HONEYWELL

GUIDE TO MULTICS WORDPRO FOR NEW USERS

SOFTWARE

GUIDE TO MULTICS WORDPRO FOR NEW USERS

SUBJECT

Description of the Multics Word Processing System (WORDPRO) for New Users of 68, 68/DPS, and DPS 8 Large Information Systems Under Control of the Multics Software Environment, Including Discussions with Examples of Most Facilities and Commands Required to Format a Document

SPECIAL INSTRUCTIONS

This manual does not require any previous knowledge of the Multics system or experience in using the Multics Word Processing System. While this manual does not discuss all facets of WORDPRO, it does, however, explain a sufficient number of commands and control lines to help a novice use WORDPRO to edit and format a document. See the Preface for a listing of reference documents.

This revision supersedes Revision 0 of the manual, dated September 1979. A new Section 6 has been added and all following sections renumbered with information formerly in Section 11 rewritten and placed in Sections 12 and 13. Because the manual has been extensively revised, change indicators have not been used.

SOFTWARE SUPPORTED

Multics Software Release 9.0 and 10.0.

ORDER NUMBER

DJ18-01

September 1982

Honeywell

Preface

This manual describes how to use the Multics Word Processing System, called WORDPRO. The Multics system is a general purpose computer operating system developed to serve a wide variety of user tasks. Jobs for the central computer are shared among many users, each of whom may use the Multics operating system from a data (computer) terminal.

WORDPRO consists of various Multics commands (typed computer instructions), which perform word processing tasks and assist users in the input, updating, and maintenance of computer-stored documents. Applications range from formatting simple form letters to creating complex technical manuals. Additionally, WORDPRO provides Multics security, ease of use, and other document management tools.

A user new to a data terminal, the Multics system, or WORDPRO should begin the manual with Section 1 and read the discussions and perform the examples in the order they are given. Users familiar with log-in and log-out procedures of the Multics system may, of course, omit Section 1 and begin with Section 2.

Although text editors, qedx and TED, as well as the Electronic Mail facility described and used herein are not parts of the Multics WORDPRO package, they are incorporated for completeness and to assist the user new to the Multics system.

Note that this manual presumes that the display is a hardcopy printer. Displays of certain visual display units may show responses different from those described in this manual and may require different typing sequences.

Supplemental reference information may be found in the following documents:

Order Number	*Title*
CH24	*New Users' Introduction to Multics – Part I*
CH25	*New Users' Introduction to Multics – Part II*
AZ98	*Multics WORDPRO Reference Guide*
CG40	*qedx Text Editor Users' Guide*
CP50	*TED Text Editor Reference Guide*

File No.: 1L13 DJ18-01

Contents

Section 1
Logging in and Logging out

A computer is a tool. Used correctly, it can perform exactly as you instruct it. Mis-instructed, it will not perform, and in many cases, it will reply with an error message informing you why the instruction cannot be carried out.

Instructions (which are called commands in the Multics system) are made to the computer by pressing keys on the typewriter-like keyboard of a data (computer) terminal. You may never see the computer, but you will have the feeling that only you are using it.

> **Do not be afraid of making typing errors or giving erroneous commands.**

YOUR IDENTIFICATION

To use the computer for word processing, you must first gain access to the computer. That procedure is called **logging in**. But, before you can **log in**, you must be identified to the computer by a personal identification and a project identification. Furthermore, you must also have a password that the computer can use to authenticate your personal identification before it will grant you use of the system.

Your personal identification will generally consist of your last name or some slight modification of it. For example, if your name is Tony or Toni Smith, your personal identification (usually shown in reference literature as **Person_id**) may be "Smith", "TSmith", "SmithT", "Smithto", etc. Your Person_id will not contain any blank spaces and must be entered exactly as given with particular attention to capital and lowercase letters.

NOTE

In Multics literature, where blank spaces would usually occur in plain text to separate words or characters, an underline is used to show the word separation and yet indicate that blank spaces are not permitted. When commands containing underlines occur and must be typed, they must be typed exactly as shown with the underline character.

Your Person_id is assigned by the computer system operations office which may also assign your project identification, **Project_id.** The project identification, likewise, does not contain any blank spaces and generally consists of an acronym of the actual project to which the computer time costs are applied. In the examples in this manual, your Person_id will be shown as "TSmith" and your Project_id will be shown as "ProjA". A reference made to a **User_id** refers to the Person_id and Project_id pair connected by a period: Person_id.Project_id as in "TSmith.ProjA".

The computer system operations office also assigns an initial password for you. This password is required by the computer to authenticate your Person_id although your Person_id is unique and no one else registered on the system has it. In many cases, this initial password will consist of lowercase initials of your name, such as "ts", again without blank spaces.

Because the password is used to verify that you are whom you claim to be and that you have the right to use the system, you have the option of changing that password any time you log in. Changing the password to one that only you know helps prevent anyone, including someone at the computer operations office, from logging in by pretending to be you. If you then forget your password, you must contact the operations office and request a new one (which you again, of course, can readily change).

TERMINAL CONNECTION

With few exceptions (where a data terminal may be wired directly to the computer), the connection of the terminal to the computer is made through a telephone line. The terminal itself consists of three parts, which may or may not be contained as a single unit. These three parts are the terminal keyboard, the display, which is either a hardcopy printer or a visual display unit (VDU), and a modem, which connects the terminal and display to the computer via the telephone.

NOTE

This manual presumes that the display is a hardcopy printer. Certain VDU displays may show responses (character strings, patterns, etc.) different from those shown here and may need a few slightly different typing sequences.

The terminal keyboard resembles the keyboard of a standard typewriter. It differs, however, by having extra keys for special characters such as brackets, braces, and backslash. To enter either a character of data to be processed or a command to the computer, simply type the character as you would type normally. In most cases, the character will be immediately displayed for you on the printer. However, because the character may first be transmitted to the computer and then echoed back to the terminal, there may be a brief delay before the character is displayed. If a delay occurs, continue typing; you may enter several characters before they are printed on the display. The character echo delay is usually only a fraction of a second.

The printer displays both the input you type and the replies from the computer.

The modem provides the connecting link between your terminal and the computer. It may contain the acoustic coupler with the cradle that holds the telephone receiver. It may be built in as part of the terminal or it may be a separate unit and cabled to the terminal.

NOTE

Because there are differences between units of the various manufacturers, always refer to the operator's manual for your terminal for the settings of special switches.

To connect your terminal to the computer via the telephone, you must follow the steps below.

1. Set the power switch of the terminal to ON. If the modem is a separate unit, it, too, may have a power switch to be set to ON.

   NOTE

   Check the terminal and modem for switches marked in such a way as to suggest "FULL" or "HALF" duplex operation. This switch on the modem should always be set to "FULL" in order to give a feel for the terminal that more closely resembles the feel of a standard typewriter as you press the keys and hear them strike.

2. Pick up the telephone and dial the phone number of your computer system. Telephone numbers are available from the computer system operations office.
3. When the telephone connection to the computer is made, you will hear a continuous shrill tone in the telephone receiver. When you hear this tone, place the receiver firmly in the cups of the acoustic coupler, making sure that the telephone speaker and transmitter are not reversed from their intended cup positions. Many couplers say "cord" or show a picture for correct telephone connection. Some terminals and modems also show a light to indicate when connection to the computer has been made.

4. If the computer does not respond within a few seconds, then press the **LF** key (line feed) on the terminal keyboard.

NOTE

Some terminals, because of keyboard differences, may not have an **LF** key. On these terminals, press the **CR** (carriage return) or **RETURN** key. If there is no response from the computer (and it is known to be running), press the **CTL** (control) and **J** keys simultaneously. If there is still no response, seek local assistance.

5. Your printer will respond shortly with a message similar to the following:

```
Multics MR9.1: Honeywell LISD Phoenix, System M
Load = 130.0 out of 180.0 units: users = 139, 07/25/99 0822.5 mst Wed
```

This particular message identifies the system and gives the location of the system, the actual number of users logged in, and the number of users the system is currently accepting. You are now ready to log in.

LOG-IN PROCEDURE

The completion of the message from the Multics system is the signal from the computer for you to log in. The first step of the log-in procedure is to type on the terminal keyboard the lowercase letters "login", a space, and then your Person_id. Most users generally type the short form for the letters login, which is simply the letter "l". When this line has been typed, you must then press the carriage return, which moves the typing mechanism to the first column of the next line (called a **newline** in the Multics system). Generally, the carriage return is the **CR** or **RETURN** key on the terminal keyboard, but it may be different depending upon the terminal type or manufacturer. **The computer will not accept any line of input until the carriage return is pressed.**

NOTE

Strictly speaking, the Multics newline is the pair of characters **CR** and **LF**. The Multics system accepts only the **LF** character as the end-of-line signal. You may have to experiment to determine which of the **CR, LF,** or **RETURN** keys actually transmits the **LF** character to the computer for the terminal you are using.

You are now ready to log in. Type "l TSmith", substituting your Person_id for "TSmith". Then press the key for the carriage return.

```
    Multics MR9.1: Honeywell LISD Phoenix, System M
    Load = 130.0 out of 180.0 units: users = 139, 07/25/99 0822.5 mst Wed
  ! l TSmith
    Password:
    ############
```

NOTE

In the examples in this manual, all user input is shown preceded by an exclamation point to distinguish it from output lines from the computer. **Do not type the exclamation point!** Remember to follow all your typed lines with a carriage return. Also, most examples shown here are preceded by the previous line printed at the terminal in order to indicate continuity.

Note that after the carriage return for your Person_id input, the computer immediately responds with the word "Password:". This response may then be followed by a string of cover-up characters typed by the system. It is on top of this cover-up mask that you now type your password. *On most systems, you will not see the string of cover-up characters, and, instead, your typed password will simply not activate the printing mechanism.* In either case, you will not be able to read your password as you type it at the terminal. In the examples here, the cover-up mask indicates where you are to type your password.

Now type your password and activate the carriage return. In other words, if your password is "ts", type "ts".

```
                                   Password:
                                 ! #############
```

Once you have successfully entered your password and the computer has accepted it with your Person_id, the computer will respond with information regarding the last time you logged in, as well as a message of the day. The message of the day tells you, as well as other users of the system, any important news or information pertinent to you or to your use of the system, such as schedules. At the end of the message of the day, the system will send you a **ready message.** This "ready" message consists of the lowercase letter "r" followed by a set of numbers. The first number after "r" is the time of day. The other numbers indicate certain usage statistics and are important only in advanced programming procedures. The ready message is the system's way of telling you that you are at **command level** and that the system is ready to accept your next command.

```
#############
You are protected from preemption until 0922.
TSmith ProjA logged in 07/25/99  0822.9 mst Wed from ASCII terminal "none".
Last login 07/19/99 1415.8 mst Thu from ASCII terminal "none".

99-07-23  Do a "help sked" for schedule July 23 - 29

   Console phones are AAA-XXXX & AAA-YYYY (AC-BBB)

          Computer Operations (weekends) AAA-ZZZZ
r 8:23 1.165 18
```

NOTE

Under certain peak load conditions, the first line of the above message will be replaced with "You are subject to preemption." The messages serve only as estimates of the "guaranteed" length of your log-in session and have no effect on operation or any user procedures.

LOG-IN ERRORS

Under certain circumstances, you may be denied access to the computer although you have logged in correctly. For example, the system operations office may not yet have registered you on the system. In that case, you will receive a response as follows:

```
  Multics MR9.0: Honeywell LISD Phoenix, System M
  Load = 130.0 out of 180.0 units: users = 139, 07/25/99 0822.5 mst Wed
! 1 TSmith
  Password:
! ############
  The user name you supplied is not registered.
  Please try again or type "help" for instructions.
```

Other reasons you may be denied access are that you may have exceeded the resource limits set by the administrator of the project to which you are assigned, or the system may be temporarily unable to accept any additional user category (called load group limit). In any case, you will generally be given a message telling you why access is denied.

The most common reason for being denied access to the system is incorrect log in. For example, suppose you try to log in as "lTSmith". In that case, you will be denied access because a space is needed between the "l" and the Person_id.

```
! 1TSmith
  Incorrect login word "1TSmith".
  Please try again or type "help" for instructions.
```

Typing your password incorrectly will produce a response from the Multics system telling you to try again or to type "help". In this case, you must type again both the log-in line ("l TSmith") and the password line.

```
! l TSmith
  Password:
! ############
  Incorrect password supplied.
  Please try again or type "help" for instructions.
```

The system operations office at each computer installation sets a limit to the number of times you can attempt to log in at one sitting. If, for example, the limit is set at five tries, the fifth attempt will cause the Multics system to tell you to hang up.

```
! l TSmith
  Password
! ############
  Incorrect password supplied.
  hangup
```

CORRECTING TYPING ERRORS

Errors that cause the system to deny you access usually result because of typing errors in the log-in procedure. Although you cannot backspace to erase a typing error, typing errors can be corrected. However, the typing errors must be corrected before the carriage return at the end of the line containing the mistake.

Either an individual letter or the entire line may be corrected. To correct an individual mistyped character, type "#" immediately after the mistake and follow that immediately with the corrected character. To correct an entire line, type "@" immediately after the line, and *on the same line*, type the corrected wording from the beginning of the line. As examples, note the following which the Multics system recognizes as corrected log-in lines:

```
! l TA#Smith

! l TSmu#ith

! lT# TSmith

! l TAmith@l TSmith

! l TSM@L#l TSmi@l TSmith
```

The character erase symbol "#" can be used any number of times in a line to correct a mistyped word. It is used to erase the character immediately preceding it. However, only one erase symbol is needed to delete *all* white space (any combination of blank spaces and/or horizontal tabulations) preceding it. In the following example, the first "#" erases the "T" and the second "#" erases all three spaces preceding the "T":

```
! l   T## TSmith
```

One final point about the erase symbol (#) and the kill symbol (@). These symbols may be used not only when logging in but also while using a text editor or at any time a typing error is made while you are working at the data terminal. (See Section 4 for more discussion of the erase and kill symbols.)

LOG-OUT PROCEDURE

You may **log out** to discontinue using your terminal and hang up the telephone receiver only after a ready message. The log-out procedure uses a Multics command. **No Multics command can be invoked until a ready message is received.** Furthermore, you should always follow the log-out procedure before leaving the terminal in order to avoid wasting computer time and possibly preventing others from logging in (unless, of course, the telephone connection is unintentionally broken or the system itself fails and you cannot give the log-out command).

To log out, you need only type the word "logout" after the ready message. The system will respond by printing your identification with the time and date of your log-out command. Computer usage information will then be given. Finally, the word "hangup" will be printed. This response tells you to break the connection by hanging up the telephone.

```
  r  08:28 0.596 44
! logout
  TSmith ProjA logged out 07/25/99 0828.2 mst Wed
  CPU usage 2 sec, memory usage 0.3 units.
  hangup
```

BRIEF LOG-IN

In many cases, you will use a terminal several times a day and will not want to receive the message of the day each time you log in. You will not receive the message of the day if the log-in command line contains an ending "–brief" or its short form "–bf". The hyphen with the letters "bf" is a **control argument**. Most Multics commands have several optional control arguments that are used to modify some aspect of the command. A control argument is separated on a command line from a command by spaces. In this case, "–bf" is a control argument that lets you log in but at the same time suppresses the message of the day.

To log in using the "–bf" control argument, you would access the system as follows:

```
  Multics MR9.1: Honeywell LISD Phoenix, System M
  Load = 132.5 out of 180.0 units: users = 141, 07/25/99 0830.9 mst Wed
! l TSmith -bf
  Password:
! [redacted]
  r  08:31 0.203 14
```

LOG-OUT HOLD

Occasionally, someone will want to use your terminal immediately upon your completion. Thus far, you have been told that you would need to log out and hang up the telephone. The second user would then need to re-establish a telephone connection with the computer before that person could log in. The Multics system will, however, recognize a control argument for the log-out command that eliminates reconnecting the telephone. That control argument is "-hold" or its short form "-hd". It is typed on the log-out command line.

You will observe that the Multics response to the "-hold" log-out command does not include the word "hang-up". Instead, the response continues with the message usually obtained for any log-in command, and the new user can immediately give his or her Person_id and password.

```
  r  08:32 0.574 33

! logout -hd
  TSmith ProjA logged out 07/25/99 0832.4 mst Wed
  CPU usage 1 sec, memory usage 3.2 units.

  Multics MR9.1: Honeywell LISD Phoenix, System M
  Load = 126.5 out of 180.0 units: users = 135, 07/25/99 0833.0 mst Wed
! l Jones
  Password:
! ############
```

CHANGING YOUR PASSWORD

One of the first things you should do when registered on the system is to change your password. The procedure is relatively simple and is aided by the computer. Furthermore, you can change your password any time you log in.

Suppose, for example, that you want to change your password from "ts" to include your middle initial as in "tzs". The first step is to log in with your Person_id, but you must follow your Person_id with a space and the control argument "–change_password" or its short form "–cpw". The response, as usual, will be for you to give your present password. Thus, type your old password "ts" on the cover-up mask in order to gain access to the system.

You will now be requested to type your new password. Type "tzs" on a second cover-up mask. Next, you will be asked to confirm your new password on a third cover-up mask. Again type "tzs". If the confirmation matches, your password will be changed and you must use this new password the next time you log in. Only you know this password, thus establishing the security of your personal identity as known to the Multics operating system.

```
    Multics MR9.1: Honeywell LISD Phoenix, System M
    Load = 132.0 out of 180.0 units: users = 140, 07/25/99 0838.2 mst Wed
  ! l TSmith -cpw
    Password:
  ! ############
    New Password:
  ! ############
    New Password Again:
  ! ############
    Password changed.
```

Section 2
Creating a File

WORDPRO is a word processing capability available under the Multics operating system. With WORDPRO, you can generate a simple memorandum or a highly technical manual; you can generate letters, mailing lists, proposals, or contracts of almost unlimited length in a format you desire.

A Multics text editor is used with WORDPRO to enter and edit these documents. The text editor lets you predefine your document format for line length, page length, margins, headings, and other formatting features. It lets you add, insert, delete, or change words, lines, sentences, or paragraphs without having to retype several pages of text as might be required if only a typewriter were used.

This manual is designed for use with either a text editor called **qedx** or a text editor called **Ted**. Ted is an extension of qedx and provides for all of the functions generally available in a computer programming language. This manual describes and uses only the basic editing features of Ted and gives only those requests that are equivalent in both Ted and qedx. The programming capabilities of Ted are neither described nor incorporated.

In this manual, where a difference may exist between a basic editing request of qedx and Ted, the Ted example is shown to the right of the qedx example.

TEXT EDITOR ACCESS

The last line of the log-in sequence is a line consisting of the lowercase letter "r" followed by the time of day and two other numbers that reflect usage of the computer system. This line is called the "ready message".

```
r  8:55 3.222 61
```

The ready message is given by the Multics system to indicate it is at **command** level and ready to receive a command. The ready message is equivalent to having the system say, "I am free of other commands and ready to do as you command me. What is your command?"

Commands to the Multics system are always made from the keyboard of a data terminal by typing single words or simple phrases that, in most cases, have an abbreviated form called a **short name**. For example, by typing only a few letters, you can command the Multics system to format a document (if you had previously entered the document text through the text editor), check the spelling of the words in the document against the Multics dictionary (and/or your own dictionary), print the entire document, or list all the documents (called "segments") that may be stored in the system under your Person_id.

For the case at hand, you want to invoke the text editor. The command to the Multics system to invoke the text editor is given by typing the name of the editor in lowercase letters on the keyboard of the terminal. The command is completed by the carriage return. The terminal type determines which key or keys (e.g., **CR**, **LF**, **RETURN**, **LINE FEED**, **NL**) you press to obtain a new line. If you are using qedx, type "qedx" after the ready message and complete the carriage return. If you are using Ted, type "ted" after the ready message and give the carriage return.

```
    r  8:55 3.222 61                                 r 08:55 3.222.61

  ! qedx                                           ! ted
```

NOTE

In most examples given, all user input is shown preceded by an exclamation point to distinguish it from output lines from the computer. **Do not type the exclamation point!** Remember to follow all typed lines with a carriage return. And in most examples, all new input is shown preceded by the previous line printed at the terminal in order to show continuity.

You have now commanded the Multics System to enter the text editor. *Observe that there is no reply, nor prompt, nor signal of any kind to slow your operation or to indicate that the text editor has been invoked.* You are immediately ready to use the text editor!

To get out of the text editor and back to command level, you must type a request to the editor. That request is the lowercase letter "q", which means "quit the editor".

When the letter "q" is typed alone on a separate line (and followed by the carriage return), the Multics system will respond with a ready message to indicate it is no longer in the text editor and is ready to accept a new command. Type "q".

```
    qedx                                             ted
  ! q                                              ! q
    r  8:56 0.252 18                                 r 08:56 0.252 18
```

If you make a typing error at command level, the Multics system will indicate it does not recognize the command. For example, if by chance you type "qqedx" (or "tted"), the Multics system will respond immediately on your terminal with a phrase "Segment xxx not found." and another ready message. This response from Multics means that the system does not recognize the command, is not going to act on it, and instead, is ready for a command it can carry out. Type "qqedx" if you are using qedx, otherwise type "tted".

```
    r   8:56 0.252 18                                r 08:56 0.256 18

  ! qqedx                                          ! ted
    Segment qqedx not found.                         Segment tted not found.
    r  8:56 0.158 10                                 r 08:56 0.158 10
```

NOTE

The exact meaning of the "Segment xxx not found." message has to do with the way the Multics system searches for command procedures and is not important to this discussion.

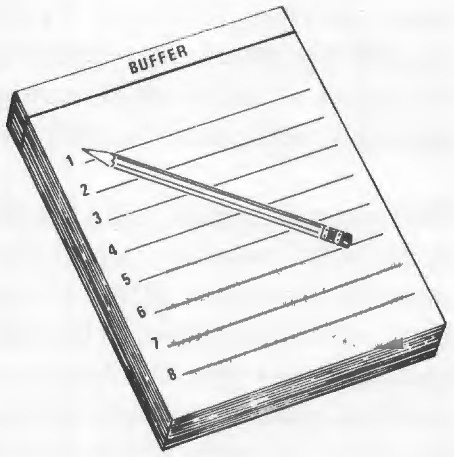
INTRODUCTION TO TEXT EDITOR REQUESTS

Typing the name of the text editor in lowercase letters after a ready message commands the Multics system to invoke that editor. For those using qedx, note that the Multics system will recognize either the full command, "qedx", or its short name, "qx", as the same command and will react accordingly. If you are using qedx, invoke the editor by typing the short name "qx". If you are using Ted, type "ted" to maintain continuity in this discussion.

```
    r 8:56 0.158 10                                  r 08:56 0.158 10

  ! qx                                             ! ted
```

Once in the text editor, you use the editor's input and edit **requests**. You use requests to the editor in order to enter any text you may type on your terminal, change your input, insert new text, delete or print all or portions of your input text, or substitute words or phrases. These requests, with one exception, are all single lowercase letters or symbols. You have already used one request, the quit editor request "q", which is always used as a stand-alone character (that is, no other characters may be typed on the line except, of course, the carriage return).

When the Multics system is commanded to invoke the text editor, it is as if it says, "I have a scratch pad with numbered lines and a pencil with an eraser. I also have access to a copying machine, a file drawer, and a waste basket. What do you now request of me?"



The equivalent of a lined scratch pad in the text editor is called a **buffer**. And just as anything written or typed on a scratch pad can be changed, copied, filed, erased, or totally discarded, so can the contents of the buffer be treated the same way by requests to the editor.

At this point, the buffer is empty. That is, the scratch pad is blank. You can confirm that the buffer is empty by using another request to the editor, the print request. Typing the lowercase letter "p" requests the text editor to print the line it is currently working on. Type the letter "p" and observe that you get an immediate printed response "Buffer empty." on your terminal.

```
      r 8:56 0.158 10

      qx
    ! p
      Buffer empty.
```

```
      r 08:56 0.158 10

      ted
    ! p
      Buffer empty.
```

APPEND REQUEST

Because the buffer is empty at this time, there is nothing for the text editor to edit. Consequently, your first request to the editor is a request to input new text. The request to input new text is called the **append** request. It is used to add or append input text to the contents already in the buffer. However, in this case of the empty buffer, any input appended becomes the only contents of the buffer. The input text you type on your terminal keyboard following this request may be an entire document, memo, or proposal, or a part of any such document. This input text is generally referred to as either a **file** or a **segment**.

The append request is an "input" request as opposed to an "edit" request such as the "q" or "p" request. It uses the lowercase letter "a" (all letter requests to the editor are lowercase). When the buffer is empty and you type the letter "a" alone on a line (followed by the carriage return), you are telling the editor, "Start on the first available blank line in the buffer and continue putting in everything I type, exactly as I type it, and continue doing so until I terminate this request."

As an example of input text, you will request the editor to append (put into the buffer or put on the scratch pad) Patrick Henry's quotation* in the form given as follows:

> "Now is the time
> For all good men
> To come to the aid
> Of their country."

For this append request, type the lowercase letter "a" on a line by itself. Then type three lines of the quotation omitting the line "For all good men" and deliberately using a lowercase letter "t" to begin the line "to come to the aid".

*This saying is often attributed in modified form to Charles Weller. However, since Patrick Henry is more commonly considered the originator, he will be cited as the source in this document. — Ed.

a
"Now is the time
to come to the aid
Of their country."

Other typing errors can, of course, be corrected by use of the erase (#) and kill (@) symbols. Remember that the end of each line must be followed by the carriage return including the line having only the request "a". Observe that there is no response from the editor that it has been given this request or is accepting the typed input (the quotation) in its buffer.

```
  Buffer empty.
! a
! "Not as@"Now is the time
! to come to the aid
! Of there##ir country."
```

NOTE

The text editor will accept input on the same line as the request "a", but it is better for a new user of the editor to form a habit of using separate lines for input requests.

When the editor is first invoked after a ready message, it is in its **edit** mode. The append request places the editor in its **input** mode. Any input request requires a **terminator** or **escape sequence** in order to tell the system that the input has been completed and to return back to the edit mode. The editor will recognize edit requests only if it is in the edit mode. While the editor is in its input mode, attempts to give edit requests such as "p" or "q" are recognized only as part of the input text and not as editor requests and the editor will not react to them. You cannot, for example, give the "q" edit request to quit the editor in order to log out while the editor is in the input mode; you must first give an escape sequence to leave the input mode.

The text editor uses several special characters that are not treated as requests. One of those special characters is a backslash (\ ) which is recognized to mean: "Take the following character for its defined special meaning rather than as text". The terminator or escape sequence from the input mode consists of the backslash followed by the lowercase letter "f". Thus the input mode escape sequence becomes " \f".

Input Requests
(Input terminator required)

a	append
i	insert
c	change

Edit Requests
(No terminator needed)

q	quit
p	print
w	write
r	read
s	substitute
d	delete
/XXX/	locate
=	print line number

NOTE

If your terminal does not have the backslash symbol, use the cents (¢) symbol instead to form "¢f"; if your terminal has neither "\" nor "¢", seek local assistance.

This " \f" sequence gets you out of the input mode so you can give the editor a new request, either an edit or input request, including another append request if you so desire. Without this escape sequence, the editor will continue to append the input regardless of what is typed.

NOTE

Typing " \f" does not remove you from the text editor. " \f" is an escape sequence for an input mode request and is not a request itself. In fact, if you inadvertently type " \f" while you are in the edit mode, qedx will immediately say: "\ not recognized as a request." while Ted will say: "Invalid request \".

After the last line of your input, give a carriage return as usual, type " \f", and again give a carriage return. Observe that the editor does not give any indication that you have indeed escaped from the input mode.

```
  Of there##ir country."
! \f
```

NOTE

If "\f" is not placed on a separate line, the last line of input will not have a carriage return, which will cause problems to new users during editing.

ABSOLUTE LINE NUMBERS

All typed input lines of text are given addresses by the editor. These addresses consist of line numbers as indicated by the "scratch pad" example, and all input text lines, including those lines deliberately left blank, are numbered consecutively by the editor.

When you add, insert, or delete lines, the editor immediately reassigns the line numbers. For example, if your document consists of 75 lines and you delete line 47 by a request to the editor, line 48 becomes the new line 47, line 49 becomes line 48, and so on, until line 75 becomes line 74.

All but three editor requests require the number or numbers of the line or lines upon which they are to act. When a line number or numbers are not supplied with those requests, the editor assumes a line number by default. Usually, the line number assumed by the editor by default is the number of the line the editor worked on last.

ADDRESSING THE APPEND REQUEST

In the case of the append request when the buffer is empty, the input starts on line 1. In the example, your input ended on line 3, which is the last line worked on by the editor (where the pencil point stopped on the scratch pad). The last line worked on by the editor (in this case, the last line of input) is given a special name; it is called the **current** line.

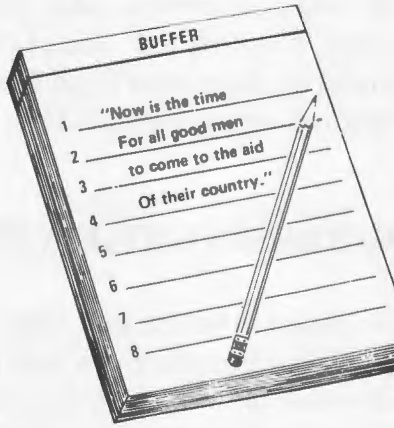
BUFFER
1 "Now is the time
2 to come to the aid
3 Of their country."
4
5
6
7
8

If you now give the editor the request "a" in order to add more text to the contents of your buffer, the editor will assume, by default, that you mean: "Add the new text after the current line." The text editor will then add the new text after line 3 (that is, start the text addition on line 4). However, you can append your input *after* any line you wish by specifying the number of the line with the request.

For example, in the input, the phrase, "For all good men", was purposely omitted. It should appear after line 1. You can request the editor to add this line after line 1 by specifying "1" as part of the append request. Thus, append the contents of your buffer by requesting "1a" and typing the omitted line. (Do not forget the carriage returns and the input escape sequence.) The text editor immediately will renumber all the lines of the contents of the buffer and then designate line 2 as the "current" line.

```
  \f
! 1a
! For all good men
! \f
```

You can find the line that the editor has designated the current line by typing a period any time you are in the edit mode. The period is another of the special characters such as the backslash that the text editor recognizes as having a special meaning. The period stands for the "current" line, and because it is not a request, the editor, by default, assumes a "p" or print request and responds by printing the current line.



Type "." and note that line 2 of your buffer contents (the last line worked on by the editor) has indeed become the current line.

```
  \f
! .
  For all good men
```

PRINT REQUEST

Often, when you put words or symbols on paper, you go back to review the material. In the same sense, you can check the contents of the buffer. To check the contents of the buffer, you can request the editor to print on the terminal one line or as many lines of your input as you desire.

The **print** request is an edit request and it does not require an escape sequence as does an input request. When the request is not preceded by a line address, the editor uses the number of the current line as the address for the print request and therefore prints only the current line. The print request uses the lowercase letter "p".

Recall that the current line is line 2. Because the text editor uses the current line as the default address in a print request, in this case typing "p" is the same as typing "2p". Thus, type "p" and observe that the text editor prints "For all good men".

```
  For all good men
! p
  For all good men
```

To request the editor to print a particular line other than the current line, precede the request letter with the line number. For example, type "3p" and observe that line 3 is printed. Line 3 has now become the last line worked on by the editor; in other words, the current line. Again type "p" and observe that line 3 is printed. Then, type "." and observe that line 3 has indeed become the current line.

```
  For all good men
! 3p
  to come to the aid
! p
  to come to the aid
! .
  to come to the aid
```

You can also request the editor to print several lines. The request to the editor to print more than one line uses the line numbers of the first and last lines you want printed, with the line numbers separated by a comma. For example, type "2,4p" to see lines 2, 3, and 4. To see lines 1, 2, and 3, type "1,3p".

```
  to come to the aid
! 2,4p
  For all good men
  to come to the aid
  Oftheir country."
! 1,3p
  "Now is the time
  For all good men
  to come to the aid
```

Another special character recognized by the editor is the dollar sign. When it is used as a line address, it means: "The last line of the contents in the buffer." Thus, although you will receive a print of line 4 by typing "4p", type "$p" instead and observe that the printed line is the last line of the contents of the buffer. (The dollar sign causes the system to print the last line in the buffer; the period causes the system to print the current line, which may or may not be the last line in the buffer.)

```
  to come to the aid
! $p
  Of their country."
```

The last line of the quotation is the last line worked on by the editor and is therefore the current line, easily confirmed by typing "p". This line is also printed by typing either "4p", "$p", or simply "4". In the case of "4", and as with typing the period, the editor finds the line and by default assumes the print request and reacts as if you had requested "4p". Now type "2".

```
  Of their country."
! p
  Of their country."
! 4p
  Of their country."
! $p
  Of their country."
! 4
  Of their country."
! 2
  For all good men
```

To receive a print of all four lines, use either of two requests: "1,4p" or "1,$p". The latter request using the "$" is most used because it eliminates having to remember the total number of lines of input. Because the dollar symbol means the last line of the contents of the buffer, "1,$p" is a request to the editor to print all lines from the beginning of the contents (line 1) through to the end of the contents ($). Type "1,$p".

```
  For all good men
! 1,$p
  "Now is the time
  For all good men
  to come to the aid
  Of their country."
```

It is always good practice to intermittently use the print request of the edit mode, if only for one line. Not only does this practice permit you to review the contents, but it also confirms that you have not accidentally gone into the input mode. If there is no response from the editor after a print request, chances are good that you probably and unintentionally gave the editor an input mode request and "p" has been placed as input text in your buffer. Type "\f" immediately to escape the input mode if the print request does not bring a response from the editor. Then give another print request; the editor should print a line containing the print request (which you will want to edit out later) for which nothing happened.

WRITE REQUEST

The buffer serves as a temporary storage area for your input and, as a scratch pad, is the area where all editing changes are made to the input. Because it is only as temporary as a scratch pad, the buffer can easily be emptied of its contents. The buffer will be emptied when you quit the editor by use of the quit request (crumple the scratch pad sheets and discard them), when you deliberately delete the buffer contents by request (erase everything on the scratch pad), or in the event of a system failure such as a broken telephone connection to your terminal (you lose your scratch pad).

However, if a copy of the buffer contents were made and filed in a permanent storage area, then even if the buffer were emptied of its contents, the copy could still be retrieved and editing continued. Furthermore, if a copy were made and filed, the buffer could be used for different input.

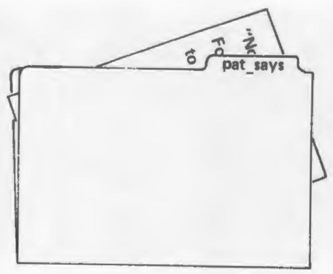
The **write** request to the editor is equivalent to using a copying machine for the buffer contents. You will first need to assign a name to the buffer contents so that the editor can find the copy in the permanent file when you call for it later. The name you assign to the buffer is called a **pathname**. If a pathname is not given as part of the write request, the editor will respond with an error message and wait for a new request.

The pathname you supply can be any combination of up to 32 alphanumeric characters including uppercase or lowercase letters but without spaces. By convention, the underscore is used instead of a space in the event you want to give a name having two or more words. Also, you should not use symbols such as >, <, *, =, ?, $, or left and right parentheses, because these characters have special significance in a pathname. Thus, for the present contents of your buffer, you may supply any pathname you wish, such as "example", "Quote", "quote_1","Henry_quote", or simply "pat_says".

The write request to the editor to file your present buffer contents is an edit request so you must be out of the input mode. To make the write request, type the lowercase letter "w" followed by the pathname you are supplying. When the request is given, there will be no indication from the editor that it has recognized the request. Also the editor will assume by default that all lines of input in the buffer are to be filed when an address is not given. That is, the editor assumes that "w" means "1,$w". However, if you wish to save only certain lines, then you must supply an address. For example, if you wish to save only lines 19 through 27, the request you make to the editor would be typed as "19,27w pathname". Type "w pat_says" to file all the lines of the contents of your buffer in permanent storage.

```
  Of their country."
! w pat_says
```

Note that requesting the editor to write your buffer contents in permanent storage does not at this time empty the buffer of its contents. At this time, the contents in both places (in the buffer and in the permanent file) are identical, but only the contents of the buffer can be edited. Editing changes do not affect the permanent file until or unless you issue another write request. Remember, the buffer is emptied of its contents only in the event of system failure, when you deliberately delete the contents, or when you use the quit request to get out of the editor. The permanent file remains untouched until you issue another write request with the same pathname.



You can easily verify that the contents of the buffer have not been removed at this time by using the print request for all the contents or for only one line. Type "p".

```
  w pat_says
! p
  Of their country."
```

QUIT REQUEST

Now that your input has been preserved in permanent storage, you can quit the editor and not have to repeat your input (which could easily be several pages). The **quit** request uses the lowercase letter "q" without an address and puts you back at Multics command level. The indication that you are back at command level is the ready message. Type "q".

```
                      Of their country."
                    ! q
                      r  8:59 2.602 129
```

Because you are now back at command level, you can use your terminal for new commands such as to log out, or if you so desire, to return to the editor. To return to the editor, again type the name of the editor in lowercase letters. Then, to ascertain the contents of your buffer, use the editor print request. Note that your original input is no longer in the buffer. The buffer was emptied of its contents when you gave the quit request to the text editor.

```
  r 8:59 2.602 129                      r 08:59 2.602 129

! qx                                  ! ted
! 1,$p                                ! 1,$p
  Buffer empty.                         Buffer empty.
! q                                   ! q
  r 9:00 0.028 7                        r 09:00 0.028 7
```

As another example of the use of the four requests described thus far, use the append request to input the following child's nursery rhyme:

Mary had a little lamb
Its fleece was white as snow
And everywhere that Mary went
The lamb was sure to go

Then, review your input by use of the print request, use the write request to make a permanent file with a pathname "Mary", and finally quit the text editor and log out. You will now have two permanent files — one called "pat_says" and one called "Mary".

Commands and Requests **Analogy**

```
   r 9:00 0.028 7
```

Multics: "What is your command?"

```
! qedx                    ! ted
```

Let me use an editor.

Editor: "What is your request?"

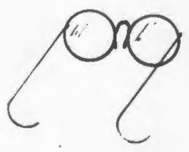
```
! a
! Mary had a little lamb
! Its fleece was white as snow
! And everywhere that Mary went
! The lamb was sure to go
! \f
```

Append

Put my input on the scratch pad.

Editor: "What is your next request?"

```
! 1,$p
  Mary had a little lamb
  Its fleece was white as snow
  And everywhere that Mary went
  The lamb was sure to go
```

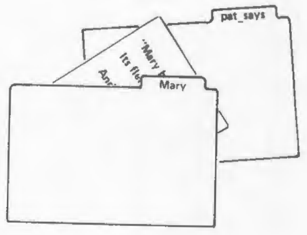


Print

Let me review the scratch pad contents.

Editor: "What is your next request?"

```
! w Mary
```

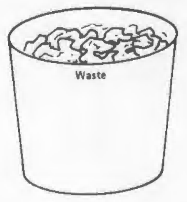


Write

File a copy of the scratch pad contents under the name "Mary".

Editor: "What is your next request?"

```
! q
```



Quit

Put me back at command level and discard the scratch pad contents.

```
  r 9:01 0.602 52
```

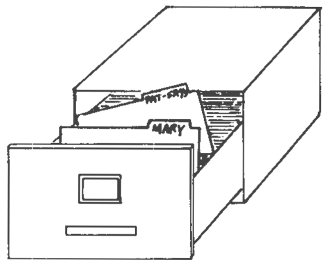
Multics: "What is your command?"

```
! logout
```

Let me quit my terminal.

Section 3
Editing a File

In Section 2, you entered two files through your text editor and placed them as "pat_says" and "Mary" in permanent storage. Each time after writing a file, you quit the editor which cleared your buffer of all its contents. However, it is only in the buffer that editing of those files can be performed.



Note

The correct terminology for "file" is "segment", but usage of the term "file" is widespread and it is used interchangeably with "segment" in these discussions.

READ REQUEST

If changes are to be made to either or both of your files in permanent storage, those files must be placed back in the buffer. The **read** request to the editor does just that. When the editor receives a read request for a particular file, it places a copy of that permanent file in the buffer (on your scratch pad) and leaves the original in permanent storage. Thus, immediately after a read request for a file, there are two identical files, one permanent and one temporary.

The read request is given in a manner similar to the way the write request is given. That is, you must also give the name (pathname) of the file you request and there will be no indication from the editor that the request is recognized or carried out. On the contrary, if an incorrect file name is given, the editor will reply with a message that the entry cannot be found.

The read request is an edit request and does not need an escape sequence. It uses the lowercase letter "r", and as stated, must be followed by the pathname of the file to be read. This request, unless given an address, will add the file contents after the last line of the buffer contents. Therefore, the editor assumes the last line of the buffer contents as a default address for the read request. That is, unless you supply an address, giving a read request "r pathname" is assumed by the editor to mean "$r pathname".

If you are not logged in, log in, and after the ready message, give the Multics system the command to invoke the text editor. Then give the editor the request to print the contents of the buffer, "1,$p". Observe that the buffer is empty. Now have the editor read a nonexistent file such as "Patrick" by typing "r Patrick". Observe the reply from the editor.

```
     r 12:57 0.486 27

   ! qx
   ! 1,$p
     Buffer empty.
   ! r Patrick
     qedx: Entry not found. Patrick


                                   r 12.57 0.486 27

                                 ! ted
                                 ! 1,$p
                                   Buffer empty.
                                 ! r Patrick
                                   Entry not found. >udd>ProjA>TSmith>Patrick
```

Next, give the request to read your file "pat_says" by typing "r pat_says". Note that there is no prompt before nor recognition after this last request.

```
     qedx: Entry not found. Patrick
   ! r pat_says



                                   Entry not found. >udd>ProjA>TSmith>Patrick
                                 ! r pat_says
```

Thus, to confirm that the editor has indeed read the "pat_says" file and placed it as contents of your buffer, request a print of your buffer contents by typing the now familiar request "1,$p".

```
  r pat_says
! 1,$p
  "Now is the time
  For all good men
  to come to the aid
  Of their country."
```

Prefixing a line number to a read request causes the editor to add the file after that line number of the existing buffer contents. Remember that if the request is not preceded by a line number, the editor assumes you mean "$r pathname". If you wish the file you are reading in to be placed at the beginning of the existing buffer contents (that is, before line 1), use a zero line number for the address and type "0r pathname".

As an example of an addressed read request, place the contents of the file called "Mary" after line 2 of your buffer contents by typing "2r Mary". Now type "p" to see that the current line is the last line worked on by the editor, the last line of the file just read. Then, type "1,$p" to confirm that the editor has acted upon your request and placed the file called "Mary" after line 2 of the buffer contents, which was the first file read, "pat_says".

```
  Of their country."
! 2r Mary
! p
  The lamb was sure to go
! 1,$p
  "Now is the time
  For all good men
  Mary had a little lamb
  Its fleece was white as snow
  And everywhere that Mary went
  The lamb was sure to go
  to come to the aid
  Of their country."
```

Observe that your buffer now contains copies of the contents of both files in permanent storage. But, neither of those files has been destroyed nor have they been altered. Both exist in permanent storage in their original form.

Because the contents of your buffer may be placed in permanent storage at any time, you could assign a pathname to the present contents, give the editor a write request, and create a third permanent file. Instead, however, request the editor to write "Mary" again. Then quit the editor, which will empty the buffer of its contents and return you to command level.

```
  Of their country."
! w Mary
! q
  r 12:58 0.482 45
```

After the ready message, again invoke the text editor, request a print of your buffer contents to confirm that the buffer is empty, and then request the editor to read "Mary".

```
  r 12:58 0.482 45                          r 12:58 0.482 45

! qx                                      ! ted
! 1,$p                                    ! 1,$p
  Buffer empty.                             Buffer empty.
! r Mary                                  ! r Mary
```

NOTE

Recall that the read request will add the new file to existing buffer contents. The quit request is used here only to clear the buffer. (A delete request to be described later could have been used instead and would have eliminated the return to command level.)

Again request a print of your buffer contents and note that they are identical to those when the buffer was used last. In other words, the four lines originally assigned to the file "Mary" have been replaced by eight new lines. By analogy, the editor has made a copy of the latest buffer contents and placed the copy in the permanent file named "Mary". Then, the editor destroyed the original contents of that same permanent file. This action of replacing the original contents of a permanent file with the contents of your buffer is called "overwriting".

```
                 r Mary
               ! 1,$p
                 "Now is the time
                 For all good men
                 Mary had a little lamb
                 Its fleece was white as snow
                 And everywhere that Mary went
                 The lamb was sure to go
                 to come to the aid
                 Of their country."
```

PRINT LINE NUMBER REQUEST

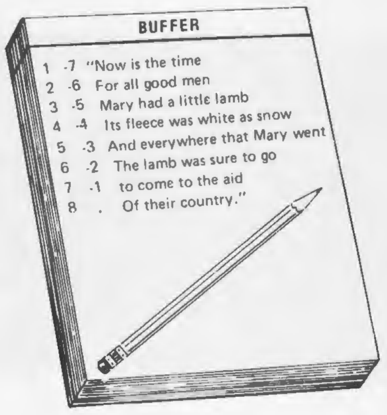
The contents of your buffer presently consist of eight lines. Eight lines are not too difficult to count nor find for addressing requests. However, in a much more lengthy file, trying to keep track of **absolute line numbers** for addressing purposes can be a time-consuming process of giving several requests of "1,$p" and then physically counting lines. Remember that the text editor is constantly changing the line numbers as you request it to read, append, insert, or delete your input.

You can, of course, easily find the current line — by typing the period by itself or by typing the letter "p" by itself. The text editor, however, recognizes a request that will give you the absolute line number of the current line. That request is the equal sign. Thus, type "=" and observe that the line number of the current line of your buffer contents is "8". (Recall that you can change the current line by simply typing the number of the line you wish to be the current line).

```
  Of their country."
! =
  8
```

RELATIVE LINE NUMBERS

So far you have worked with absolute line numbers, but the editor lets you use other kinds of line numbers too, **relative line numbers.** Relative line numbers are given with respect to the current line designated by the period. These line numbers are indicated by either plus (+) or minus (−) signs. A minus sign indicates a line before the current line and a plus sign indicates a line after the current line. The first line before the current line is designated as "−1", the second line before the current line is designated as "−2", etc. The first line after the current line is given as "+1", and so on. If you wished to print the line two lines before the current line (the period), the editor would recognize the request "−2p". If you wanted to print a line three lines after the current line, the editor would recognize the request "+3p".


BUFFER
1 -7 "Now is the time
2 -6 For all good men
3 -5 Mary had a little lamb
4 -4 Its fleece was white as snow
5 -3 And everywhere that Mary went
6 -2 The lamb was sure to go
7 -1 to come to the aid
8 . Of their country."

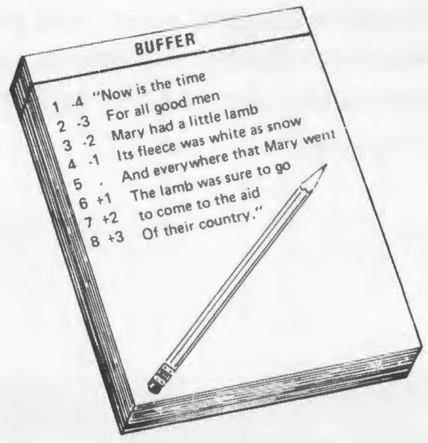
The present current line of the file in your buffer is the line "Of their country." Type "−3p" and observe that the editor prints the line three lines before the current line. This line (whose absolute line number is 5) is the last line worked on by the editor and thus is now the new current line.

```
  8
! -3p
  And everywhere that Mary went
```

Type "+2p" and observe that the line two lines after this current line is printed.

```
  And everywhere that Mary went
! +2p
  to come to the aid
```

The text editor also recognizes relative line numbers in two-part addresses. For example, type "−3,−1p" and note that the editor prints out the lines from three lines before the current line through the last line before the current line.


BUFFER
1 -4 "Now is the time
2 -3 For all good men
3 -2 Mary had a little lamb
4 -1 Its fleece was white as snow
5 . And everywhere that Mary went
6 +1 The lamb was sure to go
7 +2 to come to the aid
8 +3 Of their country."

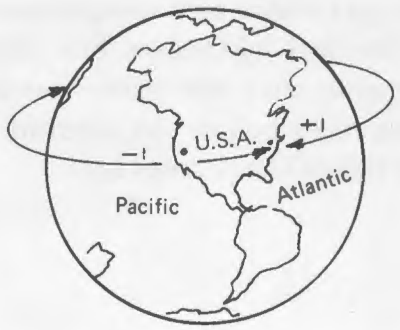
```
  to come to the aid
! -3,-1p
  Its fleece was white as snow
  And everywhere that Mary went
  The lamb was sure to go
```

Relative line numbers can be used to change your current line in the same way absolute line numbers are used. To change your current line to the following line, simply type "+1". (Note that the current line could also have been changed by typing the absolute line number, "7", instead.)

```
  The lamb was sure to go
! +1
  to come to the aid
```

Consider the effect if the plus and minus values are reversed. For example, suppose you wished a print of the lines before and after the current line and requested "+1,−p". The message given by the editor, "Address wrap−around." or "Addr− wrap−around.", means that the editor would have to print from the current line to the end of the file, "wrap around" to the top of the file, and then print from the top of the file down to the current line. The text editor assumes that that is not what you wanted as a request. (By analogy, to travel from the west coast of the United

States to the east coast, you probably would not start west over the Pacific Ocean, go around the globe, and end on the east coast over the Atlantic Ocean.) Type "+1,−1p".



```
    To come to the aid                          to come to the aid
  ! +1,-1p                                    ! +1,-1p
    Address wrap-around                         Addr- wrap-around.
```

The present current line is the next to last line of your file. If you were to request a print such as "−1,+3p", that request would take the editor beyond the end of your file. The text editor would reply with an error message and would not complete the request. Likewise, a request such as "−8,+1p" would take the editor before the beginning of your file and the editor would react similarly.

Recall that the dollar sign means the last line in your file in the buffer. It can therefore also be used in the two-part address involving relative line numbers. Type ".,$p" and observe that the editor prints the current line through the last line of your file.

```
    Address wrap-around.                        Addr- wrap-around.
  1 .,$p                                      ! .,$p
    to come to the aid                          to come to the aid
    Of their country."                          Of their country."
```

LOCATE REQUEST — CONTEXT ADDRESSING

Text to be edited usually consists of several lines, and in many cases, may consist of several pages. When you review the contents of your buffer (with the print request), you may see several lines needing correction because of typographical errors or misused wording. You can, of course, find the line in question by using the print line number request, changing the current line, and so on, but the process becomes slow and often frustrating.

However, the text editor recognizes a **locate** request that is given to search for the line containing a particular string of alphanumeric characters (such as a misspelled word). This string of characters is called a **regular expression.** The locate request uses this regular expression, which may consist of complete words (including those misspelled), short phrases, or sometimes single letters, placed within slash marks. The slash marks are called **delimiters** because they set off the regular expression. The delimiters used in the locate request must be slash marks. When you give the locate request, the editor will search for the line containing the regular expression within the slash marks, print that line, and make that line the current line.

For example, the present current line of the contents of your buffer is the very last line. But, suppose that you desire to make a change in the line containing the word (regular expression) "good". Because there are only eight lines in this example, it is not too difficult to give an absolute line number of "2" or a relative line number of "−6". If your text consisted of several hundred lines, determining the line number could become quite tedious. Using the locate request simplifies the task.

Thus, type "/good/" and observe that this edit request causes the system to print immediately the line containing the regular expression, which, in this case, is the word "good". In this example, the printed response is the line "For all good men". Type "p" and note that this line has become the current line.

```
  Of their country."
! /good/
  For all good men
! p
  For all good men
```

The text editor locates the regular expression (the word, phrase, numbers, etc.) within the slash marks by searching from immediately after the current line down to the end of the file and then from the top of the file down through the current line. In other words, the editor uses "address wrap-around" when it looks for the regular expression given within the slash marks. For example, the word "the" appears in the line immediately preceding the current line which is "For all good men", but it also appears later in the file. Type "/the/" and observe that the editor prints the later line, "to come to the aid".

```
  For all good men
! /the/
  to come to the aid
```

Now type "/the/" again and be prepared for an unexpected result!

```
  to come to the aid
! /the/
  Of their country."
```

The text editor has acted on this locate request and found the regular expression (letters) "t-h-e" in the last line of the quotation in the word "their". Consequently, you must exercise care when selecting the word or letters used as the regular expression within the slash marks. In this case, because you were searching for the word "the" rather than the string of three characters "t-h-e", it would have been better to have used "/ the /" with spaces before and after "the". Another locate request for "/the/" will give the first line of your file.

```
                                        Of their country."
                                      ! /the/
                                        "Now is the time
```

The text editor does not need complete words or phrases as the regular expression in a locate request. For example, if you request "/ei/", the text editor will locate the last line of the contents of your buffer because that is the only line in which "ei" appears. However, the editor will differentiate between uppercase letters and lowercase letters. Type "/ei/", wait for the response, and then type "/its/" (rather than "/Its/". Observe that the editor cannot locate a line in your file containing the expression "its" because "its" with a lowercase "i" is not the same as "Its" with an uppercase "I".

```
    "Now is the time                       "Now is the time
  ! /ei/                                 ! /ei/
    Of their country."                     Of their country."
  ! /its/                                ! /its/
    Search failed.                         Line search failed. "/its/"
```

The dollar sign thus far in these discussions has meant the last line of the contents of your buffer. When used as part of the regular expression in a locate request, it refers to the last character on a line. (It actually refers to an imaginary character following the last character on a line.) It is most often used to locate a line ending with a unique regular expression. As an example, the letter "e" appears several times in the contents of your file, but only in the first line does it appear at the end. Type "/e$/" and note that the editor searches and finds the line ending with the letter "e".

```
    Search failed.                         Line search failed. "/its/"
  ! /e$/                                 ! /e$/
    "Now is the time                       "Now is the time
```

The circumflex (^) is a special character recognized by the editor to mean an imaginary character preceding the first character on a line. Therefore, the circumflex can be used as part of a regular expression in a locate request to find a line beginning with that regular expression. As an example, the file contains the lowercase letter "t" several times, but only one line begins with "t". Type "/^t/" and observe that the text editor finds that line.

NOTE

If your terminal does not have the circumflex, use the "not" symbol (¬) or the upward arrow (↑).

```
  "Now is the time
! /^t/
  to come to the aid
```

In reality, the locate "request" is not an editor request at all, but simply another form of addressing called **addressing by context.** However, the locate function is so handy and used so often that it is treated here like a request for your convenience.

SUBSTITUTION REQUEST

The **substitution** request to the text editor is an editing request used to alter any regular expression. It can be used to correct spelling errors, delete words, insert words or to change entire lines. The substitution request uses the lowercase letter "s" and takes the form "s/old/new/". If you supply an address with this request, the address can be either absolute, relative, or by context. That is, you can address the line by its absolute line number, its relative line number, or by the locate request. The default address is the current line.

The second line of your buffer contents contains the word "men". To change the regular expression "men" to "men and women", the "old" part of the subsitution request is "men" and the "new" part is "men and women". The request can be addressed either as "2", "−5" (because the current line is "to come to the aid"), or "/all/" as an example of a context address. This context address could, of course, be any regular expression that would designate this line, including the first letter, or "oo", or "men", etc. Type "2s/men/men and women/". Then request a print of the changed line. The substitution request does not, by itself, cause a response from the text editor.

```
  to come to the aid
! 2s/men/men and women/
! p
  For all good men and women
```

In the following line, change the word "little" to "small". The address can be given as either "3", "+1", or possibly "/Mar/". Type the request as "+1s/little/small/" followed by the print request "p".

```
  For all good men and women
! +1s/little/small/
! p
  Mary had a small lamb
```

Not only was a substitution request made in these last two examples and carried out by the text editor, but also a second request, the print request, had to be made each time in order to see the corrected line. That is, the editor does not give a response or indication that the substitution has been made. However, in most cases, the editor will recognize more than one request on the same line. For example, the print request can be made following the final slash mark. Likewise, a locate request can be made before the substitution request to give addressing by context. Now, locate the line having the word "time", substitute "hour" for "time", and request a print. Type the request "/time/ s/time/hour/ p".

```
  Mary had a small lamb
! /time/ s/time/hour/ p
  "Now is the hour
```

NOTE

Spaces are not necessarily required between requests on the same line, but they aid the new user in finding procedural errors and in editing.

Deleting a word by use of the substitution request requires some care. Consider deleting the word "good" in the second line. Type "2s/good// p".

```
  "Now is the hour
! 2s/good// p
  For all  men and women
```

The two delimiters (in this case, slash marks) typed together without space separation is called a **null expresssion.** Here it means that "good" is to be deleted and is not to be replaced with any other expression. Note, however, that in the resulting print of the line there are one too many spaces between the words "all" and "men" because you have not included an extra space in the regular expression within the delimiters. (The correct request to the editor should have been "2s/good // p" to show that "good" and the following space should have been deleted.)

You can correct this error by typing "s/l m/l m/ p" with two spaces between the "l" and "m" in the "old" and one space between the "l" and "m" in the "new" part of the request in order to delete the existing extra space between the "l" in "all" and the "m" in "men".

```
  For all  men and women
! s/l  m/l m/ p
  For all men and women
```

Because the editor recognizes multiple requests on the same line and carries them out in the order given, the locate and substitution requests can be shortened somewhat. That is, the editor will "remember" the last regular expression it has seen. Thus, if the regular expression in the locate request is the same as the regular expression in the "old" part of the substitution request (as in the case of "time" in the example above), a null expression can be used to replace the "old" part of the request. A null expression in the "old" part of a substitution request thus differs from its meaning when it is used in a "new" part. For example, use the null expression and type "/come/ s//go/ p" to change "come" to "go" in the line "to come to the aid".

```
  For all men and women
! /come/ s//go/ p
  to go to the aid
```

The current line requires a capital "T" at its beginning. A substitution request here will point out again why care is needed when using this request. Type "s/to/To/ p" and note that both times the word "to" appears in the line it gets capitalized.

```
  to go to the aid
! s/to/To/ p
  To go To the aid
```

Change both to lowercase by typing "s/T/t/ p". Correct the line by using the circumflex to indicate to the editor that you mean the regular expression at the beginning of the line and no expression elsewhere on the line. (You can, of course, achieve the same correction by typing "s/to g/To g/" as the request because "to g" appears only once in the line.)

```
  To go To the  aid
! s/T/t/ p
  to go to the aid
! s/^t/T/ p
  To go to the aid
```

The slash marks are called delimiters because they identify the beginning and end of the expressions used. The locate request *always* requires slash marks as delimiters. The substitution request uses them by convention because they seldom appear in either the "old" or "new" part of the request. However, any character that is in neither the "old" nor "new" can be used as delimiters for the substitution request. For example, if you wanted to change "aid" to "aid/help" in the current line (where this slash means "and or") and used slash marks as delimiters, the editor will send an error message. Thus, type the request, "s/aid/aid/help/ p". Note that the substitution of "aid" for "aid" will have been carried out but the editor does not recognize the request "h" on the request line following what the editor sees as a third delimiter or final delimiter of the "new" part.

```
  To go to the aid                          To go to the aid
! s/aid/aid/help/ p                       ! s/aid/aid/help/ p
  qedx: h not recognized as a request.      Bad decimal digit. h
```

A more serious problem can occur if care is not made in the substitution request and other recognizable requests are made. Consider substituting the expression "help/aid" rather than "aid/help". Type "s/aid/help/aid/ p" and observe that the editor seems to have quit editing.

```
  qedx: h not recognized as a request.      Bad decimal digit. h
! s/aid/help/aid/ p                       ! s/aid/help/aid/ p
```

Here, the text editor has substituted the word "help" (the "new") for "aid" (the "old") and then recognized the letter "a" after the third slash mark as an input request. Furthermore, the editor has input "id/ p" after your current line. Type " \f" to escape this input mode and request a print to see how your file has been altered.

```
  s/aid/help/aid/ p
! \f
! 1,$p
  "Now is the hour
  For all men and women
  Mary had a small lamb
  Its fleece was white as snow
  And everywhere that Mary went
  The lamb was sure to go
  To go to the help
  id/ p
  Of their country."
```

If the last substitution request had been written using, for example, vertical bars, question marks, or for that matter, the letter "x", as delimiters, then the editor would have carried out the substitution as you wished. That is, the request would have been carried out if you had typed "s%aid%help/aid% p" where any character, in this case percent symbols, not appearing in either the "old" or "new" parts of the request had been used for delimiters.

Your file now contains an unwanted line. The substitution request cannot be used to remove this line. The substitution request can, as you have seen, be used to remove regular expressions within a line, but any attempt to remove an entire line will leave you with a blank line. Attempt to remove this unwanted line by typing "–1s?id/ p??" using the null expression (two delimiters typed together) for the "new". (Remember any character not in either the "old" or "new" can be used as delimiters.) Then, print the line before the current line, the curent line, and the line after the current line by typing "−1,+1p". Note that you are left with a blank line.

```
                                  Of their country."
                                ! -1s?id/ p??
                                ! -1,+1p
                                  To go to the help

                                  Of their country."
```

To find the blank line, type "−1" to change your current line; then type "=" and observe that the blank line has an absolute line number.

```
                                  Of their country."
                                ! -1

                                ! =
                                  8
```

Although the substitution request cannot be used to remove a blank line, it can be used to place a regular expression on a blank line. Attempt to replace the expression "id/ p" on this blank line by typing the request, "s??id/ p" and observe that the substitution fails. The editor "remembers" the last regular expression used in the "old" if that expression is not a null expression. In this case, the last remembered regular expression in the "old" was "id/ p" but it was replaced by a blank and the "old" regular expression no longer exists on the line and therefore cannot be replaced.

```
     8                                          8
   ! s??id/ p? p                              ! s??id/ p? p
     Substitution failed.                       Substitute failed.
```

The only characters that have meaning on a blank line are "^" and "$". Thus, type "s?^?id/ p" followed on the next line by "−1,+1p" and observe that the editor carries out your request. The circumflex, of course, tells the editor to place the expression at the beginning of the line.

```
  Substitution failed.
! s?^?id/ p?
! -1,+1p
  To go to the help
  id/ p
  Of their country."
```

```
  Substitute failed.
! s?^?id/ p?
! -1,+1p
  To go to the help
  id/ p
  Of their country."
```

The circumflex, when used in the "old" part, means the imaginary character preceding the first character on a line. The dollar sign, when used in the "old" part of the substitution request means the imaginary character following the last character on a line. For example, suppose that you wanted to place a period after the end of the line "The lamb was sure to go". First, locate the line using "sure" as a context address. Second, make the substitution request by typing "s/$/./". Third, request a print of the line. (Note that the dollar sign could have been used when "men and women" was substituted for "men" earlier by typing "2s/$/ and women/" — don't forget the space before "and".)

```
  Of their country."
! /sure/ s/$/./ p
  The lamb was sure to go.
```

NOTE

When the "^" and "$" special characters are used in the substitute request, they are special only in the "old" part of the request. In the "new" part of the request, they are treated as ordinary alphanumerics.

The substitution request can be used with a two-part address as with the print request. For example, suppose that you wished all occurrences of the lowercase letter "o" to be replaced by capital letters in the first four lines of your file. Type "1,4s/o/O/" and then the print request for those lines, "1, 4p".

```
  The lamb was sure to go.
! 1,4s/o/O/
! 1,4p
  "NOw is the hOur
  FOr all men and wOmen
  Mary had a small lamb
  Its fleece was white as snOw
```

To replace the same regular expression throughout your file, the address would be "1,$". Thus, replace the word "the" throughout your file with "our" by typing "1,$s/ the / our /" and request a print of your file contents (include the spaces in the request). Observe that all words "the" have been changed to "our" except the line having the uppercase "T" in "The".

```
  Its fleece was white as snOw
! 1,$s/ the / our /
! 1,$p
  "NOw is our hOur
  FOr all men and wOmen
  Mary had a small lamb
  Its fleece was white as snOw
  And everywhere that Mary went
  The lamb was sure to go.
  To go to our help
  id/ p
  Of their country."
```

DELETE REQUEST

The **delete** request is given to the editor in order to delete one or more lines of a file. This request is an edit request that uses the lowercase letter "d". If an address is not given, the editor will, by default, use the address of the current line and thus delete the current line. And, as for several of the other requests, the editor will not provide any indication to show that it has recognized and performed the request.

Your current line is the last line of the contents of your buffer. In this example, you will delete the line preceding this current line. Again, addressing can be absolute, relative, or by context. In this case, relative addressing is easiest and you will delete the line by requesting "−1d". Type "−1d" and then request "−1,$p" to print the lines starting with the line before the present current line to the end of the file. Observe that the line has been deleted.

```
  Of their country."
! -1d
! -1,$p
  To go to our help
  Of their country."
```

When a line is deleted by the delete request, the current line becomes the line immediately following the line deleted. Also, the editor immediately renumbers all the lines of the file in your buffer. Type "2d" to delete the second line of your file and then determine the current line by typing a period. Now type "1,3p" to see that the former line 3 has now become line 2, former line 4 has become line 3, etc.

```
  Of their country."
! 2d
! .
  Mary had a small lamb
! 1,3p
  "NOw is our hOur
  Mary had a small lamb
  It fleece was white as snOw
```

If the last line of a file is the current line and that line is deleted, the question arises, "What is the new current line?" Type "$d" to delete the last line of your buffer contents, and then type "p". Note that because the current line becomes the line immediately following a deleted line, deleting the last line makes the current line one line beyond the end of the buffer. For qedx, the error message given is "Address out of buffer.", while for Ted, the error message states that the current line, which is denoted as ".", is not defined.

```
     Its fleece was white as snOw                Its fleece was white as snOw
   ! $d                                        ! $d
   ! p                                         ! p
     Address out of buffer.                      "." undefined.
```

To delete more than one line, use the same addressing format as for the print request; that is, the numbers for the first and last lines to be deleted are separated by a comma. For example, to delete lines 4 through 6, you would type "4,6d". To delete the entire contents of the buffer, use "1,$d". Now, delete the entire contents of the buffer and verify that the contents have been deleted by using the print request. Type "q" to get out of the editor and back to command level. Then log out.

```
     Address out of buffer.                      "." undefined.
   ! 1,$d                                      ! 1,$d
   ! 1,$p                                      ! 1,$p
     Buffer empty.                               Buffer empty.
   ! q                                         ! q
     r 13:10 6.600 391                           r 13:10 6:600 391

   ! logout                                    ! logout
```

Section 4
Editing Refinements

Section 2 showed you how to create a file using the append input request and how to place that file in permanent storage using the write request. Section 3 gave techniques for editing that input and showed some of the special characters recognized by the text editor. This section describes two other input requests recognized by the editor and elaborates on the use of special characters. This section also describes the commands used to remove files from permanent storage when you no longer need them.

INSERT REQUEST

You have used the append request to place new text in your buffer. The text editor also recognizes an **insert** request that can be used for the same purpose. In fact, when the buffer is empty, the editor will recognize either request to input new typing.

The insert request is an input request that uses the lowercase letter "i". Because is it an input request, it requires an escape sequence at the end of your new input — the same "\f" terminator used as the escape sequence for the append request. If you do not supply an address with the request, the editor will by default use the address of the current line and place your input immediately before that line. The current line after an insert request is the last line inserted.

Whereas the append request places new text as input *after* a designated address, the insert request places the input *before* the designated address. Consequently, the insert request cannot be used if you wish to add text at the end of the contents of your buffer.

If you are not logged in, log in, and invoke the text editor, Read "pat_says" into your buffer from permanent storage and refresh your memory as to the contents of the file by using the print request. You will now use the insert request to supply a heading to this quotation.

```
     r 08:23 0.096 11                                r 08:23 0.096 11

   ! qx                                            ! ted
   ! r pat_says                                    ! r pat_says
   ! 1,$p                                          ! 1,$p
     "Now is the time                                "Now is the time
     For all good men                                For all good men
     to come to the aid                              to come to the aid
     Of their country."                              Of their country."
```

Request the text editor to insert the words "Famous Quote" as a heading. Because you want the heading to appear before the first line of the contents of your buffer, and because an insert request places input before an addressed line, use line "1" as your request address. Type "1i", input the heading, and escape from the input mode. Request a print of the first three lines to confirm the placement of this inserted input.

```
                                  Of their country."
                                ! 1i
                                ! Famous Quote
                                ! \f
                                ! 1,3p
                                  Famous Quote
                                  "Now is the time
                                  For all good men
```

NOTE

The text editor will recognize either "0a" ("zero a") or "1i" ("one i") as the request to place new typing before the existing contents of the buffer. (Recall that to place a file from permanent storage at the start of the contents of your buffer, the request is "0r path-name".)

Observe that the text would look better if the heading were separated from the first line of the quotation by a blank line. Likewise, it would look better if the heading were preceded by a blank line. Two separate input requests will be needed for blank lines before and after the heading. Input text for a blank line is simply the carriage return, **CR**. Thus, type "/Now/ i", **CR**, and "\f". Next type "−1i", **CR**, and "\f". Now request a print of the contents of your buffer.

```
    For all good men
  ! /Now/ i
  !
  ! \f
  ! -li
  !
  ! \f
  ! 1,$p

    Famous Quote

    "Now is the time
    For all good men
    to come to the aid
    Of their country."
```

As you can see, you have now used all three forms of addressing for the three insert requests made, that is, "1i" (absolute), "/Now/ i" (by context), and "−1i" (relative). All three forms of addressing can be used with all three input requests — the append request, the insert request, and the change request (to be described later). Likewise, one insert request could have sufficed instead of the three in order to insert the three lines at the start of the contents of your buffer. That request would have had line "1" as its address. (Two-part addresses may not be used with append or insert requests.)

The default address for an insert request is the current line, which is presently the last line of the contents of your buffer. Attempt to add a signature, such as "Henry", by using the insert request with the default address by typing "i", "Henry", and "\f".

Now request a print of the current line to confirm that the last line of input from an insert request becomes the current line. Then request "–2,$p" to see the placement of "Henry".

```
                          Of their country."
                        ! i
                        ! Henry
                        ! \f
                        ! .
                          Henry
                        ! -2,$p
                          For all good men
                          to come to the aid
                          Henry
                          Of their country."
```

Use the delete request to remove the line last inserted ("Henry"). Recall that the delete request causes the next line (the last line again) to become the current line. (Also, at this time, substitute "To c" for "to c" in the line "to come to the aid".) Request a print of the last two lines of your file.

```
                          Of their country."
                        ! -1d
                        ! -1s/to c/To c/
                        ! .,$p
                          To come to the aid
                          Of their country."
```

Now attempt to insert the signature after the current line by using the request "+1i", and observe that the text editor replies immediately with an error message. An attempt to give an append request "+1a" at this time will produce the same reply.

```
    Of their country."                          Of their country."
  ! +1i                                       ! +1i
    Address out of buffer (too big).            Addr- after  buffer
  ! +1a                                       ! +1a
    Address out of buffer (too big).            Addr- after  buffer
```

Use the append request to attach the name "Henry" to your quotation. Remember that the append request adds *after* a designated line and, by default, after the current line. Leave a blank line before and after the signature. Request a print of the entire contents of your buffer.

```
    Address out of buffer (too big).          Addr- after  buffer
  ! a                                       ! a
  !                                         !
  ! Henry                                   ! Henry
  !                                         !
  ! \f                                      ! \f
  ! 1,$p                                    ! 1,$p

    Famous Quote                            Famous Quote

    "Now is the time                        "Now is the time
    For all good men                        For all good men
    To come to the aid                      To come to the aid
    Of their country."                      Of their country."

    Henry                                   Henry
```

CHANGE REQUEST

The **change** request recognized by the text editor is a combination of the delete request and the insert request. As a single request, it deletes the line to be changed (making the following line the current line) and inserts the new line (before the current line). This request uses the lowercase letter "c" and, because it is an input request, it also requires the escape sequence for termination. The default address for a change request is the current line.

As examples of the use of the change request, consider the heading and signature of the quotation. For the signature, change "Henry " to "Patrick Henry", indenting this change by three spaces. (Remember that the current line is a blank line at the end of the contents of your buffer.) Type "–1c", "Patrick Henry" (preceded by three spaces), and "\f". Then, change the heading from "Famous Quote" to "FAMOUS QUOTATION". Type "/Fam/ c", "FAMOUS

QUOTATION", and the escape terminator. Finally, request a print of the entire contents of your buffer. (Note that the change request was used here as an example although, of course, the substitute request could have been used instead.)

```
  Henry

! -lc
!    Patrick Henry
! \f
! /Fam/ c
! FAMOUS QUOTATION
! \f
! 1,$p

  FAMOUS QUOTATION

  "Now is the time
  For all good men
  To come to the aid
  Of their country."

     Patrick Henry
```

Two-part addresses can be used with the change request. For example, if you request "9,12c", the editor will cause lines 9, 10, 11, and 12 to be replaced by your new input.

OVERWRITING

Recall that in Section 3 you read "pat_says" into your buffer. Then you read "Mary" into the contents of your buffer. Neither file in permanent storage was altered by these two requests. However, recall that when you later requested the editor to write "Mary", you found that your existing file "Mary" in permanent storage had been replaced by a new file. The process was called **overwriting.**

The write request can again be used to overwrite (replace) the existing permanent storage file "pat_says" with this new version presently in your buffer. At your option, you can either use the file name in this write request or omit it. That is, you can either request "w" or "w pat_says". The text editor "remembers" that you read the file "pat_says" into your buffer, and if you simply type "w" as your request, the editor will "know" that you mean to replace the file with the same name in permanent storage. Type "w", delete the contents of your buffer, and confirm that your buffer is empty. Then read "pat_says" and request a print of the contents of your buffer to confirm that your permanent file was indeed overwritten by the contents of your buffer prior to the delete request.

```
                 Patrick Henry

          ! w
          ! 1,$d
          ! 1,$p
            Buffer empty.
          ! r pat-says
          ! 1,$p

            FAMOUS QUOTATION

            "Now is the time
            For all good men
            To come to the aid
            Of their country."

                 Patrick Henry
```

SPECIAL CHARACTERS

You have become familiar with the use of some special characters recognized by either the Multics system or the text editor or both. You know, for example, that the Multics system recognizes both the "#" symbol to erase a single character or all the white space preceding the symbol and the " @" symbol to kill an entire line. Because the text editor is a Multics command, it, too, will recognize both symbols. Other characters, such as the period and the dollar sign, have special meaning to the text editor.

Erase and Kill

Both the "#" and " @" symbols work only on the line on which they are typed. Once the carriage return has been pressed, you cannot use either "#" or "@" to repair the line. The "#" character is used to erase either a single character or white spaces. The "@" symbol kills all the characters that precede it on a line. Therefore, you must be careful in order not to eliminate wanted text. The text editor recognizes both the erase and kill symbols in both the input and edit modes.

To see why you must be careful when you use these symbols, clear your buffer and input the example, typing the lines *exactly* as shown. Be sure to place two spaces between "Monday" and "is" and three spaces between "is" and "n". Request a print.

```
                              Patrick Henry

                          ! 1,$d
                          ! a
                          ! Monday  is####
                          ! is a@
                          ! wirj@work day.  Today
                          ! is   n## not.
                          ! \f
                          ! 1,$p
                            Monda

                            work day.  Today
                            is not.
```

Note the following:

1. Line 1 of your input, with four "#" symbols, has one erase symbol too many. One "#" is sufficient to eliminate both white spaces as shown by the example of the fourth line ("is not"). In line 1, the first "#" erased the "s", the second erased the "i", the third erased *both* white spaces, and the fourth "#" erased the "y" in "Monday".
2. Line 2 is a blank line. Had more text been placed after the "@" symbol, that text would have appeared in the buffer (as shown in line 3).
3. In line 4, one "#" erased the "n" it follows, and one "#" erased all three white spaces preceding the "n". A space was then added after the last "#" to replace one of the erased white spaces.

Period

You have used the period as a special character to print the current line of the buffer. The period has another meaning to the editor: it matches any single character on a line.

For example, suppose you wished to replace the period in the current line (line 4) with an exclamation point by the use of the substitution request, "s/./!/". Type this request and observe that the editor has matched all characters (seven including the space) in the line "is not." with exclamation points. (If you had requested "s/not./not!/" instead, the editor would have made the substitution you actually wanted and would have printed "is not!".)

```
  is not.
! s/./!/ p
  !!!!!!!
```

Use the change request to change your current line from exclamation points to its original form. Also change line 2 to "is a".

```
  !!!!!!!
! c
! is not.
! \f
! 2c
! is a
! \f
! 1,$p
  Monda
  is a
  work day.  Today
  is not.
```

Because the period matches any character on a line, it can easily be used to save typing time in both locate and substitution requests. For example, to locate the line "is not." (rather than "is a"), the locate request could be typed as "/i..n/" where the periods match the "s" and space on the line. That is, the periods match any two characters following "i" and preceding "n" anywhere on a line. In a substitution request, the periods can be used in the same manner as in "/Tod/ s/w..k/week/". Type both requests and request a print for the last one.

```
  is not.
! /i..n/
  is not.
! /Tod/ s/w..k/week/ p
  week day.  Today
```

Asterisk

The asterisk (*) is a special character recognized by the editor to mean any number (including none) of characters preceding it. That is, if it were used in a locate request such as "/ab*c/", the editor would search for a line that had either "ac" (no "b"), "abc", "abbc", "abbbc", or "abbbbc", etc. in it.

The asterisk is most often used in combination with the period. Because the period matches any character on a line and the asterisk matches any number of its preceding characters, the combination of the period and the asterisk will match anything. For example, to locate "week day", your locate request could be either "/week day/" or "/w......y/"; or, you can locate the expression by using the period and asterisk combination "/w.*y/". If you were to use the asterisk, you would eliminate the effort of counting the number of periods you would need if you were to use only periods. Type this locate request using the period and asterisk.

```
  week day.  Today
! /w.*y/
  week day.  Today
```

You can likewise use the asterisk in a substitution request. For example, type "s/w.*y/holiday/ p" and observe that the line is altered.

```
  week day.  Today
! s/w.*y/holiday/ p
  holiday.  Today
```

The asterisk always requires a preceding character. If it is tried alone within delimiters, the result will be an error message from the editor. Type "/*/".

```
    holiday.  Today
  ! /*/
    Invalid use of * in regular expression.
```

```
    holiday.  Today
  ! /*/
    Line search failed. "/*/"
```

Dollar Sign

The dollar sign ($) has two meanings as a special character. When it is used as part of an address, it means the last line of the contents of your buffer. When it is used within delimiters in the "old" part of a substitution request, it means the imaginary character following the last character on a line.

The dollar sign can be used to advantage with the period and asterisk. For example, to change the last line of the present contents of your buffer from "is not." to "is a work day.", type "$s/i.*$/is a work day./ p". Recall that "$s" means to substitute in the last line of your buffer. That is, request the editor to substitute the new regular expression "is a work day." on the last line of the buffer beginning with the letter "i" and continuing for any character on the line (.), for any number of those characters on the line (*), through the last character on the line ($).

```
    Invalid use of * in regular expression.
  ! $s/i.*$/is a work day./ p
    is a work day.
```

```
    Line search failed. "/*/"
  ! $s/i.*$/is a work day./ p
    is a work day.
```

For line 2, type "2s/i.*$/". Now request a print of the contents of your buffer and observe that the regular expression "is a" has been replaced by a blank line because the null expression in the "new" part of the substitution request caused the editor to substitute a blank line.

```
  is a work day.
! 2s/i.*$//
! 1,$p
  Monda

  holiday.  Today
  is a work day.
```

NOTE

You should exercise caution when experimenting with these context addressing forms in order to ensure that the intended text string is indeed the one found by the text editor. Observe that line 3 of the buffer contains two occurrences of "/o.*day/" but only one of "/oday/".

Circumflex

The circumflex (^) is recognized by the text editor to mean the imaginary character before the first character on a line. It is, therefore, useful in a locate request to find a line beginning with a particular expression as discussed in Section 3. However, because the dollar sign means the imaginary character after the last character on a line, the combination of these two special characters can be used to search for a blank line. For example, type "/^$/" and observe that the editor replies with a newline character (the carriage return only). Type "=" and note that this last request has located line 2, which is indeed blank.

```
  is a work day.
! /^$/

! =
  2
```

Ampersand

The ampersand symbol (&) is a special character recognized by the editor in "new" parts of substitution requests. (Remember the format of the substitution request is "s/old/new/".) The ampersand is used to replace parts of the "old" in the "new" expression. The ampersand has a special meaning only in the "new" part of the substitution request and not in the "old" part.

For example, you may request the editor to add the letter "y" to the first line of the contents of your buffer by typing "1s/Monda/&y/". Thus, the locate request will find line 1 and the regular expression "Monda". The first part of the "new" part of the substitution request is identical to the "old" as indicated by the "&". The letter "y" is then requested to be placed following this regular expression. Type "1s/Monda/&y/ p". (Note that an easier way of correction is "1s/$/y/".)

```
  2
! 1s/Monda/&y/ p
  Monday
```

Forward Slash

Forward slash marks (/) are used in substitution requests as delimiters more often than other characters because they are easy to see and do not occur in text too often. These special characters must be used as the delimiters for the locate request. Recall, however, that any character that does not appear in either the "old" or the "new" parts of the substitution request can be used as delimiters in the substitution request.

SPECIAL CHARACTER ESCAPE SEQUENCE

The contents of your buffer may at various times contain any of the special characters recognized by the editor. Editing requests using regular expressions involving these characters can easily lead to errors and unwanted results. For example, the locate request requires slash marks as delimiters. Thus, requesting the editor to locate a regular expression such as "and/or" would require "/and/or/" to which the text editor will reply with an error message.

Clear your buffer and then input the text shown in this example. Then, type "/and/or/" and observe the response. (Ted has a special "o" request that is not used here and cannot be preceded by an address.)

```
   Monday                                     Monday
 ! 1,$d                                     ! 1,$d
 ! a                                        ! a
 ! Monday is                                ! Monday is
 ! a                                        ! a
 ! work and/or play                         ! work and/or play
 ! holiday                                  ! holiday
 ! Today is not.                            ! Today is not.
 ! \f                                       ! \f
 ! /and/or/                                 ! /and/or/
   qedx: o not recognized as a request        No addrs allowed. o
```

To overcome this type of situation, you can use a special escape character that allows the character to assume its literal form in the regular expression. That escape sequence is a backslash and the letter "c". Type "/and\c/or/" and note that the search succeeds.

```
   qedx: o not recognized as a request        No addrs allowed. o
 ! /and\c/or/                               ! /and\c/or/
   work and/or play                           work and/or play
```

As another example, suppose you wanted to substitute the symbol "&" for the letters "and" in this line. Type "s/and/&/ p" and note that it appears no substitution has been made although the editor actually carried out the request because "&" means "identical to the old".

```
   work and/or play
 ! s/and/&/ p
   work and/or play
```

Now, type "s/and/\c&/ p" and observe the result. Thus, to distinguish between the times you want the editor to recognize the character as literal rather than as special, you should prefix the character with the escape sequence "\c".

```
  work and/or play
! s/and/\c&/ p
  work &/or play
```

The "#" and "@" symbols require special care because they carry meaning within the entire Multics system and are not special to the text editor alone. The escape for these special symbols is a backslash only. Thus if a line in your file were to read "Six #2 cans @ .69 each.", typing the # symbol would, of course, erase the space preceding it and then typing @ symbol would erase the entire line except for " .69 each.".

Clear your buffer and input "Six #2 cans @ .69 each."; then request a print to see that only ".69 each." remains. Again clear your buffer, but this time, input "Six \#2 cans \@ .69 each.". Request a print and observe that your input has now been accepted as you desired. (Note that if "\c" is used to precede "#", the "#" would simply erase the "c".)

```
  work &/or play
! 1,$d
! a
! Six #2 cans @ .69 each.
! \f
! p
   .69 each.
! d
! a
! Six \#2 cans \@ .69 each.
! \f
! p
  Six #2 cans @ .69 each.
```

PROGRAM INTERRUPT

All terminals have a key marked **ATTN, BRK, INTRPT, INTERRUPT, BREAK,** or some similar name to mean: "Interrupt processing my work immediately". When this key is pressed, the work in process or being executed at that time is immediately interrupted, but the work remains preserved.

As soon as the system receives a signal from this interrupt key, the Multics system stops executing the work in progress and prints "QUIT", which is followed by a ready message. This QUIT signal is not to be confused with the quit *request*, "q". The quit request gets you out of the text editor and back to command level, simultaneously emptying the contents of your buffer. The QUIT *signal* puts you at command level, but your buffer is not emptied.

The QUIT signal can be used to advantage when you are in the text editor. For example, suppose you have a file of several hundred lines of which you wish to see only a few. Suppose further that instead of addressing a print request for only these few lines, you mistakenly type "1,$p". To halt the print of the entire contents of your buffer, press the interrupt key on your terminal. The print request is immediately stopped at the point the interrupt key is pressed. The Multics system prints "QUIT" and a ready message.

This ready message has a form different from those you have seen so far. This ready message contains the word "level" and a number. The level information indicates that you are at a new Multics command level (as indicated by the number following "level"), but that the interrupted work is being preserved. Because the Multics system is at command level, it is available for a new command, such as "list" or "compose", which are described later, or it can be commanded to continue processing the interrupted work in the text editor.

In the case of the editor, you may continue the interrupted work by typing "program_interrupt" or its short name "pi" after the ready message. Typing this command returns you to the editor, and it now awaits your next request just as if it had finished the "1,$p" request and you wanted to continue editing the contents of your buffer.

As an example, clear your buffer and read in your file "Mary". Request a print of this file, but in the middle of the third line, press your interrupt key (e.g., **BRK, INTRPT**). Note that the Multics system immediately interrupts the print, prints the word "QUIT", and then a ready message with level information.

```
  Six #2 cans @ .69 each.
! 1,$d
! r Mary
! 1,$p
  "Now is the time
  For all good men
! Mary
  QUIT
  r 09:10 0.166 20 level 2
```

NOTE

In some cases, the word "QUIT" will not be printed. Instead, you may see other symbols preceding the ready message. Likewise, the "QUIT" and the ready message may be on the same line as the line interrupted.

To continue editing, type "pi". Then type "." to show your current line. Observe that the current line is the last line of the contents of your buffer just as if the editor had actually completed the "1,$p" request.

```
  r 09:10 0.166 20 level 2

! pi
! .
  Of their country."
```

Continue your print by requesting "/Mary/,$p", and again press your interrupt key before the end of the print. Once more, your work is interrupted and a ready message is given.

```
  Of their country."
! /Mary/,$p
  Mary had a little lamb
! Its fleece was white
  QUIT
  r 09:10 0.618 44 level 2
```

PENDING WORK IN QEDX

If the text editor you are using is Ted, disregard these next paragraphs and continue on to the next heading.

You can see that using the interrupt has not changed the contents of your buffer. However, suppose that you either forget that you were in the text editor or you type "qx" by mistake. Type "qx" after this last ready message and observe the printed response. Type "yes" in answer to the question. Then request a print of the contents of your buffer and observe that the previous work (the contents of your buffer) apparently has been lost. The buffer is empty.

```
    r 09:10 0.618 44 level 2

  ! qx

    qedx: Pending work in previous invocation will be lost if you proceed;
  ! do you wish to proceed?  yes
  ! 1,$p
    Buffer empty.
```

Suppose that you did not want the "pending work" to be lost because you had not completed your input, you wanted to complete a print request, or you had made several editing changes and had not yet given a write request. In this case, the answer to the question from qedx should be "no."

Again read "Mary" into your buffer. Remember that you are still in qedx as evidenced by the resonse of qedx ("Buffer empty.") to your last "1,$p" request. Again, request "1,$p" and press your interrupt key before the request has been completed. Observe that you move from level 2 to level 3.

After this ready message, attempt to invoke qedx and now type "no" to the question prompted by qedx. At the answer "no", you will receive a ready message after which you can give the Multics system a new command. However, qedx and the contents of your buffer are still at hand. Type "program_interrupt", or preferably the short name "pi", in order to continue editing the contents of the buffer. To see that the contents of your buffer have not been lost, type "." to see your current line.

```
  Buffer empty
! r Mary
! 1,$p
  "Now is the time
  For all good men
! Mary had a little
  QUIT
  r 09:11 0.646 53 Level 3

! qx

  qedx: Pending work in previous invocation will be lost if you proceed;
! do you wis to proceed?  no
  r 09:11 0.156 16 level 3

! pi
! .
  Of their country."
```

PENDING WORK IN TED

If the text editor you are using is qedx, disregard these next paragraphs and continue to the next heading.

You can see that using the interrupt has not changed the contents of your buffer. However, suppose that you either forget that you were in the text editor or you type "ted" by mistake. Type "ted" after this last ready message and then request a print of the contents of your buffer and observe that the previous work (the contents of your buffer) apparently has been lost. The buffer is empty.

```
  r 09:10 0.618 44 level 2

! ted
! 1,$p
  Buffer empty
```

Again read "Mary" into your buffer. Remember that you are still in Ted as evidenced by the response of Ted ("Buffer empty.") to your last "1,$p" request. Again, request "1,$p" and press your interrupt key before the request has been completed.

Observe that you have moved from level 2 to level 3. However, Ted and the contents of your buffer are still at hand. Type "program_interrupt", or preferably the short name "pi", in order to continue editing the contents of the buffer. To see that the contents of your buffer have not been lost, type "." to see your current line.

```
  Buffer empty.
! r Mary
! 1,$p
  "Now is the time
  For all good men
! Mary had a little
  QUIT
  r 09:11 0.646 53 level 3

! pi
! .
  Of their country."
```

RELEASING WORK HELD

There is one command allied to the QUIT signal that has yet to be defined. That command to the Multics system is the **release** command. Each time you interrupt the work in progress, the Multics command level is changed. When you log in, you are at the first level (no level information in the ready message). An interrupt at this level takes you to level 2; an interrupt at level 2 takes you to level 3, etc. (unless you have commanded "program_interrupt").

You are presently in the text editor in level 3. You can return to level 2 by simply giving the quit editor request. Type "q" and note that you receive a ready message reporting level 2. To return to the first level, the command to the Multics system after the ready message is "release" or its short name "rl". Type "rl" and observe that you are now back at your initial first level.

```
  Of their country."
! q
  r 09:12 0.020 4 level 2

! rl
  r 09:12 0.000 0
```

A control argument of "−all" or its short name "−a" will release all levels and, for example, bring you immediately from level 3 to the first level. When this command is used, it is typed as **release −all** or "rl −a" and releases all the levels (all the work being held).

However, you may not want to release all levels. For example, suppose you have invoked the editor but you would also like a list of your files. The interrupt key can be used to bring you to level 2 where you can give a "list" command (described next) while preserving the work in your buffer. You may, at this point, use the interrupt key again to go to the next higher level to give a third command (such as the "compose" command to be described later). If, at the completion of this last command, you command "rl −a", you would be returned to the initial command level and you would lose the work in your buffer. But, by giving "rl" commands, you would be placed at the level where you could command "pi" and continue in the editor.

LISTING AND DELETING FILES

Presently, and unless you have added other segments, your permanent storage area contains two files: one under the pathname "pat_says" and one with the pathname "Mary". These files were created and edited to show how to use the text editor in creating and editing document segments.

Although you may have only these two files in permanent storage, you may easily have several more. Some you will want to retain, especially longer ones that are to be updated periodically, such as pages of proposals that, with minor changes, can be used in other proposals and thus read into your buffer with new proposal input.

It is not uncommon to forget how many files you may have or the pathnames of those files. Also, because all files take up valuable space in the permanent storage area of your computer, those files that are not needed should be removed. The Multics system recognizes commands for both situations.

The Multics system recognizes the command **list** or its short name "ls" to give you a list of all your files in permanent storage. Typing this command after a ready message will cause the system to print the number of files you have and their pathnames. You will also be given the lengths of those files and whether you have read or write access or both.

Type "ls" after the ready message. Unless you have added files to those described here, Multics will respond showing only two segments in permanent storage. The segment entered last (or overwritten last) will always be at the top of the list.

```
  r 09:12 0.000 0

! ls

  Segments = 2, Lengths = 2.

  r w    1  pat_says
  r w    1  Mary

  r 09:12 0.272 16
```

You no longer need the files listed. Therefore, you should remove them from permanent storage. To remove them, the Multics system recognizes a command **delete** whose short name is "dl". (Do not confuse this command with the delete request that is used in the text editor.) To delete the segment "pat_says", type "dl pat_says" after the ready message. To delete "Mary", type "dl Mary". Alternatively, both files can be deleted from permanent storage by typing on one line "dl pat_says Mary" with spaces between the command and the pathnames.

```
  r 09:12 0.272 16

! dl pat_says
  r 09:13 o.650 25

! dl Mary
  r 09:13 0.462 21
```

Invoke the editor and read "Mary". Observe that the text editor cannot find the file. Read "pat_says" and note that the text editor cannot find this file either. Quit the editor and, after the ready message, command a list of your files and note that neither file "pat_says" nor "Mary" are in permanent storage.

```
  r 09:13 0.462 21                  r 09:13 0.462 21

! qx                              ! ted
! r Mary                          ! r Mary
  qedx: Entry not found. Mary       Entry not found. >udd>ProjA>TSmith>Mary
! r pat_says                      ! r pat_says
  qedx: Entry not found. pat_says   Entry not found. >udd>ProjA>TSmith>pat_says
! q                               ! q
  r 09:14 1.004 59                  r 09:14 1.004 59

! ls                              ! ls

  Segments = 0, Lengths = 0.        Segments = 0, Lengths = 0.

  r 09:14 0.246 14                  r 09:14 0.246 14
```

TEXT EDITOR EXERCISE

This section, as well as Sections 2 and 3, discussed exclusively the use of the text editor. You have been shown that a file can be created and stored. You have also been shown a number of requests to the text editor that permit you to alter that file. As a final exercise, type the following example as input in as near to the form shown as you can. (Note the blank lines at the beginning and separating the title lines.) Use the edit requests to correct any and all typing errors. Finally, write the file in permanent storage using any pathname you desire except that the pathname must have the suffix "fdocin" with a period, as in "Preamble.fdocin". (There should be no spaces in the request except after the letter "w", which is an optional space.)

```
   r 09:14 0.246 14

 ! qx         ! ted
 ! a
 !
 ! CONSTITUTION
 ! OF THE UNITED STATES
 ! OF AMERICA
 !
 ! PREAMBLE
 !
 ! We the people of the United
 ! States, in Order to form a
 ! more perfect Union, establish Justice, insure
 ! domestic Tranquility, provide for the common defense, promote the
 ! general Welfare,
 ! and secure the Blessings of Liberty to
 ! ourselves and our Posterity, do ordain and establish this
 ! Constitution for the United States of
 ! America.
 ! \f
 ! w Preamble.fdocin
 ! q
   r 09:18 0.658 58

 ! logout
```

Section 5
Formatting a File

The final exercise given in Section 4 was to input the Preamble to the Constitution of the United States of America. Your typed input deliberately consisted of several lines, some short and some long. And, to that file you supplied a pathname with a period and a suffix, "fdocin". In this section, you will see how to format that file using control lines and the Multics format_document command.

THE FORMAT_DOCUMENT (FDOC) COMMAND

Log in. After the ready message, invoke your text editor. Verify that your fdocin (pronounced "f-doc-in") file is in permanent storage by requesting a read request and a print request. Quit your editor and after the ensuing ready message, type "fdoc xxxxx", substituting your chosen pathname for "xxxxx". That is, if your chosen pathname were "Preamble", you would give the command "fdoc Preamble" to the Multics system.

After a short hesitation, you will receive a print of this file on your terminal. At the end of the printed text, the line feed on your terminal will continue to operate. You will be given several blank lines before you will again be given a ready message. After the ready message, log out and study the results of your Multics format_document command (fdoc) for this file which you input earlier through the text editor.

```
  r 13:36 4.160 64

! qx
! r Preamble.fdocin
! p
  America.
! q
  r 13:37 0.008 3
```

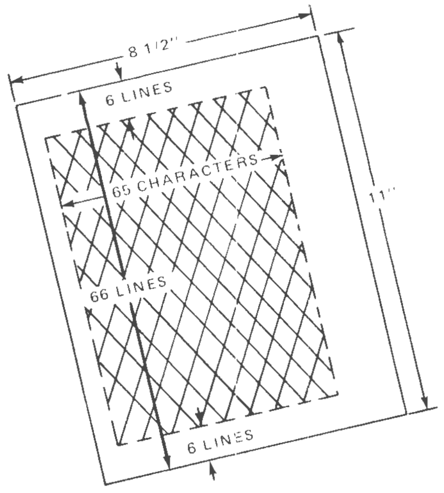
```
  r 13:36 3.160 64

! ted
! r Preamble.fdocin
! p
  America.
! q
  r 13:37 0.008 3
```

```
! fdoc Preamble



 CONSTITUTION OF THE UNITED STATES OF AMERICA

 PREAMBLE

 We  the people  of the United States,  in Order  to  form  a more
 perfect  Union, establish  Justice,  insure domestic Tranquility,
 provide for the common defense,  promote the general Welfare, and
 secure the  Blessings of Liberty to ourselves  and our Posterity,
 do ordain  and establish this Constitution for  the United States
 of America.




 r 13:37 7.842 112

! logout
```

AUTOMATIC FDOC CONTROLS

Probably the most obvious result of the fdoc (pronounced "f-doc") command for your fdocin file is that the text margins have been justified both right and left although your text editor input consisted of lines of various lengths. Short lines such as your heading lines and the last line of the text are justified on the left. The next most obvious result is that the output printed on your terminal fits an 8½ by 11 inch sheet of paper.

Without any effort on your part, the fdoc command has counted 66 lines for a page. Of those 66 lines, the fdoc command has given six blank lines between the top of the page and the first line of text. Six lines are also left blank between the last line of the text and the bottom of the page. Thus, the fdoc command automatically provides for 54 lines of straight text on a page.



In this formatting example, your first line of input to your editor for your fdocin file is a blank line; it becomes the seventh line from the top of the page in the formatted output. The input line "CONSTITUTION" is printed on the eighth line of the output. Less apparent, of course, are the six blank lines at the bottom of the page (before the ready message) because there is insufficient text in the file to fill the page.

Also, without any undue effort on your part, the fdoc command automatically caused 65 characters (including spaces) to be formatted on each line, and in addition, caused both the left and right margins of the text to be justified.

FORMAT_DOCUMENT CONTROL LINES

The result of this format_document command for the "Preamble.fdocin" file is an output of relatively fixed format. Of course, you may not want the right margin justified, or you may prefer that the paragraph be indented. Your formatted output example was produced by **control lines** the fdoc command uses by default.

"By default" implies that you may change the input fdocin segment to alter the format of the output material. You make those changes for the output format by inserting control lines within the fdocin segment. Those control lines are used to format the text to indent the left margin, indent for paragraphs, undent for a heading, align both margins or only the left margin, and control page width and length.

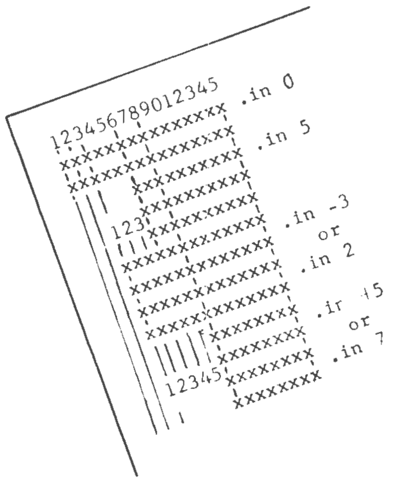
Control lines for formatting are inserted as separate lines in your fdocin file through the text editor. These control lines consist of a period followed by two or three lowercase letters and often a number. The period must always be typed in the first column position of your input fdocin file.

INDENT CONTROL LINE

The first control line you will use to alter your formatted output is the **indent** control line. You type this control line as a line of input within your text editor as ".in N", where "N" is the number of character spaces you want your text indented to the right from the left margin of the formatted output. That is, the indent control serves much like the manual left margin setting on an ordinary typewriter. When you use this control line, be sure to leave a blank space before "N".

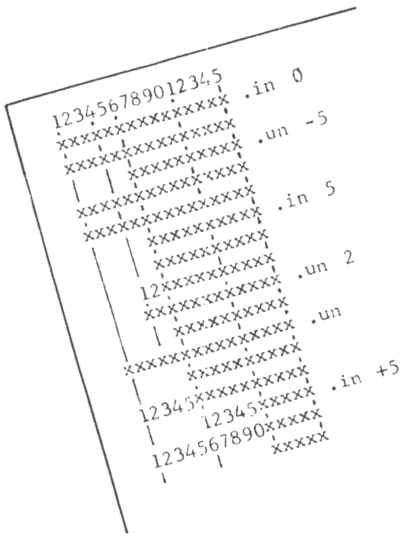
The default indent control line is ".in 0" which may be typed as ".in" (without a digit zero) and which means: "Indent zero spaces from the left and start each line of text at column position 1." In your fdocin example, the fdoc command used this default value so that all your formatted lines began at column position 1. If you had typed an indent control line at the beginning of your input fdocin segment as ".in 5", for example, all lines of your output would have been indented five spaces from the left margin and each line of text would have had its first character printed in column position 6 of the 65 column positions on a line.

The indent control line applies to all lines of your formatted output that follow it. However, you can change it for any succeeding line or any number of succeeding lines. The number of indent spaces can be either absolute, with respect to column position 0, or relative, with respect to the current left margin. Relative values require plus (to the right) or minus (to the left) signs with respect to the current left margin. Absolute values are given without the sign.



Suppose, for example, that you have input an indent control line such as ".in 5" (absolute number) for several lines of text so that the first character of each line is printed in column position 6. Suppose you now wish the first output character of each line of another block of text to be printed in column position 3. For this change, you may input either ".in 2" (absolute) or ".in −3" (relative to position 6). If you next decide you want lines of a third block of text to have each line start at column position 8, you may input either ".in 7" or ".in +5" (five spaces to the right of the current left margin).

UNDENT CONTROL LINE



The **undent** *control line applies only to the next single line of output following this control line.* The control line is of the form ".un N" with a space before "N" and where "N" is the number of spaces undented from the value specified in the indent left control line. In other words, the undent control line is used to change the indent value of the formatted output — but *only* for the first line of text output following this control line, as, for example, to indent five spaces to start a new paragraph.

If this control line is given as, for example, ".un +5" or ".un 5" (where the plus sign is assumed by the fdoc command), the text line following the control line would be started five spaces to the left of the current indent value (for a subheading, for example). To indent a paragraph, the undent control line might be given as ".un −5" which means: "Undent the following line five spaces to the right from the column position specified by the last indent control line." Notice that the signs of the values of the undent control line are opposite to the relative directions for the indent control line. For undent control lines, plus is always toward the margin and minus is toward the center of the line.

FILL AND ALIGN CONTROL LINES

Unless you insert control lines in the fdocin segment to indicate otherwise, the fdocin command fills in all lines of text with white space so that both left and right margins are justified. In other words, the fdoc command uses the **fill on** control line. When the fill control is on, text words (and extra white spaces) are moved within a line and from line to line so that the last word in the line does not extend past the right margin. In fact, each line is filled so that the last character of the last word on the line is in the 65th character position. There are times, of courses, when you would prefer that the formatted output be line-for-line exactly like your input or that the right margin be ragged.

As as example, the text of your original fdocin segment from Section 4 consists of both short and long lines. If you had wanted the formatted output to appear the same way you would have had to cancel the fill on control line by inserting the **fill off** control line, ".fif", before this body of text. The fill on control line, which is input as ".fin" is the default control line.

The **align** control line works in conjunction with the fill controls. That is, you cannot have both left and right margins justified when the fill control is off. Consequently, the fill on control is usually used except in the case of some inserts such as notes or data in a table. For such material, you would insert ".fif" before the material and then insert ".fin" after the material.

The default align control line is ".alb" meaning: "Align both left and right margins." To align the text at the left margin only (with the right margin remaining ragged), the control line is ".all".

PAGE LENGTH

The fdoc command, by default, automatically formats for 66 lines to a page (11 inches on a 10-pitch typewriter-like terminal or a line printer). You may on occasion have reasons to change the page length. The page length can be changed by the **page length** control line within your segment. That control line is ".pdl N", where "N" is the desired number of lines to a page. By default, the fdoc command recognizes ".pdl 66".

Remember that the fdoc command places six blank lines at the top of the page and six blank lines at the bottom of the page. If a page number is to be incorporated, the fdoc command can be used to add an additional blank line and a line for the page number.

PAGE WIDTH

The fdoc command, by default, automatically formats 65 characters to a line. The number of characters per line can be adjusted by inserting a **page width** control line within your segment. That control line is ".pdw N", where "N" is the desired number of characters to a line. By default,the fdoc command uses ".pdw 65".

NOTE

If you want to use a nonstandard page size, then ".pdl" and "pdw" control lines *must* be placed in the fdocin segment at the *beginning* of your input.

USING FDOC CONTROL LINE

The eight control lines given above for formatting a document are the only control lines available for formatting an fdocin segment. These control lines permit simple formatting of letters and memos; for more elaborate document formatting to include centered headings, footnotes, etc., you will need to use the Multics compose command.

To show how the fdoc control lines are used, you will invoke the text editor and read your fdocin file. You will insert a fill off control line to give a heading of three lines in the formatted output. You will indent this three-line heading 30 spaces while indenting the remainder of the segment only 15 spaces. You will also indent the first line of the Preamble five spaces with respect to the rest of the body of the text.

Log in, invoke the text editor, and read your fdocin file. Request a print of the first few lines of your file to refresh your memory as to its contents.

```
   r 13;38 0.346 28                            r 13;38 0.346 28

 ! qx                                        ! ted
 ! r Preamble.fdocin                         ! r Preamble.fdocin
 ! 1,/We/p                                   ! 1,/We/p

   CONSTITUTION                                CONSTITUTION
   OF THE UNITED STATES                        OF THE UNITED STATES
   OF AMERICA                                  OF AMERICA

   PREAMBLE                                    PREAMBLE

   We the people of the United                 We the people of the United
```

NOTE

The following editing changes are given as only one method of several that might be used. For example, you could delete all file lines from line 1 to "We" and append or insert all new input rather than performing the procedures given.

Because the heading is to be indented 30 spaces, the first editing request is to append ("0a") or insert (1i") an indent left control line before the first input line of the heading. Thus, insert ".in 30" before line 1 of your fdocin file.

```
   We the people of the United
 ! 1i
 ! .in 30
 ! \f
```

This fdoc control line will indent all lines of text 30 spaces from the left from line 1 to the end of the file unless the indent control line is changed. Because only the heading is to be indented 30 spaces and the remainder of the file is to be indented 15 spaces, another indent control line must be inserted in the file. For this input, locate "AMER" and append ".in 15" after this context address.

```
  \f
! /AMER/a
! .in 15
! /f
```

As you saw when you first gave the fdoc command and as stated in the discussion on fill and align control lines, the fdoc command by default moves text words and white spaces in an attempt to justify each line. The fdoc command by default attempts to justify all input lines between blank lines. Although the heading in your fdocin file is given as three lines, the fdoc command combined those lines to form only one.

To keep your heading as three lines requires inserting control lines in your fdocin file. You have two choices. First, when any control line (other than .pdl and .pdw control lines) is inserted in the fdocin file, the fdoc command will automatically stop filling a line at that point. For example, if you were to insert a control line in your fdocin file one line immediately after the line "CONSTITUTION", the fdoc command would print "CONSTITUTION" and stop printing the line at that point and not add "OF THE UNITED STATES OF AMERICA" as you saw. The next line would then contain "OF THE UNITED STATES OF AMERICA" unless a control line were inserted on a line in your fdocin file after "OF THE UNITED STATES". Thus, you could insert ".in 30" in your file after "CONSTITUTION" and after "OF THE UNITED STATES" to produce a three-line heading. (Using ".in 30" preserves the 30-space indentation for the lines of the heading.)

Your second choice is to turn the fill control off for these three lines so that they are printed as they appear in the fdocin file. This choice is the one used here. Thus, at the beginning of your fdocin file, insert

".fif" (by either appending before the first line by requesting "0a" or inserting before the first line by "1i"). Then, turn the fill control on after "OF AMERICA" by appending ".fin".

```
  \f
! 0a
! .fif
! \f
! /OF A/a
! .fin
! \f
```

The last control line to add is the control line to indent the first word of the body of the text five spaces. The undent control line is used for this purpose. Thus, immediately before "We", insert ".un –5".

```
  \f
! /We/ i
! .un -5
! \f
```

The examples show these editing changes. There are, of course, several other ways of editing this file to insert the required fdoc control lines. Request a print of your buffer contents to confirm all the editing changes. Then, overwrite those changes in your permanent fdocin file by using the write request to your text editor.

```
  \f
! 1,/We/p
  .fif
  .in 30

  CONSTITUTION
  OF THE UNITED STATES
  OF AMERICA
  .fin
  .in 15

  PREAMBLE

  .un -5
  We the people of the United
! w
```

The fdoc command will not recognize any input file not having the "fdocin" suffix with the period as part of the file name. Consequently, when you invoke the fdoc command, you do not need to type the suffix ".fdocin". For your present file, the system will give a formatted output when the command line is only "fdoc Preamble".

If you make errors in typing the control lines, the system will provide an error message. You will receive a message at the point in your formatted output where the error occurs. For example, suppose you insert ".fim" rather than ".fin" after "OF AMERICA" to turn the fill control back on. Not only will the fdoc command disregard this control line (you will see the effect in the formatted output) but also at that point in your formatted output, you will be given the line number of the control line and the control line in error. At the end of the printed output just before the ready message, you will receive a message similar to the following: "format_document: Requested operation completed but non-fatal errors or inconsistencies were encountered. >user_dir_dir>ProjA>TSmith>Preamble.fdocin".

Quit the text editor and issue the fdoc command to get a formatted output of your fdocin file. Type "fdoc Preamble". Note that your heading consists of the same three lines of your input fdocin file with these lines indented from the left by 30 spaces. Also note that the subheading and the body of the text are indented by 15 spaces and the first line has a paragraph indentation of 5 spaces.

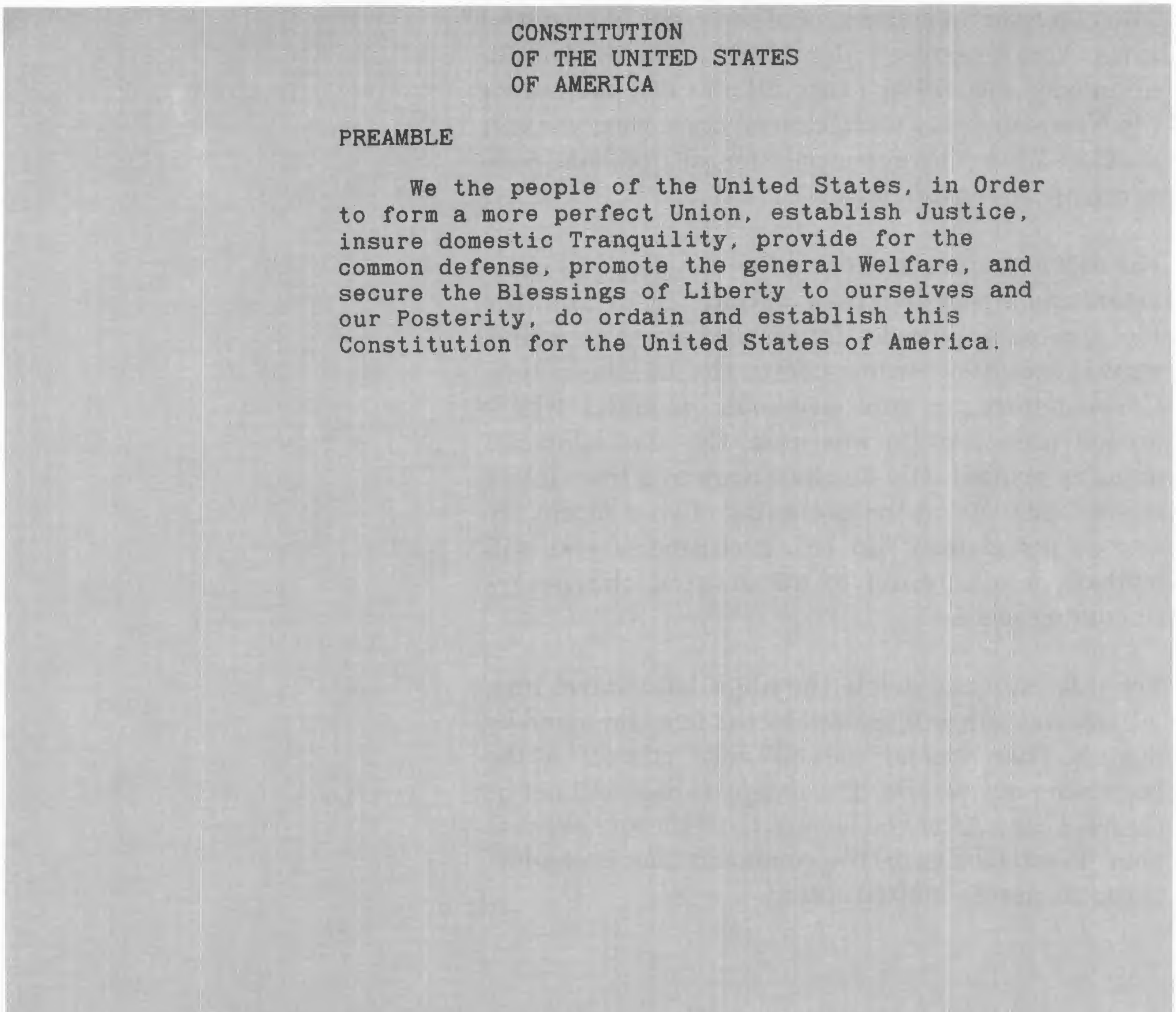
```
  w
! q
  r 13:44 1.250 116

! fdoc Preamble






                         CONSTITUTION
                         OF THE UNITED STATES
                         OF AMERICA

                 PREAMBLE

                       We the people of  the United States, in Order
                  to form  a more perfect  Union, establish Justice,
                  insure  domestic  Tranquility,  provide  for  the
                  common defense,  promote the general  Welfare, and
                  secure the  Blessings of Liberty  to ourselves and
                  our  Posterity,  do  ordain  and  establish  this
                  Constitution for the United States of America.
































  r 13:44 9.908 114
```

To see further how control lines are used, again invoke your text editor and read your fdocin segment. You are now going to justify only the left margin of the formatted output. Insert the align control line ".all" at the beginning of your file. Although you need the fill control on for the align control line to be effective, you will not delete the ".fif" and ".fin" control lines that give your three-line heading. Placing the align control line ".all" at the beginning of the file causes the fdoc command to recognize that control throughout the file whenever the fill control is on. Do not forget the write request to overwrite this change to your fdocin segment.

```
  r 13:44 9.908 114                      r 13:44 9.98 114

! qx                                   ! ted
! r Preamble.fdocin                    ! r Preamble.fdocin
! 0a                                   ! 0a
! .all                                 ! .all
! \f                                   ! \f
! w                                    ! w
```

Quit the text editor and use the fdoc command to format your fdocin file. As you can see, only the left margin is justified, with the right margin ragged. The fill on control automatically has moved the text words from line to line in such a way that the last word on a line does not extend past the right margin (past column position 65) but white spaces have not been inserted to cause the right margin to be justified as is the left.

```
  w
! q
  r 13:47 0.484 48

! fdoc Preamble
```



```
                    CONSTITUTION
                    OF THE UNITED STATES
                    OF AMERICA

            PREAMBLE

                 We the people of the United States, in Order
            to form a more perfect Union, establish Justice,
            insure domestic Tranquility, provide for the
            common defense, promote the general Welfare, and
            secure the Blessings of Liberty to ourselves and
            our Posterity, do ordain and establish this
            Constitution for the United States of America.















r 13:47 2.780 73
```

Thus far, you have used six of the eight fdoc control lines. You have used align both (.alb) by default, align left (.all), fill on (.fin), fill off (.fif), and indent (.in N) and undent (.un N) control lines. Next you will see the effects of using the page length (.pdl) and page width (.pdw) control lines.

For this example, consider a line length of 50 characters and a page length of 35 lines. By default, the fdoc command provides for 65 characters on a line. Any indentation deducts from the 65 characters. Consequently, in your previous examples where an indentation of 15 was used, the fdoc command actually printed only 50 characters to a line. If you insert ".pdw 50" at the beginning of your fdocin file and do not change ".in 15", each line of text will contain a maximum of 35 printed characters (including spaces).

For this example, delete the align left control line, .all, so that .alb will be used by the fdoc command by default. Then append ".pdl 35" and ".pdw 50" at the beginning of your file. The indent values will not be changed here. After you have entered these changes to your "Preamble.fdocin" file, command "fdoc Preamble" to see the new formatted output.

```
  r 08:27 0.217 2

! qx
! r Preamble.fdocin
! /.all/c
! .pdl 35
! .pdw 50
! \f
! 1,6p
  .pdl 35
  .pdw 50
  .fif
  .in 30

  CONSTITUTION
! w
! q
  r 08:31 0.227 35

! fdoc Preamble
```

```
  r 08:27 0.217 2

! ted
! r Preamble.fdocin
! /.all/c
! .pdl 35
! .pdw 50
! \f
! 1,6p
  .pdl 35
  .pdw 50
  .fif
  .in 30

  CONSTITUTION
! w
! q
  08:31 0.227 35
```

```
                    CONSTITUTION
                    OF THE UNITED STATES
                    OF AMERICA

     PREAMBLE

          We  the  people  of the United
     States, in Order  to  form  a  more
     perfect  Union,  establish Justice,
     insure     domestic     Tranquility,
     provide  for  the  common  defense,
     promote the  general  Welfare,  and
     secure  the Blessings of Liberty to
     ourselves  and  our  Posterity,  do
     ordain     and     establish     this
     Constitution for the United  States
     of America.




  r 08:31 0.323 11
```

As a final exercise in this section on using the Multics format_document command, you will make several changes to your fdocin file. The page length will be changed from 35 lines to 25. The heading will be indented 20 spaces rather than 30 and the body of the text will be indented only 5 spaces rather than 15. In addition, the word "CONSTITUTION" in the heading will be undented four spaces to the right while "OF AMERICA" will be undented five spaces to the right in an attempt to center the three lines of the heading.

Invoke your text editor to make these changes but do not yet give the fdoc command to see how your changes have affected the printed output.

```
  r 08:31 0.323 11                          r 08:31 0.323 11

! qx                                      ! ted
! r Preamble.fdocin                       ! r Preamble.fdocin
! /35/ s//25/                             ! /35/ s//25/
! /30/ s//20/                             ! /30/ s//20/
! /15/ s//5/                              ! /15/ s//5/
! /CON/i                                  ! /CON/i
! .un -4                                  ! .un -4
! \f                                      ! \f
! /OF A/i                                 ! /OF A/i
! .un -5                                  ! .un -5
! \f                                      ! \f
! 1,/We/p                                 ! 1,/We/p
  .pdl 25                                   .pdl 25
  .pdw 50                                   .pdw 50
  .fif                                      .fif
  .in 20                                    .in 20

  .un -4                                    .un -4
  CONSTITUTION                              CONSTITUTION
  OF THE UNITED STATES                      OF THE UNITED STATES
  .un -5                                    .un -5
  OF AMERICA                                OF AMERICA
  .fin                                      .fin
  .in 5                                     .in 5

  PREAMBLE                                  PREAMBLE

  .un -5                                    .un -5
  We the people of the United               We the people of the United
! w                                       ! w
! q                                       ! q
  r 08:43 0.072 4                           r 08:43 0.072 4
```

CONTROL ARGUMENTS

You are again ready to give the fdoc command to format your file. However, the command you will give this time will be "fdoc Preamble -pgno -ind 5". An addition to the fdoc command is called a **control argument**. Control arguments are always placed on the command line following the basic command. Control arguments always begin with a minus sign and are always separated by a space.

The fdoc command permits three control arguments: -page_numbers (-pgno), -indent N (-ind N), and -output_file (-of). The -page_numbers argument causes the fdoc command to end each page with two blank lines and a centered page number. The -indent control argument indents the entire document from the left for "N" spaces. (Remember that the indent control line within the fdocin file indents only from the left margin and does not affect the right margin.)

The -output_file control argument causes the formatted document to be placed in a separate file instead of being sent to the terminal. If you entered the control argument as, for example, "-of Exercise", the formatted file would be named "Exercise". If you were to command "fdoc Preamble -of" without giving a pathname, your formatted fdocin file would be placed in a separate file called "Preamble.fdocout". Either file can then be seen by using the read and print requests of your text editor and treated as any other permanent file.

In this example, the fdoc command will automatically number the pages of your document and indent (move) the entire document five spaces to the right. Give the fdoc command with the "-pgno" and "-ind 5" control arguments and then log out.

```
   r 08:43 0.072 4

  ! fdoc Preamble -pgno -ind 5




                              CONSTITUTION
                          OF THE UNITED STATES
                              OF AMERICA

              PREAMBLE

                   We  the  people of the United States, in
              Order to form a more perfect Union, establish
              Justice, insure domestic Tranquility, provide
              for the common defense, promote  the  general
              Welfare,  and secure the Blessings of Liberty


                                   1











              to ourselves and our Posterity, do ordain and
              establish this Constitution  for  the  United
              States of America.










                                   2




   r 08:43 0.316 12

  ! logout
```

Section 6

Using the WORDPRO Text Formatter

Section 5 gave an introduction to the format_document command and showed how this command will format a fdocin segment without your having inserted special control lines. Eight control lines were then described to show how you can change the format of a page of that output material. Those control lines were the indent, undent, fill on, fill off, align both margins, align the left margin only, page width, and page length.

This section describes the WORDPRO compose text formatter. The format_document command is a subset of the compose text formatter and serves ideally for relatively short, simple documents such as letters and memos that do not contain header lines, footer lines, footnotes, and sectional page numbers. The compose subsystem provides control lines to format for these specialties as well as for other more complex formatting situations.

As a first example of the compose command, invoke your text editor and read "Preamble.fdocin". Then write this segment as "Preamble. compin" where the pathname has now been changed and contains the suffix ".compin". (If you wish, you may now release your fdocin file; you will not use it again within this manual.) Quit the editor and give the compose command in the form of "compose Preamble.compin" and compare the results with your last fdoc command. Note that this compose command does not contain any control arguments ("–pgno" or "–ind 5") because control arguments for the compose command differ from those for the fdoc command. Nevertheless, you can see a great similarity between the two formatted documents.

```
  r 08:47 0.593 12                                 r 08:47 0.593 12

! qx                                             ! ted
! r Preamble.fdocin                              ! r Preamble.fdocin
! w Preamble.compin                              ! w Preamble.compin
! q                                              ! q
  r 08:49 0.090 4                                  r 08:49 0.090 4

! compose Preamble.compin




                      CONSTITUTION
                  OF THE UNITED STATES
                       OF AMERICA

        PREAMBLE

             We the  people of the  United States, in
        Order to form a more perfect Union, establish
        Justice, insure domestic Tranquility, provide
        for the  common defense, promote  the general
        Welfare, and secure  the Blessings of Liberty
        to ourselves and our Posterity, do ordain and







        establish this Constitution for  the United
        States of America.








  r 08:49 0.969 54
```

AUTOMATIC COMPOSE CONTROLS

The compose command you just gave applied to a compin segment that contained control lines. Without those control lines as you originally entered the file in Section 4, the compose command would have formatted the file identically as your first fdoc command did in Section 5. (You would, of course, have needed the ".compin" suffix instead of the ".fdocin" suffix in your pathname.)

By default, the compose command uses control lines similar to those of the fdoc command. These default control lines provide for a printout designed for 8½ by 11-inch paper with six blank lines for top and bottom margins, 65 characters to a line, and both left and right margins justfied.

ELEMENTARY COMPOSE CONTROLS

Control lines for the compose command are inserted as separate lines in your compin file through the text editor as for the fdoc command. These control lines usually consist of a period followed by three lower-case letters and often a number. The period must always by typed in the first column position of your input compin file.

Indent Left Control Line

Although the compose command, as you have just seen, recognized the indent control line, ".in N", inserted originally in your fdocin file, the actual compose command control line for indenting from the left is the **indent left** control line given as ".inl N". It is always best to type these three letters for the compose command, not only to form a habit but also because the compose command has a control line for indenting from the right, ".inr N". To avoid any question and possibly some frustration when a more complex document is formatted, always use ".inl N" for indenting from the left.

The indent left control line applies to all lines of your formatted output that follow it as does the indent control line used with the fdoc command. And as for the fdoc command, you can change the left indentation for any succeeding line or any number of succeeding lines. The number of indent left spaces can be either absolute, with respect to column position 0, or relative, with respect to the current left margin.

Undent Left Control Line

The **undent left** *control line applies only to the next single line of output following this control line.* The control line is of the form ".unl N" with a space before "N" and where "N"is the number of spaces undented from the value specified in the indent left control line. In other words, the undent left control line is used to change the indent value of the formatted output — but *only* for the first line of text output following this control line, as, for example, to indent five spaces to start a new paragraph.

Again, although the compose command recognized ".un N" of the fdoc command, it is best to type the three letters not only to form a habit of typing three letters for a compose command but also because the compose command has a control line for undenting from the right, ".unr N". As for the indent left control line and to avoid any question and possible frustration when a more complex document is formatted, always use ".unl N" for undent from the left.

Space Block Control Line

In your compin example, you used the carriage return (line feed) to provide blank lines between blocks of text (i.e., between the headings and the text). The preferred input for the compose command uses the **space block** control line which has the form ".spb N" (with a space before "N") where "N" is the number of lines to be left blank between blocks of text. A control line given as ".spb" is the same as ".spb 1" (one blank line).

The use of the space block control line is preferred over actual blank lines of input when editing compin segments. Because a compin segment may be edited for insertions and deletions, blank lines that are not indicated in your input file may be lost. Also, blank lines not stipulated by control lines may fall at a page fold of the formatted output and not be noticed until more input is appended at that point. You may, therefore, need to do extra editing.

Break Format Control Line

In your original fdocin segment input, you typed "CONSTITUTION OF THE UNITED STATES OF AMERICA" on three separate lines. The fdoc command, and the compose command, unless otherwise controlled, will attempt to fill each line to 65 characters. Thus, the fdoc command put all three heading lines of your fdocin file on one formatted output line the first time you used the fdoc command. The fdoc command considered those three lines of input as a single block of text.

To have the three lines formatted in the output as you typed them in your input required turning the fill control off and then back on, or alternatively, inserting control lines between the input lines. The compose command has control lines that will break this block of text before and after "OF THE UNITED STATES". The control lines are **break format** control lines that mean: "Interrupt the processing of this text block at this point and begin a new line." With the break format control lines, given as ".brf", there is no need to change the fill controls.

Fill and Align Control Lines

As you have seen, unless you insert control lines in the compin segment to indicate otherwise, the compose command fills in all lines of text with white space so that both left and right margins are justified. The compose command uses the same fill on (".fin") and fill off (".fif") control lines and align both (".alb") and align left (".all") control lines that the fdoc command uses. However, the compose command also has other align control lines.

The default align control line for the compose command, as for the fdoc command, is ".alb" meaning: "Align both left and right margins." to align the text at the left margin only (with the right margin remaining ragged), the control line is ".all"; to align the right margin only, the control line is ".alr". If you wish to center your material and leave both margins ragged, the control line is ".alc".

Page Width and Page Length Control Lines

The compose command, by default, automatically formats for 66 lines to a page. For the fdoc command, you used ".pdl N" as the control line to adjust the length of the page. Compose uses this same control line, called the **page definition length** control line, within your compin segment to accomplish the same purpose.

The compose command, by default, automatically formats 65 characters to a line. The number of characters per line, as for the fdoc command, can be adjusted by inserting a ".pdw N" control line, called a **page definition width** control line within your compin segment.

USING COMPOSE CONTROL LINES

The control lines given above for formatting a document are elementary to all compin segments. To show how they are used, you will invoke the text editor and read your compin file. You will replace each blank line with a space block control line and you will delete the ".fif" and ".fin" control lines and use the break format control lines to give a heading of three lines in the formatted output. You will change the fdoc command control lines for indent and undent to the indent left and undent left control lines used by the compose command. You will also indent the heading 30 spaces and the text body 15 spaces as you first did with fdoc command. The first line of the Preamble will be indented five spaces as a paragraph.

Log in, invoke the text editor, and read your compin file. Request a print of the first few lines to refresh your memory as to its contents.

```
       r 08:49 0.969 54

     ! qx
     ! r Preamble.compin
     ! 1,/We/p
       .pdl 25
       .pdw 50
       .fif
       .in 20

       .un -4
       CONSTITUTION
       OF THE UNITED STATES
       .un -5
       OF AMERICA
       .fin
       .in 5

       PREAMBLE

       .un -5
       We the people of the United
```

```
       r 09:49 0.969 54

     ! ted
     ! r Preamble.compin
     ! 1,/We/p
       .pdl 25
       .pdw 50
       .fif
       .in 20

       .un -4
       CONSTITUTION
       OF THE UNITED STATES
       .un -5
       OF AMERICA
       .fin
       .in 5

       PREAMBLE

       .un -5
       We the people of the United
```

The first step to altering your compin file is to delete those control lines you will not be using. Thus, delete ".fif", ".fin", ".un –4" and ".un –5" (in the heading), ".pdl 25", and ".pdw 50"

```
   We the people of the United
 ! /.fif/ d
 ! /.fin/ d
 ! /.un -4/ d
 ! /.un -5/ d
 ! /.pdl 25/ d
 ! /.pdw 50/ d
```

Because the heading is to be indented 30 spaces, the next editing request is to change the indent control line ".in 20" to the indent left control line ".inl 30". Because the remainder of the file is to be indented 15 spaces, change the indent control line ".in 5" to the compose indent left control line ".inl 15". Also change the undent control line ".un –5" for the paragraph indentation to the compose undent left control line ".unl –5".

```
  /.pdw 50/ d
! /.in 20/ c
! .inl 30
! \f
! /.in 5/ c
! .inl 15
! /f
! /.un -5/ c
! .unl -5
! \f
```

You can substitute the space block control line for each blank line throughout the entire file by typing "1,$s/^$/.spb/". (Remember that "/^$/" matches a blank line.) Type this request.

```
  \f
! 1,$s/^$/.spb/
```

To have the heading formatted as three lines in the output, insert a ".brf" control line both before and after "OF THE UNITED STATES". That is, use the locate request to find this line and insert ".brf" before it and then append ".brf" after it.

```
  1,$s/^$/.spb/
! /OF THE/ i
! .brf
! \f
! /OF THE/ a
! .brf
! \f
```

The examples show these editing changes. There are, of course, several other ways of editing this file to insert the required compose control lines. Request a print of your buffer contents to confirm all the editing changes. Then overwrite those changes in your permanent compin file by using the write request to your text editor.

```
      \f
    ! 1,/We/p
      .inl 30
      .spb
      CONSTITUTION
      .brf
      OF THE UNITED STATES
      .brf
      OF AMERICA
      .inl 15
      .spb
      PREAMBLE
      .spb
      .unl -5
      We the people of the United
    ! w
```

Most Multics commands have short names. The compose command also has a short name: **comp**. Furthermore, the compose command will not recognize any input file not having the "compin" suffix with the period as part of the file name. Consequently, when you invoke the compose command, you do not need to type the suffix ".compin". For your present file, the system will give a formatted output when the line is only "comp Preamble".

If you make errors in typing the control lines, the system will provide an error list. This list appears just before the ready message at the end of your formatted output. For example, suppose you insert ".brk" in line 4 instead of ".brf". Not only will the compose command disregard this control line (you will see the effect in the formatted output), but it will also give a message similar to the following: compose error list: (Vers. XXXX) File Preamble; Line 4; Unknown control request. .brk

Quit the text editor and issue the compose command to get a formatted output of your compin file. Type "comp Preamble" and note that the formatted output is identical to the output you received the second time you gave the fdoc command in Section 5.

```
  w
! q
  r 13:44 1.250 116

! comp Preamble




                              CONSTITUTION
                              OF THE UNITED STATES
                              OF AMERICA

                    PREAMBLE

                         We the people of  the United States, in Order
                    to form  a more perfect  Union, establish Justice,
                    insure   domestic   Tranquility,   provide   for   the
                    common defense,  promote the general  Welfare, and
                    secure the  Blessings of Liberty  to ourselves and
                    our  Posterity,  do   ordain  and  establish  this
                    Constitution for the United States of America.










  r 13:44 9.908 114
```

To see how the align control lines are used by the compose command, invoke your text editor and read your compin segment. You are now going to center your headings and justify only the left margin of the formatted output. (The fill on control will be left on by default.)

First, substitute ".inl 15" for ".inl 30" so that your headings can be centered with respect to the body of your text, which is indented 15 spaces. Then, append the align center control line ".alc" to center these headings. Next, insert the align control line ".all" before the text body (your subheading "PREAMBLE" will also be centered). Do not forget the write request to overwrite these changes.

```
  r 13:44 9.908 114                         r 13:4 9.908 114

! qx                                      ! ted
! r Preamble.compin                       ! r Preamble.compin
! /.inl 30/ s//.inl 15/                   ! /.inl 30/ s//.inl 15/
! a                                       ! a
! .alc                                    ! .alc
! \f                                      ! \f
! /.unl -5/i                              ! /.unl -5/i
! .all                                    ! .all
! /f                                      ! \f
! w                                       ! w
```

Quit the text editor and use the compose command to format your compin file. As you can see, the headings of your formatted output are centered over the body of the text. And, the left margin only is justified with the right margin ragged. The fill on control automatically has moved the text words from line to line in such a way that the last word does not extend past the right margin (past column position 65).

```
  w
! q
  r 13:47 0.484 48
```

```
! comp Preamble




                        CONSTITUTION
                    OF THE UNITED STATES
                         OF AMERICA

                          PREAMBLE

                 We the people of the United States, in Order
             to form a more perfect Union, establish Justice,
             insure domestic Tranquility, provide for the
             common defense, promote the general Welfare, and
             secure the Blessings of Liberty to ourselves and
             our Posterity, do ordain and establish this
             Constitution for the United States of America.




 r 13:47 2.780 73
```

As a final exercise in this section, replace ".all" with ".alb" in order to justify both margins again. (Note that if you delete ".all", the control line ".alc" above will be in effect for the entire file.)

```
      r 13.47 2.780 73                                    r 13:47 2.780 73

    ! qx                                                ! ted
    ! r Preamble.compin                                 ! r Preamble.compin
    ! /.all/ s//.alb/                                   ! /.all/ s//.alb/
    ! w                                                 ! w
```

Have the compose command format this compin file. However, this time type the compose command as "comp Preamble -hyph". The system will now use the Multics dictionary in order to hyphenate those words that can be hyphenated at the ends of lines in order to reduce the amount of white space within each line (between words) for fill in. This control argument for the compose command is only one of several. Control arguments for the compose command are always separated from each other and from the pathname by a space and they always begin with a hyphen.

```
                                      w
                                    ! q
                                      r 13:48 0.428 30
```

```
! comp Preamble -hyph



                          CONSTITUTION
                     OF THE UNITED STATES
                          OF AMERICA

                            PREAMBLE

             We the people of  the United States, in Order
        to form  a more perfect  Union, establish Justice,
        insure  domestic  Tranquility,  provide  for  the
        common defense,  promote the general  Welfare, and
        secure the  Blessings of Liberty  to ourselves and
        our Posterity,  do ordain and  establish this Con-
        stitution for the United States of America.



  r 13:48 3.292 68

 ! logout
```

Section 7
Formatting Headers and Footers

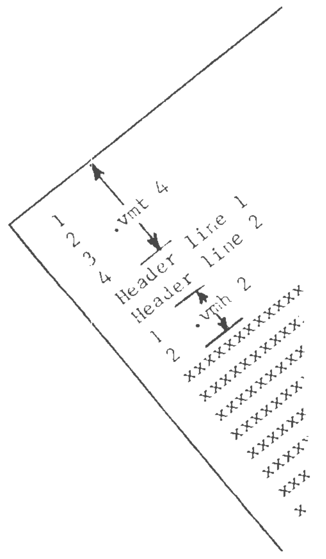
Section 6 gave an introduction to the compose command and showed how this command will format a compin segment without your having inserted special control lines. Elementary control lines were then described to show how you can change the format of a page of that output material. Those lines were the indent left, undent left, space block, and break format control lines. You were also introduced to the fill and align control lines as well as the page definition length and page definition width control lines.

This section describes various other control lines used to alter the format of your output file. Among these control lines are those for formatting header lines, footer lines, titles, and page numbers.

HEADER LINES

Unless control lines to the contrary are inserted within a compin file, the compose command will automatically leave six blank lines from the top of a page to the first line of text. These six lines are divided into two parts: a **vertical margin top** of four lines and a **vertical margin header** of two lines.

The control line used to change the vertical margin top is ".vmt N", where "N" is the number of blank lines separating the top of the page from the first header line. The control line for the vertical margin header is ".vmh N", where "N" is the number of blank lines separating the last header line from the body of the text.



In your "Preamble.compin" file of Section 6, you did not have a header line, nor did you insert any vertical margin control lines. The compose command, by default, used ".vmt 4" and ".vmh 2", which gave the six blank lines before the start of the text input.

A header line may be placed at the upper left, center, or right on a page. It may be placed on all pages, on only even-numbered pages, or on only odd-numbered pages. The control line for a header line takes the form of ".hla N M ||||" where ".hla" stands for "header line, all pages". The vertical bars are delimiters that separate the left, center, and right positions of your header line. All four vertical bars must be included in the control line and the position of the header in the formatted output depends upon where that header line is typed between the bars. If the header line is to be on only even-numbered pages, ".hla" is replaced by ".hle". If the header line is to be on only odd-numbered pages, ".hla is replaced by ".hlo". The letter "N" is the number of the header line (1 for the first, 2 for the second, etc.). The letter "M" denotes the position of the left portion of the header line with respect to the current left indent or to the left margin.

As an example, suppose you wanted the header line "Chapter XX" to appear at the upper left on all pages of your formatted output and aligned with the current left margin. Your control line for this header line would be ".hla 1 0 |Chapter XX|||". If you wanted this header line to appear in the center, your control line would be typed as ".hla 1 0 ||Chapter XX||". If you wanted the line on the right, the control line would be ".hla 1 0 |||Chapter XX|".

The number "1" in these control lines is the identification number for the header. If you wanted a two-line header, you would have the second line placed in your compin segment as ".hla 2 0 |Part YY|||". The digit "0" refers to the header position with respect to the current indent left value or the left margin. If the current indent left value is 0, thereby providing for 65 characters to a line, and "M" is 5, the compose command will have 60 spaces in which to format the header, and a header on the left would start at column position 6.

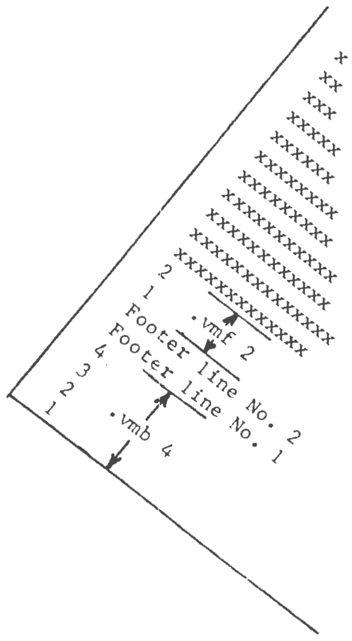
If you want the header line on the first page, you must place the control line before any titles or text in your compin segment. By definition, header lines precede text on a page; if you give text before specifying your header lines, the compose command will format the text and hold the header lines for the next page. If the header line is to appear after the first page or if a different header line is to be used after the first page (and on succeeding pages), the header control line would be inserted in the compin segment after the first line of the text.

FOOTER LINES

The compose command automatically provides six blank lines between the last line of text and the bottom of the page. These lines are numbered upward from the bottom and, as for headers, can be changed by control lines in your compin segment. These six lines are divided into two parts: a **vertical margin bottom** of four lines and a **vertical margin footer** of two lines.

The control line used to change the vertical margin bottom is ".vmb N", where "N" is the number of blank lines separating the bottom of the page from the first footer line. The control line for the vertical margin footer is ".vmf N", where "N"is the number of blank lines separating the last footer line from the last line of the text. By default, these two control lines are ".vmb 4" and ".vmf 2".

The control line for a footer line has the same form as the control line for a header line except for the control line letters. If a footer line is to be centered on all pages, this control line would be typed as ".fla N M ||Footer line||", where ".fla" stands for "footer line all pages". For only even-numbered pages, ".fle" would be used, and for only odd-numbered pages, ".flo" would be used. "N" and "M" have the same meaning as for a header line except that "N" is counted up from the bottom instead of down from the top.



NOTE

"N" and "M" have default values. If they are omitted from the control line such as ".fla ||||", the compose command will format as if you had included the values in the control line. The default values are 1 for "N" and 0 for "M".

PAGE NUMBERING

The compose command, when given a special expression within a header control line or footer control line, will automatically cause each page of the formatted output to be numbered. For example, to have each page of your formatted output numbered automatically and consecutively from "1", you would insert your header or footer control line as "%PageNo%" using percent symbols, no spaces, and with vertical bars used as delimiters. Note the use of uppercase letters. If you wanted a section or chapter number to appear with the page number as, for example, 3-1, 3-2, etc., your control line would contain "3-%PageNo%". Thus, to have sectional numbering appear centered at the bottom of the formatted output pages, your actual control line would be ".fla 1 0 ||3-%PageNo%||".

There are times when you will not want your first page to begin as page 1. Front matter, for example, may not contain page numbers until the table of contents, which often begins on page iii. To force the numbering to begin with a specific page, a **break page** control line is used. This control line is ".brp N" where "N" is the number of the new page.

Lowercase Roman numerals are usually used instead of Arabic numerals for page numbering in the front matter of a document. The compose command automatically gives Arabic numerals unless the compin segment has a control line for Roman numerals. The control line uses the break page control line given as ".brp N rl" where "rl" means "Roman lowercase". "N" may be either Roman or Arabic. With the table of contents as an example, the control line to be placed within the compin segment could be typed as ".brp 3 rl". The compose command will automatically translate "3" into "iii" if this control line precedes the first use of "%PageNo%".

The normal use of the break page control line, however, is to force a new page. And, in the absence of a number for "N", give the new page the next sequential page number.

DATE

Some documents require that the current date appear on each page. The compose command automatically provides that date on each formatted page if, within a header or footer control line, a special expression is used. This expression, within the delimiters, is "%Date%". Each page will then be dated in the form of month/day/year as in 10/03/99 for October 3, 1999.

TITLE LINES

In Section 6, you used the align center control line in order to center your title lines. You used the break format control line to keep the title lines as three lines rather than as one continuous line. You further used the space block control line to separate the title lines from the text. You could have used one control line instead for each line of your title.

The control line for titles has the same format as the control lines for headers and footers: ".tlh N M ||title||". In this case, ".tlh" stands for "title line header" and "M" is the reference of the title line with respect to the current indent line value for the left margin. The difference between this control line and that for headers and footers is the value for "N". The letter "N" is the number of blank lines separating the title line from the next formatted line (which could be the second line of the title).

In the example of Section 6, the three lines of your title could have been given as follows:

```
.tlh 0 0 ||CONSTITUTION||
.tlh 0 0 ||OF THE UNITED STATES||
.tlh 1 0 ||OF AMERICA||
```

where "1" is the number of blank lines separating "OF AMERICA" from the subtitle "PREAMBLE". This centered subtitle, likewise, could have been input in your compin segment as:

```
.tlh 1 0 ||PREAMBLE||
```

NOTE

"N" and "M" have default values. If they are omitted from the control line such as ".tlh ||||", the compose command will format as if you had included the values in the control line. The default values for "N" and "M" are "0" for the title line control line.

TITLE BLOCKS

The compose control lines for the headers, footers, and title lines cause the compose command to place those headers, footers, and title lines at specific places in the formatted output. Equations, notes, etc. often have the same form but are usually set off within a block of text. In those cases, a **block begin title** control line ".bbt" and a **block end title** control line ".bet" are needed. The equation is then placed within delimiters on a separate compin line between these block control lines as "||equation||". In other words, for a one-line note to be formatted within the text body, three control lines could be required. If you wish to have the note separated from the text by blank lines, you must remember to also insert the ".spb" control lines.

If you know the number of lines required for the title block, the block begin title control line can be given as ".bbt N", where "N" is the required number of lines. In this case, the block end title control line is not needed. Thus, if you type ".bbt" without a number because you do not want to count lines or you know you have several lines, you must also use ".bet"; otherwise ".bet" is not needed.

WIDOW LINES

A widow line is a single line of text standing alone and either begins a paragraph (text block) at the bottom or a page or ends a paragraph at the top of a page. The compose command does not allow widow lines and automatically formats the output pages so that no less than two lines of text are kept together at the bottom or top of a page.

A control line to change the number of widow lines in the formatted output is the **widow text** control line, given as ".wit N", where "N" is the number of lines allowed at the top of a page or at the bottom of a page. By default, "N" is "2" or ".wit 2", which means that the text block (paragraph) must be at least four lines long before it will be split between pages.

EXAMPLE USING HEADERS AND FOOTERS

You will now use header and footer control lines as well as other control lines for the compose command to format Abraham Lincoln's Gettyburg Address. For this example, some control lines will be omitted from the compin file and the compose command will use the default values instead. Those control lines that you will not alter from their default values are as follows:

.vmh (vertical margin header, 2 lines)
.vmf (vertical margin footer, 2 lines)
.wit 2 (widow text, 2 lines)
.inl 0 (indent left, 0 spaces)

In this example, you are to format for a length of 32 lines, ".pdl 32", and for a width of 35 characters, ".pdw 35".

The vertical margin top will be three lines, ".vmt 3", as will be the vertical margin bottom, ".vmb 3".

All pages will be numbered in the center at the bottom of the page as Page 1, Page 2, etc. by a footer control line: ".fla 1 0 ||Page %PageNo%||".

The first page will have a header line "Speech" on the right while all succeeding pages will have a header line "Speech Continued" on the left. The front page header control line will thus be typed: ".hla 1 0 |||Speech|". The heading on the succeeding pages will be input as ".hla 1 0 |Speech Continued|||". Note that the second header control line, in most cases, can be inserted in your compin segment immediately after the first line of regular text. In some rare cases, it must be inserted after the end of your first formatted text block.

The title will be given as two lines and separated from the text body by three blank lines. The control lines for the title thus become:

```
.tlh 0 0 ||GETTYSBURG||
.tlh 3 0 ||ADDRESS||
```

Each paragraph of the text body will be separated from the next by one blank line. Also, the first line of each paragraph will be indented five spaces.

A title block will be used at the end of the compin segment for the signature. Three control lines will be required:

```
.bbt
|||Abraham Lincoln|
.bet
```

(Alternatively, only two lines could be used: ".bbt 1" and "|||Abraham Lincoln|".) This signature will be separated from the last line of the text by 2 blank lines, ".spb 2".

Log in, and after the ready message, invoke the text editor and create your compin segment. Your work should be similar to the following:

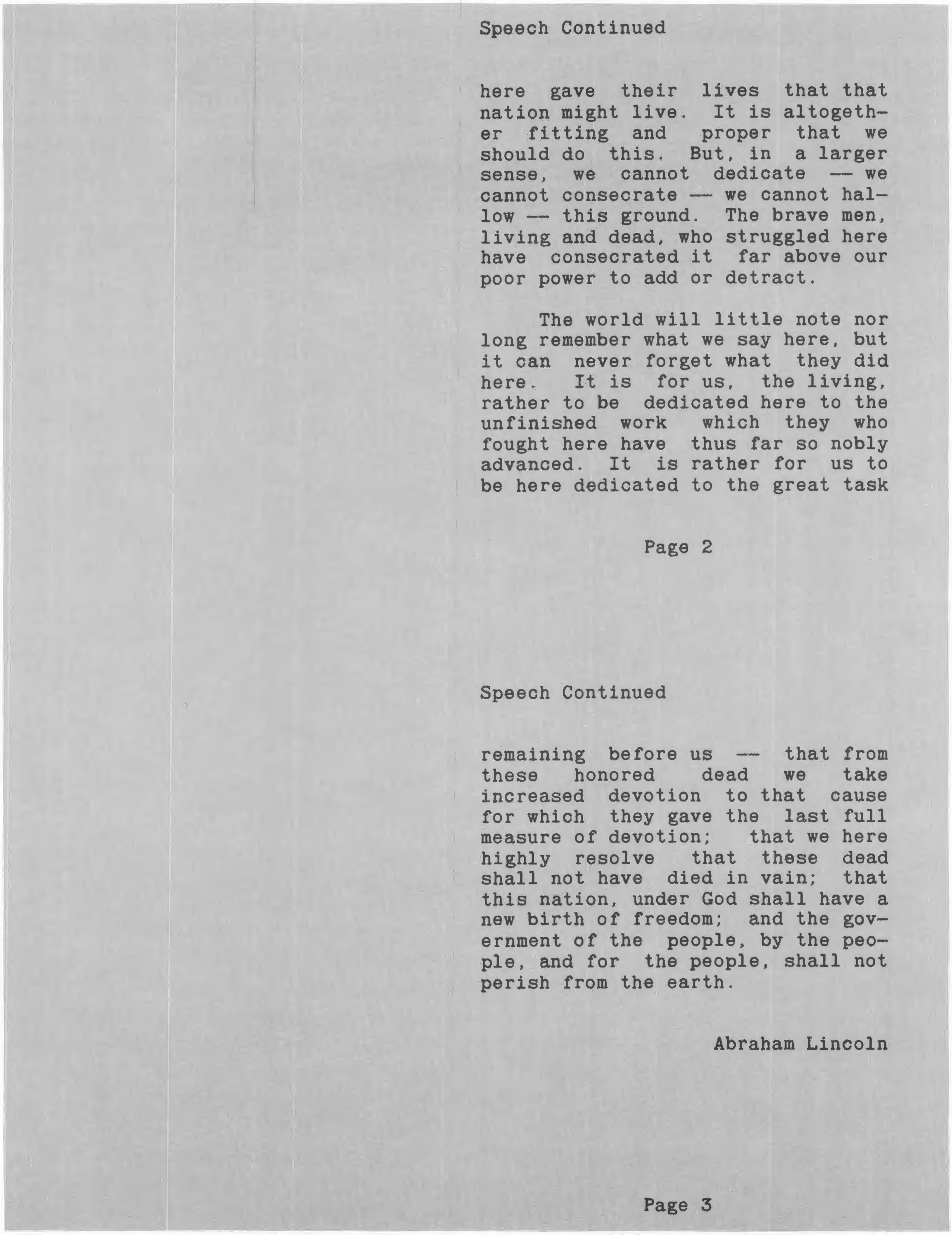
```
! qx                                                    ! ted
! a
! .pdl 32
! .pdw 35
! .vmt 3
! .vmb 3
! .fla 1 0  ||Page %PageNo%||
! .hla 1 0  |||Speech|
! .tlh 0 0  ||GETTYSBURG||
! .tlh 3 0  ||ADDRESS||
! .unl -5
! Fourscore and seven years ago
! our fathers brought forth on this continent
! a new nation, conceived in liberty and dedicated to
! the proposition that all men are created equal.  Now
! we are engaged in a great civil war,
! testing whether that nation, or any nation
! so conceived and so dedicated, can long endure.
! .hla 1 0 |Speech Continued|||
! .spb
! .unl -5
! We are met on a great battlefield of that war.
! We have come to
! dedicate a portion of that field as a final resting place
! for those who here gave their lives
! that that nation might live.   It
! is altogether fitting and proper that we should do this.
! But, in a larger sense, we cannot dedicate -- we
! cannot consecrate -- we cannot hallow -- this ground.
! The brave men, living and dead, who struggled here
! have consecrated it
! far above our poor power to add or detract.
! .spb
! .unl -5
! The world will little note nor long remember
! what we say here, but it can never forget
! what they did here.
! It is for us,
! the living, rather to be dedicated here
! to the unfinished work which they who fought here
! have thus far so nobly advanced.
! It is rather for us to be here
! dedicated to the great task remaining
! before us -- that from these honored dead
! we take increased devotion to that cause for
! which they gave the last full measure of devotion;
! that we here highly resolve
! that these dead shall not have died
! in vain;  that this nation, under
! God shall have a new birth of freedom;
! and the government of the people,
! by the people, and for the people,
! shall not perish from the earth.
!.spb 2
!.bbt
!|||Abraham Lincoln|
!.bet
! \f
! w Lincoln compin
! q
```

For any editing changes, read your compin file back into your buffer (if you have quit your editor) and use the text editor for changes, additions, substitutions, inserts, and deletions. Observe that those control lines such as page definition length and page definition width control lines that affect the entire document are placed as housekeeping control lines prior to any text, title, footer, or header control lines. If an indent left control line were input as part of this file, it would likewise have to be placed at the beginning of this housekeeping section or the position of the headers, footers, and titles would be displaced from the rest of the document in the formatted output.

In this example, three lines are used for the signature block.Instead of the title block, the signature could have been preceded by an align right control line, ".alr", in which case, the same formatted result would occur.

After you have ascertained that your compin file is accurate, give the compose command. Use the hyphenation control agrument as you did for your "Preamble" file. Also use an additional control argument. This additional control argument is the "–indent N" control argument. This control argument will cause the entire document to be positioned to the right by "N" number of spaces when it is given in the compose command line. In this case, use the control argument "–indent 20" which can be given in its short form as "–ind 20". This control agrument is placed on the same line as the compose command and the hyphenation control argument. Do not forget to space between the control arguments and do not forget the hyphen preceding each control argument. Your formatted output will be similar to the following:

```
! comp Lincoln -hyph -ind 20

                                                 Speech


                                       GETTYSBURG
                                        ADDRESS


                                   Fourscore and  seven years ago
                              our  fathers brought forth  on this
                              continent  a new  nation, conceived
                              in  liberty  and dedicated  to  the
                              proposition   that   all   men   are
                              created equal.  Now  we are engaged
                              in  a  great   civil  war,  testing
                              whether that nation,  or any nation
                              so conceived and  so dedicated, can
                              long endure.

                                   We are met  on a great battle-
                              field of that war.  We have come to
                              dedicate a portion of that field as
                              a final resting place for those who


                                         Page 1
```



Speech Continued

here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate — we cannot consecrate — we cannot hallow — this ground. The brave men, living and dead, who struggled here have consecrated it far above our poor power to add or detract.

The world will little note nor long remember what we say here, but it can never forget what they did here. It is for us, the living, rather to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task

Page 2

Speech Continued

remaining before us — that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion; that we here highly resolve that these dead shall not have died in vain; that this nation, under God shall have a new birth of freedom; and the government of the people, by the people, and for the people, shall not perish from the earth.

Abraham Lincoln

Page 3

Section 8

Formatting Footnotes and Tables

UNREFERENCED FOOTNOTES

Unreferenced footnotes generally appear at the bottom of the first page of a document and usually apply to the entire document. These footnotes most often are not numbered whereas referenced footnotes are numbered.

The compose command will format an unreferenced footnote when you input a **footnote unreferenced** control line within the compin file. This control line must be placed before the actual footnote and is usually placed near the top of the compin segment (within the housekeeping section). This control line is input as ".ftu".

The footnote itself is input in the compin file as a block similar to the title block described in Section 7. That is, there must be block begin and block end control lines. The **block begin footnote** control line is ".bbf" and the **block end footnote** control line is ".bef". If the block end footnote control line is omitted, all text following the block begin footnote control line will be treated as a single footnote.

Only one stipulation applies to these footnote blocks although you may, of course, have more than one unreferenced footnote. These footnote blocks cannot be inserted in your compin file prior to a text block such as a paragraph. You must input at least one word of a text block before you can input your footnote.

As an example of formatting an unreferenced footnote, invoke the text editor and read "Lincoln.compin". Request a print of the lines 1 through the first line of the first text block ("Fourscore..."). Somewhere near the top of the file, such as after the control line ".vmb 3", append the control line ".ftu". Then, after the first line of the text block, append the footnote block.

Append the footnote block by first typing the block begin footnote control line ".bbf" on a separate line. Next, type the footnote, which in this case, will be "This speech was given in 1863." Finally, end the footnote block with the block end footnote control line ".bef".

```
  r 14:41 0.462 21

! qx
! r Lincoln.compin
! 1,/Fourscore/ p
  .pdl 32
  .pdw 35
  .vmt 3
  .vmb 3
  .fla 1 0   ||Page %PageNo%||
  .hla 1 0   |||Speech||
  .tlh 0 0   ||GETTYSBURG||
  .tlh 3 0   ||ADDRESS||
  .unl -5
  Fourscore and seven years ago
! /.vmb 3/ a
! .ftu
! \f
! /Fourscore/ a
! .bbf
! This speech was given in 1863.
! .bef
! \f
! w
! q
  r 14:43 1.430 82
```

```
  r 14:41 0.462 21

! ted
! r Lincoln.compin
! 1,/Fourscore/ p
  .pdl 32
  .pdw 35
  .vmt 3
  .vmb 3
  .fla 1 0   ||Page %PageNo%||
  .hla 1 0   |||Speech|
  .tlh 0 0   ||GETTYSBURG||
  .tlh 3 0   ||ADDRESS||
  .unl -5
  Fourscore and seven years ago
! /.vmb 3/ a
! .ftu
! \f
! /Fourscore/ a
! .bbf
! This speech was given in 1863.
! .bef
! \f
! w
! q
  r 14:43 1.430 82
```

The compose command will place this unreferenced footnote at the bottom of the first page of your formatted output. The footnote will be separated from the last line of the text on the page by a horizontal line (or a line of symbols depending upon the terminal used) extending from the left margin to the right margin.

After you have quit your editor (do not forget to save your file), you are ready to invoke the compose command again. If you issue the compose command as you did in Sections 6 and 7, the compose command will format and output the entire file. If, as in this case, you might wish to see page 1 only, you could wait for compose to output that page and then press your interrupt key (**BRK, INTERRUPT,** etc.). The Multics system would then give the QUIT signal and the ready message with the level information. The release –all command line would then need to be used to remove the level information.

NOTE

> The program interrupt command "pi" does not apply to the compose command. As used in this document, it applies only when you are in the text editor and wish to continue using the text editor.

A better way to see only a page or several pages of the formatted output from the compose command is to add another control argument to the command line. In this case, that control argument is simply "–pages X,Y". For example, to see pages 2 and 3 only, the control argument would be given as "–pages 2,3". To see page 1 only, the control argument would be given as "–pages 1". The short name "–pgs X,Y" can be used instead of "–pages X,Y".

Consequently, to see the formatted output of page 1, with the unreferenced footnote, give the compose command to format page 1, hyphenate, and indent 20 spaces to center your output on the printed page. Type "comp Lincoln –hyph –ind 20 –pgs 1".

NOTE

> If you wish to see the remaining pages of your formatted output, give the compose command again, replacing the command "–pages 1" with "–from 2".

```
   r 14:43 1.420 82

 ! comp Lincoln -hyph -ind 20 -pgs 1


                                                 Speech


                                 GETTYSBURG
                                   ADDRESS


                              Fourscore and  seven years ago
                         our  fathers brought forth  on this
                         continent  a new  nation, conceived
                         in  liberty  and dedicated  to  the
                         proposition   that   all   men   are
                         created equal.  Now  we are engaged
                         in  a  great   civil  war,  testing
                         whether that nation,  or any nation
                         so conceived and  so dedicated, can
                         long endure


                         __________________________________

                         This speech was given in 1863.


                                   Page 1



   r 14:43 21.270 219
```

Note that the compose command has indeed formatted the unreferenced footnote. Also note the amount of space separating the last line of the first text block from the horizontal line. By default, the compose command has a widow text control line of ".wit 2". Consequently, the compose command did not begin the next paragraph on this page because there was room for only the first line of the next paragraph. (You could, of course, force the compose command to allow widow lines by entering in your compin segment ".wit 1" or ".wit 0", but it is not good formatting practice.)

REFERENCED FOOTNOTES

Referenced footnotes provide information allied to a subject within a portion of a document by way of giving a reference, comment, or explanation. The compose command with proper control lines in the compin segment, will format these footnotes, consecutively numbering them as they occur on a page. That is, if two footnotes appear on a page, the compose command will automatically number them (1) and (2). If a third footnote exists on a different page, the compose command will number it as (1) on that page.

For footnotes referenced to a page, the compin **footnote paged** control line is ".ftp". As for unreferenced footnotes, this control line must be placed before the first footnote. The footnotes themselves have the same compin form as unreferenced footnotes: they must be preceded by the block begin footnote control line ".bbf" and end with the block end footnote control line ".bef". In addition, the point at which the compin segment is broken to insert the footnote is the point compose inserts the footnote number in the formatted output.

On occasion you may want some footnotes numbered while others are to be unreferenced. For those cases, the footnote page control line ".ftp" is still used but the footnote blocks that are to be unreferenced must be given a different block begin footnote control line. For those unreferenced footnotes, the **block begin footnote unreferenced** control line ".bbf u" must replace the ".bbf" control line.

To see how to insert referenced footnote blocks and how the compose command then formats the compin segment, again invoke the text editor and read the "Lincoln.compin" file. Then substitute ".ftp" for ".ftu".

Change the unreferenced footnote line "This speech was given in 1863." to "87 years after 1776." Also within this text block, after the expression "civil war,", insert "1861 through 1865." as a footnote. In the second text block, after the expression "resting place", insert "Gettysburg, Pennsylvania." as a footnote. Also insert the footnote "November, 1863." after the expression "who struggled here". Finally, add an unreferenced footnote "Lincoln died in April, 1865." to the last line of the last text block. Then, request the editor to overwrite these changes and insertions.

```
   r  15:06 1.030 60                          r 15:06 1.030 60

 ! qx                                       ! ted
 ! r Lincoln.compin                         ! r Lincoln.compin
 ! /.ftu/ s//.ftp/                          ! /.ftu/ s//.ftp/
 ! /This speech/ c                          ! /This speech/ c
 ! 87 years after 1776.                     ! 87 years after 1776.
 ! \f                                       ! \f
 ! /civil war,/ a                           ! /civil war,/ a
 ! .bbf                                     ! .bbf
 ! 1861 through 1865.                       ! 1861 through 1865.
 ! .bef                                     ! .bef
 ! \f                                       ! \f
 ! /resting place/ a                        ! /resting place/ a
 ! .bbf                                     ! .bbf
 ! Gettysburg, Pennsylvania.                ! Gettysburg, Pennsylvania.
 ! .bef                                     ! .bef
 ! \f                                       ! \f
 ! /who struggled here/ a                   ! /who struggled here/ a
 ! .bbf                                     ! .bbf
 ! November, 1863.                          ! November, 1863.
 ! .bef                                     ! .bef
 ! \f                                       ! \f
 ! /from the earth./ a                      ! /from the earth./ a
 ! .bbf u                                   ! .bbf u
 ! Lincoln died in April, 1865.             ! Lincoln died in April, 1865.
 ! .bef                                     ! .bef
 ! \f                                       ! \f
 ! w                                        ! w
```

Request a print of this compin file in order to check that these changes and insertions have been included at the suggested points.

```
! 1,$p
  .pdl 32
  .pdw 35
  .vmt 3
  .vmb 3
  .ftp
  .fla 1 0 ||Page %PageNo%||
  .hla 1 0 |||Speech|
  .tlh 0 0 ||GETTYSBURG||
  .tlh 3 0 ||ADDRESS||
  .unl -5
  Fourscore and seven years ago
  .bbf
  87 years after 1776.
  .bef
  our fathers brought forth on this continent
  a new nation, conceived in liberty and dedicated to
  the proposition that all men are created equal.  Now
  we are engaged in a great civil war,
  .bbf
  1861 through 1865.
  .bef
  testing whether that nation, or any nation
  so conceived and so dedicated, can long endure.
  .hla 1 0  |Speech Continued|||
  .spb
  .unl -5
  We are met on a great battlefield of that war.
  We have come to
  dedicate a portion of that field as a final resting place
  .bbf
  Gettysburg, Pennsylvania.
  .bef
  for those who here gave their lives
  that that nation might live.  It
  is altogether fitting and proper that we should do this.
  But, in a larger sense, we cannot dedicate -- we
  cannot consecrate -- we cannot hallow -- this ground.
  The brave men, living and dead, who struggled here
  .bbf
  November, 1863.
  .bef
  have consecrated it
  far above our poor power to add or detract.
  .spb
  .unl -5
  The world will little note nor long remember
  what we say here, but it can never forget
  what they did here.
```

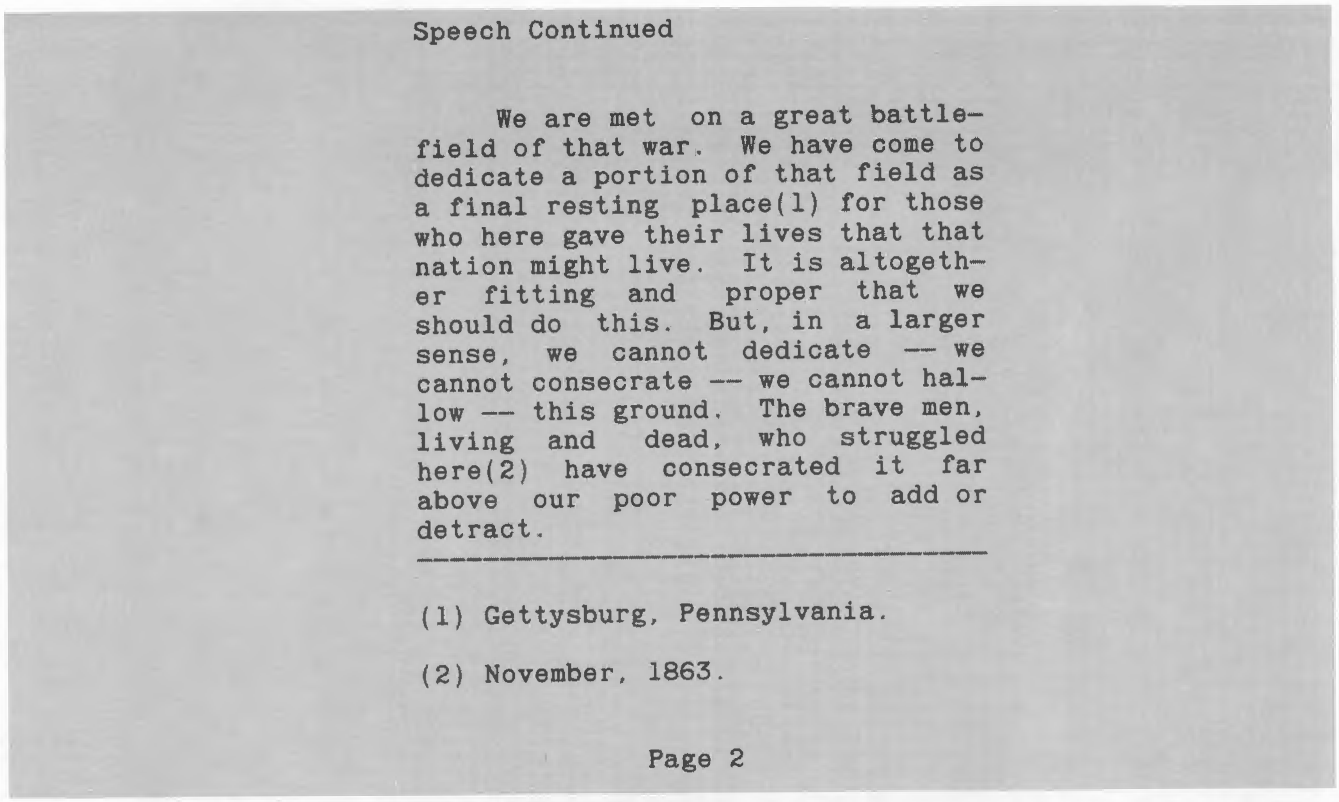
```
    It is for us,
    the living, rather to be dedicated here
    to the unfinished work which they who fought here
    have thus far so nobly advanced.
    It is rather for us to be here
    dedicated to the great task remaining
    before us -- that from these honored dead
    we take increased devotion to that cause for
    which they gave the last full measure of devotion;
    that we here highly resolve
    that these dead shall not have died
    in vain;  that this nation, under
    God shall have a new birth of freedom;
    and the government of the people,
    by the people, and for the people,
    shall not perish from the earth.
    .bbf u
    Lincoln died in April, 1865.
    .bef
    .spb 2
    .bbt
    |||Abraham Lincoln|
    .bet
  ! q
    r 15:13 3.196 228
```

You are now ready to quit your editor and give the compose command again. And again, give the command to format your "Lincoln.compin" file using hyphens and with the output indented 20 character spaces on the page.

NOTE

A compose command error list at the end of the formatted output may state that the footnote extends the margin on page 2. The compose command may change either the vertical margin footnote or the vertical margin header because of the default widow text control line ".wit 2".

```
  r 15:13 3.196 228

 ! comp Lincoln -hyph -ind 20



                                              Speech


                                   GETTYSBURG
                                    ADDRESS



                            Fourscore   and   seven   years
                        ago(1) our fathers brought forth on
                        this   continent   a   new   nation,
                        conceived in  liberty and dedicated
                        to the proposition that all men are
                        created equal.  Now  we are engaged
                        in  a great  civil  war,(2) testing
                        whether that nation,  or any nation
                        so conceived and  so dedicated, can
                        long endure.
                        ___________________________________

                        (1) 87 years after 1776.

                        (2) 1861 through 1865.

                                   Page 1
```



Speech Continued

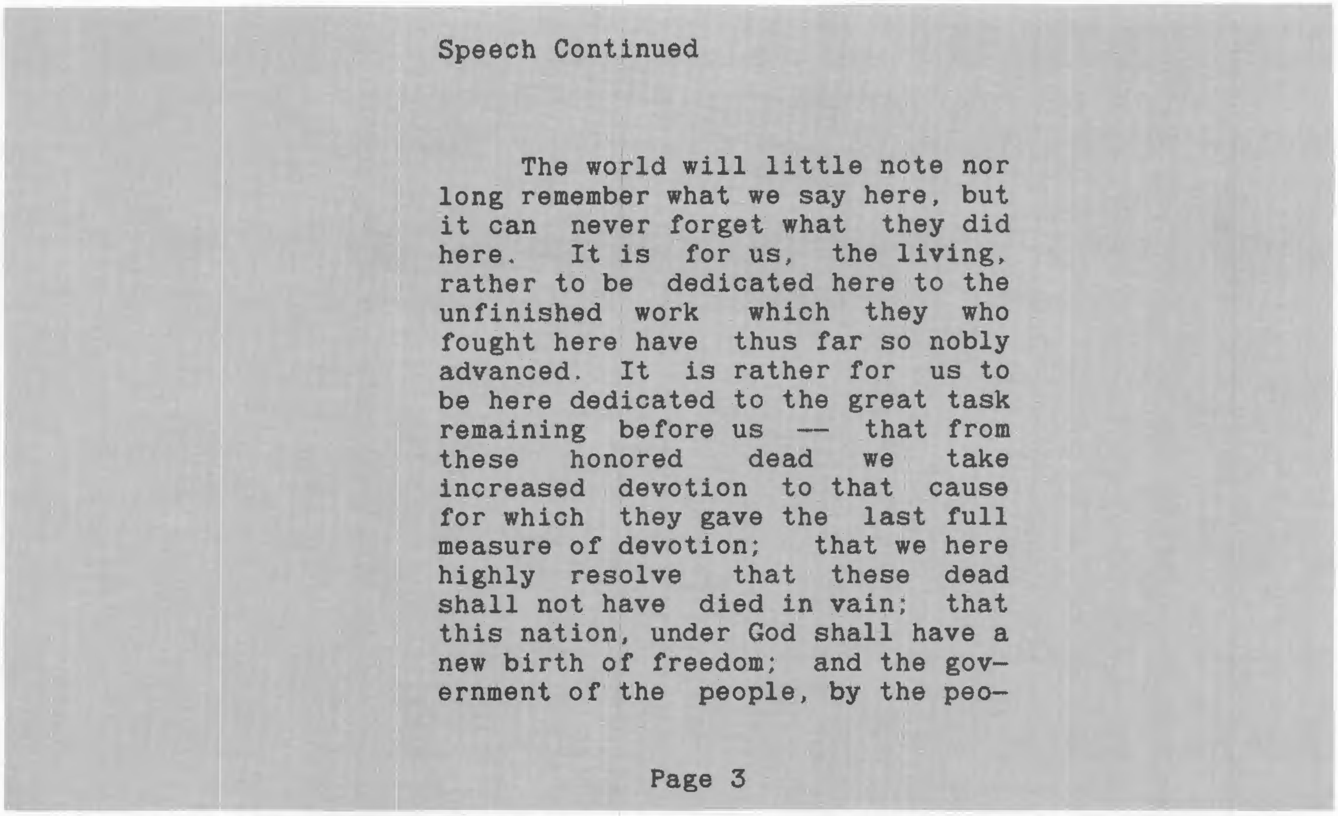
We are met on a great battle-field of that war. We have come to dedicate a portion of that field as a final resting place(1) for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate -- we cannot consecrate -- we cannot hallow -- this ground. The brave men, living and dead, who struggled here(2) have consecrated it far above our poor power to add or detract.

---

(1) Gettysburg, Pennsylvania.

(2) November, 1863.

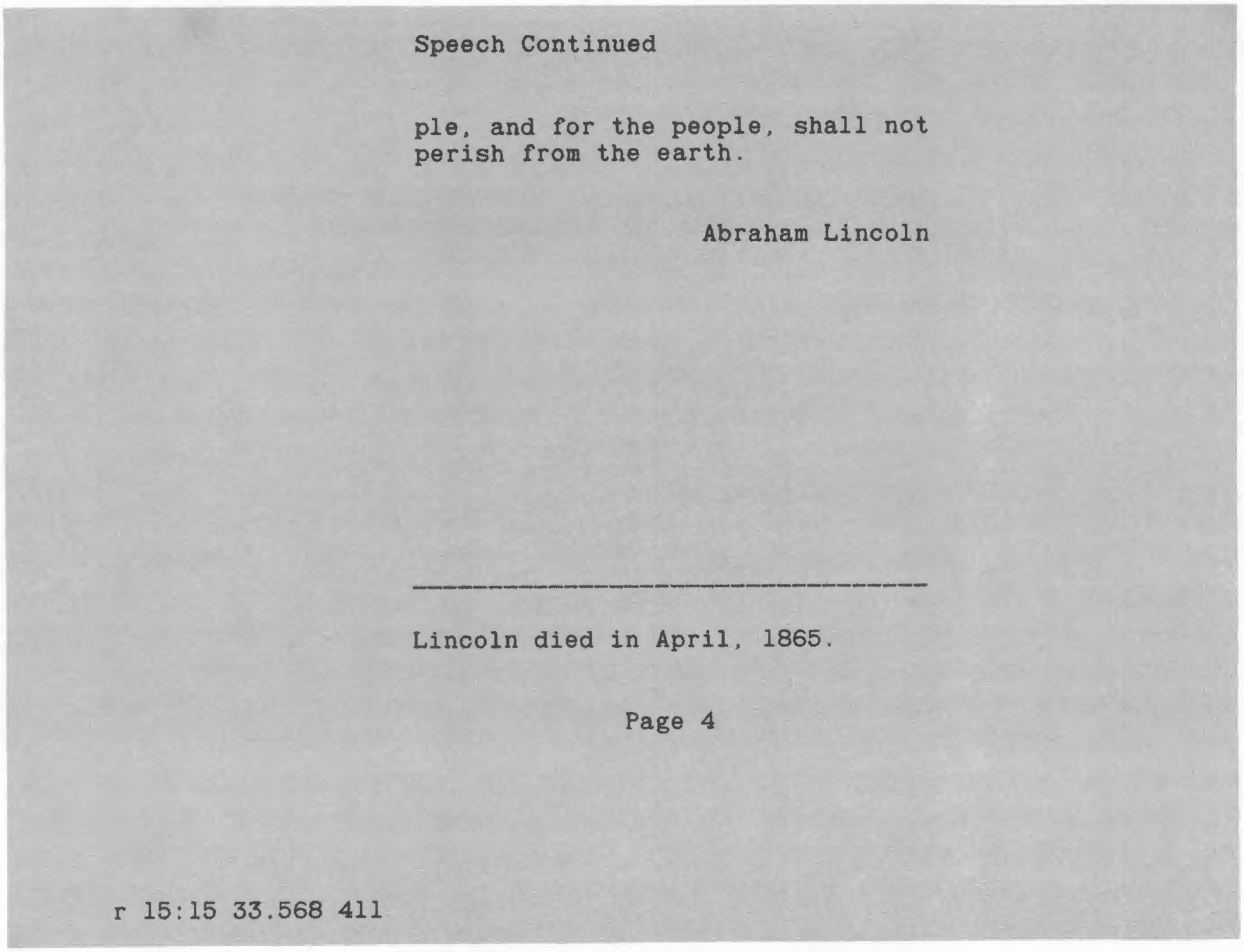
Page 2



Speech Continued

The world will little note nor
long remember what we say here, but
it can never forget what they did
here. It is for us, the living,
rather to be dedicated here to the
unfinished work which they who
fought here have thus far so nobly
advanced. It is rather for us to
be here dedicated to the great task
remaining before us — that from
these honored dead we take
increased devotion to that cause
for which they gave the last full
measure of devotion; that we here
highly resolve that these dead
shall not have died in vain; that
this nation, under God shall have a
new birth of freedom; and the gov-
ernment of the people, by the peo-

Page 3



```
Speech Continued

ple, and for the people, shall not
perish from the earth.

                   Abraham Lincoln




__________________________________

Lincoln died in April, 1865.

              Page 4



r 15:15 33.568 411
```

HOLDING FOOTNOTES

The Multics Software Release 10 has a footnote control line that is not available with the Multics Software Release 9, which was used in this document. That control line is the **footnote hold** control line given as ".fth". With this control line, the compose command will not place footnotes on the page of their reference and will instead hold them by default for insertion at the end of the document. An **insert footnote** control line ".ift" permits the user to insert accumulated footnotes at any point, such as at the end of a section, rather than at the end of the document. If you have Multics Software Release 10, change ".ftp" in your "Lincoln. compin" file to ".fth", overwrite the change, and give the compose command to see the result.

TABLES

The compose command formats tables when horizontal control lines are inserted in the compin segment. Three horizontal tab control lines are required for each table. The first horizontal tab control line to be inserted in the compin segment to format a table is the **horizontal tab define** control line. This control line contains the name of the table and the column positions for the table items (horizontal tab settings). The second control line is the **horizontal tab on** control line, which not only turns on the formatting of the table by the compose command, but also shows the character that is to be used to indicate the tab positioning for the column items. The third control line is the **horizontal tab off** control line.

As an example, consider the following formatted table. In this table, the tab positions are at 5, 10, 20, and 30 spaces with respect to the left margin.

```
No.  Unit 1    Unit 2    Unit 3

1    Col.      Gen.      Capt.
2    Capt.               Lt.
3    Sgt.      Col.
4    Corp.
5              Lt.       Sgt.
6                        Corp.
```

Note that a blank line appears between the column headings and the first line of the column items. For this blank line, a special **space format** control line is used. The space format control line is typed in the compin segment as ".spf" and the compose command considers the formatted blank line as part of the input block rather than as a block separator as it does when the space block control line ".spb" is given. The space format control line can be used with any block such as a text block or a footnote block. Because it is considered as part of a block, it then affects the formatting of text lines.

To format the table, the first control line input to your compin file is the horizontal tab define control line. It is input as ".htd example 5,10,20,30". The expression "example" is your name for the table (not necessarily the name of your compin segment). This name should not contain any spaces. The tab stops are separated by commas and also are input on the control line without spaces.

Following this control line is the horizontal tab on control line which is input as ".htn ; example". Here, the semicolon (;) is the character used to indicate to the compose command which column items are to be placed at which tab stops. A semicolon does not need to be the tab stop character; any character not used within the table can be used although it is best not to use the period (.), greater than (>), or less than (<) symbols. The character chosen should be one easy to recognize when editing is to be performed.

At the end of the table, the horizontal tab off control line must be given. It is typed as ".htf ;" (if the semicolon is used as the tab stop character).

The first line of the example is input in your compin segment as ";No.;Unit 1;Unit 2;Unit 3". The semicolon preceding "No." indicates to the compose command that the "N" is to be formatted at tab stop 5. If the first semicolon were missing, the compose command would place "N" at the left margin (tab stop 1 for the table). Remember that the tab stops are set with respect to the left margin.

Several lines in the table have blank items. The input for line 5, for example, is typed as ";5;;Lt.;Sgt.". The compose command will move past a given tab stop to the next when two tab stop symbols are typed together, just as a typewriter will when the tab key is struck twice.

Input this table using the text editor as follows and give the compose command to see how it is formatted.

NOTE

The control line ".pdl 25" is inserted before the table only as a convenience to keep the compose command from automatically counting 66 lines before the end of the formatted page.

```
! qx                                     ! ted
! a                                      ! a
! .pdl 25                                ! .pdl 25
! .htd example 5,10,20,30                ! .htd example 5,10,20,30
! .htn ; example                         ! .htn ; example
! ;No.;Unit 1;Unit 2;Unit 3              ! ;No.;Unit 1;Unit 2;Unit 3
! .spf                                   ! .spf
! ;1;Col.;Gen.;Capt.                     ! ;1;Col.;Gen.;Capt.
! ;2;Capt.;;Lt.                          ! ;2;Capt.;;Lt.
! ;3;Sgt.;Col.;                          ! ;3;Sgt.;Col.;
! ;4;Corp.;;                             ! ;4;Corp.;;
! ;5;;Lt.;Sgt.                           ! ;5;;Lt.;Sgt.
! ;6;;;Corp.                             ! ;6;;;Corp.
! .htf ;                                 ! .htf ;
! \f                                     ! \f
! w table.compin                         ! w table.compin
```

Note how simple it is to correct or alter your table. For example, if "Lt." were to be spelled out as "Lieutenant", the allotted 10 spaces given — the tab separation — would be used. Only the ".htd" control line (and, of course, the lines containing "Lt.") needs to be changed for new tab stops, rather than the entire table.

Another example using horizontal tab control lines is the compin segment for a table of contents, where most often, dot leaders (periods separated by single white spaces) extend from a title to the page number. Consider the sample table of contents shown below where dot leaders extend backwards from horizontal tab stop 40 to the title for alignment. The other tab stops are 5 and 10.

```
Chapter One . . . . . . . . . . . . .  1
     Subject 1. . . . . . . . . . . .  3

Chapter Two . . . . . . . . . . . . . 15
     Subject 1. . . . . . . . . . . . 17
     Subject 2. . . . . . . . . . . . 23

Chapter Three . . . . . . . . . . . . 33
     Review . . . . . . . . . . . . . 33
     Test . . . . . . . . . . . . . . 41

Appendix A. . . . . . . . . . . . . .A-1
```

The horizontal tab define control line for this example table of contents is input as: .htd ToC 5,10,40" .". That is, following the tab stop 40 is the dot leader expression enclosed in quotation marks. This expression is used by the compose command to separate the title from the page number.

It should also be noted that "Chapter One", "Chapter Two", etc. extend from tab stop 5 past tab stop 10. Consequently, if for those input lines you include the tab stop character for tab stop 10, the compose command will consider that character as extra and part of the page number. That is, when the compose command encounters a character you have defined as a tab stop, it goes to the next available tab position similarly to the way it is done on a standard typewriter. Thus, when a column item extends into the following column space, do not input the tab stop character that will be passed.

Your text editor input for this sample table of contents should be similar to the following:

```
! qx                                    ! ted
! a                                     ! a
! .pdl 25                               ! .pdl 25
! .htd ToC 5,10,40" ."                  ! .htd ToC 5,10,40" ."
! .htn : ToC                            ! .htn : ToC
! :Chapter One:  1                      ! :Chapter One:  1
! ::Subject 1:  3                       ! ::Subject 1:  3
! .spb                                  ! .spb
! :Chapter Two: 15                      ! :Chapter Two: 15
! ::Subject 1: 17                       ! ::Subject 1: 17
! ::Subject 2: 23                       ! ::Subject 2: 23
! .spb                                  ! .spb
! :Chapter Three: 33                    ! :Chapter Three: 33
! ::Review: 33                          ! ::Review: 33
! ::Test: 41                            ! ::Test: 41
! .spb                                  ! .spb
! :Appendix A:A-1                       ! :Appendix A:A-1
! .htf :                                ! .htf :
! \f                                    ! \f
! w ToC.compin                          ! w ToC.compin
! q                                     ! q
```

Section 9
Formatting Refinements

A **text block** consists of a block of formatted output that is treated as a unit. In the preceding sections, you have encountered text blocks in the form of paragraphs. You have also encountered header blocks, footer blocks, and title blocks. Footnotes, likewise, form text blocks.

Tables are text blocks that usually should not be broken (part of the table on one page and part on the following page). Title blocks consisting of mathematical expressions or notes, etc. should not be broken between pages. For those text blocks that you wish not to be broken, the compose command will recognize block keep control lines that are typed within the compin segment.

KEEP BLOCKS

To keep a block of text, such as a table, on one page instead of being split between two pages, two block keep control lines are required. Prior to the horizontal tab define control line, the **block begin keep** control line ".bbk" must be inserted. Then immediately after the end of the table, after the horizontal tab off control line, the **block end keep** control line ".bek" must be typed.

When these two control lines are incorporated in the compin segment, the compose command will alter the formatted output in such a way that the keep block will not be split. The point within the compin file at which the block begin keep control line is placed is the point where the compose command will immediately stop formatting its current text block. That is, if ".bbk" in inserted within the middle of a sentence, the compose command will immediately begin designating lines of output as lines that cannot be split. After ".bek", the compose command will return to formatting the broken sentence on a new line.

If the compose command cannot find a way to format a page so as to accommodate a keep block within the space that is available on the page where it received the block begin keep control line, it will break the block as though there were no keep block. That is, if the keep block is too large for the page, part will appear in the available space while the remainder of the block will be placed on the following page. The compose command will give an error message in the compose error list at the end of the formatted file when it cannot accommodate a keep block. You may then want to change your formatting control lines so that the keep block is formatted on one page.

If you know the exact number of output lines to be kept, you can replace the two block keep control lines with one. Thus, if you know the block to be kept contains 5 lines of input text, you can replace the block begin keep control line ".bbk" with ".bbk 5" and you do not need to input the block end keep control line ".bek" in your compin segment. Note that this technique should be used only for small keep blocks in order to avoid errors due to text changes and the miscounting of lines.

TITLE LINE CAPTION

Many documents require the formatted output to contain space where illustrations or photographs must be inserted prior to final printing. And, of course, this space cannot be split between two pages. One way to include this space is to use a **title line caption** control line, which is similar to the title line header control line. The title line caption control line is typed as ".tlc N M ||Figure caption||" where the figure caption may be placed either at the left, right, or center of the formatted output. In the title line header control line, "N" is the number of blank lines from your title *down* to the next text block. However, in the title line caption control line, "N" is the number of lines (illustration space) *up* from your caption to the last line of text preceding the illustration. "M" is the same for both control lines — the left indentation with respect to the current left margin.

KEEP AND CAPTION EXAMPLES

To see how the compose command will keep a block as an entity and how the compose command will format a title line caption control line, again invoke the text editor and read your "Lincoln.compin" file. For the keep block example, you will insert your "table.compin" file immediately after the second footnote. You will place five lines and a caption for an illustration after the phrase "this ground".

NOTE

> You may, at this time, want to see a print of the first text paragraph by requesting "Fourscore/,/endure./ p" or you may prefer to review the last print of this file in Section 8. If you changed ".ftp" to ".fth" in section 8, change ".fth" back to ".ftp" here before continuing on.

Locate "testing" and insert in front of this expression a space block control line, ".spb", and the block begin keep control line ".bbk". Next read your "table.compin" file. Recall that your table contains a page length control line, ".pdl 25". You will need to delete this control line. At the end of the table, after ".htf;", append the block end keep control line ".bek" and a space block control line ".spb". The space block control lines are used to separate the table from adjacent text blocks.

```
  r 09:28 0.090 9

! qx
! r Lincoln.compin
! /testing/ i
! .spb
! .bbk
! \f
! /.bbk/ r table.compin
! /.pdl 25/ d
! /.htf/ a
! .bek
! .spb
! \f
```

```
  r 09:28 0.090 9

! ted
! r Lincoln.compin
! /testing/ i
! .spb
! .bbk
! \f
! /.bbk/ r table.compin
! /.pdl 25/ d
! /.htf/ a
! .bek
! .spb
! \f
```

You will also need to insert your second page header line control line before this table or the first page header line will appear on page 2. That is, the insertion of the table causes the present second header control line to be too far in the compin segment to print on page 2. Although it will be disregarded by the compose command, the present second header control line should then be deleted in order to avoid possible confusion if this segment should again be edited.

```
  \f
! /we are en/ a
! .hla 1 0 |Speech Continued|||
! \f
! /.hla/ d
```

Recall that for a locate request, the text editor searches from the current line down. Therefore, to delete the second header control line, simply request a search for ".hla" and then request its deletion. (Alternatively, you could search for "endure" and then delete the header control line by "+2d".)

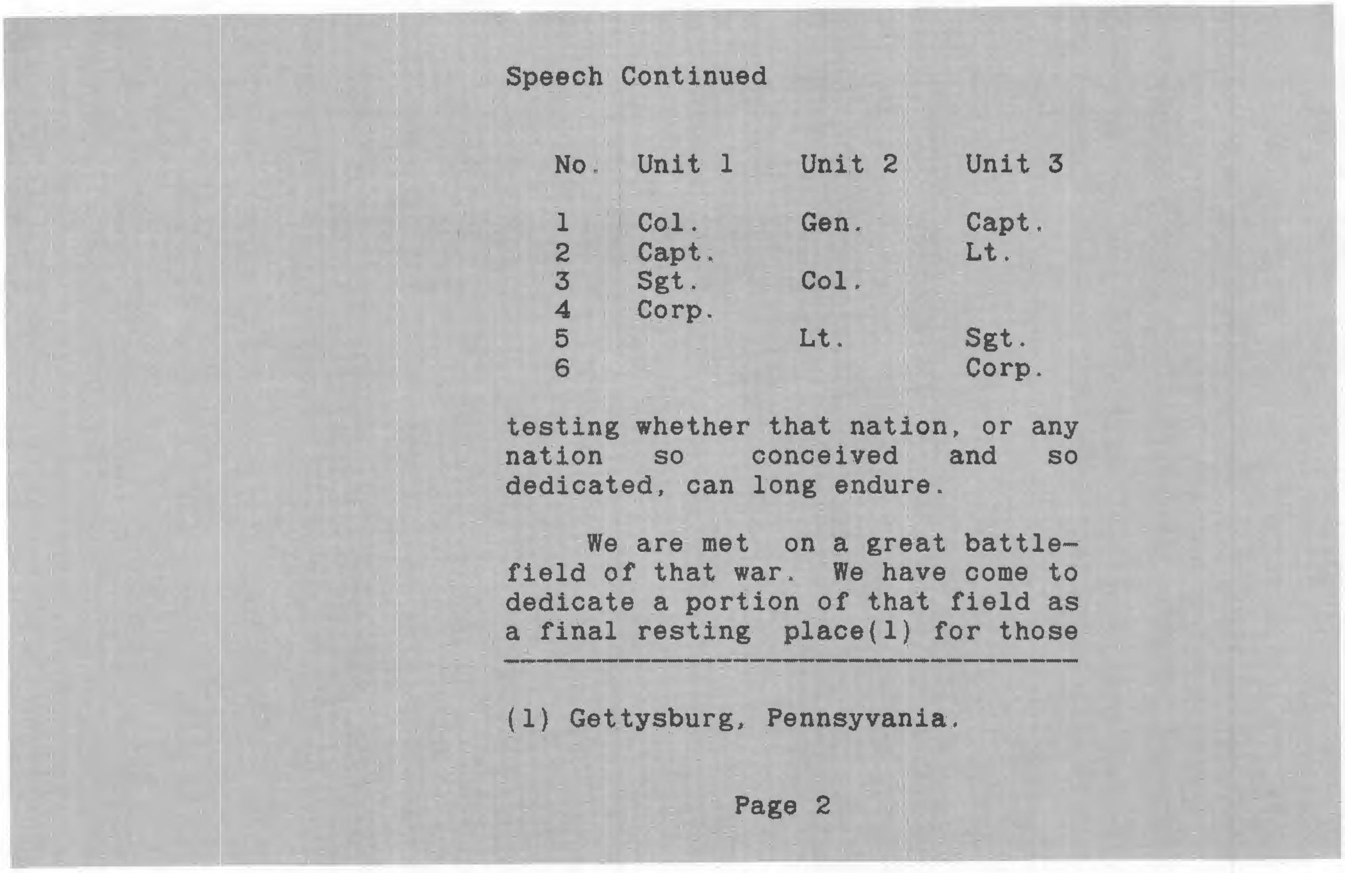
Append ".tlc 5 0 ||Figure No. 1||" after the expression "this ground." to format the title line caption control line.

```
  /.hla/ d
! /this ground./ a
! .tlc 5 0 ||Figure No. 1||
! \f
! w Lincoln.compin.
```

Request the editor to write your file. Note that because your buffer contains two files (Lincoln.compin and table.compin), you must in this case supply the pathname "Lincoln.compin" or the editor will print the message "No pathname given." Ordinarily, you will recall, you could request the editor to overwrite by simply typing "w". Before you quit the editor, request a print of the segment from "civil war" to "brave men" to check your input. Then give the compose command to format this compin segment using the hyphenation and indent control arguments you used for earlier formatted output. Your compin segment and your formatted output should appear similar to the following:

```
  w Lincoln.compin
! /civil war/,/brave men/ p
  we are engaged in a great civil war,
  .hla 1 0 |Speech Continued|||
  .bbf
  1861 through 1865.
  .bef
  .spb
  .bbk
  htd example 5,10,20,30
  .htn ; example
  ;No.;Unit 1;Unit 2;Unit 3
  .spf
  ;1;Col.;Gen.;Capt.
  ;2;Capt.;;Lt.
  ;3;Sgt.;Col.;
  ;4;Corp.;;
  ;5;;Lt.;Sgt.
  ;6;;;Corp.
  .htf ;
  .bek
  .spb
  testing whether that nation, or any nation
  so conceived and so dedicated, can long endure.
  .spb
  .unl -5
  We are met on a great battlefield of that war.
  We have come to
  dedicate a portion of that field as a final resting place
  .bbf
  Gettysburg, Pennsylvania.
  .bef
  for those who here gave their lives
  that that nation might live.  It
  is altogether fitting and proper that we should do this.
  But, in a larger sense, we cannot dedicate -- we
  cannot consecrate -- we cannot hallow -- this ground.
  .tlc 5 0 ||Figure No. 1||
  The brave men,  living and dead, who struggled here
! q
  r 09:32 0.628 62
```

```
! comp Lincoln -hyph -ind 20

                                                  Speech


                                      GETTYSBURG
                                        ADDRESS


                                    Fourscore   and  seven   years
                              ago(1) our fathers brought forth on
                              this   continent   a   new  nation,
                              conceived in  liberty and dedicated
                              to the proposition that all men are
                              created equal.  Now  we are engaged
                              in a great civil war,(2)

                              ___________________________________

                              (1) 87 years after 1776.

                              (2) 1861 through 1865.


                                        Page 1
```



Speech Continued

No.	Unit 1	Unit 2	Unit 3
1	Col.	Gen.	Capt.
2	Capt.		Lt.
3	Sgt.	Col.	
4	Corp.		
5		Lt.	Sgt.
6			Corp.

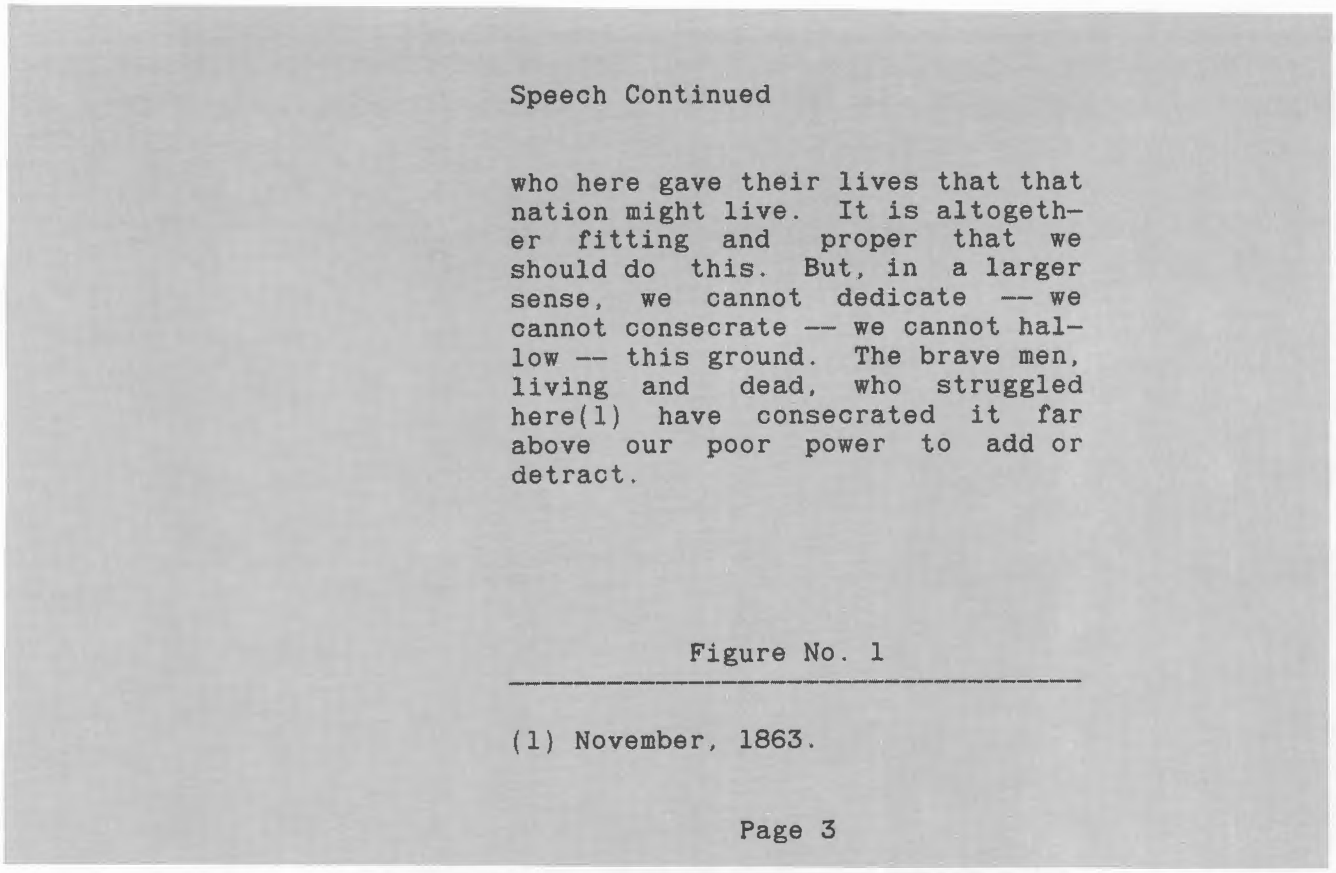
testing whether that nation, or any nation so conceived and so dedicated, can long endure.

We are met on a great battle-field of that war. We have come to dedicate a portion of that field as a final resting place(1) for those

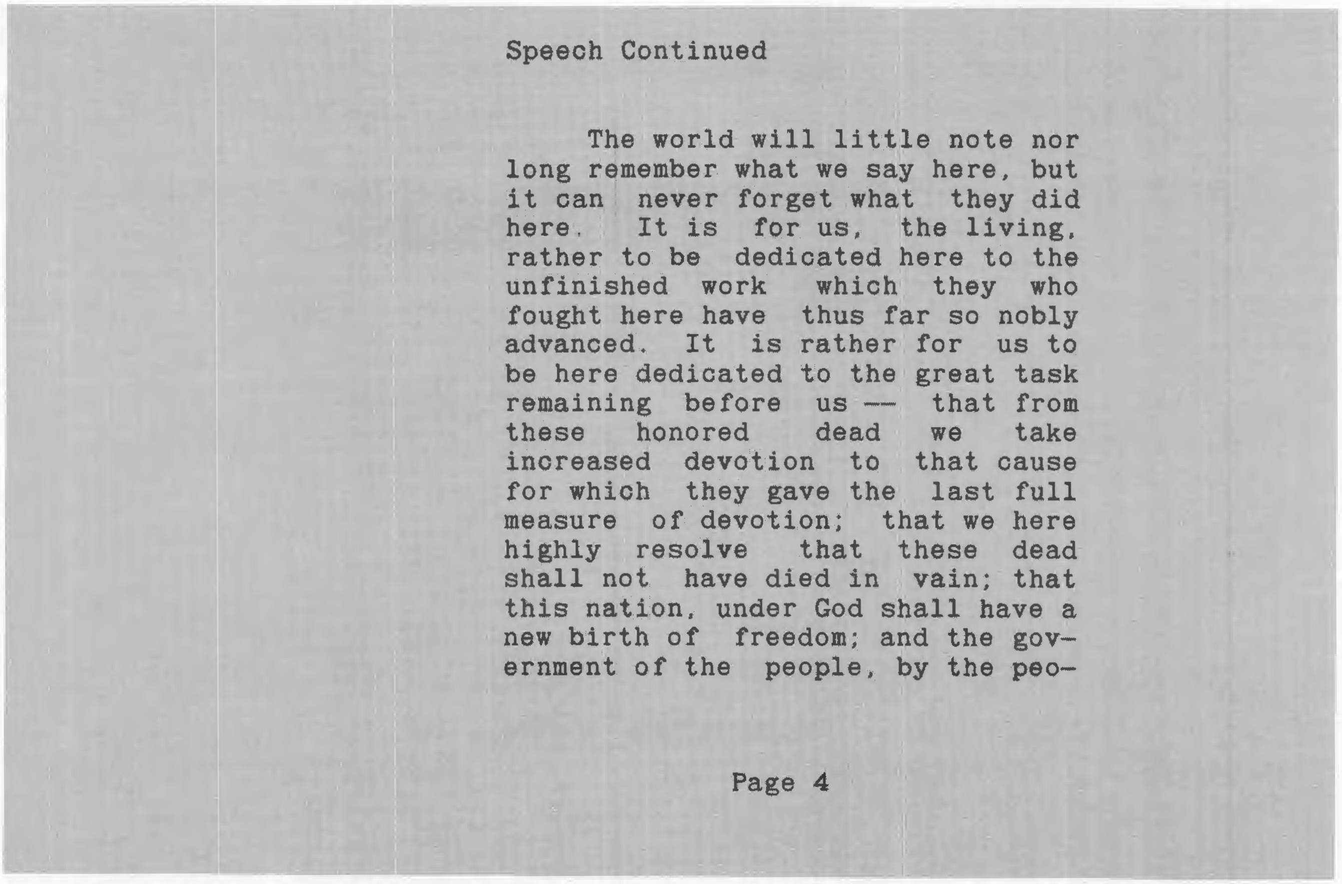
---

(1) Gettysburg, Pennsyvania.

Page 2



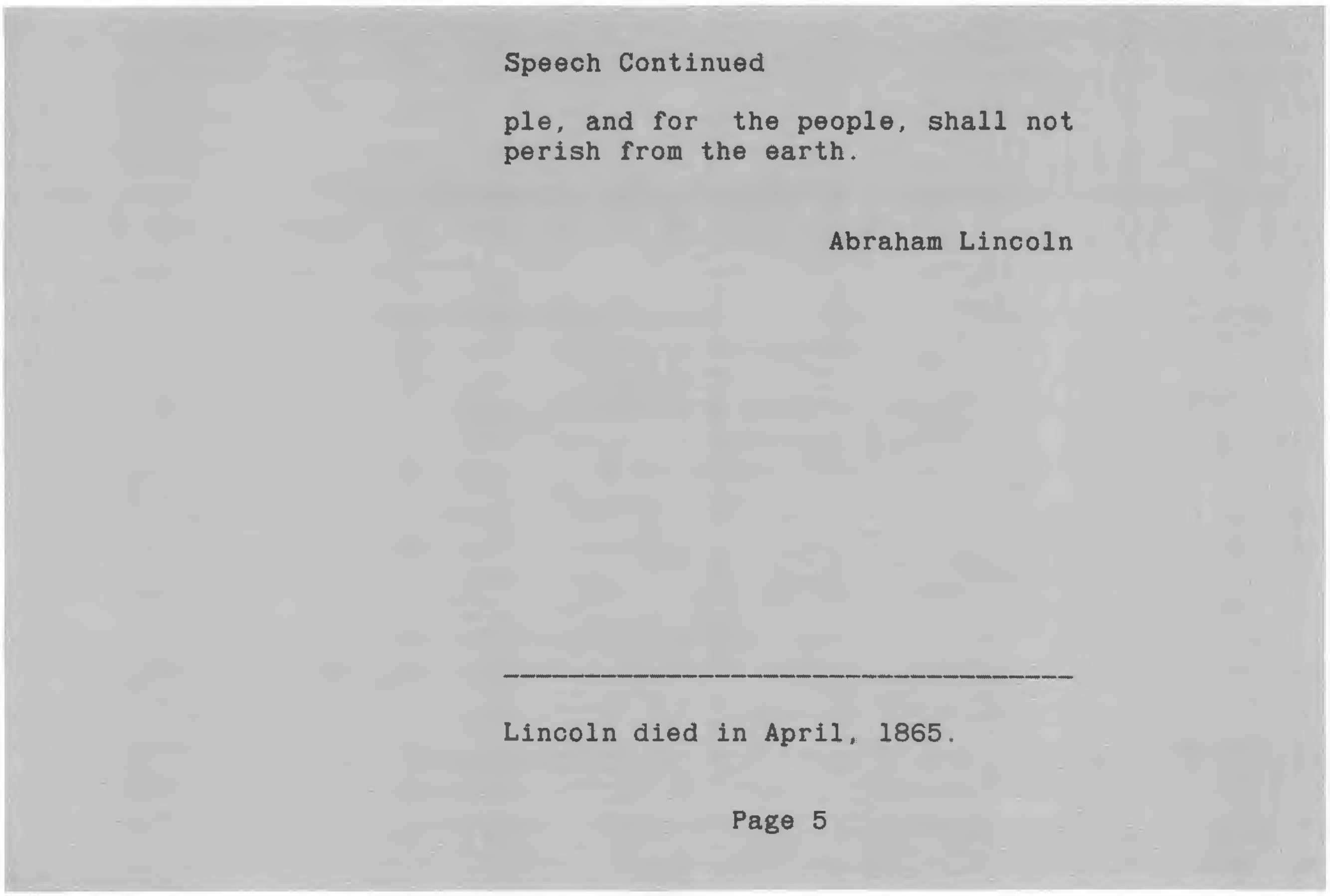
```
                   Speech Continued


                   who here gave their lives that that
                   nation might live.  It is altogeth-
                   er  fitting  and   proper  that  we
                   should do  this.  But, in  a larger
                   sense,  we  cannot  dedicate  -- we
                   cannot consecrate -- we cannot hal-
                   low -- this ground.  The brave men,
                   living  and   dead,  who  struggled
                   here(1)  have  consecrated  it  far
                   above  our  poor  power  to  add or
                   detract.




                             Figure No. 1
                   ___________________________________

                   (1) November, 1863.


                                Page 3
```



Speech Continued

The world will little note nor
long remember what we say here, but
it can never forget what they did
here. It is for us, the living,
rather to be dedicated here to the
unfinished work which they who
fought here have thus far so nobly
advanced. It is rather for us to
be here dedicated to the great task
remaining before us — that from
these honored dead we take
increased devotion to that cause
for which they gave the last full
measure of devotion; that we here
highly resolve that these dead
shall not have died in vain; that
this nation, under God shall have a
new birth of freedom; and the gov-
ernment of the people, by the peo-

Page 4



```
                        Speech Continued

                        ple, and for  the people, shall not
                        perish from the earth.

                                          Abraham Lincoln










                        -----------------------------------

                        Lincoln died in April, 1865.

                                      Page 5
```

In this latest formatted output of your "Lincoln.compin" segment, observe that the compose command has formatted page 1 with white space and placed that table at the top of page 2. The last text block line on page 1 is a short line and the first text block line after the table starts on a new line because the block begin keep control line stops normal formatting. (Compare this output with the one in Section 8.)

The compose command has inserted space for Figure No. 1 at the end of the paragraph in which the title line caption control line was inserted in the compin segment. That is, the control line was inserted after the expression "this ground.", but the compose command waited until space was available (at the end of the paragraph) without breaking the text block format.

Also observe that all footnotes have been renumbered to apply to the pages of their reference.

UNDERLINING

The compose command, when given a control line in the compin segment, will underline any expression you want in the formatted output. The control line required is the **use reference** control line. This control line contains the expression between percent symbols (%) and square brackets ([,]). The line is typed as ".ur %[underline expression]%".

The expression in the control line can be any single word; several words; or the text of a footer, header, or title. To underline a single word within a compin segment (an input line), the control line would be typed similarly to the following:

.ur abc defg %[underline hijkl]% mnop

The compose command will format this line as "abc defg hijkl mnop." For two or more words in a regular expression, the input line would take the form of ".ur abc %[underline defg hijkl]%" for which the compose command will format this line as "abc defg hijkl".

Quotation marks around the expression will cause the compose command to underline the entire expression including the white spaces. For example, ".ur abc %[underline "defg hijk"]%" will be formatted as "abc defg hijk".

NOTE

> Use care in underlining spaces in long expressions. The compose command recognizes material within quotation marks as a single word and may give an impression that the fill on control feature has failed.

You may underline a header line by typing a control line: ".ur .hla 1 0 |%[underline header]%||| ". Note that in all cases, you must distinguish between words and spaces to be underlined from those that should not be underlined.

As an example of these three possible control lines, invoke the text editor and read your "Lincoln.compin" file again. Request "/ground/,/the living/ p" and note the word "power", the line "have consecrated it", and the title line caption control line. Change these lines so that the compose command will underline the word "power", the entire phrase "have consecrated it" (including the spaces), and the figure caption. After you have overwritten these changes in your file and quit your editor, give the compose command to format page 3 of this compin segment. Use hyphenation and indent 20 spaces. The input and output should appear similar to the following:

```
! qx                                          ! ted
! r Lincoln.compin
! /ground/,/the living/ p
  cannot consecrate -- we cannot hallow -- this ground.
  .tlc 5 0 ||Figure No. 1||
  The brave men, living and dead, who struggled here
  .bbf
  November, 1863.
  .bef
  have consecrated it
  far above our poor power to add or detract.
  .spb
  .unl -5
  The world will little note nor long remember
  what we say here, but it can never forget
  what they did here.
  It is for us,
  the living, rather to be dedicated here
! /power/ c
! .ur far above our poor %[underline power]% to add or detract.
! \f
! /have consecrated it/ c
! .ur %[underline "have consecrated it"]%
! \f
! /.tlc/ c
! .ur .tlc 5 0 ||%[underline Figure No. 1]%||
! \f
! /ground/,/the living/ p
  cannot consecrate -- we cannot hallow -- this ground.
  .ur .tlc 5 0 ||%[underline Figure No. 1]%||
  The brave men, living and dead, who struggled here
  .bbf
  November, 1863.
  .bef
  .ur %[underline "have consecrated it"]%
  .ur far above our poor %[underline power]% to add or detract.
  .spb
  .unl -5
  The world will little note nor long remember
  what we say here, but it can never forget
  what they did here.
  It is for us,
  the living, rather to be dedicated here
! w Lincoln.compin
! q
  r 15:06 0.660 149

! comp Lincoln -hyph -ind 20 -pgs 3
```

```
                    Speech Continued


                    who here gave their lives that that
                    nation might live.  It is altogeth-
                    er  fitting  and   proper  that  we
                    should do  this.  But, in  a larger
                    sense,  we   cannot  dedicate  -- we
                    cannot consecrate -- we cannot hal-
                    low -- this ground.  The brave men,
                    living  and   dead,  who  struggled
                    here(1)  have consecrated it   far
                    above  our  poor  power  to  add or
                    detract.



                              Figure No. 1
                    ___________________________________

                    (1) November, 1863.


                                 Page 3
```

TRANSLATE FORMAT

Often there are times when you may wish to keep words of an expression together in a document so that part of the expression does not appear formatted on one line and part on another (analogous to a keep block of text). For example, if the expression "Dr. John Smith " appeared within a compin segment, the compose commands may very well format "Dr." at the end of one line and "John Smith" at the beginning of a new line. (Although splitting a name and title between lines is typographically correct, the preferred style in modern English is to keep them together.) To offset the possibility of splitting an expression such as this, the compose command will recognize a **translate format** control line.

The translate format control line is usually typed at the beginning of your compin segment (in the housekeeping section) as ".trn !". The result will be that every time the compose command sees the exclamation point, it will replace that exclamation point in the formatted output with a blank space and simultaneously treat the expression containing the exclamation point as one unbreakable expression. Thus, to keep "Dr. John Smith" formatted on one line, ".trn !" would be typed at the beginning of the compin segment and the expression would be typed in your input text as "Dr.!John!Smith".

The translate format control line actually uses paired characters. In other words, suppose you typed the control line as ".trn abcd". The compose command when formatting the output will translate and replace every occurrence of "a" in the compin segment with "b" in the formatted output and every occurrence of "c" in the compin segment with "d" in the output. When you type ".trn !", the compose command sees a blank space following the exclamation point and replaces every occurrence of the exclamation point with a blank space.

The use of the exclamation point is common in the translate control line, but as shown above where "abcd" was used, any alphanumeric character can be used. It is best, however, not to use those characters considered "special" and it is simpler not to use a character appearing literally in the segment. If you were to use the exclamation point as the translate format control line character and also needed it for punctuation, you can turn off the control by typing ".trn !!" (replace every occurrence of the exclamation point with an exclamation point). The control can then be turned on when needed by again inserting ".trn !".

NOTE

> In some cases you may need to turn off the translate format control line several text lines before the translate symbol is required as a literal in your text.

As an example, append to line 5 of your "Lincoln. compin" segment ".trn !". Locate the expression "nor long remember" and change it to read "nor!long!remember". Overwrite your file, quit your editor, and give the compose command to format page 4 of this file with hyphenations and indenting 15 spaces. Compare this page with a previously formatted page 4.

```
! qx                                             ! ted
! r Lincoln.compin
! 5a
! .trn !
! \f
! /world/ c
! The world will little note nor!long!remember
! \f
! w
! q
  r 15:10 0.284 90
```

```
! comp Lincoln -hyph -ind 15 -pgs 4


                 Speech Continued


                      The   world   will   little   note
                 nor long remember what we say here,
                 but it  can never forget  what they
                 did here.  It is for  us, the liv-
                 ing, rather to be dedicated here to
                 the unfinished work  which they who
                 fought here have  thus far so nobly
                 advanced.  It  is rather for  us to
                 be here dedicated to the great task
                 remaining  before  us --  that from
                 these    honored    dead    we    take
                 increased  devotion  to  that cause
                 for which  they gave the  last full
                 measure  of devotion;  that we here
                 highly  resolve   that  these  dead
                 shall not  have died in  vain; that
                 this nation, under God shall have a
                 new birth of  freedom; and the gov-
                 ernment of the  people, by the peo-


                            Page 4
```

CONTROL ARGUMENTS FOR THE COMPOSE COMMAND

In the previous sections, you have been introduced to several control arguments that are typed following the compose command. Without these control arguments, the compose command will format the compin segment from the beginning to the end without hyphenating or stopping. Other control arguments provide for stopping after each page, a galley copy, line numbers, and double-spacing. With those control arguments described thus far, the compose command will hyphenate, indent, and format only those pages you want formatted. All control arguments are preceded by a hyphen and are separated from each other by a blank space.

Hyphenate

–hyph

The –**hyphenate** control argument causes the compose command to compare words in the compin segment with the Multics online dictionary. If the word is in the dictionary and can be hyphenated, the compose command, if necessary, will use a portion of that hyphenated word to fill a line.

Indent

–indent N or –ind N

Without the –**indent** control argument, the compose command begins formatting (according to the control lines in the compin segment) at the left margin of the paper used for the output. The –indent control argument thus lets you place your formatted output as many spaces to the right of the left margin as you desire (to center the output on the page, for example).

Pages

–pages N,M or –pgs N,M

The –**pages** control argument used by the compose command produces a print of only those pages you want printed. For a single page of the formatted output, the control argument is "–pages N", where "N" is the page number of the wanted page. For two or more continuous pages, "N" is the number of the first page wanted and "M" is the number of the last page wanted.

From

–from N or –fm N

The **–from** control argument is used to output from a given page to the end of the document. If a document, for example, has 50 pages and the –from control argument is given as "–fm 23", the compose command will output pages 23 through 50.

To

–to N

The **–to** control argument is the opposite of the "–from" control argument. When the –to control argument is given, the compose command formats page 1 through page N. The –to control argument is usually used with the –from control argument to get a particular range of pages. (Note that "–fm N –to M" can be replaced by "–pages N,M".)

Stop

–stop or –sp

The **–stop** control argument lets you review each page as it is formatted. However, its greatest use occurs when you need time to change paper or each page is a special form. Before the compose command begins formatting, it waits for a carriage return. As the compose command formats, it will stop at the end of page 1 and continue only after you have pressed the carriage return. Then, the compose command will format page 2 and stop, again waiting for a carriage return before continuing.

Galley

–galley X,Y or–gl X,Y

The –**galley** control argument provides a continuous formatted output without headers or footers from compin segment line number X through line number Y. It provides an easy way to check the format of a portion of your compin segment, such as the structure of horizontal tab material. Default values for the –galley control argument are the first line and the last line of the compin segment. Because the print is not divided into pages, any footnotes that may be present are numbered consecutively throughout the document and appear at the end of the paragraph in which they are given. Note that the –galley control argument cannot be used with the –from, –to, –page, or –stop control arguments.

NOTE

Conflicting control arguments usually result in an error message and no formatted output from the compose command.

Number

–number or –nb

On occasion, for example, with draft copies, you may desire a correlation of the formatted output with the compin segment. The –**number** control argument causes the compose command to list the absolute line number within the compin segment at the left of the corresponding line of the formatted output. The line number given is the line on which the first word of the formatted line is found in the compin segment.

Linespace

–Linespace N or –ls N

The –**linespace** control argument is used when extra space is wanted for marking changes to a draft copy. By default, this control argument is "–ls 1". If you want your formatted output to be double-spaced or triple-spaced, the –linespace control arguments are "–ls 2" and "–ls 3" respectively.

CONTROL ARGUMENT EXAMPLE

To see how the compose command formats when some of these control arguments are added, give the compose command to format your "Lincoln.compin" segment from lines 40 through 70 in galley form indented 5 spaces. Add control arguments for double-spacing and line numbering. That is, give the compose command as follows:

```
comp Lincoln –gl 40, 70 –ind 5 – ls 2 –nb
```

Your output will be double-spaced with line numbers corresponding to the compin segment. The output will be continuous and without hyphenation, but it will show footnotes. (The column of "l's" refers to an index file, which you may disregard.)

```
! comp Lincoln -gl 40,70 -ind 5 -ls 2 -nb
  1    40    dedicated, can long endure.

  1    43         We   are   met   on   a  great

  1    43    battlefield  of that  war.  We have

  1    44    come to dedicate  a portion of that

  1    44    field as  a  final resting place(3)

  1    48    for those who here gave their lives

  1    49    that that nation might live.  It is
```

```
1   50   altogether fitting  and proper that
1   50   we  should  do  this.  But,  in  a
1   51   larger sense, we cannot dedicate --
1   51   we  cannot consecrate  -- we cannot
1   52   hallow  -- this  ground.  The brave
1   54   men, living and dead, who struggled
1   54   here(4)  have consecrated it   far
1   59   above  our  poor  power  to  add or
1   59   detract.
1

1   53             Figure No. 1
         -----------------------------------
1   45   (3) Gettysburg, Pennsylvania.
1   55   (4) November, 1863
         -----------------------------------
1   62        The  world  will  little  note
1   62   nor long remember what we say here,
1   63   but it  can never forget  what they
1   64   did  here.  It  is  for  us,  the
1   66   living, rather to be dedicated here
1   67   to  the unfinished  work which they
1   67   who  fought here  have thus  far so
1   68   nobly  advanced.  It  is rather for
1   69   us  to  be  here  dedicated  to the
1   70   great task remaining
 1   Lincoln
```

Section 10
Dictionary Usage

A standard dictionary is incorporated in the Multics system, and as you have seen, can be used by the compose command for hyphenation in order to reduce white space within a line when formatting a text block. You can use this same dictionary to detect spelling errors within any of your files (compin or not). Additionally, you can create and use only your own dictionary (or dictionaries). Or, you can create your own dictionary and use it with the standard dictionary.

This section describes how to use the standard dictionary to check spelling of the words within a file; how to create your own dictionary; and how to use your own dictionary and the standard dictionary together.

ONLINE STANDARD DICTIONARY

The standard online Multics dictionary contains approximately 30,000 words. In general, this dictionary does not contain specialized, technical, scientific, medical, or foreign words. Hyphenations have been selected as those most commonly used because different authorities vary in many cases in the way a word is to be hyphenated. (You can change the way a word is hyphenated by including that word in the dictionary you create.) Those words that can be hyphenated are hyphenated except homographs, which are words of identical spelling but of different meaning and pronunciation depending upon the use of the word. For example, the word "resume" is not hyphenated in this dictionary because the syllabification for the word as a noun meaning "a summary" is different from the syllabification of the word when it is used as a verb meaning "begin again".

CREATE WORDLIST

In order to see how the online dictionary can be used for checking spelling, invoke your editor, read your compin file "Lincoln.compin", and deliberately substitute misspelled words "sevne" for "seven", "natiom" for "nation", and preposition" for "proposition" in the first text block.

```
  r 07:37 0.197 27

! qx
! r Lincoln.compin
! /seven/ s//sevne/
! /nation/ s//natiom/
! /proposition/ s//preposition/
! w
! q
  r 07:39 0.369 29
```

```
  r 07:37 0.197 27

! ted
! r Lincoln.compin
! /seven/ s//sevne/
! /nation/ s//natiom/
! /proposition/ s//preposition/
! w
! q
  r 07:39 0.369 29
```

The first step toward checking the spelling of the words in a file is to create a list of all the words in that file. You create this list with the **create_wordlist** command whose short name is "cwl". When the Multics system receives the command line "cwl pathname", it counts all the words in the file named in the command line. Then, all duplicated words are discarded and a count is made of all the remaining words, called **unique words**. These unique words are put in a new wordlist segment (created by the "cwl" command) whose name is the name you give in the "cwl" command line, plus a suffix of "wl".

Thus, to find the total number of words in your "Lincoln.compin" file and the number of unique words, issue the command "cwl Lincoln.compin". If your compin file is identical to the one described in this document, and if there are no errors other than the three just inserted, you will receive a response from the Multics system as shown. The example shows that there are a total of 343 words in the file of which 197 are unique (based on the current standard dictionary that periodically may be updated or revised). These unique words are in a new segment named "Lincoln.compin.wl".

```
! cwl Lincoln.compin
  total number of words = 343
  number of unique words = 197
  r 07:39 0.723 20
```

TRIM WORDLIST

The **trim_wordlist** command is used next; its short name is "twl." After receiving the command line "twl pathname", the Multics system compares the unique words from the created wordlist with the words in the standard dictionary. All the words in the unique list that are in the dictionary are discarded (trimmed) and the number of the remaining words is given as a response.

Type "twl Lincoln.compin" and note that 126 words of the 197 are listed in the online dictionary while 71 are not.

```
! twl Lincoln.compin
  number of words trimmed = 126
  number of words remaining = 71
  r 07:39 0.779 133
```

PRINT WORDLIST

The **print_wordlist** command, whose short name is "pwl", is invoked as "pwl pathname". It commands the Multics system to print the words in a wordlist segment. Type "pwl Lincoln.compin" and note that all 71 words are listed, among them all the various control line forms and other expressions you normally would not find in a dictionary. Also note that the words are listed alphabetically by category.

```
    ! pwl Lincoln.compin


      April                   resting          .tlc
      birth                   sevne            .tlh
      brave                   struggled        .trn
      conceived               u                .unl
      consecrated             vain             .ur
      continued|||            war              .vmb
      dead                    %PageNo%||       .vmt
      died                    %[underline      1;Unit
      earth                   bbf              2;Unit
      fathers                 .bbk             ;No.;Unit
      fought                  .bbt             ;1;Col.;Gen.;Capt
      Gettysburg              .bef             ;2;Capt.;;Lt
      God                     .bek             ;3;Sgt.;Col
      ground                  .bet             ;4;Corp
      it"]%                   .fla             ;5;;Lt.;Sgt
      Lincoln                 .ftp             ;6;;;Corp
      Lincoln|                .hla             |Speech
      lives                   .htd             ||ADDRESS||
      men                     .htf             ||GETTYSBURG||
      natiom                  .htn             ||Page
      nor!long!remember       .pdl             ||%[underline
      Pennsylvania            .pdw             |||Abraham
      poor                    .spb             |||Speech|
      power]%                 .spf


    r 07:40 0.290 14
```

NOTE

Both "twl" and "pwl" work only on word-list segments created by the "cwl" command; that is, segments having the suffix "wl". However, the suffix "wl" does not need to be typed in the command line.

In general, those words not listed are expressions consisting only of numbers, punctuation marks, or other special characters. If an expression contains punctuation marks and/or special characters (such as "power]%") or numbers with alphabetical characters and without spaces, that expression is not trimmed and is listed.

Usually, punctuation marks preceding and following a word are removed. For example, the abbreviation "i.e." would be listed as "i.e" with the trailing period removed. But a period preceding a word, as in a compin control line such as ".spb", is not removed.

You can now check this printed list for all words that might be misspelled in your file. Observe that two of the three errors deliberately inserted in the file earlier are listed. "Sevne" and "natiom" are listed but "preposition" is not. "Preposition" is a correctly spelled word, and the Multics system has matched it with its mate in the dictionary. The Multics system cannot differentiate whether a word is used correctly or not – it can check only the spelling of a given word.

Had you given the "pwl" command before the "twl" command, you would have received a print of all 197 words including "the" and "The", "but" and "But", and "it" and "It". The trim_wordlist command, of course, has matched the spelling of a word regardless of its initial capitalization.

The three commands given at this point have produced a list of 71 words and expressions. Control arguments can be used with these commands to further reduce this list. For example, you may have no need to see the control line listing or you may want to check the spelling of only a portion of your file.

To trim the control lines from your listing, add the **-no_control_lines** (or "-ncl") control argument to the create_wordlist command line. To check only a portion of your file, the added control arguments are the "-from" and "-to" control arguments with the short names "-fm N" and "-to N", where each "N" is the absolute line number of that portion of your file you wish to check. The -from control argument is used to indicate the line number for the start of the portion you want to check. By default, "N" is line 1. The -to control argument gives the last line of the portion you want to check and, by default, this "N' is the last line of your file.

Thus, to create a wordlist of your "Lincoln.compin" file for absolute line numbers from 5 through 55 and without including any of the control lines, "cwl Lincoln.compin -ncl -fm 5 -to 55" would be the command that you would give. Then trim your list by typing "twl Lincoln.compin".

```
! cwl Lincoln.compin -ncl -fm 5 -to 55
  total number of words = 140
  number of unique words = 97
  r 07:41 0.222 28

! twl Lincoln.compin
  number of words trimmed = 74
  number of words remaining = 23
  r 07:41 0.328 93
```

As shown, there are 140 words total within lines 5 through 55 of your compin segment and 97 are neither control lines nor duplicated words. The trim_wordlist command matches 74 of those 97 with words in the standard dictionary. Only 23 words remain.

Type "pwl Lincoln.compin" again to get a listing of the remaining 23 words. Note that your listing does not contain any control lines from your compin file.

```
! pwl Lincoln.compin


  brave            natiom          ;No.;Unit
  conceived        Pennsylvania    ;1;Col.;Gen.;Capt
  dead             resting         ;2;Capt.;;Lt
  fathers          sevne           ;3;Sgt.;Col
  Gettysburg       struggled       ;4;Corp
  ground           war             ;5;;Lt.;Sgt
  lives            1;Unit          ;6;;;Corp
  men              2;Unit


  r 07:41 0.110 11
```

LOCATE WORDS

The **locate_words** command, whose short name is "lw", prints the word and the absolute line number of the line in which the word appears in your file. The Multics locate_words command is invoked as "lw pathname word1 word2 wordN". In this example, there are two known misspelled words, "sevne" and "natiom". Additionally, include "men" in the command. Thus, type the locate_words command line as "lw Lincoln.compin sevne natiom men". The response shows that "men" appears in two different lines, while "sevne" and "natiom" each appears in only one.

```
! lw Lincoln.compin sevne natiom men
  men                         18   54
  natiom                      17
  sevne                       12
  r 07:42 0.180 22
```

Often it is helpful to see how a word may be used within a line or in what context the word may be used. A control argument added to the locate_words command will give a print of the line in which the particular word appears. This control argument is given as "-long" or "-lg". Type the command line "lw Lincoln.compin sevne natiom men -lg".

```
! lw Lincoln.compin sevne natiom men -lg


  men

       18    the preposition that all men are created equal.  Now
       54    The brave men, living and dead, who struggled here


  natiom

       17    a new natiom, conceived in liberty and dedicated to


  sevne

       12    Fourscore and sevne years ago
  r 07:43 0.184 27
```

Another control argument can be given to show the lines before and after the line in which the questioned word exists. This control argument is given as "–lines N", where "N" is the number of lines on either side of the line in question. If "N" is 2, for example, five lines will be printed: the two immediately preceding the line of the questioned expression, the line having the questioned expression, and the two lines immediately following. If "N" is 1, three lines will be printed. Invoke the "lw" command, "lw Lincoln.compin sevne natiom –lines 1".

```
    ! lw Lincoln.compin sevne natiom -lines 1


      natiom

          16      our fathers brought forth on this continent
          17  *   a new natiom, conceived in liberty and dedicated to
          18      the preposition that all men are created equal.  Now


      sevne

          11      .unl -5
          12  *   Fourscore and sevne years ago
          13      .bbf
      r 07:44 0.200 33
```

REVISE WORDS

The **revise_words** command (with the short name "rw") lets you change (revise) words in your file without going into the text editor. It is invoked as "rw pathname word1 rev1 word2 rev2 wordN revN". Because "sevne" and "natiom" should be changed (as well as "preposition"), type "rw Lincoln.compin sevne seven natiom nation preposition proposition" with the corrected word following as a pair with the misspelled or misused word. Note that there is one revision for each word. With this revise_words command, the Multics system will change every occurrence of the word within the compin segment.

```
! rw Lincoln.compin sevne seven natiom nation preposition proposition
  1 revision for natiom
  1 revision for preposition
  1 revision for sevne
  r 07:44 0.196 39
```

You may at times need to separate one word into two words. For example, suppose you had "thisis" which should be "this is". Because the revise_words command works with pairs of words, if you were to type "rw pathname thisis this is sevne seven", you will have "thisis" replaced by "this" and all occurrences of "is" replaced by "sevne". However, if you invoke the command with quotation marks around "this is" as in "rw pathname thisis "this is" sevne seven" then the Multics system will make the change you want.

In addition, the revise_words command will change "men" to "men/women" without affecting words such as "mention". In your editor, if you were to request "1,$s/men/men\c/women/", you might have "men/womention".

As a further exercise, again have the Multics system locate these three words you just revised and observe that they are not on any line in the file. You can further verify this revision by invoking your editor and giving a locate request for these words in your "Lincoln.compin" file. Note that the search fails in all three instances. Your compin file has been corrected.

```
! lw Lincoln.compin sevne natiom preposition
  natiom                 NONE
  preposition            NONE
  sevne                  NONE
  r 07:45 0.175 9


! qx                                             ! ted
! r Lincoln.compin                               ! r Lincoln.compin
! /sevne/                                        ! /sevne/
  Search failed.                                   Line search failed. "/sevne/"
! /natiom/                                       ! /natiom/
  Search failed.                                   Line search failed. "/natiom/"
! /preposition/                                  ! /preposition/
  Search failed.                                   Line search failed. "/preposition/"
! q                                              ! q
  r 07:46 0.178 11                                 r 07:46 0.178 11
```

ADD DICTIONARY WORDS

In many cases, the "cwl", "twl", and "pwl" commands are the only dictionary commands you will need. The **add_dict_words** command, with the short name "adw", is the command used to create your own dictionary and the command you will use to add more words later to your dictionary. This command is invoked as "adw pathname word1 word2 wordN" where "pathname" is the name that you assign to your dictionary, and "word1 word2" are those words or expressions you want in this dictionary.

When you create a dictionary or add words to a dictionary, it is best to include the hyphenation of the words at that time. However, there are some words that you may want to add but never want hyphenated (such as "ar-my" or "na-vy"). For other words, you may not want to include all possible hyphenations (some words ending in "ed" or "ing", for example).

The first step toward creating your dictionary is to select its pathname. For the examples here, use "mywords. dict". The pathname must have the suffix "dict", although for most commands, the Multics system will assume it is present. Thus, to create your own dictionary, select a few words from the printed word list of your "Lincoln.compin" file. Type "adw mywords.dict brave con-ceived dead fa-thers ground lives men". You will receive a message that a dictionary is being created for you:

```
! adw mywords.dict brave con-ceived dead fa-thers ground lives men
  add_dict_words: Creating >udd>ProjA>TSmith>mywords.dict
  r 07:47 0.828 126
```

There are certain conventions to follow if the words that are to be added are spelled with hyphens such as the words "in-house", "x-ray", "day-to-day", and "all-American". If the word can be broken at the hyphen, then two hyphens are added when the word is included in your dictionary. For example, the words "in–house" and "day–to–day" would be added to your dictionary as "in––house" and "day––to––day". The word "all–American" would be added to the dictionary by typing "all––Amer–i–can".

If the word is not to be broken at the hyphen, then the hyphen is followed by an equal sign when the word is added to your dictionary. Thus, "x–ray" would be typed as "x–=ray".

When the compose command was used to format your "Lincoln.compin" file using hyphenations, it hyphenated the word "battlefield" on page 2. Assume that this word should be spelled "battle–field" and cannot be broken at the hyphen. That is, to your dictionary now type "adw mywords bat–tle–=field". Also, so that you can verify this hyphenation later, revise your "Lincoln.compin" segment by typing "rw Lincoln.compin battlefield battle–field". (Alternatively, you could invoke your editor, read your file, request the substitution, and overwrite the file.)

```
! adw mywords bat-tle-=field
  r 07:58 0.442 51

! rw Lincoln.compin battlefield battle-field
  1 revision for battlefield
  r 07:58 0.374 35
```

Often, your documents will contain words that you will not want trimmed because of their usage, company policy regarding their usage, or because they may be confused with other terms. As an example, you may not want to use the word "current" meaning "today" in documents where this word is applied in relationship to electricity or water. Thus, you may want to verify the usage of this word and will not want it trimmed when the trim_wordlist command is given. To add words of this sort to your dictionary, they would be typed preceded by a circumflex as in "^current".

In the case of your dictionary example, assume a word that is not to be trimmed is "fourscore". Therefore, add this word to your dictionary.

```
! adw mywords ^four-score
r 07:59 0.148 23
```

COUNT DICTIONARY WORDS

You may have, on occasion, a desire to know the number of words you have in your dictionary. A **count_dict_words** command whose short name is "cdw" will tell you. This command is invoked by "cdw pathname" where "pathname" (either with or without the "dict" suffix' is the name of your dictionary. Thus, type "cdw mywords" and the Multics system will respond with the message that you, at this time, have nine words in your "mywords.dict" file.

```
! cdw mywords
number of dictionary words = 9
r 07:59 0.093 7
```

Alternatively, a control argument to the same effect can be given with the add_dict_words command at the same time you add a word or words. Thus, add the word "^con-ti-nent" to your dictionary and use the control argument "-count" or "-ct" for a count of the words in that file.

```
! adw mywords ^con-ti-nent -ct
number of words added = 1
number of dictionary words = 10
r 07:5 0.205 51
```

LIST DICTIONARY WORDS

The **list_dict_words** command whose short name is "ldw" is used either to determine if a word exists in your dictionary or to give a list of all the words in your dictionary. The "ldw" command is typed in as "ldw pathname word1 word2 wordN". For example, type "ldw mywords seven forth continent" and note that you do not have the words "seven" or "forth" in your dictionary but you do have the word "continent".

```
! ldw mywords seven forth continent
  list-dict-words: Word not in dictionary.  seven
  list-dict-words: Word not in dictionary.  forth
  ^con-ti-nent
  r 08:00 0.329 57
```

If you desire a complete list of all the words in your dictionary, do not include "word1 word2" etc. in the command line. Thus, type "ldw mywords" and receive a list of the words presently in your dictionary.

```
! ldw mywords
  bat-tle-=field
  brave
  con-ceived
  ^con-ti-nent
  dead
  fa-thers
  ^four-score
  ground
  lives
  men
  r 08:00 0.186 29
```

DELETE DICTIONARY WORDS

The command to delete words from your dictionary is **delete_dict_words** with a short name of "ddw". The command is given as "ddw pathname word1 word2". Therefore to delete the words "dead" and "men" from your dictionary, type "ddw mywords dead men". Repeat the list_dict_words command to verify that these two words have been deleted from your dictionary.

```
! ddw mywords dead men
  r 08:00 0.151 47

! ldw mywords
  bat-tle-=field
  brave
  con-ceived
  ^con-ti-nent
  fa-thers
  ^four-score
  ground
  lives
  r 08:01 0.169 5
```

Words to be deleted from your dictionary must be spelled without hyphens unless the word is hyphenated such as "x-ray". To delete a word with the no-trim attribute, do not type the circumflex in the delete command line.

NOTE

Because your dictionary is a file in permanent storage, the entire dictionary can be deleted by using the delete command; that is, "dl pathname" (the "dict" suffix must be included). If you type the list command "ls", you will find your "mywords.dict" file listed.

ADD SEARCH PATH

You have so far seen how the compose command uses the standard dictionary for hyphenation and how this same dictionary is used for trimming word lists to confirm spelling and consistent word usage. Now you have created your own dictionary that can be used with this standard dictionary.

In order to use your dictionary, a dictionary search path must be assigned. By default, this search path contains only the standard dictionary. To add your dictionary to the search path, the **add_search_path** command, whose short name is "asp", must be invoked as "asp dict pathname". Thus, if you now type "asp dict mywords.dict", your dictionary will be added to the search path. However, this command places your dictionary in the search path *after* the standard dictionary.

NOTE

There are several different search paths within the Multics system. The term "dict" in this command line refers to manipulating the dictionary search paths only.

Once a word is found in the search path, the search is discontinued. Thus, for example, you have in your dictionary a no-trim word "fourscore". But, this word also exists in the standard dicitonary which is searched first in the search path. The word "fourscore" will therefore be trimmed if the trim_wordlist command is given because it will be found in the standard dictionary and discarded. The search will stop before your dictionary is reached in the search path.

There is a control argument that can be applied to the add_search_paths command to place your dictionary in the search path of the standard dictionary. That control argument is **-first**. Thus, now type "asp dict mywords.dict-first" and your dictionary will be placed ahead of the standard dictionary in the search path. In the case of the no-trim word "fourscore", the word will be found in your dictionary first and the search discontinued before it would be found in the standard dictionary.

```
! asp dict mywords.dict -first
r 08:01 0.236 27
```

If you have more than one dictionary, they will be added to the dictionary search paths in the order you specify. That is, the "-first" control argument puts that dictionary in front of any others.

FIND DICTIONARY WORDS

The **find_dict_words** command, with the short name "fdw", tells you whether a given word exists in any of the dictionaries in the dictionary search paths. This command is invoked as "fdw word1 word2 wordN". A control argument can be added to this command to tell in which dictionary the word exists, if it exists. This control argument is **-dictionary,** with the short name "-dict".

As an example, type "fdw battlefield -dict" and note that this word appears in the standard dictionary. Then type "fdw battle-field -dict" and note that this word is one in your "mywords" dictionary.

```
! fdw battlefield -dict
  bat-tle-field      >udd>standard.dict
  r 08:01 0.308 45

! fdw battle-field -dict
  bat-tle-=field      >udd>ProjA>TSmith>mywords.dict
  r 08:02 0.061 14
```

If the word does not exist in either of the dictionaries in the search paths, you will receive a response saying that the word was not found. For example, type "fdw struggled -dict".

```
! fdw struggled -dict
  find_dict_words: Word not found.  struggled
  r 08:03 0.082 41
```

USING YOUR DICTIONARIES

By default, the dictionary used in the dictionary search paths is the standard dictionary. To include your dictionaries in the search paths at any one sitting at your terminal, you must define the search path by the add_search_path command. That is, if you have finished your work at your terminal and logged out, you must again define the dictionary search path you are going to use when you log back in.

How the compose command reacts to the dictionaries is best shown by an example from your "Lincoln.compin" file where you have altered the spelling of "battlefield" to "battle–field". Your "mywords" dictionary has this word spelled with the hyphen but with the no-break-at-hyphen attribute.

Refer to page 2 of your last formatted output of your "Lincoln.compin" file and note that the compose command hyphenated "battlefield" between the "e" and "f". Now again give the compose command to format page 2 of this file with hyphenations. (If you have just logged in, be sure to type "asp dict mywords.dict –first" before you invoke the compose command.)

In this latest formatted output, note that the word "battle–field" has been hyphenated as you have it in your dictionary, but not at the hyphen between the "e" and "f" as you also stipulated in your dictionary by including the equal sign ("bat–tle–=field").

```
! comp Lincoln -hyph -ind 15 -pgs 2


                    Speech Continued


                        No.  Unit 1    Unit 2    Unit 3

                        1    Col.      Gen.      Capt.
                        2    Capt.               Lt.
                        3    Sgt.      Col.
                        4    Corp.
                        5              Lt.       Sgt.
                        6                        Corp.

                    testing whether that nation, or any
                    nation   so    conceived   and   so
                    dedicated, can long endure.

                         We  are met   on a  great bat-
                    tle-field  of  that   war.  We have
                    come to dedicate  a portion of that
                    field as a final resting place(1)
                    -----------------------------------

                    (1) Gettysburg, Pennsylvania.


                                  Page 2


    r 08:18 3.065 240
```

Next use the create_wordlist command used earlier for lines 5 through 55 and with the -no_control_lines control argument. Then again invoke the command, trim_wordlist. Finally, give the print_wordlist command to see the words that have not been trimmed because they do not appear in either your dictionary or the standard dictionary or because your dictionary has the word with the no-trim attribute.

```
! cwl Lincoln.compin -ncl -fm 5 -to 55
  total number of words = 140
  number of unique words = 96
  r 08:21 0.365 36

! twl Lincoln.compin
  number of words trimmed = 78
  number of words remaining = 18
  r 08:21 0.539 111

! pwl Lincoln.compin




  continent          resting          ;1;Col.;Gen.;Capt
  dead               struggled        ;2;Capt.;;Lt
  Fourscore          war              ;3;Sgt.;Col
  Gettysburg         1;Unit           ;4;Corp
  men                2;Unit           ;5;;Lt.;Sgt
  Pennsylvania       ;No.;Unit        ;6;;;Corp


  r  08:22 0.095 7

! logout
```

Observe that this list does not contain any of the words of your dictionary (battle-field, brave, conceived, fathers, ground, or lives) except those that you gave no-trim attributes (fourscore and continent). The words remaining in this final list, except the two no-trim words, do not appear in either your dictionary or the standard dictionary.

Section 11
Speedtype

Speedtype is a form of shorthand you can use when you input any file in the text editor. It has several advantages which include typing your input more quickly and improving your typing accuracy, because a few symbols (characters) may be typed to represent words commonly misspelled. Speedtype uses symbols of one to seven characters to represent regular expressions of up to 56 characters. For example, you may use "0" to represent "of" or "vpus" to represent "Vice President of the United States". Speedtype will expand the shorthand input into the longer regular expressions upon command.

In order to use Speedtype, you must first develop a symbols dictionary that lists your symbol with its expansion. The input is then typed using those symbols listed in your symbols dictionary. After your input has been completed, the command to expand the symbols will result in the replacement of all the symbols with their corresponding regular expressions.

NOTE

A symbol is an abbreviation for a regular expression, but the term "abbreviation" is not used here in order to avoid confusion with another Multics operation.

The first example to be used in this section is extracted from the Constitution of the United States and reads as follows:

"The Senate of the United States shall be composed of two Senators from each State, chosen by the Legislature thereof for six Years; and each Senator shall have one Vote.

The Vice President of the United States shall be President of the Senate, but shall have no Vote, unless they be equally divided."

USE SYMBOLS

The **use_symbols** command, with the short name "usb", sets the current symbol dictionary, and all Speedtype commands then use this dictionary. If this command is given and the dictionary does not exist, you will be asked by the Multics system if it should be created.

To invoke the use_symbols command, you need to type "usb pathname", where "pathname"is the name you supply to the dictionary. This pathname must have a suffix of "symbols". For example, you might call your symbols dictionary "my.symbols". Thus, type "usb my.symbols", and to the question asked by the Multics system, reply "yes" You will now have a symbols dictionary called "my.symbols", although at this point it is empty.

```
  r 15:51 1.209 98

! usb my.symbols

  Speedtype: >user_dir_dir>ProjA>TSmith>my.symbols not found.
! Do you want to create it?  yes
  r 15:52 0.103 27
```

ADD SYMBOLS

The **add_symbols** command, with the short name "asb", is used to place your Speedtype symbols with their expansions into your "my.symbols" dictionary. In general, none of the characters used for a symbol should be punctuation marks or other special characters recognized either by the text editor or by the compose command. Likewise, capital letters are not allowed as the first character of a symbol. Also, it is best not to use common expressions for symbols. For example, it may cause confusion when editing or composing if you use "to" or "is" as symbols.

The add_symbols command is invoked as follows: "asb symbol expansion". Type "asb e the" and "e" will be the Speedtype symbol in your symbols dictionary for the word "the". You now have one symbol with its expansion added to your "my.symbols" dictionary.

```
! asb e the
  r 1552 0.152 17
```

Note that only one symbol and its expansion are allowed on any one add_symbols command line. Note too that if a symbol is to be expanded into a phrase that contains spaces or tabs, the expansion must be typed within quotation marks, as in the example, "asb vpus "Vice President of the United States"". Other symbols with their expansions that you should add to your "my.symbols" dictionary at this point are as follows:

se	senate	6	six
o	of	ye	year
un	unit	a	and
sta	state	h	have
sha	shall	1	one
b	be	vo	vote
com	compose	vpus	"Vice President of the United States"
2	two		
sen	senator	pres	president
fm	from	bt	but
ea	each	n	no
cho	chosen	unl	unless
leg	legislature	ty	they
tr	there	eq	equal
f	for	div	divide

NOTE

This list, in violation of the use of common expressions for symbols, nevertheless uses "a" for "and" as will be described later.

```
! asb se senate
  r 15:54 0.027 3

! asb o of
  r 15:54 0.030 0

! asb un unit
  r 15:54 0.030 0

! asb sta state
  r 15:55 0.029 0

! asb sha shall
  r 15:55 0.030 0

! asb b be
  r 15:55 0.032 0

! asb com compose
  r 15:55 0.029 0

! asb 2 two
  r 15:55 0.032 0

! asb sen senator
  r 15:55 0.041 0

! asb fm from
  r 15:55 0.031 0

! asb ea each
  r 15:55 0.033 0

! asb cho chosen
  r 15:56 0.037 1

! asb leg legislature
  r 15:56 0.032 3

! asb tr there
  r 15:56 0.031 0

! asb f for
  r 15:56 0.044 0

! asb 6 six
  r 15:56 0.034 0

! asb ye year
  r 15:56 0.032 0

! asb a and
  r 15:56 0.028 0

! asb h have
  r 15:56 0.030 0

! asb 1 one
  r 15:56 0.031 0
```

```
! asb vo vote
 r 15:57 0.029 0

! asb vpus "Vice President of the United States"
 r 15:57 0.049 21

! asb pres president
 r 15:57 0.032 4

! asb bt but
 r 15:58 0.035 0

! asb n no
 r 15:58 0.027 0

! asb unl unless
 r 15:58 0.029 0

! asb ty they
 r 15:58 0.028 0

! asb eq equal
 r 15:58 0.029 0

! asb div divide
 r 15:58 0.028 0
```

LIST SYMBOLS

The **list_symbols** command, with the short name "lsb", gives the complete name for your symbols dictionary, the total number of symbols in your dictionary, and a listing of each symbol with its expansion. Type "lsb" for a list of all the symbols in your "my.symbols" dictionary.

```
! lsb
  Speedtype symbol dictionary: >user_dir_dir>ProjA>TSmith>my.symbols
  Total:  30 symbols
  1         one
  2         two
  6         six
  a         and
  b         be
  bt        but
  cho       chosen
  com       compose
  div       divide
  e         the
  ea        each
  eq        equal
  f         for
  fm        from
  h         have
  leg       legislature
  n         no
  o         of
  pres      president
  se        senate
  sen       senator
  sha       shall
  sta       state
  tr        there
  ty        they
  un        unit
  unl       unless
  vo        vote
  vpus      Vice President of the United States
  ye        year
  r 15:58 0.399 13
```

As indicated by this print of the list of symbols in your symbols dictionary, the number of symbols can become quite lengthy in a very short time and you may not always want a complete listing in order to determine whether a particular symbol has been added to your dictionary. Instead of a complete list of symbols, you can use the list_symbols command for individual symbols by adding those symbols to the command line. For example, type "lsb sta e fw" to see the expansions for these symbols. Note that symbol "fw" has not been defined; it is not in your list of symbols.

```
! lsb sta e fw
  sta       state
  e         the
  Speedtype: Symbol "fw" not defined
  r 16:00 0.045 24
```

FIND SYMBOLS

The list_symbols command uses one or more of your symbols in its command line and prints the expansion. However, it is often easy to forget the symbol used for an expansion. The **find_symbols** command, whose short name is "fsb", accepts the expansion in the command line and provides you with the symbol. Type "fsb state follow" and observe that the symbol for "state" is "sta", but again, "follow" has not been defined in your symbols dictionary.

```
! fsb state follow
  state  -  sta
  Speedtype: Expansion "follow" not defined
  r 16:00 0.051 6
```

SUFFIXES

When a symbol is entered into a symbols dictionary, Speedtype automatically provides suffixes of "s", "ed", "ing", "er", and "ly" so that each symbol can be expanded into five words besides the original given. These suffixes can be called out when the symbol is typed in your input text segment by adding "+", "−", "*", "=", or "|" respectively to your symbol. These suffixes are added to the expanded word in a manner following usual English rules (American usage) for adding suffixes whether the expanded word with the suffix is legitimate or not.

For example, for your Speedtype symbol "sta", whose expansion is "state", you can call for "states", "stated", "stating", "stater", or "statly" (not "stately") by adding the appropriate default character to the symbol. That is, these defined suffix expansions can be called by typing "sta+", "sta–", "sta*", "sta=", or "sta|", respectively. (The expansion of "state" to "stately" requires a change_symbols command to be described.)

To see how the symbol may be expanded with these suffixes, add the **−long** control argument (or "–lg") to the list_symbols command line with the symbol (or symbols) involved. Type "lsb sta –lg" for all the defined suffix expansions for "sta".

```
! lsb sta -lg
  sta          state
          (+)  states
          (-)  stated
          (*)  stating
          (=)  stater
          (|)  statly

  r 16:00 0.062 1
```

Next, because generalized rules for adding suffixes to English words do not always apply, use the "–long" control argument with the find_symbols command and note how Speedtype expands the words "the", "year", and "unit" which exist in your symbols dictionary. Observe that most of the expansions are not legitimate spellings in the English language.

```
! fsb the year unit -lg
 the  -  e
            (+) thes
            (-) thed
            (*) thing
            (=) ther
            (|) thly

 year  -  ye
            (+) years
            (-) yeared
            (*) yearing
            (=) yearer
            (|) yearly

 unit  -  un
            (+) units
            (-) unitted
            (*) unitting
            (=) unitter
            (|) unitly

 r 16:01 0.181 0
```

CHANGE SYMBOLS

The word "unit" and the symbol "un" were added to your "my.symbols" dictionary (rather than the word "unite", as you shall see later) in order to show how an expansion can be corrected or deleted. Consider the expansions "unitted", "unitting", and "unitter" above, which would be acceptable words if they were spelled "united", "uniting", and "uniter". The word "unitly" above has no meaning and can be deleted from the expansion.

The **change_symbols** command, with the short name "csb", is used to correct those expansions that result when generalized English rules for adding suffixes do not apply. This same command is used to delete those expansions you do not want.

In the command line, the symbol is first given after the command name; then the suffix is given and the suffix is followed by the corrected spelling. Thus, to make corrections to "unitted" and "unitting", type "csb un -ed united -ing uniting".

```
! csb un -ed united -ing uniting
  r 16:01 0.043 8
```

To delete expansions such as "unitter" and "unitly", type "csb un -er off -ly off".

```
! csb un -er off -ly off
  r 16:01 0.033 2
```

Now type "lsb un -lg" for a new list of defined suffix expansions for the symbol "un".

```
! lsb un -lg
  un          unit
          (+) units
          (-) united
          (*) uniting

  r 16:02 0.072 7
```

Symbols themselves can be changed by using a **-force** control argument (or "-fc") with the add_symbols command. With this control argument, the expansion for a symbol can be replaced. For example, to replace "unit" with "unite" for the symbol "un", type "asb un unite -fc". Your symbols dictionary now has "un" as the symbol for "unite" rather than as the symbol for "unit".

```
! asb un unite -fc
  r 16:02 0.026 4
```

In your symbols dictionary, you have the symbol "sta" for "state". The "–force" control argument cannot be used to change the symbol. If you were to type "asb st state –fc", you would not force "st" to be the symbol for "state". Instead, you would simply add a new symbol to your dictionary so that either "sta" or "st" could be used for "state". Type "asb st state –fc". Then type "asb st street" and observe that you are asked whether you want your dictionary changed. Type "no" to the question. If you type "yes", the replacement will be made and "st" would then become the symbol for "street" instead of for "state".

```
! asb st state -fc
  r 16:02 0.033 0

! asb st street

  Speedtype: st already defined.
! Do you want to replace it?   no
  r 16:02 0.048 3
```

DELETE SYMBOLS

At this point, you have in your "my.symbols" dictionary two symbols with identical expansions, "sta" and "st" for "state". To avoid confusion, one of these symbols should be removed. The **delete_symbols** command, with the short name "dsb", will delete an entry from your dictionary. Type "dsb sta" to remove the symbol "sta" and its expansion "state" from your "my.symbols" dictionary. Verify your symbols listing now by typing "lsb un st sta" and "fsb unit unite state".

```
! dsb sta
  r 16:02 0.027 1

! lsb un st sta
  un        unite
  st        state
  Speedtype: Symbol "sta" not defined
  r 16:02 0.039 0

! fsb unit unite state
  Speedtype: Expansion "unit" not defined
  unite  -  un
  state  -  st
  r 16:03 0.048 0
```

TRANSPARENT ESCAPE

Now that your symbols dictionary has been developed, you are ready to use it with input to a file segment. However, there is one more expansion process to describe first. That process involves a **transparent escape** which consists of a colon (:). The transparent escape is used to combine symbols for an expansion to one expression when otherwise the symbols would be processed separately. For example, your symbols dictionary contains "tr" for "there" and "o" for "of". If your input is typed "tr:o", Speedtype will expand this string into "thereof".

USE OF SPEEDTYPE — EXAMPLE 1

To show how Speedtype can be used and how symbols are expanded, invoke your editor and input the following segment: Note that to capitalize a word in its expansion, the first letter of the symbol must be capitalized. Write your segment as "file1.compin".

```
! qedx
! a
! .pdl 35
! .pdw 35
! .spb
! E Se o e Un- St+ sha be com-
! o 2 Sen+ fm ea st, cho by e
! Leg tr:o f 6 Ye+; a ea Sen sha
! h 1 Vo.
! .spb
! E vpus sha
! b Pres o e Se, bt
! sha h n Vo, unl ty b eq| div-
! .spb
! \f
! w file1.compin
! q
  r 16:06 0.339 36
```

```
! ted
! a
! .pdl 35
! .pdw 35
! .spb
! E Se o e Un- St+ sha be com-
! o 2 Sen+ fm ea st, cho by e
! Leg tr:o f 6 Ye+; a ea Sen sha
! h 1 Vo.
! .spb
! E vpus sha
! b Pres o e Se, bt
! sha h n Vo, unl ty b eq| div-
! .spb
! \f
! w file1.compin
! q
  r 16:06 0.339 36
```

When you give the command to expand the symbols, all symbols of the file will be expanded and the expanded file will overwrite the original. Thus, before this file is expanded, again invoke your editor, read "file 1.compin" and then write an identical "file2.compin" segment. This step permits "file1.compin" to remain as a reference.

```
! qx                                      ! ted
! r file1.compin                          ! r file1.compin
! w file2.compin                          ! w file2.compin
! q                                       ! q
  r 16:06 0.144 5                           r 16:06 0.144 5
```

EXPAND SYMBOLS

The **expand_symbols** command (short name, "esb") expands all the symbols in your file when you type "esb pathname", where pathname is the name of your file. If you have quit working at your terminal and logged out, remember that to use your symbols dictionary when you log back in, you must first give the use_symbols command.

Expand your "file2.compin" segment by typing "esb file2.compin".

```
! usb my.symbols
  r 16:07 0 0.030 4

! esb file2.compin
  r 16:08 0.151 8
```

To see how your "file2.compin" segment has been expanded, invoke your editor again, read this file, and then request a print of the file. You can then use the compose command to format the file (using your dictionary if you so desire for hyphenation) to see that formatting is performed as described in earlier sections.

```
! qx                                        ! ted
! r file2.compin
! 1,$p
  .pdl 35
  .pdw 35
  .spb
  The Senate of the United States shall be composed
  of two Senators from each state, chosen by the
  Legislature thereof for six Years; and each Senator shall
  have one Vote.
  .spb
  The Vice President of the United States shall
  be President of the Senate, but
  shall have no Vote, unless they be equally divided.
  .spb
! q
  r 16:06 0.116 4

! comp file2




  The  Senate  of  the  United States
  shall  be composed of  two Senators
  from  each  state,  chosen  by  the
  Legislature thereof  for six Years;
  and  each  Senator  shall  have one
  Vote.

  The  Vice President  of the  United
  States  shall be  President of  the
  Senate,  but  shall  have  no Vote,
  unless they be equally divided.








  r 16:06 1.492 149
```

TEMPORARY ESCAPE

Symbols chosen for Speedtype generally should not be commonly used English words. For example, in your list of symbols, "a" is deliberately given as the symbol for "and" when a better symbol would be "x". There are times when you will not want a symbol expanded and, instead, want the symbol to be used literally in your text segment, such as the letter "a" as used here.

To keep a string of characters used as a symbol from being expanded, the **temp escape** character "~" must be typed immediately in front of the string. Thus, for an expansion to be "A cat and a dog", for example, where "a" is the Speedtype symbol for "and", your input would be typed "~A cat a ~a dog".

CAPITALIZATION

As shown by the example, when the first letter of a symbol is capitalized in your input segment, the first letter of the expansion is capitalized. Speedtype will also recognize an upper prefix (|) which will capitalize all letters of the expanded symbol. Thus, to expand "se" to "SENATE", your input would be typed as "|se".

UNDERLINING

Speedtype will underline any expanded symbol if you prefix the symbol with the under prefix (_). Thus, if "se"is to be expanded as "<u>Senate</u>", your input segment would be typed as "_Se". The under and upper prefixes can be used together as either "|_" or "_|".

USE OF SPEEDTYPE — EXAMPLE 2

For a second example using Speedtype, first add the following symbols to your "my.symbols" dictionary:

per	person	wn	when
wo	who	cous	"Citizen of the United States"
kt	not	ele	elect
att	attain	inh	inhabitant
30	thirty	tt	that
bn	been	wh	which
9	nine		

```
! asb per person
  r 16.09 0.035 7

! asb wo who
  r 16:09 0.34 0

! asb kt not
  r 16.10 0.31 0

! asb att attain
  r 16:10 0.030 0

! asb 30 thirty
  r 16:10 0.032 0

! asb bn been
  r 16:10 0.030 0

! asb 9 nine
  r 16:10 0.033 0

! asb wn when
  r 16:10 0.032 0

! asb cous "Citizen of the United States"
  r 16:10 0.040 1

! asb ele elect
  r 16:10 0.033 0

! asb inh inhabitant
  r 16:11 0.030 0

! asb tt that
  r 16:11 0.034 0

! asb wh which
  r 16:11 0.032 0
```

Now invoke your text editor and create a file as follows using the temp escape and the under and upper prefixes. Expand the file and note the difference between "a" and "and" in the requested print. Also note the underscoring and capitalization.

```
! qx                                        ! ted
! a
! N _Per sha b ~a  |sen wo
! sha kt h att- to e age o
! 30 Ye+, a b 9 Ye+
! ~a  |cous, a wo sha kt,
! wn _|ele-, b an  |_inh o tt
! st f wh he sha be cho.
! \f
! w file3.compin
! q
  r 16:13 0.243 94

! usb my.symbols
  r 16:14 0.034 16

! esb file3.compin
  r 16:14 0.108 20

! qx                                        ! ted
! r file3.compin
! 1,$p
  No Person shall be a SENATOR who
  shall not have attained to the age of
  thirty Years, and be nine Years
  a CITIZEN OF THE UNITED STATES, and who shall not,
  when ELECTED, be an INHABITANT of that
  state for which he shall be chosen.
! q
  r 16:15 0.122 28
```

RETAIN SYMBOLS

As you become more proficient and familiar with Speedtype, you will tend to add more symbols to your symbols dictionary. Also, you may on occasion have someone else using a different symbols dictionary correct or alter a file you have already expanded. It is possible that if the expand_symbols command is again given, certain words not expanded previously (because they were not in your symbols dictionary) will now be expanded.

For example, add to your symbols dictionary the symbol "an" for "another", expand your "file3.compin" segment, invoke the editor, read the file, and request a print of the file.

```
! asb an another
  r 16:25 0.159 23

! esb file3.compin
  r 16:26 0.095 12

! qx                                        ! ted
! r file3.compin
! 1,$p
  No Person shall be a SENATOR who
  shall not have attained to the age of
  thirty Years, and be nine Years
  a CITIZEN OF THE UNITED STATES, and who shall not,
  when ELECTED, be another INHABITANT of that
  state for which he shall be chosen.
! q
  r 16:26 0.103 1
```

Observe that although the file had previously been expanded, the word "an" in the fifth line has been recognized as a new symbol to now be expanded, thereby producing incorrect phrasing. This result could have been avoided if the **retain_symbols** command, with the short name "rsb", had been invoked immediately after the first expansion.

Invoke your editor again, read "file3.compin", and correct the line by substituting "an" for "another". Request "w" and quit your editor. Now give the retain_symbols command to preserve your file and keep it from being altered by a second expand_symbols command. Give the expand_symbols command and then invoke the editor again to see a print of your file. Note that "an" does not get expanded after the retain_symbols command.

```
! qx                                  ! ted
! r file3.compin
! /another/ s//an/
! w
! q
  r 16:26 0.083 7

! rsb file3.compin
  r 16:26 0.031 2

! esb file3.compin
  r 16:27 0.109 6

! qx                                  ! ted
! r file3.compin
! 1,$p
  No Person shall be a SENATOR who
  shall not have attained to the age of
  thirty Years, and be nine Years
  a CITIZEN OF THE UNITED STATES, and who shall not,
  when ELECTED, be an INHABITANT of that
  state for which he shall be chosen.
! q
  r 16:27 0.086 11
```

OPTION SYMBOLS

In the last Speedtype example, the vertical bar was used as an upper prefix to capitalize all letters of an expanded symbol. Consider the possible result if Speedtype is used in a compin file where vertical bars are further used as delimiters for headers, footers, titles, etc. In the expansion of that file, Speedtype would not only capitalize expressions following the vertical bars but also would remove several of those vertical bars used as delimiters. Thus the file would require extensive editing before the compose command could format it as you had envisioned.

The **option_symbols** command, with the short name "osb", permits you to change any Speedtype symbol, such as the vertical bar, to a different symbol. In fact, the upper prefix (|), under prefix (_), temp (~), trans (:), and the suffixes (+, −, *, =, and |) are used by default and can all be changed as you desire by the option_symbols command.

The option_symbols command is given with a control argument showing which symbol is to be changed. Where "X" is the new symbol, the control arguments are: "-upper X", "-under X", "-temp X", "-trans X", "-plural X", "-ed X", "-ing X", "-er X", and "-ly X". For example, to use a plus sign instead of the vertical bar as the symbol for the upper prefix, the command is given as "osb −upper +". Type this command.

```
  r 16:27 0.086 11

! osb -upper +
  r 16:34 0.028 0
```

Invoke the editor, read your "file1.compin" segment (previously kept as reference), and substitute "+S" throughout for "S". That is, each expression in the file beginning with an uppercase letter "S" is to be capitalized in the expansion. Overwrite this substitution and quit your editor.

```
! qx
! r filel.compin
! 1,$p
  .pdl 35
  .pdw 35
  .spb
  E Se o e Un- St+ sha b com-
  0 2 Sen+ fm ea st, cho by e
  Leg tr:o f 6 Ye+; a ea Sen sha
  h l Vo.
  .spb
  E vpus sha
  b Pres o e Se, bt
  sha h n Vo, unl ty b eq| div-.
  .spb
! 1,$s/S/+S/
! w
! q
  r 16:37 0.180 28
```

```
! ted
! r filel.compin
! 1,$p
  .pdl 35
  .pdw 35
  .spb
  E Se o e Un- St+ sha b com-
  o 2 Sen+ fm ea st, cho by e
  Leg tr:o f 6 Ye+; a ea Sen sha
  h l Vo.
  .spb
  E vpus sha
  b Pres o e Se, bt
  sha h n Vo, unl ty b eq| div-
  .spb
! 1,$s/S/+S/
! w
! q
  r 16:37 0.180 28
```

Now give the expand_symbols command for this file. Again invoke your editor to read and print the file. Observe that wherever the plus sign was used as a prefix to a symbol, the expansion of the symbol has all letters of the expression capitalized as if you had used the vertical bar instead of the plus sign.

```
  r 16:37 0.180 28

! esb filel.compin
  r 16:37 0.181 5

! qx                                                 ! ted
! r filel.compin
! 1,$p
  .pdl 35
  .pdw 35
  .spb
  The SENATE of the United STATES shall be composed
  of two SENATORS from each state, chosen by the
  Legislature thereof for six Years; and each SENATOR shall
  have 1 Vote.
  .spb
  The Vice President of the United States shall
  be President of the SENATE, but
  shall have no Vote, unless they be equally divided.
  .spb
! q
  r 16:39 0.305 9

! logout
```

Section 12
List Processing

A list consists of a series of names, words, numbers, or groupings of items set forth in an order (sometimes random) developed by the originator of the list. **List processing** involves placing the items (data records) of a list in order, and sorting, maintaining, deleting, adding, or selecting items from the list.

A list is composed of one or more data records which further contain one or more fields. In a dentist's office, for example, the entire file of patients is a list. Each patient has a data record that contains information about the patient. Each item of information in the data record is called a field. Thus, in the **list** of patients, the **data record** for each patient contains **fields** for the patient's name, address, city, state, zip code, telephone number, date of last checkup, etc. Steps involved in list processing include creating an input file of the items to appear in the list and developing a format for the list. List processing by WORDPRO uses three files: a **listin** file, a **lister** file, and a **listform** file.

The listin file is used to input and update a list. Records in the list can be added, deleted, or updated by editing this file with the text editor. The listin file is identified by the suffix "listin", such as in the file "patients.listin".

The lister file is a file containing the list in a form that can be processed. All sorting, trimming, and processing of records is performed in this file. The lister file is identified by the suffix "lister", such as in "patients.lister". However, the lister file is in a form that is not easily read. It consists of characters and binary information useful only to the Multics system.

The listform file is used to define the format of the document to be produced from the information input in the listin file and processed in the lister file. This file is identified by the suffix "listform", as in the file "patients.listform". In list processing, you may have several listform files that use information from one lister file. Alternatively, you may have only one listform file that uses information from several lister files.

LISTIN FILE

A listin file consists of three parts: (1) a header which specifies the record and field delimiters and the field names, (2) the data records which contain some or all of the fields defined in the header, and (3) fields which contain specific items of the data record.

Header

The **header** is the first item of the listin file. It contains five statements that define the input information for list processing. The first statement is an optional **Comment_delimiter** statement, whose short name is "Cd". This statement permits you to insert various comments in your listin file. These comments are disregarded by the system but may be used to describe certain attributes of a record or file name for your reference.

The Comment_delimiter statement specifies the single character or delimiter that is used to set off these comments, which may be placed anywhere that white space is allowed. The character must precede the comment and end the comment. The character to be used as the Comment_delimiter must be chosen from $, %, &, *, =, ?, |, ^, or ~. The Comment_delimiter is typed in as "Cd: c;", where "c" is the character chosen to be the Comment_delimiter. (Note the colon, semicolon, and spacing.)

The second statement is the **Record_delimiter** statement, whose short name is "Rd". This statement specifies the single character or delimiter that is used to separate data records. The character to be used as the Record_delimiter must be chosen from the same list as for the Comment_delimiter but it cannot be the same character. The Record_delimiter statement is optional. By default, the Record_delimiter is the dollar sign ($). It is typed in as "Rd: r;" where "r"is the character chosen to be the Record_delimiter.

The third statement is the **Field_delimiter** statement, also optional like the Record_delimiter statement. The Field_delimiter statement has the short name "Fd". This statement specifies the character used to separate fields within a record. The Field_delimiter character is chosen from the same list of characters given for the Record_delimiter and the Comment_delimiter, but it cannot be the same chosen character. By default, the Field_delimiter character is the equal sign (=). The statement is typed in as "Fd: f;" where "f " is the character chosen to be the Field_delimiter.

The **Field_names** statement, with the short name "Fn", is a required statement that specifies the names of the fields to be used in the listin segment. Field_names can contain only alphabetic, numeric, or underscore characters, but each name can be from 1 to 32 characters long. Field_names are separated from one another by commas. The Field_names statement is typed in as "Fn: fn1,fn2,...,fnk;" where "fn1", "fn2", and "fnk" are individual names of fields. (Again note the colon, semicolon, spacing, and commas.)

The last of the five statements of the header is the **Records** statement which is typed in as "Records:". The Records statement specifies the end of the header and the beginning of the data records information section of the "listin" segment.

Your header will appear in generalized form in the order that follows, where "c" is the character chosen as the Comment_delimiter, "r" is the character chosen as the Field_delimiter, and "fn1', "fn2", etc. are the names for the fields:

```
Cd: c;
Rd: r;
Fd: f;
Fn: fn1,fn2,fn3,fn4;
Records:
```

Remember that if default values "$" and "=" are to be used as the record and field delimiters respectively, neither the Record_delimiter nor the Field_delimiter statements need to be typed.

Data Records

The beginning of a data record is generally denoted by typing the Record_delimiter alone on a line. The Record_delimiter is then followed by a list of fields within that record. The list of fields does not need to contain all those defined in the Field_names statement given in the header. If a field is not listed, it is not acted upon when the record is processed.

Fields

A field within a data record portion of a listin file consists of three parts: the Field_delimiter that separates each field, the Field_name chosen by you, and the value of the field for the record in which the field exists. For example, the Field_delimiter by default is the equal sign. The name of the field might be "lname" (chosen by you to mean "last name"), and the value of the field might be "Smith". This field would be typed in under the Record_delimiter as "=lname Smith". Observe that there is no space between the Field_delimiter and the Field_name. However, the amount of spacing between the Field_name ("lname") and the value of the field ("Smith") is not important. Any extra white space will be trimmed during the processing of the information.

If the value of the field contains either a Field_delimiter or Record_delimiter character, then quotation marks are required around the value, Likewise, if you wish to include white space with the value, quotation marks must be inserted. For example, if the value of the field is "x=y" and you desire two spaces before and after this value, your input would be typed as "=lname " x=y "."

LISTIN FILE EXAMPLE

As an example of a listin file, consider the last and first names of the first four presidents of the United States, their home states, year of birth, year of death, wife's first name, and years in office.

By default, the dollar sign will be used as the Record_delimiter and the equal sign will be used as the Field_delimiter. If the default values were not given, the Record_delimiter statement would need to be input (as, for example, "Record_delimiter: $;" or "Rd: $;") as well as the Field_delimiter statement (as, for example, "Field_delimiter: =;" or "Fd: =;"). Field_names will be "No", "lname", "fname", "state", "born", "died", "wife", and "term".

Invoke your text editor and create your listin file. Enter as the first item a Command_delimiter using the question mark as the character to be used. Assign the pathname "presidents.listin". Your input should be similar to the following where comments have been placed in records 1 and 4.

```
  r 15:16 0.189 13

! qx                                                  ! ted
! a
! Cd: ?;
! Field_names: No,lname,fname,state,born,died,wife,term;
! Records:
! $
! =No 1
! =lname Washington    ?Father of our country?
! =fname George
! =state VA
! =born 1732
! =died 1799
! =wife Martha
! =term 1789-1797
! $
! =No 2
! =lname Adams
! =fname John
! =state MA
! =born 1735
! =died 1826
! =wife Abigail
! =term 1797-1801
! $
! =No 3
! =lname Jefferson
! =fname Thomas
! =state VA
! =born 1743
! =died 1826
! =wife Martha
! =term 1801-1809
! $
! =No 4
! =lname Madison
! =fname James
! =state VA
! =born 1751
! =died 1836
! =wife Dorothea      ?Nicknamed Dolly?
! =term 1809-1817
! \f
! w presidents.listin
! q
  r 15:17 0.254 23
```

LISTER FILE

One command to the Multics system creates a lister file from your listin file. That command is the **create_list** command, which has the short name "cls". It is input as "cls pathname.lister", where "pathname" is the same pathname as that of the listin file. If the suffix "lister" is not typed, the Multics system assumes it is in the command line.

Except for an expand_list command, the creation of a lister file is the only list processing operation performed on the listin file by the Multics system. Each time an editing change, addition, deletion, or insertion is made to a listin file (using your text editor), a new lister file must be created. That is, each time the create_list command is given, the existing copy of the lister file is overwritten by the new listin file. You cannot add to the lister file except through the listin file or by using an append_list command, and you cannot change the lister record except through the listin file or by using a modify_list command. Also, the lister file consists of characters used by the Multics system in processing a list and therefore is in a form understandable only by the other list processing commands (and some professional computer programmers).

One control argument can be used with the create_list command. The **-totals** (or "-tt") control argument causes a print of the number of records in the lister file.

Type "cls presidents -tt" to create your lister file called "presidents.lister" and receive a count of the number of records in the file.

```
! cls presidents -tt
  create_list: 4 records.
  r 15:18 0.139 10
```

LISTFORM FILE

The listform file is the file that defines the format of the document produced from a list. Information from the list may be copied into the document. You may have several listform files to be used with only one list, or you may have only one listform file that is used with several lists.

The listform file is input through the text editor and has the suffix "listform". It contains three sections which define the document: a **before** section, a **record** section, and an **after** section.

Before Section

The before section of a listform file is optional. The beginning of the section is identified by typing "<Begin before:>" and the end of the section is identified by "<end;>".

The before section contains material to be printed before any record information and is useful for including headings and introductory material for the document. It may contain any text that you may desire, and if the document is to be formatted by a compose command, it can contain compose command control lines. The before section is processed before any records and appears only once in the document.

Record Section

The record section is a required section of the listform segment. The beginning of this section is identified by typing "<Begin record:>" and the end of the section is identified by "<end;>".

The record section can contain text and compose command control lines as for the before section. The section may also contain field values copied from the record being processed. The functions of field insertion, sorting, and selection require the presence of this section in the listform file.

After Section

The after section is an optional section that is added to the document after all records are processed. It may contain text and compose command control lines similar to the before section. The beginning of this section is identified by typing "<Begin after:>" while the end of the section is identified by "<end;>".

The three sections of the listform file are therefore input as follows:

```
<Begin before:>
Optional text, compose command control lines,
etc.
<end;>
<Begin record:>
Text, compose command control lines, etc.
Field insertion values
<end;>
<Begin after:>
Optional text, compose command control lines,
etc.
<end;>
```

NOTE

To avoid blank lines in your finished document, the first line of the section should be put on the same line as its corresponding "Begin". For example:

```
<Begin before:> .pdw 72
.pdl 60
.hla ...
```

Field Insertion

In order to insert information from the list into the document, a Field_name may be included in the text of the record section. The Field_name must be enclosed by angle brackets as, for example, "<lname>". For each record processed, the value of the field replaces the Field_name. Any number of fields of a record input in your listin file may be specified in the record section.

You can also specify the field width if you desire. For example, if you want space for ten characters each time "lname" is processed, you would input the Field_name in your listform file as "<lname,10>". The value of the field is then allotted space for ten characters. If the value is less than ten characters, white space (blanks) fills out the ten spaces. If the value of the field is greater than ten characters, the value is cut off at ten characters.

You can also specify whether the value of the field is to be centered, "c", aligned left, "l", or aligned right, "r". For example, for the value of the field, "lname", to be centered within space for ten characters, your input would be "<lname,10,c>". By default, the value of the field is left-aligned.

LISTFORM FILE EXAMPLE

You have created both a "presidents.listin" and a "presidents.lister" file. To format a document using records of those files, your listform file must contain a record section. Because the before and after sections are optional. They will not be included in this first example.

Invoke your editor and create a listform file. Observe that the width for each Field_name is specified (except for "term"). By default, the values of the fields will be left-aligned. Note, too, that the pathname is of your choosing but must contain the suffix "listform". You may prefer to keep the word "presidents" in the pathname for continuity (because both your lister and listin files are so-named). The example, however, uses a shorter pathname. Note, too, that in this example "<Begin record:>" is alone on a line and will produce blank lines between each record.

```
! qx                                        ! ted
! a
! <Begin record:>
! <No,5><lname,12><fname,8><state,5><born,6><died,6><wife,10><term>
! <end;>
! \f
! w pres.listform
! q
  r 15:19 0.279 14
```

THE PROCESS_LIST COMMAND

The **process_list** command, with the short name "pls", produces a document from all or selected records in a lister file. The process_list command must contain the command, the pathname of the lister file, the pathname of the listform file, and optional control arguments. If the pathname of the listform file is not included in the command line, the listform file used will be one that has the same entryname as the pathname of the lister file.

Type "pls presidents pres" and observe that you receive a print on your terminal of the document formatted in your "pres.listform" file using the records of your lister file. Note that had you assigned the pathname "presidents.listform" to your listform file rather than "pres.listform", your process_list command line could have been shortened so as to be simply "pls presidents".

```
! pls presidents pres

 1     Washington  George   VA    1732   1799   Martha      1789-1797

 2     Adams       John     MA    1735   1826   Abigail     1797-1801

 3     Jefferson   Thomas   VA    1743   1826   Martha      1801-1809

 4     Madison     James    VA    1751   1836   Dorothea    1809-1817
 r 15:19 0.170 22
```

Observe that each of the four records is printed on one line with the field values inserted in columns as specified by the order of the field names in the listform file. Observe, too, that by giving a field width to each Field_name, the listing gets processed in the form of columns. Also note that each record is separated by one blank line.

THE SELECT CONTROL ARGUMENT

The **-select** control argument (or "-sel") enables the process_list command as well as several other list processing commands to select only certain records for processing. This control argument consists of the following three parts: the **Field_name,** a **comparison_operator,** and a **test_string.**

The comparison_operator specifies what comparison is to be performed. This operator must be one of the following: contains, not contains, equal, not equal, greater, not greater, less, and not less. The term test_string is the value to which the Field_name is compared.

For example, to have the process_command select all records of your lister file that have a last name less than "M" (names preceding "M" in alphabetical order), type "pls presidents pres -sel "lname less M"". Note the quotation marks and also observe that this selection process is made with regard to a distinction between uppercase and lower case letters as are most of the comparison operators.

```
! pls presidents pres -sel "lname less M"

  2    Adams      John     MA   1735  1826  Abigail    1797-1801


  3    Jefferson  Thomas   VA   1743  1826  Martha     1801-1809
  r 12:42 0.348 49
```

As another example of the comparison_operator, type "pls presidents pres -sel "died equal 1826"".

```
! pls presidents pres -sel "died equal 1826"

  2    Adams      John     MA   1735  1826  Abigail    1797-1801


  3    Jefferson  Thomas   VA   1743  1826  Martha     1801-1809
  r 12:43 0.216 36
```

A reserved field name ":any" may be used to specify any field in a record. As an example of its use, type "pls presidents pres -sel ":any contains 1836"".

```
         !  pls presidents pres -sel ":any contains 1836"

            4    Madison     James   VA   1751  1836  Dorothea   1809-1817
            r 12:43 0.204 49
```

A special test_string ":null" is used to select a field missing from a record.

THE APPEND_LIST COMMAND

Suppose you wanted to add a new record to your "presidents.lister" file for formatting. Specifically, suppose you wanted to add a record for the fifth president, James Monroe. One method would be to call your editor, read "presidents.listin", append the record, and overwrite the file. Then, to create a new "presidents.lister" file, you would need to give the create_list command again.

A second method is to use the **append_list** command, given as "als". This command will add the new record directly to the lister file without affecting the listin file. The append_list command does require control arguments "-fn" (**-field_name**) in order to assign the field values to be entered. Thus, to add James Monroe to your "presidents.lister" file, type "als presidents -fn No 5 -fn lname Monroe -fn fname James -fn wife Elizabeth".

```
  ! als presidents -fn No 5 -fn lname Monroe -fn fname James -fn wife Elizabeth
    r 14:04 0.047 18
```

Verify that James Monroe has been added to your list by giving the process_list command, "pls presidents pres".

```
        ! pls presidents pres

          1    Washington  George  VA  1732  1799  Martha     1789-1797

          2    Adams       John    MA  1735  1826  Abigail    1797-1801

          3    Jefferson   Thomas  VA  1743  1826  Martha     1801-1809

          4    Madison     James   VA  1751  1836  Dorothea   1809-1817

          5    Monroe      James                   Elizabeth
          r 14:05 0.248 14
```

Next, verify that your "presidents.listin" file does not contain the James Monroe record. Invoke your editor and read the file, backup two lines and print the last three lines of "presidents.listin". Note that the last record of this file is as you originally entered it with field values for James Madison. Using the append_command has not affected this file.

```
! qx                                 ! ted
! r presidents.listin
! -2,$p
  =died 1836
  =wife Dorothea     ?Nicknamed Dolly?
  =term 1809-1817
! q
  r 14:05 0.137 10
```

THE EXPAND_LIST COMMAND

The **expand_list** command, whose short name is "els" may be used to create a listin segment from a lister segment. That is, the expand_list command operates oppositely to the create_list command, which formats a lister segment from a listin segment. Give the expand_list command to place the James Monroe record in your "presidents.listin" file by typing "els presidents". Then invoke your editor and read this file to see that the record has indeed been added.

```
! els presidents
  r 14:06 0.090 6

! qx                                      ! ted
! r presidents.listin                     ! r presidents.listin
  /Doro/,$p                                 /Doro/,$p
  =wife Dorothea                            = wife Dorothea
  =term 1809-1817                           =term 1809-1817
  $                                         $
  =No 5                                     =No 5
  =lname Monroe                             =lname Monroe
  =fname James                              =fname James
  =wife Elizabeth                           =wife Elizabeth
! q                                         ! q
  r 14:07 0.180 11                          r 14:07 0.180 11
```

You will note that the comment "Nickname Dolly" does not appear after "=wife Dorothea" in this printout although you inserted the comment originally in your listin file. Comments are not processed in the lister file. When the expand_list command is given, that command operates only on the lister file and overwrites your listin file. Because the Comment_delimiters do not appear in the lister file, they will then not appear in the listin file.

Your present "Monroe" record is also a good example to use to see how the special test_string ":null" can be used with the -select control argument. Type "pls presidents pres -sel "born equal :null" and note that the record not having a field value for "born" is printed.

```
pls presidents pres -sel "born equal :null"

5     Monroe       James                         Elizabeth
r 14:36 0.190 25
```

THE DISPLAY_LIST COMMAND

The **display_list** command, whose short name is "dils" is used to display or print selected portions of selected lister records. This command requires a -field_name control argument and usually uses the -select control argument. If the -select control argument is not included in the command line, all the records are used for the display of a Field_name. As an example, type "dils presidents -sel "lname equal Monroe" -fn fname" to find the first name in the record containing "Monroe". Then, type "dils presidents-sel "fname equal James "-fn lname" " to find which records in your lister file contain a first name of James.

```
! dils presidents -sel "lname equal Monroe" -fn fname
  James

  r 07:46 0.258 3

! dils presidents -sel "fname equal James" -fn lname
  Madison
  Monroe

  r 07:47 0.170 3
```

You may use more than one Field_name in the control line. For example, if you wanted to see the number and the first name, your -field_name control arguments would be "-fn No -fn fname". Your display would show the values of these fields in the order you type them. If a value does not exist, you will be given a blank.

THE DESCRIBE_LIST COMMAND

The **describe_list** command, typed as "dls", displays the information about a lister file. It is often easy to forget the characters used for the Record_delimiter or the Field_delimiter, as well as Field_names in a segment. The describe_list command displays that information for you as well as the name of the file, the time and date the command was given, and the total number of records involved. Selected records may be incorporated by using the -select control argument. As an example, type "dls presidents" to receive a display of all the information and the total number of records in your "presidents.lister" file. Then, type "dls presidents -sel "fname equal James" " to see the information involving the records that contain a first name of James. Note that two records are involved but there are a total of five records in the file.

```
! dls presidents
                         presidents.lister   05/07/99  0748.4 mst Fri

  Total Records:          5
  Record_delimiter:       $;
  Field_delimiter:        =;
  Field_names:            No, lname, fname, state, born, died, wife, term;

  r 07:48 0.153 26

! dls presidents -sel "fname equal James"
                         presidents.lister   05/07/99  0749.6 mst Fri

  Total Selected Records: 2
  Record_delimiter:       $;
  Field_delimiter:        =;
  Field_names:            No, lname, fname, state, born, died, wife, term;

  r 07:49 0.118 9
```

THE MODIFY_LIST COMMAND

Your record of James Monroe in your "presidents.lister" file is incomplete. This record does not contain field values for state, born, died, or term. The amend_list command may be used to insert a new record in the file but it cannot be used to change field values of that record. One alternative to adding the missing field values and/or correcting field values is by using the text editor on your listin file and then creating a new lister file by the create_list command. An easier solution is to use the **modify_list** command, given as "mdls".

The modify_list command can be used to insert, correct, or delete field values in your lister file. If the -select control argument is not given, all records will be modified. If the -select control argument is given, then only the values of the Field_names designated in the command line are modified in the selected record.

As a first example, give the "mdls" command to insert values for the state and born fields of the "Monroe" record in your lister file. However, rather than giving the correct "VA" value to state, give "MA" instead. Then give the process_list command to see your segment. Note that your lister file has been updated.

```
! mdls presidents -sel "lname equal Monroe" -fn state MA -fn born 1758
  r 07:50 0.098 27

! pls presidents pres

  1    Washington  George  VA   1732  1799  Martha     1789-1797

  2    Adams       John    MA   1735  1826  Abigail    1797-1801

  3    Jefferson   Thomas  VA   1743  1826  Martha     1801-1809

  4    Madison     James   VA   1751  1836  Dorothea   1809-1817

  5    Monroe      James   MA   1758        Elizabeth
  r 07:51 0.257 8
```

You have just seen how the modify_list command can be used to enter field values in a selected record. This same command is used to correct or delete field values. To correct a value, such as "MA" to "VA", the command line is typed with the -field_name control argument containing the correct value. To delete a value, the argument would be given with a field value containing only a number of spaces equal to the number of characters in the value, and with quotation marks enclosing the spaces. Thus, to change "MA" to "VA", the argument "-fn state VA" would be used. To delete "MA", the argument would be "-fn state" "".

Correct "MA" to "VA" by using the modify_list command. Then use the process_list command (selecting the "Monroe" record) to see the result. Finally, modify this record to include the died and term values. Remember that to use the -select control argument, you need select only one unique identifier in a record. For the "Monroe" records, all but the "fname" field values are unique to that record. "James" also appears in the "Madison" record and therefore is not a unique identifier.

```
   r 07:51 0.101 3

 ! mdls presidents -sel "lname equal Monroe" -fn state VA
   r 07:52 0.059 1

 ! pls presidents pres -sel "lname equal Monroe"

   5    Monroe      James    VA    1758          Elizabeth
   r 07:53 0.221 10

 ! mdls presidents -sel "No equal 5" -fn died 1831 -fn term 1817-1825
   r 07:54 0.064 4

 ! pls presidents pres -sel "No equal 5"

   5    Monroe      James    VA    1758  1831  Elizabeth 1817-1825
   r 07:55 0.219 2
```

Section 13
Using Listed Records

In Section 12, you saw how lists can be developed and formatted through WORDPRO's list processing facility. You were introduced to the listin file, the lister file, and the listform file. You were further introduced to several list processing commands such as the process_list command. This section expands those commands and shows how lists may be sorted, trimmed, and placed in output files for further use.

ADDING TO THE LISTIN FILE

Consider a record in which a column item may require more than one line. For example, while the tenth president, John Tyler, was in office his first wife died and he remarried. To incorporate this information, another Field_name (wife2) must be added to the header section of the listin file. The Field_name must also be added to the record section of your listform file (before "term" perhaps). Here, "wife2" will be placed directly under "wife" in the processed record.

For this example, evoke your text editor and append the following to your "presidents.listin" file. "No 10", "lname Tyler","fname John", "wife Letitia", "wife2 Julia". Then, substitute "term,wife2" for "term" in the header section of the file. Do not forget to overwrite. (Note that the record information could be added to your lister file by using the append_list command and then could be placed in your listin file by the expand_list command. However, because you need to add "wife2" to your listin file and also will need to use the create_list command, appending the listin file here saves time.)

```
! qx                                        ! ted
! r presidents.listin                       ! r presidents.listin
! a                                         ! a
! $                                         ! $
! =No 10                                    ! =No 10
! =lname Tyler                              ! =lname Tyler
! =fname John                               ! =fname John
! =wife Letitia                             ! =wife Letitia
! =wife2 Julia                              ! =wife2 Julia
! \f                                        ! \f
! /Field/ s/term/term,wife2/                ! /Field/ s/term/term,wife2/
! w                                         ! w
! q                                         ! q
  r 14:32 0.300 87                            r 14:32 0.300 87
```

Because a record has been added to the listin file, and you will want that record processed, the lister file must also be updated. Therefore, again command "cls presidents –tt" to update your lister file and note that you now have six records for processing.

```
! cls presidents -tt
  create_list: 6 records.
  r 14:32 0.344 43
```

The next step is to adjust the listform file to accommodate formatting the Field_name "wife2". Thus, invoke your editor and read "pres.listform". In your format, "wife2" should be placed to appear directly beneath "wife" in a column. Also in the format, "wife" begins at column position 43 (42 blank spaces before the value). Consequently, insert the Field_name "<wife2,10>", preceded by 42 spaces, before "<end;>".

```
! qx                                                          ! ted
! r pres.listform
! /end;/ i                                          <wife2,10>
!
! \f
! w
! q
  r 14:34 0.180 39
```

An alternative procedure is to input "Julia" in your listin file with 42 blank spaces preceding this name and the entire field value surrounded by quotation marks. That is, your appended input would then appear as:

```
a
$
=lname Tyler
=fname John
=wife Letitia
=wife2 "                                          Julia"
\f
```

However, if you used this procedure, you would have had to insert "wife2" at the beginning of its line in your listform file rather than 42 spaces in order to compensate for the 42 blank spaces in the value of the field. You could not have inserted "<wife2,10>" in you listform file because the value of the field exceeds the ten characters dictated within the Field_name. If you were to insert "<wife2,10>" in this alternative case, your processed list would show only a blank line because only the first ten characters (blank spaces) of the value of the field would be printed. The remaining 32 blank spaces and the name "Julia" would be cut off.

You are now ready to process the list again. Type "pls presidents pres" again and observe that "wife2" has been inserted beneath "wife" and that "null" values (no Field_name or field value) were left blank in the columns of their respective Field_names.

```
! pls presidents pres
  1    Washington  George  VA   1732  1799  Martha     1789-1797

  2    Adams       John    MA   1735  1826  Abigail    1797-1801

  3    Jefferson   Thomas  VA   1743  1826  Martha     1801-1809

  4    Madison     James   VA   1751  1836  Dorothea   1809-1817

  5    Monroe      James   VA   1758  1831  Elizabeth  1817-1825

  10   Tyler       John                     Letitia
                                            Julia
  r 14:36 0.197 34
```

UNIQUE IDENTIFIERS

As you have seen, selecting any one record in a lister file for the process_list command or the modify_list command requires the use of the -select control argument and a **unique identifier**. In the examples given thus far, most of the unique identifiers have been "lname", but as you can readily imagine, a long file of records could have several "lname" values that are identical. In fact, in a long file of records, many of the field values could be identical and choosing one that is unique to a record could become a major selection process.

Although you have not seen it, a number is assigned to each record as a unique identifier within the file when any lister record is created. This identifier remains assigned to that record for the duration of the record. If the record is deleted from the file, its unique identifier is also deleted and is not reused. However, whenever the create_list command is given to recreate a lister file, all unique identifiers are changed.

You can use this unique identifier whenever you use the -select control argument by specifying the reserved field name, ":uid". That is, you can select a given record by typing "-sel ":uid equal XXX"" If you have first entered "<:uid>" in the Record section of your listform file, and where "XXX" is the unique identifier assigned to the record you want selected.

As an example, invoke your text editor, read your "pres.listform" file, and then insert this reserved field name in the Record section. The position you place this reserved field name in the Record section will determine the position of the unique identifier in the record listing on the printout. For example, if you place "<:uid>" immediately after "<Begin record:>" on a line before your field insertion values, the unique identifier will print on a separate line ahead of the line to which it belongs. To place the identifier on the same line as the field insertion values, you must incorporate an allotted space or the identifier will run into the first field value (for the example here, the identifier would run into the field value for "No" without leaving an intervening space).

For the example at hand, insert (using the substitute request) "<:uid,5>" to precede "<No,5>", thereby placing the identifier on the same line as the record on the printout and allowing five spaces for it. Be sure to overwrite your file. Then, quit your editor and give the process_list command. You will see the unique identifier on each record as a number (beginning with "1") beginning the line of each record.

```
! qx                                                     ! ted
! r pres.listform
! /<No/ s//<:uid,5><No/
! 1,$p

  <Begin record:>
  <:uid,5><No,5><lname,12><fname,8><state,5><born,6><died,6><wife,10><term>
                                               <wife2,10>
  <end;>
! w
! q
  r 13:15 0.442 28

! pls presidents pres

  1    1    Washington  George  VA   1732  1799  Martha    1789-1797


  2    2    Adams       John    MA   1735  1826  Abigail   1797-1801


  3    3    Jefferson   Thomas  VA   1743  1826  Martha    1801-1809


  4    4    Madison     James   VA   1751  1836  Dorothea  1809-1817


  5    5    Monroe      James   VA   1758  1831  Elizabeth 1817-1825


  6    10   Tyler      John                      Letitia
                                            Julia
  r 13:15 0.242 15
```

In this printout, observe that each record has its own unique decimal identifier. Also, observe that all records are shifted to the right to accommodate the five spaces you just allocated for the identifier. The second line of "Tyler", however, was not affected, and to be complete, "Julia" should be shifted five places, too. Use your modify_list command with the -select control argument and the reserved field name ":uid" to find and correct this record. Then, again give the process_list command to see the corrected record.

```
! mdls presidents -sel ":uid equal 6" -fn wife2 "      Julia"
  r 13:20 0.070 27

! pls presidents pres -sel ":uid equal 6"

  6    10    Tyler        John                                Letitia
                                                              Julia

  r 13:21 0.228 13
```

SORT CONTROL ARGUMENT

The process_list command has optional control arguments that include sorting records, selecting only certain records for processing (by using the -select control argument which you have seen), and specifying an output file such as a compin file.

The **-sort** control argument (or "-st") is used to change the order in which the list is processed. As an example of the use of the -sort control argument, type "pls presidents pres -st lname" and observe that the records are processed with the last names of the presidents listed in ascending (A to Z) alphabetical order.

```
! pls presidents pres -st lname

  2    2     Adams        John     MA   1735   1826   Abigail    1797-1801

  3    3     Jefferson    Thomas   VA   1743   1826   Martha     1801-1809

  4    4     Madison      James    VA   1751   1836   Dorothea   1809-1817

  5    5     Monroe       James    VA   1758   1831   Elizabeth  1817-1825

  6    10    Tyler        John                        Letitia
                                                      Julia

  1    1     Washington   George   VA   1732   1799   Martha     1789-1797
  r 12:40 0.217 28
```

An optional **–order** control argument can be used to list the presidents in descending (Z to A) order. Type "pls presidents pres –st "lname –dsc"" and observe that the list is now processed with the last names in descending alphabetical order. Do not forget the quotation marks around "lname –dsc". By default, as in the first–sort example, the–order control argument is "–asc" for ascending.

```
! pls presidents pres -st "lname -dsc"

  1    1    Washington  George   VA   1732   1799  Martha     1789-1797

  6    10   Tyler       John                       Letitia
                                                   Julia

  5    5    Monroe      James    VA   1758 1831    Elizabeth  1817-1825

  4    4    Madison     James    VA   1751 1836    Dorothea   1809-1817

  3    3    Jefferson   Thomas   VA   1743 1826    Martha     1801-1809

  2    2    Adams       John     MA   1735 1826    Abigail    1797-1801

  r 12:40 0.217 34
```

Two "wife" fields have values of "Martha" in your lister file. If the –sort control argument is given for "wife", the listing will be processed with "Martha" listed in the order it appears in the lister file. That is, Martha Washington will appear before Martha Jefferson because Washington precedes Jefferson in your present lister file. However, by typing the –sort control argument as "–st "wife lname"", the sort will first alphabetically list the records for "wife". Then, with the choice of two "wife" fields of equal value, sort will compare the field values for "lname". Use this sort by typing "pls presidents pres –st "wife lname"" (again note the quotation marks around the Field_names).

```
! pls presidents pres -st "wife lname"
  2   2   Adams       John     MA   1735  1826  Abigail    1797-1801

  4   4   Madison     James    VA   1751  1836  Dorothea   1809-1817

  5   5   Monroe      James    VA   1758  1831  Elizabeth  1817-1825

  6   10  Tyler       John                      Letitia
                                                Julia

  3   3   Jefferson   Thomas   VA   1743  1826  Martha     1801-1809

  1   1   Washington  George   VA   1732  1799  Martha     1789-1797
  r 12:41 0.221 56
```

Type "pls presidents pres –st died" and observe that this control argument causes the years to be placed in numerical order. In Tyler's record, the "died" field value is **null** and thus the processing causes this record to be placed in front of the remaining records.

```
! pls presidents pres -st died
  6   10  Tyler       John                      Letitia
                                                Julia

  1   1   Washington  George   VA   1732  1799  Martha     1789-1797

  2   2   Adams       John     MA   1735  1826  Abigail    1797-1801

  3   3   Jefferson   Thomas   VA   1743  1826  Martha     1801-1809

  5   5   Monroe      James    VA   1758  1831  Elizabeth  1817-1825

  4   4   Madison     James    VA   1751  1836  Dorothea   1809-1817
  r 12:41 0.203 36
```

You have seen the effect of the descending –order control argument used with the –sort control argument. When no –order control argument is specified, an ascending order is used by default. The ascending order follows the alphanumeric sequence "01234...89AaBa....Zz". An –alphabetic control argument, given as "–alp", will cause the list to be sorted alphabetically only, while a –numeric control argument, given as "–num", will result in a list sorted numerically.

When there is no numeric value in the field of a record being sorted, or if the field is null, that field is treated as having a zero value and that record will not be listed when the numeric control argument is used with the –sort control argument. For example, type "pls presidents pres –st "died –num"" (note the quotation marks) and compare this list with the previous list. Observe that "Tyler" is not listed here when the –numeric control argument is used because the "died" field has no value.

```
! pls presidents pres -st "died -num"
  1   1   Washington  George  VA   1732  1799  Martha     1789-1797

  2   2   Adams       John    MA   1735  1826  Abigail    1797-1801

  3   3   Jefferson   Thomas  VA   1743  1826  Martha     1801-1809

  5   5   Monroe      James   VA   1758  1831  Elizabeth  1817-1825

  4   4   Madison     James   VA   1751  1836  Dorothea   1809-1817
  r 12:44 0.243 55
```

THE SORT_LIST COMMAND

The **sort_list** command is used to sort records of a specialized lister file. The sort_list command is similar to the -sort control argument of the process_list command in operation except that the order of the records in the lister file is changed permanently (until a new create_list command or another overriding command is given). The lister file is not changed when the -sort control argument is used with the process_list command.

The sort_list command is given as "sls pathname -control_arg", where "pathname" is the name of the existing lister file (with the suffix "lister" given or assumed) and "-control_arg" is similar to the -sort control arguments of the process_list command.

Type "sls presidents -st lname" to permanently place the present records of your lister file alphabetically (ascending) by "lname". Then, type the process_list command, "pls presidents pres" and observe that the records have indeed been placed in a new order.

```
! sls presidents -st lname
  r 12:45 0.066 29

! pls presidents pres

  2    2    Adams        John     MA   1735   1826  Abigail    1797-1801


  3    3    Jefferson    Thomas   VA   1743   1826  Martha     1801-1809


  4    4    Madison      James    VA   1751   1836  Dorothea   1809-1817


  5    5    Monroe       James    VA   1758   1831  Elizabeth  1817-1825


  6    10   Tyler        John                       Letitia
                                                    Julia

  1    1    Washington   George   VA   1732   1799  Martha     1789-1797

  r 12:45 0.228 24
```

THE TRIM_LIST COMMAND

The **trim_list** command is used to delete selected records from a specified lister file without affecting the listin file. In other words, you may want to maintain your listin file intact (where records, if you wished, could be deleted by using your text editor) but process only selected ones. This command works similarly to the -select control argument of the process_list command without giving the -select control argument each time. However, as with the sort_list command, the lister file is permanently altered.

The trim_list command is given as "tls pathname -control_arg" where "pathname" is the pathname of the existing lister file and "-control_arg" may be "-tt" to print the number of records deleted and/or the -select control argument similar to the -select control argument for the process_list command.

Type "tls presidents -sel "died equal :null" -tt" to delete any records having a null "died" field in your "presidents.lister" file and also to receive a tally of the number of deleted records.

```
! tls presidents -sel "died equal :null" -tt
  trim_list: 1 record deleted.
  r 13:01 0.099 43
```

Note that if the test_string contains spaces, the test_string must further be enclosed in quotation marks within the -select statement that itself must be surrounded by quotation marks.

THE COPY_LIST COMMAND

In the sort_list and trim_list commands, your present lister file is altered permanently (until a new create_list command is given). You may have occasion to want to save your altered lister file with a new pathname.

The copy_list command creates a new lister file from an existing lister file. The copy_list comand is given as "cpls path1 path2 -control_args", where "path1" is the pathname of your existing lister file and "path2" is the pathname of the second lister file; "-control_args" are "-tt" for a print of the number of records copied and "-sel" to select records to be copied. If "-sel" is not specified, all records in "path1" are copied into "path2". The "-sel" control argument is identical to the "-sel" control argument of the process_list command.

Type "cpls presidents firstpres" to copy the existing records of your "presidents.lister" file into a "firstpres.lister" file.

```
! cpls presidents firstpres
 r 13:02 0.174 34
```

Next, create a new "presidents.lister" file (actually, the same file you had prior to the sort_list and trim_list commands) by giving the create_list command, "cls presidents".

```
! cls presidents
 r 13:02 0.153 12
```

Verify the records within each lister file by first typing the command, "pls presidents pres" and then typing "pls firstpres pres".

```
! pls presidents pres

   1   1   Washington  George  VA  1732  1799  Martha     1789-1797

   2   2   Adams       John    MA  1735  1826  Abigail    1797-1801

   3   3   Jefferson   Thomas  VA  1743  1826  Martha     1801-1809

   4   4   Madison     James   VA  1751  1836  Dorothea   1809-1817

   5   5   Monroe      James   VA  1758  1831  Elizabeth 1817-1825

   6   10  Tyler       John                    Letitia
                                               Julia

   r 13:02 0.216 25

! pls firstpres pres

   1   2   Adams       John    MA  1735  1826  Abigail    1797-1801

   2   3   Jefferson   Thomas  VA  1743  1826  Martha     1801-1809

   3   4   Madison     James   VA  1751  1836  Dorothea   1809-1817

   4   5   Monroe      James   VA  1758  1831  Elizabeth 1817-1825

   5   1   Washington  George  VA  1732  1799  Martha     1789-1797
   r 13:03 0.189 37
```

One aspect of the "firstpres.lister" file that can easily go unnoticed has to do with the unique identifiers. The sort_list command and the trim_list command both permanently changed your then existing "presidents.lister" file. The copy_list command took that file and then created a new lister file having one less record and with the remaining records in a new sequence. Because a new lister file was created, new unique identifiers were assigned to your "firstpres.lister" file.

OUTPUT FILE FORMATTING

One of the control arguments of the process_list command is the **-output_file path** control argument, with the shortname "-of path". This control argument specifies that the document produced by the process_list command is saved in a segment specified by "path". In the examples to be given here, the processed documents are to be saved in a compin file. Therefore, the control argument must be typed as "-of pathname.compin".

Consider placing a title line and column headings for your presidents listing. This information, using the compose command control lines, can be placed in the before section of your listform file. As an example, include "PRESIDENTS" as a title line centered 3 lines above the column headings. For column headings, a horizontal tab format may be used (or the headings may be input as a line of text with the fill mode off). Include a page definition length control line of 30 lines.

A major point to consider in this format using the compose command is the fill on control line. The compose command, by default, has a fill on control line that must be turned off. If a fill off control line is not included prior to the record section (or at the very beginning of the record section), the compose command will automatically fill in to cancel the field width spaces stipulated with the Field_names. Thus, after your column headings and before your first list format line of the record section of this listform file, be sure to include a ".fif" control line.

In the after section of this listform file, include a statement "First 5 presidents, their wives, dates".

Use the text editor to create this file. It should appear similar to the following. Note the new pathname used for this example and the rearrangement of the Field_names for the columns as compared to your "pres.listform" file.

```
! qx                                          ! ted
! a
! <Begin before:>
! .pdl 30
! .tlh 3 0 ||PRESIDENTS||
! .htd names 6,25,31,37,47,53,
! .htn ; names
! No.;President;State;Born;Term;Died;Wife
! .spb 2
! .htf ;
! .fif
! <end;>
! <Begin record:>
! <No,5><lname,12><fname,8><state,5,c><born,6><term,10><died,6><wife>
! <end;>
! <Begin after:>
! First 5 presidents, their wives, dates.
! <end;>
! \f
! w table.listform
! q
  r 13:29 0.192 32
```

NOTE

No input lines are allowed between any "<end;>" and the next "Begin".

Give the "pls" command using the "presidents.lister" file. Select the first five presidents for your document and output the file to a "presidents.compin" segment. Type "pls presidents table -sel "lname not equal Tyler" -of presidents.compin". (Note that your -select control argument could be typed as "-sel ":uid not equal 6"" instead of as "-sel "lname not equal Tyler"" to receive the same record selection.) Observe that your file is not printed and you receive only the ready message.

```
! pls presidents table -sel "lname not equal Tyler" -of presidents.compin
  r 13:34 0.181 44
```

Now format yor "presidents.compin" file by using the compose command. Type "comp presidents".

```
! comp presidents




                         PRESIDENTS


No.  President            State Born  Term       Died  Wife


1    Washington  George    VA   1732  1789-1797  1799  Martha

2    Adams       John      MA   1735  1797-1801  1826  Abigail

3    Jefferson   Thomas    VA   1743  1801-1809  1826  Martha

4    Madison     James     VA   1751  1809-1817  1836  Dorothea

5    Monroe      James     VA   1758  1817-1825  1831  Elizabeth

First 5 presidents, their wives, dates.



r 13:35 1.663 239
```

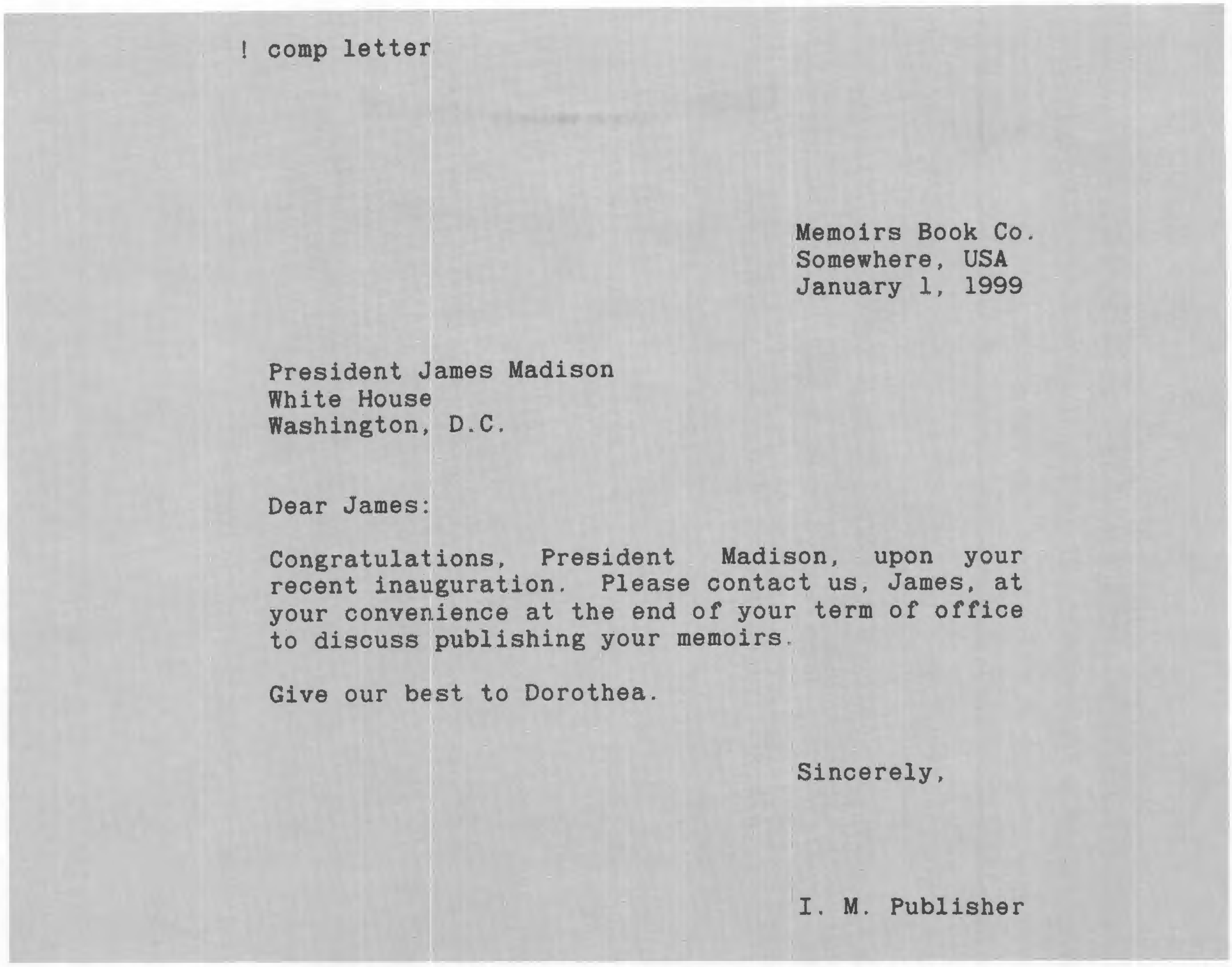
In this example, you used the "presidents.lister" file in the process_list control line and then selected five of the six records. You could alternatively have used your trimmed file, "firstpres.lister", instead and then not needed the -select control argument.

Observe that the -select control argument used in the example specifically calls out the Tyler record. Because you wanted the first five records, it would have been tempting to ask "-sel "No less 10"". However, the greater, not greater, less, and not less comparison_operators select only alphabetically and not numerically. Likewise, it might be tempting to ask for only those records that do not have a null field. But, had you asked "-sel ":any not equal :null"", you would not have received any records. Although the Tyler record has null fields for state, born, died, and term, the other five records have null fields for wife2.

Another point has to do with the before and after sections of the listform file, which are optional sections. If the text in these sections is placed in the record section of the file, then that text would be repeated for each record.

As a final example, input the following listform file which is to be used as a form letter. Observe the use of the compose command control lines, especially the fill on and fill off control lines, and the use of field names. Write the file as "letter.listform". Give the command "pls presidents letter -sel "lname equal Madison" -of letter.compin". Then give the command "comp letter".

```
! qx
! a
! <Begin before:>
! .pdw 50
! .inl 35
! .fif
! Memoirs Book Co.
! Somewhere, USA
! January 1, 1999
! <end;>
! <Begin record:>
! .spb 2
! .inl
! President <fname> <lname>
! White House
! Washington, D.C.
! .spb 2
! Dear <fname>:
! .fin
! .spb
! Congratulations, President <lname>, upon your recent
! inauguration.  Please contact us, <fname>, at your
! convenience at the end of your term of office to
! discuss publishing your memoirs.
! .spb
! Give our best to <wife>.
! <end;>
! <Begin after:>
! .spb 2
! .inl 35
! .fif
! Sincerely,
! .spb 4
! I. M. Publisher
! <end;>
! \f
! w letter.listform
! q
  r 13:49 0.540 115

! pls presidents letter -sel "lname equal Madison" -of letter.compin
  r 13:50 0.186 49
```



```
! comp letter



                                    Memoirs Book Co.
                                    Somewhere, USA
                                    January 1, 1999


President James Madison
White House
Washington, D.C.


Dear James:

Congratulations, President  Madison, upon your
recent inauguration. Please contact us, James, at
your convenience at the end of your term of office
to discuss publishing your memoirs.

Give our best to Dorothea.

                                    Sincerely,



                                    I. M. Publisher
```

Section 14
Electronic Mail

The Multics WORDPRO system permits a user to communicate and to electronically mail messages and documents to other users by data terminal through the Multics system. Six commands (with control arguments) are described in this discussion to show how one-line messages as well as lengthy documents and existing files may be sent by one WORDPRO user to another.

The mail used in the examples in the following discussions is fictitious and is presented for demonstration purposes only.

THE SEND_MAIL COMMAND

The **send_mail** command, with the short name "sdm" is the command used to send mail from one user to one or more other system users.The mail sent remains stored in a **mailbox segment** at its destination until a command to print the mail is given by the addressee.

As in the postal service, mail to be sent must be correctly addressed. In the case of electronic mail, a mailbox address is a user's User_id (Person_id.Project_id). Thus, if your Person_id is TSmith and your Project_id is ProjA, your User_id and mailbox address are TSmith.ProjA. In the examples that follow, the mailbox addresses are assumed to be Washington.Chief, Jefferson.State, Hamilton.Treas, Knox.War, and Randolph.Attgen.

The send_mail command is followed on the same line by the mailbox address of the destination. Thus, if Hamilton were to send mail to Washington, Hamilton's command (after a ready message) would be "send_mail Washington.Chief" or, if he used the short name, "sdm Washington.Chief". The carriage return at the end of this mailbox address causes the system to respond by printing "Subject:".

```
  r 10:20 0.053 1

! sdm Washington.Chief
  Subject:
```

At this point, the user can type in a subject line such as "Silver dollar" or press the carriage return if a subject for the message is not appropriate or not desired. The system next responds with the word "Message:" and the message can now be typed.

```
  sdm Washington.Chief
! Subject: Silver dollar
  Message:
```

The message input is ended (terminated) by typing a period alone on a line. (Use the space bar and carriage return to insert a blank line.) If the message has been correctly addressed, the system will respond with the message the mail has been delivered.

```
  r 10:20 0.053 1

! sdm Washington.Chief
! Subject: Silver dollar
  Message:
! Mr. President:
!
! Have report you threw silver dollar across Potomac.
! What department provided funding?
!
!         Alexander Hamilton, Secretary of the Treasury
! .
  Mail delivered to Washington.Chief.
  r 10:22 0.641 198
```

THE PRINT_MAIL COMMAND

The user's mailbox is created automatically the first time the user invokes the **print_mail** command (or print_messages or accept_messages commands described later). The command is given as either "print_mail" or, by the short name, "prm".

When the print_mail command is given, the system first tells the user the number of messages there are in the mailbox. Then each message is printed, preceded by the date, time, sender, subject, and information that identifies the recipients. When user Washington commands "print_mail", he will receive Hamilton's message as follows:

```
 r 14:10 1.179 52

! prm
 You have one message.

  #1 (6 lines)  10/23/93 10:22  Mailed by: Hamilton.Treas
 Date: 23 October 1793 10:22 est
 From:  Hamilton.Treas
 Subject:  Silver dollar
 To:  Washington.Chief

 Mr. President:

 Have report you threw silver dollar across Potomac.
 What department provided funding?

          Alexander Hamilton, Secretary of the Treasury

 print_mail: Delete #1?
```

At the end of each message, the system asks the user whether the message can be deleted. A "no" answer preserves the message for later review; it will be printed each time a print_mail command is given by the user. A "yes" answer deletes the message from the mailbox. In the example at hand, Washington answers "no" and the message remains in his mailbox.

```
! print_mail: Delete #1?  no
 r 14:11 1.397 175
```

A **-list** control argument can be used with the print_mail command. This control argument, with the short name "-ls", gives a short summary of all messages in the mailbox before the first message is printed. Thus, in the case of Hamilton's message to Washington, if Washington had typed "prm -ls", he would have received the following summary, followed immediately by the first message (only the first line is repeated here). In the case of several messages, a one-line summary of each message is given and then the messages are printed in sequence.

```
    r 15:48 1.248 18

  ! prm -ls
    You have one message.

    Msg#  Lines   Date      Time   From                          Subject
      1     (6)  10/23/93  10/22  Hamilton.Treas                Silver dollar

     #1 (6 lines)  10/23/93 10:22  Mailed by: Hamilton.Treas
```

SEND_MAIL CONTROL ARGUMENTS

Several control arguments can be added in the send_mail command line. One control argument permits you to send copies of the original message to secondary recipients much as you would send carbon copies of office correspondence. Another control argument permits you to find out if mail you have sent has been printed and read by the addressee.

Copies

A **carbon copy** control argument typed as "-cc" added to the send_mail command line permits secondary recipients to receive a copy of a message addressed to a primary recipient. For example, if Washington wished a copy of his reply to Hamilton to be sent to Randolph, the Attorney General, the command line would by typed as "sdm Hamilton.Treas -cc Randolph.Attgen".

```
! sdm Hamilton.Treas -cc Randolph.Attgen
! Subject: Silver dollar
  Message:
! Alexander:
!
! River was Rappahannock, not Potomac.
!
! I did not use government funds.  I earned the silver
! dollar selling wood from a cherry tree and from selling
! cherry pies Martha baked.
!
!      George Washington, President
! .
  Mail delivered to Hamilton.Treas.
  Mail delivered to Randolph.Attgen.
  r 14:27 0.870 301
```

When the –cc control argument is included on the send_mail command line, all recipients, primary and secondary (in this case Hamilton and Randolph), receive the same message.

```
  r 14:30 0.180 31

! print_mail
  You have one message.

   #1 (9 lines)  10/23/93 14:27  Mailed by: Washington.Chief
  Date:  23 October 1793 14:27 est
  From:  Washington.Chief
  Subject:  Silver dollar
  To:  Hamilton.Treas
  cc:  Randolph.Attgen

  Alexander:

  River was Rappahannock, not Potomac.

  I did not use government funds.  I earned the silver
  dollar selling wood from a cherry tree and from selling
  cherry pies Martha baked.

       George Washington, President

! print_mail: Delete #1?   no
  r 14:30 0.645 167
```

No Abort

The send_mail command, by default, does not send any mail if any User_id (mailbox address) of any recipient is misspelled or otherwise incorrect. That is, by default, the send_mail command has an **-abort** control argument". However, a **-no_abort** control argument can be placed on the send_mail command line to cause the mail to be sent to as many recipients as possible of those listed in the command line. In other words, the -no_abort control argument permits the send_mail command to send mail to all correctly addressed mailboxes even when other addresses in the command line do not exist or are input incorrectly.

NOTE

The -no_abort control argument should be used sparingly, although it will be used in future examples here. This control argument may mean sending duplicate messages, some recipients receiving one message while others receive the same message more than once, and not all recipients knowing who has or has not received the message.

Acknowledge

Very often, you will want to know whether your mail has been received and read. By default, the send_mail command has a **-no_acknowledge** control argument. However, the **-acknowledge** control argument, or its short name "-ack", when added to the send_mail command line, will cause a message to be sent back to you when a recipient has given a print_mail command, and you will indeed know that your mail has been read.

If you are not logged in, log in and send a message from Hamilton to Washington with a copy to Randolph. Because you can, of course, send mail to yourself, include your User_id in the -cc control argument. The -no_abort control argument must be included now so that you will receive the message because the other mailboxes are obviously fictitious, and without the -no_abort control argument, the message will not be sent to anyone. In addition, include the -ack control argument. Thus, type the message as shown in the following example, substituting your User_id for "TSmith.ProjA". (Disregard typing errors you may make although you can use "#" and "@" which apply to all Multics system input for correction. Editing of messages using qedx, but not using Ted, is described later.)

```
! sdm Washington.Chief -cc Randolph.Attgen TSmith.ProjA -no_abort -ack
! Subject: Silver dollar
  Message:
! George
!
! I have heard that three senators have left on a fact-finding tour to
! the rivers.  It seems that the opposition party wants
! a complete congressional investigation.  The situation
! is getting serious.
!
! I have also learned the IRS will audit your returns
! for unreported income.
!
!      Alex
!
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>Chief>Washington>Washington.mbx
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>Attgen>Randolph>Randolph.mbx
  Mail deliverd to TSmith.ProjA.
  send_mail (send): The message was sent to some but not all destinations.

  send_mail:
```

Observe that although you have terminated the message (with the period), the system adds information telling you it cannot deliver the message to either Washington or Randolph. However, the mail has been delivered to your mailbox. (You are a secondary recipient of your own mail by virtue of the "–cc" control argument.)

Note that the system suddenly stops after printing "send_mail:". At this point you are to give an answer, a send_mail request, of which you have various choices. However, at this time you are ready to return to command level because the system has said the mail has been delivered, at least to you. To return to command level and to exit from the electronic mail sending facility, your response to this prompt of "send_mail:" is "quit". With this response, the system will then ask if you really want to quit and return to command level. The answer here is "yes". Thus, type "quit" to the system prompt "send_mail:" and to the question that follows, type "yes". (Do no confuse this "quit" with the text editor request "q" or the break signal "QUIT".)

```
  send_mail (send): The message was sent to some but not all destinations.

! send_mail: quit

  send_mail (quit): Message has not been sent, saved, or written.
! Do you still wish to quit?   yes
  r 13:49 0.825 135
```

NOTE

If you wish to try the earlier examples, and wish to receive the mail, be sure to include the –no_abort control argument and your User_id in a –cc control argument in the sdm command line.

If you now give the command "prm -ls", you will find you have only this latest message in your mailbox. On the other hand, Hamilton will have four messages after you, Washington, and Randolph have all given the print_mail command and seen the mail he sent (three of the messages simply acknowledging reception of the mail). The beginning of the print Hamilton will receive when he gives his "prm -ls" command would appear as follows:

```
! prm -ls
  You have 4 messages.


  Msg#  Lines   Date     Time    From               Subject
     1    (9) 10/23/93   14:27   Washington.Chief   Silver dollar
     2    (1) 10/24/93   13:49   Washington.Chief   Acknowledge message<MORE>
     3    (1) 10/24/93   13:49   Randolph.Attgen    Acknowledge message<MORE>
     4    (1) 10/24/93   13:49   TSmith.ProjA       Acknowledge message<MORE>

   #1 (9 lines)  10/23/93 14:27  Mailed by: Washington.Chief
  Date:  23 October 1793 14:27 est
  From:
```

The use of "<MORE>" in the subject column refers to the fact that the subject line has been cut because of its length. That is, the lines that are given as "Acknowledge message<MORE>" are shortened so that the message summary can appear on one line with a length of no more than 72 characters. The "<MORE>" in this case completes the subject line which otherwise would appear similar to the following:

```
Acknowledge message of 10/24/93  1348.7 est Wed
```

The actual acknowledge message is treated as any other message and therefore appears in the body of mail. As an example, the acknowledge message to Hamilton from Randolph is generated automatically when Randolph gives his print_mail command and will appear at Hamilton's terminal as:

```
 #3 (1 line)  10/24/93 13:49  Mailed by: Randolph.Attgen
Acknowledge message of 10/24/93  1348.7 est Wed

print_mail: Delete #3?
```

To

Although you can send mail to several primary recipients by typing their mailbox addresses after "sdm", the **-to** control argument permits you to send the message to even more primary recipients. (Remember that the -cc control argument lets you send copies to as many secondary recipients as you desire.)

From

On occasion, you may have need to send a message for someone else. The **-from** control argument is used to replace your name in the "From:" line of the header of the received message with the name of the author (or authors) of the message. The recipient of the mail, however, will know that you were the actual sender by virtue of the "Mailed by:" statement in the first line of the received message. Both the -to and -from control arguments require correct mailbox addresses as part of the control argument.

Fill

The send_mail command, by default, has a **-no_fill** control argument so that the message is sent with line lengths as you type them as input. The **-fill** control argument produces a message with a justified left margin but with a ragged right edge. However, the lines are filled out with as many characters as possible and yet remain within a line length limit that, by default, is 72 characters. The –fill control argument works similarly to the fill on and align left control lines (".fin" and ".all") used for the compose command.

Line Length

The **–line length N** control argument, with the short name "–ll N", is used to adjust the default line length of 72 characters for the message when the –fill control argument is used. "N" is the number of character spaces for the desired line length.

Input File

The **–input_file** control argument, with the short name "–if", is used in the send_mail command line when an existing file is to be sent without change and without an additional message. That is, if you were to send, by electronic mail, a file you already have in permanent storage, you would use this control argument with the file name included. For example if you wished to send your "Preamble.compin" file (see Section 6), this control argument would be typed as "–if Preamble.compin" and that file would be sent as mail without change. (Note that you generally would not want to send a compin file because all the compose command control lines would be included.)

As an example of the use of some of these control arguments, consider having Washington send a message to Hamilton using the –fill control argument and a line length of 50 characters. Include a copy for yourself and be sure to add the –no_abort control argument. Use the –to control argument to add Knox.War as a primary recipient. (Note that the –to control argument is used in this example, but send_mail will recognize more than one addressee immediately following "sdm". That is, the system will recognize either "sdm Hamilton.Treas Knox.War" or "sdm Hamilton.Treas –to Knox.War" as being the same command.)

```
! sdm Hamilton.Treas -to Knox.War -fill -ll 50 -no_abort -cc TSmith.ProjA
! Subject: Silver dollar and cherry tree/pies
  Message:
! Alex and Henry:
!
! I cannot tell a lie.  I cut down my father's cherry tree
! with my little hatchet and reported the profit from
! the wood and the pies Martha baked to the IRS.  The silver dollar
! fell into the river and was reported as a short term capital
! loss.
!
! The pies were very good but I chipped a tooth of
! my wooden dentures on a cherry pit.
!
!      George
!
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>Treas>Hamilton>Hamilton.mbx
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>War>Knox>Knox.mbx
  Mail delivered to TSmith.ProjA.
  send_mail (send): The message was sent to some but not all destinations.

! send_mail:  quit

  send_mail (quit): Message has not been sent, saved, or written.
! Do you still wish to quit?  yes
  r 15:11 1.492 198
```

Type "prm –ls" and note that you have two messages. Recall that as soon as the print_mail command finishes giving the summary of messages it begins printing the messages, one by one, and gives you a chance to either save or delete each message as it is printed. Save both messages. The second message is the one you just sent to Hamilton, Knox, and yourself. Observe that the heading of the message gives the name of the sender (Mailed by:); the author (From:); the subject; the primary recipients (To:); and the secondary recipients (cc:). The body of the message has been formatted for a maximum of 50 characters with a ragged right margin as commanded by the –fill and "–ll 50" control arguments.

```
! print_mail: Delete #1?   no

   #2 (12 lines)  10/25/93/ 15:11  Mailed by: TSmith.ProjA
  Date:  25 October 1793 15:11 est
  From:  TSmith.ProjA
  Subject:  Silver dollar and cherry tree/pies
  To:  Hamilton.Treas, Knox.War
  cc:  TSmith.ProjA

  Alex and Henry:

  I cannot tell a lie.  I cut down my father's
  cherry tree with my little hatchet and reported
  the profit from the wood and the pies Martha baked
  to the IRS.  The silver dollar fell into the river
  and was reported as a short term capital loss.

  The pies were very good but I chipped a tooth of
  my wooden dentures on a cherry pit.

       George

! print_mail: Delete #2?  no
  r 10:07 1.506 159
```

EDITING YOUR MESSAGE

In the examples given thus far in this discussion of the electronic mail facility, you have used the period to mean: "The message is complete and ready to be sent". You have not had the opportunity to make corrections to the message (other than using "#" or "@"). If, however, you type the text editor input mode terminator, "\f", instead of the period, you will immediately invoke the text editor and all text editor requests will be honored. But, the editor requests will be honored only for the message and not for any part of the heading.

The text editor quit request "q" at the end of your editing cycle puts you into a "send_mail:" request loop where you can make changes to the heading, such as adding or removing addresses. However, for these changes, *you can use only the qedx text editor,* you cannot use Ted.

The following example of a message from Knox to Washington shows the use of "\f" instead of the period and how the body of the message can be edited. Note that typing "q" results in a system response of "send_mail:". Typing "send" to this system prompt delivers the mail and typing "quit" after the mail has been sent returns you to the Multics command level.

```
! sdm Washington.Chief -no_abort -cc Jefferson.State TSmith.ProjA
! Subject: Silver dollar salvage
  Message:
! Prez:
!
! Is their@Is there dangg#er British can calvage silver
! dolar from river to help finance future Was of 1812, and
! can hatchet be manufactured for national armament stockpile?
!
!      Henry
! \f
! /calvage/ s//salvage/
! /dolar/ s//dollar/
! /Was/ s//War/ p
  dollar from river to help finance future War of 1812, and
! /Henry/ i
! Sorry aboutyour tooth.  I know an excellent woodcarver.
!
! \f
! /aboutyour/ s//about your/
! q

! send_mail:  send
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>Chief>Washington>Washington.mbx
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>State>Jefferson>Jefferson.mbx
  Mail delivered to TSmith.ProjA.
  send_mail (send): The message was sent to some but not all destinations.

! send_mail:  quit

  send_mail (quit): Message has not been sent, saved, or written.
! Do you still wish to quit?  yes
  r 11:10 1.545 363
```

Type "prm –ls" to see the messages you now have received, deleting any you may not want to keep by typing "yes" to the delete mail prompt. Observe that the latest mail, in which the text editor was invoked, is printed with all errors corrected.

```
     #3 (9 lines)  10/26/93 11:10  Mailed by: TSmith.ProjA
    Date: 26 October 1793 11:10 est
    From:  TSmith.ProjA
    Subject:  Silver dollar salvage
    To:  Washington.Chief
    cc:  Jefferson.State, TSmith.ProjA

    Prez:

    Is there danger British can salvage silver
    dollar from river to help finance future War of 1812, and
    can hatchet be manufactured for national armament stockpile?

    Sorry about your tooth   I know an excellent woodcarver.

         Henry

  ! print_mail: Delete#3?   no
    r 11:13 0.747 11
```

SEND_MAIL REQUESTS

When the send_mail request loop is entered (signified by a prompt of "send_mail:"), you have the opportunity to make further changes to your message and heading. Several of the requests are identical to the "sdm" command line control arguments but are given without the hyphen prefix.

You have already seen that the request **send** to the "send_mail:" prompt causes the mail to be delivered. You have also seen how the "quit" request to the "send_mail:" prompt returns you to command level. Requests such as cc, fill, from, and to (note the absence of the hyphens) perform identically to "sdm" command line control arguments –cc, –fill, –from, and –to, respectively. If you wish to remove an address from the header, the request is **remove**, which has the short name "rm", and, of course, must contain the User_id (the mail address) of the mail recipient.

Additional requests are **print**, with the short name "pr", which gives a print of the message and heading as it will be received, and the request **print_header**, with the short name "prhe", which lets you see only the header as it will be received at a mailbox.

Type the following message from Washington to Knox and observe how these "send_mail:" requests can be applied to alter your mail. Do not forget to type "\f" and then "q" instead of a period at the end of your input in order to get into the send_mail request loop. However, to repeat, editing can be done only with the qedx text editor and cannot be done with Ted.

```
! sdm Knos.War -no_abort -cc Randolph.Attgen TSmith.ProjA
! Subject: Hatchet and cherry pies
  Message:
! Hank:  I do not think the British can salvage the silver dollar,
! but I suggest that you look into the use of cherry pits as musket
! balls in the event this crisis worsens.The hatchet was a tomahawk
! I converted for use to blaze trails when I served as a
! surveyor.  By the way, I will need the services of your
! woodcarver.
!
! And Tommy,  I would like to know who will succeed me as president.
! Please send me the list astrologer Nostradamus made of my
! successors.  I think you should also send a copy of the Preamble
! of our Constitution to Randolph.
!
! Also Tom, I am sending a packet by earliest post.  Distribute
! its contents for me if you will.
!
!
! Thanks,
!
!
!          Georgie
!
! \f
! q

  send_mail:
```

qedx Request

The **qedx** request to the "send_mail:" prompt can be used to edit your message just as typing "\f" is used to invoke the text editor at the end of a message. However, neither request ("qedx" nor "\f") can be used to edit the heading information. But, adding a **-header** control argument, with the short name "-he", to the qedx request does permit you to, for example, change the subject line or correct a mailbox address. Type "qedx -he" ("qx -he" can be used) after the "send_mail:" prompt in order to edit your latest message and to correct the spelling of "Knos" and to change "cherry pies" to "successors" in the heading.

```
  q

! send_mail:  qedx -he
! /Knos/ s//Knox/
! /cherry pies/ s//successors/
! /sens.The/ s//sens.  The/
! /packet/ s//package/
! q

  send_mail:
```

To Request

The **to** request to a "send_mail:" prompt is used to add additional primary recipients to your message heading. Type "to Jefferson.State" following the "send_mail:" prompt to add Jefferson as a primary recipient with Knox.

```
  q

! send_mail:  to Jefferson.State

  send_mail:
```

cc Request

The **cc** request to a "send_mail:" prompt is used to add additional secondary recipients to your message. Type "cc Hamilton.Treas" to add Hamilton as a secondary recipient with yourself and Randolph.

```
  send_mail:  to Jefferson.State

! send_mail:  cc Hamilton.Treas

  send_mail:
```

Remove Request

The **remove** request, which can be given as "rm", is used to remove any recipients or other addresses given in the heading. It can also be used to delete the subject line. However, assume you have Knox typed in as both a primary recipient and as a secondary recipient and you wish him to be only a primary recipient. Typing "rm Knox.War" will remove Knox throughout the heading. If a control argument is included in the request line, however, then only the addressee following the control argument will be removed. That is, typing "rm –cc Knox.War" will remove Knox as a secondary recipient but keep him as a primary recipient. However, if you were to type "rm –to Knox.War", then Knox would be removed as a primary recipient and remain as a secondary recipient. Type "rm Randolph.Attgen" to remove Randolph from your message heading.

```
  send_mail:  cc Hamilton.Treas

! send_mail:  rm Randolph.Attgen

  send_mail:
```

From Request

The **from** request is used to change the name of the author of the message, but not the name of the sender. By default, the system uses the sender as the author of the message. The from request (or –from control argument in the "sdm" command line) is used to overwrite the sender's name as the author. Type "from Washington.Chief" as the author of this message.

```
  send_mail:  rm Randolph.Attgen

! send_mail:  from Washington.Chief

  send_mail:
```

Print Header Request

The **print_header** request with the short name "prhe" is used after a "send_mail:" prompt to obtain a print of the header of your message as it will be received by the recipients. Type "prhe" and observe that the changes made by the previous requests have been incorporated. Note especially that the "Sender:" line does not get changed.

```
  send_mail:  from Washington.Chief

! send_mail: prhe

  (22 lines in text):
  Date:  30 October 1793 08:28 est
  From:  Washington.Chief
  Subject:  Hatchet and successors
  Sender:  TSmith.ProjA
  To:  Knox.War, Jefferson.State
  cc:  TSmith.ProjA, Hamilton.Treas

  send_mail:
```

Fill Request

The **fill** request is used in exactly the same way the –fill control argument is used in the "sdm" command line. This request aligns the left margin of the message and fills in the lines to give a ragged right margin. If a **–line_length N** control argument is added to this request, the line length is changed to "N" characters from a default value of 72. Now type "fill –1 50".

```
            cc:  TSmith.ProjA, Hamilton.Treas

          ! send_mail:  fill -ll 50

            send_mail:
```

Print Request

The **print** request provides a print of your message as it will be sent with a short heading. Adding a **–header** or "–he" control argument to this request will show you how both the message and header will be received by the recipients. Type "pr".

```
  send_mail:  fill -ll 50

! send_mail:  pr

  (24 lines in text):
  Subject:  Hatchet and successors
  To:  Knox.War, Jefferson.State
  cc:  TSmith.ProjA, Hamilton.Treas

  Hank:  I do not think the British can salvage the
  silver dollar, but I suggest that you look into
  the use of cherry pits as musket balls in the
  event this crisis worsens.  The hatchet was a
  tomahawk I converted for use to blaze trails when
  I served as a surveyor.  By the way, I will need
  the services of your woodcarver.

  And Tommy,  I would like to know who will succeed
  me as president.  Please send me the list
  astrologer Nostradamus made of my successors.  I
  think you should also send a copy of the Preamble
  of our Constitution to Randolph.

  Also Tom, I am sending a package by earliest post.
  Distribute its contents for me if you will.


  Thanks,


        Georgie


  send_mail:
```

Send and Quit Requests

As you have seen, the **send** request to the "send_mail:" prompt causes the mail to be delivered. The **quit** request returns you to command level. Send the mail and return to command level.

```
        Georgie

 ! send_mail:  send
   send_mail (send): Some directory in path specified does not exist.
   \c>udd>War>Knox>Knox.mbx
   send_mail (send): Some directory in path specified does not exist.
   \c>udd>State>Jefferson>Jefferson.mbx
   send_mail (send): Some directory in path specified does not exist.
   \c>udd>Treas>Hamilton>Hamilton.mbx
   Mail delivered to TSmith.ProjA.
   send_mail (send): The message was sent to some but not all destinations.

 ! send_mail:  quit

   send_mail (quit): Message has not been sent, saved, or written.
 ! Do you still wish to quit?  yes
   r 08:32 1.990 553
```

NOTE

Had you made a mailbox correction instead of typing "quit" (followed by "send" and then "quit") with a -no_abort control argument, you would receive a duplicate of this message.

SENDING FILES

Although the electronic mail system can transmit quite lengthy messages, it is courteous to keep the messages brief. Existing files (segments) can also be transmitted, but again, those files should be relatively short. Large files should not be sent using the electronic mail system, although the system is capable of transmitting them.

You have in permanent storage the two files referred to in the previous message from Washington to Knox and Jefferson. The files you have, however, are compin files, "presidents.compin" and "Preamble.compin". You can, of course, send these files by electronic mail but all control lines the compose command uses in formatting the file will be sent, too. These **compin** files, therefore, should be first converted to **compout** files; then they can be sent in formatted form.

To obtain a compout file, you first must give the compose command to format the compin file. You must then also give a new command, by means of an **-output_file** control argument, with the short name "-of", to output that file into permanent storage so that you can retrieve it. Furthermore, an additional control argument, the **-galley** control argument, with the short name "-gl", should be used to make the file a single page, regardless of its actual length; this control argument will also eliminate page marker symbols used by the system that would otherwise appear but are not wanted. Thus, format the "presidents.compin" and "Preamble.compin" files and output those formatted files into permanent storage as follows:

```
  r 08:37 3.245 609

! compose presidents -of -gl
  r 08:38 2.285 150

! compose Preamble -of -gl
  r 08:38 1.513 52
```

You now have two additional files in permanent storage: "presidents.compout" and "Preamble.compout". If you give the list command, you will see these two files listed in your directory.

Two examples of sending existing files by electronic mail are shown here. The first involves only the -input_file control argument in the "sdm" command line. The second includes a message with the existing file.

In the first example, there is no message to be included. The "Preamble.compout" file is sent by using the -if control argument in the "sdm" command line. You need to input on this command line only the subject, "quit", and "yes".

```
! sdm Randolph.Attgen -if Preamble.compout -no_abort -cc TSmith.ProjA
! Subject: Preamble
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>Attgen>Randolph>Randolph.mbx
  Mail delivered to TSmith.ProjA.
  send_mail (send): The message was sent to some but not all destinations.

! send_mail:  quit

  send_mail (quit): Message has not been sent, saved, or written.
! Do you still wish to quit?  yes
  r 08:49 0.339 94
```

In the second example, Jefferson is to send your "presidents.compout" file to Washington with a message. Thus, input the message as shown ending the message with the message terminator "\f", followed by "q" to put you in the "send_mail:" request loop. Invoke qedx to edit your message and read in your "presidents.compout" file. In the example, the file will be included at the end of the message after the signature, "Thomas". Send the mail and return to command level.

```
! sdm Washington.Chief -from Jefferson.State -no_abort -cc TSmith.ProjA
! Subject: List of successors and package
  Message:
! George.
!
! List of successors put together by Nostradamus is shown
! below.  Gave hatchet to Henry along with your broken
! dentures for his woodcarver to fix.  Sent silver dollar
! to Hamilton and will send Preamble to Randolph.  Henry says
! our cherry pit musket ball factory doing great.
!
! Thanks for the cherry pie.
!
!      Thomas
! \f
! q


! send_mail:  qedx
! r presidents.compout
! q

! send_mail:  send
  send_mail (send): Some directory in path specified does not exist.
  \c>udd>Chief>Washington>Washington.mbx
  Mail delivered to TSmith.ProjA.
  send_mail (send): The message was sent to some but not all destinations.

! send_mail:  quit

  send_mail (quit): Message has not been sent, saved, or written.
! Do you still wish to quit?  yes
  r 08:45 0.789 223
```

This mail sent by Jefferson will be received by Washington and Randolph respectively as shown below.

```
 #4 (31 lines) 10/30/93 08:45 Mailed by:  TSmith.ProjA
Date:  30 October 1793 08:45 est
From:  Jefferson.State
Subject:  List of successors and package
Sender:  TSmith.ProjA
To:  Washington.Chief
cc:  TSmith.ProjA

George,

List of successors put together by Nostradamus is shown
below.  Gave hatchet to Henry along with your broken
dentures for his woodcarver to fix.  Sent silver dollar
to Hamilton and will send Preamble to Randolph.  Henry says
our cherry pit musket ball factory doing great.

Thanks for the cherry pie.

     Thomas

                          PRESIDENTS


No.  President           State Born  Term        Died  Wife


1    Washington  George   VA   1732  1789-1797   1799  Martha

2    Adams       John     MA   1735  1797-1801   1826  Abigail

3    Jefferson   Thomas   VA   1743  1801-1809   1826  Martha

4    Madison     James    VA   1751  1809-1817   1836  Dorothea

5    Monroe      James    VA   1758  1817-1825   1831  Elizabeth

First 5 presidents, their wives, dates.

print_mail: Delete #4?
```

```
 #3 (14 lines)  10/30/93 08:48 Mailed by: TSmith.ProjA
Date:  30 October 1793 08:48 est
From:  TSmith.ProjA
Subject:  Preamble
To:  Randolph.Attgen
cc:  TSmith.ProjA

                          CONSTITUTION
                      OF THE UNITED STATES
                           OF AMERICA

                            PREAMBLE

                We the people of  the United States, in Order
              to form  a more perfect Union,  establish Justice,
              insure  domestic  Tranquility,  provide  for  the
              common defense,  promote the general  Welfare, and
              secure the  Blessings of Liberty to  ourselves and
              our  Posterity,  do  ordain  and  establish  this
              Constitution for the United States of America.

print_mail: Delete #3?
```

MESSAGE FACILITY

The message facility is used to send short, relatively urgent messages that may be read immediately or to permit an interactive conversation between users at different terminals. The message facility might be compared to a telephone system whereas the mail facility might be compared to a postal system.

The message facility, as part of the electronic mail facility, differs from the mail facility by permitting online communication between users on different data terminals. The message facility is similar to the mail facility in that if the user who is to receive the message is not logged in (or has commanded "defer_messages", to be described), the message is stored in a mailbox.

To use the message facility, you need use only four commands: **send_message, accept_messages, defer_messages,** and **print_messages.**

Send Message

The **send_message** command, with the short name "sm" sends the message to the recipient. If the recipient is logged in and is accepting messages (by command to the system), the message is immediately printed at that user's terminal.

Sending the message may be done two ways. In the first method, the command is given with the recipient's User_id and the message typed on the same line. The message is sent when the carriage return is pressed. If the recipient is not receiving messages at that time, the sender is notified and the message is placed in the recipient's mailbox. For example, if Jefferson is not logged in and Washington sent him a message, the result could be as follows:

```
 r 15:41 0.092 9

! sm Jefferson.State  Thanks for sending the list of presidents.
 send_message: User not accepting messages or not logged in. Jefferson.State
 r 15:42 0.386 35
```

On the other hand, if Jefferson is logged in, Washington will not receive acknowledgement, only a ready signal for the next Multics command.

```
! sm Jefferson.State  Thanks for the list of presidents.
 r 15:42 0.084 39
```

If the recipient does not have a mailbox, the system also tells the sender.

```
! sm Jefferson.State  Thank you for the Nostradamus list.
 send_message: No mailbox for Jefferson.State
 r 15:43 0.055 23
```

In this example, the carriage return denotes the end of the message. Any other input you may try to type on a succeeding line will be interpreted as another command to the Multics system and probably will not be recognized.

Send yourself a message and note that you are not receiving or accepting messages at this time.

```
  r 07:44 0.207 16

! sm TSmith.ProjA  Thanks for sending the list of presidents.
  send_message: User not accepting messages or not logged in.  TSmith.ProjA
  r 07:45 0.880 44
```

Suppose Jefferson is logged in and is accepting messages (by having given an accept_messages command described later). Suppose further that Jefferson has invoked his text editor and has requested "1,$p" for a "Lincoln.compin" segment. Washington's message will interrupt the print with a result that may appear as follows on Jefferson's terminal:

```
so conceived and so dedicated, can long endure.
.spb
.unl -5
We are met oFrom Washington.Chief 10/31/93  15:42.3 est Thu:  Thanks fo
\cr the list of presidents.
n a great battle-field of that war.
We have come to
```

The second form of message ending is often referred to as a "conversation mode". This form lets you send messages while, at the same time, it lets you receive messages, thereby forming a conversation between users.

In the second form of message sending, the message is not typed on the command line. Instead, a carriage return after the User_id of the recipient will result in a system response of "Input:" after which you would then type your message. Now you are allowed to input more than one line of message, but each line of input is considered a separate message to the recipient. The end of the message is signified by a period alone on a line. If the recipient is not logged in or does not have a mailbox, you will be given the same message by the system as shown above for a message on the command line. Thus, a second message from Washington to Jefferson could appear as follows if Jefferson is logged in and accepting messages:

```
          ! sm Jefferson.State
            Input:
          ! Tom, sorry I forgot to thank you for distributing
          ! the package for me.  Have not heard about my wooden
          ! teeth.  Glad you liked the pie.
          ! .
            r 12:51 0.308 94
```

If Jefferson is logged in and again has invoked his editor and requested "1,$p" for the "Lincoln.compin" file, this message will be received as three separate messages, distributed throughout his print and separated by the amount of time it takes to type the message. That is, Jefferson might receive the message as follows:

```
  the proposition that all men are created equal.  Now
  we are engaged in a gFrom Washington.Chief 11/01/93  1303.8 est Thu:
  Tom, sorry I forgot to thank you for distributing
  reat civil war.

                         .
                         .
                         .

  so conceived and so dedicated, can long endure.
 .spFrom Washington.Chief 11/01/93  1304.4 est Thu:
  the package for me.  Have not heard about my wooden
  b
  .unl -5

                         .
                         .
                         .

  shall not perish from the earth.
  .bFrom Washington.Chief 11/01/93  1304.6 est Thu:   teeth.   Glad you l
  \ciked the pie.
  bf s
  Lincoln died in April, 1865.
```

Send this same message to yourself.

```
! sm TSmith.ProjA
  send_message: User not accepting messages or not logged in.  TSmith.ProjA
  Input:
! Tom, sorry I forgot to thank you for distributing
! the package for me.  Have not heard about my wooden
! teeth.  Glad you liked the pie.
! .
  r 14:31 0.273 107
```

Give the print_mail -list command and observe the amount of mail in your mailbox. If Jefferson had not deleted any mail sent to him and had not accepted messages, his mailbox would appear with a total of six messages. The last message from Washington, consisting of three lines, is considered as three separate messages. Jefferson's mailbox (if he had not been logged in or was not accepting messages) would be as shown next. Do not delete your messages at this time.

```
! prm -ls
  You have 6 messages.

  Msg#  Lines   Date      Time   From                    Subject
     1    (9)   10/26/93  11:10  Knox.War                Silver dollar salvage
     2   (24)   10/30/93  08:32  Washington.Chief        Hatchet and successors
     3    (1)   11/01/93  14:25  Washington              Thanks for sending<MORE>
     4    (1)   11/01/93  14:31  Washington              Tom, sorry I forgo<MORE>
     5    (1)   11/01/93  14:31  Washington              the package for me<MORE>
     6    (1)   11/01/93  14:31  Washington              teeth.  Glad you l<MORE>

   #1 (9 lines)  10/26/93 11:10  Mailed by: Knox.War
                        .
                        .
                        .
! print_mail: Delete #1?   no
                        .
                        .
                        .
! print_mail: Delete #2?   no

   #3 (1 line)  11/01/93 07:45  Mailed by: Washington(Washington.Chief)
  Thanks for sending the list of presidents.

! print_mail: Delete #3?   no

   #4 (1 line)  11/01/93  14/31  Mailed by: Washington (Washington.Chief)
  Tom, sorry I forgot to thank you for distributing

! print_mail: Delete #4?   no

   #5 (1 line)  11/01/93 14:31  Mailed by: Washington (Washington.Chief)
  the package for me.  Have not heard about my wooden

! print_mail: Delete #5?   no

   #6 (1 line)  11/01/93 14:31  Mailed by: Washington (Washington.Chief)
  teeth.  Glad you liked the pie.

! print_mail: Delete #6?   no
  r 14:34 0.645 160
```

Accept Messages

If you are logged in and want to accept messages any time some other user sends a message to you, you need to command the system accordingly. That command is **accept_messages** with the short name "am". When you have given this command, the message sent to you will be printed immediately on your terminal as in the case of the Jefferson/text editor examples discussed earlier. In addition, if you are logged in and accepting messages, that message does not get sent to your mailbox.

Type "am" to accept messages. Then, send yourself a message and note that it gets printed immediately. Thirdly, give the print_mail -list command and note that because this message to yourself has now been printed, it does not appear in the mail listing. (Use the interrupt key, **BREAK, BRK,** etc. and then "rl" if you do not want to repeat a print of all your mail.) If Jefferson has not accepted messages as discussed earlier and now gave the accept_messages command and sent himself a message, his print would look as follows:

```
  am
  r 14:34 0.209 25

! sm Jefferson.State Remember to see about teeth.
  r 14:34 0.084 42

  From Jefferson.State 11/01/93  1434.9 est Thu: Remember to see about teeth.
! prm -ls
  You have 6 messages.

  Msg#  Lines   Date      Time   From                Subject
     1    (9)   10/26/93  11:10  Knox.War            Silver dollar salvage
     2   (24)   10/30/93  08:32  Washington.Chief    Hatchet and successors
     3    (1)   11/01/93  14:25  Washington          Thanks for sending<MORE>
     4    (1)   11/01/93  14:31  Washington          Tom, sorry I forgo<MORE>
     5    (1)   11/01/93  14:31  Washington          the package for me<MORE>
     6    (1)   11/01/93  14:31  Washington          teeth.  Glad you l<MORE>

   #1 (9 lines)  10/26/93  11:10  Mailed by:  Knox.War
! Date:  26 October 1793
  QUIT
  r 14:35 0.539 74 level 2

! rl
  r 14:35 0.621 53
```

Note that if you do not have a mailbox, the first time you give an accept_messages command, you will have a mailbox created for you just as if you had given a print_mail command. If you already have a mailbox, you do not get a second one.

Defer Messages

Two of the previous Jefferson examples had him accepting messages while he was using a text editor. Suppose he had just given a compose command to format a file. He probably would not want an incoming message to be printed within this formatted output.

```
                                 CONSTITUTION
                             OF THE UNITED STATES
                                  OF AMERICA

                                   PREAMBLE

                      We the people of  the United States, in Order
                  to form  a more perfect Union,  establish Justice,
                  insure  domestic  Tranquility,  provide  for  the
                  cFrom Washington.Chief 11/02/93  0814.6 est Fri:  Where are
\c my teeth?
ommon defense,     promote the general Welfare, and
                  secure the  Blessings of Liberty to  ourselves and
                  our Posterity,  do ordain and establish  this Con-
                  stitution for the United States of America.
```

If the accept_messages command has been given, but you do not want to have your work interrupted by a message, you can give a **defer_messages** command, with the short name "dm". Now, any incoming messages will be placed in your mailbox until you again give the accept_messages command. (This second accept_messages command, however, does not cause a print of those messages received and placed in your mailbox, while you were deferring messages.)

Print Messages

You can any time you are logged in, and after a ready message, give a **print_messages** command, with the short name "pm". This command causes all messages in your mailbox to be printed without disturbing other mail in your mailbox. However, once this print_messages command is given and the messages printed, those messages are deleted from the mailbox. You do not need to answer "yes" or "no" as you would with a print_mail command; you are not even asked for a "yes" or "no" answer.

Type "pm" to receive a print of the messages presently in your mailbox. Observe that four lines are printed (unless you have received messages other than the two examples you sent youself in this discussion). Jefferson's mailbox contains the following:

```
! pm
  From Washington.Chief 11/01/93  1425.9 est Thu:
  Thanks for sending the list of presidents.
  From Washington.Chief 1431.0:
  Tom, sorry I forgot to thank you for distributing
  =: the package for me.  Have not heard about my wooden
  =: teeth.  Glad you liked the pie.
  r 12:05 0.303 9
```

Repeat the print_messages command. You will find that your only response to the command is the ready message. Messages in your mailbox have been deleted. (You may want to verify that the messages have been deleted by giving the print_mail command.)

```
! pm
  r 12:06 0.038 2
```

The print_messages command has one control argument that may be used. This control argument is the **–last** control argument, with the short name "–lt". When the print_messages command is given, all messages are deleted from the mailbox, as you have seen. However, the last message (last message line) is held in storage. The print_messages command with the –last control argument can be given to recall this message (especially useful if the message seems to be garbled). Type "pm –lt" and observe that your last message (message line) is printed.

```
! pm -lt
  From Washington.Chief 11/01/93  1431.6 est Thu:
  teeth.  Glad you liked the pie.
  r 12:06 0.070 4
```

TERMINAL-TO-TERMINAL CONVERSATION

Terminal-to-terminal conversation is available using the message facility because messages can be received and printed immediately. Conversation is based on having two users, at two separate terminals, each accepting messages. Also, if the second send_message method of transmission using "Input:" and the period for message termination are used, conversation can continue without having to give the send_message command each time information is to be sent.

NOTE

Once you have gone into the send_ message conversation mode, you will remain there until the period, alone on a line, is typed.

A conversation between Jefferson and Knox might be as follows, where Jefferson's messages are shown first. Observe that after the initial reply, the messages from Knox are preceded by "=:". Using the **–short** control argument gives this symbol and replaces the heading for each message line.

```
! am -short
  r 13:43 0.078 55

! sm Knox.War
  Input:
! Henry, George wants to know about his teeth.
  From Knox.War 11/02/93  1344.6 est Fri:
  I have given his teeth to my woodcarver.
! When do you hope to get them back repaired?
  =: Sometimes next week.
! OK, I'll tell George.  Bye.
  =: Fine.  I'll see you.
! .
  r 13:47 0.795 202
```

The print at Knox's terminal will appear almost the same with the exception of "=:", but with heading information for each Jefferson transmission.

```
  am
  r 13:42 0.755 54

  From Jefferson.State 11/02/93  1342.7 est Fri:
  Henry, George wants to know about his teeth.
! sm Jefferson.State
  Input:
! I have given his teeth to my woodcarver.
  From Jefferson.State 11/02/93  1345.2 est Fri:
  When do you hope to get them back repaired?
! Sometime next week.
  From Jefferson.State 11/02/93  1346.4 est Fri:  OK, I'll tell George.  Bye.
! Fine  I'll see you.
! .

  r 13.47 0.889 191
```

Appendix A
Summary of qedx or Ted Requests

INPUT REQUESTS

All input requests are terminated by "\f".

a (append)
The append request is used to enter input lines from the terminal to create a new segment, or to append input lines *after* the line addressed by the request. The request "0a" can be used to append text before line 1 of the existing buffer contents.

c (change)
The change request is used to enter input lines from the terminal and to input new text *in place of* the addressed line or lines.

i (insert)
The insert request is used to enter input lines from the terminal and to insert new text *before* the addressed line.

EDIT REQUESTS

d (delete)
The delete request is used to delete the addressed line or lines from the contents of the buffer.

/xxxxx/ (locate)
The locate request sets the value of "." to a specific line and prints the line. This request needs no letter to tell the editor what operation to perform; the user merely types a valid address followed by a newline character (carriage return). In context addressing, a search is done through the file starting with the first line after the current line to the last line in the file, and then from line 1 to the current line. In context addressing the request is given as "/xxxxx/".

p (print)
The print request is used to print the addressed line or set of lines on the user's terminal.

= (print line number)
The print line number request is used to print the absolute line number of an addressed line.

q (quit)
The quit request is used to exit from the text editor. This request does not itself save the results of any editing that might have been done. If the user wishes to save the modified contents of the buffer, an explicit write request must be issued.

r pathname (read)
The read request is used to put the contents of an already existing segment into the buffer. This request appends the contents of a specified segment after the addressed line. The request "0r pathname" is used to place the contents of a segment *before* line 1 of the buffer.

s (substitute)
The substitute request is used to modify the contents of the addressed line or set of lines by replacing all strings that match a given regular expression with a specified character string. The request has the form of "s/old/new/" where "/" is a delimiter chosen by the user and must not be in either the "old" or "new" parts of the request line.

w pathname (write)
The write request is used to write an addressed line or set of lines into a specified segment.

Appendix B Summary of the Format_Document Command Control Lines and Control Arguments

FORMAT_DOCUMENT CONTROL LINES

The format_document command assumes the following control lines (with their default values) are at the beginning of each fdocin segment:

```
.alb
.fin
.in 0
.pdl 66
.pdw 65
```

In addition, the fdoc command provides for a top margin of six lines and a bottom margin of six lines.

The following are the fdoc control lines used to format relatively short, simple documents (such as letters) that do not require the more complex formatting features of the WORDPRO Text Formatter (compose).

.alb (align both left and right margins)

.all (align left margin only)

.fif (fill off)
Text is formatted exactly as input.

.fin (fill on)
Text is formatted to fill each line.

.in ±N (indent left margin)
N without the sign is the absolute number of spaces the left margin is indented toward the center of the page. +N indents toward the center of the page N spaces relative to the current left margin. –N indents toward the left margin N spaces relative to the current margin. (By default, N is 0.)

.pdl N (page length)
N is the number of lines per page. (By default, N is 66.)

.pdw (page width)
N is the number of columns (characters) per line. (By default, N is 65.)

.un ±N (undent left margin)
Either N without the sign or +N is the number of spaces the following line of output text is undented to the left from the current left indent. –N is the number of spaces the following line of text is undented toward the center of the page from the current left indent.

FDOC COMMAND CONTROL ARGUMENTS

All format_document command control arguments are given on the fdoc command line and are preceded by a hyphen (–) and separated by a space.

–ind N (–idnent N)
The –indent control argument adds N spaces at the left margin of the formatted output (in addition to the left indent control lines within the fdocin segment) to shift the entire document N spaces to the right.

–of path (–output_file path)
The –output_file control argument directs the output to a file named "path" instead of to the user's terminal.

–pgno (page_numbers)
The –page_numbers control argument ends each page with two blank lines and a centered page number beginning with "1" and numbering consecutively.

Appendix C Summary of the Compose Command Control Lines and Control Arguments

COMPOSE COMMAND CONTROL LINES

The compose command assumes the following control lines (with their default values) are at the beginning of each compin segment:

```
.alb
.fin
.inl 0
.pdl 66
.pdw 65
.vmb 4
.vmf 2
.vmh 2
.vmt 4
.wit 2
```

The following are the most often used compose command control lines. The listing given does not contain all compose command control lines. Nevertheless, all those described in this manual are given, as well as some less often used control lines allied to those described.

.alb (align both left and right margins)

.alc (align center; center between margins)

.alr (align right margin only)

.all (align left margin only)

.bbf (block begin footnote; used with .bef)

Example:

```
.bbf
footnote
block
.bef
```

.bbk (block begin keep; used with .bek)

Example:

```
.bbk
text block not to
be split between
pages
.bek
```

.bbt (block begin title; used with .bet)

Example:

```
.bbt
||centered equation, note, etc.||
||equation, etc. continued||
.bet
```

Note that .bbt and .bet can be replaced by .bbt N (block begin title, N lines, where N is the number of known lines in the block).

.bef (block end footnote; used with .bbf)

.bek (block end keep; used with .bbk)

.bet (block end title; used with .bbt)

.brf (break format)
Interrupts text formatting and resumes with a new line of the current block.

.brp N (break page)
Interrupts text formatting on one page and resumes with a new line on a new page. If N is given (that is, the page number for the new page), the page number counter is set to N and the new page is assigned that number. If N is not given, the page number for the new page is the next sequential page number. A mode_string may be added as follows: rl (lowercase Roman, ru (uppercase Roman), ar (Arabic), al (lowercase alphabetic), or au (uppercase alphabetic).

.fif (fill off)
Text is formatted exactly as input.

.fin (fill on)
Text is formatted to fill each line.

.fla (footer line all pages)
   Example:
   .fla N M |left|center|right|
N is the number of the footer line counted up from the bottom of the page and by default is 1. M is the column number for the left alignment of the footer line and by default is 0. A blank footer line on all pages would be given as ".fla ||||". To cancel the footer line, use ".fla".

.fle (footer line even pages)
   Example:
   .fle N M |left|center|right|
N is the number of the footer line counted up from the bottom of the page and by default is 1. M is the column number for the left alignment of the footer line and by default is 0. A blank footer line on even pages would be given as ".fle ||||". To cancel the footer line, use ".fle".

.flo (footer line odd pages)
   Example:
   .flo N M |left|center|right|
N is the number of the footer line counted up from the bottom of the page and by default is 1. M is the column number for the left alignment of the footer line and by default is 0. A blank footer line on odd pages would be given as ".flo ||||". To cancel the footer line, use ".flo".

.ftp (footnote page)
Footnotes on each page are numbered consecutively from 1. To insert an unreferenced footnote when this control is in effect, use ".bbf u" for the footnote block instead of ".bbf".

.ftu (footnote unreferenced)
Footnotes are not numbered.

.hla (header line all pages)
Example:
.hla N M |left|center|right|
N is the number of the header line counted down from the top of the page and by default is 1. M is the column number for the left alignment of the header line and by default is 0. A blank header line on all pages would be given as ".hla ||||". To cancel the header line, use ".hla".

.hle (header line even pages)
Example:
.hle N M |left|center|right|
N is the number of the header line counted down from the top of the page and by default is 1. M is the column number for the left alignment of the header line and by default is 0. A blank header line on even pages would be given as ".hle ||||". To cancel the header line, use ".hle".

.hlo (header line odd pages)
Example:
.hlo N M |left|center|right|
N is the number of the header line counted down from the top of the page and by default is 1. M is the column number for the left alignment of the header line and by default is 0. A blank header line on odd pages would be given as ".hlo ||||". To cancel the header line, use ".hlo".

.htd (horizontal tab define; used with .htn and .htf)
Example:
.htd pattern_name q,r,s,t
.htn C pattern_name
Pattern block with tab C stops where
q,r,s,t are tab stops referenced to left
indent and C is the tab character
.htf C

.htf (horizontal tab off)

.htn (horizontal tab on)

.inb ±N (indent both margins)
N without the sign is the absolute number of spaces the left margin is indented toward the center of the page and the number of spaces the right margin is indented toward the center of the page. +N indents towards the center of the page N spaces relative to the current indent. −N indents toward the margins N spaces relative to the current margins. (By default, N is 0.)

.inl ±N (indent left margin)
N without the sign is the absolute number of spaces the left margin is indented toward the center of the page. +N indents toward the center of the page N spaces relative to the current left margin. −N indents toward the left margin N spaces relative to the current margin. (By default, N is 0.)

.inr ±N (indent right margin)
N without the sign is the absolute number of spaces the right margin is indented toward the center of the page. +N indents toward the center of the page N spaces relative to the current right margin. −N indents toward the right margin N spaces relative to the current right margin. (By default, N is 0.)

.pdl N (page definition length)
N is the number of lines per page. (By default, N is 66.)

.pdw N (page definition width)
N is the number of columns (characters) per line. (By default, N is 65.)

.spb N (space block)
N is the number of blank lines between text blocks. (By default, N is 1.)

.spf N (space format)
N is the number of blank lines within a text block. (By default, N is 1.)

.tlc (title line caption)
Example:
.tlc N M |left|center|right|
N is the number of blank lines preceding the caption, and M is the column number for the left alignment of the title line. (By default, N and M are 0.)

.tlh (title line header)
Example:
.tlhN M|left|center|right|
N is the number of blank lines following the header, and M is the column number for the left alignment of the title line. (By default, N and M are 0.)

.trn (translate format)
Example:
.trn abcd
Every occurrence of "a" in the input file is replaced by "b" on output and every occurrence of "c" in the input file is replaced by "d" on output.

.unl ±N (undent left margin)
Either N without the sign or +N is the number of spaces the following line of output text is undented to the left from the current left indent. −N is the number of spaces the following line of text is undented toward the center of the page from the current left indent.

.unr ±N (undent right margin)
Either N without the sign or +N is the number of spaces the following line of output text is undented to the right from the current right margin. −N is the number of spaces the following line of text is undented toward the center of the page from the current right undent.

.ur (use reference)

Examples:

.ur %[long_date]%

gives current date in long form (July 4, 1776)

ur%[underline this expression]%

gives <u>this</u> <u>expression</u>

.ur%[underline "this expression"]%

gives <u>this expression</u>

.vmb N (vertical margin bottom)
N is the number of lines from the bottom of the page up to the lowest footer line. (By default, N is 4.)

.vmf N (vertical margin footer)
N is the number of lines from the last text line to the first footer line. (By default, N is 2.)

.vmh N (vertical margin header)
N is the number of lines from the last header line to the first text line. (By default, N is 2.)

.vmt N (vertical margin top)
N is the number of lines from the top of the page to the first header line. (By default, N is 4.)

.wit N (widow text)
N is the miminum number of lines to be left at the bottom or moved to the top of a page when text is split between pages. (By default, N is 2.)

The following are special symbols and page numbering controls used by the compose command:

%Date%

gives the current date in the form mm/dd/yy (07/04/76)

%PageNo%

gives the number of the current page

Example:

.fla 1 0 ||%PageNo%|%Date%|

COMPOSE COMMAND CONTROL ARGUMENTS

All compose command control arguments are given on the compose command line and are preceded by a hyphen (–) and separated by a space.

–fm N (–from N)
This control argument starts the formatted output at page N. The default value is page 1.

–gl X,Y (–galley X,Y)
This control argument produces galley format output (one long, continuous page) without running headers or footers for lines X through Y of the input file. By default, X is line 1 and Y is the last line of the input segment.

–hyph (–hyphenate)
This control argument causes the compose command to search through the dictionaries to hyphenate words to be used to fill a line.

–ind N (–indent N)
The –indent control argument adds N spaces at the left margin of the formatted output (in addition to the left indent control lines within the compin segment) to shift the entire document N spaces to the right.

–ls N (–linespace N)
The –linespace control argument changes the default line spacing value from 1 to N.

–nb (–number)
This control argument gives the absolute line number of the corresponding input compin segment to the left of the formatted output.

–of path (–output_file path)
The –output_file control argument directs the output to a file named "path" instead of to the user's terminal.

–pgs N,M (–pages N,M)
The –pages control argument permits calling for the output of page N, or pages N through M.

–sp (–stop)
This control argument causes the compose command to wait for a carriage return from the user before beginning the first page of formatted output and after each page of output (useful for special forms).

–to N (–to N)
This control argument ends the formatted output after page N.

Appendix D
Helpful Hints for New WORDPRO Users

1. You cannot issue a command unless you are at command level. The command level is signified by the Multics ready message.
2. "qedx", "ted", "fdoc", and "compose" are commands. Text editor requests (a, i, c, r, p, w, s, etc.) are understood by the text editor only.
3. To create a new file, invoke the text editor, issue an append request, type in the text, terminate the input mode, write the file, edit the file (if necessary), overwrite the file, quit the editor, and return to command level.

```
qx (or ted)
a
type text line 1
type text line 2
.
.
.
type text last line
\f
w pathname.compin (or w pathname.fdocin)
/locate 1/ d
/locate 2/ s/old/new/
.
.
.
w
q
```

4. Remember to use the escape sequence "\f" to terminate input. After issuing an input request, all lines until "\f" are considered input by the editor, including write and all other edit requests.

5. To edit an existing file, invoke the editor, read the existing file, issue the necessary edit requests, input any new material using the append, change, or insert requests (and terminate the input mode), write the file, quit the editor, and return to command level.

```
qx (or ted)
r pathname.compin (or r pathname.fdocin)
/locate a/ s/error/correct/
/locate b/ c
input new line b1
input new line b2
\f
/mistake/ s//no mistake/
.
.
.
w
q
```

6. You may, on occasion, unknowingly put the editor in an input mode by mistyping an edit request. If the editor does not respond to editing requests, such as a "p" request, chances are that the editor is in an input mode. You should then type "\f", print the current line and preceding lines to see if they need to be deleted, and then continue editing.

7. If you have a large amount of editing or inputting to do, it is wise to occasionally issue the write ("w") request to ensure that all work up to that time is permanently recorded. In that way, you will lose only the work done since the last write request should there be a problem with the system, terminal, or telephone line.

8. You must include the "compin" (of "fdocin") suffix in both the "r" and "w" requests when giving a file (path) name to the editor. You do not need to type the "compin" (or "fdocin") suffix when giving a file name to the compose (or fdoc) command.

9. When creating a compin file, the placement of certain control lines is important. In general, those controls that apply to the entire file should be given at the top of the file as housekeeping controls. If those control lines are not given, default values will be used on the text. Housekeeping control lines are:
   - page definition
     - .pdl N (By default, N is 66 lines.)
     - .pdw N (By default, N is 65 characters.)
   - vertical margin
     - .vmb N (By default, N is 4 lines.)
     - .vmf N (By default, N is 2 lines.)
     - .vmh N (By default, N is 2 lines.)
     - .vmt N (By default, N is 4 lines.)
   - alignment of text
     - .alb (By default, both margins are aligned.)

   Additional housekeeping includes:

   Header lines (if they are to appear on the first page, too) must be placed before any text line.

   Footer lines are best placed at the top of the file with the header lines.

   If the file is to contain footnotes, it is good practice to set up the numbering scheme (.ftp or .ftu) before any footnotes are encountered in the text input.

   Translate format character preparation is best done at the top of a file so that it can be used throughout without problems.

10. If you wish your dictionary to be used for hyphenation by the compose command, remember to give the add_search_paths command before you give the compose command.

11. To stop whatever the Multics system is doing, issue a QUIT signal by pressing the appropriate key (e.g., **BRK, INTERRUPT**). The Multics system will then print "QUIT" to let you know it has received the signal. The Multics system will hold the interrupted work and signify that is it holding the work by adding "level" information to the ready message. At this point, you have two choices:
    - Invoke the program_interrupt "pi" command to get back to the interrupted command and continue working. (The compose command does not support the program_interrupt command.)
    - Invoke the release "rl" command to release the latest work held. (The "release-all" command releases all of the work held.)

12. If qedx gives you an error message saying that the previous invocation will be lost if you proceed, and asks if you want to proceed, type "no". Wait for the ready message, and then type the "pi" command. You are now back in qedx and can continue your work. If you type "yes", you will be returned to command level and anything current in the buffer is destroyed.

Appendix E Speedtype Symbol Dictionary Start

The following symbols with their expansions are suggested as a beginning to your Speedtype symbols dictionary:

ab	about	o	of	wn	when
af	after	ot	out	wo	who
al	all	otr	other	wr	were
ay	any	ov	over	wrt	write
b	be	p	up	wa	was
bef	before	pl	please	wt	what
bn	been	q	question	x	and
bt	but	r	are	y	you
c	call	rd	read	yr	your
cd	could	s	is	z	zero
cn	can	sa	say		
d	do	sd	should		
e	the	sm	some		
f	for	sn	send		
fm	from	t	to		
ft	first	ta	than		
g	go	te	then		
gd	good	ti	time		
h	have	tir	their		
hd	had	tk	take		
hr	here	tm	them		
hs	has	tnk	think		
j	just	tr	there		
k	kept	ts	this		
kt	not	tt	that		
l	list	ty	they		
lt	last	u	use		
ma	may	v	very		
mk	make	w	with		
mt	must	wd	would		
n	in	wh	which		
no	into	wk	work		
nw	now	wl	will		

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	GUIDE TO MULTICS WORDPRO FOR NEW USERS	ORDER NO.	DJ18-01
		DATED	SEPTEMBER 1982

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

CUT ALONG LINE

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

Honeywell

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	GUIDE TO MULTICS WORDPRO FOR NEW USERS	ORDER NO.	DJ18-01
		DATED	SEPTEMBER 1982

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ____________________ DATE ____________

TITLE ____________________

COMPANY ____________________

ADDRESS ____________________

____

CUT ALONG LINE

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

Honeywell