MicroEMACS

Full Screen Text Editor
Reference Manual (preliminary draft)

Version 3.8i
April 27, 1987

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 powerfull 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 text may be moved freely from one window 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 wordprocessing 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 controling, 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 are connected, or bound, to.

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 of DOS commands and filtering of text through other programs (like SORT to allow sorting text).

History

EMACS was originally a text editor written by Richard Stahlman at MIT in the early 1970s for Digital Equipment computers. Various versions, rewrites and clones have made an appearence 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 has made extensive modifications and additions to it over the course of the next two years. Updates and support for the current version is still in progress. The current program author can be contacted by writing to:

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

UUCP: ihnp4!pur-ee!pur-phy!duncan!lawrence
ARPA: nwd@j.cc.purdue.edu
FIDO: Fido 201/2 The Programmer's Room (317) 742-5533

Chapter 1

Basic Concepts

The current version of MicroEMACS is 3.8i (Third major re-write, eighth public release, Ith (or ninth) minor release), and for the rest of this document, we shall simply refer to this version as "EMACS". Any modifications for later versions will be listed in the appendixes at the end of this manual.

1.1 Keys and the Keyboard

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

<NL> NewLine which is also called RETURN or ENTER, 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 as well. 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 D to learn what key is used for META on your computer).

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

1.2 Getting Started

In order to use EMACS, you must call it up from your system's or computer's command prompt. On UNIX and MSDOS machines, just type "emacs" from the main 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 other icon based operating systems, double click on the uEMACS icon. Shortly after this, a screen similar to the one below should appear.

1.3 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.

```
================================================================================
f1 search      f2 search back : F1 toggle function list F2 toggle help file
f3 hunt        f4 hunt back   : F3 find command/apropos F4 describe key
f5 next window f6 exec macro  : F5 reformat paragraph   F6 ref undented region
f7 find file   f8 exec file   : F7 indent region        F8 undent region
f9 save file  f10 exit emacs  : F9 execute DOS command F10 shell up
================================================================================
-- MicroEMACS 3.8i () -- Function Keys -----------------------------------------
================================================================================
```

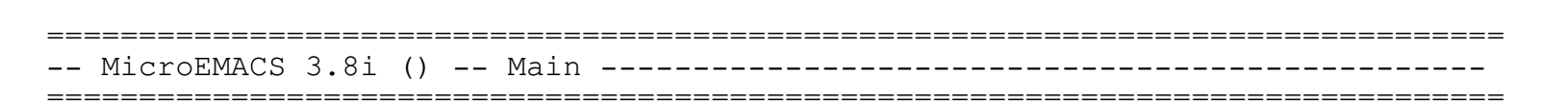

Fig 1: EMACS screen on an IBM-PC

1.4 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 the text to the left of the cursor disappears and a '$' sign appears. Don't panic--your text is safe!!! You've just discovered that EMACS doesn't "wrap" text to the next line like most word processors unless you hit <NL>. But since EMACS is used for both word processing, and text editing, it has a bit of a dual personality. You can change the way it works by setting various modes. In this case, you need to set WRAP mode, using the add-mode command, by typing ^X-M. The command line at the base of the screen will prompt you for the mode you wish to add. Type wrap followed by the <NL> key and any text you now enter will be wrapped. However, the command doesn't wrap text already entered. To get rid of the truncated line, delete characters with the <BACKSPACE> key until the '$' goes away. 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.

1.5 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, 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 about. Also, if there is one available, moving the mouse will move the cursor.

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

1.6 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.

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 1 Summary

In chapter 1, 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 2

Basic Editing--Simple Insertions and Deletions

2.1 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.

2.2 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 exisiting 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.

2.3 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 obeginning 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 2 Summary

In Chapter 2, 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 3

Using Regions

3.1 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.

3.2 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 3 Summary

In Chapter 3, 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
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 4

Search and Replace

4.1 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 occurance 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.

4.2 Exact Searches

If the mode EXACT is active in the current buffer, EMACS searches on a case sensitive basis. Thus, for example you could search for Publishing as distinct from publishing.

4.3 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.

4.4 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 occurances 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 captialize 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.

4.5 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 occurance 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 4 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 ocurrences 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 5

Windows

5.1 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.

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 ^X-2 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 ^X-P previous-window. Notice the cursor moves to the upper or previous window. Entering ^X-O 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 find-file

and press return. Then 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-^U 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.

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.

Experiment with splitting the windows on your screen. Open windows into different buffers and experiment with any other files you may have. ry 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.

5.2 Deleting Windows

5.3 Resizing Windows

5.4 Other Window commands

Chapter 5 Summary

In Chapter 5 you learned how to manipulate windows and the editing flexibility they offer.

Key Binding	Keystroke	Effect
Open-Window	^X-2	Splits current window into two windows if space is available
Close-Windows	^X-1	Closes all windows except current window
Next-Window	^X-O	Moves point into next (i.e. downward) window
Previous-Window	^X-P	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	^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-^U	Scrolls next window down by one line

Chapter 6

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 opoened 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:

^X-X 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 ^X-X again places you in the publish.txt buffer. If you are on a machine that supports function keys, using ^X-X again places you in the Function Keys buffer. Using ^X-X 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 ^X-B 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.

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, ^X-1.

To delete any buffer, type ^X-K 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 6 Summary

In Chapter 6 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	^X-B	Switch to a particular buffer
List-Buffers	^X-^B	List all buffers
Delete-Buffer	^X-K	delete a particular buffer if it is off-screen

Chapter 7

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 ^X-M 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 modeline 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 modeline.

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 forground 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 forground 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 intellegent guess at what color to use when you ask for one which is not there.

7.1 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.

7.2 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 current 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 delete one tab position preceeding itself as it is typed. This should line up the close brace with its matching IF, FOR or WHILE statement.

A pound sign (#) with only leading whitespace will delete all the whitespace preceeding 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 breifly flash to it, and then back. This makes balencing expresions, and matching blocks much easier.

7.3 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.

7.4 EXACT mode

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.

7.5 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 flexability when using the search command. However, they do not affect 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 loose 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, just 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 * is known as closure, and means that zero or more of the preceding character will match. If there is no character preceding, * has no special meaning, and since it will not match with a newline, * will have no special meaning if preceded by the beginning of line symbol ^ or the literal newline character <NL>.

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 guarenteed to match no letter 'a' , which satifies the search conditions. If you wanted to search for one or more of the letter 'a', you would search for "aa*", which would match the letter a, then zero or more of them.

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 a lot of 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 escape character \ is for those times when you want to be in MAGIC mode, but also want to use 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. The escape character will also let you

put ^, -, or] inside a character class with no special side effects.

7.6 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.

7.7 WRAP mode

Wrap mode is used when typing in continuous text. Whenever the cursor is past the currently set fill column (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 continous stream of words and EMACS automatically inserts <NL>s at appropriate places.

NOTE to programmers:

> EMACS actually calls up the function bound to the illegal keystroke M-FNW. This is bound to the function wrap-word by default, but can be re-bound to activate different functions and macros at wrap time.

7.8 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 7 Summary

In Chapter 7 you learned about modes and their effects.

Key Binding	Keystroke	Effect
Add-Mode	^X-M	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 8

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 filenames. 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 filenames in EMACS,

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

If you do not specify a disk and directories, the default disk 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 ^X-N. 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 ^X-N. 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. 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 8 Summary

In Chapter 8 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	^X-N	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 9

Screen Formatting

9.1 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 appearence 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 ^X-M and responding to the command line prompt with wrap.

You can also set your own wrap margin with the command ^X-F set-fill-column. Notice EMACS responds "[Fill column is 1]." Now try typing some text. You'll notice some very strange things happening -- your text wraps at every word!! This effect occurs because the set wrap margin command must be preceeded by a numeric argument or EMACS sets it to the first column. Thus any text you type that extends past the first column will wrap at the most convenient line break.

To reset the wrap column to 72 characters, press the <META> key and enter 72. EMACS will respond "Arg: 72". Now press ^X-F. EMACS will respond "[Fill column is 72]". Your text will again wrap at the margin you've been using up to this point.

9.2 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. The process is complex, and (especially with longer paragraphs) may take a little time.

9.3 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 back 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.

9.4 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.

Many times you would like to take a line which has been created using the tab character and change it to use just spaces. The command ^X-^D detab-line changes any tabs from the point to the end of the current line into the right number of spaces so the line does not

change. This is very usefull for times when the file must be printed or transfered to a machine which does not understand tabs.

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

Chapter 9 Summary

In Chapter 9 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	^X-M[WRAP]	Add wrap mode to current buffer
Delete-Mode/WRAP	^X-^M[WRAP]	Remove wrap mode from current buffer
Set-Fill-Column	^X-F	Set fill column to given numeric argument
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-Line	^X-^E	Changes multiple spaces to tabs characters where possible
Detab-Line	^X-^D	Changes tab characters to the apprpriate number of spaces

Chapter 10

Keyboard Macros

In many applications, it may be necessary 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 macros 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 ^X-E 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 preceed it with a numerical argument.

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 macro or procedure facilities detailed in chapter <X>.

Chapter 10 Summary

Chapter 10 covered keyboard macros. 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	^X-E	Entire sequence of recorded keystrokes is replayed

Chapter 11

MicroEMACS Macros

Macros 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 impliment via macros, as well as the example menu system. The execute-macro-<n> commands cause the macro, numbered from 1 to 40, to be executed. The execute-file command allows you to execute a macro stored in a disk file, and the execute-buffer command allows you to execute a macro stored in a buffer. Macros are stored for easy execution by executing files that contain the store-macro command.

There are many different aspects to the macro langage within MicroEMACS. Editor commands are the various commands that manipulate text, buffers, windows, etc, 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.

11.1 Variables

Variables in MicroEMACS 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 (^X-A) 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
```


11.1.1 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 expresion. All environmental variable names begin with a dollar sign ($) and are in lower case.

$fillcol	Current fill column
$pagelen	Number of screen lines used currently
$curwidth	Number of columns used currently
$curcol	Current column of point in current buffer
$curline	Current line of point in current buffer
$flicker	Flicker Flag set to TRUE if IBM CGA set to FALSE for most others
$cbufname	Name of the current buffer
$cfname	File name of the current buffer
$sres	Current screen resolution (CGA, MONO or EGA on the IBM-PC driver. LOW, MEDIUM, HIGH or DENSE on the Atari ST1040, NORMAL on all others)
$debug	Flag to trigger macro debugging (try it... you'll like it!)
$status	return status of the success of the last command (TRUE or FALSE) usually used with !force
$palette	string used to control the palette register settings on graphics versions. The usually form consists of groups of three octal digits setting the red, green, and blue levels.
$asave	The number of inserted characters between automatic file-saves in ASAVE mode.
$acount	The coundown of inserted characters until the next save-file.
$lastkey	Last keyboard character typed
$curchar	Character currently at the point

$discmd	Flag to disable the echoing of messages on the command line
$version	Contains the current MicroEMACS version number
$progname	Always contains the string "MicroEMACS" for standard MicroEMACS. Could be something else if used as part of someone else's program
$seed	integer seed of the random number generator
$disinp	Flag to disable the echoing of characters during command line input

Obviously, many more of these variables will be availible in future releases of MicroEMACS. (Yes, send a vote for your favorite new environmental variables today).

11.1.2 User variables

User variables allow you, the user, 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 macros execute. At the moment, up to 100 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 10 characters are significant (ie differences beyond the tenth character are ignored). Most operators will truncate strings to a length of 128 characters.

11.1.3 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 3.8i (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 3.8i (WRAP) == rigel2 == File: /data/rigel2.txt =====
```

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

11.1.4 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.

11.2 Functions

Functions can be used to manipulate 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 (^X-A) command, we would write:

```
set $fillcol &add $fillcol 2
 \       \       \       \      \____second operand
  \       \       \       \_________first operand
   \       \       \_________________function to execute
    \       \_____________________________variable to set
     \_________________________________set (^X-A) command
```

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

<num> an ascii string of digits which is interpeted 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 availible functions follows: (Once again, send in those votes on what kind of functions you would like to see added!) 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
&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>

String manipulation functions: (returns <str>)

&CAT	<str> <str>	Concatinate 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.
&UPPer	<str>	Uppercase <str>
&LOWer	<str>	lowercase <str>
&CHR	<num>	return a string with the character

		represented by ascii code <num>
&GTK		return a string containing a single keystroke from the user

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, or equal to <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.

Special Functions:

&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"
se      "giraffe"
set %t  "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.

11.3 Directives

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

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

11.3.1 !ENDM Directive

This directive is used to terminate a 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

26      store-macro
        find-file @"File to view: "
        add-mode "view"
        add-mode "red"
!endm

write-message "[Consult macro has been loaded]"
```

only the lines between the store-macro command and the !ENDM directive are stored in macro 26.

11.3.2 !FORCE Directive

When MicroEMACS executes a macro, if any command fails, the macro is terminated at that point. If a line is preceeded by a !FORCE directive, execution continues weather 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 irregardless
add-mode "red"
```

11.3.3 !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 expresion following the !IF directive evaluates to a TRUE value. For example, the following macro segment 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
```

11.3.4 !GOTO Directive

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

11.3.5 !RETURN Directive

The !RETURN Directive causes the current macro 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"
```

Appendix A

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	^X-M	Add a mode to the current buffer
add-global-mode	M-M	Add a global mode for all new buffers
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	^X-N	Change the name of the file in the current buffer

change-screen-size	M-^S	Chnage the number of lines of the screen currently being used
change-screen-width	M-^T	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
delete-blank-lines	^X-^O	Delete all blank lines around the cursor
delete-buffer	^X-K	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	^X-1	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-window	^X-0	Remove the current window from the screen
describe-bindings	(none)	Make a list of all legal commands
describe-key	^X-?	Describe what command is bound to a keystroke sequence
detab-line	^X-^D	Change all tabs in a line to the equivilant spaces
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

entab-line	^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	FNB	Execute a file as a macro
execute-macro	^X-E	Execute the keyboard macro (play back the recorded keystrokes)
execute-macro-<n>	(none)	Execute numbered macro <N> where <N> is an integer from 1 to 40
execute-named-command	M-X	Execute a command by name
execute-procedure	M-^E	Execute a procedure by name
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
forward-character	^F	Move cursor one character to the right
goto-line	M-G	Goto a numbered line
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	FN=	Hunt for the next match of the last search string
hunt-backward	FN>	Hunt for the last match of the last search string
help	M-?	Read EMACS.HLP into a buffer and display it
i-shell	^X-C	Shell up to a new command processor
incremental-search	^X-S	Search for a string, incrementally

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
list-buffers	^X-^B	List all existing buffers
meta-prefix	<ESC>	Key used to preceed all META commands
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
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 preceeding line
next-buffer	^X-X	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	^X-O	Move cursor to the next window
next-word	M-F	Move coursor to the beginning of the next word
open-line	^O	Open a line at the cursor
pipe-command	^X-@	Execute an external command and place its output in a buffer
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	^X-P	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
query-replace-string	M-^R	Reaplace all of one string with another string, interactivly quering 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
resize-window	^X-W	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 occurences of one string with another string from the cursor to the end of the buffer
reverse-incremental-search	^X-R	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	^X-B	Select a buffer to display in the current window
set	^X-A	Set a variable to a value
set-encryption-key	M-E	Set the encryption key of the current buffer
set-fill-column	^X-F	Set the current fill column
set-mark		Set the mark
shell-command	^X-!	Execute an external command

shrink-window	^X-^Z	Make the current window smaller
split-current-window	^X-2	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-line	^X-^T	Trim any trailing whitespace from line
unbind-key	M-^K	Unbind a key from a function
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
wrap-word	M-FNW	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 B

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 3.8i

^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 CR-LF, then indent		
^K	Kill forward	ESC K	Bind Key to function
^L	Refresh the screen	ESC L	Lower case word
^M	Insert CR-LF	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		
^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

ESC ^C	Count words in region	ESC ~	Unmark current buffer
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	Change screen rows		
ESC ^T	Change screen columns		
ESC ^V	Scroll next window down		
ESC ^W	Delete Paragraph		

ESC ^Z	Scroll next window up		
^X ?	Describe a key	^X !	Run 1 command in a subjob
^X =	Show the cursor position	^X @	Pipe DOS command to buffer
^X ^	Enlarge display window	^X #	Filter buffer thru DOS filter
^X 0	Delete current window	^X (	Begin macro
^X 1	Delete other windows	^X)	End macro
^X 2	Split current window		
		^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 processer
		^X D	Suspend MicroEMACS (BSD4.2 only)
		^X E	Execute macro
^X ^F	Find file	^X F	Set fill column
^X ^I	Insert file		
		^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 ^U	Upper case region		
^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

Only under PCDOS:

<ALT>-S Hunt forward	SHIFT <F1> - <F10>
<ALT>-R Hunt backward	Execute macroes 1 - 10

Usable Modes

WRAP	Lines going past right margin "wrap" to a new line
VIEW	Read-Only mode where no modifications are allowed

CMODE	Change behavior of some commands to work with C better
EXACT	Exact case matching on search strings
OVER	Overwrite typed characters instead of inserting them
CRYPT	Current buffer will be encrypted on write, decrypted on read
MAGIC	Use regular expresion matching in searches
ASAVE	Save the file every 256 inserted characters

WHITE/CYAN/MAGENTA/YELLOW/BLUE/RED/GREEN/BLACK	Sets forground color
white/cyan/magenta/yellow/blue/red/green/black	Sets background color

Appendix C

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	
	UNIX V7	native	
	BSD 4.2	native	job control supported
	*VMS	native	only some terminals supported
NCR Tower	UNIX V5	native	
Fortune 32:16	UNIX V7	native	
IBM-PC	MSDO 2.0 & 3.2	LATTICE 2.15	Large CODE/Large DATA
		AZTEC 3.4e	Small CODE/Large DATA
		*MSC 4.0	
		*MWC 86	

HP150	MSDOS	Lattice 2.15	Function key labels for the touch screen
HP110	MSDOS	Lattice 2.15 Aztec 3.4e	
*Data General 10	MSDOS	Lattice 2.15	
*Texas Instruments Professional	MSDOS	Lattice 2.15	
Amiga	Intuition	Lattice 3.03 *Aztec 3	no mouse or menus yet
ST520	TOS	Lattice 3.10	no menus yet, no shell commands

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

Macintosh	Finder 5.0	Aztec

*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 D

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.

D.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
Enhanced graphics Adapter	EGA	CGA

EMACS also takes advatage 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.

IBM PC function keys in MicroEmacs

	function	Function	^function	Alt-function
f1)	FN;	FNT	FN^	FNh
f2)	FN<	FNU	FN_	FNi
f3)	FN=	FNV	FN`	FNj
f4)	FN>	FNW	FNa	FNk
f5)	FN?	FNX	FNb	FNl
f6)	FN@	FNY	FNc	FNm
f7)	FNA	FNZ	FNd	FNn
f8)	FNB	FN[	FNe	FNo
f9)	FNC	FN\	FNf	FNp
f10)	FND	FN]	FNg	FNq
home)	FNG		FNw	
CuUp)	FNH			
PgUp)	FNI		FNä {Alt 132}	
CuLf)	FNK		FNs	

```
 5  )
CuRt)     FNM                              FNt
 End)     FNO                              FNu
CuDn)     FNP
PgDn)     FNQ                              FNv
 Ins)     FNR
 Del)     FNS
```


D.2 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.

```
Monitor $sres size #color $palette format
Color   LOW     40x25   16      0001112223334445556667777
        MEDIUM  80x25   4       000111222333
```

Mono	HIGH	80x25	2	000
	DENSE	80x40	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

Note: DENSE mode is not yet supported in 3.8i. It will be soon

Also the mouse generates FN prefix codes when moved, or when one of the two buttons is pressed. Initially the movement of the mouse is bound to movement of the cursor, and the left mouse button generates a set-mark (M-space) command. The cursor keys and the function keys are bound similarly to to IBM-PC.

D.3 Amiga 1000

The Commodore AMIGA 1000 version of MicroEMACS does not have

extensive support of the mouse or of pull down menus as of yet. It does however come up in a window, and it is possible to re-size it to run in different sized windows. The M-^S change-screen-size takes its numeric argument as the new number of lines for EMACS to use. The M-^T change-screen-width command allows you to change the number of columns EMACS will use. The defaults for these are 23 lines and 77 characters across for a full screen window.

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 compiliation.

D.4 UNIX V5, V7, and BSD4.[23]

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

Under systems which support job control, you can use the ^X-D 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 forground.

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: explination".

```
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: explination".

Index

Table of Contents