MicroEMACS

Full Screen Text Editor
Reference Manual – *Release Version*

Version 4.0
March 20, 1996

Introduction

MicroEMACS is a tool for creating and changing documents, programs, and other text files. It is both relatively easy for the novice to use, but also very powerful in the hands of an expert. MicroEMACS can be extensively customized for the needs of the individual user.

MicroEMACS allows several files to be edited at the same time. The screen can be split into different windows and screens, and text may be moved freely from one window on any screen to the next. Depending on the type of file being edited, MicroEMACS can change how it behaves to make editing simple. Editing standard text files, program files and word processing documents are all possible at the same time.

There are extensive capabilities to make word processing and editing easier. These include commands for string searching and replacing, paragraph reformatting and deleting, automatic word wrapping, word move and deletes, easy case controlling, and automatic word counts.

For complex and repetitive editing tasks editing macroes can be written. These macroes allow the user a great degree of flexibility in determining how MicroEMACS behaves. Also, any and all the commands can be used by any keystroke by changing, or rebinding, what commands various keys invoke. A full multilevel Undo facility allows the user to recover from most editing errors with little effort.

Special features are also available to perform a diverse set of operations such as file encryption, automatic backup file generation, entabbing and detabbing lines, executing operating system commands and filtering of text through other programs (like SORT to allow sorting text).

History

EMACS was originally a text editor written by Richard Stallman at MIT in the early 1970s for Digital Equipment computers. Various versions, rewrites and clones have made an appearance since.

This version of MicroEMACS is derived from code written by Dave G. Conroy in 1985. Later modifications were performed by Steve Wilhite and George Jones. In December of 1985 Daniel Lawrence picked up the then current source (version 2.0) and made extensive modifications and additions to it over the course of the next eleven years. Updates and support for the current version are still available. Commercial support and usage and resale licences are also available. The current program author can be contacted by writing to:

USMAIL: Daniel Lawrence
617 New York St
Lafayette, IN 47901

Internet: dan@aquest.com

Support is provided through:

The Programmer’s Room
14,400 Baud
Account: uemacs
Password: support
(317) 742-5533 no parity 8 databits no stop bits

Credits

Many people have been involved in creating this software and we wish to credit some of them here. Dave Conroy, of course, wrote the very first version of MicroEMACS, and it is a credit to his clean coding that so much work was able to be done to expand it. John Gamble is responsible for writing the MAGIC mode search routines, and for maintaining all the search code. Jeff Lomicka wrote the appendix on DEC VMS and has supplied a lot of code to support VMS and the ATARI 1040ST versions. Curtis Smith wrote the original VMS code, helped supporting the Commodore AMIGA and did much development on various UNIX versions. Also Lance Jones has done a lot of work on the AMIGA code. Professor Suresh Konda at Carnegie Mellon University has put a lot of effort into writing complex macroes and finding all the bugs in the macro language before anyone else does.

A special thanks to Dana Hoggatt who has provided an almost daily sounding board for ideas, algorythms and code. He is responsible for the encryption code directly and has prodded me into adding many features with simple but poignant questions (Dan? How do we move the upper left corner of the screen? . . . which forced me to write the text windowing system).

Pierre Perrot dealt with my restrictive path to a generalized windowing version, and produced an excellent version for MicroSoft Windows. He continues to assist with this environment, forcing me to keep pace with him, making all the version more interesting.

As to people sending source code and text translations over computer networks like USENET and ARPA net, there are simply more than can be listed here. [The comments in the edit history in the history.c file mention each and the piece they contributed]. All these people should be thanked for the hard work they have put into MicroEMACS.

And after all this is the patients of the people waiting for my all too infrequent releases. A lot of work goes into each release, and as it is not my primary vocation, often much time passes between releases. We are committed to continue to improve and strenghten MicroEMACS with each new release into the indefinite future.

Daniel M. Lawrence

Chapter 1

Installation

MicroEMACS is a programmer's text editor which is very powerfull, customizable, and exists for a large number of different types of computer systems. It is particularly usefull for people who work on a lot of different computers and want to have a familiar and powerful editor which works identically no matter what computer they are using. But before using MicroEMACS, you must INSTALL it on your computer system. Since each computer is different, there is usually a different way to install MicroEMACS for each type of computer.

1.1 MSDOS, Windows 95, Windows NT, OS/2 – IBM-PCs

MicroEMACS is packaged for IBM-PC compatable machines as an installable package which will configure to the target machine. Instillation is very simple. Insert the distribution floppy in the target machine's A: drive. Then type, from the command prompt:

a:install

The install program will lead you through the intallation process, step by step, installing the appropriate versions on your computer's hard drive.

Under **Windows 95** or **Windows NT**, bring up the file manager, select the A: drive, and double click on the icon forthe install program.

1.2 UNIX

There are more available varieties of UNIX than any other currently popular operating system. MicroEMACS is provided in source to attempt to allow the compilation of a version appropriate to any UNIX system. This usually consists of a TAR or .zip file containing the tree of directories needed to build any version of MicroEMACS. Follow these steps to build an executable:

1. Copy the distribution file to an appropriate empty directory on the target system.

2. Unpack the distribution file, using tar or unzip or whatever appropriate unpacking program.

3. Change directory to the directory most closely approximating the target system's version of unix. These include:

AIX	IBM's propriatary UNIX varient
BSD	University of Berkeley's popular BSD varients
DGUX	Data General UNIX
FREEBSD	FreeBSD running on an IBM-PC
HPUX	Hewlett Packard's UNIX
SUN	Sun Microsystem's UNIX
XENIX	Microsoft UNIX

4. Copy ../h/estruct.h to the current directory.

5. Edit estruct.h to turn on the proper Operating system, Compiler and option switches by changing the value of their #defines from 0 to 1.

6. Issue a "make" command to build the executable.

7. Change to the BIN subdirectory.

8. copy the resulting executable, and the files of the CMD subdirectory from the root of the package into an appropriate directory on the UNIX system which will be in the user's path.

9. Rename the emacs.rc startup file to .emacsrc.

10. Make sure the copied executables have the proper permissions to allow them to be executed and read by the target users.

At this point, MicroEMACS should be ready for use. Remember to issue a REHASH command if you want to invoke it before logging out of the system immediately after building it.

Chapter 2

Basic Concepts

The current version of MicroEMACS is 4.0 (Fourth major re-write, initial public release), and for the rest of this document, we shall simply refer to this version as “EMACS”. Any modifications for later versions will be in the file README on the MicroEMACS distribution disk.

2.1 Keys and the Keyboard

Many times throughout this manual we will be talking about commands and the keys on the keyboard needed to use them. There are a number of “special” keys which can be used and are listed here:

<NL>	NewLine which is also called RETURN, ENTER, or <NL>, this key is used to end different commands.
^	The control key can be used before any alphabetic character and some symbols. For example, ^C means to hold down the <CONTROL> key and type the C key at the same time.
^X	The CONTROL-X key is used at the beginning of many different commands.
META or M-	This is a special EMACS key used to begin many commands.This key is pressed and then released before typing the next character. On most systems, this is the <ESC> key, but it can be changed. (consult appendix E to learn what key is used for META on your computer).

Whenever a command is described, the manual will list the actual keystrokes needed to execute it in **boldface** using the above conventions, and also the name of the command in *italics*.

2.2 Getting Started

In order to use EMACS, you must call it up from your system or computer’s command prompt. On UNIX and MSDOS machines, just type “emacs” from the command prompt and follow it with the <RETURN> or <ENTER> key (we will refer to this key as <NL> for “new-line” for the remainder of this manual). On the Macintosh, the Amiga, the ATARI ST, and under OS/2, Windows 95, Windows NT and other icon based operating systems, double click on the uEMACS icon. Shortly after this, a screen similar to the one below should appear.

2.3 Parts and Pieces

The screen is divided into a number of areas or **windows**. On some systems the top window contains a function list of unshifted and shifted function keys. We will discuss these keys later. Below them is an EMACS **mode line** which, as we will see, informs you of the present mode of operation of the editor—for example “(WRAP)” if you set EMACS to wrap at the end of each line. Under the mode line is the **text window** where text appears and is manipulated. Since each window has its own mode line, below the text window is it’s mode line. The last line of the screen is the **command line** where EMACS takes commands and reports on what it is doing.

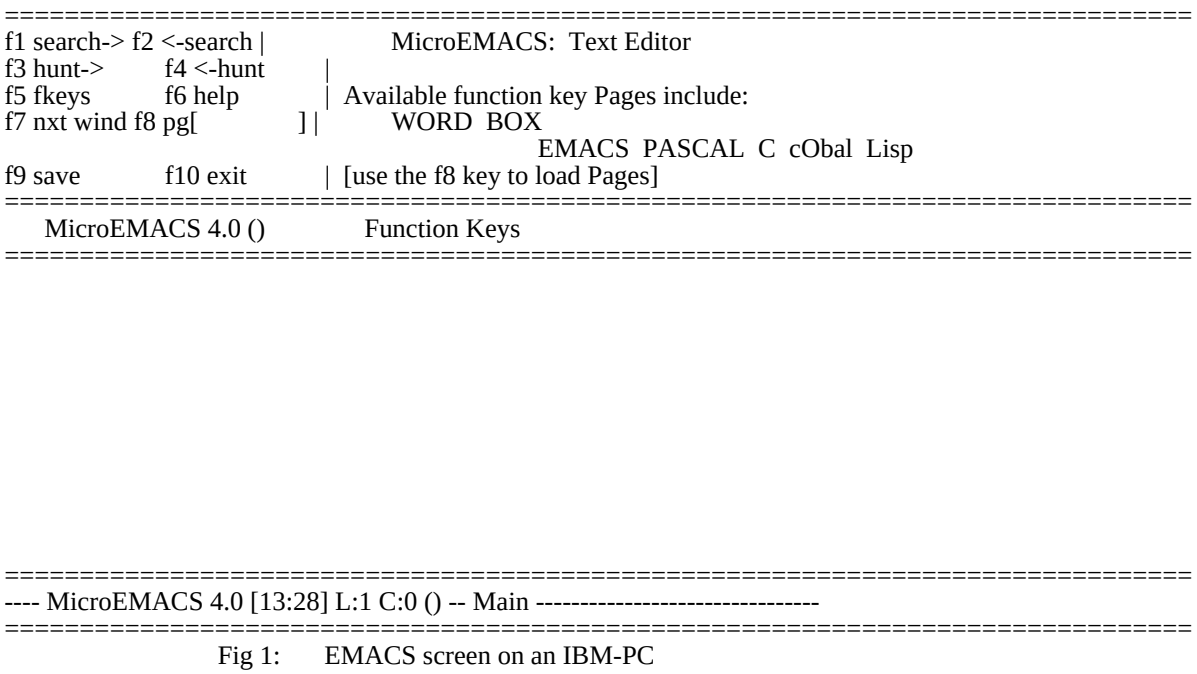

Fig 1: EMACS screen on an IBM-PC

2.4 Entering Text

Entering text in EMACS is simple. Type the following sentence fragment:

```
Fang Rock lighthouse, center of a series of mysterious
and
```

The text is displayed at the top of the text window. Now type:

```
terrifying events at the turn of the century
```

Notice that some of your text has dissapeared off the left side of the screen. Don't panic—your text is safe!!! You've just discovered that EMACS doesn't "wrap" text to the next line like most word processors unless you hit <NL>. But since EMACS is used for both word processing, and text editing, it has a bit of a dual personality. You can change the way it works by setting various **modes**. In this case, you need to set **WRAP** mode, using the *add-mode* command, by typing **^XM**. The command line at the base of the screen will prompt you for the mode you wish to add. Type **wrap** followed by the <NL> key and any text you now enter will be wrapped. However, the command doesn't wrap text already entered. To get rid of the long line, press and hold down the <BACKSPACE> key until the line is gone. Now type in the words you deleted, watch how EMACS goes down to the next line at the right time. *(In some versions of EMACS, **WRAP** is a default mode in which case you don't have to worry about the instructions relating to adding this mode.)*

Now let's type a longer insert. Hit <NL> a couple of times to tab down from the text you just entered. Now type the following paragraphs. Press <NL> twice to indicate a paragraph break.

```
Fang Rock lighthouse, center of a series of mysterious
and terrifying events at the turn of the century, is
built on a rocky island a few miles of the Channel
coast. So small is the island that wherever you stand
its rocks are wet with sea spray.

The lighthouse tower is in the center of the island. A
steep flight of steps leads to the heavy door in its
base. Winding stairs lead up to the crew room.
```

2.5 Basic cursor movement

Now let's practice moving around in this text. To move the cursor back to the word "Winding," enter **M-B** *previous-word*. This command moves the cursor backwards by one word at a time. Note you have to press the key combination every time the cursor steps back by one word. Continuously pressing META and toggling B produces an error message. To move forward to the word "stairs" enter **M-F** *next-word*, which moves the cursor forward by one word at a time.

Notice that EMACS commands are usually mnemonic—F for forward, B for backward, for example.

To move the cursor up one line, enter **^P** *previous-line*, down one line **^N** *next-line*. Practice this movement by moving the cursor to the word "terrifying" in the second line.

The cursor may also be moved forward or backward in smaller increments. To move forward by one character, enter **^F** *forward-character* , to move backward, **^B** *backward-character*. EMACS also allows you to specify a number which is normally used to tell a command to execute many times. To repeat most commands, press META and then the number before you enter the command. Thus, the command META 5 ^F (**M-5^F**) will move the cursor forward by five characters. Try moving around in the text by using these commands. For extra practice, see how close you can come to the word "small" in the first paragraph by giving an argument to the commands listed here.

Two other simple cursor commands that are useful to help us move around in the text are **M-N** *next-paragraph* which moves the cursor to the second paragraph, and **M-P** *previous-paragraph* which moves it back to the previous paragraph. The cursor may also be moved rapidly from one end of the line to the other. Move the cursor to the word "few" in the second line. Press **^A** *beginning-of-line*. Notice the cursor moves to the word "events" at the beginning of the line. Pressing **^E** *end-of-line* moves the cursor to the end of the line.

Finally, the cursor may be moved from any point in the file to the end or beginning of the file. Entering **M->** *end-of-file* moves the cursor to the end of the buffer, **M-<** *beginning-of-file* to the first character of the file.

On the IBM-PC, the ATARI ST and many other machines, the cursor keys can also be used to move the cursor.

Practice moving the cursor in the text until you are comfortable with the commands we've explored in this chapter.

2.6 Saving your text

When you've finished practicing cursor movement, save your file. Your file currently resides in a **BUFFER**. The buffer is a temporary storage area for your text, and is lost when the computer is turned off. You can save the buffer to a file by entering **^X^S** *save-file*. Notice that EMACS informs you that your file has no name and will not let you save it.

On the IBM-PC and other machines with function keys, the F9 key can often be used do the same as ^X^S.

To save your buffer to a file with a different name than it's current one (which is empty), press **^X^W** *write-file*. EMACS will prompt you for the filename you wish to write. Enter the name **fang.txt** and press return. On a micro, the drive light will come on, and EMACS will inform you it is writing the file. When it finishes, it will inform you of the number of lines it has written to the disk.

Congratulations!! You've just saved your first EMACS file!

Chapter 2 Summary

In chapter 2, you learned how to enter text, how to use wrap mode, how to move the cursor, and to save a buffer. The following is a table of the commands covered in this chapter and their corresponding key bindings:

Key Binding	Keystroke	Effect
abort-command	**^G**	aborts current command
add-mode	**^XM**	allows addition of EMACS mode such as **WRAP**
backward-character	**^B**	moves cursor left one character
beginning-of-file	**M-<**	moves cursor to beginning of file
beginning-of-line	**^A**	moves cursor to beginning of line
end-of-file	**M->**	moves cursor to end of file
end-of-line	**^E**	moves cursor to end of line
forward-character	**^F**	moves cursor right one character
next-line	**^N**	moves cursor to next line
next-paragraph	**M-N**	moves cursor to next paragraph
next-word	**M-F**	moves cursor forward one word
previous-line	**^P**	moves cursor backward by one line
previous-paragraph	**M-P**	moves cursor to previous paragraph
previous-word	**M-B**	moves cursor backward by one word
save-file	**^X^S**	saves current buffer to a file
write-file	**^X^W**	save current buffer under a new name

Chapter 3

Basic Editing—Simple Insertions and Deletions

3.1 A Word About Windows, Buffers, Screens, and Modes

In the first chapter, you learned how to create and save a file in EMACS. Let's do some more editing on this file. Call up emacs by typing in the following command.

emacs fang.txt

*On icon oriented systems, double click on the uEMACS icon, usually a file dialog box of some sort will appear. Choose **FANG.TXT** from the appropriate folder.*

Shortly after you invoke EMACS, the text should appear on the screen ready for you to edit. The text you are looking at currently resides in a **buffer**. A buffer is a temporary area of computer memory which is the primary unit internal to EMACS — this is the place where EMACS goes to work. The mode line at the bottom of the screen lists the buffer name, **FANG.TXT** and the name of the file with which this buffer is associated, **FANG.TXT**

The computer talks to you through the use of its **screen**. This screen usually has an area of 24 lines each of 80 characters across. You can use EMACS to subdivide the screen into several separate work areas, or **windows**, each of which can be 'looking into' different files or sections of text. Using windows, you can work on several related texts at one time, copying and moving blocks of text between windows with ease. To keep track of what you are editing, each window is identified by a **mode line** on the last line of the window which lists the name of the **buffer** which it is looking into, the file from which the text was read, and how the text is being edited.

An EMACS **mode** tells EMACS how to deal with user input. As we have already seen, the mode 'WRAP' controls how EMACS deals with long lines (lines with over 79 characters) while the user is typing them in. The 'VIEW' mode, allows you to read a file without modifying it. Modes are associated with buffers and not with files; hence, a mode needs to be explicitly set or removed every time you edit a file. A new file read into a buffer with a previously specified mode will be edited under this mode. If you use specific modes frequently, EMACS allows you to set the modes which are used by all new buffers, called **global** modes.

3.2 Insertions

Your previously-saved text should look like this:

```
Fang Rock lighthouse, center of a series of mysterious
and terrifying events at the turn of the century, is
built on a rocky island a few miles of the Channel
coast. So small is the island that wherever you stand
its rocks are wet with sea spray.

The lighthouse tower is in the center of the island. A
steep flight of steps leads to the heavy door in its
base. Winding stairs lead up to the crew room.
```

Let's assume you want to add a sentence in the second paragraph after the word "base." Move the cursor until it is on the "W" of "Winding". Now type the following:

```
This gives entry to the lower floor where the big steam
generator throbs steadily away, providing power for the
electric lantern.
```

If the line fails to wrap and you end up with a '$' sign in the right margin, just enter **M-Q** *fill-paragraph* to reformat the paragraph. This new command attempts to fill out a paragraph. Long lines are divided up, and words are shuffled around to make the paragraph look nicer.

Notice that all visible EMACS characters are self-inserting — all you had to do was type the characters to insert and the existing text made space for it. With a few exceptions discussed later, all non-printing characters (such as control or escape sequences) are commands. To insert spaces, simply use the space bar. Now move to the first line of the file and type **^O** *open-line* (Oh, not zero). You've just learned how to insert a blank line in your text.

3.3 Deletions

EMACS offers a number of deletion options. For example, move the cursor until it's under the period at the end of the insertion you just did. Press the backspace key. Notice the "n" on "lantern" disappeared. The backspace implemented on EMACS is called a **destructive** backspace—it removes text immediately before the current cursor position from the buffer. Now type **^H** *delete-previous-character*. Notice that the cursor moves back and obliterates the "r"—either command will backspace the cursor.

Type in the two letters you erased to restore your text and move the cursor to the beginning of the buffer **M->** *beginning-of-file*. Move the cursor down one line to the beginning of the first paragraph.

To delete the forward character, type **^D** *delete-next-character*. The "F" of "Fang" disappears. Continue to type **^D** until the whole word is erased EMACS also permits the deletion of larger elements of text. Move the cursor to the word "center" in the first line of text. Pressing **M-<backspace>** *delete-previous-word* kills the word immediately before the cursor. **M-^H** has the same effect.

Notice that the commands are very similar to the control commands you used to delete individual letters. As a general rule in EMACS, control sequences affect small areas of text, META sequences larger areas. The word forward of the cursor position can therefore be deleted by typing **M-D** *delete-next-word*. Now let's take out the remainder of the first line by typing **^K** *kill-to-end-of-line*. You now have a blank line at the top of your screen. Typing **^K** again or **^X^O** *delete-blank-lines* deletes the blank line and flushes the second line to the top of the text. Now exit EMACS by typing **^X^C** *exit-emacs*. Notice EMACS reminds you that you have not saved your buffer. Ignore the warning and exit. This way you can exit EMACS without saving any of the changes you just made.

Chapter 3 Summary

In Chapter 3, you learned about the basic ‘building blocks’ of an EMACS text file—buffers, windows, and files.

Key binding	Keystroke	Effect
delete-previous-character	**^H**	deletes character immediately before the current cursor position
delete-next-character	**^D**	deletes character immediately after current cursor position
delete-previous-word	**M-^H**	deletes word immediately before current cursor position
delete-next-word	**M-D**	deletes word immediately after current cursor position
kill-to-end-of-line	**^K**	deletes from current cursor position to end of line
insert-space	**^C**	inserts a space to right of cursor
open-line	**^O**	inserts blank line
delete-blank-lines	**^X^O**	removes blank line
exit-emacs	**^X^C**	exits emacs

Chapter 4

Using Regions

4.1 Defining and Deleting a Region

At this point its time to familiarize ourselves with two more EMACS terms—the **point** and the **mark**. The point is located directly behind the current cursor position. The mark (as we shall see shortly) is user defined. These two elements together are called the current **region** and limit the **region** of text on which EMACS performs many of its editing functions.

Let's begin by entering some new text. Don't forget to add **wrap** mode if its not set on this buffer. Start EMACS and open a file called **PUBLISH.TXT**. Type in the following text:

```
One of the largest growth areas in personal computing is
electronic publishing. There are packages available for
practically every machine from elegantly simple programs
for the humble Commodore 64 to sophisticated
professional packages for PC and Macintosh computers.

Electronic publishing is as revolutionary in its way as
the Gutenburg press. Whereas the printing press allowed
the mass production and distribution of the written
word, electronic publishing puts the means of production
in the hands of nearly every individual. From the class
magazine to the corporate report, electronic publishing
is changing the way we produce and disseminate
information.

Personal publishing greatly increases the utility of
practically every computer. Thousands of people who
joined the computer revolution of this decade only to
hide their machines unused in closets have discovered a
new use for them as dedicated publishing workstations.
```

Now let's do some editing. The last paragraph seems a little out of place. To see what the document looks like without it we can cut it from the text by moving the cursor to the beginning of the paragraph. Enter **M-<space>** *set-mark*. EMACS will respond with "[Mark set]". Now move the cursor to the end of the paragraph. You have just defined a region of text. To remove this text from the screen, type **^W** *kill-region*. The paragraph disappears from the screen.

On further consideration, however, perhaps the paragraph we cut wasn't so bad after all. The problem may have been one of placement. If we could tack it on to the end of the first paragraph it might work quite well to support and strengthen the argument. Move the cursor to the end of the first paragraph and enter **^Y** *yank* . Your text should now look like this:

```
One of the largest growth areas in personal computing is
electronic publishing. There are packages available for
practically every machine from elegantly simple programs
for the humble Commodore 64 to sophisticated
professional packages for PC and Macintosh computers.
Personal publishing greatly increases the utility of
practically every computer. Thousands of people who
joined the computer revolution of this decade only to
hide their machines unused in closets have discovered a
new use for them as dedicated publishing workstations.

Electronic publishing is as revolutionary in its way as
the Gutenburg press. Whereas the printing press allowed
the mass production and distribution of the written
word, electronic publishing puts the means of production
in the hands of nearly every individual. From the class
magazine to the corporate report, electronic publishing
is changing the way we produce and disseminate
information.
```

4.2 Yanking a Region

The text you cut initially didn't simply just disappear, it was cut into a buffer that retains the 'killed' text appropriately called the **kill buffer**. **^Y** "yanks" the text back from this buffer into the current buffer. If you have a long line (indicated, remember, by the "$" sign), simply hit **M-Q** to reformat the paragraph.

There are other uses to which the kill buffer can be put. Using the method we've already learned, define the last paragraph as a region. Now type **M-W** *copy-region*. Nothing seems to have happened; the cursor stays blinking at the point. But things have changed, even though you may not be able to see any alteration.

To see what has happened to the contents of the kill buffer, move the cursor down a couple of lines and "yank" the contents of the kill buffer back with **^Y**. Notice the last paragraph is now repeated. The region you defined is "tacked on" to the end of your file because **M-W copies** a region to the kill buffer while leaving the original text in your working buffer. Some caution is needed however, because the contents of the kill buffer are updated when you delete any regions, lines or words. If you are moving large quantities of text, complete the operation before you do any more deletions or you could find that the text you want to move has been replaced by the most recent deletion. Remember—a buffer is a temporary area of computer memory that is lost when the machine is powered down or switched off. In order to make your changes permanent, they must be saved to a file before you leave EMACS. Let's delete the section of text we just added and save the file to disk.

Chapter 4 Summary

In Chapter 4, you learned how to achieve longer insertions and deletions. The EMACS terms **point** and **mark** were introduced and you learned how to manipulate text with the kill buffer.

Key Binding	Keystroke	Effect
set-mark	**M-<space>**	Marks the beginning of a region
delete-region	**^W**	Deletes region between point and mark and places it in KILL buffer
copy-region	**M-W**	Copies text between point and mark into KILL buffer
yank-text	**^Y**	Inserts a copy of the KILL buffer into current buffer at point

Chapter 5

Search and Replace

5.1 Forward Search

Load EMACS and bring in the file you just saved. Your file should look like the one below.

```
One of the largest growth areas in personal computing is
electronic publishing. There are packages available for
practically every machine from elegantly simple programs
for the humble Commodore 64 to sophisticated
professional packages for PC and Macintosh computers.
Personal publishing greatly increases the utility of
practically every computer. Thousands of people who
joined the computer revolution of this decade only to
hide their machines unused in closets have discovered a
new use for them as dedicated publishing workstations.

Electronic publishing is as revolutionary in its way as
the Gutenburg press. Whereas the printing press allowed
the mass production and distribution of the written
word, electronic publishing puts the means of production
in the hands of nearly every individual. From the class
magazine to the corporate report, electronic publishing
is changing the way we produce and disseminate
information.
```

Let's use EMACS to search for the word "revolutionary" in the second paragraph. Because EMACS searches from the current cursor position toward the end of buffers, and we intend to search forward, move the cursor to the beginning of the text. Enter **^S** *search-forward*. Note that the command line now reads

"Search [] <META>:"

EMACS is prompting you to enter the **search string** — the text you want to find. Enter the word **revolutionary** and hit the **META** key. The cursor moves to the end of the word "revolutionary."

Notice that you must enter the <META> key to start the search. If you simply press <NL> the command line responds with "<NL>". Although this may seem infuriating to users who are used to pressing the return key to execute any command, EMACS' use of <META> to begin searches allows it to pinpoint text with great accuracy. After every line wrap or carriage return, EMACS 'sees' a new line character (<NL>). If you need to search for a word at the end of a line, you can specify this word uniquely in EMACS.

In our sample text for example, the word "and" occurs a number of times, but only once at the end of a line. To search for this particular occurrence of the word, move the cursor to the beginning of the buffer and type **^S**. Notice that EMACS stores the last specified search string as the **default** string. If you press **<META>** now, EMACS will search for the default string, in this case, "revolutionary."

To change this string so we can search for our specified "and" simply enter the word **and** followed by **<NL>**. The command line now shows:

"search [and<NL>]<META>:"

Press **<META>** and the cursor moves to "and" at the end of the second last line.

5.2 Exact Searches

Most of the time EMACS does not take the case of the strings being searched for into account. Thus **And** matches **AND** and **and**. However, if the mode EXACT is active in the current buffer, EMACS will only look for matches with the same upper and lower case for each character. Thus, for example you could search for **Publishing** as distinct from **publishing**.

5.3 Backward Search

Backward searching is very similar to forward searching except that it is implemented in the reverse direction. To implement a reverse search, type **^R** *search-reverse*. Because EMACS makes no distinction between forward and backward stored search strings, the last search item you entered appears as the default string. Try searching back for any word that lies between the cursor and the beginning of the buffer. Notice that when the item is found, the point moves to the beginning of the found string (i.e., the cursor appears under the first letter of the search item).

Practice searching for other words in your text.

5.4 Searching and Replacing

Searching and replacing is a powerful and quick way of making changes to your text. Our sample text is about electronic publishing, but the correct term is ‘desktop’ publishing. To make the necessary changes we need to replace all occurrences of the word “electronic” with “desktop.” First, move the cursor to the top of the current buffer with the **M-<** command. Then type **M-R** *replace-string*. The command line responds:

“Replace []<META>:”

where the square brackets enclose the default string. Type the word **electronic** and hit **<META>**. The command line responds:

“with []<META>”

type **desktop<META>**. EMACS replaces all instances of the original word with your revision. Of course, you will have to capitalize the first letter of “desktop” where it occurs at the beginning of a sentence.

You have just completed an **unconditional replace**. In this operation, EMACS replaces every instance of the found string with the replacement string.

5.5 Query-Replace

You may also replace text on a case by case basis. The **M-^R** *query-replace-string* command causes EMACS to pause at each instance of the found string.

For example, assume we want to replace some instances of the word “desktop” with the word “personal.” Go back to the beginning of the current buffer and enter the **M-^R** *query-replace* command. The procedure is very similar to that which you followed in the unconditional search/replace option. When the search begins however, you will notice that EMACS pauses at each instance of “publishing” and asks whether you wish to replace it with the replacement string. You have a number of options available for response:

Response	Effect
Y(es)	Make the current replacement and skip to the next occurrence of the search string
N(o)	Do not make this replacement but continue
!	Do the rest of the replacements with no more queries
U(ndo)	Undo just the last replacement and query for it again (This can only go back ONE time)

^G	Abort the replacement command (This action does not undo previously-authorized replacements
.	Same effect as ^G, but cursor returns to the point at which the replacement command was given
?	This lists help for the query replacement command

Practice searching and searching and replacing until you feel comfortable with the commands and their effects.

Chapter 5 Summary

In this chapter, you learned how to search for specified strings of text in EMACS. The chapter also dealt with searching for and replacing elements within a buffer.

Key Binding	Keystroke	Effect
search-forward	**^S**	Searches from point to end of buffer. Point is moved from current location to the end of the found string
search-backward	**^R**	Searches from point to beginning of buffer. Point is moved from current location to beginning of found string
replace	**M-R**	Replace ALL occurrences of search string with specified (null) string from point to the end of the current buffer
query-replace	**M-^R**	As above, but pause at each found string and query for action

Chapter 6

Windows

6.1 Creating Windows

We have already met windows in an earlier chapter. In this chapter, we will explore one of EMACS' more powerful features — text manipulation through multiple windowing. Windows offer you a powerful and easy way to edit text. By manipulating a number of windows and buffers on the screen simultaneously, you can perform complete edits and revisions on the computer screen while having your draft text or original data available for reference in another window.

You will recall that windows are areas of buffer text that you can see on the screen. Because EMACS can support several screen windows simultaneously you can use them to look into different places in the same buffer. You can also use them to look at text in different buffers. In effect, you can edit several files at the same time.

Let's invoke EMACS and pull back our file on desktop publishing by typing

```
emacs publish.txt
```

When the text appears, type the **^X2** *split-current-window* command. The window splits into two windows. The window where the cursor resides is called the **current** window — in this case the bottom window. Notice that each window has a text area and a mode line. The **command line** is however, common to all windows on the screen.

The two windows on your screen are virtually mirror images of each other because the new window is opened into the same buffer as the one you are in when you issue the *open-window command*. All commands issued to EMACS are executed on the current buffer in the current window.

To move the cursor to the upper window (i.e., to make that window the current window, type **^XP** *previous-window*. Notice the cursor moves to the upper or **previous** window. Entering **^XO** *next-window* moves to the **next** window. Practice moving between windows. You will notice that you can also move into the Function Key menu by entering these commands.

Now move to the upper window. Let's open a new file. On the EMACS disk is a tutorial file. Let's call it into the upper window by typing:

```
^X^F
```

and press return.

Enter the filename **emacs.tut**.

In a short time, the tutorial file will appear in the window. We now have two windows on the screen, each looking into different buffers. We have just used the **^X^F** *find-file* command to find a file and bring it into our current window.

You can scroll any window up and down with the cursor keys, or with the commands we've learned so far. However, because the area of visible text in each window is relatively small, you can scroll the current window a line at a time.

Type **^X^N** *move-window-down*

The current window scrolls down by one line — the top line of text scrolls out of view, and the bottom line moves towards the top of the screen. You can imagine, if you like, the whole window slowly moving down to the end of the buffer in increments of one line. The command **^X^P** *move-window-up* scrolls the window in the opposite direction.

As we have seen, EMACS editing commands are executed in the current window, but the program does support a useful feature that allows you to scroll the **next** window. **M-^Z** *scroll-next-up* scrolls the next window up,

M-^V *scroll-next-down* scrolls it downward. From the tutorial window, practice scrolling the window with the desktop publishing text in it up and down.

When you're finished, exit EMACS without saving any changes in your files.

Experiment with splitting the windows on your screen. Open windows into different buffers and experiment with any other files you may have. Try editing the text in each window, but don't forget to save any changes you want to keep — you still have to save each buffer separately.

6.2 Deleting Windows

Windows allow you to perform complex editing tasks with ease. However, they become an inconvenience when your screen is cluttered with open windows you have finished using. The simplest solution is to delete unneeded windows. The command **^X0** *delete-window* will delete the window you are currently working in and move you to the next window.

If you have a number of windows open, you can delete all but the current window by entering **^X1** *delete-other-windows*.

6.3 Resizing Windows

During complex editing tasks, you will probably find it convenient to have a number of windows on the screen simultaneously. However this situation may present inconveniences because the more windows you have on the screen the smaller they are; in some cases, a window may show only a couple of lines of text. To increase the flexibility and utility of the window environment, EMACS allows you to resize the window you are working in (called, as you will recall, the **current** window) to a convenient size for easier editing, and then shrink it when you no longer need it to be so large.

Let's try an example. Load in any EMACS text file and split the current window into two. Now type **^X^(Shift-6)**, *grow-window*. Your current window should be the lower one on the screen. Notice that it increases in size upwards by one line. If you are in the upper window, it increases in size in a downward direction. The command **^X^Z**, *shrink-window* correspondingly decreases window size by one line at a time.

EMACS also allows you to resize a window more precisely by entering a numeric argument specifying the size of the window in lines. To resize the window this way, press the META key and enter a numeric argument (remember to keep it smaller than the number of lines on your screen display) then press **^XW** *resize-window*. The current window will be enlarged or shrunk to the number of lines specified in the numeric argument. For example entering:

```
M-8 ^XW
```

will resize the current window to 8 lines.

6.4 Repositioning within a Window

The cursor may be centered within a window by entering **M-! or M-^L** *redraw-display*. This command is especially useful in allowing you to quickly locate the cursor if you are moving frequently from window to window. You can also use this command to move the line containing the cursor to any position within the current window. This is done by using a numeric argument before the command. Type **M-<n> M-^L** where <n> is the number of the line within the window that you wish the current line to be displayed.

The **^L** *clear-and-redraw* command is useful for 'cleaning up' a 'messy' screen that can result of using EMACS on a mainframe system and being interrupted by a system message.

Chapter 6 summary

In Chapter 6 you learned how to manipulate windows and the editing flexibility they offer.

Key Binding	Keystroke	Effect
open-window	**^X2**	Splits current window into two windows if space available
close-windows	**^X1**	Closes all windows except current window
next-window	**^XO**[oh]	Moves point into next (i.e. downward) window
previous-window	**^XP**	Moves point to previous (i.e. upward) window
move-window-down	**^X^N**	Scrolls current window down one line
move-window-up	**^X^P**	Scrolls current window up one line
redraw-display	**M !** or **M ^L**	Window is moved so line with point (with cursor) is at center of window
grow-window	**M-X ^**	Current window is enlarged by one line and nearest window is shrunk by one line
shrink-window	**^X^Z**	Current window is shrunk by one line and nearest window is enlarged by one line
clear-and-redraw	**^L**	Screen is blanked and redrawn. Keeps screen updates in sync with your commands
scroll-next-up	**M-^Z**	Scrolls next window up by one line
scroll-next-down	**M-^V**	Scrolls next window down by one line
delete-window	**^X0**	Deletes current window
delete-other-windows	**^X1**	Deletes all but current window
resize-window	**^X^W**	Resizes window to a given numeric argument

Chapter 7

Using a Mouse

On computers equipped with a mouse, the mouse can usually be used to make editing easier. If your computer has a mouse, let's try using it. Start MicroEMACS by typing:

emacs publish.txt

This brings EMACS up and allows it to edit the file from the last chapter. If the function key window is visible on the screen, press the F5 key to cause it to disappear. Now use the **^X2** *split-current-window* command to split the screen into two windows. Next use the **^X^F** *find-file* command to read in the **fang.txt** file. Now your screen should have two windows looking into two different files.

Grab the mouse and move it around. On the screen an arrow, or block of color appears. This is called the mouse cursor and can be positioned on any character on the screen. On some computers, positioning the mouse cursor in the extreme upper right or left corner may bring down menus which allow you to access that computers utilities, sometimes called **Desk Accessories**.

7.1 Moving around with the mouse

Using the mouse button (or the left button if the mouse has more than one), position the mouse over some character in the current window. Click the mouse button once. The **point** will move to where the mouse cursor is. If you place the mouse cursor past the end of a line, the point will move to the end of that line.

Move the mouse cursor into the other window and click on one of the characters there. MicroEMACS will automatically make this window the current window (notice that the mode line changes) and position the point to the mouse cursor. This makes it very easy to use the mouse to switch to a different window quickly.

7.2 Dragging around

Besides just using the mouse to move around on the screen, you can use the same button to move text. Move the mouse cursor to a character in one of the windows, and click down… but don't let the button up yet! The point will move to where the mouse cursor is. Now move the mouse cursor up or down on the screen, and release the button. The point will again move to where the mouse cursor is, but this time it will bring the text under it along for the ride. This is called **dragging**, and is how you can make the text appear just where you want it to. If you try to drag text out of the current window, EMACS will ignore your attempt and leave the point where you first clicked down.

Now, click down on a word in one of the windows, and drag it directly to the left. Release the button and watch as the entire window slides, or **scrolls** to the left. The missing text has not been deleted, it is simply not visible, off the left hand side of the screen. Notice the mode line has changed and now looks similar to this:

==== MicroEMACS 4.0 [15:12] [<12] L:4 C:23 () == fang.txt == File: fang.txt ==========

The number insided the brackets [] shows that the screen is now scrolled 12 characters from the left margin.

Now grab the same text again, and drag it to the right, pulling the rest of the text back into the current window. The [<] field will disappear, meaning that the window is no longer scrolled to the left. This feature is very useful for looking at wide charts and tables. Remember, MicroEMACS will only scroll the text in the current window sideways if you drag it straight to the side, otherwise it will drag the text vertically.

Now, place the mouse cursor over a character on the upper mode line, click down, move the mouse cursor up or down a few lines and let go of the button. The mode line moves to where you dragged it, changing the size of the windows above and below it. If you try to make a window with less than one line, EMACS will not let you. Dragging the mode lines can make it very fast and easy for you to rearrange the windows as you would like.

If you have a number of different windows visible on the screen, positioning the mouse over the mode line of one window and clicking the right mouse button will cause that window to be deleted.

7.3 Cut and Paste

If your mouse has two buttons, then you can use the right button to do some other things as well. Earlier, we learned how to define a **region** by using the **M-<space>** *set-mark* command. Now, position the mouse over at the beginning of a region you would like to copy. Next click and hold down the right mouse button. Notice that the point jumps to the mouse cursor and EMACS reports "[Mark Set]". Holding the button down move the mouse to the end of the text you wish to copy and release the mouse button. Emacs reports "[Region Copied]" to let you know it has copied the region into the KILL buffer. This has done the same job as the **M-W** *copy-region* command.

If you now click the right mouse button, without moving the mouse, the region you defined dissapear, being **cut** from the current buffer. This works just like the **^W** *kill-region* command.

If you move the mouse away from where you cut the text, and click the right mouse button down and up without moving the mouse, the text in the KILL buffer gets inserted, or pasted into the current buffer at the point.

7.4 Screens

MicroEMACS can use more than one screen at once. Each screen is a collection of *windows* along with a mode line. These screens usually fill the terminal or computer screen on text based systems, but can also be held in different **windows** on graphically based systems like MicroSoft Windows, OS/2, the Macintosh Finder and X-Windows. Don't be confused by the two different uses of the term "window". Inside EMACS style editors, a *window* lets you view part of a buffer. Under graphical operating systems, a **window** holds a "virtual terminal", allowing you to manipulate more than one job, editing session or program at once. Within MicroEMACS, these operating system **window**s are called screens. All these screens are displayed on your current desktop.

7.5 Resizing a Screen

You can change the size of a screen. Move the mouse to the last position of the command line. Press the left mouse button down. Holding it, move the mouse to the place you want the new lower right corner. Release the mouse. The desktop redraws, with your newly resized screen. MicroEMACS will ignore size changes that can not be done, like attempting to pull the lower left corner above the upper right corner of the current screen.

7.6 Moving a Screen

To change where on the desktop a screen is placed, move the mouse to the upper right corner of the screen, press the left mouse button down, move the mouse and release it where you want the screen displayed. Again, MicroEMACS will ignore placements that can not be done.

7.7 Creating a Screen

Creating a new screen is just like moving a screen, but using the right button. Move to the upper right of an existing screen, press the right mouse button down, and move the mouse, releasing the button where the new screen should appear. A new screen will have a single **window**, containing the contents of the current window in the copied screen, and will have that **window**'s colors. The new screen will have the copied screen's size.

7.8 Switching to a Screen

This is simple. Any mouse command can be done in any screen by placing the mouse on a visible part of the screen and clicking. The last screen the mouse is used on comes to front and is the current screen. Also, the **A-C** *cycle-screens* command brings the rearmost screen to front.

7.9 Deleting a Screen

Place the mouse on the command line of the screen you want to delete. Click the right mouse button, the screen will disapear. If you delete the only remaining screen on the desktop, MicroEMACS will exit.

7.10 Mousing under WINDOWS 95

The **Windows 95** version of MicroEMACS loads a special macro when it comes up which makes the mousing more similar to the CUA standard. The left mouse button is used to outline regions which then can be cut and pasted to and from the clipboard from the menus. Text placed in the clipboard can be pasted to other documents.

If you wish to disable this and use the standard EMACS mousing, rename CUA.CMD in the installation directory to another name.

Chapter 7 Summary

In Chapter 7, you learned how to use the mouse to move the point, switch windows, drag text, and resize windows. You also learned how to use the right mouse button in order to copy and delete regions and yank them back at other places. And lastly, you learned how to control multiple screens with the mouse.

Action	Mouse Directions
Move Cursor	position mouse cursor over desired location click down and up with left button
Drag Text	position mouse cursor over desired text click left button down move to new screen location for text release mouse button
Resize Windows	position mouse cursor over mode line to move click left button down move to new location for mode line release mouse button
Delete Window	position mouse cursor over mode line of window to delete click right mouse button
Activate Screen	Move mouse over existing screen click left button down and up
Resize Screen	position mouse cursor over last character on message line click left button down move to new lower right corner of screen release mouse button
Copy Region	position mouse at beginning of region click right button down move to end of region release mouse button
Cut Region	position mouse at beginning of region click right button down move to end of region release mouse button click right button down and up
Paste Region	position mouse at place to paste click right button down and up
Create Screen	position mouse at upper left corner of existing screen click right button down move to position of new screen release mouse button
Resize Screen	position mouse at lower right corner of screen click left button down move to new lower left corner release mouse button
Move Screen	position mouse at upper right corner of screen click left button down move to new screen position release mouse button
Delete Screen	position to command line of existing screen click right button down release mouse button

Chapter 8

Buffers

We have already learned a number of things about buffers. As you will recall, they are the major internal entities in EMACS — the place where editing commands are executed. They are characterized by their **names**, their **modes**, and by the file with which they are associated. Each buffer also "remembers" its **mark** and **point**. This convenient feature allows you to go to other buffers and return to the original location in the "current" buffer.

Advanced users of EMACS frequently have a number of buffers in the computer's memory simultaneously. In the last chapter, for example, you opened at least two buffers — one into the text you were editing, and the other into the EMACS on-line tutorial. If you deal with complex text files — say, sectioned chapters of a book, you may have five or six buffers in the computer's memory. You could select different buffers by simply calling up the file with **^X^F** *find-file*, and let EMACS open or reopen the buffer. However, EMACS offers fast and sophisticated buffering techniques that you will find easy to master and much more convenient to use.

Let's begin by opening three buffers. You can open any three you choose, for example call the following files into memory: **fang.txt**, **publish.txt**, and **emacs.tut** in the order listed here. When you've finished this process, you'll be looking at a screen showing the EMACS tutorial. Let's assume that you want to move to the fang.txt buffer. Enter:

^XX *next-buffer*

This command moves you to the next buffer. Because EMACS cycles through the buffer list, which is alphabetized, you will now be in the **fang.txt** buffer. Using **^XX** again places you in the **publish.txt** buffer. *If you are on a machine that supports function keys, using* ***^XX*** *again places you in the* ***Function Keys*** *buffer*. Using **^XX** one last time cycles you back to the beginning of the list.

If you have a large number of buffers to deal with, this cycling process may be slow and inconvenient. The command **^XB** *select-buffer* allows you to specify the buffer you wish to be switched to. When the command is entered, EMACS prompts, "Use buffer:". Simply enter the buffer name (NOT the file name), and that buffer will then become the current buffer. If you type in part of the file name and press the space bar, EMACS will attempt to complete the name from the list of current buffers. If it succeeds, it will print the rest of the name and you can hit <NL> to switch to that buffer. If EMACS beeps the bell, there is no such buffer, and you may continue editing the name on the command line.

Multiple buffer manipulation and editing is a complex activity, and you will probably find it very inconvenient to re-save each buffer as you modify it. The command **^X^B** *list-buffers* creates a new window that gives details about all the buffers currently known to EMACS. Buffers that have been modified are identified by the "buffer changed" indicator (an asterisk in the second column). You can thus quickly and easily identify buffers that need to be saved to files before you exit EMACS. The buffer window also provides other information — buffer specific modes, buffer size, and buffer name are also listed. To close this window, simply type the close-windows command, **^X1**.

To delete any buffer, type **^XK** *delete-buffer*. EMACS prompts you "Kill buffer:". Enter the buffer name you want to delete. As this is destructive command, EMACS will ask for confirmation if the buffer was changed and not saved. Answer Y(es) or N(o). As usual **^G** cancels the command.

Chapter 8 Summary

In Chapter 8 you learned how to manipulate buffers.

Key Binding	Keystroke	Effect
next-buffer	**^X^X**	Switch to the next buffer in the buffer list
select-buffer	**^XB**	Switch to a particular buffer
list-buffers	**^X^B**	List all buffers
delete-buffer	**^XK**	Delete a particular buffer if it is off-screen

Chapter 9

Modes

EMACS allows you to change the way it works in order to customized it to the style of editing you are using. It does this by providing a number of different **modes**. These modes can effect either a single buffer, or any new buffer that is created. To add a mode to the current buffer, type **^XM** *add-mode*. EMACS will then prompt you for the name of a mode to add. When you type in a legal mode name, and type a <NL>, EMACS will add the mode name to the list of current mode names in the mode line of the current buffer.

To remove an existing mode, typing the **^X^M** *delete-mode* will cause EMACS to prompt you for the name of a mode to delete from the current buffer. This will remove that mode from the mode list on the current mode line.

Global modes are the modes which are inherited by any new buffers which are created. For example, if you wish to always do string searching with character case being significant, you would want global mode EXACT to be set so that any new files read in inherent the EXACT mode. Global modes are set with the **M-M** *add-global-mode* command, and unset with the **M-^M** *delete-global-mode* command. Also, the current global modes are displayed in the first line of a **^X^B** *list-buffers* command.

On machines which are capable of displaying colors, the mode commands can also set the background and foreground character colors. Using *add-mode* or *delete-mode* with a lowercase color will set the background color in the current window. An uppercase color will set the foreground color in the current window. Colors that EMACS knows about are: white, cyan, magenta, yellow, blue, red, green, and black. If the computer you are running on does not have eight colors, EMACS will attempt to make some intelligent guess at what color to use when you ask for one which is not there.

9.1 ABBREV mode

EMACS can save much tedious typing by automatically expanding predefined abrreviations as you type. Placing EMACS into ABBREV mode tells to to access its table of abbreviations, and look for these abbreviations while you type. When EMACS sees that you have typed one of these, it replaces it with the expanded definition. Information on setting up abbreviation definitions is in chapter 18.

9.2 ASAVE mode

Automatic Save mode tells EMACS to automatically write out the current buffer to its associated file on a regular basis. Normally this will be every 256 characters typed into the file. The environment variable $ACOUNT counts down to the next auto-save, and $ASAVE is the value used to reset $ACOUNT after a save occurs.

9.3 CMODE mode

CMODE is useful to C programmers. When CMODE is active, EMACS will try to assist the user in a number of ways. This mode is set automatically with files that have a .c or .h extension.

The <NL> key will normally attempt to return the user to the next line at the same level of indentation as the last non blank line, unless the current line ends with a open brace ({) in which case the new line will be further indented by one tab position.

A close brace (}) will search for the corresponding open brace and line up with it.

A pound sign (#) with only leading white space will delete all the white space preceding itself. This will always bring preprocessor directives flush to the left margin.

Whenever any close fence is typed, IE)]>}, if the matching open fence is on screen in the current window, the cursor will briefly flash to it, and then back. This makes balancing expressions, and matching blocks much easier.

9.4 CRYPT mode

When a buffer is in CRYPT mode, it is encrypted whenever it is written to a file, and decrypted when it is read from the file. The encryption key can be specified on the command line with the –k switch, or with the **M-E** *set-encryption-key* command. If you attempt to read or write a buffer in crypt mode and now key has not been set, EMACS will execute *set-encryption-key* automatically, prompting you for the needed key. Whenever EMACS prompts you for a key, it will not echo the key to your screen as you type it (IE make SURE you get it right when you set it originally).

The encryption algorithm used changes all characters into normal printing characters, thus the resulting file is suitable for sending via electronic mail. All version of MicroEMACS should be able decrypt the resulting file regardless of what machine encrypted it. Also available with EMACS is the stand alone program, MicroCRYPT, which can en/decrypt the files produced by CRYPT mode in EMACS.

9.5 EXACT mode

When this mode is active, all string searches and replacements will take upper/lower case into account. Normally the case of a string during a search or replace is not taken into account.

9.6 MAGIC mode

In the MAGIC mode certain characters gain special meanings when used in a search pattern. Collectively they are know as regular expressions, and a limited number of them are supported in MicroEmacs. They grant greater flexibility when using the search command. They have no affect on the incremental search command.

The symbols that have special meaning in MAGIC mode are ^, $, ., &, ?, *, +, <, >, [(and], used with it), and \.

The characters ^ and $ fix the search pattern to the beginning and end of line, respectively. The ^ character must appear at the beginning of the search string, and the $ must appear at the end, otherwise they lose their meaning and are treated just like any other character. For example, in MAGIC mode, searching for the pattern “t$” would put the cursor at the end of any line that ended with the letter ‘t’. Note that this is different than searching for “t<NL>”, that is, ‘t’ followed by a newline character. The character $ (and ^, for that matter) matches a position, not a character, so the cursor remains at the end of the line. But a newline is a character that must be matched like any other character, which means that the cursor is placed just after it – on the beginning of the next line.

The character . has a very simple meaning — it matches any single character, except the newline. Thus a search for “bad.er” could match “badger”, “badder” (slang), or up to the ‘r’ of “bad error”.

The character [indicates the beginning of a character class. It is similar to the ‘any’ character ., but you get to choose which characters you want to match. The character class is ended with the character]. So, while a search for “ba.e” will match “bane”, “bade”, “bale”, “bate”, et cetera, you can limit it to matching “babe” and “bake” by searching for “ba[bk]e”. Only one of the characters inside the [and] will match a character. If in fact you want to match any character except those in the character class, you can put a ^ as the first character. It must be the first character of the class, or else it has no special meaning. So, a search for [^aeiou] will match any character except a vowel, but a search for [aeiou^] will match any vowel or a ^.

If you have many characters in order, that you want to put in the character class, you may use a dash (-) as a range character. So, [a-z] will match any letter (or any lower case letter if EXACT mode is on), and [0-9a-f] will match any digit or any letter ‘a’ through ‘f’, which happen to be the characters for hexadecimal numbers. If the dash is at the beginning or end of a character class, it is taken to be just a dash.

The ? character indicates that the preceding character is optional. The character may or may not appear in the matched string. For example, a search for “bea?st” would match both “beast” and “best”. If there is no preceding charcter for ? to modify, it is treated as a normal question mark character.

The * character is known as closure, and means that zero or more of the preceding character will match. If there is no preceding character,
* has no special meaning and is treated as a normal asterisk. The closure symbol will also have no special meaning if it is preceded by the beginning of line symbol ^, since it represents a position, not a character.

The notion of zero or more characters is important. If, for example, your cursor was on the line

```
This line is missing two vowels.
```

and a search was made for “a*”, the cursor would not move, because it is guaranteed to match no letter ‘a’ , which satisfies the search conditions. If you wanted to search for one or more of the letter ‘a’, you could search for “aa*”, which would match the letter a, then zero or more of them. A better way, however, is to use the + character.

The + character behaves in every respect like the * character, with the exception that its minimum match range is one, not zero. Thus the pattern “a+” is identical to “aa*”.

The < and > characters matches a position at the beginning and end of a word respectively. Thus a pattern like \<wo only matches the string “wo” if it is at the beginning of a word.

The \ is the escape character. With the exception of groups, which are explained below, the \ is used at those times when you want to be in MAGIC mode, but also want a regular expression character to be just a character. It turns off the special meaning of the character. So a search for “it\.” will search for a line with “it.”, and not “it” followed by any other character. Or, a search for “TEST*+” would match the word TEST followed by one or more asterisks. The escape character will also let you put ^, –, or] inside a character class with no special side effects.

The character pair \(represent the start of a group in a search string. A group is ended by the character pair \). All characters matched within the \(and \) are part of a numbered group, and may be referenced with the &GROUP function, or with a \ followed by the group number in the replacement string of *replace-string* or the *query-replace-string* commands. For example, a search for “INDEX\([0-9]+\)”, to be replaced by “getind(\1)” would change

```
indptr := INDEX42
```

to

```
indptr := getind(42)
```

.

There may be up to nine groups. Groups may be nested.

The character & (ampersand) is a replacement character, and represents all the characters which were matched by the search string. When used in the **M-R** *replace-string* or the **M-^R** *query-replace-string* commands, the & will be substituted for the search string.

9.7 OVER mode

OVER mode stands for overwrite mode. When in this mode, when characters are typed, instead of simply inserting them into the file, EMACS will attempt to overwrite an existing character past the point. This is very useful for adjusting tables and diagrams.

9.8 WRAP mode

Wrap mode is used when typing in continuous text. Whenever the cursor is past the currently set $fillcol (72 by default) and the user types a space or a <NL>, the last word of the line is brought down to the beginning of the next line. Using this, one just types a continuous stream of words and EMACS automatically inserts <NL>s at appropriate places.

NOTE to programmers:

The EMACS variable $wraphook contains the name of the function which executes when EMACS detects it is time to wrap. This is set to the function *wrap-word* by default, but can be changed to activate different functions and macroes at wrap time.

9.9 VIEW mode

VIEW mode disables all commands which can change the current buffer. EMACS will display an error message and ring the bell every time you attempt to change a buffer in VIEW mode.

Chapter 9 Summary

In Chapter 9 you learned about modes and their effects.

Key Binding	Keystroke	Effect
add-mode	**^XM**	Add a mode to the current buffer
delete-mode	**^X^M**	Delete a mode from the current buffer
add-global-mode	**M-M**	Add a global mode to the current buffer
delete-global-mode	**M-^M**	Delete a global mode from the current buffer

Chapter 10

Files

A file is simply a collection of related data. In EMACS we are dealing with text files — named collections of text residing on a disk (or some other storage medium). You will recall that the major entities EMACS deals with are buffers. Disk-based versions of files are only active in EMACS when you are reading into or writing out of buffers. As we have already seen, buffers and physical files are linked by associated file names. For example, the buffer “ch7.txt” which is associated with the physical disk file “ch7.txt.” You will notice that the file is usually specified by the drive name or (in the case of a hard drive) a path. Thus you can specify full file names in EMACS,

e.g. disk:\directories\filename.extension

If you do not specify a disk and directories, the default disk and the current directory is used.

IMPORTANT — If you do not explicitly save your buffer to a file, all your edits will be lost when you leave EMACS (although EMACS will prompt you when you are about to lose edits by exiting). In addition, EMACS does not protect your disk-based files from overwriting when it saves files. Thus when you instruct EMACS to save a file to disk, it will create a file if the specified file doesn’t exist, or it will overwrite the previously saved version of the file thus replacing it. Your old version is gone forever.

If you are at all unsure about your edits, or if (for any reason) you wish to keep previous versions of a file, you can change the name of the associated file with the command **^XN** *change-file-name*. When this file is saved to disk, EMACS will create a new physical file under the new name. The earlier disk file will be preserved.

For example, let’s load the file **fang.txt** into EMACS. Now, type **^XN**. The EMACS command line prompts “Name:”. Enter a new name for the file — say **new.txt** and press <NL>. The file will be saved under the new filename, and your disk directory will show both **fang.txt** and **new.txt**.

An alternative method is to write the file directly to disk under a new filename. Let’s pull our “publish.txt” file into EMACS. To write this file under another filename, type **^X^W** *write-file*. EMACS will prompt you “write file:”. Enter an alternate filename — **desktop.txt**. Your file will be saved as the physical file “desktop.txt”.

Note that in the examples above, although you have changed the names of the related files, the buffer names remain the same. However, when you pull the physical file back into EMACS, you will find that the buffer name now relates to the filename.

For example — You are working with a buffer “fang.txt” with the related file “fang.txt”. You change the name of the file to “new.txt”. EMACS now shows you working with the buffer “fang.txt” and the related file “new.txt”. Now pull the file “new.txt” into EMACS. Notice that the buffer name has now changed to “new.txt”.

If for any reason a conflict of buffer names occurs,(if you have files of the same name on different drives for example) EMACS will prompt you “use buffer:”. Enter an alternative buffer name if you need to.

For a list of file related commands (including some we‘ve already seen), see the summary page.

Chapter 10 Summary

In Chapter 10 you learned some of the more advanced concepts of file naming and manipulation. The relationship between files and buffers was discussed in some detail.

Key Binding	Keystroke	Effect
save-file	**^X^S**	Saves contents of current buffer with associated filename on default disk/ directory (if not specified)
write-file	**^X^W**	Current buffer contents will be saved under specified name
change-file-name	**^XN**	The associated filename is changed (or associated if not previously specified) as specified
find-file	**^X^F**	Reads specified file into buffer and switches you to that buffer, or switches to buffer in which the file has previously been read
read-file	**^X^R**	Reads file into buffer thus overwriting buffer contents. If file has already been read into another buffer, you will be switched to it
view-file	**^X^V**	The same as read-file except the buffer is automatically put into VIEW mode thus preventing any changes from being made

Chapter 11

Screen Formatting

11.1 Wrapping Text

As we learned in the introduction, EMACS is not a word processor, but an editor. Some simple formatting options are available however, although in most cases they will not affect the appearance of the finished text when it is run through the formatter. We have already encountered WRAP mode which wraps lines longer than a certain length (default is 75 characters). You will recall that WRAP is enabled by entering **^XM** and responding to the command line prompt with **wrap**.

You can also set your own wrap margin by changing the $fillcol variable. Do this by typing ^XA $fillcol <NL> 20. Now the fill column is set at column 20. Now try typing some text. You'll notice that EMACS only allows you to input text in a 20 character wide column!

To reset the wrap column to 72 characters, type ^XA $fillcol <NL> 72. Your text will again wrap at the margin you've been using up to this point.

11.2 Reformatting Paragraphs

After an intensive editing session, you may find that you have paragraphs containing lines of differing lengths. Although this disparity will not affect the formatted text, aesthetic and technical concerns may make it desirable to have consistent paragraph blocks on the screen. If you are in WRAP mode, you can reformat a paragraph with the command **M-Q** *fill-paragraph*. This command 'fills' the current paragraph reformatting it so all the lines are filled and wrap logically.

11.3 Changing Case

There may be occasions when you find it necessary to change the case of the text you've entered. EMACS allows you to change the case of even large amounts of text with ease. Let's try and convert a few of the office traditionalists to the joy of word processing. Type in the following text:

```
Throw away your typewriter and learn to use a word
processor.  Word processing is relatively easy to learn
and will increase your productivity enormously. Enter
the Computer Age and find out just how much fun it can
be!!
```

Let's give it a little more impact by capitalizing the first four words. The first step is to define the region of text just as you would if you were doing an extensive deletion. Set the mark at the beginning of the paragraph with **M-<space>** *set-mark* and move the cursor to the space beyond "typewriter." Now enter **^X^U** *case-region-upper*. Your text should now look like this:

```
THROW AWAY YOUR TYPEWRITER and learn to use a word
processor.  Word processing is relatively easy to learn
and will increase your productivity enormously. Enter
the Computer Age and find out just how much fun it can
be!!
```

If you want to change the text back to lower case, type **^X^L** *case-region-lower*. You can also capitalize individual words. To capitalize the word "fun", position the cursor in front of the word and type **M-U** *case-word-*

upper. The word is now capitalized. To change it ck to lower case, move the cursor back to the beginning of the word and type **M-L** *case-word-lower*.

You may also capitalize individual letters in EMACS. The command **M-C** *case-word-capitalize* capitalizes the first letter after the point. This command would normally be issued with the cursor positioned in front of the first letter of the word you wish to capitalize. If you issue it in the middle of a word, you can end up with some strAnge looking text.

11.4 Tabs

Unless your formatter is instructed to take screen text literally (as MicroSCRIBE does in the 'verbatim' environment for example), tabs in EMACS generally affect screen formatting only.

When EMACS is first started, it sets the default tab to every eighth column. As long as you stay with default, every time you press the tab key a tab character, **^I** is inserted. This character, like other control characters, is invisible — but it makes a subtle and significant difference to your file and editing.

For example, in default mode, press the tab key and then type the word **Test**. "Test" appears at the eighth column. Move your cursor to the beginning of the word and delete the backward character. The word doesn't move back just one character, but flushes to the left margin. The reason for this behavior is easily explained. In tab default, EMACS inserts a 'real' tab character when you press the tab key. This character is inserted at the default position, but NO SPACES are inserted between the tab character and the margin (or previous tab character). As you will recall, EMACS only recognizes characters (such as spaces or letters) and thus when the tab character is removed, the text beyond the tab is flushed back to the margin or previous tab mark.

This situation changes if you alter the default configuration. The default value may be changed by entering a numeric argument before pressing the tab key. As we saw earlier, pressing the **META** key and entering a number allows you to specify how EMACS performs a given action. In this case, let's specify an argument of 10 and hit the tab key.

Now hit the tab key again and type **Test**. Notice the word now appears at the tenth column. Now move to the beginning of the word and delete the backward character. "Test" moves back by one character.

EMACS behaves differently in these circumstances because the **^I** *handle-tab* function deals with tabbing in two distinct ways. In default conditions, or if the numeric argument of zero is used, *handle-tab* inserts a true tab character. If, however, a non-zero numeric argument is specified, *handle-tab* inserts the correct number of spaces needed to position the cursor at the next specified tab position. It does NOT insert the single tab character and hence any editing functions should take account of the number of spaces between tabbed columns.

The distance which a true tab character moves the cursor can be modified by changing the value of the $hardtab environment variable. Initially set to 8, this will determine how far each tab stop is placed from the previous one. (Use the ^XA *set* command to set the value of an environment variable).

Many times you would like to take text which has been created using the tab character and change it to use just spaces. The command **^X^D** *detab-region* changes any tabs in the currently selected region into the right number of spaces so the text does not change. This is very useful for times when the file must be printed or transferred to a machine which does not understand tabs.

Also, the inverse command, **^X^E** *entab-region* changes multiple spaces to tabs where possible. This is a good way to shrink the size of large documents, especially with data tables. Both of these commands can take a numeric argument which will be interpreted as the number of lines to en/detab.

Another function, related to those above is provided for by the **^X^T** *trim-region* when invoked will delete any trailing white space in the selected region. A preceding numeric argument will do this for that number of lines.

Chapter 11 Summary

In Chapter 11 introduced some of the formatting features of EMACS. Text-wrap, paragraph reformatting, and tabs were discussed in some detail. The commands in the following table were covered in the chapter.

Key Binding	Keystroke	Effect
add-mode/WRAP	**^XM**[WRAP]	Add wrap mode to current buffer
delete-mode/WRAP	**^X^M**[WRAP]	Remove wrap mode from current buffer
fill-paragraph	**M-Q**	Logically reformats the current paragraph
case-word-upper	**M-U**	Text from point to end of the current word is changed to uppercase
case-word-lower	**M-L**	Text from point to end of the current word is changed to lowercase
case-word-capitalize	**M-C**	First word (or letter) after the point is capitalized
case-region-upper	**^X^U**	The current region is uppercased
case-region-lower	**^X^L**	The current region is lowercased
handle-tab	**^I**	Tab interval is set to the given numeric argument
entab-region	**^X^E**	Changes multiple spaces to tabs characters where possible
detab-region	**^X^D**	Changes tab characters to the appropriate number of spaces
trim-region	**^X^T**	Trims white space from the end of the lines in the current region

Chapter 12

Undoing the Damage

Often, while editing, we make mistakes. It would be hard not to do so. But when you are in EMACS, most commands can be undone. Each buffer in EMACS stores a list of the changes made to that buffer since the last time it was read from the disk. This list holds every character typed in, every character deleted, and where these occured. If you make a mistake while editing, typing the **^_ undo** command undoes the last command you typed. If you precede this with a numeric argument, it undoes that many commands. For example:

<META>12^_

undoes the last 12 commands.

You can actually look at the list of commands stored to be undone by typing the **^XU list-undos** command. It will place a list of positions, repeat counts, actions and text arguments in a pop-up buffer. This was mostly useful for the authors to debug the undo code, but is interesting to examine. You can see how EMACS records actions, as basic insertions and deletions, seperated by command bounderies.

When undo information is collected by EMACS, it does occupy memory. On most machines this presents no difficulty, but on some machines without virtual memory, and in smaller memory models, *Like real mode under MSDOS*, EMACS will discard the oldest actions in the most ancient visited buffer as it needs more memory.

You can cause EMACS to discard its list of actions to be undone on the current buffer by using the **M-^U delete-undos** command. This can be helpfull in speeding up the operation of a few of the very memory intensive commands. Normally you should not need to use this command, but its use is recommended in macros that manupulate large amounts of text to reclaim the memory used by the undo stack of temporary buffers.

Two environment variables control the actions of the undo mechanism. $undoflag, which defaults to TRUE, can be set to FALSE to disable the undo mechanism altogether. $dispundo directs EMACS to display an additional number on the current modeline which contains the number of primitive actions currently stored in its list of actions for the current buffer.

Chapter 12 Summary

In Chapter 12 We learned how to undo recent editing changes. The commands in the following table were covered in this chapter.

Key Binding	Keystroke	Effect
delete-undos	**M-^U**	Clear out all undo information
list-undos	**^XU**	Pop up a list of all undo information for the current buffer
undo	**^_**	Undo the last editing operation

Chapter 13

Access to the Outside World

EMACS has the ability to interface to other programs and the environment of the computer outside of itself. It does this through a series of commands that allow it to talk to the computer's **command processor** or **shell**. Just what this is varies between different computers. Under MSDOS or PCDOS this is the **command.com** command processor. Under UNIX it is the **csh** shell. On the Atari ST is can be the Mark Williams **MSH** or the Beckmeyer shell. In each case, it is the part of the computer's operating system that is responsible for determining what programs are executed, and when.

The **^X!** *shell-command* command prompts the user for a command line to send out to the shell to execute. This can be very useful for doing file listings and changing the current directory or folder. EMACS gives control to the shell, which executed the command, and then types **[END]** and waits for the user to type a character before redrawing the screen and resuming editing. If the *shell-command* command is used from within the macro language, there is no pause.

^X@ *pipe-command* command allows EMACS to execute a shell command, and if the particular computer allows it, send the results into a buffer which is automatically displayed on the screen. The resulting buffer, called "command" can be manipulated just like any other editing buffer. Text can be copied out of it or rearranged as needed. This buffer is originally created in **VIEW** mode, so remember to **^X^Mview<NL>** in order to change it.

Many computers provide tools which will allow you to **filter** text, making some modifications to it along the way. A very common tool is the **SORT** program which accepts a file, sorts it, and prints the result out. The EMACS command, **^X#** *filter-buffer* sends the current buffer through such a filter. Therefore, if you wished to sort the current buffer on a system which supplied a sort filter, you would type **^X#sort<NL>**. You can also create your own filters by writing programs and utilities which read text from the keyboard and display the results. EMACS will use any of these which would normally be available from the current shell.

If you would like to execute another program directly, without the overhead of an intervening shell, you can use the **^X$** *execute-program* command. It will prompt you for an external program and its arguments and attempt to execute it. Like when EMACS looks for command files, EMACS will look first in the HOME directory, then down the execute PATH, and finally in the current directory for the named program. On some systems, it will automatically tack the proper extension on the file name to indicate it is a program. On some systems that don't support this function, **^X$** will be equivalent to **^X!** *shell-command*.

Sometimes, you would like to get back to the shell and execute other commands, without losing the current contents of EMACS. The **^XC** *i-shell* command shells out of EMACS, leaving EMACS in the computer and executing another command shell. Most systems would allow you to return to EMACS with the "exit" command.

On some systems, mainly advanced versions of UNIX, you can direct EMACS to "go into the background" with the ***^XD*** *suspend-emacs command. This places EMACS in the background returning you to the original command shell. EMACS can then be returned to at any time with the "fg" foreground command.*

Chapter 13 Summary

In Chapter 13 introduced different ways to access the computers shell or command processor from within EMACS. The commands in the following table were covered in the chapter.

Key Binding	Keystroke	Effect
execute-program	**^X$**	Execute an external program directly
filter-command	**^X#**	Send the current buffer through a shell filter
i-shell	**^XC**	Escape to a new shell
pipe-command	**^X@**	Send the results of an external shell command to a buffer
shell-command	**^X!**	Execute one shell command
suspend-emacs	**^XD**	Place EMACS in the background (some UNIX systems only)

Chapter 14

Keyboard Macroes

In many applications, you may need to repeat a series of characters or commands frequently. For example, a paper may require the frequent repetition of a complex formula or a long name. You may also have a series of EMACS commands that you invoke frequently. Keyboard macroes offer a convenient method of recording and repeating these commands.

Imagine, for example, you are writing a scholarly paper on *Asplenium platyneuron*, the spleenwort fern. Even the dedicated botanist would probably find it a task bordering on the agonizing to type *Asplenium platyneuron* frequently throughout the paper. An alternative method is ‘record’ the name in a keyboard macro. Try it yourself.

The command **^X(** *begin-macro* starts recording the all the keystrokes and commands you input. After you’ve typed it, enter **Asplenium platyneuron**. To stop recording, type **^X)** *end-macro*. EMACS has stored all the keystrokes between the two commands. To repeat the name you’ve stored, just enter **^XE** *execute-macro*, and the name “Asplenium platyneuron” appears. You can repeat this action as often as you want, and of course as with any EMACS command, you may precede it with a numerical argument to repeat it many times.

Because EMACS records keystrokes, you may freely intermix commands and text. Unfortunately, you can only store one macro at a time. Thus, if you begin to record another macro, the previously defined macro is lost. Be careful to ensure that you’ve finished with one macro before defining another. If you have a series of commands that you would like to ‘record’ for future use, use the procedure facilities detailed in chapter 15.

Chapter 14 Summary

Chapter 14 covered keyboard macroes. You learned how to record keystrokes and how to repeat the stored sequence.

Key Binding	Keystroke	Effect
start-macro	**^X(**	Starts recording all keyboard input
end-macro	**^X)**	Stops recording keystrokes for macro
execute-macro	**^XE**	Entire sequence of recorded keystrokes is replayed

Chapter 15

MicroEMACS Procedures

Procedures, or macroes, are programs that are used to customize the editor and to perform complicated editing tasks. They may be stored in files or buffers and may be executed using an appropriate command, or bound to a particular keystroke. Portions of the standard start-up file are implemented via procedures, as well as the built in help system. The **M-^E** *run* command causes named procedures to be executed. The *execute-file* command allows you to execute a procedure stored in a disk file, and the *execute-buffer* command allows you to execute a procedure stored in a buffer. Procedures are stored for easy execution by executing files that contain the store-procedure command.

In a command file, the *store-procedure* command takes a string argument which is the name of a procedure to store. These procedures than can be executed with the **M-^E** *run* command. Also, giving the name of a stored procedure within another procedure will executed that named procedure as if it had been called up with the *run* command.

Some fairly length examples of MicroEMACS procedures can be seen by examining the standard files that come with EMACS. The **emacs.rc** file (called **.emacsrc**) under UNIX) is the MicroEMACS command file which is executed when EMACS is normally run. It contains a number of different stored procedures along with the lines to setup and display the Function key window and to call up other procedures and command files using function keys.

There are many different aspects to the language within MicroEMACS. Editor commands are the various commands that manipulate text, buffers, windows, et cetera, within the editor. Directives are commands which control what lines get executed within a macro. Also there are various types of variables. Environmental variables both control and report on different aspects of the editor. User variables hold string values which may be changed and inspected. Buffer variables allow text to be placed into variables. Interactive variable allow the program to prompt the user for information. Functions can be used to manipulate all these variables.

15.1 Constants

All constants and variable contents in EMACS are stored as strings of characters. Numbers are stored digit by digit as characters. This allows EMACS to be “typeless”, not having different variables types be legal in different contexts. This has the disadvantage of forcing the user to be more careful about the context of the statements variables are placed in, but in turn gives them more flexibility in where they can place variables. Needless to say, this also allows EMACS’s expression evaluator to be both concise and quick.

Wherever statements need to have arguments, it is legal to place constants. A constant is a double quote character, followed by a string of characters, and terminated by another double quote character. To represent various special characters within a constant, the tilde (~) character is used. The character following the tilde is interpreted according to the following table:

Sequence	Result	
~n		EMACS newline character (breaks lines)
~r	^M	carriage return
~l	^J	linefeed
~~	~	tilde
~b	^H	backspace
~e	^[	escape
~f	^L	formfeed
~t	^I	tab
~"	"	quote

Any character not in the table which follows a tilde will be passed unmodified. This action is similar to the **^Q** *quote-character* command available from the keyboard.

EMACS may use different characters for line terminators on different computers. The ~n combination will always get the proper line terminating sequence for the current system.

The double quotes around constants are not needed if the constant contains no internal white space and it also does not happen to meet the rules for any other EMACS commands, directives, variables, or functions. This is reasonable useful for numeric constants.

15.2 Variables

Variables in MicroEMACS procedures can be used to return values within expressions, as repeat counts to editing commands, or as text to be inserted into buffers and messages. The value of these variables is set using the set **^XA** command. For example, to set the current fill column to 64 characters, the following macro line would be used:

set $fillcol 64

or to have the contents of **%name** inserted at the point in the current buffer, the command to use would be:

insert-string %name

15.2.1 Environmental Variables

"What good is a quote if you can't change it?"

These variables are used to change different aspects of the way the editor works. Also they will return the current settings if used as part of an expression. All environmental variable names begin with a dollar sign ($) and are in lower case.

$abbell	Ring the bell on an expansion of an abbreviation.
$abcap	Match capitols typed in abbreviation while expanding.
$abquick	Expand abbreviations whenever any character is typed, instead of just when whitespace is typed.
$acount	The countdown of inserted characters until the next save-file.
$asave	The number of inserted characters between automatic file-saves in ASAVE mode.
$bufhook	The function named in this variable is run when a buffer is entered. It can be used to implement modes which are specific to a paricular file or file type.
$cbflags	Current buffer attribute flags (See appendix G for details).
$cbufname	Name of the current buffer.
$cfname	File name of the current buffer.
$cmdhook	Name of function to run before accepting a command. This is by default set to *nop*.
$cmode	Integer containing the mode of the current buffer. (See Appendix F for values).
$curchar	Ascii value of the character currently at the point.
$curcol	Current column of point in current buffer.
$curline	Current line of point in current buffer.
$curwidth	Number of columns used currently.
$curwind	Current window number.
$cwline	Current display line in current window.
$debug	Flag to trigger macro debugging.

$deskcolor	Color to use for current desktop, default to BLACK.
$diagflag	If set to TRUE, diagonal dragging of text and mode lines is enabled. If FALSE, text and modelines can only be dragged horizontally or vertically at one time.
$discmd	Controls the echoing of command prompts. Default is TRUE.
$disinp	Controls the echoing of input at the command prompts. Default is TRUE.
$disphigh	If set to TRUE, high-bit characters (single byte characters that are greater than 127 in value) will be displayed in a pseudo-control format. The characters "^!" will lead off the sequence, followed by the character stripped of its high bit. Default is FALSE.
$dispundo	If TRUE, EMACS displays the current depth of the undo stack on the current modeline.
$exbhook	This variable holds the name of a function or macro which is run whenever you are switching out of a buffer.
$exithook	The procedure named in this variable is run when MicroEMACS is about to exit. Setting $gflags bit 4 during the execution of this procedure prevents the exit from occuring.
$fcol	The current line position being displayed in the first column of the current window.
$fillcol	Current fill column.
$flicker	Flicker Flag set to TRUE if IBM CGA set to FALSE for most others.
$fmtlead	lists all formatter command leadin characters. Lines beginning with these characters will be considered the beginning of paragraphs.
$gflags	Global flags controlling some EMACS internal functions (See appendix G for details).
$gmode	Global mode flags. (See Appendix F for values).
$hardtab	Number of spaces between hard tab stops. Normally 8, this can be used to change indentation only within the editor.
$hjump	The number in here tells EMACS how many columns to scroll the screen horizontally when a horizontal scroll is required.
$hscroll	This flag determines if EMACS will scroll the entire current window horizontally, or just the current line. The default value, TRUE, results in the entire current window being shifted left and right when the cursor goes off the edge of the screen.
$kill	This contains the first 127 characters currently in the kill buffer and can be used to set the contents of the kill buffer.
$language	[READ ONLY]Contains the name of the language which the current EMACS's message will display. (Currently EMACS is available in English, French, Spanish, Latin, Portuguese, Dutch, German, Japanese, and Pig Latin).
$lastkey	[READ ONLY]Last keyboard character typed.
$lastmesg	[READ ONLY]Contains the text of the last message which emacs wrote on the command line.
$line	The current line in the current buffer can be retrieved and set with this environment variable.
$lterm	Character(s) to write as a line terminator when writing a file to disk. Default is null, which causes a '\n' character to be written. Not all operating systems support this.
$lwidth	[READ ONLY]Returns the number of characters in the current line.
$match	[READ ONLY]Last string matched in a search.

$modeflag	Determines if mode lines are currently displayed.
$msflag	If TRUE, the mouse (if present) is active. If FALSE, no mouse cursor is displayed, and no mouse actions are taken.
$newscreen	Create a new screen for every newly created buffer. This variable is most usefull when using a windowing system.
$numwind	The number of windows displayed.
$orgrow	The desktop row position of current screen.
$orgcol	The desktop column position of current screen.
$pagelen	The number of screen lines used currently.
$palette	A string used to control the palette register settings on graphics versions. The usual form consists of groups of three octal digits setting the red, green, and blue levels.
$paralead	A string containing all paragraph start characters.
$pending	[READ ONLY]A flag used to determine if there are user keystrokes waiting to be processed.
$popflag	Use pop-up windows. Default is TRUE.
$popwait	When TRUE, popup windows prompt the user to type a key before they disappear. When FALSE, they do not stick around to be seen. Usefull when a pop-up buffer conatins info needed from a macro. Default is TRUE.
$posflag	Display the line and column position on the modeline. Default is FALSE.
$progname	[READ ONLY]Always contains the string “MicroEMACS” for standard MicroEMACS. Could be something else if EMACS is incorporated as part of someone else’s program.
$ram	The amount of remaining memory if MicroEMACS was compiled with RAMSIZE set. A debugging tool.
$readhook	This variable holds the name of a function to execute whenever a file is read into EMACS. Normally, using the standard **emacs.rc** file, this is bound to a function which places EMACS into CMODE if the extension of the file read is .c or .h.
$region	Contains the string of the current region. It will truncate at the stringsize limit, 255.
$replace	The current replace pattern used in replace commands.
$rval	This contains the return value from the last subprocess which was invoked from EMACS.
$scrname	The current screen name.
$search	The current search pattern used in search and replace commands.
$searchpnt	Set the placement of the of the cursor on a successful search match. $searchpnt = 0 (the default), causes the cursor to be placed at the end of the matched text on forward searches, and at the beginning of the text on reverse searches. $searchpnt = 1 causes the cursor to be placed at the the beginning of the matched text regardless of the search direction, while $searchpnt = 2 causes the cursor to be placed at the end.
$seed	Integer seed of the random number generator.
$softtab	Number of spaces inserted by EMACS when the handle-tab command (which is normally bound to the TAB key) is invoked.
$sres	Current screen resolution (CGA, MONO, EGA or VGA on the IBM-PC driver. LOW, MEDIUM, HIGH or DENSE on the Atari ST1040, NORMAL on most others).

$ssave	A variable which flags EMACS's method of saving files. If set to TRUE, EMACS will write all files out to a temporary file, delete the original, then rename the temporary to the old file name. The default value of this is TRUE.
$sscroll	When set to TRUE, EMACS will smoothly scroll windows one line at a time when cursoring off the ends of the current window. Default is FALSE.
$status	[READ ONLY]Status of the success of the last command (TRUE or FALSE). This is usually used with !force to check on the success of a search, or a file operation.
$sterm	This is the character used to terminate search string inputs. The default for this is the last key bound to *meta-prefix*.
$target	Current target for line moves (setting this fools EMACS into believing the last command was a line move).
$time	[READ ONLY]Contains a string corresponding to the current date and time. Usually this is in a form similar to "Mon May 09 10:10:58 1988". Not all operating systems will support this.
$timeflag	Flag to determine if the time of day is displayed on the modeline. Default is FALSE. The time is updated only after a keystroke.
$tpause	Controls the length of the pause to display a matched fence when the current buffer is in CMODE and a close fence has been typed.
$undoflag	Enable collection of undo information and undo commands.
$version	[READ ONLY]Contains the current MicroEMACS version number.
$wchars	When set, MicroEMACS uses the characters listed in it to determine if it is in a word or not. If it is not set (the default), the characters it uses are the upper and lower case letters, and the underscore.
$wline	Number of display lines in current window.
$wraphook	This variable contains the name of an EMACS function which is executed when a buffer is in WRAP mode and it is time to wrap. By default this is bound to *wrap-word*.
$writehook	This variable contains the name of an EMACS function or macro which is invoked whenever EMACS attempts to write a file out to disk. This is executed before the file is written, allowing you to process a file on the way out.
$xpos	The column the mouse was at the last mouse button press.
$yankflag	Controls the placement of the cursor after a yank command or an insert. When $yankflag is FALSE (the default), the cursor is placed at the end of the yanked or inserted text. When it is TRUE, the cursor remains at the start of the text.
$ypos	The line which the mouse was on during the last mouse button press.

15.2.2 User variables

User variables allow you to store strings and manipulate them. These strings can be pieces of text, numbers (in text form), or the logical values **TRUE** and **FALSE**. These variables can be combined, tested, inserted into buffers, and otherwise used to control the way your macroes execute. At the moment, up to 512 user variables may be in use in one editing session. All users variable names must begin with a percent sign (%) and may contain any printing characters. Only the first 16 characters are significant (IE differences beyond the sixteenth character are ignored). Most operators will truncate strings to a length of 128 characters.

15.2.3 Buffer Variables

Buffer variables are special in that they can only be queried and cannot be set. What buffer variables are is a way to take text from a buffer and place it in a variable. For example, if I have a buffer by the name of RIGEL2, and it contains the text:

```
Richmond
Lafayette
<*>Bloomington          (where <*> is the current point)
Indianapolis
Gary
=* MicroEMACS 4.0 (WRAP) == rigel2 == File: /data/rigel2.txt =====
```

and within a command I reference #rigel2, like:

```
insert-string #rigel2
```

MicroEMACS would start at the current point in the RIGEL2 buffer and grab all the text up to the end of that line and pass that back. Then it would advance the point to the beginning of the next line. Thus, after our last command executes, the string “Bloomington” gets inserted into the current buffer, and the buffer RIGEL2 now looks like this:

```
Richmond
Lafayette
Bloomington
<*>Indianapolis          (where <*> is the current point)
Gary
=* MicroEMACS 4.0 (WRAP) == rigel2 == File: /data/rigel2.txt =====
```

as you have probably noticed, a buffer variable consists of the buffer name, preceded by a pound sign (#).

15.2.4 Interactive variables

Interactive variables are actually a method to prompt the user for a string. This is done by using an at sign (@) followed either with a quoted string, or a variable containing a string. The string is the placed on the bottom line, and the editor waits for the user to type in a string. Then the string typed in by the users is returned as the value of the interactive variable. For example:

```
set %quest "What file? "
find-file @%quest
```

will ask the user for a file name, and then attempt to find it. Note also that complex expressions can be built up with these operators, such as:

```
set %default "file1"
@&cat &cat "File to decode[" %default "]: "
```

which prompts the user with the string:

```
File to decode[file1]:
```

15.3 Functions

Functions can be used to act on variables in various ways. Functions can have one, two, or three arguments. These arguments will always be placed after the function on the current command line. For example, if we wanted to increase the current fill column by two, using emacs’s set (^XA) command, we would write:

```
set $fillcol &add $fillcol 2
 \       \       \       \     \____second operand
  \       \       \       \_________first operand
   \       \       \________________function to execute
    \       \_________________________variable to set
     \_________________________________set (^XA) command
```

Function names always begin with the ampersand (&) character, and are only significant to the first three characters after the ampersand. Functions will normal expect one of three types of arguments, and will automatically convert types when needed. Different argument types include:

<num> an ascii string of digits which is interpreted as a numeric value. Any string which does not start with a digit or a minus sign (-) will be considered zero.

<str> An arbitrary string of characters. At the moment, strings are limited to 128 characters in length.

<log> A logical value consisting of the string “TRUE” or “FALSE”. Numeric strings will also evaluate to “FALSE” if they are equal to zero, and “TRUE” if they are non-zero. Arbitrary text strings will have the value of “FALSE”.

A list of the currently available functions follows. Functions are always used in lower case, the uppercase letters in the function table are the short form of the function (IE &div for &divide).

Numeric Functions: (returns <num>)

&ADD	<num> <num>	Add two numbers
&SUB	<num> <num>	Subtract the second number from the first
&TIMes	<num> <num>	Multiply two numbers
&DIVide	<num> <num>	Divide the first number by the second giving an integer result
&MOD	<num> <num>	Return the reminder of dividing the first number by the second
&NEGate	<neg>	Multiply the arg by -1
&LENgth	<str>	Returns length of string
&SINdex	<str1> <str2>	Finds the position of <str2> within <str1>. Returns zero if not found.
&ASCii	<str>	Return the ascii code of the first character in <str>
&RND	<num>	Returns a random integer between 1 and <num>
&ABS	<num>	Returns the absolute value of <num>
&BANd	<num> <num>	Bitwise AND function
&BOR	<num> <num>	Bitwise OR function
&BXOr	<num> <num>	Bitwise XOR function
&BNOt	<num>	Bitwise NOT function

String manipulation functions: (returns <str>)

&ABBREV	<str>	return the expansion of <str>
&CAT	<str> <str>	Concatenate the two strings to form one
&LEFt	<str> <num>	return the <num> leftmost characters from <str>
&RIGht	<str> <num>	return the <num> rightmost characters from <str>
&MID	<str> <num1> <num2>	Starting from <num1> position in <str>, return <num2> characters.
&REVerse	<str>	return a string with reversed-ordered characters
&UPPer	<str>	Uppercase <str>
&LOWer	<str>	Lowercase <str>
&CHR	<num>	return a string with the character represented by ascii code <num>
&GTC		returns a string of characters

		containing a EMACS command input from the user
&GTK		return a string containing a single keystroke from the user
&ENV	<str>	If the operating system is capable, this returns the environment string associated with <str>
&BIND	<str>	return the function name bound to the keystroke <str>
&XLATE	<str1> <str2> <str3>	
&FINd	<str>	Find the named file <str> along the path and return its full file specification or an empty string if none exists
&TRIM	<str>	Trim the trailing whitespace from a string

Logical Testing functions: (returns <log>)

&NOT	<log>	Return the opposite logical value
&AND	<log1> <log2>	Returns TRUE if BOTH logical arguments are TRUE
&OR	<log1> <log2>	Returns TRUE if either argument is TRUE
&EQUal	<num> <num>	If <num> and <num> are numerically equal, return TRUE
&LESs	<num1> <num2>	If <num1> is less than <num2>, return TRUE.
&GREater	<num1> <num2>	If <num1> is greater than <num2>, return TRUE.
&SEQual	<str1> <str2>	If the two strings are the same, return TRUE.
&SLEss	<str1> <str2>	If <str1> is less alphabetically than <str2>, return TRUE.
&SGReater	<str1> <str2>	If <str1> is alphabetically greater than or equal to <str2>, return TRUE.
&EXIst	<str>	Does the named file <str> exist?
&ISNum	<num>	Is the given argument a legitimate number?

Special Functions:

&GROup	<num>	Return group <num> as set by a MAGIC mode search.
&SUPper	<str1> <str2>	Translate the first char in <str1> to the first char in <str2> when uppercasing.
&SLOwer	<str1> <str2>	Translate the first char in <str1> to the first char in <str2> when lowercasing.
&INDirect	<str>	Evaluate <str> as a variable.

This last function deserves more explanation. The &IND function evaluates its argument, takes the resulting string, and then uses it as a variable name. For example, given the following code sequence:

```
; set up reference table

set %one        "elephant"
set %two        "giraffe"
set %three      "donkey"

set %index "%two"
insert-string &ind %index
```

the string “giraffe” would have been inserted at the point in the current buffer. This indirection can be safely nested up to about 10 levels.

15.4 Directives

Directives are commands which only operate within an executing procedure, IE they do not make sense as a single command. As such, they cannot be called up singly or bound to keystroke. Used within command files, they control what lines are executed and in what order.

Directives always start with the exclamation mark (!) character and must be the first non-white space placed on a line. Directives executed interactively (via the execute-command-line command) will be ignored.

15.4.1 !ENDM Directive

This directive is used to terminate a procedure or macro being stored. For example, if a file is being executed contains the text:

```
;       Read in a file in view mode, and make the window red

store-procedure get-red-viewed-file
        find-file @"File to view: "
        add-mode "view"
        add-mode "red"
!endm

print "[Consult procedure has been loaded]"
```

only the lines between the store-macro command and the !ENDM directive are stored in procedure get-red-viewd-file. Both named procedures and numbered macroes (via the *store-macro* command) should be terminated with this directive.

15.4.2 !FORCE Directive

When MicroEMACS executes a procedure, if any command fails, the procedure is terminated at that point. If a line is preceded by a !FORCE directive, execution continues whether the command succeeds or not. For example:

```
;       Merge the top two windows

save-window             ;remember what window we are at
1 next-window           ;go to the top window
delete-window           ;merge it with the second window
!force restore-window   ;This will continue regardless
add-mode "red"
```

Often this is used together with the $status environment variable to test if a command succeeded. For example:

```
set %seekstring "String to Find: "
!force search-forward %seekstring
!if &seq $status TRUE
        print "Your string is Found"
!else
        print "No such STRING!"
!endif
```

15.4.3 !IF, !ELSE, and !ENDIF Directives

This directive allows statements only to be executed if a condition specified in the directive is met. Every line following the !IF directive, until the first !ELSE or !ENDIF directive, is only executed if the expression following the !IF directive evaluates to a TRUE value. For example, the following commands creates the portion of a text file automatically. (yes believe me, this will be easier to understand then that last explanation….)

```
!if &sequal %curplace "timespace vortex"
```

```
        insert-string "First, rematerialize~n"
!endif
!if &sequal %planet "earth"             ;If we have landed on earth...
        !if &sequal %time "late 20th century"  ;and we are then
                write-message "Contact U.N.I.T."
        !else
                insert-string "Investigate the situation....~n"
                insert-string "(SAY 'stay here Sara')~n"
        !endif
!else
        set %conditions @"Atmosphere conditions outside? "
        !if &sequal %conditions "safe"
                insert-string &cat "Go outside....." "~n"
                insert-string "lock the door~n"
        !else
                insert-string "Dematerialize..try somewhen else"
                newline
        !endif
!endif
```

15.4.4 !GOTO Directive

Flow can be controlled within a MicroEMACS procedure using the !GOTO directive. It takes as an argument a label. A label consists of a line starting with an asterisk (*) and then an alphanumeric label. Only labels in the currently executing procedure can be jumped to, and trying to jump to a non-existing label terminates execution of a procedure. For example:

```
;Create a block of DATA statements for a BASIC program

        insert-string "1000 DATA "
        set %linenum 1000

*nxtin
        update-screen                   ;make sure we see the changes
        set %data @"Next number: "
        !if &equal %data 0
                !goto finish
        !endif

        !if &greater $curcol 60
                2 delete-previous-character
                newline
                set %linenum &add %linenum 10
                insert-string &cat %linenum " DATA "
        !endif

        insert-string &cat %data ", "
        !goto nxtin

*finish

        2 delete-previous-character
        newline
```

15.4.5 !WHILE and !ENDWHILE Directives

This directive allows you to set up repetitive tasks easily and efficiently. If a group of statements need to be executed while a certain condition is true, enclose them with a while loop. For example,

```
!while &less $curcol 70
        insert-string &cat &cat "[" #stuff "]"
!endwhile
```

places items from buffer “item” in the current line until the cursor is at or past column 70. While loops may be nested and can contain and be the targets of !GOTOs with no ill effects. Using a while loop to enclose a repeated task will run much faster than the corresponding construct using !IFs.

15.4.6 !BREAK Directive

This lets you abort out of the most executing currently inner while loop, regardless of the condition. It is often used to abort processing for error conditions. For example:

```
;       Read in files and substitute "begining" with "beginning"

        set %filename #list
        !while &not &seq %filename "<end>"
!force          find-file %filename
                !if &seq $status FALSE
                        write-message "[File read error]"
                        !break
                !endif
                beginning-of-file
                replace-string "begining" "beginning"
                save-file
                set %filename #list
        !endwhile
```

This while loop will process files until the list is exhausted or there is an error while reading a file.

15.4.7 !RETURN Directive

The !RETURN Directive causes the current procedure to exit, either returning to the caller (if any) or to interactive mode. For example:

```
        ;       Check the monitor type and set %mtyp

        !if &sres "CGA"
                set %mtyp 1
                !return
        !else
                set %mtyp 2
        !endif

        insert-string "You are on a MONOCHROME machine!~n"
```

This directive can also allow the currently executing procedure to return a value which is automatically placed in the $rval environment value and passed back as the procedure’s value if the procedure is called by the &call function. For example:

```
store-procedure squared %arg

        !return &times %arg %arg
!endm

        set %a &call squared 6
```

This code sets the value of user variable %a to 36, the square of 6. The expresion after the !RETURN directive is evaluated as the value returned by the procedure when used as an argument to the &call function.

15.5 Local and Global Variables

Most user variables are considered **global**. That means that once they are defined, they can be referenced and changed from any procedure or interactively. If you wish a variable to be defined only within the currently executing

procedure, you can declare it as **local**. Local variables only can be seen and changed within the procedure where they have been defined. To declare a local variable, use the **local** keyword.

```
local %tempvar
```

This declares %tempvar as a local user variable within the procedure that the statement executes. Local declarations happen when the statement executes, not automatically when the procedure is entered, so make sure to put the local statement before any line that references that user variable. If the name of a user variable matches an already declared variable, the local variable overides the global variable while still in the function where it is declared.

There is a **global** keyword which can be used to declare a user variable to be global, and to initialize its value to an empty string. Since user variables are created on the fly when they are first referenced, this statement is not strictly needed, but is provided for documentation purposes.

15.6 Parameters to Procedures

When you store a procedure, you can follow the procedure name with a list of parameters. These parameters are treated as local variables to the procedure, and their initial values are taken from the list of expresions following the call to that procedure. For example:

```
store-procedure to-the-power %base %exp

        local %result

        set %result %base
        !while &gre %exp 1
                set %result &times %result %base
                set %exp &sub %exp 1
        !endwhile
        !return %result
!endm
insert-string to-the-power 2 8
```

When this example is executed, the number 256 (which is 2^8) is inserted into the current position of the current buffer. The function perameters are treated just like local variables in scope.

Chapter 16

Debugging MicroEMACS Procedures

When developing new procedures, it is very convenient to be able to trace their execution to discover errors. The $debug environment variable enables procedure debugging. While this variable is TRUE, emacs will stop at each line it intends to execute and allow you to view it, and issue a number of different commands to help determine how the procedure is executing.

For example, we will step through the procedure which toggles the function key window off. The first thing to do, is to set $debug, using the ^**XA** *set* command. Type ^XA and emacs will prompt you on the command line with "Variable to set: ". Type in "$debug" and press the enter key. Emacs will then ask "Value: ". Type in "TRUE" (in capital letters) and press the enter key.

While macro debugging is enabled (as it is now) emacs will report each time a variable is assigned a value, by displaying the variable and its value on the command line. Right now,

(($debug <- TRUE))

appears on the command line to tell you that $debug now has been assigned the value of TRUE. Press the space bar to continue.

Now, lets try to debug a macro. Press function key 5 which normally toggles the function key window. The first thing that appears is:

«<[toggle-fkeys]:!if %rcfkeys»>

At this point, emacs is waiting for a command. It is prepared to see if the user variable %rcfkeys is TRUE, and execute some lines if they are. Suppose we want to see the value of this variable, type the letter "e" to evaluate an expression. Emacs will prompt with "EXP: ". Type "%rcfkeys" followed by the enter key. Emacs should then respond with "TRUE" to indicate that the function key window is currently on screen.

Press the space bar to allow the !if directive to execute. Emacs will decide that it is TRUE, and then display the next command to execute.

«<[toggle-fkeys]:!goto rcfoff»>

Notice emacs tells us what procedure we are currently executing (in this case, the macro bound to toggle-fkeys). Press the space bar again to execute the !goto directive.

«<[toggle-fkeys]:save-window»>

Emacs is saving the position of the current window so that it can attempt to return to it after it has brought up the function key window. Press the space bar to continue. Now emacs displays:

«<[toggle-fkeys]:1 next-window»>

The *next-window* command moves the current point to the next window down, or when it has an argument like now, to that window from the top. This puts the point in the top window, the one with the function key window in it.

Sometimes when debugging we will want to keep track of the value of a variable or expresion. To do this, type a 't' character to **t**rack an expression. For now, type n $numwind and hit the return. The display on the command line changes to:

«<[=2][toggle-fkeys]:1 next-window»>

The expresion we typed, in this case the environment variable $numwind, is currently evaluated as 2. Press space to execute the *next-window* command. Now the command line shows:

«<[=2][toggle-fkeys]:!if &sequal “Function Keys” $cbufname»>

This macro is now checking to see if the current buffer is really named “Function Keys”. If it is not, then it does not want to delete that window. But since the function window is displayed at the moment, and is at the top of the screen, this statement is true. So we press the space to go on.

«<[=2][toggle-fkeys]:delete-window»>

We press the space again and let emacs delete the function key window. Notice the value of $numwind changes to 1 with one less window beying displayed.

«<[=1][toggle-fkeys]:!endif»>

We are at the end of the conditional code. Press space again to see:

«<[=1][toggle-fkeys]:!force restore-window»>

This restores the point to the window we started in. This command could have failed if the cursor started in the function key window, and that would have normally stopped the macro in its tracks. But the !force directive tells it to ignore any failures. Press the space again:

«<[=1][toggle-fkeys]:write-message “[Function key window OFF]”»>

Well, this lets the user know what we have done, and that we are done. Press the space again.

«<[=1][toggle-fkeys]:set %rcfkeys FALSE»>

Lastly, we set the user variable that the macro uses to tell if it should be turning the function key window on or off to FALSE, letting it know that it is off. When we press the space bar for the last time, the command line shows that variable being set to FALSE.

Anytime while we were debuging, we can hit ? to list the different keystrokes that are legal when debugging. The ^G key will stop a macro immediatly.

If a macro is executing and hits an error (without the !force directive overriding the error), it stops and prints a list of all the macroes it is in the middle of executing, and the lines that those macroes are executing at. You can then go back and examine the offending statement and correct it.

While you are debugging, the ‘c’ key lets you execute any emacs command interactively. Be carefull with this, you can easily confuse emacs as to where it is or what it is doing by changing things in the middle of a macro. But this is very usefull for testing out changes in variables or the position of the cursor, or any number of environment variables controlling emacs while your macro is running.

Lastly, hitting the current META key while debugging sets $debug to FALSE and finishes running your macro with no more debugging.

Chapter 17

Key Bindings, What they are and why

One of the features which makes MicroEMACS very adaptable is its ability to use different keystrokes to execute different commands. The process of changing the particular command a key invokes is called *rebinding*. This allows us to make the editor look like other popular editors and programs.

Each command in MicroEMACS has a name which is used for binding purposes. For example, the command to move the cursor down one page is called *next-line* and is normally bound to the ^N key. If you decided that you also wanted to use the ^D key to move the cursor down one line, you would use the M-K *bind-to-key* command. EMACS would respond with ": bind-to-key " on the command line and allow you to type in a command name. Then type in the name of the command you want to change, in this case *next-line*, followed by the <NL> key. EMACS will then wait for you to type in the keys you want to activate the named function. Type a single ^D. From now on, typing ^D will cause EMACS to move down one line, rather than its original function of deleting characters.

To find out the name of a command, consult the list of valid EMACS commands in Appendix B. Also, you can use the ^X? *describe-key* command to look up the name of a command. Type ^X? and then the key to use that command, and EMACS will show you the name of the command.

After you have experimented with changing your key bindings, you may decide that you want to change some bindings permanently. To have EMACS rebind keys to your pleasure each time you start EMACS, you can add statements to the end of your startup file (**emacs.rc** or **.emacsrc** depending on the system). For example,

bind-to-key next-line ^D

Notice, that control D character in the startup file is represented visibly as an uparrow key followed by a capital D. To know how to represent any keys you want to bind, use the *describe-key* command on the key, and use the sequence that is displayed.

bind-to-key split-current-window FN1

This example would make function key 1 activate the command that splits the current window in two.

You can also bind your own macroes to keys. You do this by using the ^X^K **macro-to-key** command. This works just like the *bind-to-key* command, but binds that key to your own macro,

```
store-procedure insert-stuff
        insert-string "<Page Header>            page
<nn>~n"
!endm
macro-to-key insert-stuff A-P
```

This example when run causes emacs to insert a page header at the current point whenever the ALT-E combination is typed.

EMACS will let you define a large number of keys, but will report "Binding table FULL!" when it runs out of space to bind keys. Normally EMACS will allow up to 512 key bindings (including approx. 300 originally bound keys).

If you want to get a current listing of all the commands and the keys bound to them, use the *describe-bindings* command. Notice, that this command is not bound to any keys!

There are some key bindings that cannot be made without special precautions. Alternative bindings for ^X, META, ^G, and ^U (which bind respectively to ctlx-prefix, meta-prefix, abort-command, and universal-argument) must be made before re-binding ^X, META, ^G, or ^U. The reason is to protect the innocent user from losing the prefix and other commands inadvertantly.

Chapter 17 Summary

In chapter 17, you learned how change what commands and macroes are executed when a particular key is typed.

Key Binding	Keystroke	Effect
bind-to-key	M-K	Associate a particular keystroke with an emacs built in command.
macro-to-key	^X^K	Associate a particular keystroke with a user written macro.

Chapter 18

Abbreviations and Their Definitions

If you have a lot of repetitive typing of large terms, names or strings of any kind, EMACS can make this simpler by looking for simple words or abbreviations that expand to these larger strings. To use this feature, you need to place the buffer you are typing in into **ABBREV** mode, and you need to define a table of the abbreviations to expand.

The *add-abbrev* command lets you add an abbreviation EMACS's current table of abbreviations. This command is usually used as part of a procedure, and can be conviniently placed at the end of your emacs.rc startup file. Here is an example abbreviation definition:

add-abbrev "um" "MicroEMACS"

When you type *um* into a buffer, it is instantly replaced with the word *MicroEMACS*. Another way to define a large number of abbreviations at one time is to place them in a buffer, one to a line, with the first space or tab seperating the abbreviation from its definition. Then use the *define-abbrevs* command, giving it the name of that buffer, and it will memorize all the definitions in that buffer.

You can always look at the list of currently defined abbreviations by using the **describe-abbrevs** command. If you wish to create a buffer to be used to define abbreviations, the *insert-abbrevs* command inserts a table of them into the current buffer at the point. And since these abbreviations take up memory, you can dispose of the lot by issuing the *kill-abbrevs* command.

If you want EMACS to beep at you when it expands an abbreviation, set the $abbell environment variable to TRUE. If you wish EMACS to match the definition to your capitalization of the typed in abbreviation, set $abcap to TRUE.

Normally EMACS will only attempt an abbreviation when it thinks that you have finished typing a word. This means these only happen when you type white space, or the return key. If you want it to be more agressive and attempt completions when the last letter of the abbreviation are typed, set $abquick to TRUE.

Inside your own preocedures, the &abbrev function will return the definition of any abbreviation given it as an argument.

Chapter 18 Summary

In chapter 18, you learned how to set up abbreviations and their definitions, and how to control their use. Here is a table of the commands covered in this chapter and their corresponding key bindings:

Key Binding	Keystroke	Effect
add-abbrev	(none)	Add an abbreviation to the global table of abbreviations
delete-abbrev	(none)	Delete an abbreviation
kill-abbrevs	(none)	Kill all abbreviations
define-abbrevs	(none)	Add abbreviations in <buffer> to the global abbreviation table
insert-abbrevs	(none)	Insert the current global abbreviation table at the point
describe-abbevs	(none)	Pop-up a list of defined abbreviations

Appendix A

MicroEMACS Command Line Switches and Startup Files

When EMACS first executes, it always searches for a file, called **.emacsrc** *under most UNIX systems* or **emacs.rc** *on most other systems* which it will execute as EMACS macroes before it reads in the named source files. This file normally contains EMACS macroes to bind the function keys to useful functions and load various useful macroes. The contents of this file will probably vary from system to system and can be modified by the user as desired.

When searching for this file, EMACS looks for it in this order. First, it attempts to find a definition for "**HOME**" in the environment. It will look in that directory first. Then it searches all the directories listed in the "**PATH**" environment variable. Then it looks through a list of predefined standard directories which vary from system to system. Finally, failing all of these, it looks in the current directory. This is also the same method EMACS uses to look up any files to execute, and to find it's help file **EMACS.HLP**.

On computers that call up EMACS via a command line process, such as MSDOS and UNIX, there are different things that can be added to the command line to control the way EMACS operates. These can be switches, which are a dash ('-') followed by a letter, and possible other parameters, or a startup file specifier, which is an at sign '@' followed by a file name.

@<file>	This causes the named file to be executed instead of the standard emacs.rc file before emacs reads in any other files. More than one of these can be placed on the command line, and they will be executed in the order that they appear.
–C	The following source files on the command line can be changed (as opposed to being in VIEW mode). This is mainly used to cancel the effects of the –v switch used previously in the same command line.
–E	This flag causes emacs to automatically run the startup file "error.cmd" instead of emacs.rc. This is used by various C compilers for error processing (for example, Mark Williams C).
–G<num>	Upon entering EMACS, position the cursor at the <num> line of the first file.
–I<var> <value>	Initialize an EMACS variable with <value>. This can be useful to force EMACS to start in a particular mode. (For example, invoke EMACS with "emacs –i$sres VGA foo.bar" to edit file foo.bar in VGA 50 line mode on an IBM-PC).
–K<key>	This key tells emacs to place the source files in CRYPT mode and read it in using <key> as the encryption key. If no key is listed immediately after the –K switch, EMACS will prompt for a key, and not echo it as it is typed.
–R	This places EMACS in "restricted mode" where any commands allowing the user to read or write any files other than the ones listed on the command line are disabled. Also all commands allowing the user access to the operating system are disabled. This makes EMACS very useful as a "safe" environment for use within other applications and especially used as a remote editor for a BBS or electronic bulletin board system.
–S<string>	After EMACS is started, it automatically searches for <string> in the first source file.
–V	This tells EMACS that all the following sources files on the command line should be in VIEW mode to prevent any changes being made to them.

Appendix B

Command Completion

Some versions of MicroEMACS will allow you to abbrieviate buffer names, command names and file names as you enter them. To use this, type in the first few characters of the name you wish, and then hit either the space bar, the META key or the TAB key. MicroEMACS will then attempt to look at the list of all the availible names and if there is only one which will fit, it will choose that name. If there are several names that quailify, as many characters as are common to ALL of them will be entered. If there are no possible matches, the bell will ring to indicate MicroEMACS can not complete the command.

For example, if you have several files in your current directory with the following names:

prog1.c
prog1.obj
prog1.exe
prog1.doc
program.one
project.one
test.c
tes

and you enter the **^X^F** *find-file* command, if you type ‘p’ and then hit the space bar, EMACS will respond by typing the ‘r’ that is common to all the above file names begining with ‘p’. If you then type ‘ogr’ and hit the tab key, EMACS will respond with ‘am.one’ and automatically hit the enter key for you.

If you were to instead type an ‘a’ and hit the space bar, EMACS will beep, informing you that there is no possible match.

If you type a ‘te’ and hit the space bar, EMACS will then type the following ‘s’, but it will not automatically enter it because it is possible you mean to get to the test.c file.

Buffer name, and command name completion is available in all versions of MicroEMACS. File name completion is available on UNIX BSD4.3, the Atari ST, the AMIGA and under MSDOS.

Appendix C

MicroEMACS Commands

Below is a complete list of the commands in EMACS, the keys normally used to do the command, and what the command does. Remember, on some computers there may also be additional ways of using a command (cursor keys and special function keys for example).

Command	Binding	Meaning
abort-command	^G	This allows the user to abort out of any command that is waiting for input
add-mode	^XM	Add a mode to the current buffer
add-global-mode	M-M	Add a global mode for all new buffers
append-file	^X^A	Write a buffer to the end of a file
apropos	M-A	List out commands whose name contains the string specified
backward-character	^B	Move one character to the left
begin-macro	^X(	Begin recording a keyboard macro
beginning-of-file	M-<	Move to the beginning of the file in the current buffer
beginning-of-line	^A	Move to the beginning of the current line
bind-to-key	M-K	Bind a key to a function
buffer-position	^X=	List the position of the cursor in the current window on the command line
case-region-lower	^X^L	Make a marked region all lower case
case-region-upper	^X^U	Make a marked region all upper case
case-word-capitalize	M-C	Capitalize the following word
case-word-lower	M-L	Lower case the following word
case-word-upper	M-U	Upper case the following word
change-file-name	^XN	Change the name of the file in the current buffer
change-screen-size	(none)	Change the number of lines of the screen currently being used
change-screen-width	(none)	Change the number of columns of the screen currently being used
clear-and-redraw	^L	Clear the physical screen and redraw it
clear-message-line	(none)	Clear the command line
copy-region	M-W	Copy the currently marked region into the kill buffer

count-words	M-^C	Count how many words, lines and characters are in the current marked region
ctlx-prefix	^X	Change the key used as the ^X prefix
cycle-screens	A-C	Bring the rearmost screen to front
delete-blank-lines	^X^O	Delete all blank lines around the cursor
delete-buffer	^XK	Delete a buffer which is not being currently displayed in a window
delete-mode	^X^M	Turn off a mode in the current buffer
delete-global-mode	M-^M	Turn off a global mode
delete-next-character	^D	Delete the character following the cursor
delete-next-word	M-D	Delete the word following the cursor
delete-other-windows	^X1	Make the current window cover the entire screen
delete-previous-character	^H	Delete the character to the left of the cursor
delete-previous-word	M-^H	Delete the word to the left of the cursor
delete-screen	A-D	Delete a screen
delete-undos	M-^U	Delete all undo information
delete-window	^X0	Remove the current window from the screen
describe-bindings	(none)	Make a list of all legal commands
describe-functions	(none)	Make a list of all legal functions
describe-variables	(none)	Make a list of all environment and user variables
describe-key	^X?	Describe what command is bound to a keystroke sequence
detab-region	^X^D	Change all tabs in a region to the equivalent spaces
display	^XG	Prompts the user for a variable and displays its current value
dump-variables	none	Places into a buffer the current values of all environment and user variables
end-macro	^X)	stop recording a keyboard macro
end-of-file	M->	Move cursor to the end of the current buffer
end-of-line	^E	Move to the end of the current line
end-of-word	(none)	Move the point just past the end of the current word
entab-region	^X^E	Change multiple spaces to tabs where possible
exchange-point-and-mark	^X^X	Move cursor to the last marked spot,

		make the original position be marked
execute-buffer	(none)	Execute a buffer as a macro
execute-command-line	(none)	Execute a line typed on the command line as a macro command
execute-file	(none)	Execute a file as a macro
execute-macro	^XE	Execute the keyboard macro (play back the recorded keystrokes)
execute-named-command	M-X	Execute a command by name
execute-procedure	M-^E	Execute a procedure by name
execute-program	^X$	Execute a program directly (not through an intervening shell)
exit-emacs	^X^C	Exit EMACS. If there are unwritten, changed buffers EMACS will ask to confirm
fill-paragraph	M-Q	Fill the current paragraph
filter-buffer	^X#	Filter the current buffer through an external filter
find-file	^X^F	Find a file to edit in the current window
find-screen	A-F	Bring the named screen to front, creating it if needed
forward-character	^F	Move cursor one character to the right
goto-line	M-G	Goto a numbered line
goto-mark	M-^G	Goto a numbered mark
goto-matching-fence	M-^F	Goto the matching fence
grow-window	^X^	Make the current window larger
handle-tab	^I	Insert a tab or set tab stops
hunt-forward	A-S	Hunt for the next match of the last search string
hunt-backward	A-R	Hunt for the last match of the last search string
help	M-?	Read EMACS.HLP into a buffer and display it
i-shell	^XC	Shell up to a new command processor
incremental-search	^XS	Search for a string, incrementally
indent-region	M-(	Indent the current region one tab
insert-file	^X^I	insert a file at the cursor in the current file
insert-space	^C	Insert a space to the right of the cursor
insert-string	(none)	Insert a string at the cursor
kill-paragraph	M-^W	Delete the current paragraph

kill-region	^W	Delete the current marked region, moving it to the kill buffer
kill-to-end-of-line	^K	Delete the rest of the current line
label-function-key	(none)	Set the text on a function key label (HP150 only)
list-buffers	^X^B	List all existing buffers
list-screens	A-B	List all existing screens
list-undos	^XU	List current buffer’s undo information
macro-to-key	^X^K	Bind a key to a macro
meta-prefix	<ESC>	Key used to precede all META commands
mouse-move-down	MSa	
mouse-move-up	MSb	
mouse-resize-screen	MS1	
mouse-region-down	MSe	
mouse-region-up	MSf	
move-window-down	^X^N	Move all the lines in the current window down
move-window-up	^X^P	Move all the lines in the current window up
name-buffer	M-^N	Change the name of the current buffer
narrow-to-region	^X<	hides all text not in the current region
newline	^M	Insert a <NL> at the cursor
newline-and-indent	^J	Insert a <NL> at the cursor and indent the new line the same as the preceding line
next-buffer	^XX	Bring the next buffer in the list into the current window
next-line	^N	Move the cursor down one line
next-page	^V	Move the cursor down one page
next-paragraph	M-N	Move cursor to the next paragraph
next-window	^XO	Move cursor to the next window
next-word	M-F	Move cursor to the beginning of the next word
nop	(none)	Does nothing
open-line	^O	Open a line at the cursor
overwrite-string	(none)	Overwrite a string at the cursor
pipe-command	^X@	Execute an external command and place its output in a buffer
pop-buffer	(none)	Display a buffer temporarily, paging

previous-line	^P	Move cursor up one line
previous-page	^Z	Move cursor up one page
previous-paragraph	M-P	Move back one paragraph
previous-window	^XP	Move the cursor to the last window
previous-word	M-B	Move the cursor to the beginning of the word to the left of the cursor
print	(none)	Display a string on the command line (a synonim to write-message)
query-replace-string	M-^R	Replace all of one string with another string, interactively querying the user
quick-exit	M-Z	Exit EMACS, writing out all changed buffers
quote-character	^Q	Insert the next character literally
read-file	^X^R	Read a file into the current buffer
redraw-display	M-^L	Redraw the display, centering the current line
remove-mark	(none)	Remove a numbered mark
resize-window	^XW	Change the number of lines in the current window
restore-window	(none)	Move cursor to the last saved window
replace-string	M-R	Replace all occurrences of one string with another string from the cursor to the end of the buffer
reverse-incremental-search	^XR	Search backwards, incrementally
run	M-^E	Execute a named procedure
save-file	^X^S	Save the current buffer if it is changed
save-window	(none)	Remember current window (to restore later)
scroll-next-up	M-^Z	Scroll the next window up
scroll-next-down	M-^V	Scroll the next window down
search-forward	^S	Search for a string
search-reverse	^R	Search backwards for a string
select-buffer	^XB	Select a buffer to display in the current window
set	^XA	Set a variable to a value
set-encryption-key	M-E	Set the encryption key of the current buffer
set-mark		Set the mark
shell-command	^X!	Causes an external shell to execute a command
show-files	(none)	Pop up a list of files from the

		specified directory
shrink-window	^X^Z	Make the current window smaller
source	(none)	Execute a file as a macro
split-current-window	^X2	Split the current window in two
store-macro	(none)	Store the following macro lines to a numbered macro
store-procedure	(none)	Store the following macro lines to a named procedure
transpose-characters	^T	Transpose the character at the cursor with the character to the left
trim-region	^X^T	Trim any trailing white space from a region
unbind-key	M-^K	Unbind a key from a function
undent-region	M-)	Remove a leading indent from a region
undo	^_	Undo the last editing operation
universal-argument	^U	Execute the following command 4 times
unmark-buffer	M-~	Unmark the current buffer (so it is no longer changed)
update-screen	(none)	Force a screen update during macro execution
view-file	^X^V	Find a file,and put it in view mode
widen-from-region	^X>	restores hidden text (see narrow-to-region)
wrap-word	(none)	Wrap the current word, this is an internal function
write-file	^X^W	Write the current buffer under a new file name
write-message	(none)	Display a string on the command line
yank	^Y	yank the kill buffer into the current buffer at the cursor

Appendix D

MicroEMACS Bindings

Below is a complete list of the key bindings used in MicroEMACS. This can be used as a wall chart reference for MicroEMACS commands.

Default Key Bindings for MicroEmacs 4.0

^A	Move to start of line	ESC A	Apropos (list some commands)
^B	Move backward by characters	ESC B	Backup by words
^C	Insert space	ESC C	Initial capitalize word
^D	Forward delete	ESC D	Delete forward word
^E	Goto end of line	ESC E	Reset Encryption Key
^F	Move forward by characters	ESC F	Advance by words
^G	Abort out of things	ESC G	Go to a line
^H	Backward delete		
^I	Insert tab/Set tab stops		
^J	Insert <NL>, then indent		
^K	Kill forward	ESC K	Bind Function to a key
^L	Refresh the screen	ESC L	Lower case word
^M	Insert <NL>	ESC M	Add global mode
^N	Move forward by lines	ESC N	Goto End paragraph
^O	Open up a blank line		
^P	Move backward by lines	ESC P	Goto Begining of paragraph
^Q	Insert literal	ESC Q	Fill current paragraph
^R	Search backwards	ESC R	Search and replace
^S	Search forward	ESC S	Suspend (BSD only)
^T	Transpose characters		
^U	Repeat command four times	ESC U	Upper case word
^V	Move forward by pages	ESC V	Move backward by pages
^W	Kill region	ESC W	Copy region to kill buffer
^Y	Yank back from killbuffer	ESC X	Execute named command
^Z	Move backward by pages	ESC Z	Save all buffers and exit
^_	Undo last editing operation		

ESC ^C	Count words in region	ESC ~	Unmark current buffer
ESC ^E	Execute named procedure		
ESC ^F	Goto matching fence	ESC !	Reposition window
ESC ^H	Delete backward word	ESC <	Move to start of buffer
ESC ^K	Unbind Key from function	ESC >	Move to end of buffer
ESC ^L	Reposition window	ESC .	Set mark
ESC ^M	Delete global mode	ESC space	Set mark
ESC ^N	Rename current buffer	ESC rubout	Delete backward word
ESC ^R	Search & replace w/query	rubout	Backward delete
ESC ^S	Source command file		
ESC ^U	Delete undo information		
ESC ^V	Scroll next window down		
ESC ^W	Delete Paragraph		
ESC ^X	Execute command line		
ESC ^Z	Scroll next window up		

^X <	Narrow-to-region	^X ?	Describe a key
^X >	Widen-from-region	^X !	Run 1 command in a shell
^X =	Show the cursor position	^X @	Pipe shell command to buffer
^X ^	Enlarge display window	^X #	Filter buffer thru shell filter
^X 0	Delete current window	^X $	Execute an external program
^X 1	Delete other windows	^X (	Begin macro
^X 2	Split current window	^X)	End macro
		^X A	Set variable value

^X ^B	Display buffer list	^X B	Switch a window to a buffer
^X ^C	Exit MicroEMACS	^X C	Start a new command processor
^X ^D	Detab line	^X D	Suspend MicroEMACS (UNIX only)
^X ^E	Entab line	^X E	Execute macro
^X ^F	Find file		
^X ^I	Insert file		
^X ^K	Bind Macro to a key	^X K	Delete buffer
^X ^L	Lower case region		
^X ^M	Delete Mode	^X M	Add a mode
^X ^N	Move window down	^X N	Rename current filename
^X ^O	Delete blank lines	^X O	Move to the next window
^X ^P	Move window up	^X P	Move to the previous window
^X ^R	Get a file from disk	^X R	Incremental reverse search
^X ^S	Save current file	^X S	Incremental forward search
^X ^T	Trim line		
^X ^U	Upper case region	^X U	List undo information
^X ^V	View file		
^X ^W	Write a file to disk	^X W	resize Window
^X ^X	Swap "." and mark	^X X	Use next buffer
^X ^Z	Shrink window	^X Z	Enlarge display window

Usable Modes

ABBREV	Enable abbreviation expantions
ASAVE	Save the file every 256 inserted characters
CMODE	Change behavior of some commands to work better with C
CRYPT	Current buffer will be encrypted on write, decrypted on read
EXACT	Exact case matching on search strings
MAGIC	Use regular expression matching in searches
OVER	Overwrite typed characters instead of inserting them
REP	Similar to OVER, handles double-byte characters and tabs differently
SPELL	Invoke MicroSPELL to check for spelling errors
VIEW	Read-Only mode where no modifications are allowed
WRAP	Lines going past right margin "wrap" to a new line

WHITE/CYAN/MAGENTA/YELLOW/BLUE/RED/GREEN/BLACK/GREY/GRAY/LRED/LGREEN/LYELLO/LBLUE/LMAGENTA/L
Sets foreground color

white/cyan/magenta/yellow/blue/red/green/black/grey/gray/lred/lgreen/lyello/lblue/lmagenta/lcyan Sets background color

Appendix E

Numeric Arguments to Commands

In general, preceding a MicroEMACS command with a numeric argument **n** causes the command to be executed **n** times. However, there are a great many commands for which this has no effect, simply because it would make no sense for the command to be executed more than once. There are also commands that take advantage of the numeric arguments to alter their behavior subtly or unsubtly. The following is a list of these commands. Commands that are not affected at all by numeric arguments are listed afterwards.

backward-character	A negative argument invokes *forward-character*.
change-screen-size	With no arguments, the number of rows defaults to the largest. Otherwise, set the screen size to **n**.
change-screen-width	With no arguments, the number of columns defaults to the largest. Otherwise, set the screen width to **n**.
clear-and-redraw	With an argument, centers the window around the current cursor position.
delete-next-character	A negative argument invokes *delete-previous-character*.
delete-next-word	With an argument of 0, will not delete the whitespace trailing the deleted word. A negative argument will cause nothing to happen.
delete-previous-character	A negative argument invokes *delete-next-character*.
delete-previous-word	An negative or zero argument will cause nothing to happen.
detab-region	Without an argument, *detab-region* changes hard tabs to spaces in the lines between the mark and the cursor. With an argument **n**, the commands detab **n** lines – forward if **n** is positive, backwards if not.
end-of-word	A negative argument invokes *next-word*.
entab-region	Without an argument, *entab-region* changes spaces to hard tabs in the lines between the mark and the cursor. With an argument **n**, the commands entab **n** lines – forward if **n** is positive, backwards if not.
exchange-point-and-mark	Swap the current cursor position and mark number **n**. Without an argument, **n** defaults to 0.
exit-emacs	Providing a numeric argument **n** causes two things to happen. First, no checking for modified buffers will occur. Second, MicroEMACS exits with a status of **n**.
forward-character	A negative argument invokes *backward-character*.
goto-line	An argument **n** will be taken as the line number to go to. Without an argument, you will be asked for a line number. In either case, the line number must be 1 or greater.
goto-mark	Go to mark number **n**. Without an argument, **n** defaults to 0.
grow-window	A negative argument invokes *shrink-window*. An argument of 0 causes no action.
handle-tab	Without an argument, *handle-tab* deals with the tab character, whether it should be a single “hard” tab, or expanded as spaces. With an argument **n**, $softtab is set to **n**.
hunt-backward	The command will hunt **n** times. The command will report failure if it cannot find its pattern the **nth** time, even if has found an occurrence of the pattern before number **n**. A negative argument invokes *hunt-forward*.

hunt-forward	The command will hunt **n** times. The command will report failure if it cannot find its pattern the **nth** time, even if has found an occurrence of the pattern before number **n**. A negative argument invokes *hunt-backward*.
kill-to-end-of-line	With no argument **n**, the command deletes all characters to the end of the line. If it is already at the end of the line, it will delete the newline. With a positive **n** as an argument, the command will delete **n** complete lines, newline character and all, starting from the cursor. With **n** equal to zero, the command deletes all text from the cursor to the beginning of the line, but will not delete past the newline character. A negative **n** is illegal.
list-buffers	With a numeric argument, INVISIBLE buffers are also listed.
move-window-down	With a negative argument, invokes *move-window-up*.
move-window-up	With a negative argument, invokes *move-window-down*.
next-buffer	With an argument **n**, the **nth** buffer after the current one is selected, and read in if necessary. Any buffers in between the current buffer and the target buffer that have not yet been read in are read.
next-line	A negative argument invokes *previous-line*.
next-page	Without an argument, the window is scrolled forward by a full page. With an argument **n**, the window is scrolled forwards by **n** lines. The cursor is placed on the upper left hand corner. Negative arguments invoke *previous-page*.
next-paragraph	A negative argument invokes *previous-paragraph*.
next-window	With a positive argument **n**, the **nth** window from the top becomes the working window. With a negative argument, the **nth** window from the bottom becomes the working window.
next-word	A negative argument invokes *previous-word*.
pop-buffer	Without an argument, the buffer is simply displayed in its pop-up screen. With an argument, the buffer is not only displayed, but also given the attribute INVISIBLE.
previous-line	A negative argument invokes *next-line*.
previous-page	Without an argument, the window is scrolled backward by a full page. With an argument **n**, the window is scrolled backwards by **n** lines. The cursor is placed on the upper left hand corner. Negative arguments invoke *next-page*.
previous-paragraph	A negative argument invokes *next-paragraph*.
previous-window	With a positive argument **n**, the **nth** window from the bottom becomes the working window. With a negative argument, the **nth** window from the top becomes the working window.
previous-word	A negative argument invokes *next-word*.
query-replace-string	With a numeric argument, **n** occurrences of the search string may be replaced, depending upon the user's response. The count is based on the number of occurrences found, not the number of positive responses from the user.
quick-exit	Saves all modifed buffers, and exits with a status of **n**.
redraw-display	With no argument, or when **n** is 0, the window is adjusted so that the cursor is in the center. When **n** is positive, the window is adjusted so that the cursor is on the **nth** line of the screen. When **n** is negative, the window is adjusted so that the cursor is on the last line of the window, regardless of the magnitude of **n**.
remove-mark	Remove mark number **n**. Without an argument, **n** defaults to 0.

replace-string	Will replace **n** occurrences of the search string with the replacement string. Otherwise, with no argument, all occurrences from the cursor position to the end of file are replaced.
resize-window	Requires an argument which must be positive.
scroll-next-down	A negative argument invokes *scroll-next-up*.
scroll-next-up	A negative argument invokes *scroll-next-down*.
search-forward	The command will search **n** times. The command will report failure if it cannot find its pattern the **nth** time, even if it has found an occurrence of the pattern before number **n**. A negative argument invokes *search-reverse*.
search-reverse	The command will search **n** times. The command will report failure if it cannot find its pattern the **nth** time, even if has found an occurrence of the pattern before number **n**. A negative argument invokes *search-forward*.
select-buffer	Without an argument, the buffer is simply displayed in the window. With an argument, the buffer is not only displayed, but also given the attribute INVISIBLE.
set	If using the *set* command interactively, preceding the command with a numeric argument then makes it unecessary for the command to ask for the variable's value (it will still ask for the variable's name). If used in a command line, then the command set <variable name> <number> is identical to <number> set <variable name>
set-mark	Set mark number **n**. Without an argument, **n** defaults to 0.
shrink-window	A negative argument invokes *grow-window*. An argument of 0 causes no action.
split-current-window	With **n** = 1, the new upper window becomes the current window. Any other numeric argument makes the new lower window the current window. With no argument, the current window becomes the new upper or lower window depending upon whether the cursor was in the upper or lower half of the old window.
store-macro	Since macroes are numbered, a numeric argument must be provided. These numbered macroes are being phased out in preference for named macroes.
store-procedure	If the command is provided a numeric argument, it will assume that *store-macro* is actually being called.
trim-region	Without an argument, *trim-region* removes spaces and tabs from the end of the lines between the mark and the cursor. With an argument **n**, the commands trim **n** lines – forward if **n** is positive, backwards if not.

E.1 Commands unaffected by numeric arguments.

abort-command
add-global-mode
add-mode
append-file
apropos
back-from-tag-word
begin-macro
beginning-of-file
beginning-of-line
bind-to-key
buffer-position
case-region-lower
case-region-upper
change-file-name
clear-message-line
copy-region
count-words
cycle-screens
delete-blank-lines
delete-buffer
delete-global-mode
delete-mode
delete-other-windows
delete-screen
delete-undos
delete-window
describe-bindings
describe-functions
describe-key
describe-variables
display
end-macro
end-of-file
end-of-line
execute-command-line
execute-program
fill-paragraph
filter-buffer
find-file

find-screen
goto-matching-fence
help
i-shell
incremental-search
insert-file
kill-region
list-undos
macro-to-key
mouse-move-down
mouse-move-up
mouse-region-down
mouse-region-up
mouse-resize-screen
name-buffer
narrow-to-region
nop
pipe-command
print
re-tag-word
read-file
restore-window
reverse-incremental-search
save-file
save-window
set-encryption-key
shell-command
suspend-emacs
tag-word
transpose-characters
unbind-key
unmark-buffer
update-screen
view-file
widen-from-region
wrap-word
write-file
write-message

Appendix F

Supported machines

The following table lists all the hardware/compilers for which I currently support MicroEMACS. This is not exclusive of all machines which MicroEMACS will run on, but I have either run it myself, or had a first hand report of it running.

Hardware	OS	Compiler	Comments
VAX 780	UNIX V5	native	
	BSD 4.2	native	job control supported
	VMS	native	SMG & ANSI support
SUN	SUNOS 3 & 4	native	
		gcc	
NCR Tower	UNIX V5	native	
IBM-RT PC	BSD 4.3	native	
	AIX	native	
HP9000	UNIX V5	native	
IBM-PC	MSDOS 3.2 and above	LATTICE 3	Large CODE/Large DATA
		AZTEC 3.4e	Large CODE/Large DATA
		TURBO C 2.0	LARGE memory model
		MSC 6.0	
		*MWC 86	
	SCO XENIX	native	
	FREEBSD 2.1	GCC	
	Windows 3.1	MSC 6.0	
	WINDOWS 95	Visual C++ 4.1	
	WINDOWS NT	Visual C++ 4.1	
HP150	MSDOS	Lattice 2.15	Function key labels
		Turbo C 2.0	for the touch screen
HP110	MSDOS	Lattice 2.15	
		Aztec 3.4e	
		Turbo C 2.0	
*Data General 10			
	MSDOS	Lattice 2.1	Texas Instruments Professional
	MSDOS	Lattice 2.15	
Amiga	Intuition	Lattice 3.03	
		Aztec 3.6	
ST520	TOS	Mark Williams C	Spawns under MSH
		Lattice 3.1	(no shell commands)
Fujitsu FMR series	MSDOS	MSC 6.0	
NEC 9800 series	MSDOS	Turbo 2.0	Function key support
		MSC 6.0	
HP3000 series	MPE	native	

Systems to be supported (IE some code is already written:)

Macintosh	System 7	Lightspeed C

*means that I do not own or have access to the listed compiler and/or machine and must rely upon others to help support it.

Appendix G

Function Keys

All environments now support a set of machine independant bindings for function keys. Below is a list of these bindings (not all of these are supported on all systems).

Function keys in MicroEmacs

	function	Function	^function	Alt-function
f1)	FN1	S-FN1	FN^1	A-FN1
f2)	FN2	S-FN2	FN^2	A-FN2
f3)	FN3	S-FN3	FN^3	A-FN3
f4)	FN4	S-FN4	FN^4	A-FN4
f5)	FN5	S-FN5	FN^5	A-FN5
f6)	FN6	S-FN6	FN^6	A-FN6
f7)	FN7	S-FN7	FN^7	A-FN7
f8)	FN8	S-FN8	FN^8	A-FN8
f9)	FN9	S-FN9	FN^9	A-FN9
f10)	FN0	S-FN0	FN^0	A-FN0
home)	FN<		FN^<	
CsUp)	FNP		FN^P	
PgUp)	FNZ		FN^Z	
CsLf)	FNB		FN^B	
5)				
CsRt)	FNF		FN^F	
End)	FN>		FN^>	
CsDn)	FNN		FN^N	
PgDn)	FNV		FN^V	
Ins)	FNC		FN^C	
Del)	FND		FN^D	

Appendix H

Machine Dependent Notes

This appendix lists some notes specific to individual implementations of MicroEMACS. Every attempt has been made to allow EMACS to be identical on all machines, but we have also tried to take advantage of function keys, cursor keys, mice, and special screen modes where possible.

H.1 IBM-PC/XT/AT and its clones

The IBM-PC family of computers is supported with a variety of different display adapters. EMACS will attempt to discover what adapter is connected and use the proper driver for it. Below is a list of the currently supported video adapters:

Adapter	$sres	Original mode used
Monochrome Graphics Adapter	MONO	MONO
Color Graphics Adapter	CGA	CGA
	CGA40	CGA40
Enhanced Graphics Adapter	EGA	CGA
Video Graphics Adapter	VGA	CGA
	VGA12	

If a driver for a Microsoft compatable mouse is installed on the system, EMACS will use the mouse in text mode and allow the user all the standard mouse functions. The mouse cursor will appear to be a block of color in the color opposite of it's background.

EMACS also takes advantage of various function keys and the keys on the keypad on an IBM-PC. The function keys are initially not bound to any particular functions (except by the emacs.rc startup file), but the keypad keys do default to the following:

Keypad key	Function
Home	beginning-of-file
CSRS UP	previous-line
Pg Up	previous-page
CSRS LEFT	backward-character
CSRS RIGHT	forward-character
End	end-of-file
CSRS DOWN	next-line
Pg Dn	Next-page

All these special keys are indicated in EMACS macroes by use of the **FN** prefix. Below is a list of many of the keys and the codes used to specify them. Also the codes may be gotten by using the describe-key (^X ?) command on the suspect key.

Compiling under TURBO C

To compile MicroEMACS under TURBO C, set the TURBO integrated environment with the following options:

Memory model	LARGE
Floating point	NONE
Default char type	UNSIGNED
Data alignment	BYTE
Merge duplicate strings	ON
Standard stack frame	off
Test stack overflow	off

Optimize for SIZE

```
Use register optimization          ON
Register optimization              ON
Jump optimization                  ON

Initialize segments        OFF
Stack warnings             OFF

Names: Code names
            Segment name   *
```

H.2 Windows 3.1, OS/2, Windows 95 and Windows NT

On the IBM-PC there is a very nice window oriented version of MicroEMACS prepared by Pierre Perot. It has all the functionality of the character oriented version, but in addition has drop down menus, and mouse functionality more in line with how the mouse is used in these environments.

There is an extensive on-line help facility for the windows version, which can be accessed from the HELP menu when MicroEMACS is running. Much of the CUA style of mousing and editing is provided by an extra startup file, **CUA.CMD**, which is run when MicroEMACS for Windows is started. If you prefer the original style of mousing (if, for instance, you work on many platforms) rename the CUA.CMD file to something else.

The windows version has been tested under Windows 3.1, Windows 95, Windows NT v3.5 and under OS/2 in the windows emulation box.

In addition, there is also a WINDOWS NT console mode version which runs under Windows NT and WIndows 95. This version is recommended for those environments, as its performance (as a 32 BIT application) is much better than the DOS version running in a DOS box on these machines.

H.3 HP 150

This machine from Hewlett Packard is very unusual for an MSDOS machine. It has a touch screen and is very function key oriented. An additional command, *label-function-key* allows you to place labels on the on screen function key labels. A numeric argument indicates which function key to label (one through eight) and then the program prompts for a 16 character label, which will be used as two lines of eight characters. To label function key three with “save file” from a macro, you would use:

```
3 label-function-key "save      file"
```

Notice the 4 spaces after “save”. This forces “file” to begin on the second line of the label.

H.4 Atari 520/1040ST

The ATARI ST family of computers have a dual personality. They may use either a monochrome or a color screen. EMACS supports two screen resolutions on each monitor.

NOTE

When you set MicroEMACS up on your system, please remember to install it on the desktop as a GEM application. If you have EMACS set as a TOS application, the mouse will not function properly, and EMACS will alert you to this problem by beeping the bell.

Monitor	$sres	size	#color	$palette	format
Color	LOW	40x25	16		000111222333444555666777
	MEDIUM				
		80x25	4		000111222333
Mono	HIGH	80x25	2		000
	DENSE	80x50	2		000

The $palette environment variable can be used to change what color is associated with each color name. With a color monitor, each group of three digits indicates an octal number specifying the RED, GREEN and BLUE levels of that color. Each color digit can vary from 0 to 7. For example, the initial setting of $palette in LOW resolution is:

000700070770007707077777

which broken up is:

000 700 070 770 007 707 077 777

which means:

000	Black
700	Red
070	Green
770	Yellow
007	Blue
707	Magenta
077	Cyan
777	White

Also the mouse buttons are bound to mouse functions as described in the chapter about mice. The cursor keys and the function keys are bound similarly to IBM-PC.

Files generated by EMACS on the ATARI ST have a single return character at the end of each line, unlike the desktop files which want to have two returns. This makes it display files strangely from GEM's [SHOW] option, but makes the files port to other computers much nicer. When compiling MicroEMACS, the ADDCR symbol in **estruct.h** will cause emacs to generate line ending sequences compatible with GEM.

Currently, when operating under the Mark Williams MSH program, EMACS can shell out and perform external commands. This capability will be added later for the Beckmeyer shell and under GEMDOS.

H.5 Amiga 1000

The Commodore AMIGA 1000 version of MicroEMACS does fully support the mouse, window resizing and the close gadget. It runs in medium resolution, using the colors defined for the workbench.

Note about Compiling MicroEMACS

If you are compiling the sources on the AMIGA to produce an executable image, and you are using the Lattice compiler, be sure to give the CLI command ‘STACK 40000’ before compiling to make sure the compiler has sufficient stack space to successfully complete compilation.

H.6 UNIX V5, FREEBSD and BSD4.[23]

MicroEMACS under UNIX utilizes the **TERMCAP** library to provide machine independent screen functions. Make sure that termcap is available and properly set on your account before attempting to use MicroEMACS.

Under systems which support job control, you can use the **^XD** *suspend-emacs* command to place EMACS into the background. This carries a much smaller overhead than bringing up a new shell under EMACS. EMACS will properly redraw the screen when you bring it back to the foreground.

If the symbol VT100 has been set to 1 in the *estruct.h* options file, EMACS will recognize the key sequence <ESC>[as the lead in sequence for the FN function key prefix.

With the addition of some very machine/operating system specific code, EMACS can prevent two or more people from modifying the same file at the same time. The upper level of a set of functions to provide file locking exist in the source file **LOCK.C**. It requires two machine specific functions written and linked into EMACS for it to operate properly.

```
char *dolock(fname)

char *fname;
```

dolock() locks a file, preventing others from modifying it. If it succeeds, it returns NULL, otherwise it returns a pointer to a string in the form "LOCK ERROR: explanation".

```
char *undolock(fname)

char *fname;
```

undolock() unlocks a file, allowing others to modifying it. If it succeeds, it returns NULL, otherwise it returns a pointer to a string in the form "LOCK ERROR: explanation".

H.7 DEC VMS operating system

TERMINALS

Depending upon the options set in ESTRUCT.H, MicroEMACS uses either the capabilities of VMS SMG, working with any terminal that is defined in SMGTERMS.TXT or TERMTABLE.TXT (see your SMG manual for more information), or the ANSI escape sequences. Full keyboard support, including function keys, is provided for VT100 and VT200 series compatible terminals. Mouse support is provided under the ANSI version only at this time. Mouse support is provided for the VSII workstation's VT220 terminal emulator, and other terminal emulators that use the same escape sequences for mouse control. (There is some partial support for the BBN BitGraph mouse sequences in the sources, but this is not yet complete). Terminals may have up to 100 lines and 160 columns.

The maximum terminal size is 256 columns and 72 rows. If you run MicroEMACS on a terminal that is larger than this, MicroEMACS will reduce it to these limits while you are editing.

Flow control

Some terminals will require the use of XON/XOFF flow control when used with MicroEMACS. When XON/XOFF flow control is used, you will not be able to use functions bound to ^S or ^Q, and should use bind-to-key to put these functions on other keys. MicroEMACS does not change the flow control characteristics of your terminal line while it is running. If your terminal requires flow control, you should:

```
$ SET TERM/HOSTSYNC/TTSYNC
```

before entering MicroEMACS. If you are on a VSII emulated workstation terminal, are using the SSU multi-session protocol (VT330 and VT340 with SSU enabled), or are certain that your terminal does not require XON/XOFF flow control, you should

```
$ SET TERM /HOSTSYNC/NOTTSYNC
```

This will allow you to use ^S and ^Q for MicroEMACS commands. Note that if you are using a VSII with VWS V3.2 or later, you must leave the /HOSTSYNC enabled in order for the cross/session cut and paste capability to work properly.

KEYBOARD

The VMS version understands the LK201 functions of VT200 series, vt300 series, and compatible terminals and terminal emulators, and allows you to bind to them as function keys. In addition, the VT100 numeric keypad, in application mode, is available as function keys. MicroEMACS will only put the keypad into application mode for you if the KEYPAD option is set in ESTRUCT.H. In this situation, MicroEmacs will detect your kepad's state, and restore it to that state upon exiting. If MicroEMACS has not been compiled with this option, you may still put the keypad into application mode by issuing the command "SET TERM /APPLICATION" before entering MicroEMACS.

VT200 keys

Note that F1 through F5 are local function keys on DEC terminals.

F6	= FN6	FIND = FNS
FN7	= FN7	INSERT = FNC
F8	= FN8	REMOVE = FND
F9	= FN9	SELECT = FN@
F10	= FN0	PREV = FNZ
F11	= S-FN1	NEXT = FNV
F12	= S-FN2	Arrow Up = FNP
F13	= S-FN3	Arrow Down = FNN
F14	= S-FN4	Arrow Right = FNF
HELP (F15)	= S-FN5	Arrow Left = FNB
DO (F16)	= S-FN6	
F17	= S-FN7	
F18	= S-FN8	
F19	= S-FN9	
F20	= S-FN0	

VT100 and VT200 numeric keypad in application mode

PF1 = FN^1	PF2 = FN^2	PF3 = FN^3	PF4 = FN^4
7 = A-7	8 = A-8	9 = A-9	- = A--
4 = A-4	5 = A-5	6 = A-6	, = A-,
1 = A-1	2 = A-2	3 = A-3	ENTER = A-E
0 = A-0	. = A-.		

WARNING

The VMS version contains code for interpreting function keys that are sent as Ansi sequences that begin with the ESC character. Because of this, MicroEMACS cannot process an incoming ESC until it knows what character follows it. This can cause problems with terminating search and replace strings. If you use ESC as the meta-prefix character (which is the default) you must type one additional keystroke following ESC before emacs will recognize that you have edited the search command prompt, and are continuing. (The additional character is processed normally be MicroEMACS, it is NOT discarded.)

MicroEMACS must wait long enough for the network delay that might be involved between seeing the ESC and seeing the characters that follow it. If holding down one of the arrow keys causes characters to drop into your file, then you may want to alter the delay yourself. The logical variable MICROEMACS$SHORTWAIT may be set to vary that delay. The default delay is 400ms (4 tenths of a second). The equivalent value in MICROEMACS$SHORTWAIT is 4000000.

Special case for BBN BItGraph

If you are using the BBN BitGraph, execute the following commands before entering MicroEMACS, and you will get mouse support:

```
$ esc[0,8] = 27
$ microemacs$mouse_enable == esc+":5;6;L"+esc+":0;63;;;;;;;;9;16;c"
$ microemacs$mouse_disable == esc+":5;1;L"+esc+":0;0c"
$ exit
```

Do NOT do this for any other terminals.

Search List for EMACS.RC

VMS MicroEMACS will first search logical name MICROEMACS$LIB:, then SYS$LOGIN:, then finally the current directory when looking for startup files or help files.

If desired, MICROEMACS$LIB may be defined to be a VMS search list that first searches a user directory, and then a system directory.

Generally, you should create a private directory where you keep all your .CMD files, and in your LOGIN.COM $DEFINE a logical name to point to this area.

In addition to whatever commands you have in your EMACS.RC file, one command you should certainly include is “set $ssave FALSE”. The “safe save” mechanism, which writes a buffer to a temporary file, deletes the old version of a file, and then moves the temporary file to its permanent name, works wonderfully on most systems, but makes no sense on VMS, which maintains older versions of a file.

Using MicroEMACS as a subprocess

MicroEmacs can now be kept in a subprocess. You can arrange to start emacs only once in a job, and to re-attach to it each time you want to use it. This is optional. To use this feature, install MicroEMACS in the following way:

1. MicroEMACS contains two images. ME.EXE is a small program for starting and stopping the Emacs subprocess. The source for ME. is in ME.C, and should not be linked into MESHR.EXE. MESHR.EXE is the actual MicroEMACS image. The name "MESHR" is required for MAIL/NOTES support, see next section for details.

2. Make sure that the SYS$SHARE search list includes MESHR.EXE. If you don’t have the privilages to move MESHR.EXE into SYS$SHARE, you can $ DEFINE the MESHR logical name to be the full name and location of

the MESHR.EXE program. For example, you could store all of these programs in the MICROEMACS$LIB: search list, and say:

```
$ DEFINE MESHR microemacs$lib:meshr.exe
```

3. Put ME.EXE in MICROEMACS$LIB and the following line in your LOGIN.COM:

```
$ me :== $microemacs$lib:me
```

4. Put a line in your EMACS.RC that will

```
bind-to-key suspend-emacs ^C  ; use your usual exit-emacs key
```

Now, use the "$ ME" command to invoke microemacs. Subseqeuent invocations in the same job will re-use the existing subprocess. You can use the full capabilty of the microemacs command line in the first and in all subsequent invocations of ME.

WARNING:

MicroEMACS will ALWAYS read in new copies of any files you specify on the command line, even if you are already editing it. If you edit a file a second time with the same MicroEMACS, you will get a NEW buffer with ANOTHER copy of the file. The old buffer is still there also. It is easy, in this situation, to accidently edit in a WRONG BUFFER, and if you write out an obsolete buffer, you will lose earlier edits!

This is considered a bug and may be fixed in a later version of MicroEMACS. To avoid this situation, do not specify a file on the command line if MicroEMACS already has that file in a buffer. Use the "find-file" MicroEMACS command instead.

Using MICROEMACS with MAIL and NOTES:

With VMS V5 and later versions, the MAIL interface to Microemacs is much simplified. With VMS V5, the MESHR.EXE image does NOT have to be installed as a known image to be used as a callable editor from MAIL. Therefore, to use MicroEMACS as your VMS MAIL editor, simply add the following lines to your LOGIN.COM:

```
$ DEFINE MAIL$EDIT CALLABLE_ME
$ MAIL :== MAIL/EDIT
```

and make sure that the SYS$SHARE search list includes MESHR.EXE. If you don't have privs or permission to move MESHR.EXE into SYS$SHARE, you can $ DEFINE the MESHR logical name to be the full name and location of the MESHR.EXE program. For example, you could store all of these programs in the MICROEMACS$LIB: search list, and say:

```
$ DEFINE MESHR microemacs$lib:meshr.exe
```

Note that this is the same location as is required for using kept MicroEMACS.

To abort sending a message, exit MicroEMACS without writing out the mail message file.

To use MicroEMACS as your VAX NOTES editor, issue the following command to VAX NOTES:

```
NOTES> SET PROFILE/EDIT=(ME,CALL)
```

Note, if you are still in the dark ages of VMS V4, you will have to either install MESHR as a known image, or following the original "Second way" instructions given in the existing appendix F.6 of the older MicroEMACS manual (previous to version 3.10).

Second way, as described in older versions

In the event that you cannot get your system manager to INSTALL MicroEMACS as known image, you can use the following technique:

1. In MICROEMACS$LIB:MEMAIL.COM, put the following command file:

```
$! Use on VAX/VMS as MAIL$EDIT for using MicroEMACS as mail editor.
$ if "''P1'" .NES. "_NL:" then if "''P1'" .NES. "" then copy 'P1' 'P2'
```

```
$ define/user sys$input sys$output
$ me 'P2'
$ exit
```

This file may have come with your MicroEMACS kit.

2. In your LOGIN.COM, put the following lines:

```
$       me :== $MICROEMACS$LIB:MESHR.EXE ! Assumes meshr.exe is there
$       define mail$edit microemacs$lib:me_edit.com
```

3. In NOTES, give the command

```
NOTES> SET PROFILE/EDIT=(@MicroEMACS$lib:me_edit.com,SPAWN)
```

System messages and EMACS

MicroEMACS will intercept system broadcast messages and display them on the message line after any input from the user. These message are stored in an INVISIBLE buffer named [-messages-]. To view these at your convenience, use the following procedure:

```
;
; Show any system messages MicroEMACS may have intercepted.
; The numeric prefix of pop-buffer ensures its invisibility.
;
store-procedure "Show_messages"
        1 pop-buffer "[-messages-]"
!endm
```

Building MicroEMACS for VMS

The configuration options are set in file estruct.h:

- Under the category of "Machine/OS definitions", set VMS to "1" and all others to "0".

- Under "Compiler definitions", set all selections to "0". Selecting VMS implies that you are using VAXC.

- Under "Special keyboard definitions", be sure "VT100" is set to "0". This option is not required for the VMS version, it is for other systems using ANSI terminal support. VMS in combination with SMG or ANSI already handles the special characteristics of Ansi keyboards.

- Under "Terminal Output definitions", set either ANSI or SMG to "1" and all others to "0". As stated previously, only ANSI supports the mouse at this time.

- Under "Configuration options", you may select as you wish, with the following notes:

- COLOR	support does not exist for VMS, even when using color workstations.
- MOUSE	support should be enabled if you have any VSII workstations. Only supported under the ANSI driver.
- KEYPAD	support recognises whether your keypad is already in application mode or not, and puts your keypad in its correct state on exit.
- XNONOFF	automatically allows you to use control-S or control-Q in MicroEMACS, by disabling the TTSYNC characteristic. This option should not be set if MicroEMACS might be used on DecStations or VT100s. It also should not be used with slow terminals or terminal emulators connected to fast terminal lines.
- RMSIO	support should absolutely be used. This option allows the writing and reading of files in VMS's

variable-length format, as opposed to STREAM-LF, and cuts down on file writing and reading time by approximately two thirds.

- OPTMEM support may be used on VMS versions 5.0 and higher. It substitutes the C library's memory allocation calls for the native VAX calls, and gives a speed improvement.

If you have MMS, you can use the supplied DESCRIP.MMS to build MicroEMACS. Otherwise, the command file MEMAKE.COM has been provided. These files assume that you are using SMG as your terminal driver. If you are using ANSI, then you must replace SMG with ANSI in the command and opt files. If you do not have MMS or are missing MEMAKE.COM, simply compile each module with "CC", and link with the command:

$ LINK MESHR/OPTION/SHARE

Note that the executable filename must end in "SHR" in order for MicroEMACS to be used as a callable editor from MAIL or NOTES. (Method 1 above.)

If you edit any of the Emacs sources, note that any global or external data must be declared as "noshare" in order for the VMS callable editor support to work properly. This applies to all global data used in the VMS version, but not to routines or to "static "data. The "noshare" declaration is #define'd away on non-VMS systems. If you fail to do this, VMS will not allow you to INSTALL MicroEMACS as a sharable library.

Appendix I

Mode Flags

The two environment variables, $cmode and $gmode, contain a number the corresponds to the modes set for the current buffer and the editor as a whole. These are encoded as the sum of the following numbers for each of the possible modes:

WRAP	1	Word wrap
CMODE	2	C indentation and fence match
SPELL	4	Interactive spell checking (Not Implemented Yet)
EXACT	8	Exact matching for searches
VIEW	16	Read-only buffer
OVER	32	Overwrite mode
MAGIC	64	Regular expressions in search
CRYPT	128	Encryption mode active
ASAVE	256	Auto-save mode
REP	512	Character replace mode
ABBREV	1024	Abbreviation expantion mode

So, if you wished to set the current buffer to have CMODE, EXACT, and MAGIC on, and all the others off, you would add up the values for those three, CMODE 2 + EXACT 8 + MAGIC 64 = 74, and use a statement like:

```
set $cmode 74
```

or, use the binary or operator to combine the different modes:

```
set $cmode &bor &bor 2 8 64
```

Internal Flags

Some of the ways EMACS controls its internal functions can be modified by the value in the $gflags environment variable. Each bit in this variable will be used to control a different function.

GFFLAG	1	If this bit is set to zero, EMACS will not automatically switch to the buffer of the first file after executing the startup macroes.
GFSDRAW	2	If this bit is set to one, supress redraw events.
GFEXIT	4	This bit is set to one while executing the $exithook. If reset to zero, the exit is canceled.

Current buffer flags

The $cbflags environment variable allows the user to modify some of the characteristics of the current buffer. The various characteristics are encoded as the sum of the following numbers:

BFINVS	1	Internal invisible buffer
BFCHG	2	Changed since last write
BFTRUNC	4	buffer was truncated when read
BFNAROW	8	buffer has been narrowed

Only the invisible and changed flags can be modified by setting the $cbflags variable. The truncated file and narrowed flags are read only.

Index

Contents