MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo No. 447

January 1978

An Introduction to the EMACS Editor

by

Eugene Ciccarelli

Abstract:

EMACS is a real-time editor primarily intended for display terminals. The intent of this memo is to describe EMACS in enough detail to allow a user to edit comfortably in most circumstances, knowing how to get more information if needed. Basic commands described cover buffer editing, file handling, and getting help. Two sections cover commands especially useful for editing LISP code, and text (word- and paragraph-commands). A brief "cultural interest" section describes the environment that supports EMACS commands.

Preface

This memo is aimed at users unfamiliar not only with the EMACS editor, but also with the ITS operating system. However, those who have used ITS before should be able to skip the few ITS-related parts without trouble. Newcomers to EMACS should at least read sections 1 through 5 to start with; after that I strongly urge them to try EMACS with what they know, relying on the help features (section 5), rather than attempting to memorize many different commands. Those with a basic knowledge of EMACS can use this memo too, skipping sections (primarily those toward the beginning) that they seem to know already. A rule of thumb for skipping sections is: skim the indented examples and make sure you recognize everything there. Note that the last section, "Pointers to Elsewhere", tells where some further information can be found.

Though EMACS can be used from a printing terminal, it is primarily intended for (and is much easier to use from) a display terminal. This memo describes the use of EMACS from a display *only*.

There can be a great deal of ambiguity regarding special characters, particularly control-characters, when referring to them in print. The following list gives examples of this memo's conventions for control-characters as typed on conventional terminals:

- ⌃A is control-A.
- ⌃@ is control-@.
  (Which you can also type, on most terminals, by typing control-space.)
- ⎋ is altmode (labeled "escape" on some terminals, but be careful: terminals with meta keys, e.g. the AI TV's, have both escape and altmode -- ⎋ is altmode).
- ⌃? is rubout, or delete.
- ⌃H is backspace.

When EMACS prints anything referring to control-characters, it will always represent them as up-arrowed letters, "^A" for control-A, "^?" for rubout. If a real up-arrow is ever intended, it will be followed by a space.

Finally, of all the people who have contributed to the development of EMACS, and the TECO behind it, special mention and appreciation go to Richard M. Stallman. He not only gave TECO the power and generality it has, but brought together the good ideas of many different Teco-function packages, added a tremendous amount of new ideas and environment, and created EMACS. Personally, one of the joys of my avocational life has been writing Teco/EMACS functions; what makes this fun and not painful is the rich set of tools to work with, all but a few of which have an "RMS" chiseled somewhere on them.

TABLE OF CONTENTS

1. Whither to Hither to Thither: starting EMACS

EMACS runs on the ITS machines, AI, ML, MC, and DM. For people coming from an Arpanet TIP, the host numbers are: 134 (AI), 198 (ML), 236 (MC), and 70 (DM). Tell the TIP to transmit every character as it is typed, and to let ITS echo:

```
@T E 1
@E R
@O 198
```

Log into ITS with a username of up to 6 letters, e.g. your initials (say XYZ). If you don't have a regular username there, you might first ensure that some other user doesn't already use your initials, with the WHOIS program (you type the colon before WHOIS):

```
:WHOIS XYZ
```

If WHOIS didn't find any regular user XYZ, you can use that name.

```
:LOGIN XYZ
```

You are now talking to DDT, the top-level, monitor program. Make sure the system knows your terminal type -- it will if you are directly connected or TELNET'ing from another ITS, but will not if you are coming from, say, a TIP. The TCTYP program tells the system your terminal type; you can get some help on using it by:

```
:TCTYP HELP
```

For instance, if you have a VT52, coming from a TIP, you can say:

```
:TCTYP VT52
```

Start up EMACS, running under DDT:

```
:EMACS
```

After you've had enough of EMACS, exit back to DDT by typing ⌘X⌘C (the normal way of exiting EMACS),

```
⌘X⌘C
```

or ⌘Z (a "BREAK" key), which will stop EMACS from whatever it was doing, and return to DDT, typing something like

```
⌘Z
4310) .IOT 1,16
```

(which is the location and instruction EMACS was executing).

Then you can log out of ITS by:

:LOGOUT

Many times, when ITS programs type out more than what will fit on the screen, they will pause after typing "--MORE--". When this happens, typeout will continue if you type a space; otherwise, the typeout is flushed (aborted).

For more information on ITS, you can read a primer entitled "An Introduction to ITS for the Macsyma User", by Ellen Lewis, available in room NE43-829. For some quick help, you can type :HELP <CR> to DDT. (If you are in EMACS, type ⌘C to get back to DDT. After getting help, you can continue EMACS with :CONTINUE <CR>.)

⌘C
:HELP
:CONTINUE

2. Basic Buffer-Editing Info

This section describes basic editing of text in the buffer from a display terminal. See the section on basic file-handling for how to transfer text in the buffer to and from ITS files.

In EMACS as soon as you type any character, some action is performed, and you see the resulting buffer. Generally, graphic, printing characters insert themselves, while control characters do the editing. Below is a simple description of what various control characters do:

^F	Move forward one character.
^B	Move backward one character.
^H	Move backward one character (same as ^B).
^D	Delete forward one character.
^?	Delete backward one character.
^A	Move to beginning of this line.
^E	Move to end of this line.
^N	Move to next line.
^P	Move to previous line.
^K	Kill rest of line.
^Y	Un-kill ("yank") what was just killed, inserting it into the buffer at the current position.
^@	Mark this place in the buffer.
^W	Kill ("wipe out") from here to marked place. (^Y will un-kill ^W-killed text too.)
^L	Clear screen, redisplay buffer.
^Q	"Quote": insert next character typed, whatever it is.
^X	Prefix character: follow with another character to make a 2-character ^X-*command*.
^X^C	Exit EMACS to DDT, normal method.
^Z	Stop EMACS, return to DDT.

An easy way to move text around is to kill it, move, then un-kill. You can un-kill any number of times to create copies in different places.

Near the bottom of the screen is a line that begins "EMACS " called the *mode line*. Ignoring for now the other information in that line, at the right end of it you may see a "--MORE--". This indicates that there is text in the buffer after what the screen is currently showing. By moving forward enough, you can get EMACS to "refresh" the screen so that it shows part of the buffer further down.

The action of most characters can be altered by providing an *argument* just before the character. This is usually a repetition count, e.g. an arg of 5 before ^D means "delete the next 5 characters". A general way of specifying an arg is typing ^U followed by digits, ending with the character to execute, e.g. ^U5^D gives ^D an arg of 5; ^U123^D gives ^D an arg of 123. Alternatively, you

can type ESC followed by the digits and the character to execute, e.g. ESC123X.[1]

For convenience, ^U can generate powers of 4 by repeated ^Us with no digits afterward: the number of ^Us is the power of 4, e.g. ^U sets an arg of 4, ^U^U an arg of 16, etc. These can be very convenient for moving around quickly, when the actual number of characters or lines moved by is not critical.

ESC123	Set an *argument* of 123 for the next character (if not a digit).
^U123	Set an *argument* of 123 for the next character.
^U	Set an *argument* of 4.
^U^U	Set an *argument* of 16.

Typing two altmodes, ESC ESC, puts you into a *mini-buffer*, occupying the top few lines of the screen, in which you can type (and edit -- it's a buffer) commands for actions not readily available by control-character keys. These commands are known as *MM-Commands*, since they must be preceded by "MM", e.g. ESC ESC MM Replace String ESCthisESCthatESCESC, which causes all strings "this" after the current position in the buffer to be changed to "that". Some of these MM-commands can take *numeric arguments*, e.g. ESC ESC 2MM Replace String ESCthisESCthatESCESC, which replaces only the next 2 "this" strings with "that". The *string arguments* following some MM-commands, like replace string, are separated by (one) altmode. Two altmodes typed consecutively ends the mini-buffer, causing the MM-command to execute. You will then be back in your normal buffer. MM-commands are described more fully in a separate section of this memo.

ESC ESC MM ... ESC ESC Enter *mini-buffer*, give an *MM-command*, and then execute it.

The EMACS "quit" is ^G. If EMACS is executing some command, that command will stop, and you will be back typing characters (or commands) into the buffer. To quit out of a mini-buffer, you may need two ^G's: one will empty the mini-buffer, and the next will quit out of it. (If you had already started an MM-command running by typing ESC ESC, then you are no longer in the mini-buffer, and so one quit will do.)

Finally, EMACS has several features to automatically document itself and help you find appropriate commands; these features are described in a separate section, but here is one feature that you can use immediately: typing ESC? (i.e. altmode then question mark), followed by a character (graphic or control) such as one of those listed above, will describe the action of that character, in case you have forgotten or are not sure of the specifics. E.g. typing ESC?^B will tell you that control-B moves back one character (or several).

ESC? Describe the next character's action.

---

1. This will *not* work on terminals with meta keys, e.g. the AI TV's. On such terminals you type the digits while holding down the control or meta key, or both.

3. Basic File-Handling Commands

While there are many more commands for manipulating files, this section will cover just enough to allow you to get started. After you have read the section on help, you can look for more file commands if you wish.

Typing ^X^R will let you type the name of a file to read into the buffer; terminate the filename with a carriage return. If there is no such file, "(new file)" is printed at the bottom of the screen -- this is an easy way to clear the buffer and announce your intention of creating a new file of the given name. This command sets the filename default, which is displayed near the bottom of the screen on the "EMACS..." mode line.

Typing ^X^S (not followed by anything) will write the buffer out if it has been modified. (The ^S is mnemonic for "Save".) The default filename is used.

You can write the buffer out to a different file than the default if you wish, by ^X^W followed by the file name, and a return.

^X^R *filename CR*	Read file into buffer.
^X^S	Save buffer if modified.
^X^W *filename CR*	Write buffer to file.

The filename can be edited a little with rubout. If you rubout past the beginning of the filename, EMACS takes that as an "abort" and cancels the command. Typing ^U will erase all that you have typed so far.

ITS files have 4 parts to their names, completely specified:

<device>: <directory>; <filename1> " " <filename2>

Generally, <device> is DSK:, and <directory> is your username, say XYZ;. Both those generally need not be specified unless they differ from the last file named.

<filename1> is usually mnemonically useful, e.g. "FOO", and may be up to 6 letters long (digits allowed). The best use for <filename2> is usually as a version number; ITS will automatically handle version numbers if <filename2> is ">" or "<". Writing file FOO > will generate a new version number as the <filename2>, e.g. writing FOO 134 (if FOO 133 existed). Reading file FOO > will read in the highest version of the file. Reading FOO < will read the lowest version of the file.

The best way to delete files is by using *MM Dired*^^, which is described in more detail in a later section, but if you want, you can run MM Dired and then type "?" to get help.

^^ MM Dired ^^ ?

4. Extended Character Set

(Unless otherwise stated, numbers are in octal.)

In normal buffer editing, EMACS runs a command whenever a character is typed; that command is determined by the character's ascii code, extended to cover two extra bits provided by the Knight Display (TV) terminals in the AI Lab. These extra bits, which are produced by special "shift" keys on the TV's, are *meta* and *control*. While on most keyboards typing control-A produces the same code as control-a, 001, on a TV the control key merely *or*'s a bit (200 octal) to the character's basic ascii code: typing control-A produces 301 octal ("A" = 101), control-a produces 341 ("a" = 141). Similarly, the meta key *or*'s a 400 bit to the character's code: meta-A is 501, meta-a is 541. The two keys may be combined: holding the control and meta keys down while typing "A", i.e. typing control-meta-A produces 701. And, characters like % ; : etc. can be controlified (or metized).

For choosing a character command to execute, EMACS maps control characters typed on conventional keyboards into the control-bit version using the upper-case character, e.g. typing ⌃E is converted into control-E, 305.

When referring to this extended character set, rather than a conventional terminal's keys, this memo (and EMACS documentation) will abbreviate the "control-" and "meta-" prefixes by "C-" and "M-".

To let the user with a normal keyboard simulate a TV's keyboard and run commands for characters like M-A, C-a, C-:, C-M-F, etc., there are three characters which change the following character's code: typing ⌥ *metizes* the following character, i.e. *or*'s in a 400 bit; typing ⌃ *controlifies* the next character, *or*'ing in a 200 bit; typing ⌃⌥ *control-metizes* the next character, *or*'ing in 600:

⌥A	becomes M-A.
⌥a	becomes M-a.
⌃A	becomes C-A, same as typing ⌃A.
⌃a	becomes C-a.
⌃?	becomes C-?.
⌃⌥A	becomes C-M-A.
⌃⌥.	becomes C-M-..
⌥⌃A	also becomes C-M-A (some people prefer it to ⌃⌥A).

As a convenience in typing, ⌃⌥A will become C-M-A, ⌃⌥B will become C-M-B, etc. This allows you to keep your finger on the control key, especially convenient for typing ⌃⌥⌃ (see section 7).

⌃⌥⌃	becomes C-M-C.

Note that this explains how mini-buffer and the character-describing commands are called: ⌥⌥ is Meta-Altmode, and the command for that character sets up the mini-buffer; ⌥? is Meta-?, which reads the next character (which may also be metized or controllified, e.g. you can type ⌥?⌃⌥A) and describes it. (The commands prefixed by ⌃X are different, however, and are described in another section; ⌃X is *not* an extended character set command.)

5. Help! -- Auto-Documenting Features of EMACS

The commands in this section are very important to know, since they allow you *not* to remember, if you so choose, or if you have no choice. Remember that each MM-command must be preceded by ⌥⌥ to run the mini-buffer, unless you are already there.

Commands may want to type more than what fits on the screen; if so, they will pause after filling the screen and position the cursor after "--MORE--" in the mode line at the bottom of the screen. If you type a space, typeout will continue; typing anything else will cause the typeout to be flushed, and the character you typed will be executed.

When a command has finished its typeout, the screen will not redisplay the buffer until you type something. Type a space to see the buffer again. If you type anything other than a space, that character will be executed (and probably the buffer displayed too).

See the "Pointers to Elsewhere" section for how to deal with suspected EMACS bugs.

5.1 MM Apropos ⌥string⌥⌥

This command lists all commands that have "string" in their names, any spaces in the string argument being significant, and briefly describes each command. Both MM-commands and character-commands (those run by typing a character in normal editing) are listed. Character-command names are preceded by "^R " to distinguish them from MM-commands (see section on environment for derivation of "^R"). And, you will be told which, if any, characters currently invoke the appropriate character-commands.

Thus, if you want to find out if there is any character-command to move forward past words, like unto C-F for characters, you could type:

⌥⌥ MM Apropos ⌥word⌥⌥

You would get a list of word-hacking commands, including:

^R Forward Word ^R Move forward over one word
which can be invoked via: Meta-F

The "^R Forward Word" is the command's name; the rest of the first line is the brief description (the "^R" there again specifies that this is a character-command). And, by typing M-F (e.g. ⌥F) you can move over a word. (And you might guess that it takes an optional argument, specifying how many words to move over.)

You can have MM Apropos look for several matches by separating the strings by ⌥6 (but since mini-buffer is an editable buffer, the ⌥6 will do something, so "quote" it: ⌥⌥6), e.g.:

⌥⌥ MM Apropos ⌥delete⌥⌥6kill⌥⌥

This would list commands with "delete" *or* "kill" in their names.

MM Apropos prints a list titled

Commands Defined by "MM ..." Variables:

MM-variables are essentially links to other MM-commands. E.g. you might type:

ESC ESC MM Apropos ^Sfind ESC ESC

And you might see an MM-variable listed:

MM FOO Find Outer Otter

This means the command MM Foo is linked to MM Find Outer Otter. Note that Apropos can find matches in the linked-from or linked-to MM-command name.

5.2 MM Describe ^S commandname ESC ESC

This gives a full description of any command, such as one listed by MM Apropos. The following two examples illustrate describing MM- and character-commands:

ESC ESC MM Describe ^S Replace String ESC ESC
ESC ESC MM Describe ^S ^R Forward Word ESC ESC

5.3 Meta-? -- Describe following character

This character-command will describe the command invoked by typing the following character. The character typed can be in the extended character set, and so may be modified by the meta-, control-, and control-meta- prefixes (ESC, ^, and ^^); in addition, a ^X-command may be given:

ESC ? ^O Describe what C-O does.
ESC ? ESC f Describe what M-f does.
ESC ? ^. Describe what C-. does.
ESC ? ^^. Describe what C-M-. does.
ESC ? ^X ^S Describe what ^X ^S does.

5.4 MM List Commands ESC ESC

This command lists all MM-commands, briefly describing each. Thus, if you do not have any good string to match appropriate commands with or are only interested in MM- and not character-commands, this is the thing to use.

5.5 MM List Redefinitions ⌘⌘

This command lists the commands invoked by every character in the extended character set that invokes something non-trivial (many of the basic control characters and graphic characters are considered "trivial" -- actually this command lists the characters that EMACS redefines from Teco; see the section on the environment). Thus you can see what Meta-this and Control-Meta-that will do, all in one fell swoop.

5.6 Control-X ? -- Describe ⌘X-command

⌘X? will describe one or all ⌘X-commands, depending on the character typed next: if you type "*", all the ⌘X-commands are listed and briefly described; if you type another character, that ⌘X-command is describe in detail:

⌘X?*	List, briefly describe all ⌘X-commands.
⌘X?⌘S	Describe ⌘X⌘S.

6. Useful MM-Commands

The following are only briefly described, to whet your appetite. For more info, do MM Describe on them.

Commands that deal with occurrences of a string in the buffer:

MM Replace String: replace all or n occurrences of a string.
MM Query Replace: show each occurrence and ask if replace.
MM Occur: list lines containing occurrence of a string.
MM How Many: count how many occurrences of a string.

Commands that help edit (English etc.) text:

MM Text Mode: set so other commands work best for editing text.
MM Auto Fill Mode: mode where typing space may break the line if it is getting too long.

(You will see that the above two commands cause the mode line to show "Text" and/or "Fill".)

Commands that help edit LISP code:

MM Lisp Mode: set so other commands work best for editing LISP. (Note that Auto Fill mode works well inside Lisp mode too.)

Directory-related commands:

MM List Files: list just the names of files in current directory. (To get a full listing do ↑↑ EY ↑↑. Note that EY is *not* an MM-command -- it is a raw teco command; do not put an "MM" in front.)
MM Clean Directory: good for cleaning excess versions of files that accumulate by use of ">" filename2 (a good practice). This command spots all files with more than 2 versions and asks if it can delete them.
MM Dired: "edit" a directory for more control over deleting files. See the section on MM Dired.

Miscellaneous others:

MM Untabify: change tabs to equivalent number of spaces, e.g. before moving a file to Multics. (ITS uses tabs every 8, Multics every 10.)
MM Tabify: change spaces to tabs wherever possible.

7. Useful Character-Commands (^R-Commands)

These sections only briefly describes some commands that are invocable in the default EMACS environment. For more info, use M-? on them. Below, these commands are named only by the characters that invoke them, not their long "^R ..." names. Arguments (e.g. specified with C-U) are written as numbers (or "n") before the character.

n C-L	Redisplay with current line made nth line on screen.
4 M-altmode	Enter mini-buffer with last mini's contents there already.
n C-Y	Un-kill n'th most recent thing killed.
M-Y	Correct an Un-kill to an earlier kill.
C-S	Flexible, incremental search for a string.
C-R	Backward search. These need to be M-?'ed.
M-<	Move to beginning of buffer.
M->	Move to end of buffer.
C-V	Move down to display the next screen.
M-V	Move up to display the previous screen.
C-M-C	Exit a "^R mode" that some commands may put you in, e.g. MM Dired's "E" command.

7.1 Useful Text Character-Commands

In the following, note the similarity to C-A, C-E, C-F etc.:

M-A	Move to beginning of sentence.
M-E	Move to end of sentence.
M-F	Move forward over word.
M-B	Move backward over word.
M-D	Delete forward word.
M-Rubout	Delete backward word.
M-Q	Fill or adjust paragraph.
M-[	Move to start of this (or last) paragraph.
M-]	Move to start of next paragraph.

7.2 Useful LISP Character-Commands

Again, note the similarity to C-A, C-E, C-F etc.:

C-M-A	Move to beginning of this (or previous) DEFUN.
C-M-E	Move to end of this DEFUN.
C-M-F	Move forward over S-expression.
C-M-B	Move backward over S-expression.
C-M-K	Delete next S-expression.
C-M-Rubout	Delete last S-expression.
C-M-N	Move forward over list.
C-M-P	Move backward over list.
C-M-(	Move up one level of list, backwards.
C-M-)	Move up one level of list, forwards.
Tab	Indent this line to make ground LISP code.
M-;	Indent for a comment, and insert ";".
C-M-Q	Indent each line of next S-expression, aligning comments as well.

The character-command ")" has an option where it will, in addition to inserting itself, show you the matching "(". You can cause this to happen by doing:

```
** lu*LISP ) Hack**
```

(Note that this is *not* an MM-command -- this is setting an EMACS variable. See the section on the environment.) After setting this option, typing ")" will insert ")", then momentarily position the cursor at the matching "(", returning to ")" after a second.

8. Mini-Buffer Details: MM-commands

This section provides you with a few of the details that may help you use or read the description for MM-commands.

The mini-buffer allows you to edit commands, which will be executed as soon as you type two consecutive altmodes. Most of the control characters still are available for editing the text (e.g. C-A, C-B, C-D, C-K...) but altmode is redefined to just insert itself to aid in separating string arguments. Thus, you do not have meta-commands available by altmode; however, ↑K is also defined to metize the next character, and can be used in place of altmode.

The general format for an MM-command is numeric arguments (at most 2 of them), followed by the MM-command name and altmode, followed by string arguments separated by altmodes:

<MM-command> ::= <numargs> MM <MM-name> $ <stringargs>
<numargs> ::= <null> | <n> | <n> , | <n> , <m>
<stringargs> ::= <null> | <string> $ <stringargs>

An MM-command name (e.g. "Replace String"), whether it occurs in <MM-name> or as a string argument to another MM-command, may be preceded by spaces, and only enough of the name to make it unambiguous need be typed: e.g. instead of "MM Replace String" you can just say "MM Replace". Another example: "MM Describe$ Replace$".

Generally, if an MM-command takes numeric args, you say either <n> or <n>,<m>. However there are some cases where it makes sense to say something like "1,MM ..." which has a *pre-comma arg* but not a second arg (which might have an undesired effect, like changing a default).

Many MM-commands may be specified in a mini-buffer. They may be concatenated on a line, but it is best to put them one to a line. (Some MM-commands return values -- they should not, but they do -- and this could provide that value as a <numarg> for the next MM-command. Putting them one to a line clears any stray values returned.) Thus, you might say:

$$
MM Auto Fill Mode$
MM Text Mode$$

The first MM-command ends with one altmode to delimit the <MM-name> (see above BNF for <MM-command>). The two altmodes at the end serve both to delimit the second MM-command and also to exit (and execute) the mini-buffer.

Many MM-commands use <string> arguments as patterns to search for, such as MM Apropos, MM Replace String, MM Query Replace, MM Occur, etc. In such <string> arguments, you may control the searching somewhat: <string1>↑Q↑Q<string2> (↑Q is the "quote" to insert the ↑Q) causes search for either <string1> *or* <string2>; ↑X in a string matches any character; there are others but this probably suffices for a while.

9. MM Dired: Directory Editing

^XMM Dired^X puts you into a buffer containing your directory and allows you to move around, marking files for deletion or examination. (Your previous buffer contents are still around; when you exit MM Dired, you will be back where you were.) To get info on using MM Dired, just run it and then type a "?". All of the EMACS commands for moving around, searching, etc. are still usable, but certain letters mark files to be deleted (type "D"), or let you look at them first (type "E"). Typing "Q" tells Dired you are ready to quit; it will list the files marked for deletion and ask if it's ok to get rid of them. Type "yes" (no return) to have the listed files deleted, "N" to continue in MM Dired (e.g. to "undelete" ("U") a highest-version file mistakenly marked for deletion). Most importantly, it will mark any file that is the highest version of that name with a ">" -- generally if you see any ">"s, think twice about saying "yes".

You can examine files using MM Dired by typing "E" while the cursor is on the desired file's line. You will then be in a buffer containing that file. This mode is not recommended for editing a file -- it is just for examining. When you are ready to return to MM Dired, type two control-C's (which becomes C-M-C, just as would ^C^C).

10. Marks and the Region

The section on basic buffer editing mentioned that typing ^@ marks the current position in the buffer, and that ^W kills text from the current position to the last-marked spot in the buffer. By definition, the *region* is that part of the buffer between the mark and the current point.

The *mark* identifies a buffer position by the character offset of that position, i.e. the number of characters in the buffer before the position, *at the time of marking.* If you insert characters before the mark, it will not indicate the same text as it did when it was set. Thus, marks are generally used either temporarily (e.g. killing text), or as approximate buffer positions.

Also, EMACS maintains a stack of marks; the command ^@ pushes a new mark. With an argument, it works differently: ^U^@ will "pop the mark into point", i.e. make the top mark be the current position and then pop it off. ^U^U^@ just pops the top mark and throws it away. The mark stack can hold up to 8 marks.

^@	Push point onto the mark stack.
^U^@	Pop the mark stack into point.
^U^U^@	Pop the mark stack.

Many commands act on the currently-defined region; for more information on these, try:
^X^H MM Apropos ^Smark^U^Uregion^X^S

11. The Environment: TECO, EMACS, Libraries

This section is included for cultural interest -- you can edit quite well without knowing any of this.

Basically, the environment is layered:

TECO:	string- and list-processing language/interpreter.
^R Mode:	the TECO command ⎋R enters real-time edit mode.
EMACS:	a set of TECO functions for powerful editing and extensible environment support.
Libraries:	extra TECO functions can be loaded, for personal tailoring of the environment or running infrequently-used functions.

11.1 TECO

TECO is a string- and list-processing language and interpreter, heavily slanted towards the writing of interactive, display-oriented programs that manipulate text, such as editors. Its features which relate to the EMACS environment comprise:

Buffer Display: TECO has primitive commands for displaying the buffer; it knows the terminal's characteristics and tries hard to redisplay as little as possible. (It keeps a hash code for each line that it thinks is on the terminal's screen and compares with hashes calculated for lines in the buffer's "window". For more detail see the section below on ^R-mode.)

Objects: An object in TECO is either a number, a string, a buffer, or an array (called a "q-vector"). Arrays contain objects, as in LISP, and can be grown or shrunk at any point (even in the middle) efficiently.

Buffers: TECO allows multiple buffers; they can be created and destroyed by the user, and are garbage-collected if need be. Each buffer comprises two contiguous areas of virtual memory, separated by a "gap". When inserting or deleting, the area before the gap contains text before the current point; the area after the gap contains text after the current point.

Windows: TECO has mechanisms for dividing the screen into several windows (only horizontal dividing lines). When a window is selected, TECO's buffer display only affects that window.

Variables: TECO provides both arbitrarily-named variables and a limited set of fast-access "q-registers" primarily for local temporaries; variables and q-registers can contain any TECO object. Some q-registers have a special system significance: e.g. whatever q-register ..O contains is the current buffer; q-register ..Q contains the symbol table for variables, whose names are written with surrounding altmodes, as in ⎋Comment Column⎋.

Functions: Any variable or q-register can contain a string, which can be evaluated ("macroed") as a function written in TECO. Functions can be given up to two evaluated prefix arguments and can read any number of unevaluated string arguments from the text following the call, as in: 1,3MqHello⎋There⎋ The function in q-register q is called with two prefix args (1 and 3), and can read the two string args "Hello" and "There". Functions may return up to two objects as values.

TECO Inits: These start-up files (functions) allow the user to set up functions, variables etc. Running EMACS is equivalent to running a TECO with the EMACS (grossly large) start-up. However, that is short-circuited: running EMACS provides an already-initialized TECO for convenience.

Pure-String Space: Teco can load specially-formatted files into an area where it will not garbage collect unused strings. This allows the EMACS functions to sit around, ready for use, without taking up q-registers or variables, and also allows these pages to be shared. TECO provides a means for finding the base of this area, and functions can then, by the loaded files' format, find functions by name. EMACS sets up such a special "pure-string function caller" in q-register M; thus typing "MM Foo⇧" calls M, which reads its string argument, looks for a pure-string function of that name, and then transfers to that function (here the one named "Foo").

11.2 Real-Time Mode (^R mode)

The ⍓R TECO command enters a (recursive) real-time mode which provides the mechanism for calling a function based on character code typed. Each code determines a special q-register name, e.g. C-A implies q-register .⍓A and M-d implies q-register ..⍓d. TECO provides built-in functions attached to some of the control characters, of which only C-B, C-D, C-F, C-O, and rubout survive in EMACS. ^R mode attempts to minimize redisplay when the buffer is changed; this mechanism includes, in addition to the basic line-hash-coding scheme mentioned above:

1) Each command reports the range of buffer that has been changed; ^R merges these together. When a line is displayed, ^R removes it from the range needing display. This exempts most lines from redisplay immediately.

2) ^R keeps track of where in the buffer each line starts, so that it can convert cursor positions to character numbers as of last redisplay (as opposed to using the current contents of the buffer).

3) ^R can detect that the bottom portion of screen text ought to be moved to a different position and move it, by using insert- or delete-line operations on terminals that support them.

4) ^R stops redisplaying whenever input appears, remembering how much was done so it knows where to start again after processing the input.

11.3 EMACS

EMACS is a set of functions, written in TECO, which reside in pure-string space; pointers to some of these are initially in ^R q-registers. Besides supplying many editing functions, EMACS provides support functions for accessing and changing the environment:

EMACS Inits: When started, EMACS will first look for a file named .EMACS (INIT). If found, its contents are evaluated, allowing the user to load his personal libraries or alter the environment set up by EMACS as he chooses.

Variables: Functions are provided to list, create, and alter variables; many variables exist already to control the action of various functions (e.g. ‡Auto Fill Mode‡ could control what space does).

Function Finder: Q-register .M (MM uses this) takes a name and returns a pointer to the pure-string function. Some search rules are used -- say it is looking for function "FOO": First, it checks for a variable named ‡MM FOO‡; this allows redefinitions and quicker access. (For instance, functions that indent use the "subroutine" function named "& Indent". A variable ‡MM & Indent‡ points to either "& Indent With Tabs" or "& Indent Without Tabs".) If no MM-variable is found, it searches loaded libraries in pure-string space, most-recently loaded first. Thus, personal libraries are checked before standard EMACS.

Buffers: EMACS allows the user to create named buffers and have many per-buffer characteristics, such as mode (Text, Lisp, PLI, Teco, Auto Fill), and default filename.

Windows: EMACS supports two windows; while in a selected window, any buffer may be selected. Typeout (e.g. by MM Describe) inside one window will stay there until the user edits in that window; thus, you can select the other window after typeout, and edit there, leaving the typeout in the previous window, a convenient feature. The sizes of the windows may be grown or shrunk by the user.

Library Generation: The user can create files of named TECO functions in a simple format, heavily commented, and then create a library from it (or combining it with other library sources). Such sources have the form <function name> <function description> <function body> <function name> <function description>... Three major functions are performed by MM Generate Library: comments and extra white-space in the function body are removed, the descriptions are separated into "functions" of their own (named "~DOC~ <function name>" -- used by MM Describe etc.), and finally the result is "purified", i.e. converted into the form for the pure-string area, with the names in a sorted list.

12. Pointers to Elsewhere: further information

EMACS: Much information about using EMACS is available through MM Apropos, MM List Commands, MM Describe, etc. For instance, multiple buffers and windows are very useful mechanisms, and there are several MM- and character-commands that deal with them; such commands generally have "window", "buffer", or "file" in their names -- so try MM Apropos on them.

You can also use the INFO program; you can run this program from within EMACS by typing ^XI or ESC MM Info$$. INFO is self-documenting; I suggest that you run it and type "H" to learn about it.

There are two files, EMACS; EMACS CHART and EMACS; EMACS DOC, which list all character- and MM-commands. EMACS CHART is a brief summary of the invocable character-commands, while EMACS DOC gives an MM Describe$-like description of all invocable character-commands and all the MM-commands.

There is a mailing list, INFO-EMACS, for messages concerning EMACS (e.g. changes, new features). You can ask someone to put you on this list and watch how it's done (not hard to do).

If you think you have encountered an EMACS (or TECO) bug, first try to ask an experienced EMACS or TECO person to make sure you are not confused about something. If it is a bug, report it by:

> :BUG EMACS <report what happened in enough detail to hopefully allow someone to repeat and fix the bug> ^C

If ever you suddenly find that EMACS (TECO actually) prints something like:

> .VAL 0; 4510>> JRST 4124

(The ".VAL" is significant -- the 4510, "JRST", and 4124 above are just examples.) This is a "never-supposed-to-happen" error which returns you to DDT; do the following:

> ^.$$0G
> :PDUMP CRASH; TECO >

Then :BUG EMACS as described above, reporting what happened, mentioning that there is a "dump" in CRASH;TECO >.

TECO: A primer on Teco is in the file ".TECO.;TECO PRIMER". Complete (though much less digestible) documentation on Teco commands exists in the file ".TECO.;TECORD >". The primer is quite short and printable; the complete documentation is very long. TECORD is frequently placed in a buffer, and EMACS used for finding information on a desired Teco command (each command starts in column 0, whereas all descriptions are indented). Alternatively, you can use the MM TECDOC command to look up Teco commands in TECORD.

Word Abbrev Mode: There is a library of EMACS commands that allow you to define

abbreviations that automatically expand as you type them. You can use the INFO program to learn about this: to INFO type "M EMACS" then "M Word Abbrev Mode".

ITS: See the memo "An Introduction to ITS for the Macsyma User" by Ellen Lewis, available in room NE43-829. After that, use the INFO program (type "M DDT" at it) to learn about DDT.