PMATE rev 2

User Manual

and

Interface Guide

Written by
Michael Aronson
Aox Incorporated

Preface
HOW TO USE THIS MANUAL

Documenting PMATE presents problems. PMATE is useful at many levels, by persons of vastly differing computer skills. The documentation must then also be geared to a wide audience. In this uncoventionally organized manual, some ideas and commands are presented repeatedly in chapter after chapter -- each time, at a different level. Depending upon your experience and aptitude, you need only read the appropriate chapters.

Chapter I is an introduction for beginners. It is intended to be a complete course in the most basic use of PMATE for those with little or no computer background.

Chapter II starts in with basic concepts to lay the foundation for a more thorough understanding of PMATE. You can certainly start here if you have experience with other text editors.

Chapter III explains some more advanced concepts and commands.

Chapter IV gives the complete command set. You can start here if you really know what you're doing.

Chapter V provides macro examples. After you are well aquainted with PMATE, work through these examples. They show how macros can greatly expand PMATE's built-in capabilities.

Chapter VI provides configuration information. Read this chapter to get PMATE up and running on your system.

Chapter VII provides further configuration information for those who would like to interface with PMATE in assembly language.

Appendix A is a summary of PMATE commands -- a useful reference.

Chapter I
INTRODUCTION FOR BEGINNERS

TEXT EDITING, WORD PROCESSING, OUTPUT PROCESSING...

A 'text editor' is a computer program which helps people to create text, and modify it. Text editors are written by programmers, and programmers write mostly programs. Therefore, text editors were traditionally geared toward entering and correcting computer programs. In particular, a line of computer code is always one line. You might wish to get rid of it, modify it, move it somewhere else, or make a copy of it, but it is still a line. Writing the English language is different. If you want to insert a few words in a line, the end of that line needs to wrap around to the beginning of the next, and the whole paragraph needs rearranging. When programmers had satisfied their own needs, they attacked this sort of problem with 'word-processing' software and hardware, ranging in complexity from typewriters a bit smarter than usual, to complete computing systems with CRT's (TV displays) instead of paper to display the typed material, disks for storing the text, and fast printers for quick, error-free type out.

Typically, a 'text editor' program was meant for use on a teletype, or some other type of 'slow' terminal. It would be intolerable to wait while the terminal prints out all your text every time you make the slightest change in it. So you work blind. Sections of text are typed out only when you request. To make up for the inconvenience, powerful text editing commands can usually be constructed that can totally rearrange all your text with a few keystrokes. These little command 'programs' are dearly loved by computer nuts.

On the other hand, word processing software usually shows you exactly what your text is looking like at that time by showing it to you on a video display. As you change the text, the display changes instantly. The penalty you pay to achieve this desirable situation is that the word processor usually allows you to make only simple-minded changes in the text.

A 'text output processor' is an attempt to make a 'text editor' compatible with the real world of pages, paragraphs, underlines, and such. You use your text editor to enter text, including a bunch of control words. These words might indicate that it's time to start a new page, indent that margin 20 spaces, or leave 5 blank lines. Then you run the output of the text editor through your text output processor, which types it out on your printer, nicely formatted, maybe even with straight right hand margins! Text output processors give you wonderful control over the form of your final document, but have the disadvantage that the required input - the part created and updated by your text editor - bears little resemblance to the final output.

...AND PMATE

PMATE is an attempt to combine some of the best features of all three.

THE EDITING PROCESS

Text is saved on the disk in the form of 'files'. Each file has a name. In CP/M, a name consists of 8 or less characters - 'JOHN', 'CHAPTER1', 'PMATE', '+&', or 'GERTRUDE' are all fine possible names for a file. A filename can optionally have an 'extension'. This is up to 3 characters following a '.' at the end of the name. So 'BOOK.ONE' and 'BOOK.TWO' are different files, possibly representing

two different chapters in a book. Certain extensions (such as 'ASM','COM', and 'HEX') have meanings to the CP/M system programs.

PMATE can be used to create a new file, or to modify (change, add to, look at, or otherwise work on) an old one. Let's start by creating a new one. With CP/M in control (there should be an 'A>' or a 'B>' on the console), type the command:

PMATE FILE

where 'FILE' is the name of the file you wish to create (which may even be 'FILE'). After a few seconds, PMATE is loaded into memory, and the screen becomes alive. Look up at the top line. There are a few words and numbers telling you something about what is going on with the editing process. This is called the 'status line'.

Right below the status line is the 'command line'. Below that, a row of dots separates that command and status area from the 'text area'. Notice the 'cursor' in the text area. It is probably blinking, or reversed video; and notice the 'cursor' on the command line - just an underline (_). These cursors show where things will happen - where the action is. Now strike a few keys on the keyboard. The corresponding letters should appear in the command line. This is because you are in 'COMMAND MODE'. The keys you strike go toward forming a command, which then can affect the text area. Strike a few more keys, and also watch what happens when you hit the carriage return key. Now hit the 'delete' or 'rubout' key. This will wipe out the last character in the command line. You can delete the whole command at once with a 'control-C'. That is the character produced by holding down the 'control' key, and striking a 'C'. Control characters will be indicated from now on by an '©' (a control-C will be written '©C').

Now clear the whole command area with a '©C'. Strike a '©N'. You have just moved into 'INSERT MODE', and this is indicated where the command line used to be. Now enter some normal letters. You are creating text! Enjoy it for a little while. Then try hitting '©X' -this will return you to COMMAND MODE. Now go back to INSERT MODE. Try the delete key. It will delete the character you just entered. It's just like typing on a typewriter, but you can correct your mistakes.

'©C',©N', and '©X' are called 'Instant Commands'. When you hit these keys, a character doesn't get entered in the text, or onto the screen, but rather, something different happens. Instant commands are usually good in either COMMAND or INSERT MODE. The most important instant commands are those that can move the cursor around in the text area. They are '©Y', '©B', '©G', and '©H'. Try them. Note that these 4 keys are located in a group on the keyboard, and that they are arranged the right way -- that is, the bottom one, '©B', moves the cursor down, the top '©Y', moves it up, '©G' moves it left, and '©H' moves it right.

Use the instant commands to move the cursor around, and then try typing in some more characters. Now you are 'editing' text, as opposed to just entering it. Learn the action of the 'delete key', it always deletes the character just before the cursor. Try the '©D' instant command - it deletes the character at the cursor. '©K' will kill the whole line, starting at the cursor. Lines are separated by the CR character. Type a carriage return, and you will start a new line. Move the cursor to a CR (the last character on any row) and try deleting it. See how the two lines on either side get run together?.

If you haven't already done so, type in more lines than there is room for on the display. Watch how the display scrolls. Move the cursor back a few lines, and the display will scroll back the other way. PMATE will display as much text as possible on both sides of the cursor. This is called 'vertical scrolling'.

Now type a line that is too long for the screen - just keep typing without hitting a CR. As soon as the cursor is about to move off the screen, the whole display 'shifts' over. This is called 'horizontal scrolling'. The display continues to

shift as necessary, as your line gets longer and longer -- up to a maximum of 250 characters. Move on to the next line, and the display will return to normal.

Try out the tab key (or '©I' if your keyboard doesn't have one). Tab stops are initially placed every 8 spaces, but later you will learn to place them arbitrarily. If your keyboard isn't capable of generating lower case, use the '/' key as a 'shift'. When you type '/', nothing will happen, but the next key you type will be shifted in case. (To actually enter a '/', just type it twice.) Try the '©S' instant command a few times. It changes the 'default case' from upper to lower, and then back again. The '/' key (like 'shift') will alter the case from the default for just one character.

As you enter text, and it appears in the text area, it is being entered into the 'current edit buffer'. This edit buffer is just a section of computer memory. The text area of the display is acting like a 'window' which allows you to see a small section of this buffer. The section you do see is the section where the action is about to occur - the place where the cursor points.

Now go back to COMMAND MODE ('©X' - remember?). Any characters you type now are entered into the 'command buffer'. The command area of the display just shows the final part of this buffer. Make sure the command area is clear (use a '©C' if necessary). Now find the 'escape' key (it might be marked 'ESC'). Strike it a few times. It echoes on the screen as a '$'. From now on, '$', when written as part of a command, will mean the 'escape key'. Commands are executed by typing an escape twice. Try this simple command:

```
DSS
```

This command deletes the character at the cursor, just like the '©D' instant command. (Incidentally, don't forget to use the instant commands in COMMAND MODE also - especially the cursor motion controls.) 'KSS' will kill the whole line. Now try 'IGARBAGESS' . The 'I' is the insert command, and what ever follows it until an escape is reached, will be inserted into the text.

Sooner or later, you are bound to give PMATE a command it doesn't like. It will complain bitterly by displaying an error message, and appearing quite dead. The only way to wake it is by typing either a 'space' or a CR. In big letters:

WHEN YOU HAVE MADE AN ERROR, AN ERROR
MESSAGE WILL APPEAR WHICH WILL REFUSE TO GO
AWAY UNLESS YOU STRIKE CR OR SPACE BAR

After a command is executed, it is still displayed in the command area. If you now type an escape again, the command will be repeated. (Try this after an 'I' command and see how fast you can fill up space). If you don't want to repeat the command, but just to enter a new one, ignore it. When you start to enter the new command, the old one will just disappear.

One very important command is 'FSS'. This takes you into 'FORMAT MODE', and if you are already there, it takes you back again. In FORMAT MODE, you don't have to worry about where a line ends, PMATE takes care of that for you. Try it. After giving the 'F' command, enter insert mode. Now start typing words, but don't enter any CR's. Notice how the text automatically wraps around to the next line, rather than going off the screen. Words are not truncated or divided, but are left intact. When you reach the end of the paragraph, and you insist that the next word begins on a new line, then enter a CR. Write a few paragraphs. Move the cursor to the middle of one of them, and insert some more words. Watch how the whole

paragraph is rearranged as those words are entered. It is important not to use a CR except when the next word must begin a new line. Otherwise, words will not be able to wrap around properly when changes are later made.

Now that you have some reasonable text in your edit buffer, why not write it on the disk?. Use the command 'XES$'. That ends the editing pass. 'XH$$' will then get you back to CP/M with a new file on the disk. Soon, you will want to modify that file. Again, give the command

PMATE FILE

where 'FILE' is the name of the file you had just created. Quickly, PMATE is back in control, and you see your old text again. Now add to it, rearrange it, chop it up, or whatever you please. If you don't like what you did, type 'XK$$'. This just cancels any changes, and leaves 'FILE' as it used to be. Otherwise, type 'XES$' and 'FILE' is updated on the disk to include all your modifications. (It is also possible to leave 'FILE' intact, and write the new version under a different name.) Don't forget that the only way to get back to CP/M is with an 'XH$$' command ('H' for 'Home').

Chapter II
BASIC IDEAS

THE BUFFERS

PMATE operates on text stored in the computer's memory. Text can be placed in any of 11 'edit buffers'. Text is stored as a continuous stream of characters. If a character is inserted in the text, all following characters have to be moved up to make room for the new one. Similarly, if a character is to be deleted, all following characters must be moved down to fill in the space. 'Lines' are separated by the carriage return character (CR).

THE DISPLAY

PMATE utilizes a video display to always show a portion of the text in the current edit buffer. As the text stream in memory is modified, the display immediately reflects this change. Just imagine your text sits behind a large wall, and your screen is an opening through which you can see part of it.

THE CURSOR

It's nice to be able to see part of your text stream, but obviously not enough. You need to be able to modify it. The 'cursor' indicates on the screen exactly where this modification is to take place. The character to which the cursor points is clearly displayed, usually as blinking, underlined, or inverse video. The cursor is only useful if it can be moved around. This can easily be done just by striking the appropriate key on your keyboard. As the cursor moves through the text, the portion of the text displayed on the screen changes in such a way that text on either side of the cursor is always visible. The screen always shows where the action is. In this way, text scrolls up or down as you move the cursor up or down (vertical scrolling). Furthermore, if a line is too long to fit on the screen, the whole text display is shifted over to prevent the cursor from moving off the right end of the screen (horizontal scrolling). Lines can be up to 250 characters in length.

MODES OF OPERATION

There are several ways to enter or modify text. 'OVERTYPE MODE' or 'INSERT MODE' provide the simplest method. Just type!! The characters you type are immediately entered into the text where the cursor is (and of course, appear on the screen). In 'INSERT MODE' any characters at or beyond the cursor are moved up to make room for the new ones. If you make a mistake, 'rubout' or 'delete' will cure it (and banish it from the screen). In 'OVERTYPE MODE', the character you type replaces the one already beneath the cursor. However, neither carriage returns, nor tabs will be overwritten. OVERTYPE or INSERT modes are indicated on the bottom line of the display, below the row of dots.

Several keystrokes (usually control codes) are not entered into the text, but serve some other function. These are called 'Instant Commands'. The keystrokes that move the cursor are instant commands. Other keystrokes can delete the character at the cursor, delete a whole line, or shift from upper to lower case. This sort of text editing is great for entering text, and making minor changes in it.

For 'serious' text editing, PMATE executes 'command strings'. For instance, you might command PMATE: "Find the third occurence of 'George' and change him to 'Harry', then from that point, delete all characters until you find an 'F', then insert the numbers from 240 to 1000 in base 5, one per line, and finally tell me how much (3*46/(5+(3*7))) is". Of course, you wouldn't use exactly that language, but such a command string could readily be constructed. PMATE executes such commands when in 'COMMAND MODE'. The 'command line' is the last line of the display. When there is no mode message in the command line to inidicate OVERTYPE or INSERT modes, PMATE is in command mode. An underline cursor indicates where the next keystroke is to be entered.

In COMMAND MODE, your keystrokes don't immediately affect the text, but rather are entered into the command buffer, and appear on the command line. A single command is usually one or two characters, but commands can be strung together to form command strings. As soon as the command is executed, the display shows the updated edit buffer, centered on the cursor. While in COMMAND MODE, PMATE also recognizes instant commands. These keystrokes are not entered into the command buffer, but are executed immediately, affecting the text and display. Included among the instant commands are ones to shift modes - enter COMMAND MODE, enter OVERTYPE MODE, or enter INSERT MODE.

LINE FORMATTING

A 'line' is just a string of characters which ends with a carriage return. When you enter a CR character, the cursor moves down to the beginning of the next line. PMATE also has an automatic line formatting facility, for entering and editing textual material. When operating in this mode, text will automatically 'wrap around' as you enter it after filling out a specified line length. Words will not be broken up, however - the complete word will be moved down to the next line. This line formatting is preserved, even as portions of the text are edited. When operating in this mode, text is normally entered without any carriage returns. Any carriage return will always indicate the end of a line. They must be used only at the end of a paragraph, or whenever the following characters must appear on the next line.

INSTANT COMMANDS

In any mode, instant commands are keystrokes which are not entered into the command or text, but rather have some other immediate effect. The instant commands and a description of their action follows. A '©' is used to indicate a control code, so '©A' indicates the character resulting from holding down the control key and striking 'A'. This choice of control characters for the instant commands was made either for mnemonic value, or for convenience of location. For instance, the 4 commands which move the cursor one position are located in the center of the keyboard. These instant commands will be used all the time. It is important that they should be easily accessible. If you are at all unhappy with this assignment, it is an easy matter to change it (see the Interface Guide). In particular, if your keyboard has a cursor pad, or other single stroke control keys, these may be used for the more important instant commands.

Mode Switching:

The following instant commands set the mode of the editor to INSERT, OVERTYPE, or COMMAND.

©X Go into COMMAND MODE.

©N Go into INSERT MODE.

©V Go into OVERTYPE MODE.

Cursor Motion:

The cursor motion instant commands allow the cursor to be easily moved throughout the text buffer. As the cursor is moved, the display updates in order to keep the display centered on the cursor. The cursor is never allowed to move outside of the text buffer.

©A Move the cursor up to the beginning of the text buffer. If it is already at the beginning, move it to the end. So hitting this key once gets you to the beginning, twice gets you to the end.

©G Move the cursor to the left.

©H Move the cursor to the right.

©B Move the cursor down one line.

©Y Move the cursor up one line.

©U Move the cursor up six lines.

©J Move the cursor down six lines.

©P Move the cursor to the beginning of the following word. Words are separated by any combination of spaces, tabs, and carriage returns.

©O Move the cursor to the beginning of the current word. if it is already there, move the cursor to the beginning of the preceding word.

Deletion:

The deletion instant commands allow characters to be removed from the text. The rest of the text buffer is moved down to fill in the space formerly occupied by the deleted characters.

©D Delete the character at the cursor.

©K Delete the rest of the line, starting at the cursor.

©W Delete the next word, starting at the cursor.

©Q Delete the word preceding the cursor.

rubout Delete the character just entered. When in command mode, this deletes the character just entered into the command. When in insert mode, this deletes the character just before the cursor (©D deletes the character at the cursor). This is usually the character just entered into

the text.

Moving text:

These commands provide an easy method of copying or moving sections of text.

©T Tag the current location. This marks one end of the text to be moved.

©E Move the section of text between the tagged location and the present location of the cursor to a special buffer. The text is deleted from the current text buffer.

©Z Insert the contents of the special buffer at the cursor location.

In other words, to move a block of text, go to the beginning of the block, type '©T', go to the end, type '©E'. Move the cursor to the desired destination, and type '©Z'. In order to copy a section of text without deleting it from its original location, it is only necessary to type '©Z' immediately after the '©E'. Subsequent '©Z's will then produce copies of the text elsewhere.

Miscellaneous:

©_ Edit command string. If an error is made in entering a long command string, the command string itself can be edited. When ©_ is typed, the old command buffer becomes the text buffer, and can be edited just like text. Striking ©_ again restores the old text, and the new updated command string is returned to the command area, ready for execution.

©C Abort. Clears the command area. If ©C is typed while a command string is being executed, execution will be aborted at the earliest opportunity.

©L Insert a line. Inserts a new line into the text, and leaves the cursor at the beginning of the new line.

©T Tag. Tag current cursor position - more on this later.

©F Redraw display.

©S Shift default case. Case will toggle between upper and lower.

©R Restore last deleted item at cursor position.

CURSOR MOTION

The cursor control 'diamond' (consisting of ©Y,©H,©G, and ©B) behaves differently, depending upon the PMATE configuration. In one possible mode, ©Y and ©B always leave the cursor at the beginning of a line. This is particularly useful for editing line-oriented text, such as programs.

In another mode, ©Y will move the cursor immediately above its current position, leaving it in the same column if possible. Since the cursor must always stay on text, it is not possible to leave the cursor to the right of the carriage return ending the line, or in the middle of a tab. The cursor is then positioned on the last

possible position to the left of the desired column.

Finally, PMATE does have the capability of being able to leave the cursor in this 'free space' at the right of a line ending, or in the middle of a tab. When configured in this manner, the cursor can move anywhere on the screen (except below the very last line of text). When you attempt to enter text in 'free space', PMATE fills in the appropriate number of spaces, so that text appears to be entered right where you expect -- at the cursor.

COMMANDS

A single command consists of one, two, or three characters which are entered into the command buffer - just type and watch them appear there. The command is then executed by striking the 'escape' key twice. Escapes appear on the screen as dollar signs '$'. In the rest of this manual, '$' will indicate an escape. Carriage returns appear in the command line as '<', and any other control character is echoed with a carat (^). An example of a command is the character 'D', which will delete the character at the cursor. Suppose the edit buffer contained

> This is an example of some text which needs
> correcting. The cursor on the scrreen is
> indicated by the underline.

Then just type in the command

DSS

(remember that '$' is an escape, and the two escapes are necessary to execute the command 'D') This is what will be left on the screen:

> This is an example of some text which needs
> correcting. The cursor on the screen is
> indicated by the underline.

NUMERIC ARGUMENTS

Many commands can take 'numeric arguments'. That is just a number that precedes the command that gives additional information to PMATE. For example, while 'D$$' deletes one character from the text, '3D$$' deletes the next 3 characters. Numeric arguments can take integer values from -32768 to +32767. They can be complex expressions, but this will be explained in greater detail later. If an argument is missing, it is usually taken to be 1. There are exceptions, but these will be explicitly mentioned. Typing just a minus sign '-' before a command is usually equivalent to -1.

COMMAND STRINGS

PMATE derives much of its real power from being able to string together a number of commands to form a command string. Commands can be typed in together to form command strings. 'M' is the command to move the cursor a number of character positions. 'M$$' will move the cursor over one character. the command string 'D5MD$$' will delete one character, then move over 5 and delete that character. As the command string is entered, it appears on the bottom of the screen, and it is not until two consecutive escapes have been entered that any

change takes place in the text. At this time, the whole command string is executed. Single escapes can be freely inserted between commands without causing execution. So 'D5MD$$', 'D5M$D$$' and 'D$5M$D$$' all have the same effect.

STRING ARGUMENTS

When certain commands are used in a command string, they require a single escape to separate them from the following command. While numeric arguments often precede commands, some commands are followed by 'string arguments'. These arguments are just a string of characters which you might insert into the text, search for, or perform some other operation. For example 'I' is the command that inserts the string argument following it into the text buffer. Suppose the text buffer contained:

PMATE is a very easy to use and helpful
text editing program.

Typing the command 'Inot $$' might help you express your true feelings, leaving the display reading:

PMATE is not a very easy to use and helpful
text editing program.

If we wish to enter an insert command 'I' as part of a command string, we are faced with the problem of how to indicate to PMATE that the string argument is finished with, and the next command is being entered. We do this by using one escape to separate the string argument from the following command.
If we wish to now change 'a very easy' to 'an easy', we want to move the cursor one position, insert an 'n' there, and then delete the next five characters. We try 'M1n5D$$', but are dismayed to find we are left with:

PMATE is not an5D_very easy to use and helpful
text editing program.

While '-2M7D' will repair the damage, we should have commanded 'M1n$5D$$' in the first place.

REEXECUTING COMMANDS

What happens to the command string after it has been executed? Simple -it's still there, all ready to be used again. It still sits in the command area of the display, followed by the two escapes which caused it to be executed. If you now type another escape this command will be repeated. If a rubout is typed, the second escape will be deleted, and the old command string can be modified or extended. If any other command character is entered, the old command string will disappear, and this new character becomes the first in a new command string.

The ability to easily repeat commands can be extremely useful. Here is one of many situations where this facility is commonly used. 'S' is the search command. 'Hello$$' will search through the text starting at the cursor, and leave the cursor pointing just after the first 'hello' that it finds. The text display shows you immediately if this is the occurence of 'hello' which you were interested in. If not, just strike the escape key again, and PMATE will find the next one. Continue until

you have located the section you want.

ERROR MESSAGES

Some commands and conditions will produce error messages. These messages are usually self-explanatory. If 'ShelloSS' command is executed, and 'hello' cannot be found, then a message saying 'STRING NOT FOUND' will appear where the text used to be. PMATE terminates execution of the command string as soon as a command produces an error. The cursor in the command display area will point to the command just after the offending one.

AT THIS TIME, THE ERROR MESSAGE WILL REFUSE TO GO AWAY UNTIL YOU STRIKE EITHER "CR" OR THE SPACE BAR.

After typing one of these keys, the command still sits in the command area as if it had finished executing. It can now be reexecuted, modified, or ignored.

BASIC COMMANDS

PMATE has enough commands to keep you busy for a long time mastering them all. However, there are a few basic ones which you will use over, and over. They are all that are really necessary to satisy most text editing needs. A complete description of all commands follows later. 'n' indicates a numeric argument.

nD — Delete n characters starting at the cursor. If n is missing, it is assumed to be 1.

nK — Kill n lines starting at the cursor. If n is missing, it is assumed to be 1.

I — Insert the string which follows. The string ends with an escape. 'IgarbageSS' inserts 'garbage' just in front of the cursor.

S — Search for the string which follows 'S'. The string ends with an escape. The search starts at the cursor. 'SgarbageSS' causes PMATE to look through the text for 'garbage' and leave the cursor pointing just after the next occurence . If the string is not found, an error message is produced. (Remember, don't forget to hit carriage return after an error!!)

C — Change the first occurence of the first string following to the next string following. 'CgarbageSjunkSS' will search for 'garbage' and if it is found, change it to 'junk'. If 'garbage' is not found, an error message is given. Remember that search for garbage begins at the cursor.

--The following commands are very useful for moving blocks of text around:

nBC — Copy n lines of text into a special buffer. If n is missing, it is assumed to be 1.

nBM — Move n lines of text into a special buffer. If n is missing, it is assumed to be 1.

BG Insert contents of special buffer into text just before cursor.

'BM' is like 'BC' except that the lines that are copied into the special buffer are then deleted from the text buffer. To move 5 lines of text, position the cursor at the beginning of the lines to be moved (using the cursor control instant commands). Then type the command '5BM$$' . The 5 lines will disappear from the text. Then move the cursor to the place you wish the lines to be, and type 'BG$$' . This restores the lines. The special buffer still contains those 5 lines.

TAGS

If you want to move a large block of text around, it may not be obvious how many lines are in this block to move. There is an alternative to counting lines. First set the cursor to the beginning of the section of interest. Use a '©T' instant command to 'tag' that location. Now move the cursor to the end of the block. The special symbol '#' will cause the next command to act on this whole block. So '#BC' will copy the block, and '#K' will delete it. The block can be defined with the tagged position at the beginning, and the cursor at the end, or vica-versa.

The '#' can be used in front of any command which takes a numeric argument to indicate the number of lines or characters to act upon (such as 'D' or 'K'). Furthermore, even if the command normally acts upon a fixed number of lines (such as the 'B' commands), by tagging a position, parts of a line can be moved.

THE GARBAGE STACK

When PMATE deletes text, it dumps it on a 'garbage stack'. A certain amount of space is reserved for this stack, and any remaining memory space not used by text is also used for piling up garbage. If you accidentally delete a line, it is then easy to recover with a '©R', which 'pops' the last item off the stack, and puts it back in the text. If you have just typed '©K©K©K©K', all the damage can be recovered by typing '©R©R©R©R'. It is the most recently deleted item which is available first, and the items long ago lost which may have gone permanently out to pasture - if there was not enough memory space left to hold all the deleted items.

The garbage stack also provides a very easy method of moving a bit of text around. For instance, to move a line of text, put the cursor at the beginning of that line and type '©K'. Then move the cursor to the required destination, and type '©R'. Use '©W', and it is very easy to move a word or two around in a sentence.

OPERATION UNDER CPM

PMATE exists as the CPM command file, PMATE.COM , and is called by typing:

PMATE

After a few seconds, PMATE will come on the screen, in Command Mode. You can now enter and edit text. If you wish to save your work on the disk, you need to define an 'output file':

XFfile Create and open 'file' for output (assuming a file by this name doesn't already exist).

Then when your done, use one of these:

XE End edit pass by writing entire text buffer to output file, and closing it.

XK End edit pass without writing anything on the disk.

Both of these commands wipe out what's in your text buffer. The first saves it on the disk, but watch out for 'XK'! To get back to CP/M, it is necessary to use:

XH Go 'Home' to CP/M. To prevent you from inadvertently exiting without writing desirable text to the disk, this command will give an error message if there are files open. You must take care of them with an 'XK' or 'XE' first.

PMATE can be used to modify an already existing file. Now you will need an 'input' file. Again use:

XFfile Edit 'file' (this time, assume 'file' already exists). 'file' is opened for input, and the text is read in.

You may now modify the text, and again finish up with an 'XE' or an 'XK'. An 'XK' will leave the original file intact - none of your changes will appear in it. An 'XE', however, will effectively update the input file to include the changes you have made. A copy of the input file before modification is retained under the same filename with the extension '.BAK' (any old backup of the same file is deleted).

If you call PMATE from CPM by typing

PMATE file

'file' is opened as the input or output file just as if you used an 'XF' command. If a file is large, it is often easier to break it into pieces. (This is necessary if the file is larger than available memory.) These pieces are called 'pages'. More details on this are given later:

GO TO IT

At this point, you know enough to utilize PMATE very effectively. Use the above commands, the instant commands (particularly the cursor motion keys), and Insert Mode and you'll quickly find yourself confidently entering and modifying programs or other text. It is important to get experience with these commands before attempting to learn the complete command set. Remember about reexecution of commands, and don't forget to try building some command strings.

Chapter III
MORE CONCEPTS

SIGNED NUMERIC ARGUMENTS

Up until now, we have assumed that all numeric arguments are positive integers. They can in fact be much more complex expressions. For now, we will just extend them to include negative numbers. What would '-3D' do, for instance? Rather than start at the cursor, and delete characters forward from there, commands with a negative argument work backwards through the text. '-3D' will delete the three characters just preceding the cursor (leaving the cursor pointing at the same character it used to be). Similarly, '-2K' will delete 2 lines preceding the cursor. '-S' will search backwards through the text, from the cursor, until it finds the string which follows the 'S'.

LINE ORIENTED COMMANDS

A number of commands, such as 'K', are 'Line-Oriented'. They all behave similarly to 'L', the command to move the cursor a specified number of lines. What happens when 'L' has a numeric argument that is less than or equal to 0? The easiest way to see is by trying it, but here is an example. Suppose the text buffer contains:

```
This is line a
This is line b
Guess which line this is?
This is line d
```

The command 'L', or '1L' would leave the cursor:

```
This is line a
This is line b
Guess which line this is?
This is line d
```

The command '0L' would have left the cursor:

```
This is line a
This is line b
Guess which line this is?
This is line d
```

The command '-2L' would have left the cursor:

```
This is line a
This is line b
Guess which line this is?
This is line d
```

Other line-oriented commands with numeric argument 'n' affect the text between the cursor, and the place the cursor would be placed if an 'nL' were executed. So the command '-K' or '-1K' would have left:

This is line a
which line this is?
This is line d

TEXT FORMATTING

Editing textual material presents very different problems than editing programs. Suppose you want to use PMATE to write a user's manual for a text editor program you have written. You write the following paragraph:

> Editing textual material presents very different problems
> than editing programs. You write the following paragraph:

Soon you decide that this doesn't make sense. A sentence is missing. You need to be able to add this sentence, and still keep the right number of words on a line. As you add words between 'programs.' and 'You', first 'paragraph', and then 'following', and so on need to wrap around to the next line. PMATE automatically takes care of this line formatting when in 'FORMAT MODE'.

To enter FORMAT MODE, use the command 'F'. Repeating the command 'F' will restore PMATE to normal. In FORMAT MODE, lines end not only on a carriage return, but also on the last possible 'space' which would keep the line from exceeding the allowed length. Words are never broken up. So the rule is to enter text without any carriage returns. PMATE will take care of the line length for you. Always use a carriage return at the end of a paragraph, or any other place where it is necessary to always begin a new line. Remember - if you put a CR at the end of a line because it looks like you are about to run off the screen, and then later you delete a few words from that line, the carriage return is still there, now stuck in the middle of the line. Moral - let PMATE divide your lines.

PMATE always keeps the screen up to date and properly formatted. You may actually find this annoying while entering text in the middle of a paragraph -for as you type, the margination can change with most every keystroke, producing a display which jumps around quite a bit. If this bothers you, a control-L instant command will insert a CR, effectively stabilizing things by putting you at the end of a paragraph. As soon as you are finished with the addition, type control-D to delete that excess CR.

You can have fun by changing the maximum number of characters allowed in a line. Use the 'F' command with a numeric argument. '30F' enters FORMAT MODE and sets the maximum line length to 30. Initially, the maximum line length is set to the number of characters in a line of the display. One reason you may wish to change it is to accomodate a printer.

It is very useful to be able to indent sections of text. One approach might be to precede each line with one or more tab characters. The problem with this is that the tab character is now fixed between two specific words. As words are deleted or inserted, these words slide around to different locations on the screen, playing havoc with your margins. For this reason, when in FORMAT MODE, PMATE is able to interpret the tab as a margin indent character. If an indent has been set to the same column as the tab stop, preceding an indented section with a tab will cause each succeeding line to indent to the same point, until a CR character is reached. An indent can be set using a 'YI' command. For example, '8YI' sets an indent at column 8, the first tab stop. So the tab (to column 8) following 'nYI' below causes the rest of the paragraph to be indented. This feature is used

throughout this manual

nYI Set an indent at column 'n'. Any tab to column 'n' will result in causing the remainder of the paragraph to be indented, until a carriage return is reached.

When operating in this manner, be sure not to use a tab to indent the first word beginning a paragraph, for it will indent the whole paragraph instead -type in the five spaces.

For some applications, you may wish to change the left and right margins for only a particular section of text. For instance, you may wish to move the left margin over 40 spaces to accomodate a picture. PMATE allows margin and tab information to be entered in a special non-printing control line. This control line begins with a ©F (F for Format), and ends in a CR. The complete set of possible entries is given later, but for example:

©FL20;R60

will change the left margin to 20 and the right to 60 from that point in the text onwards. These margins will be reflected in the text display. You might find it hard to enter that ©F in text, as this is defined as an instant command. See the section below on how to enter an arbitrary control character into the text. After altering a format line, a '©F' instant command tells PMATE to recompute its formatting, and bring everything up to date.

When in FORMAT MODE, it is very important to be able to easily see which lines end in carriage returns. In this mode, the carriage return character is actually displayed on the screen. The actual 'character' displayed depends upon the implementation, with '<' being a typical choice.

One final thought: FORMAT MODE can be very useful when writing programs too. If the language you are using supports a start and stop comment command, so that comments don't automatically end with a line (as does the PSA Macro Assembler, the language 'C', PASCAL and others), using PMATE in FORMAT MODE allows your programs to read like a book, with extensive, easily modified, comments. Of course, program lines must all be terminated with CR's, but comments can wrap around as much as desired.

UPPER AND LOWER CASE

If your keyboard is upper case only, it is still possible to generate upper and lower case characters. One character is chosen to be the 'case shift' character (usually a '/', but you can change this with a 'Q' command). When that character is typed, it is ignored. The next character to be entered is shifted in case. If it is necessary to enter the shift character itself, just type it twice. The whole keyboard can be toggled back and forth between upper and lower case by the '©S' (shift) instant command. After striking '©S' once, all following characters will be entered as lower case, unless they are shifted up by the shift character. '©S' again returns things so that characters are entered as upper case unless shifted. Some keyboards have both upper and lower case, but lower case can only be obtained by using the shift key. '©S' can also be used to make this keyboard look like a normal typewriter. On some keyboards, the shift-lock key also shifts the numeric keys

(like a typewriter). This can be extremely inconvenient if it is necessary to enter upper case only programs. In this case too, '©S' can be used instead of shift-lock to shift only the alphabetic keys.

CONTROL CHARACTERS

Since control characters are used as instant commands, it might seem difficult to actually enter a control character into the text. You can do this using a 'control shift' character. This character is usually '^', but it too can be changed using a 'Q' command. When you strike this shift character, nothing happens. The next character to be entered is shifted to the equivalent control character. So to enter a control-F, strike first '^', and then 'F'.

SOME OF ALL YOU EVER WANTED TO KNOW ABOUT INPUT FILES, OUTPUT FILES, AND PAGES

PMATE (as well as most any other text editor) needs the answer to two questions before it can do any editing. 'Where do I find the stuff to edit ??', and 'Where should I put it when I'm done ??' Running under CPM, these questions are answered when you first call the editor.

PMATE GARBAGIN GARBAGOU

is the CPM command to start editing the file GARBAGIN. It is opened as the 'input file'. Changes and additions are made, and the result is left in the file GARBAGOU - the 'output file'

Often the input file is too big to be handled in one big chunk --it might not even all fit into memory at one time. So, we break it into 'pages' -managable pieces which we can read one or two at a time from the output file, and write a few at a time to the output file. Don't worry for now about how big a page is. Just worry about how to turn them. You've already seen 'XA'. This is just the command to read in the next page, appending it to the text buffer. 'XA' can even take a numeric argument - the number of pages to read in. '5XA' will append 5 pages. 'nXW' is the command that writes out 'n' pages from the beginning of the text to the output file. All the text that is written out is also removed from the buffer - to modify it further, you must make a complete 'pass', and start editing again. 'nXR' is a very useful command. It is equivalent to 'nXAnXR'. '2XR' will 'replace' 2 pages in the text buffer by writing two pages from the beginning of the buffer to the output file, and then reading in two more from the input file. The 'all done' command, 'XE' first writes the text buffer to the output file, then reads in the rest of the input file, and writes it to the output file.

The size of a page is a fixed number of lines. This number can be set to 'n' by the 'nQP' command. '75QP' will set the page size to 75 lines, so that the command '3XA' will append 225 lines, and 'XW' will write 75 lines. Pages can be ended prematurely by a form feed character (©L). If the page size is set to 0 ('0QP'), form feeds are the only method of separating pages.

As you are entering text, if you find "memory space exhausted", do an XW to write out some of the text at the beginning of the buffer. XA will bring in more text from the disk to edit. If you need to start a new pass, 'XJ' writes all text out to the output file, and then reopens that for input, and you are ready to start editing at the beginning of the file.

Those two numbers separated by a comma in the status line at the top of the screen are just the input page number and the output page number. You should also see the name of your input file and your output file sitting up in that same line. At the beginning of the line, the letter followed by a colon is just the currently logged in disk drive.

XE does not return you to CPM - you are still in PMATE. You may return to CPM with an 'XH' command, or open some new input and output files using the 'XF' command. Just follow 'XF' with the same filenames you would follow PMATE with in the original command line, then hit two escapes.
'XFGARBAGIN GARBAGOUSS' will open the same files as above.

Often, an editing operation is performed to update a file. When you are done, you really want the new output file to have the same name the old input one did. One possibility would be to delete the old input file when you are done, and then rename the output file to the same name the old input one had. PMATE will do this for you automatically if you only specify one filename in the command line, or in an 'XF' command. This file is opened as the input file, and an output file is opened with the same name, but an extension 'SSS'. 'XE' will then output everything to the output file as usual. The old input file will be renamed to have the extension 'BAK' (a backup - any old backup is deleted), and the output file will then be renamed to the original input. For example:

PMATE JUNK.ASM

will open 'JUNK.ASM' as the input file, and 'JUNK.SSS' as the output file. 'XE' will then rename 'JUNK.ASM' to 'JUNK.BAK', and then rename 'JUNK.SSS' to 'JUNK.ASM'.

In the PMATE command line, or in an XF command, either the input file or the output file can be preceded by a drive specifier ('A:','B:','C:', etc.) to indicate which disk to find the file on. If there is no specifier, the currently logged-on disk is always referred to.

If the logged-in disk is 'A', 'PMATE B:GARBAGIN GARBAGOU' will look for input file 'GARBAGIN' on disk 'B', and output file 'GARBAGOU' on disk 'A'.

DIRECTORY MAINTENANCE

PMATE allows you to perform CPM directory lists and file deletes. One of many occasions this comes in handy is if you get a "disk full" message upon trying to write a file to disk. You can then list your directory, delete unwanted files, and again attempt to write the file you are now working on out to disk.

The command to list the entire current file directory is XL. The directory is actually entered in the text buffer, at the cursor location. This can be very useful, because now it is possible to edit this information just like any other text, and scroll through large listings. It can also be very inconvenient, because the directory may appear right in the middle of your working text. In this case, it can always be deleted. An alternative is to edit in another buffer (see below), or to use the '©_' instant command to edit the command string (then do the XL -- one further '©_' and the directory listing becomes the latest command, easily killed with a '©C').

Partial directory listings can be obtained by following 'XL' with a file name. As in the CCP 'DIR' command, the file name can have ?'s and *'s.

'XLJUNK' will insert 'JUNK' at the cursor if file 'JUNK' exists, otherwise it will do nothing.

'XL*.COM' will insert the names of all files with extension 'COM' at the cursor.

Files can be deleted with the 'XX' command. 'XXfile' deletes 'file' from the disk. The file name cannot contain the ambiguous characters '?' or '*'. DO NOT DELETE THE CURRENTLY DEFINED INPUT OR OUTPUT FILES.

It is possible to switch the currently logged in disk drive. This is done with the XS command. 'XSA' selects drive 'A', and 'XSC' selects drive 'C'.

ITERATION

It is often very useful to be able to repeat a command or a command string a whole lot of times. The iteration brackets ('[' and ']') allow us to easily do that. This command string

```
5[Igood morning!
$]
```

produces this text display:

```
good morning!
good morning!
good morning!
good morning!
good morning!
```

'3[K]' will produce the same result as '3K'. Beware! What will 3[Ihello] do?

Iteration brackets can be 'nested'. Make sure you have the same number of left and right iteration brackets. The command

```
100[40[I*$]I
$]
```

will fill up your text buffer with 100 lines of 40 stars each.

If there is no numeric argument in front of the iteration brackets, the operation will be repeated forever (that is, about 65,000 times) or until some sort of error occurs. '[I*$]' will fill up all available memory with stars, and then complain that it has no more memory left. '[K]' will start killing off lines, and continue until it has none left to kill. '[CgoodSbadS]' will change all occurences of 'good' (after the cursor) to 'bad'.

OTHER BUFFERS

PMATE actually has 11 different buffers into which text can be entered (as well as 2 buffers for command strings). These buffers are not of fixed size, any of them can expand to grab all the remaining available space. If you delete text from one buffer, this space is now available to any of the others.

Usually, you will be editing in the 'T' buffer. ('T' stands for 'text'). The 10 other buffers are labeled 0-9. Actually, you already know about the '0' buffer - all the special buffer commands, like 'BC' and 'BG' copy to and from buffer 0. Those commands could have also been written 'B0C' and 'B0G' (for instance, 'B3C' will copy to buffer 3). To start editing a buffer other than the 'T' buffer, type 'BnE' (buffer n edit), where n is '0'-'9' or 'T'. 'B3E' gets you to buffer 3, and then 'BTE' gets you back again. The buffers other than the 'T' buffer are useful mainly for storing blocks of text that need moving around, and for storing whole command strings, or 'macros'.

MACROS

A macro is like a subroutine. If you have written a command string that performs a function you will use a number of times, you can put that command string in buffer 'n'. Any time you wish, you may execute it with the command '.n'. There are several possible methods to put this command string into a buffer. The most straight-forward is just to start editing in this buffer using the 'BnE' command, and then to go into Insert Mode and enter the command right into the buffer. (What would happen if you tried to enter a command string, complete with escapes, into the buffer by using an 'I' command?)

Just as subroutines can be nested, macros can in turn call other macros. Just as it is often necessary to pass arguments to a subroutine, macros too can require passed string arguments. You may find you wish to use some macros over and over again. These can easily be incorporated as a permanent part of PMATE. These 'permanent macros' are executed by the command '.x' where 'x' is any character except for the digits 0-9. ('.1' will execute buffer 1, not a permanent macro.) You will learn later how to pass arguments to macros and how to create your own permanent macros.

ERROR TRACEBACK

Sometimes, errors will occur while executing a macro. The usual error message will appear in the text area of the screen. Down in the command area, the macro string which caused the error will be displayed, with the cursor pointing to the command character just past the offending one. The status line up top tells which buffer (or which permanent macro) was being executed at the time of the error. Now you have a choice of either hitting a CR or the space bar. A carriage return behaves as usual -- you're all ready to enter the next string. Striking the space bar 'pops a level'. It allows you to view the command string which 'called' the troublesome macro. As long as this command string is itself a macro, you may continue hitting the space bar, and popping levels. Once this command string is just the original one entered into the command buffer, the space bar and CR keys have the same effect. If a macro is called from several places in a command string, this error traceback allows you to find out exactly where the trouble occured.

AUXILLIARY FILE I/O

At any time, PMATE allows you to output sections of your current edit buffer to the disk, or to input disk files into this buffer. This can occur while input and outpput files are defined, and will not upset them. 'XIfile' will input all of 'file' (if

there is enough room in memory), and place it just before the cursor. 'nXIfile' will read in 'n' pages from 'file'. More pages can subsequently be read in by 'nXI' (if no file name is specified, input continues from the last named auxilliary input file).

'nXOfile' outputs the next 'n' lines of text (after the cursor) to 'file'. If there is no numeric argument 'n', the entire text buffer is output.

The many uses of these commands include merging sections of files (even if larger than available memory), loading macros into buffers to be executed, and using the disk for scratch storage as you might the special buffers.

THE CLONING OF PMATE

You probably have noticed that PMATE has a number of 'parameters' which can easily be changed (usually with an appropriate 'Q' command). On rare ocassions (usually), your favorite parameters will differ from mine. If you like a page size of 100 lines, you can give the command '100QP' every time you begin editing, or you can create yourself a customized verion of PMATE. Here's how:

First execute PMATE with no input or output files. Now make any desired changes (use the appropriate commands, create some permanent macros, or even - carefully please - get in there with your system monitor, and really start hacking away). Now give the command 'XDfile' ('D' for Duplicate) -where 'file' is the name of your new customized version of PMATE (the .COM extension is added automatically). Name it PMATE1 or PMATE2 or anything you like (please -only nice names). Use 'XH' to return to the CCP, and then verify the new version. If you are happy with it, you can erase the original PMATE.COM, and rename your new one to that - or keep several versions around for different purposes.

GET SOME HARD COPY

PMATE has a facility to output text to a printer. The command 'XT' will output the entire current edit buffer to the listing device. If there is a numeric argument, 'nXT' will print 'n' lines of text, starting at the cursor. Use this feature in order to print out just the changes you have made to your long files. When you are feeling ambitious, you can write macros to output text in almost any format you would want. For instance, you might have page numbers, titles, and even an index added to the text output.

Chapter IV
COMPLETE COMMAND SET

NUMERIC ARGUMENTS AND VARIABLES

Numeric arguments are integers. Usually they are signed numbers between -32,768 and 32,767. Sometimes they are considered as unsigned numbers from 0 to 65,535. However, numeric arguments can be more than just decimal numbers. They can be complex expressions consisting of numbers, variables, arithmetic and logical operations, and parenthesis. Operations are performed from left to right. Any operator precedence must be determined by parenthesis. So 5+3*2 has the value 16 and 5+(3*2) has the value 11. There can be up to 15 levels of parenthesis in an expression.

Numbers in command strings are usually interpreted as decimal numbers (base 10). However, the base, or 'current input radix' can be changed (see the 'Q' commands). So '10D55' usually deletes 10 characters, but if the input radix is 8 (octal), it will delete only 8 characters.

There are very few times when a radix other than decimal is useful, but if the radix is greater than 10, several rules must be observed. For instance, in hex, PMATE must know if 'D' is the hex digit 'D' or if it is the command to delete a character. The rule followed is that any number must begin with a digit from 0-9; then each succeeding character is interpreted as a digit if that is at all possible. For example, if the input radix is hex 'DDK' is interpreted to mean to delete two characters and then kill a line. '0DDK' however, would kill 221 lines (the value of 'DD' in hex). If it is necessary to terminate a hex number, an escape can be used. '0D5DK' will delete 13 characters and then kill a line. '2SD' will delete two characters, while 2D will be interpreted as 45 (decimal).

Numeric arguments can be displayed on the status line. Typing just a numeric argument, followed by 2 escapes will display the value of that argument in the current output radix (decimal by default) after the words "ARG=". In this way, the editor can be used to do integer arithmetic. By making the output radix different than the input radix, number conversions (such as hex to decimal) can be performed.

Arithmetic operations:

The following are valid arithmetic operations within a numeric argument.

+	Addition.
-	Subtraction or negation. -(3+4) is a valid expression.
*	Multiplication.
/	Division. Integer division, leaving just the quotient. The remainder of the last division performed is available as '@R' (see the '@' numeric arguments below).

Logical operations:

Logical operations leave the value -1 if true, and 0 if false. The following are valid logical operations within a numeric argument. In the expression '3=5', '3' will be referred to as the first operand, and '5' will be referred to as the second operand.

=	Equal--true if the first and second operand are equal.
<	Less than--true if the first operand is less than the second.
>	Greater than--true if the first operand is greater than the second.

&	And--true if both operands are true.
!	Or--true if either operand is true.
'	Logical complement

Examples:

3<2	has the value 0.
3<2'	has the value -1
2<3	has the value -1
2<3!(5=2)	has the value -1
2<3&(5=2)	has the value 0
5+3=(1+7)	has the value -1
5+3=(1+7)'	has the value 0

Variables and the Number Stack:

There are ten numeric variables (labeled 0-9) available for use. These variables can be set using the 'V' command (see below). They can be used as part of a numeric argument using an '@' argument. In addition, there is a Number Stack available. Any numeric argument can be 'pushed' on this stack (see ',' below), and 'popped off' later (see '@S'). The stack will hold up to 20 entries during the execution of a command, but is cleared upon completion.

Some of the variables used internally in the editor are also available for use in numeric arguments. The complete list of '@' arguments follows.

@i	The value of variable 'i', where 'i' is a digit from 0-9.
@A	The numeric argument preceding the last macro call.
@C	The current character number. This is the number of characters from the beginning of the text buffer to the character at the cursor. when the cursor is at the beginning of the buffer, this has the value 0.
@E	The value of the error flag.
@FfileS	Returns -1 if 'file' exists on the current directory, 0 if it doesn't.
@G	The length of the string argument just referenced (by an 'I','S', or 'C' command).
@HstrngS	Compares 'strng' to the characters at the cursor in the current text buffer. Returns 0 if they match, otherwise 1 or -1, depending upon which is 'greater'. Wildcards (as in 'S' command) are acceptable in the command string.
@I	The number of the current input page.
@K	The ASCII value of the key struck after an 'A' command.
@L	The current line number. if the cursor is within the first line, this is 0.
@M	The amount (in bytes) of working memory space remaining.
@O	The number of the current output page.
@P	The absolute memory address to which the cursor is pointing.
@R	The remainder of the last division performed.
@S	The value of the top of the number stack. The number stack is popped.
@T	The ASCII value of the character pointed to by the cursor.
@V	The current mode -- 0 for Command, 1 for Insert, 2 for Overtype.
@W	The current right hand margin.
@X	The current column that the cursor is in.
@Y	The current left hand margin.
@Z	The column of the next tab stop.
@@	The value of the byte in memory pointed to by variable 9.
'x	The ASCII value of the character x, where x is any character.

Block operations:

Commands which take a numeric argument to indicate the number of characters or lines can also be used to act upon a defined 'block'.

T Tag the current cursor position as beginning of block. (Equivalent to '©T' instant command.)

= Move the cursor to the tagged position, and use the difference between the old cursor and the tagged position as the numeric argument.

If you wish to type out a large block of text, move the cursor to the beginning of the block, use 'T5S' or '©T' to tag that position, then move the cursor to the end of the block and print out your text with '#XT'. '#' is also very useful with delete commands, and with buffer commands.

VARIABLE AND NUMBER STACK COMMANDS

nVi Set variable i (i is a digit from 0-9) to the value of numeric argument n. So '@C+3V2' sets variable 2 to 3 more than the current character position.

nVAi Add the value of numeric argument n to variable i. If n is missing it has the default value of 1, so variable i is incremented. '3VA5' adds 3 to variable 5.

n, Push numeric argument n on the number stack.

THE ERROR FLAG

Certain commands can produce 'non-fatal' error conditions. For example, if the cursor is already at the end of the text buffer, an 'M' command cannot move the cursor any further. The command string execution will not be interrupted to give an error message. However, it is possible to determine that an error condition existed by looking at the 'error flag'. This error flag is set to -1 to indicate an error condition following certain commands (these commands will be specified later). In addition, it is possible to suppress some 'fatal errors', such as would occur if a string cannot be found during a search command. If these error messages are suppressed, the error flag will indicate whether an error has occurred.

@E Gets the value of the error flag. The error flag is reset before executing a command string, and every time it is tested by '@E'. It is also reset when beginning an iteration.

E Set the error suppress flag. This flag is reset before executing a command string, and by every command which might test it.

MODE AND FORMAT COMMANDS

nN Change modes. If n=0, remain in COMMAND MODE. If n=2, go into OVERTYPE MODE. For any other 'n', or 'n' missing, go into INSERT MODE. This is similar to the instant commands '©N' or '©V', except that when '©X' returns to command mode, execution of the original command can continue.

PMATE has an automatic 'word-wrap' feature, active when in 'FORMAT MODE'. line will then end on the last complete word which fits within the allowed line

length. A carriage return is entered only to indicate that the next word must begin on a new line (end of paragraph).

nF Enter FORMAT MODE. The line length is set to 'n'

F Toggle in and out of FORMAT MODE. Go into FORMAT MODE if you're not there already, and leave it if you are.

CURSOR MOTION COMMANDS

The following commands move the cursor. While the cursor can also be moved using instant commands, the construction of powerful command strings requires cursor motion command characters.

+/-nM Move the cursor n characters. If n is positive, the cursor is moved forward. If n is negative, the cursor is moved backward. If n is 0, no action is taken.

+/-nL Move the cursor n lines. Consider the following example:

line a
line b
line c
line d
line e

Suppose the cursor is on the 'e' in 'line c'. '1L' or 'L' will move the cursor to the beginning of line d. '2L' will move it to the beginning of line e. '0L' moves to the beginning of the current line, line c. '-L' or '-1L' move the cursor to the beginning of line b, while '-2L' moves it to line a.

+/-nP Move the cursor n paragraphs. When not in FORMAT MODE, this behaves just like 'L'. When in FORMAT MODE, it seeks only the CR which forces the next word to begin on a new line.

+/-nW Move the cursor n words. Words are separated by any combination of any number of spaces, tabs, and carriage returns. 0W moves to the beginning of the current word. If n is negative, the cursor is moved to the beginning of the nth preceding word. If n is positive, the cursor is moved to the beginning of the nth following word.

If in the execution of an 'M','L','P', or 'W' command, the cursor would be moved past the end of the edit buffer, it is placed at the end, and the error flag is set. Similarly, if the cursor would be moved before the beginning of the edit buffer, it is placed at the beginning, and the error flag is set. The value of the error flag is obtainable by the numeric argument @E. It is -1 (true) when set, 0 (false) when clear.

A Move the cursor to the beginning of the current text buffer.
Z Move the cursor to the end of the current text buffer.

DELETION COMMANDS

+/-nD — Delete n characters starting at the cursor. If n is positive, characters are deleted beginning with the one pointed to by the cursor, and proceeding towards the end of the current text buffer. If n is 0, no action takes place. If n is negative, the first character to be deleted is the one just before the cursor. Characters aree then deleted proceeding towards the beginning of the text.

+/-nK — Kill n lines starting at the cursor. 'Lines' are defined as in the 'L' command. So 'K' deletes all characters starting at the cursor, up to and including the cr at the end of the line. '2K' will delete this and the next line too. '0K' deletes characters starting just before the cursor and proceeding back through the text until, but not including, the cr at the end of the preceding line. '-1K' deletes this much, and the line before also. '0KK', for example, will delete the line which contains the cursor, no matter where within the line the cursor is.

INSERTION COMMANDS

I — Insert the string which follows into the text immediately before the cursor. 'Istring$' would insert 'string'.

nI — If 'I' has a numeric argument, the character represented by that ASCII value is inserted into the text. If the input radix is decimal, '65I' will insert 'A'. Using this command, any character at all can be inserted into the text.

n\ — Insert the ASCII string representing the value of argument n in the current output radix. The string is inserted immediately before the cursor. If variable 0 has the value 23, '@0\I S@0+3\' will insert '23 26' into the text.

If any of these text insertions would fill up the available memory space (leaving 30 bytes free for more commands), the error message 'MEMORY SPACE EXHAUSTED' is given, and the text is not inserted.

R — Replace the text immediately following the cursor with the string which follows. No text is moved around. The new characters just overwrite what used to be there. If the cursor is near the end of the text buffer, and there are not enough characters to replace, an error message is given, and the substitution is not performed.

nR — When 'R' has a numeric argument, the character represented by that ASCII value replaces the character already at the cursor position.

STRING SEARCH COMMANDS

+nS Search forward, starting at the cursor, for an occurence of the string which follows. If n is present, search only through the next n lines (defined as in the 'L' command). If n is missing, continue the search until the end of the edit buffer is reached. The cursor is left positioned just after the located string.

-nS Search backward, starting just before the cursor, for an occurence of the string which follows. If n is present, search only through the preceding n lines (defined as in the 'L' command). If n is missing (i.e. '-Sstring') continue the search back to the beginning of the edit buffer. The cursor is left positioned on the first character of the located string.

If the string is not found, normally an error message is given. However, in some instances it is important to be able to continue execution of a command string after all occurences of the string have been found. No error message will be given, and command execution will continue, if the "Error Message Suppress Flag" is set. This flag is set by the 'E' command, and is reset upon the completion of every search. If this flag is set, and an error does occur, the "Error Flag" will be set. The value of the error flag is given by '@E'. It is -1 (true) when set, 0 (false) when clear.

Upper case characters in the search string will match only upper case characters in the text. Lower case characters will match either upper or lower case in text. (To match only lower case, see ©L wildcard below).

The following 'wildcards' can be used in the search string to match any of several specified characters.

©N Match anything but the character following. 'SMA©NTES' will find 'MALE' or 'MADE' but not 'PMATE'.

©E Match any character. 'MA©EE' will match 'MALE', 'MADE', and 'PMATE'.

©L Take next character literally. This allows an actual wildcard character to be searched for. 'SMA©L©EE' matches neither 'MALE' nor 'MADE', but only 'MA©EE'.

©S Matches either a space or a tab.

©W Matches any word terminator (any character other than a letter or a number).

STRING CHANGE COMMANDS

nC Search forward or backward for the string which follows as in 'nS' (wildcards are allowed). Change the located string to the second following string. 'Cstring1$string2$' locates the first occurence of 'string1' and replaces it by 'string2'. If the string cannot be located, errors are treated as for 'S'. In particular, error messages can be suppressed.

SETTING TAB STOPS

By default, tab stops are set every 8 spaces, but this assignment can easily be modified. A maximum of 10 tab stops can be defined.

YK — Kill all tab stops. A tab is now equivalent to a space.

nYS — Set a tab stop at column 'n'.

nYD — Delete the tab stop at column 'n' (if there is one).

nYE — Kill all old tab stops, and set new ones at every 'n'th column. '8YE' restores the conventional settings.

nYI — Set the default indent to column 'n'. If 'n' is 0, no indent is used. See the next section for use of indents.

For example, 'YK10YS30YS' would set up tab stops at columns 10 and 30. This setting might be useful for assembly language programming with labels in the first column, then instructions, and then comments. You could then save a version of PMATE permanently containing these tab settings (see 'XD' command).

The following commands make it easy to change tab settings without altering the current position of the text.

nYF — For the next 'n' lines, beginning at the cursor, replace all tabs with the appropriate number of spaces.

nYR — For the next 'n' lines, beginning at the cursor, replace blocks of spaces by tabs wherever possible.

IN-LINE TEXT FORMATTING

When in FORMAT MODE, it is possible to set tab stops and left and right margins in non-printing control lines embedded directly in the text. This is necessary when these parameters must change within the text. Even if not, it is still useful to put this format information on the first line of the text file so you do not need to remember which margins and tab stops you used the last time you edited this file. PMATE will recognize up to 30 embedded formats in memory at one time.

These control lines must begin with a '©F' and end in a carriage return. Any such line is not printed by the XT command so that any unprintable language can be entered here. Certain letters are recognized as 'commands', and must often be followed by a number. These commands can be strung together when separated by a semicolon.

Ln — Set the left margin to column 'n'.

Rn — Set the right margin to column 'n'.

K — Kill all tab stops.

:n — Set a tab stop at column 'n'.

Dn Delete the tab stop at column 'n'.

En Kill old tab stops, and set new ones at every 'n'th column.

In Set an indent to column 'n'. If 'n' is also a tab stop, tabbing to this column will cause all subsequent text to indent to this column until a CR is reached. For instance, the '©FI8' line at the beginning of this document causes the tab after the 'In' to indent this entire paragraph

The line:

©FL5;R50;E10

sets the left margin to column 5, the right to column 50, and sets a tab stop at every 10th column.

Any margin or tab stop information not specified in the format line reverts to the default. That is 0 for the left margin; the right margin default is set by the 'F' command; and the tab stop defaults by the 'Y' commands.

FLOW CONTROL COMMANDS

Conditional branching and iteration within commands make possible the construction of command strings equivalent to small text editing programs.

Iteration is accomplished as follows:

n[. . . m]

'. . .' represents any command string. This command string will be executed n times. If n is missing, it will be iterated 64K times. If n is 0, the command string in brackets will be skipped over. If n is -1, the command string will be executed once. Thus, if iteration brackets are preceded by a logical expression, the enclosed command string will be executed once if the expression is true, and skipped over if the expression is false. m is an optional numeric argument. If it is present, iteration of the loop will end prematurely if m becomes non-zero (true). If m is missing, its value is that of the error flag. That is, the iteration of the loop will be terminated if the error flag has been set.

'5[D]' has the same effect as '5D'.

'5V0[D-VA0@0=0]' also has the same effect as '5D'. '5V0' initializes variable 0. Within the iteration brackets, -VA0 decrements variable 0. the iteration will continue until the final numeric argument is true, when variable 0 is 0.

'[ChelloSgoodbye5]' changes all occurences of 'hello' to 'goodbye'.

'[ChelloSgoodbye]' changes the first occurence of 'hello' to 'goodbye]' (remember, all string arguments must be terminated by an escape).

Iterations can be nested to a maximum depth of 15.

I[...] Execute the expression in brackets if logical expression 'I'is true. Skip past matching bracket if it is false.

I[...][...]

Execute instructions within first set of brackets if logical expression 'I' is true; otherwise execute instructions within second set.

Further control of these iteration and if-then loops is offered by the 'next' and 'break' commands which are only meaningful within matching iteration brackets.

n^ Next -- if 'n' is non-zero (true) or missing, proceed to the next iteration

n_ Break -- if 'n' is non-zero (true) or missing, exit immediately from the enclosing iteration brackets.

As with other command characters, either upper or lower case brackets ({ } or [ ]) can be used for iteration. However, the above 'break' and 'next' commands do distinguish case. They skip right past '}' to the next ']'. Typically, put if-then-else constructions in upper case ({ }) so that any 'break' or 'next' command within will exit the desired iteration loop (not just the 'if' clause).

Conditional and unconditional branching within a command string is permitted. The proper point to branch to is designated by a label. A label is any character, preceded by a ':'. ':A' and ':#' are examples of valid labels. The branch command is:

nJ If n is missing or non-zero (true), transfer control to the command immediately following the referenced label. If n is 0, proceed with normal command execution. '@M>100JLS10K:L' will kill 10 lines if there aren't more than 100 bytes of memory left. 'JLS1000K:L' does nothing.

Be sure never to jump in or out of an iteration loop. this will lead to very erratic results.

Finally, it is possible to exit at any point from an entire macro.

n% Exit macro -- if 'n' is non-zero (true) or missing, exit from macro.

If you are a structured programming fanatic, you may find the 'J' command as useless as the 'GO TO'. All the control structure you will ever need can be found in the iteration brackets. You've got the 'IF-ELSE-THEN', and the 'DO-UNTIL'. Have fun!!

BUFFER COMMANDS

The editor actually contains 11 buffers into which text can be entered. The buffer which is initially used is called the 'T' (for 'text') buffer. The other buffers are labelled 0-9. Independent text can be contained in each of these buffers. Text can also be transferred from one to the other. The buffer which is currently being edited is displayed in the status line when in command mode. The instant command '©_' causes the command buffer to become the current edit buffer, and 'C' is displayed in the status line.

All buffer areas, including the command buffer, expand and contract dynamically. Any buffer grabs as much memory as it needs, until the total available memory is used up.

In the following buffer commands, 'b' refers to a buffer number, either 0-9, or 'T'. In all cases, b can be left out, and buffer 0 will be referenced. Some commands have a numeric argument, 'n', which refers to the number of lines to be moved or copied. 'n' can be positive or negative, and the effected lines are determined as in the 'L' and 'K' commands.

BbK — Kill buffer b. All the text in buffer b is deleted, and any space it took up is reclaimed.

BbE — Buffer b becomes the current edit buffer. Buffer 'T' is the initial edit buffer. When the edit buffer is changed, the cursor location of the old edit buffer is preserved. When the old edit buffer is reinstated, the cursor is restored.

nBbC — n lines from the edit buffer are copied to buffer b. The old contents of buffer b are destroyed. The cursor in buffer b is placed at the end of the entered lines. The copied lines in the edit buffer are preserved, and the cursor is placed after them.

nBbD — n lines from the edit buffer are inserted into buffer b (just before the cursor). The copied lines in the edit buffer are preserved, and the cursor is placed after them.

nBbM — n lines from the edit buffer are moved to buffer b. The old contents of buffer b are destroyed. The cursor in buffer b is placed at the end of the entered lines. The copied lines in the edit buffer are deleted.

nBbN — n lines from the edit buffer are inserted into buffer b (just before the cursor). The copied lines in the edit buffer are deleted.

BbG — Get the contents of buffer b. This is inserted just before the cursor. The contents of buffer b are not effected.

One common application of these buffer commands is to move or copy blocks of text. For example, 'BM' would move one line of text to buffer 0, after deleting any old text there. 'BN' could then be repetitively executed (keep hitting escape), each time moving the next line of text to the end of buffer 0. A whole block of text can in this manner be assembled in buffer 0. This is just an alternative to counting lines and typing '15BMSS'. The cursor in the edit buffer can then be moved somewhere else, and 'BG' will get back that block of text to this new position.

EXECUTING MACROS

The contents of any buffer can be executed as if it were a command.

.b Execute buffer b. Note: there is no default option, b must be present.

An executed buffer can in turn execute another buffer. This can be done to a level of 15 deep. There are two methods that can be used to easily insert a command string into a buffer for execution as a macro. The most straight-forward is to change the edit buffer to the one which is to hold the command string. Then the command can easily be entered and edited in insert mode. In command mode, it is difficult to enter an escape into the text area. Then change the edit buffer back to the original. An alternative method is to just type the command string, as if it were to be executed now. When it is done, the instant command '©_' is used to edit the command string. 'BbM' can then be used to move the macro to buffer b where it can be executed by the command '.b'.

I% Return early from macro if 'I' is true (non-zero) or missing. This is like a subroutine 'RET' statement. It makes it easy to return when a specified condition is met.

STRING ARGUMENTS

Commands such as 'I','S', and 'C' take string arguments. String arguments usually follow the command directly, but there are methods to get the arguments from other places. The character that signals to the editor that this is not an ordinary string argument is a '©A'. One place that string arguments can be taken from is the contents of a buffer.

©A@b Get string argument from buffer b.

For example, suppose buffer 2 contains 'trash'. then 'S©A@2S' will search through the text for 'trash'. 'I©A@0S' is equivalent to 'BG'.

When a buffer is executed as a macro, it is possible for the macro command to get string arguments from the command string which called it.

©Aa get string argument from calling command. 'a' is a letter from A-Z. 'A' refers to the first passed argument, 'B' the second, etc.

This should be clearer after an example. Suppose buffer 1 contains:

IDear Mr. SI©AASI,
You, Mr. SI©AASI have the opportunity to be the first on your block in beautiful SI©ABSI to own your own copy of an exciting new editor. Imagine what you and Mrs. SI©AASI can do with it. The rest of SI©ABSI will be so jealous. Blahhh, blahhh, blahhh$

then the command '.1Jones$Cambridge$S' would enter the following in into the text:

Dear Mr. Jones,
You, Mr. Jones have the opportunity to be the first on your block in beautiful Cambridge to own your own copy of an exciting new editor. Imagine what you and

Mrs. Jones can do with it. The rest of Cambridge will be so jealous. Blahhh, blahhh, blahhh

Unfortunately, that is not all this command will do. After '.1' is executed, the editor will come back and execute the command 'J'. When it goes off to execute buffer 1, PMATE has no idea how many string arguments will be required, and so it doesn't know where in the command string to return to execute the next command. It is necessary for buffer 1 to tell it where. The number of passed string arguments must be set in the macro by the 'QA' command (see 'Q' commands).

If buffer 1 contains '1QAI@AAS' , the command '.1' would have the same effect as the command 'I'. Similarly, if buffer 1 contained '2QAC@AAS@ABS', '.1' would have the same effect as 'C'.

When macros are nested several levels deep, the string arguments can also be nested.

COMMAND STRING FORMATTING

Since command strings are in fact text editing programs, facilities have been added for formatting these command strings for easy reading and modification. Spaces, tabs, and carriage returns are all ignored as commands.

Space
Tab
CR

Spaces, tabs, and carriage returns (as commands) are all ignored. Thus they can be placed between commands to enhance readability.

; A semicolon indicates that what follows is a comment. All characters through the next CR are ignored.

A command string can then be written to look like a well commented program. For example, here's a short command string that will change all upper case alphabetic characters to lower case, leaving everything else alone.

```
A           ;START AT BEGINNING OF EDIT BUFFER
[           ;BEGIN ITERATION
@T<'A JA    ;IF THE CURRENT TEXT CHARACTER IS NOT AN
            ;ALPHABETIC CHARACTER (IF IT'S ASCII VALUE
            ;IS LESS THAN THAT OF 'A'), JUMP TO LABEL 'A'

@T!' V0     ;CHANGE CHARACTER TO UPPER CASE BY 'OR'ING
            ;IT WITH ASCII VALUE OF SPACE (20H).
            ;SAVE RESULT IN VARIABLE 0.
D @0I       ;DELETE OLD CHARACTER AND INSERT SHIFTED ONE.
-M          ;MOVE BACK TO SAME CHAR

:A          M           ;MOVE CURSOR TO NEXT CHARACTER, SETTING ERROR
            ;FLAG IF IT IS AT THE END
            ]           ;CONTINUE WITH NEXT CHARACTER, UNLESS ERROR
```

```
;FLAG HAD BEEN SET
```

Of course, the whole command could also have been written as:

```
A[@T<'A]A@T!' V0D@01-M:AM]
```

And here's a much better way to do the same thing:

```
A[@T<'A[M][@T!' R]@T=0]
```

PERMANENT MACROS

You will find that you will write some macros that you will wish to use over and over again. These can be made permanent. Permanent macros are given a label that can be any character other than a digit.

.a Execute permanent macro 'a', where 'a' is any character other than 0-9.

To add or remove a permanent macro, it is necessary to edit the 'permanent macro area'. This area can be copied to or from the text buffer by the 'QMG' and the 'QMC' commands (see 'Q' commands). This area must begin and end with a '©X'. The '©X' is also used to separate different macros within the area. Immediately following each '©X' is the character which labels the macro, followed by the macro itself. Here is a macro area containing macros '#' and 'C':

```
©X= Iyou have just executed macro #S
©XC 2QAEC©AAS©ABS
©X
```

Executing the command '.#' will then insert 'you have just executed macro #' into the text. The command '.C' will behave just like 'C', except it will not generate an error message if the string is not found. You can think of this permanent macro facility as an ability to add your own commands to PMATE's command set. A new version of PMATE can now be generated incorporating these new commands (see the 'XD' command).

It is possible to define a macro which PMATE executes initially every time it is entered. The first macro in the permanent macro area will be executed as part of PMATE's initialization procedure if it is preceded by a ©I (tab), rather than the usual ©X. This macro can even end in 'XH', generating a program that acts on a file and returns, never displaying anything on the screen.

BREAKPOINTS

To aid in debugging complex commands and macros, PMATE includes a 'breakpoint' and 'trace' facility.

? Cease executing the command. PMATE is now in TRACE MODE. The cursor in the command area points to the next command to be executed. The current value of the numeric argument is displayed in the status line. Instant commands are active, and you can go into, and out of INSERT MODE. If you strike the escape key, command execution will resume as normal, until the next '?' command. However,

if you strike any other key that is not an instant command, PMATE will execute just the next command, and remain in TRACE MODE.

If you cannot figure out why your macro isn't behaving, insert several ?'s into the macro at strategic locations, and use them to examine what has happened after partial execution of the command.

KEYBOARD INPUT

G — Get a key from the keyboard as follows. Pause during the execution of the command and update the display. The string argument following 'G' is displayed as a prompt in the command display area.. Instant commands are active. Execution of the command is continued as soon as any character (other than an instant command) is entered from the keyboard. The ASCII value of this key is available by using '@K' in a numeric argument.

This command gives PMATE I/O POWER. PMATE can stop in the middle of an editing operation, and ask you how to proceed from there. Macros can be written to expand upon the power of the 'G' command - accepting either character strings (putting them in an available text buffer), or numbers (putting them in variables).

MISCELLANEOUS COMMANDS

nQA — Set the number of passed string arguments to n. See macro description.

QB — Ring bell. This is useful for indicating to the operator that a long command string has finished executing. It is also useful for playing annoying rhythms.

nQC — Set the control shift character to the character represented by the ASCII value n. This shift character will itself be ignored when input, but will enter the next character as a control character. This is useful if you wish to enter a character into the text which would otherwise be interpreted as an instant command.

nQD — Delay for a time proportional to n. This can be used in conjunction with 'L' to implement variable speed scrolling. It can also be used to arrange impressive demonstrations, whereby PMATE appears to have a mind of its own, displaying various messages.

nQE — Set type-out mode to 'n'. (See 'XT'.)

nQF — Set the 'form feed' character to that represented by the ASCII value n. This is the character that separates pages on the disk files.

nQH — Insert 'n' spaces at the cursor. This is useful for operations such as centering. Since all spaces are inserted at once, this operation is much faster than 'n[1 S]'.

nQI Set the input radix to numeric argument n. If n is missing, the radix is set to decimal. Remember, if the old input radix is octal, '10QI' will not set it to decimal, but rather, since the 10 is interpreted in the old radix, the input radix would remain octal.

nQK Set backup mode for files. If 'n' is 0, don't create a .BAK file from the old input file. If 'n' is non-zero or missing, create them.

nQL Set number of lines for instant commands ©U and ©J to scroll.

QMG Get the contents of the permanent macro area, and insert it into the current text buffer just before the cursor.

QMC Copy the entire current text buffer to the permanent macro area. The previous contents of the macro area are lost!! If you wish to save them, be sure to do a 'QMG' first, then add to or modify the text before copying it back.

nQO Set the output radix to n. If n is missing, the radix is set to decimal.

nQP Set page size to n. This is the number of lines appended or written as one page by the disk input and output routines. If n is 0, pages are delimited by form feed characters, instead of being a fixed number of lines.

nQR Redraw screen. The argument '@K' will now contain the value of any key struck, or 0 if none. Use this for creating interactive command strings where PMATE goes on doing something and showing you the results until you tell it to do something else.

nQS Set the upper case/lower case shift character to the character represented by the ASCII value n. This shift character will itself be ignored when input, but will shift the case of the next character entered. This is useful if you are using an upper case only keyboard.

nQT Type the character represented by the ASCII value n on the listing device.

nQX Set screen cursor to column 'n' on the same line it is now on. Depending on the state of the 'free-space' flag (see QY), the cursor may or may not be able to move past the last character in a line if the required column is off the end of the text.

nQY Set the 'free-space' flag if 'n' is 0 to allow the screen cursor to move into free space, past the end of a line. When a character is inserted at such a cursor position, the necessary amount of spaces is inserted into text to extend the line out to where the cursor appears. If 'n' is non-zero, reset the flag, so that the cursor is restricted to remain on actual text.

nQZ Don't allow cursor to move past column 'n'. Use this when you wish to restrict the width of entered text - usually to provide clean output on a limited width printer. When the cursor reaches the restricted column, it is inhibited from advancing, and the bell rings as a warning. If 'n' is

missing, the default width of 250 columns is restored.

nQ! Set byte in memory whose address is held in variable 9 to 'n'. This command allows PMATE to alter any byte in memory (and of course, crash the system). In conjunction with '@@', a monitor could be constructed in macros. Other macros might change I/O driver parameters. However, for altering text, just move the cursor there and use nR.

nQ- Sets flag to indicate whether numbers are displayed as signed or positive only. If 'n' is 0, display as positive only, otherwise display as signed number. This effects the argument display (ARG) in the status line, as well as numbers inserted in the text by the '\' command. If you type the command '@m$$' in order to discover how much memory remains, and you see 'ARG=-30536' in the display (which will happen if more than 32K of memory remains), you may wish to enter '0q-' to get a more meaningful display.

nQm Set user variable m to value n. m is a digit from 0 to 9. These 10 user variables are available for use by user written I/O drivers. For instance, you may wish to use one of these to control whether hard copy output goes to a TTY console, or to a line printer. Or, you may wish to be able to easily go between black on white, or white on black video.

INPUT, OUTPUT, AND DIRECTORY MAINTENANCE COMMANDS

All input and output commands begin with an 'X'. This should help prevent accidental I/O, which could cause great upheaval.

Disk I/O:

Files which are too large for memory must be broken up into 'pages'. Pages are divided by a user definable character (usually a form feed), or can be defined to be a fixed number of lines (see the 'QP' command).

nXA Append n pages from the input file to the current edit buffer.

nXW Write n pages from the current edit buffer to the output file, deleting them from the buffer.

nXR Replace n pages, appending n pages from the input file, and writing n pages to the output file.

nXY Yank n pages from the input file. Each page overwrites the old one, without writing it to the output file. BE CAREFUL - this command is only useful for reviewing an existing file, and except in special circumstances, the file should be 'XK'ed when done.

XFfile1

If 'file' already exists, open it as the input file, and open 'file.$$$' as the output file (in this case, if 'file.$$$' already exists, an error message will be given). If 'file' does not exist, create it and make it the output file (this is the way to create a new file). 'file' can be

preceded by a drive specifier ('A:', 'B:', 'C:', or 'D:')

XFfile1 file2

Open 'file1' as input and 'file2' as output. 'file1' should already exist on the disk (if not, it is opened as the output file), and 'file2' should not (if it does, an error message is given). Both 'file1' and 'file2' may be preceded by drive specifiers.

XE — End of editing pass. Write the current text buffer to the output file. Read in the remainder of the input file and write it to the output file. Close the input and output files and clear the text buffer. If the output file is the same as the input (with a $$$ extension), rename the input file to 'file.BAK', deleting any old backup, and rename the output file to have the same name as the old input file.

XEfile — End of editing pass, as above - but output file is renamed to 'file', and the original input file is left undisturbed.

XJ — Start a new editing pass. Equivalent to an XE and then an XF of the original file name. Useful for editing a page already written out with XW or XR. Even on files which fit entirely in memory, don't go too long without an XJ. This ensures that your editing work will be saved on the disk in case of power failure, or catastrophic error.

XJfile — Equivalent to 'XEfile', followed by reopening the new file.

XC — Close input and output files as they are. Neither the contents of the text buffer, nor the rest of the input file is written to the output file. Even if the output file is a temporary one (with extension '$$$') no file renaming takes place.

XK — Delete the output file and clear the text buffer.

XH — Reboot CPM, and return to its CCP (Console Command Processor). This is the usual way to exit from the editor.

XDfile — Duplicate PMATE. Write it as it now exists to 'file.COM'. This output file can later be renamed PMATE.COM

nXIfile — Auxilliary input. Read the first 'n' pages of 'file' into current edit buffer at cursor location, even if another file is 'open' as the input file. If 'n' is missing, read in the entire file. If the entire file is not read in, the remainder can be read in later:

nXI — Input the next 'n' pages from the input file last defined by the 'XIfile' command. If 'n' is missing, input the entire remainder of the file.

nXOfile

Create 'file' and write 'n' lines of text, beginning at the cursor, out to it. If n is missing, write out the entire current edit buffer.

Directory Maintenance:

XSd — Change the currently logged in disk to 'd' ('A', 'B', etc.) For instance,

XSB logs in drive 'B'. PMATE will not respond to this command while input and output files are defined. This command also resets the CPM disk system, and should be used when the current diskette is changed. (If you are on drive B and change the diskette, type 'XSBSS'.)

XLfile — Like CPM 'DIR' command. List all files which match 'file' (* and ? can be included in the file specification). If 'file' is missing, the entire directory is listed. The directory listing is inserted in the text buffer, at the cursor. This allows the directory to be printed, and otherwise manipulated like text. However, if desired text is already in the text buffer, it may be necessary to delete the directory text. Alternatively, change the current text buffer before giving the XL command.

XXfile — Delete 'file' from the disk. Ambiguous file names (containing * and ?) are not permitted.

Other:

XM — Call the system monitor. You can return without losing any text by either executing a RET instruction, or jumping to location 103H.

nXT — Type n lines, starting at the cursor, on the listing device. If n is missing, type out the entire current edit buffer. There are 3 type-out modes (set by the 'QE' command). Mode 1 (the default mode) is intended for printing programs or text on a regular printer. Tabs are expanded to spaces. Format lines are not printed, but affect the margins and tab stop settings. Other control characters are sent through to the printer. Mode 0 prints text almost exactly like it is displayed. Format lines are printed, escapes type out as '$', and other control characters are printed as an up-arrow followed by an upper case letter. This mode is useful for printing macros and for draft output. Mode 2 is intended for use with intelligent printers which do their own formatting. Carriage returns are only sent at the end of a paragraph, tabs are not expanded to spaces, and all control sequences are passed on to the printer.

Chapter V
MACRO EXAMPLES AND IDEAS

SOME DETAILED EXAMPLES

This chapter contains examples of macros, provided for use or study. These macros are not intended to be polished final products, but are illustrative, and are meant to provide you with a foundation on which to build, as well as stimulate your imagination.

The best way to understand how and why these macros work is to enter them, try them, and then run them in trace mode. You should read up on trace mode and breakpoints in chapter 4, but here's a summary. Put a question mark (?) at the beginning of the macro, or at the place where you cease to understand what's going on. At this point, the macro will 'single step', show you the results of its latest operation, and wait for you to strike a key to continue.

This section contains some relatively simple macros, explained in greater detail than later ones. Here's the first. Programmers often 'comment out' sections of code. This is a way of deleting them from the program, but preserving the code just in case. In many languages, this can be accomplished just by putting a semicolon at the beginning of each line. You could go into insert mode, enter ';', then move the cursor down, enter ';', move the cursor, and on and on. This isn't bad for a few lines, but for more, try the command 'I;SL$$'. This will insert the semicolon and move the cursor all at once. Keep striking escapes, and the command will be repeated, until you have reached your last line. Finally, try '20[I;SL]$$'. This command will repeat the above sequence 20 times, commenting out 20 lines at a time. Any time you need to perform a repetitive sequence, **think macro.**

Now that you can quickly create comments, the reverse problem might come to mind. Have you ever needed to delete all the comments from a file? If you've ever done that by hand, you will appreciate this macro which does it for you automatically. Use it on programs, or on PMATE macros themselves -generating a version that will better fit in available memory (of course, always keep a copy of the original). This macro assumes that comments begin with a semicolon, and it deletes the comment starting at the semicolon, as well as any preceding tabs.

```
[S;S -M -S©N©IS M K I
$]
```

The left bracket starts a loop - all comments will be deleted. Next, find a comment by searching for ';'. Now we need to find all tabs preceding the semicolon. Since the 'S' command left the cursor on the character just past the semicolon, we must move back one (-M) before looking for tabs. The next 'S' searches backwards until it finds anything other than a tab (the ©N©I matches anything except a control-I, which is a tab). The cursor will be left on that first character found which isn't a tab. Then after 'M', the cursor points to the entire comment which needs deleteing. 'K' deletes the entire comment, as well as the carriage return at the end of the line. The carriage return is then restored by the 'I', and the right bracket loops back to the start, looking for the next comment. The macro will terminate when the first 'S' command is unable to find any more comments, and so will produce an error message.

Escape characters in text present problems for macro strings which need to operate on them. For instance, if you wish to insert an escape into text, 'ISSS' will clearly not work, but '27I' will. However, you may feel search or change is hopeless. Well, here's a routine to change all escapes in text to dollar signs (in case you ever need to write a chapter like this one).

```
[@T=27[36R][M]@T=0]
```

'[' starts iteration, for we wish to do the entire text buffer. '@T=27' tests the character under the cursor to see if it's an escape (ASCII code 27). If it is, the expression in the first set of brackets, '36R' is executed. This just replaces the escape with a dollar sign (ASCII code 36). This could have also been expressed 'RSS', but it wouldn't be obvious to the reader that the first 'S' is a dollar sign, and the second 'S' is an escape.
Anyway, if the the character at the cursor is not an escape, the expression in the second set of brackets is executed - just move the cursor on to the next character. '@T=0' tests to see if the cursor has reached the end of the text buffer (always a null). If so, the iteration ends; if not, go back and check the next character.

The command [CblahSblewS] will change all occurences of 'blah' in the text buffer to 'blew'. An often requested editing feature is a 'conditional change' (A.K.A 'interactive search and replace'). This command would not change all occurences of 'blah', but would stop at each one and ask you whether or not you would like a replacement to be made. Put this command string in buffer 1, and type '.1blahSblewSS'.

```
2QA
[
        S@AAS
        GType escape to replaceS
        @K=27[-C@AAS@AB]
]
```

The first line sets the number of string arguments required from the calling command (in this case, 'blah' is the first, and 'blew' is the second). The next line searches for the first argument (blah). The 'G' command then gives a prompt, displays the text buffer with the cursor pointing past the next 'blah', and waits for you to respond. If you respond with an escape, '@K=27' is true, and the expression in brackets will be executed. This will change 'blah' to 'blew' (the '-C' is necessary because the cursor has already been moved past 'blah'). If any key other than an escape is hit, the expression in brackets is ignored. The last line iterates back to the first '[' - keep looking for the blahs. The process will continue until the last blah, or until you hit control-C. Remember, control-C will halt any runaway macro.

TEXT OUTPUT PROCESSING

PMATE does not internally perform many print functions often associated with word processors. PMATE can be used with a separate output processor, or macros can be written to do the job. Here are a few ideas to get you started.

This macro will center a line. Start with the cursor anywhere on the line to be centered.

```
L-M             ;move to end of current line
@W-@X/2V0       ;get one half the distance from right margin
                ;to current cursor position
                ;save it in variable 0.
0L              ;back to beginning of line
@0QH            ;insert number of spaces computed above
L               ;move on to next line
```

A macro to move the line flush with the right margin is also easy - just get rid of the '/2' after the '@W-@X'.

Try this next example. Whichever character you leave the cursor on will be replicated, leaving the rest of the line flush with the right margin. Use it, for example, on a table of contents. Start with

```
Chapter 1.pg 1
Chapter 2.pg 24
Chapter 3.pg 30
```

Put the cursor on each decimal point in turn, execute the macro three times, and you are left with

```
Chapter 1.....pg 1
Chapter 2.....pg 24
Chapter 3.....pg 30
```

```
@XV0            ;save the current column in variable 0
L-M             ;find end of line
@W-@XV1         ;amount of space needing fill to variable 1
@0QX            ;back to original cursor position
@TV2            ;save the character there in V2
@1QH            ;fill out line with spaces
@0QX            ;back to original cursor position again
@1[@2R]         ;now overtype the spaces
                ;with the original character
```

The last three lines could have been replaced with '@1[@2I]'. However, replaces require much less overhead than inserts, so that the suggested method will execute faster.

Now we can start to tackle page headings and numbering. Here is one simple-minded approach. Suppose buffer 1 contains a one line heading which you would like printed at the top of every page. Suppose further you have put a '#' in that line at the place where you wish a page number to be inserted. Buffer 1 might contain:

```
Chapter 2          EXCITING DOCUMENT!          page #
```

Enter into variable 0 the first page number: '5V0SS' would be appropriate here if

chapter 2 started on page 5. Then the following macro will print out your file, using the above header, and printing page numbers:

```
[                       ;start iteration -- will type till end of buffer
B2K                     ;empty buffer 2
B2E                     ;edit buffer 2
B1G                     ;get prototype page header from buffer 1
A                       ;find its beginning
S#5-D                   ;find '#' and delete it
@0\                     ;insert page number there instead
VA0                     ;increment page number - ready for next page
XT                      ;type header
10QT                    ;send a line-feed to skip line after header
BTE                     ;back to text buffer
60XT                    ;type next 60 lines of document
4[10QT]                 ;send 4 line-feeds to complete a 66 line page.
@T=0]                   ;keep typing until the text buffer is finished
```

- There are lots of ways to expand upon this. For documents larger than available memory, have the macro read in successive pages. Define a print format line, starting with a unique character (maybe '©P', or whatever pleases you). The print macro will not type this line, but use its information for further formatting. The print format can include output functions like double space, center (see macro above), etc. Header information no longer needs to to be put in a buffer beforehand, but can be moved there from the print format line as the macro proceeds.

FORMS AND MATH

The 'G' command gets a character from the keyboard. Often, you may need a whole string. The next macro gets a string from the keyboard, echoes what has been typed up in the command/prompt line, and saves that string in buffer 9. The string ends on a carriage return. In order to correct mistakes on entry, 'rubout' will delete the last character entered.

```
B9K                     ;delete old contents of buffer 9
[                       ;start iteration
G©A@9$                  ;get a character, displaying contents
                        ;of buffer 9 on command line
@K=13_                  ;if character is a CR, break (all done)
B9E                     ;now go into buffer 9
@K=127[-D][@KI]         ;if character is a rubout
                        ;delete previously entered character
                        ;otherwise, insert new character
BTE                     ;back to text buffer
]
```

One typical application of the above is in creating an interactive macro to fill out forms. For instance, a preexisting invoice 'skeleton' might be read in. The operator could then use the full capabilities of PMATE to fill in the blanks, or an 'invoice macro' could set the cursor into each field, and prompt for information. The entry would be accumulated in buffer 9, as above, and inserted in the text when all done. The invoice macro could check for illegal entries, and prevent the

user from totally destroying the invoice form. Furthermore, the operator need not know how to use PMATE.

Along with forms often goes the desire to add up some numbers. Some word processors supply 'Math Packs' of varying sophistication. Here's a macro to get you started in that direction. It adds the number pointed to by the cursor (the cursor can point anywhere within) to a number stored in buffer 9.

```
[ M (@T>'9) ! (@T<'0)]  ;Move cursor until end of number is found
0V1                     ;initialize carry
B9E                     ;number to add to is in buffer 9
Z                       ;move to end of that number
[                       ;iterate one digit at a time
                        ;starting with least signficant
BTE                     ;back to first number
-M                      ;get next most significant digit
(@T>'9) ! (@T<'0)       ;not a digit?
[M 0V0][                ;no, don't move past it
                        ;0 to V0 is number to be added
@T-'0V0]                ;a digit - gets its numeric value to V0.
B9E                     ;now go to buffer 9
-M                      ;get next most significant digit
@E_                     ;done if out of digits
@T+@0+@1V0              ;add digit from text, and carry to it
                        ;result to V0
@0>'9[1V1 @0-10 R       ;if greater than 9, set carry to 1, subtract 10
                        ;and store result in text
][0V1 @0R]              ;not greater than 9, set carry to 0
                        ;and store in text
-M                      ;R has moved cursor, so move back
]                       ;on to next digit
BTE
```

The number of digits stored in buffer 9 controls the precision of the result. For example, if you start with '000000000', numbers up to 999,999,999 can be accumulated. Automate by using further macros to call this one: add up rows, columns, or whatever your format requires. The result can be moved back into the main text buffer. How about subtraction, multiplication, or division?

MORE FUN WITH PRINTERS

This simple macro lets you type directly on your printer, using the keyboard, as if it were a typewriter.

```
[
GDIRECT TYPES
@K=13[13QT 10QT][@KQT]
]
```

The third line implements an auto-linefeed. If it finds a carriage return, it sends a

line feed also -- any other character is sent 'as is'.

Here's a macro which prints an alphabetized directory listing. It should suggest many other applications:

```
B1K                     ;clear buffer 1 to hold directory list
B1E                     ;go into buffer 1
XLS                     ;get a directory listing
A                       ;go to beginning of directory
[                       ;begin overall loop
BC                      ;copy first file name to buffer 0 -- will try to
                        ;find file names earlier alphabetically.
[                       ;this loop finds earliest file name
@H⍟A@05<0[BC][L]        ;compare next file name to earliest already
                        ;found - if this one is earlier, copy it to
                        ;buffer 0, otherwise, advance to next
@T=0]                   ;iterate until end of directory list
A                       ;back to top of directory list
S⍟A@05                  ;match the earliest entry stored in buffer 0
-1XT                    ;type it out
-K                      ;and then delete it
A@T=0]                  ;back to beginning - continue unless
                        ;list is now empty
BTE                     ;back to text buffer when all done
```

CURSOR MOTION

Presented here, without comment, are the macros used by PMATE to implement the 'mixed' cursor motion instant commands. If you wish to customize cursor motion to your own taste, this should give you a place to start.

up:

```
@V=2[@X,-L@SQX][-M0L]
```

down:

```
@V=2[@X,L@SQX][L]
```

left:

```
@V=2[@X>0[@X-1QX]][-M]
```

right:

```
@V=2[@X+1QX][M]
```

Chapter VI
CONFIGURATION GUIDE

GETTING STARTED

Your PMATE diskette arrives with the following programs on it.

PMATE.COM ← renamed to PMATEDIS.COM
CONPMATE.COM
IOPATCH.ASM
CONFIG.DOC

and a bunch of files with the extension CNF. These are ASCII files which contain configuration information for a number of popular video terminals and memory mapped displays. First type out the file CONFIG.DOC. It contains a list of all the .CNF files on the disk, and which terminal each supports. If you have a Lear Seigler ADM-3A, CONFIG.DOC tells you to use file ADM3A.CNF. Then type:

CONPMATE ADM3A.CNF

After disk activity has ceased, CONPMATE will think for about 15-30 seconds, create a customized version of PMATE in memory, and then PMATE will sign on. (In case of an error in the format of the .CNF file, CONPMATE will just return control back to CP/M.) It is now necessary to save this version on disk. Type

XDPMATE$$

(if you haven't already read through the rest of the User's Manual, '$' represents the ESCAPE key). This duplicates the current version of PMATE -giving it the name PMATE.COM. If PMATE.COM already exists on this disk, you better use PMATE1 (XDPMATE1$$) and rename it later.

GENERATING YOUR OWN CONFIGURATION FILE

If a configuration file for your terminal is not included among those provided on the disk, you will need to generate your own. Even if a file for your terminal does exist, you will probably later wish to modify it to tailor the interaction more to your tastes.

The CNF files contain a series of questions and answers (in ASCII). You will need to use your current text editor (such as 'ED') in order to create a custom configuration file. Later you will be able to use PMATE to make additional changes.

Configuration questions require either a yes/no answer, or a series of numbers. All answers follow 3 stars (***). Numbers may be in decimal or hex. Hex numbers are identified by ending in 'H'. If more than 1 number is required, separate them by spaces.

CONFIGURING A VIDEO TERMINAL

Are you using a memory mapped display?
  Answer no.

Number of lines

Enter the number of lines on the terminal display (usually 24)

Characters per line
Enter the number of characters on each line of the terminal display (usually 80)

Now you need to enter the codes required by your terminal to perform certain display functions. For each function, enter a sequence of from 1 to 9 bytes (remember, hex codes require an 'H' - decimal is assumed). In the next line, following 'Delay ***', enter any delay your terminal requires after performing the specified operation. This delay is given in milliseconds and assumes a 4MHZ machine. If your computer runs at a different speed, it is necessary to scale the delays accordingly. For instance, if a 20 msec delay is required, enter 10 to get the proper delay for your 2MHZ processor.

Clear screen code
Enter the sequence which clears the screen.

Clear to end of line code
If your terminal has this feature, enter the sequence which clears from the current cursor location to the end of the line. If your terminal does not have this feature, leave blank.

Line insert and delete can be used to support much faster scrolling of display text. If your terminal supports these features, fill in both:

Code to insert line
Enter codes to insert a line at the current cursor position (which is guaranteed to always be at the beginning of a line).

Code to delete line
Enter codes to delete the line the cursor is on (again, the cursor is guaranteed to be at a line beginning).

There are many methods in use of sending cursor addressing information to video terminals. Most all can be included in the following generalization.

1) send a bunch of codes
2) send either the x or the y coordinate
3) send a bunch more codes
4) send the coordinate not already sent
5) send a final bunch of codes.

Cursor addressing lead in
Enter the preliminary sequence. (This will always be present.)

Is Y coordinate given first?
Enter yes if row information is sent before column information, otherwise enter no.

Is cursor position information in ASCII?
Actual cursor coordinates can be sent as 1 byte (binary), or as an ASCII sequence, 'spelling out' the coordinate in decimal. Answer 'yes' for the latter.

X offset
Enter a number which is added to the desired column before it is shipped to the terminal. If a byte of 0 gets the left-most column, just enter an offset of 0.

Y offset
Enter the number which gets to the top-most row.

Enter any codes between X and Y
Enter the sequence described in 3) above. Often, this is blank.

Enter any terminating sequence
Enter the sequence described in 5) above. This too is often blank.

Delay
Enter any delay needed after the entire cursor addressing sequence.

Is it necessary to suppress sending a character to bottom right position of screen?
In many terminals, any time a character is written to the last column of the last row, the entire display will scroll up one line. Note that some terminals (such as the Lear Siegler ADM-3A) contain a switch to enable or disable this feature. But if your terminal persists in such behavior, enter 'yes'. PMATE will then refrain from entering any characters in this corner. Alternatively, decreasing the line size (usually from 80 to 79) will also prevent any characters from being written in the last column, and will keep all displayed lines the same length.

CONFIGURING A MEMORY MAPPED DISPLAY

Are you using a memory mapped display?
Answer yes.

Now skip past the TERMINAL CONFIGURATION section to the MEMORY MAPPED CONFIGURATION section, and answer:

Number of lines
Number of text lines on the screen.

Characters per line
Number of characters on each text line.

Location of video memory
Fill in the initial RAM address of the video memory. Remember, add an 'H' to a hex address.

Spacing between lines
The address difference between the beginning of adjacent lines. This is usually just the number of characters on each line - but makers of 80 character video boards often space lines by 128 characters in memory. (This makes for easier cursor calculations, but uses more addressing space).

Next, assembly language routines to perform certain simple functions are required. Since CONPMATE does not have an assembler built in, these routines must be hand assembled, and the resulting bytes entered. (Or, use the alternative configuration procedure given in the next chapter - modify IOPATCH.ASM, and use an assembler) Each of the following routines can be up to 21 bytes in length.

Routine to store reg A in memory pointed to by DE

This is usually just:

STAX D	12H
RET	C9H

If the byte to be displayed needs any processing (such as setting the high order bit), this is the place to do it.

Routine to display cursor at memory pointed to by HL

This routine displays a cursor at the specified position. If this requires destroying the character in the video memory at that position, then this character must be saved for later retrieval by the clear-cursor routine below. (Location 014EH may be used for this purpose, provided the maximum size of the cursor display routine is cut to 20 bytes). In many displays, setting the high order bit of a byte in video memory reverses the video at that point. This makes an effective cursor, and the cursor set routine is:

MOV A,M	7EH
ORI 80H	F6H 80H
MOV M,A	77H
RET	C9H

Routine to clear cursor from memory pointed to by HL

This routine must clear the cursor set above, restoring the video memory to its original state. To clear the cursor of the example above:

MOV A,M	7EH
ANI 7FH	E6H 7FH
MOV M,A	77H
RET	C9H

Cursor blink count

Enter a number which determines the cursor blink count. While waiting for keyboard input, PMATE delays an amount proportional to this number, and then sending alternately a set cursor sequence, then a clear cursor sequence. This will turn an inverse video cursor into a blinking cursor. If you do not wish your cursor to blink, or the hardware blinks the cursor already, enter a 0. Otherwise, enter a number to suit your taste. Here's a starting point -30 works well on a 4mhz processor (equivalent to 15 at 2mhz).

MORE CONFIGURATION INFO

The questions in the MISCELLANEOUS section deal mostly with PMATE display interaction parameters. Using serial terminals, even at 19,200 baud, presents the problem that the display cannot always keep up with your keystrokes. If you scroll 1 screenful, it can take 2 seconds (at 9600 baud) to redraw the screen. PMATE is not 'dead' during this time, but will halt redraw, and respond to further commands. PMATE can be customized to respond in different ways in this situation. Try various combinations of answers until you find the 'feel' that's best for you.

How many lines from the center of screen can cursor wander?

Since the display screen can only hold a small portion of the entire text file being edited, it is necessary to 'scroll' the display as the cursor moves off of it. Typically, the display scrolls to prevent the cursor from moving down past the bottom line, or up past the top. However, it is often better to keep 1 or 2 lines above or below the cursor at all times, so you can better see the context you are working in.

The number entered indicates how far from the center line of the text display the cursor will be allowed to move before a scroll occurs. If this number is 0, the cursor will remain on the middle line of the display --cursor motion up or down will cause a screen scroll. Using 0 (or a small number) keeps maximum context, requires the most screen scrolling, and is therefore not recommended for serial displays which do not support insert and delete line. For these terminals, or if your preference runs in that direction, use 1/2 the size of the display area, less 1 or 2 lines of 'context' on each side. For example, on a 24 line screen, 21 lines are dedicated to text display. Entering 10 (don't use anything bigger!!) will produce a display which scrolls only at either limit; 8 leaves 2 lines on top or bottom before scrolling; and 1 will restrict the cursor to the 3 center lines.

How many lines do you wish redrawn in foreground?

This determines the number of lines which will be redrawn on the screen before the next keystroke is responded to. In other words, this many lines are kept up to date at all times, the rest will be redrawn when PMATE has the time. The smaller this number, the faster PMATE's overall response is, but the less you can see what effect each of your keystrokes has had.

Should display proceed from top to bottom (or from cursor outward)

PMATE screen redraws can proceed in one of two ways. The traditional method is to start at the top, and work down. Alternatively, PMATE can start drawing on the line the cursor is on, and work outward, alternately displaying lines on either side. This also means that if the cursor is down on the bottom line, the display proceeds from bottom up, and if the cursor is at the top, the display proceeds in the usual top-down manner. This method has the advantage of first showing you text where you are most interested in it - near the cursor. When used with a cursor constrained to the center few lines of text (see 'cursor wander' question above), some people find the inside-out redraws annoying. Answer 'yes' to get a top-down display, and 'no' to get a display proceeding from the cursor outwards. 'Yes' is recommended for memory mapped displays, as the inside-out display takes slightly longer (on video terminals, the serial interface limits the display speed, so this makes no difference).

Should cursor be displayed before each line is redrawn?

By addressing the terminals cursor to its final position before each line is redrawn, you don't lose track of where PMATE's cursor is as the screen redraw proceeds. As usual, there is a trade-off to be made. Twice as many cursor addressing sequences now need to be performed. If your terminal requires a significant delay after each cursor addressing operation, this can slow down a screen redraw noticably. As usual, take your pick.

Maximum number of instant commands to buffer

PMATE is constantly polling the keyboard to make sure it doesn't miss any keysrokes while it is performing other tasks -- such as moving text around, displaying a line, or executing a command. However, this buffering can allow certain instant commands (such as deletes or cursor motion) to 'gun away' when used with auto-repeat. When you take your finger off the key, things can keep happening on the screen, as buffered keystrokes are executed. You can limit the maginitude of this run-away by answering the above question with a small number (but it should be at least 1) - but if you quickly strike four control-d's, and only 2 characters are deleted, you will know why. As always, compromise.

Number of characters to shift for horizontal scroll

PMATE allows lines up to 250 characters in length. Since displays will rarely show more than 80 of those, PMATE shifts the entire display over to prevent the cursor from moving off the right end. Enter the number of characters you wish shifted at one time. If this is 1, the display will scroll 1 character at a time as you enter a long line. This is very 'natural', but you'll notice continual screen activity as the line progresses. If this bothers you, choose a larger number.

Are carriage returns and tabs to be inserted while in overtype mode?

Normally (answer 'no'), in Overtype Mode, carriage returns are only inserted at the end of text, and tabs are only inserted at the end of a line. At all other times, these characters just move the cursor - to the beginning of the next line, or to the character following the next tab. However, if you answer 'yes', these characters will be inserted any time they are typed (and the cursor motion keys must used for moving the cursor).

Is this machine Z-80 based?

Finally, a simple question. Just answer 'yes' if your computer is based on a Z-80 processor, and 'no' for an 8080, 8085, or if you have no idea.

Is there a system monitor?

If your system contains a permanently resident monitor, you can CALL it with an 'XM' command from PMATE if you answer 'yes' to this question, then fill in its entry point after address?. (Return to PMATE with an RET, or jump to location 103H.)

Do you wish .BAK files to be generated automatically?

Most CP/M based text editors do not delete the original input file after a completed edit pass, but rename it, giving it the extension '.BAK' (any old file by

that name is deleted). PMATE will do that too (just answer 'yes'), but there are those of us who do not like to clutter our floppies with two copies of most everything. (If you have a hard disk, you may not be sensitive to this problem.) We answer 'no'. See also the 'QK' command to change this while editing.

Reserved size of garbage area

PMATE stacks its garbage (deleted text) in any available memory space -ready to be retrieved later if needed. By permanently reserving some space for garbage, you can be sure you can recover at least a small item or two, and be able to use the stack for moving text. Enter the number of bytes you wish reserved. It must be at least 1, and please leave some room to edit text.

Size of permanent macro area

Enter the amount of memory (in bytes) you wish to reserve for permanent macros. PMATE will not allow you to load permanent macros requiring more space than you have allocated.

CUSTOMIZING THE KEYBOARD

PMATE allows customizing the keystrokes required to perform instant commands in order to better suit your preferences and available hardware. At first, stick with the default assignments - that will make understanding the User Manual, and learning PMATE, much easier.

CONPMATE will configure a version of PMATE which will assign any keystrokes you wish to any of a list of commands. If you wish to assign more than one sequence to a command, or if you wish to create your own instant commands (as PMATE macros) and assign them to keystrokes, it will be necessary to modify IOPATCH, as explained in the next chapter.

Maximum number of codes entered for instant commands below

An Instant Command can require up to eight codes to execute. This could be a series of keystrokes, or the multi-code sequence sent out by many terminal's function keys. Enter here the maximum number of codes entered for any of the commands below.

Shift character

If your keyboard does not have a 'shift' key, enter the ASCII code of a character which will serve as a case shift. (See the QS command for more details.)

Control shift character

If you are using control codes for instant commands (hard to avoid), a 'control shift character' is needed if you wish to actually enter these control characters in text (see the 'QC' command). Enter the ASCII code for that character here (up-arrow is the usual choice).

Next follows a list of instant command functions. Enter the ASCII codes of the required keystroke sequence following each function. Not all functions need be implemented (just leave it blank if not). The .CNF files provided implement the standard PMATE instant command set.

The array of cursor motion commands requires further explanation. PMATE rev 1.0 implemented cursor motion as follows:

left: Move left 1 character. If already at the beginning of a line, move to the last character of the preceding line.

right: Move right 1 character. If already on the last character of a line, move to the beginning of the following line.

up: Move to the beginning of the current line. If already at the beginning, move to the beginning of the preceding line.

down: Move to the beginning of the following line.

This combination of cursor motion can be selected by entering codes next to **Move left**, **Move right**, **Move up**, and **Move down**. This set of commands make it very easy to get to either end of a line, and is well suited to editing programs. However, it does not allow you to easily move the cursor down through columnar material.

Another approach to vertical cursor motion is to move the cursor 'geometrically'. If the cursor is at column 5, moving up one line will leave the cursor on the preceding line, still on column 5. Normally, the cursor will not land past the carriage return at the end of a line, or in the middle of a tab -- it will only land on a text character. Thus, as you move up or down the screen, the cursor can be pushed farther and farther over to the left. However, by answering **Allow cursor to move into 'free space'?** with a 'yes', the cursor will be allowed to land anywhere, and will stay in the same column as you scroll up or down through any document. If you attempt to insert a character while the cursor is 'floating', the appropriate number of spaces will be inserted so that the character is actually inserted where you expect. **Move right (geometric)** and **Move left (geometric)** always keep the cursor on the same line, and always move by exactly one column at a time. Note that this causes trouble if the cursor has not been allowed into free space -- whenever the cursor reaches a tab, it tries to move over 1 column, can't land there, so moves back to the beginning of the tab, stuck. If you do not allow the cursor into free space, there is no advantage to the geometric horizontal motion anyway.

A final option mixes the above two approaches. Overtype Mode is well suited for working on columns, as is a geometric cursor (and carriage return can be used to move the cursor to the beginning of a line). When working on line-oriented material, Insert Mode is usually used. By entering codes in the **Move up (mixed)** and other **(mixed)** categories, the line-oriented cursor routines are used while in Insert Mode, and the geometric routines are used in Overtype Mode.

The last section of the configuration file allows you to redefine the codes which perform certain built in PMATE functions. If you wish to redefine one of these, just enter the new code (or codes) following the '***', as for any of the instant commands. You may want to use backspace (control-H) instead of rubout to delete the last entered character (but then you must find a new key for cursor motion). On many terminals, the 'rub' or 'del' code is generated by shift-underline. If you find this inconvient, just enter a 5FH after **Rubout** and a 7FH after **Underline**. This will effectively redefine the underline key so that the unshifted character is a rubout, and the shifted one is an underline. Escape, tab, and carriage return can also be redefined, but you will rarely want to. If you wish to redefine any other keys, you will have to read the next chapter. (Maybe you always mix up 'q' and 'w', and figure its easier to fix your keyboard than to learn to type.)

Chapter VII
INTERFACE GUIDE

GENERATING CUSTOM VERSIONS OF PMATE

If you really want to mess with PMATE, and you know some 8080 assembly language, then this chapter's for you. A module called IOPATCH.ASM contains the tables and code which interface PMATE to your keyboard and display. The configuration program CONPMATE just modifies the same parameters shown here, but by working with IOPATCH, you have even more control.

After you've modified IOPATCH to your needs, and assembled (getting a HEX output file), here's how to include it in PMATE. Use the CP/M dynamic debugger to load PMATE.COM and IOPATCH.HEX. Just type 'DDT PMATE.COM'. When 'DDT' responds with its prompt '-', you should enter the command 'IIOPATCH.HEX' , and then 'R'. 'I' sets up IOPATCH.HEX as an input file, and then 'R' reads it, overlaying the appropriate portions of PMATE.

There are now two possible ways to save this new version. One is to hit control-C, and return to CP/M, then use the SAVE command and your new creation will be preserved. The preferred method is to now use the debugger to run PMATE ('G100' will do it). If it seems to be working okay, use the 'XD' command to create a duplicate.

OVERVIEW OF MEMORY ORGANIZATION

Like all CP/M command files, PMATE begins at location 100H (see fig. 6-1). Location 100H just contains a jump to the starting address of PMATE. What follows is a table of vectors and constants, which you can modify or make use of. Next comes the main body of object code. Finally, the last part of the code includes the Instant Command table, and the user initialization routine. This entire portion can be overlayed by user written or modified software, extending as far as you wish.

At the end of the executable code (and still a part of the PMATE.COM file) is the permanent macro area, which you can define to be as large as you wish. Then the remaining RAM is devoted to text buffers for the editing process.

TABLE OF VECTORS AND POINTERS

There are several sections of PMATE which the user can interface to. They are all contained in IOPATCH, and are explained below. The first is a table of vectors, constants, and variables, starting at location 100H. Here's the first part consisting of vectors and pointers:

```
        ORG 100H

        JMP INIT        ;INITIALIZE
        JMP RESET       ;RESET
        JMP PINIT       ;PARTIAL INITIALIZATION
UINITL: JMP UINIT       ;USER INITIALIZATION
UEXIT:  RET             ;USER EXIT ROUTINE
        NOP
        NOP
```

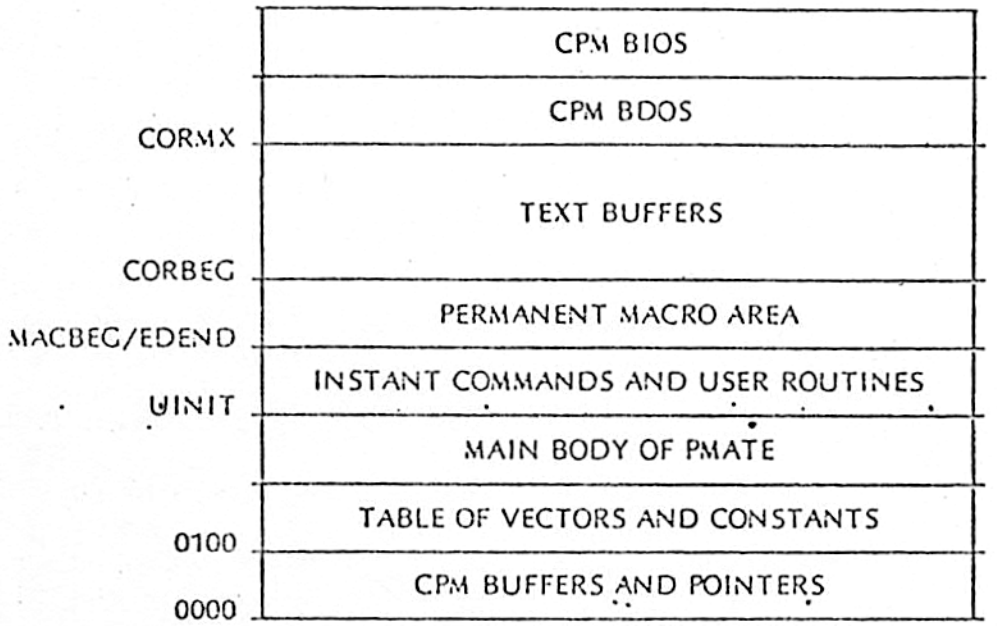

Fig 6-1. PMATE memory allocation.

```
CI:       JMP 0          ;CONSOLE INPUT VECTOR
CSTS:     JMP 0          ;CONSOLE STATUS VECTOR
COUT:     JMP 0          ;CONSOLE OUTPUT VECTOR
LO:       JMP 0          ;LIST VECTOR
LSTS:     DB 0,0,0       ;LIST STATUS VECTOR
MONTR:    RET            ;MONITOR VECTOR
          NOP
          NOP
KEYTAB:   DW KEYTB       ;POINTER TO INSTANT COMMAND
                         ;KEYSTROKE TABLE
ICSIZ:    DB 3           ;SIZE OF SLOT IN TABLE
UINST:    DW UINCM       ;POINTER TO TABLE OF USER DEFINED
                         ;INSTANT COMMANDS
UINSZ:    DB 2           ;NUMBER OF ENTRIES IN TABLE
UCOM:     DW USRCOM      ;POINTER TO USER COMMAND EXECUTED
                         ;BEFORE INPUT OF NEXT COMMAND
```

The first 3 entries are jumps to PMATE restart locations. INIT is the main initialization entry point. It clears all the text buffers, and opens input/output files defined in the command line. If you interrupt PMATE, and then try to restart by going to location 100, you may get some strange files being opened because nonsense has been written into the command line buffer at location 80H - just 'XK' them.

Jumping to RESET is equivalent to striking a control-C while in PMATE. The command area is cleared, the text buffer is left unchanged, and PMATE returns in Command Mode. After an 'XM' command gets you to the system monitor, this is the place to return.

PINIT is the partial initialization address. This is useful if there is an already existing block of text in memory which you wish to edit. Set CORBEG to the first character of the block, and put a null (0) at the end of the block. Make sure there are no nulls anywhere in between. Then enter at PINIT, and this block of text should be in your edit buffer. This can also be useful in recovering from major catastrophies. For example, if you 'XK' your edit buffer, it appears to be wiped out, but most of it will usually be sitting in memory - just a few nulls got moved down to make the buffer look empty. If you can examine memory with your system monitor, you should be able to find most of it (look following the CORBEG). Then set CORBEG to point to the beginning of usable text, and jump to PINIT through location 106H.

The next two locations are vectors into user written routines. UINITL jumps to a routine which is called from INIT (see UINIT description below) and UEXIT is called just before returning to CP/M from an 'XH' command (not normally needed, but provided for your special requirements). Next come 5 vectors to system I/O routines. These routines can be user written - however, they are compatible with CP/M BIOS routines, and can just vector there. In fact, the addressess of these routines can be computed at user initialization time (see UINIT), so that you don't need to generate a new version of PMATE to run in a CP/M system configured for a different memory size. The responsibilities of these routines are as follows:

CI	Get a character from the console, and put it in register A. If there is no character currently available, wait until there is.
CSTS	Check to see if there is a character presently available from the console. If there is, return 0FFH in register A, otherwise, return 0.
COUT	Output the character in register C to the console output device. When configured for a memory mapped display, PMATE calls this routine only to ring a bell. When configured for a serial terminal, PMATE sends its display output through here.
LO	Output the character in register C to the listing device. This is the place that 'XT' typeouts are sent.
LSTS	Return lister status -- not currently implemented.

The final jump vector at MONTR points to the system monitor. An 'XM' command gets you there. If you have no system monitor (and you don't want to tack one on at the end of PMATE), just make this a RET.

KEYTAB contains the address of the instant command table, so that you can modify it if your keyboard has special cursor keys. This is discussed later (as is ICSIZ, UINST, and UINSZ). UCOM points to an optional PMATE command string (ending in 0) which is executed every time PMATE is preparing to input another command string while in Command Mode. Use this for your special customizing requirements.

DISPLAY PARAMETERS

Next come some display constants. Most of these can be set by CONPMATE and were explained in the previous chapter - so make sure you've read that.

```
TDPSZ:    DB 21       ;NUMBER OF LINES IN TEXT SECTION
                      ;OF DISPLAY = DPSZ-3
DPSZ:     DB 24       ;TOTAL NUMBER OF LINES IN VIDEO DISPLAY
CHRLN:    DB 80       ;NUMBER OF CHARACTERS IN EACH LINE
SHFTCT:   DB 1        ;SIZE OF SHIFT WHEN CURSOR MOVES
                      ;OFF RIGHT SIDE OF SCREEN
SCRLCT:   DB 2        ;MAXIMUM NUMBER OF LINES TO SCROLL USING
                      ;INSERT AND DELETE LINE CAPABILITIES
WANDER:   DB 4        ;NUMBER OF LINES FROM CENTER CURSOR
                      ;CAN WANDER
CONTXT:   DB 3        ;NUMBER OF LINES REDRAWN IN FOREGROUND
NOLSTC:   DB 0        ;0FFH IF LAST CHARACTER OF DISPLAY
                      ;MUST BE SUPPRESSED

DOWN:     DB 0FFH     ;IF SET TO 0FFH, DISPLAY PROCEEDS FROM
                      ;TOP TO BOTTOM
EVRYLN:   DB 0        ;IF SET TO 0FFH, DISPLAY CURSOR
                      ;BEFORE DRAWING EACH LINE
BLNKCT:   DB 25       ;CURSOR BLINK COUNT -- 0 FOR NO BLINK
IGNRIC:   DB 3        ;IGNORE INSTANT COMMANDS IF DEEPER
                      ;THAN THIS IN BUFFER
```

DPSZ holds the total number of lines on your display. Make sure TDPSZ (the size of the text area on the display) is exactly three smaller than DPSZ. Of the remaining constants above, the only one not described in the CONFIGURATION chapter is SCRLCT. If you are using a serial terminal which supports insert and delete line, PMATE will use that capability to scroll anywhere up to SCRLCT lines - if it is necessary to scroll further, it will simply redraw the entire display. CONPMATE sets SCRLCT to one-third the total size of the display. Note that the flags DOWN and NOLSTC must take the values 0 or -1 (which must be 0FFH on many assemblers).

MEMORY MAPPED DISPLAYS

PMATE uses the same memory area for different parameters, depending upon whether you have a terminal, or a memory mapped screen. Notice that in IOPATCH, there is a constant, MEMMAP, which is set to -1 for a memory mapped display, or 0 for a serial display. This controls the conditional assembly of the appropriate code. This is the module for a memory mapped display:

```
MMAP:     DB 0FFH     ;SET TO 0FFH FOR MEMORY MAPPED DISPLAY
VRAM0:    DW 0E000H   ;BEGINNING OF VIDEO RAM
LSPAC:    DW 128      ;SPACE BETWEEN LINES OF VIDEO RAM
DSPCUR:               ;ROUTINE TO DISPLAY CURSOR AT MEMORY
                      ;POINTED TO BY HL
          MOV A,M
```

```
        ORI 80H
        MOV M,A
        RET
        DS 16           ;21 BYTES TOTAL

CLRCUR:                 ;ROUTINE TO CLEAR CURSOR FROM MEMORY
                        ;POINTED TO BY HL
        MOV A,M
        ANI 7FH
        MOV M,A
        RET
        DS 16           ;21 BYTES TOTAL

DSPCHR:                 ;ROUTINE TO STORE CHAR IN REG A IN
                        ;VIDEO MEMORY POINTED TO BY DE
        STAX D
        RET
        DS 19           ;21 BYTES TOTAL
```

These all correspond to entries in the .CNF configuration files, and have already been explained. DSPCUR, CLRCUR, AND DSPCHR must all be exactly 21 bytes in length (so PMATE knows where to find them) - so adjust the DS's accordingly.

VIDEO TERMINALS

For video terminals, set MEMMAP to 0 in IOPATCH, and the following will be assembled:

```
MMAP:   DB 0            ;0 FOR MEMORY MAPPED DISPLAY

        ;VIDEO CODES - SEQUENCES END IN 0, THEN NEXT BYTE
        ;IS NUMBER OF MILLISECONDS TO DELAY
VIDCLS: DB 26,0,0,0,0,0,0,0,0 ;SEQUENCE OF BYTES TO CLEAR SCREEN
VIDCLL: DB 27,'T',0,0,0,0,0,0,0 ;CLEAR TO END OF LINE

        ;CURSOR ADDRESSING
VIDASC: DB 0            ;ASCII FLAG - -1 FOR ASCII
VIDXY:  DB 0            ;XY FLAG -- 0 FOR Y COORDINATE FIRST
                        ;-1 FOR X FIRST
VIDOF1: DB 32           ;OFFSET OF FIRST COORDINATE
VIDOF2: DB 32           ;OFFSET OF SECOND COORDINATE

VIDCUL:                 ;LEAD IN
        DB 27,'=',0,0,0,0,0,0,0
VIDCUM:                 ;MIDDLE
        DB 0,0,0,0,0,0,0,0,0
VIDCUE:                 ;END
        DB 0,5,0,0,0,0,0,0,
```

```
VIDIL:                          ;SEQUENCE TO INSERT LINE -- END IN 0
        DB 27,'E',0,10,0,0,0,0,0
VIDDL:                          ;SEQUENCE TO DELETE LINE -- END IN 0
        DB 27,'R',0,10,0,0,0,0,0
```

All of the above sequences must be exactly 9 bytes long. Each sequence includes up to 7 characters which are sent to the display, followed by a null, followed by any delay required after sending this sequence. This delay is in milliseconds for a 4MHZ CPU. In the example above, a 10msec delay is inserted after insert and delete line, a 5 msec delay after the entire cursor addressing sequence, and no delay after clearing the screen or clearing to end of line. The cursor addressing sequences are exactly as described in the previous chapter. VIDCUL is the lead-in sequence. VIDCUM is sent between the X and the Y coordinates, and then VIDCUE is sent. It is possible to insert a delay after each of these sequences, but one should only be needed at the very end. VIDASC is set to -1 (0FFH) if the cursor positioning information must be sent out as an ASCII string (e.g. '2','3'), rather than as one binary byte. VIDXY indicates whether the row or column information is sent out first, and VIDOF1 and 2 hold offsets. If the offsets are different, make sure VIDOF1 holds the offset of the first coordinate sent out, be that X or Y.

MORE

```
Z80:    DB 0FFH         ;0FFH IF Z-80, 0 IF 8080
DELAY:  DB 100          ;DELAY TIME FOR QD COMMAND
ABRT:   DB 'C'-40H      ;ABORT CHARACTER
ICRFL:  DB 0            ;0FFH IF CR'S AND TABS ARE TO BE
                        ;INSERTED IN OVERTYPE

CORBEG:                 ;FIRST AVAILABLE CORE LOCATION
        DW EDEND+MACSZ+1
CORMX:  DW 0            ;LAST AVAILABLE CORE LOCATION
GBGSZ:  DW -1000        ;NEGATIVE MINIMUM SIZE OF GARBAGE AREA
TXTEND: DW 0            ;LAST LOCATION AVAILABLE FOR TEXT
                        ;REST IS RESERVED FOR GARBAGE
MACBEG: DW EDEND        ;FIRST AVAILABLE LOCATION
                        ;FOR PERMANENT MACROS
MACEND:                 ;LAST AVAILABLE LOCATION
                        ;FOR PERMANENT MACROS
        DW EDEND+MACSZ

        ;THE FOLLOWING VARIABLES CAN BE SET BY THE Q COMMANDS

UVAR0:  DW 0            ;USER DEFINABLE VARIABLES
UVAR1:  DW 0
UVAR2:  DW 0
UVAR3:  DW 0
UVAR4:  DW 0
UVAR5:  DW 0
UVAR6:  DW 0
UVAR7:  DW 0
```

```
UVAR8:    DW 0
UVAR9:    DW 0

SHFCHR:   DB 0          ;UPPER OR LOWER CASE SHIFT CHARACTER -- QS
CNTCHR:   DB '@'        ;CONTROL CHARACTER -- QC
PAGSZ:    DW 0          ;NUMBER OF LINES IN PAGE -- QP
PAGSEP:   DB 'L'-40H    ;PAGE SEPARATOR -- QF
SCRLNS:   DW 6          ;NUMBER OF LINES TO SCROLL
                        ;IN INSTANT COMMANDS -- QL
BKUFL:    DB 0FFH       ;0FFH IF BACKUPS ARE TO BE MADE -- QK
XMAX:     DB 250        ;MAXIMUM ALLOWED X CURSOR POSITION -- QZ
CRCHR:    DB '<'        ;DISPLAYED FOR END OF PARAGRAPH
```

The variable 'Z80' may be set to 0FFH if the host computer is Z-80 based. PMATE will then use the Z-80 block move instructions to speed up text insertions and deletions. The 'QD' command delays for a time proportional to the number stored in 'DELAY'. If you wish to make that delay an exact time interval for your hardware, you may play with this number. The character in 'ABRT' is the abort character (normally control-C). If you change the abort character in the instant command table, you must also change it here. Set ICRFL to -1 if you wish carriage returns and tabs to be inserted while in overtype mode, rather than just moving the cursor.

The next 6 words help define PMATE's memory allocation. Normally, the permanent macro area begins immediately after the end of the editor (defined by EDEND), and is as large as you want it. Any remaining memory space up to the BDOS (resident part of CP/M which must remain in RAM) is usually allocated to PMATE. You can do this 'by hand' - just put the address of the last available memory location in CORMX. Usually, however, this is done by 'UINIT' (user initialization routine) so that all available memory is automatically gobbled up. UINIT can also initialize TXTEND. This is the highest address of memory allocated to text buffers. Any memory between here and CORMX is reserved for the garbage stack. Fill in the negative of the amount of space you wish reserved for garbage in GBGSZ. See the 'User Initialization' section for more details.

The ten variables, UVAR0 through UVAR9 can be set by a 'Q' command. They can be used to control your keyboard, CRT, or printer drivers, (or anything else you wish to control with PMATE commands). For instance, if you had 3 different printers connected to your system, UVAR0 might control which printer the next 'XT' output would go to.

The next 7 bytes just initialize paramaters which can be changed with the 'Q' commands shown (see COMPLETE COMMAND SET chapter). CRCHR holds the character which is displayed at the end of a paragraph while in Format Mode.

USER INITIALIZATION

The user initialization routine is entered after PMATE is started at its main initialization point (100H). You should use this routine to initialize any display hardware (some 'DMA' displays need a byte or two to initialize their mode).

A routine to initialize the I/O vectors CONIN, CONST, CONOUT, and LIST to point to the corresponding BIOS routines in CP/M is given below. This routine also

sets CORMX, appropriating all available memory for PMATE. The BIOS jump table is located by using the jump to the warm-boot vector normally stored beginning at location 0000H. CORMX is set using location 0006H, which CP/M sets up to point to the beginning of the BDOS. The reserved 'garbage stack' area is allocated by using GBGSZ, and the result is stored in TXTEND.

```
LHLD 06H        ;POINTER TO BEGINNING OF FDOS
DCX H
SHLD CORMX      ;LAST AVAILABLE CORE LOCATION
XCHG
LHLD GBGSZ      ;NEGATIVE OF SIZE ALLOWED
                ;FOR GARBAGE AREA
DAD D
SHLD TXTEND

LHLD 01H        ;POINTER TO WARM BOOT VECTOR
LXI D,3
DAD D           ;CONSOLE STATUS VECTOR
SHLD CSTS+1
DAD D
SHLD CI+1       ;CONSOLE IN VECTOR
DAD D
SHLD COUT+1     ;CONSOLE OUT
DAD D
SHLD LO+1       ;LIST DEVICE
RET
```

THE INSTANT COMMAND TABLE

The instant command table allows you to define 1 or more keystrokes which taken together cause any predefined (or user defined) command string to be executed. The table provided in IOPATCH implements the default instant commands. UINST, in the table of vectors and pointers, contains a pointer to this table. ICSIZ contains the maximum number of codes needed for any command. The first byte in each 'slot' gives the command number. These start at 128. If a number of less than 128 is entered, this is treated as an ASCII code, and not a command (this can be used to implement translations - such as backspace=>del, or ^X to esc). The following character(s) in the slot are the keystrokes which together execute the command (or are translated into the ASCII code) in the first byte. ICSIZ indicates how many bytes are needed here - fill in the remainder with 0's.

Here is a list of the predefined command codes. Further explanation of their function is given in the preceding chapter.

128 -- Move to beginning of buffer, if there, move to end
129 -- Move to end of buffer
130 -- Move cursor left
131 -- Move cursor left one word
132 -- Move cursor right
133 -- Move cursor right one word

134 -- Move up one line
135 -- Move up 6 lines
136 -- Move down one line
137 -- Move down 6 lines
138 -- Delete character
139 -- Kill line
140 -- Go to insert mode
141 -- Edit command
142 -- Abort
143 -- Shift case
144 -- Redraw and reformat display
145 -- Tag current cursor position
146 -- Delete word forward
147 -- Delete word backward
148 -- Pop garbage stack
149 -- Go to command mode
150 -- Go to overtype mode
151 -- Insert line
152 -- Move cursor left (geometric)
153 -- Move cursor right (geometric)
154 -- Move cursor up (mixed)
155 -- Move cursor down (mixed)
156 -- Move block
157 -- Get block
158 -- Move cursor up (geometric)
159 -- Move cursor down (geometric)
160 -- Move to top of buffer
161 -- Move cursor left (mixed)
162 -- Move cursor right (mixed)

Several sets of keystrokes can all enact the same command, and not all commands need be implemented. Add as many slots as you wish to the table, and end the table with a byte of 0FFH. If you change the size of a slot, make sure you change all of them, and ICSIZ.

You can add your own commands by using a command code starting at 128+64. The pointer UINST contains the address of a table of user defined instant commands - here called UINCM. This table is an index containing pointers to command strings you provide. The size of the table UINCM (i.e. the number of two byte entries) must be stored in UINSZ. Command number 192 will execute the macro string pointed to by the first entry in the table, while 193 refers to the second, etc. IOPATCH contains, as an example, two added user commands. Use this facility to customize your own PMATE, or to emulate other editors you are used to working with.

Appendix A
COMMAND SUMMARY

The following instant commands are keystrokes which are not entered into the command or text buffers, but are executed immediately:

Cursor Motion:

©A	Move to the beginning of the text buffer, and if already there, move to the end
©G	Move left one character
©H	Move right one character
©Y	Move up one line
©B	Move down one line
©U	Move up multiple lines
©J	Move down multiple lines
©O	Move left one word
©P	Move right one word

Delete:

©D	Delete the character at the cursor
©K	Kill the line beginning at the cursor
©W	Delete one word beginning at cursor
©Q	Delete one word backwards from cursor

Text movement and recovery:

©T	Tag the current cursor location
©E	Move block between tag and cursor to special buffer
©Z	Get contents of special buffer to cursor location
©R	Pop garbage stack, restoring last deleted item

Mode:

©X	Go to COMMAND MODE
©V	Go to OVERTYPE MODE
©N	Go to INSERT MODE

Other:

©L	Insert line
©C	Abort any operation in progress, resetting to command mode
©	Edit the command string
©S	Shift default case (useful for non-typewriter style keyboards)
©F	Redraw and reformat display

These characters are not really instant commands, but they do have special meanings:

ESC	The escape key separates commands in command mode. Two consecutive escapes execute the command.
TAB	The tab character in text positions the following character at the next tab stop.
DELETE/RUBOUT	The key marked 'delete' or 'rubout' deletes the last character entered.
^	The up-arrow itself is ignored, but it shifts the next character entered to a control character. Strike it twice to enter an up-arrow.

Now come the real commands. When in Command Mode, these are entered into the command buffer and then executed.

The following commands move the cursor:

L	forward 1 line
+nL	forward 'n' lines
-nL	backward 'n' lines
M,nM,-nM	'n' characters
W,nW,-nW	'n' words
P,nP,-nP	'n' paragraphs
A	to beginning of text buffer
Z	to end of text buffer

The following commands delete characters:

D	Delete character at cursor
nD	Delete n characters, from cursor forward
-nD	Delete n characters, from cursor backwards
K,nK,-nK	Delete lines

The following commands insert text into the text buffer:

Istring	Insert 'string' immediately after cursor
nI	Insert character with ASCII code 'n'
Rstring	Overwrite text with 'string'
nR	Overwrite character at cursor with ASCII code 'n'
n \	Insert number 'n' into the text

Search and change:

Sstring	Search for next occurence of 'string'
nSstring	Search forward for next occurence of 'string', confining search to 'n' lines
-Sstring	Search backward for next occurence of 'string'
-nSstring	Search backward for next occurence of 'string', confining search to 'n' lines
Cstrng1Sstrng2	Change next occurence of 'strng1' to 'strng2'
nC,-C,-nC	Search for 'strng1' as in equivalent 'S' command, then change it to 'strng2'

Iteration and control:

I]label	Jump if 'I' is true to 'label'
I[. .]	Execute expression in brackets only if 'I' is true
I[. .][. .]	Execute expression in first brackets if 'I' is true, otherwise execute expression in second set of brackets
n[. .]	Iterate expression in brackets 'n' times
[. . I]	Iterate until 'I' is true
I ^	Proceed to next iteration if 'I' is true
I_	Exit enclosing iteration loop if 'I' is true

Some miscellaneous commands:

E	Suppress error messages
nF	Enter Format Mode, setting line width to 'n'
F	Toggle in and out of Format Mode
Gstrng	Get key from keyboard, giving user prompt 'strng'
N	Go into 'Insert Mode'
T	Tag current cursor position

.b	Execute macro 'b'
l%	Return early from macro if 'l' is true
:x	Label this position in command with character 'x'
;	Comment - ignore all characters until end of line
?	Enter trace mode

There are 10 numeric variables and a Number Stack for storing intermediate results, and some commands to set them:

nVi	Set variable 'i' to value 'n'
VAi	Increment variable 'i'
nVAi	Add 'n' to variable 'i'
n,	Push 'n' on number stack

The following 'Q' commands perform miscellaneous functions, usually setting some internal parameter:

nQA	Set the number of passed string arguments in a macro call
QB	Ring the bell
nQC	Set control shift character to ASCII 'n'
nQD	Delay for a time proportional to 'n'
nQE	Set type-out mode to 'n'.
nQF	Set page separator character to ASCII 'n'
nQH	Insert 'n' spaces at cursor position
nQI	Set input radix to 'n'
nQK	Create .BAK files unless 'n' is 0
nQL	Set number of lines for ©U andd ©J commands to scroll
QMC	Copy to permanent macro area
QMG	Get contents of permanent macro area
nQO	Set output radix to 'n'
nQP	Set page size to 'n'
nQR	Redraw screen -- return any key struck as @K
nQS	Set lower case shift character to ASCII 'n'
nQX	Move screen cursor to column 'n'
nQY	Allow cursor motion in 'free space' if n=0
nQZ	Don't allow cursor to move past column 'n'.
nQ!	Store 'n' in memory at location pointed to by variable 9.
nQ-	Display numbers as positive only if n=0
nQm	Set user variable 'm' (0-9) to 'n' - these 10 user variables are available to user written I/O drivers.

The following 'X' commands generally perform disk I/O. They begin with an 'X' so that they are hard to accidentally execute, as they cause major upheaval.

XA	Append next page of input file
nXA	Append next 'n' pages of input file
XW	Write next page to output file
nXW	Write next 'n' pages to output file
XR	Write one page to output file, read one from input file
nXR	Do this 'n' times
XE	End of editing - write out all remaining text from buffer and input file
XJ	Do XE, then reopen file
XF	Define new input and output files
XK	Delete output file and scratch edit buffer
XC	Close input and output files as they are.

XH	Return to CPM
XIfile	Input entire file 'file'
nXIfile	Input n pages of 'file'
nXI	Input n pages of last named auxilliary input file
XOfile	Output entire edit buffer to 'file'.
nXOfile	Output 'n' lines, beginning at cursor, to 'file'.
XM	Go to monitor
XDfile	Create new version of PMATE, including any new changes, or permanent macros - new version called 'file.COM'
XSd	Log in disk drive 'd' ('d' is A,B,C, etc.). Also reset disk system.
XT	Type entire text buffer on printer
nXT	Type 'n' lines, beginning at cursor
XL	List disk directory at cursor
XLfile	List just those files in directory which match 'file'
XXfile	Delete 'file' from disk

The following 'B' commands act on buffers 0-9, or the text buffer 'T' -buffer 0 is assumed, unless the buffer number is placed between the two characters of the command.

BK	Kill the entire contents of the specified buffer
BG	Get the contents of the specified buffer
nBC	Copy 'n' lines to the specified buffer
nBD	Append 'n' lines to the specified buffer
nBM	Move 'n' lines to the specified buffer
nBN	Append move 'n' lines to the specified buffer
BE	Edit the specified buffer

Tab stop commands:

nYD	Delete tab stop at position 'n'
nYS	Set a tab stop at position 'n'
YK	Kill all tab stops
nYE	Set a tab stop every 'n' spaces
YF	Fill tabs with appropriate number of spaces
YR	Replace spaces with tabs where possible
nYI	Set indent at column n

Numeric arguments (which have been referred to as 'n') can in fact be complex expressions, involving up to 15 levels of parenthesis, and the following operations:

+	Addition
-	Subtraction
*	Multiplication
/	Division
!	Logical or
&	Logical and
'	Logical complement
<	Less than
>	Greater than
=	Equal

In addition to numbers, the following expressions can be used with the above operations to form numeric arguments

'a	The ASCII value of character 'a'

@i	The value of numeric variable 'i'
@A	The numeric argument when macro was called
@C	The character number
@E	The value of the error flag
@Ffile	-1 if 'file' exists on the current directory, 0 if it doesn't
@G	The length of the last referenced string
@Hstrng	Compare 'strng' to text at cursor - return 0 if equal, otherwise 1 or -1, depending upon which string is greater.
@I	The current input page
@K	The ASCII value of the key struck after a 'G' or QR command
@L	The line number
@M	The amount of memory remaining
@O	The current output page
@P	The absolute memory address to which the cursor is pointing
@R	The remainder of the last division
@S	Pop the number stack -- get value of top
@T	The ASCII value of the character pointed to by the cursor
@V	The current mode
@W	The current right margin
@X	The current column
@Y	The current left margin
@Z	The column of the next tab stop
@@	The byte pointed to by variable 9
=	Move cursor to tagged position, and get difference between tagged position and current position as argument - can be used with any character or line oriented command to operate on a block of text