MARGIN control word specifies the number of blank lines which are to left between the bottom of formatted text and any footing line.

ALL effects the footing margin settings of even and odd numbered pages.
EVEN effects the footing margin setting of even numbered pages only.
ODD effects the footing margin setting of odd numbered pages only.

The .FM control word defines the footing margin, which is the number of blank lines which will be left between the bottom of formatted text and the first footing line on the even and odd numbered pages. The first footing line is in the top of the FOOTING SPACE area. An parameter of the form "+n" or "-n" adds this value algebraically to the current footing margin setting, so long as the resulting value is not negative.

This control word does not create a break when encountered and until then n = 1 will be in effect. The FOOTING MARGIN plus FOOTING SPACE must always be less than or equal to the BOTTOM MARGIN.

FORMAT control word

```
┌─────────┬──────────────────────────────────────────────┐
│         │                                              │
│   .FO   │   < ON | OFF >                               │
│         │                                              │
└─────────┴──────────────────────────────────────────────┘
```

The FORMAT control word combines the effect of CONCATENATE and JUSTIFY. The FORMAT control word is a short-hand way to specify CONCATENATE and JUSTIFY. This control word specifies that output lines are to be formed by shifting words to or from the next line (concatenate) and padded with extra blanks to produce an even right margin (justify). The "OFF" parameter is equivalent to the control words CONCATENATE OFF and JUSTIFY OFF.

This control word creates a break. ".FO ON" is in effect unless a ".FO OFF" is encountered.

FOOTING SPACE control word

```
┌─────────┬──────────────────────────────────────────────┐
│         │                                              │
│   .FS   │   < ALL | EVEN | ODD > < n | +n | -n >       │
│         │                                              │
└─────────┴──────────────────────────────────────────────┘
```

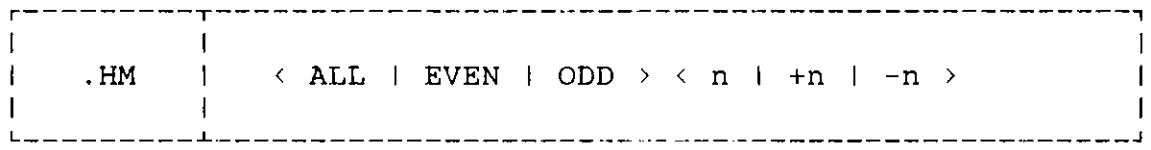
The FOOTING SPACE control word specifies the number of footing lines to be printed at the bottom of both even and odd numbered pages.

ALL effects the footing space values of even and odd numbered pages.
EVEN effects the footing space value of even numbered pages only.
ODD effects the footing space value of odd numbered pages only.

The .FS control word controls the number of footing lines to be printed at the bottom of a page. Up to thirty footings may be defined and printed at the bottom of each page. An parameter of the form "+n" or "-n" adds this value algebraically to the current footing space setting, so long as the resulting value is not negative.

This control word does not create a break when encountered and until then n = 1 will be in effect. The value of the parameter may range from zero to thirty. The FOOTING MARGIN plus FOOTING SPACE must always be less than or equal to the BOTTOM MARGIN.

HEADING MARGIN control word



The HEADING MARGIN control word specifies the number of blank lines which are to be left between the HEADING SPACE area and the first line of the text area.

ALL effects the heading margin values of even and odd numbered pages.
EVEN effects the heading margin value of even numbered pages only.
ODD effects the heading margin value of odd numbered pages only.

The last heading line produced on subsequent output pages will be separated from the first line of text by "n" blank lines. An parameter of the form "+n" or "-n" adds this value algebraically to the current heading margin setting, so long as the resulting value is not negative.

This control word does not create a break until it is encountered n = 1 will be in effect. The HEADING MARGIN plus HEADING SPACE must be less than or equal to the TOP MARGIN.

HEADING SPACE control word

```
┌---------┬-------------------------------------------------┐
|         |                                                 |
|   .HS   |   < ALL | EVEN | ODD > < n | +n | -n >         |
|         |                                                 |
└---------┴-------------------------------------------------┘
```

The HEADING SPACE control word specifies the number of title lines to be printed at the top of both even and odd numbered pages.

ALL effects the heading space values of even and odd numbered pages.
EVEN effects the heading space value of even numbered pages only.
ODD effects the heading space value of odd numbered pages only.

This control word controls the number of heading lines to be printed at the top of a page. Up to thirty headings may be defined and printed at the top of each page. An parameter of the form "+n" or "-n" adds this value algebraically to the current heading space setting, so long as the resulting value is not negative.

This control word does not create a break and until it is encountered n = 1 will be in effect. The value of the parameter may range from zero to thirty. The HEADING MARGIN plus HEADING SPACE must be less than or equal to the TOP MARGIN.

INDENT control word

.IN	< n | +n | -n >

The INDENT control word causes the logical left margin of the printout to be indented a specified number of spaces. The .IN control word causes the output to be indented the value "n" from the left margin. This indentation remains in effect for all subsequent lines until another .IN control word is encountered. An parameter of the form "+n" or "-n" adds this value algebraically to the current indent setting.

When this control word is encountered, it creates a break. The initial value for the indent is zero.

INDENT RIGHT control word

.IR	< n | +n | -n >

The INDENT RIGHT control word causes the logical right margin of the output to be indented. The .IR control word causes formatted output to be indented from the right margin. This indentation remains in effect for all subsequent output lines until another .IR control word is encountered. An parameter of the form "+n" or "-n" adds this value algebraically to the current indent right setting, so long as the resulting value is not negative.

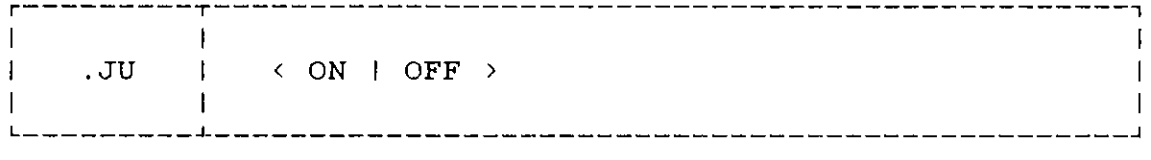
This control word will cause a break. The initial value of indent right setting is zero.

INDEX control word

The INDEX control word builds the index structure. The next input text record will be put into the index structure as an index entry, and this input text record is also used to create the formatted text line. The index structure will be printed in alphabetic order when the PRINT INDEX control word (.PI) is encountered. For purpose of alphabetizing the index entries, all indexes are treated as lowercase.

This control word does not create a break.

JUSTIFY control word



The JUSTIFY control word causes output lines to be padded with extra blanks so that the right margin is justified. The .JU control word specifies that all subsequent output lines are to be formed by padding with extra blanks to cause the right margin to be justified. If a line exceeds the current line length, and CONCATENATE OFF is in effect, the line is printed as it is.

This control word creates a break. ".JU ON" is in effect until a ".JU OFF" or a ".NJ" is encountered.

LINE LENGTH control word

.LL	< n | +n | -n >

The LINE LENGTH control word specifies the number of horizontal character positions which are to be printed in subsequent output lines. The .LL control word sets the length of subsequent output lines to width "n". An parameter of the form "+n" or "-n" adds this value algebraically to the current line length, so long as the resulting value is not zero.

When this control word is encountered it creates a break. Unless otherwise specified "n = 60" characters per line (including blanks) will be in effect.

LINE SPACING control word

.LS	< n | +n | -n >

The LINE SPACING control word causes zero or more blank lines to be skipped between each line of formatted output. Subsequent output lines will have "n" blank lines inserted after each line of body text. These blank lines are produced as conditional spaces. If a SPACE ".SP" control word is used to create blank lines between text then the maximum of the SPACE value and the LINE SPACING value will be used.

This control word creates a break. Single spacing ".LS 0" is the normal mode.

NO CONCATENATE control word

.NC	

The NO CONCATENATE control word stops words from shifting to or from the next line. There will be a one-to-one correspondence between the words in input and output lines. If JUSTIFY ON is in effect, the right margin will still be adjusted. This control word is a short-hand way of ".CO OFF" control word.

This control word does create a break and is not in effect until encountered.

NO FILL control word

.NF	

The NO FILL control word causes lines to be output as they appear in the input by negating the effect of JUSTIFY and CONCATENATE. The .NF control word is a short-hand way to specify .NC and .NJ. Subsequent output lines will be printed exactly as they appear in the input until a .FO, .JU, .FI, or .CO control word is encountered.

This control word create a break and is not in effect until encountered.

NO JUSTIFY control word

.NJ	

The NO JUSTIFY control word stops justification of text to the right margin. It is the short-hand way to specify the ".JU OFF" control word. The .NJ control word causes SCRIPT system, to cease inserting extra blanks into lines in order to justify the right margin. If CONCATENATE is in effect, the output lines will be as long as possible without exceeding the current line length and without breaking words in the middle.

This control word does create a break and is not in effect until encountered.

ODD PAGE ADJUST control word

.OA	< n | +n | -n >

The ODD PAGE ADJUST control word causes all output of the odd numbered pages to be moved to the right of the physical left print margin. This control word only effects the output of the odd numbered page, and remains in effect for all the following input lines. An parameter of the form "+n" adds the value to the current odd page adjustment value. An parameter of the form "-n" subtracts the value from the current odd page adjustment value.

This control word creates a break when encountered.

ODD PAGE BOTTOM TITLE control word

.OB	n /sl/s2/s3/

The ODD PAGE BOTTOM TITLE control word is used to define three items of title information to be printed at the bottom of odd numbered pages.

The value of "n", from one to the maximum value of the FOOTING SPACE (.FS), gives the footing line number and sl, s2, s3 are character strings not containing the delimiter character "/". The delimiter character can be any punctuation, defined as the first character of the parameter. Any of the fields may be omitted, but the delimiter character must be included to indicate missing fields.

The .OB control word is used in a way similar to the .BT control word. The title items defined with .OB control word will be printed in footing lines near the bottom of odd numbered pages. The number of footing lines printed is set by .FS (FOOTING SPACE).

A break is not created by this control word. Unless otherwise specified ".OB ////" will be in effect.

ODD PAGE EJECT control word

.OP	

The ODD PAGE EJECT control word causes output to continue on an odd numbered page. The .OP control word causes the formatted output to continue at the top of the next odd numbered page. If output is at the top of an odd numbered page then no action is performed.

This control word creates a break.

ODD PAGE TOP TITLE control word

.OT	n /s1/s2/s3/

The ODD PAGE TOP TITLE control word is used to define three items of title information to be printed at the top of odd numbered pages.

The value of "n", from one to the maximum value of the HEADING SPACE (.HS), gives the heading line number and s1, s2, s3 are character strings not containing the delimiter character "/". The delimiter character can be any punctuation, defined as the first character of the parameter. Any of the fields may be omitted, but the delimiter character must be included to indicate missing fields.

The .OT control word is used in a way similar to the .TT control word. The title items defined with .OT control word will be printed in heading lines near the top of odd numbered pages. The number of heading lines printed is set by .HS (HEADING SPACE).

A break is not created by this control word. Unless otherwise specified ".OT ////" will be in effect.

PAGE EJECT control word

.PA	< %+1 | n | +n | -n | EVEN | ODD >

The PAGE EJECT control word causes an output page eject. When the .PA control word is encountered, the rest of the current page is skipped, the footing space lines are printed, and a new page is begun whose number is specified by the parameter.

The "EVEN" parameter causes the formatted output to continue at the top of the next even numbered page, or no action if output is at the top of an even numbered page. Similarly, "ODD" parameter causes the output to continue on the next odd page. An parameter of the form "n" will change the current page number to the value specified; "+n" will increment the current page number by "n"; "-n" will decrement the current page number by "n".

This control word does create a break when encountered. If the parameter is omitted then the current page number plus one is assumed.

PRINT INDEX control word

.PI	

The PRINT INDEX control word causes the index structure to be printed. The index entries of the index structure are printed in alphabetic order. This control word should be used in the end of the script file. Otherwise, the index printed in the middle of the document may be not the complete index table that you want. The index entries of the index structure are not destroyed after printing. You may add another index entries to the index structure after you print the index.

This control word creates a break when encountered.

PAGE LENGTH control word

.PL	< n | +n | -n >

The PAGE LENGTH control word specifies the physical size of the output page in units of typewriter lines. The .PL control word allows the use of various paper sizes for output, by setting the length of subsequent output pages to "n". An parameter of the form "+n" or "-n" adds this value algebraically to the current page length, so long as the resulting value is greater than .TM plus .BM.

This control word will create a break and unless otherwise specified n = 66 will be in effect.

PAGE NUMBER TYPE control word

.PN	< ARABIC | ROMAN >

The PAGE NUMBER TYPE control word allows the user to control the printing of page numbers.

ARABIC causes page numbers produced in headings and footings to be printed in arabic numerals.

ROMAN causes page numbers produced in headings and footings to be printed in lowercase or uppercase roman numerals. You may use ROMAN control word (.RO) to specify that you want to use lowercase or uppercase roman numerals.

This control word will not create a break. Unless otherwise specified ".PN ARABIC" will be in effect. If the page number is less than 1 or greater than 999, then the ".PN ROMAN" is not used. The page numbers will be printed in arabic numerals.

PAGE NUMBER SYMBOL control word

.PS	character

The PAGE NUMBER SYMBOL control word defines the character in headings and footings to be replaced by the current page number. The percent sign "%" is usually reserved for use as the page number symbol. SCRIPT system substitutes the current page number for each occurrence of the percent sign within a top or bottom title line.

To allow the percent sign to appear in titles the .PS control word is supported to change the character to be replaced by the current page number. The character specified in the PAGE NUMBER SYMBOL control word takes effect for all subsequent headings and footings. This means heading and footing specifications may have to be changed at the same time.

This control word does not create a break when encountered. The initial default page number symbol character is the percent sign "%". Any punctuation may be specified as an parameter.

PRINT TABLE OF CONTENTS control word

.PT	

The PRINT TABLE OF CONTENTS control word causes the table of contents to be printed out. This control word should be used in the very last of the script file, or you may obtain an incomplete table of contents. The entries in the table of contents are not destroyed after printing. You may add more entries into the table of contents after it is printed.

This control word does create a break when encountered.

RIGHT ADJUST control word

.RI	< 1 | n | ON | OFF >

The RIGHT ADJUST control word causes the next "n" input line to be right adjusted in the output line.

n specifies that the next "n" input text records are to be right adjusted. If "n" is omitted, a value of 1 is assumed to right adjust the next input record. If the value of "n" is zero, then it is the same as the "OFF".

ON specifies that all following input text records are to be right adjusted.

OFF terminates the right adjustment after an "ON" was specified. If "n" was given and has not yet been exhausted, an "OFF" parameter will terminate right adjustment also.

The next "n" input lines, only including input text lines, will be right adjusted in the output lines. If the next line is longer than the current line length, it will not be right adjusted.

This control word will create a break when encountered. A numeric parameter will right adjust the following "n" input text lines. An "ON" parameter will right adjust all following input lines until an "OFF" parameter, a "0" parameter or a CENTER control word is encountered.

ROMAN control word

.RO	< BIG | SMALL >

The ROMAN control word causes page numbers to use uppercase or lowercase roman numerals, if the page numbers were produced using roman numerals. If the page numbers were produced in arabic numerals, then this control word will no effect at this moment. But when the page number type changes to roman numerals, then the parameter of this control word is in effect.

This control word will not create a break and unless otherwise specified ".RO SMALL" will be in effect.

SKIP control word

.SK	< 1 | n >

The SKIP control word causes several blank lines to be printed out. This control word is similar to the SPACE control word, except that if the end of the page is reached before satisfying the request, the remaining space lines are also printed at the top of the next page.

This control word does create a break. If the parameter is omitted, then n = 1 is assumed to print one blank line. In spite of the line spacing setting, exact "n" blank lines are printed out.

SPACE control word

The SPACE control word generates a specified number of blank output lines. The SPACE control word causes "n" blank lines to be printed. If the end of the page is reached before satisfying the request or if output is at the top of a page, the remaining space lines are normally not output at the top of the next page.

This control word create a break when encountered and if the parameter is omitted n = 1 will be assumed.

SINGLE SPACE control word

The SINGLE SPACE control word causes output to be single spaced. It is the short-hand way to specify the ".LS 0" control word. All following formatted lines of text will be single spaced.

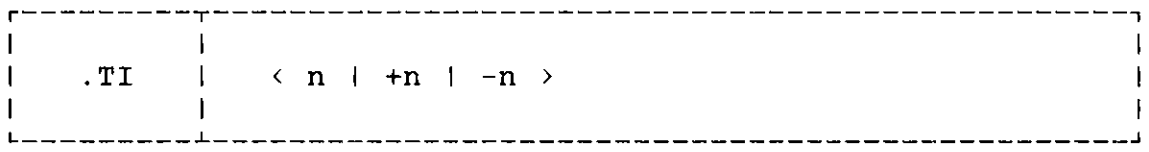
When this control word is encountered it will create a break. Unless otherwise specified single spacing will be in effect.

TABLE OF CONTENTS control word

The TABLE OF CONTENTS control word adds an input text line to the Table of Contents.

This control word does not create a break. The value of "n" is the level of the text stored in the Table of Contents. When the Table of Contents is printed by .PT (PUT TABLE OF CONTENTS) control word, the entries will indent 3 * (level - 1) blanks at the left margin if its level is not 1.

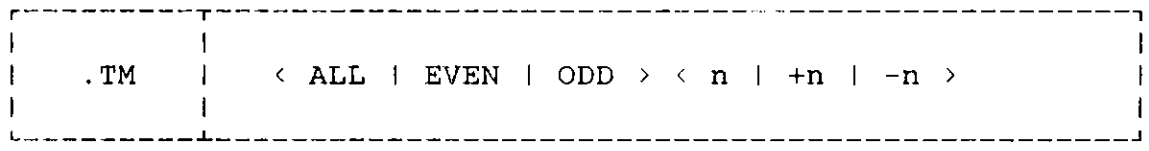
TEMPORARY INDENT control word



The TEMPORARY INDENT control word causes next input text record to be indented to the right of a specified column. This control word is similar to the .IN control word, but it only effects next input text record.

This control word does create a break when encountered.

TOP MARGIN control word



The TOP MARGIN control word specifies the number of lines which are to be placed between the physical top of the output page and the first line of the text area.

ALL effects the top margin values of even and odd numbered pages.
EVEN effects the top margin value of even numbered pages only.
ODD effects the top margin value of odd numbered pages only.

Subsequent output pages will begin with "n" lines (which may includes heading lines) before the first line of text. An parameter of the form "+n" or "-n" adds this value algebraically to the current value of the top margin. The TOP MARGIN must never be smaller than the sum of the HEADING MARGIN plus the HEADING SPACE.

This control word will not create a break and until encountered n = 6 will be in effect.

TOP TITLE control word

.TT	< ALL | EVEN | ODD > n /s1/s2/s3/

The TOP TITLE control word is used to define three items of title information to be printed at the top of both even and odd numbered pages.

ALL effects the top titles of even and odd numbered pages.
EVEN effects the top title of even numbered pages only.
ODD effects the top title of odd numbered pages only.

The value of "n", from one to the maximum value of the HEADING SPACE (.HS), gives the heading line number and s1, s2, s3 are character strings not containing the delimiter character "/". The delimiter character can be any punctuation, defined as the first character of the parameter. Any of the fields may be omitted, but the delimiter character must be included to indicate missing fields.

The .TT control word is used in a way similar to the .BT control word. The title items defined with .TT control word will be printed in heading lines near the top of even and odd numbered pages. The number of heading lines printed is set by .HS (HEADING SPACE). The s3 may overlay s2 if necessary, and s2 may overlay s1 if necessary.

This control word does not create a break. The default top title on each page is ".TT ////".

UNDERSCORE and CAPITALIZE control word

.UC	< 1 | n | ON | OFF >

The UNDERSCORE and CAPITALIZE control word underscores and capitalizes an input text line. The .UC control word applies the rules of UPPERCASE (.UP) and UNDERSCORE (.US) to subsequent text records. An "ON" parameter will cause all following input text records to be operated on until a "OFF" parameter or a "0" parameter is encountered. A numeric parameter will cause the specified number of input text records to be operated on.

This control word does not create a break. If no parameter is present, the next input text record will be underscored and capitalized.

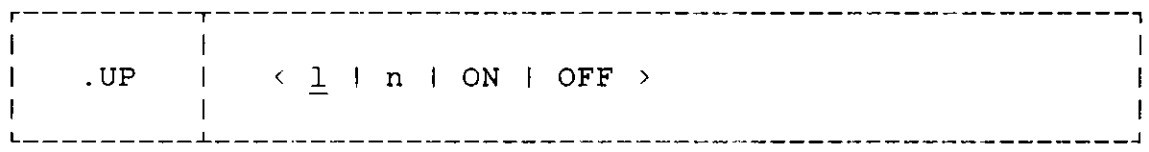
UNDENT control word

.UN	< n | +n | -n >

The UNDENT control word forces the next output line to start a specified number of columns to the left of the current indent. The .UN control word causes only the next output line to begin to the left of the current indent value. A relative change as in "+n" or "-n" changes the previously specified .IN control word and .TI control word. The result of undentation may not exceed the current indention.

This control word create a break when encountered.

UPPERCASE control word



The purpose of the UPPERCASE control word is to capitalize an input line.

n specifies that the next "n" input text records are to be capitalized. If "n" is omitted, a value of 1 is assumed to capitalize the next input text record. If the value of "n" is zero, then it is the same as the "OFF".

ON specifies that all following input text records are to be capitalized.

OFF terminates the capitalization after an "ON" was specified. If "n" was given and has not yet been exhausted, an "OFF" parameter will terminate capitalization also.

The .UP control word converts each lowercase alphabetic of next "n" input text lines. The UPPERCASE control word operates independently of other control words that modify text. When more than one of ".BD", ".BI", ".UC", ".US", ".UP" are in effect, the result is the best equivalent of the sum of the effects. Each must be disabled in any order to cancel them all.

This control word does not create a break. If no parameter is present, the next input text record will be converted to uppercase.

UNDERSCORE control word

The UNDERSCORE control word is used to underscore an input text line.

n specifies that the next "n" input text records are to be underscored. If "n" is omitted, a value of 1 is assumed to underscore the next input text record. If the value of "n" is zero, then it is the same as the "OFF".

ON specifies that all following input text records are to be underscored.

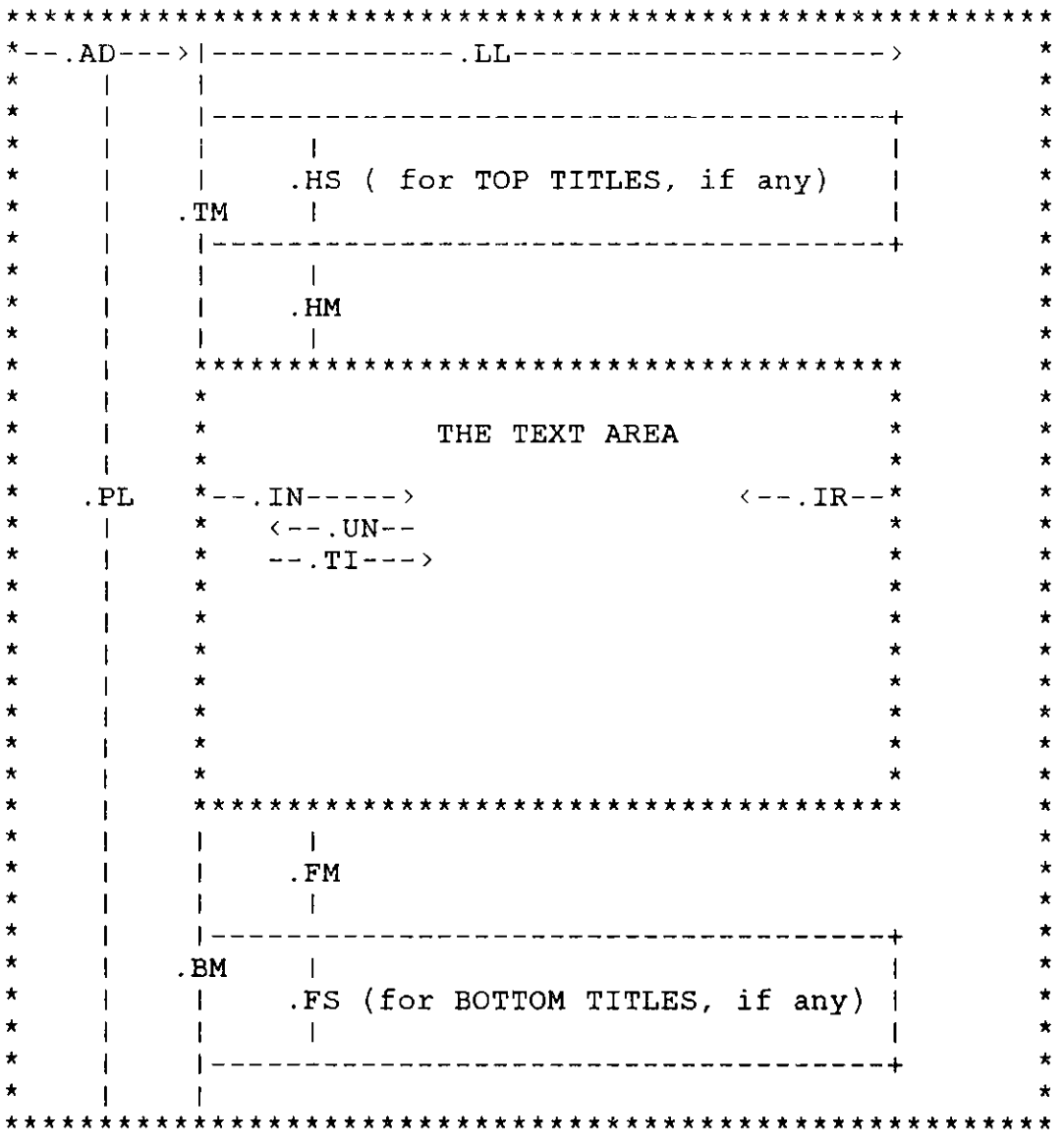
OFF terminates the underscoring after an "ON" was specified. If "n" was given and has not yet been exhausted, an "OFF" parameter will terminate underscoring also.

All the alphanumeric characters of the following input text records are underscored. The blanks and punctuations are normally not underscored.

This control word does not create a break. If no parameter is present, the next input text record will be underscored.

PAGE SETUP

The outer rectangle of *'s in the diagram below indicate the physical dimensions of a sheet of paper. The inner rectangle of *'s represent the "text area" (that part of the physical page into which SCRIPT system formats the input text).



The default values for the control words show above are as follows :

.AD 0	(ADjust)	.IN 0	(INdent)
.BM 6	(Bottom Margin)	.IR 0	(Indent Right)
.FM 1	(Footing Margin)	.LL 60	(Line Length)
.FS 1	(Footing Space)	.UN 0	(UNdent)
.HM 1	(Heading Margin)	.PL 66	(Page Length)
.HS 1	(Heading Space)	.TM 6	(Top Margin)
.TI 0	(Temporary Indent)		

END OF STF USER'S MANUAL

Appendix E

Bibliography

1. Waterloo SCRIPT - version 82.1 (82OCT18) Reference Manual, Department of Computing Services, University of Waterloo, Waterloo, Oritario, Canada N2L 3G1, 1982.

2. Waterloo SCRIPT User's Guide, Department of Computing Services, University of Waterloo, Waterloo, Oritario, Canada N2L 3G1, 1982.

3. MUSIC Script User's Manual, McGill University, Montreal, Ouebec, Canada, 1981.

4. VAX-11 PASCAL Language Reference Manual, Digital Equipment Corporation, October 1982.

5. VAX-11 PASCAL User's Guide, Digital Equipment Coaportion, October 1982.

6. Achugbue, J. O. "On the Line Breaking Problem in Text Formatting," in Proc. ACM SIGPLAN/SIGOA Conf. Text Manipulation (Portland, ore., June 8-10, 1981), ACM, New York, 1981, pp. 117-122.

7. Furuta, R., Scofield, J., and Shaw, A. "Document Formatting Systems : Survey, Concepts, and Issues," Computing Survey, 14, 3 (Sep. 1982).

8. Reid, B. K. "A High-level Approach to Computer Document Formatting," in Proc. 7th Annu. ACM Symp. Programming Languages (Jan. 1980), ACM, New York, 1980, pp. 24-30.

9. Chamberlin, D. D., Bertrand, O. P., Goodfollow, M. J., King, J. C., Slutz, D. R., and Todd, S. J. P. "JANUS : An Interactive Document Formatter Based on Declarative Tags," IBM System Journal, 21:3, 250-271, 1982.

10. Chamberlin, D. D., King, J. C., Slutz, D. R., and Todd, S. J. P. "JANUS : An Interactive System for Document Composition," in Proc. ACM SIGPLAN/SIGOA Conf. Text Manipulation (Portland, Ore., June 8-10, 1981), ACM, New York, 1981, pp. 82-91.

11. Samet, H. "Heuristics for the Line Division Problem in Computer Justified Text," Comm. of ACM, 25, 564-571, Aug. 1982.

12. Mouney, J. D. "MFS : A Modular Text Formatting System," National Computer Conference Proceedings, AFIPS Press.

13. Talmage, L. A. "Design and Implementation of an Integrated Screen-Oriented Text Editing and Formatting System," Noval Postgraduate School, June 1980.

14. Noot, H. "Structured Text Formatting," Software Practice and Experience, 13:1, 79-94, Jan. 1983.

15. Knuth, D. E., and Plass, M. F. "Breaking Paragraphs into Lines," Software Practice and Experience, 11:11, 1119-1184, Nov. 1981.

16. Hanson, H., and Steensgaard-Madson, J. "Document Preparation Systems," Software Practice and Experience, 11:9, 983-997, Sep. 1981.

17. Kernighan, B. W., and Plauger, P. J. Software Tools in PASCAL, Addison-Wesley Publishing Company, 1981, pp. 227-264.

18. RUNOFF User's Manual, Digital Equipment Corporation, October 1982.