Section 1: Overview and Introduction

1.1 What is Mince?

Mince is a display-screen oriented text editor, originally optimized for use on small computer systems. It is patterned after an editor called "Emacs", a text editor heretofore found generally at very large research computer installations.

1.2 Mince for Text Entry

Mince text entry has been made as easy as possible for the terminal operator. To enter text into a document, merely type it. No complicated commands are needed for this inherently simple operation. There are no "input" or "edit" modes, as in some text editors. Text which is entered is displayed as it is typed in. The screen display always shows text exactly as it will appear in the computer file and upon output; what you see is what you get.

1.3 Mince for Editing Text

Editing with Mince is easily and quickly learned. In general, Mince commands are mnemonically assigned to keys. For example, the editor commands which move forward in the text are associated with the "F" key and commands which move backward in the text are associated with the "B" key. Mince has an extremely powerful command set, but with the knowledge of only a small subset of those commands, most standard editing tasks can be accomplished with ease. Each key performs a command, and the result of the command happens immediately. There is no need to type carriage-returns to enter the editing commands. The screen display reflects the result of these commands as they are typed. This mnemonic command-naming and continuous interaction with the text rather than using a structured set of "editing requests" make Mince easy for the novice to learn yet efficient for experienced users to operate. The continuously-updated display helps all classes of users keep track of exactly what they are doing to the text.

1.4 Mince for Editing Programs

Programmers as a class appreciate superior speed and power from their editors. Mince supplies both. Several programs may be edited at once, and many of the textual editing features regarding sentences, words, and paragraphs are easily used with structured languages where such concepts translate to tokens,

statements, and blocks.

1.5 Mince and the Future

Mince is an editor which will not become obsolete with changes in hardware; it easily adapts to different terminal types without extra (and expensive!) software. Furthermore, Mince is written in a high-level language whose transportability will accompany the program development cycle for several years to come. The time spent learning Mince will not be wasted when 8080/Z80 systems have been replaced by newer hardware.

People who have not read either the Programmer's Introduction to Mince or the Mince Lessons should stop here rather than continuing with this user's guide. New users are best advised to take one of the introductory sets of lessons, sit down at a terminal, and get started. Mince is easiest to explain and learn by using it!

1.6 A Mince Glossary

The following symbols or terms are used in the command descriptions presented below. Familiarity with these items is a prerequisite for complete understanding of the Mince commands.

Point - This is the position in the text where editing occurs. All text which is typed into the document is entered at the Point and all text which is deleted is removed there as well. Each text buffer has its own Point. The Point is always BETWEEN any two characters. The cursor is always on the second of those two; hence the Point is just before the character that the cursor is on. Thus, many Mince commands do nothing but move the cursor to the right place on the screen (and thus move the Point to the right place in the text) to perform the next editing operation. It is perfectly easy to think of editing in terms of the cursor, but certain text entry and deletion operations can only be explained properly by thinking of this Point as between two characters.

Newline - This item is the character which causes Mince's display to move the cursor on the screen to the beginning of the next line. Newlines are always typed by hitting the carriage-return key on the terminal keyboard. Newlines are translated to the carriage-return/linefeed combination during text output, but they are treated as single characters for the purposes of text entry and deletion. During string input for commands which ask for string

arguments, the Newline will display as <NL>, in order to definitely indicate that a Newline is part of the string to be passed to the command.

move - This verb always means "move the Point" or "move the cursor". All Mince editing operations are done by "moving to" the appropriate place in a document (and on the screen) and making whatever textual changes are necessary there.

word - A word, as defined for the word movement and deletion commands, begins at the first alphanumeric character found and extends until a non-alphanumeric character.

sentence - A sentence, as defined for the sentence movement and deletion commands, begins at the first word found and extends until a punctuation mark is found.

paragraph - A paragraph, as defined for the paragraph movement or filling commands, extends until either a blank line (two Newline characters in a row), a line beginning with a tab (a Newline followed by a tab character), or a line beginning with an at-sign ("@") or period (".") (used by some text formatters to begin commands).

Mark - This is an invisible indicator in the text buffer. It may be set at a particular position by using the "Set Mark" command. Like the Point, it rests between two characters. Also like the Point, there is one Mark for each text buffer.

"C-" - This is the prefix which we will use in the text to refer to Control commands. These are formed by holding down the key marked "CONTROL", "CNTRL", "CTRL", or the like on the terminal keyboard while typing another character, usually alphabetic. Control characters are displayed on the screen as a caret or uparrow (^) followed by that character.

"M-" - This is the prefix which we will use in the text to refer to Meta-commands. These are formed by typing the key marked "ESCAPE", "ESC", "ALT", or "ALT MODE" on the terminal, then typing another character.

"M-C-" - This prefix is for the Meta-Control-commands. It is formed by typing the ESCAPE key, then typing the appropriate Control-command.

<CR> - A symbol used to refer to either the carriage-return key (labelled "RETURN", "CR", or "ENTER" on the keyboard). This key sends ASCII ^M, decimal 13.

<LF> - A symbol used to refer to either the linefeed key (labelled "LINEFEED" or "LF" on the keyboard, if it is

present at all). This key sends ASCII ^J, decimal 10.

<DEL> - A symbol used to refer to either the delete key (labelled "DELETE", "DEL", or "RUBOUT" on the keyboard). This key sends ASCII ^?, decimal 127.

<ESC> - A symbol used to refer to either the escape key (labelled "ESC", "ALT", "ESCAPE", or "ALTMODE" on the keyboard). This key sends ASCII ^[, decimal 27.

<TAB> - A symbol used to refer to the tab key. This key sends ASCII ^I, decimal 9.

<BS> - A symbol used to refer to the backspace key (sometimes labelled "BS" on the keyboard). This key sends ASCII ^H, decimal 8.

Section 2: The Display

The Mince screen display is divided into three areas. The major portion is the "window", where the text to be edited is displayed. Two smaller portions, the "mode line" and the "echo line", appear at the bottom of the screen, beneath the window.

2.1 The Window

The window occupies the upper portion of the screen. This area is where all text which is typed and all text which is about to be modified is displayed. The window displays about twenty or so consecutive lines of the document which is being edited. You may think of this as a window onto a larger entity of text, the entire document. The purpose of this window is to always show what the portion of the document contained in it looks like. Therefore, as text is inserted or deleted, the screen updates immediately. A fundamental principle of Mince is that what you see on the screen is what you actually have in your text.

2.1.1 The cursor

The screen display always has the terminal's cursor at some point in it, and as you will see, while the cursor is in the window, it is at the position where Mince commands will affect the text. The cursor is always positioned to the right of the Point in the document. (You may think of this as the Point being attached to the left edge of the cursor instead.) This property makes the cursor a very important object in the Mince display. Many Mince commands do nothing but move the cursor to the right place on the screen (and thus move the Point to the right place in the text) to perform the next editing operation.

2.1.2 Redisplaying the window

Since the window is always supposed to look exactly like its twenty-line portion of the text, any changes in the text must cause changes to the window display. Terminals have only one cursor, and screen updates are always performed at the cursor, since the text updates are also performed there. (Remember, the Point is attached to the cursor while the cursor is in the window.) Because of this, during screen update the cursor may move a great deal and there is no way to tell where the cursor is "supposed to be", (and thus where the Point actually is in the text) except by memory or dead reckoning. Fortunately, this update occurs very rapidly, and the cursor always returns in the window to where the Point is in the text.

It is possible to type very very quickly or to execute so many editor commands which have such large effects on the text that the screen update will not be able to keep up. In this case, Mince assumes that you know what you are doing and are not relying upon the screen redisplay. In these cases, it keeps trying to update the display to reflect the text in the buffer and the cursor's position in it, but it always processes typed-in characters before doing any other work. This priority can cause the screen to lag your typing somewhat. When you finally stop typing at full tilt, Mince will then update the screen to reflect the final state of the text in the window.

2.1.3 Special characters in the window

Special characters in the ASCII character set which are normally non-printing are displayed in Mince's window. This allows you to edit any type of file at all, with no worries about whether or not any information is missing from the screen displays. All the nonprinting characters can be represented by the ASCII control characters. The way Mince displays control characters on the display screen is by printing a caret or uparrow ("^") followed by the character which represents the control code in the ASCII collating sequence. (For example, ASCII Control E, decimal 5 (also known as ASCII "ENQ"), will be displayed as "^E".) All commands treat these as single characters, except the screen display which translates them for output. The Mince commands act on the "text" itself, not on the printed representation of it.

Mince displays the tab character (ASCII ^I, decimal 9) as moving the text which follows it on the line over to the next tab stop, rather than as a "^I". It displays a Newline as moving text which follows it to the beginning of the next line of the screen. Occasionally, Mince will display a Newline where there is not really one in the text. This is the only exception to the rule that what is on the screen is identical to what is in the text. If the text to be displayed would have run off the right-hand edge of the screen, Mince will display the rest of the line on the next line of the screen. It is usually easy enough to identify this condition, as the last word on the too-long line will frequently be cut in half.

2.2 The Mode Line

The second screen area of major importance is the line just below the text window. It is called the "mode line" and should look something like:

```
Mince Version 2.5 (Normal) buf1: X:FIRSTNAM.LST -23%- *+
```

This line tells you several things, namely:
(1) You are talking to Mince rather than the operating system.

(2) You are typing in a mode called "Normal". In this mode, all commands typed are treated just as they are explained in the Mince Command List. There are other modes available, for example "Fill" mode or "Page" mode, each of which changes the Mince command set slightly. (See Section 6.)
(3) You are editing text in a buffer named "buf1". This name is used when switching from buffer to buffer. (Buffers are explained in Section 5.)
(4) You are editing a file called "X:FIRSTNAM.LST". This name is used when reading and writing files from the text buffer and the file system. (See Sections 4 and 5.)
(5) The Point is approximately 23% of the way through the file. (Since the measure is only approximate, you may never see 100%, even if you are at the end of a file. For an exact measure of the cursor's position, try the "Where Am I" command, "C-X =".)
(6) The buffer has been modified since it was last written out to the file. (This is indicated by the asterisk at the right edge of the mode line.)
(7) The next text deletion command which stores deleted text in the kill buffer will append what it deletes to whatever is already contained in the "kill buffer" (see Section 5). (This is indicated by the plus sign at the right edge of the mode line.)

2.3 The Echo Line

The third important screen area is the one line of the screen left below the mode line. This area is called the echo line, for three reasons. First, it will echo any prefix characters (the first keystroke of any of the two-keystroke commands) typed. If any of these is typed and a particularly long amount of time (between .5 and 1 second) passes before the following character is typed, the prefix character will be indicated in the echo line. For example, if the Escape key is typed, the phrase "Meta:" will appear so that you know that you have indeed typed the prefix character. If not much time passes between the two keystrokes, this display is not performed. This behavior is designed to give confident users optimum response without the screen always flashing messages, yet give nervous users information on what they are doing. Secondly, the echo line is used for reading and displaying the arguments for some commands which require strings as input (e.g. the Forward String Search command). (See Section 3.4.2.) During these string-argument-read operations, the cursor will not be in the window in its usual spot attached to the Point. Finally, error messages from commands will appear at the far right edge of the echo line, as do informative messages which may be displayed while some commands are executing. Error messages displayed in the echo line are accompanied by a bell indicator, to alert the user of the error condition.

Section 3: The Commands

3.1 Text Insertion Commands

To Mince, everything you type is a command. Yes, everything. Even ordinary letters, numbers, and punctuation which you type are commands. They are very simple commands, though; they merely instruct Mince to insert themselves into the text. Because of this arrangement, Mince has no "insert mode" and "edit mode" which determine whether characters typed are commands or text to be inserted.

3.1.1 Ordinary text

All printing characters: a-z, A-Z, 0-9, space, and !"#$%&'()*+,-./:;<=>?@[]^_{|}~` self-insert. The characters are inserted at the Point, that is, they are inserted just in front of where the cursor is on the screen. The Point is then left after the new character which was inserted. This definition means that on the screen, the cursor will move over one character position. Simply stated, text typed on a blank line moves the cursor just as a typewriter carriage would move after each character. Since characters are inserted at the Point, if the Point happens to be in the middle of some line of text, the rest of the line is moved over to make room for the new text.

3.1.2 The <TAB> key

This command, while nominally self-inserting (It inserts an ASCII ^I, decimal 9.) has a characteristic which affects the text screen display. When the tab character is displayed, the cursor is moved over to the next tab stop on the display screen before continuing to display text. This is not unusual, but is mentioned here because it is one of the characters which, when inserted in the text, takes up more than one column on the display screen.

3.1.3 The <CR> key

The carriage-return key inserts a character, the Newline, which causes the cursor to go the beginning of the next line on the screen. If a <CR> is typed when the Point is in the middle of a line, the line is split in two, with the portion of the line after the Point being moved down and turned into the next line. This is a little counterintuitive if you haven't used an editor before, but if you consider the Newline to be a character just like the other self-inserting ones, it makes

sense: <CR> inserts a Newline character at the Point, and moves the Point past the character. What this looks like to the operating system, when the file is stored in the file system, is a carriage-return/linefeed combination (ASCII ^M/^J or decimal 13/10). In fact, this is what usually causes the display screen action of going to the beginning of the next line. But it's easier to consider the Newline character to be a single one, for text insertion and deletion purposes. (It is possible to enter just a carriage-return (^M) or a linefeed (^J) by means of the C-Q command, but typing the carriage-return key causes the Newline character to be entered into the text.)

3.1.4 The <DEL> key

While this character is not an insertion command, it is so intimately tied to them that it must be mentioned here. The <DEL> key is used for correcting typing mistakes; it deletes the last character typed. Since a Newline is treated as an ordinary character, a <DEL> can delete <CR>'s typed just like any other character. A more complete description may be found in the Command List.

3.2 The Simplest Editing Commands

How are "true" editing commands differentiated from actual text? By the use of the terminal's CONTROL key. The most frequently used editing commands are one character long, and that character is a control character. For example, a C-F causes the cursor on a video display screen to be moved forward a character, and C-B causes it to be moved backward a character. The control characters are generated by holding down the control key (just like the Shift key) and typing the character. The case of a letter makes no difference to Mince command characters; "C-f" is the same as "C-F".

The commands are mnemonically named. Occasionally the mnemonics are stretched a little, but by and large the commands have some relation to the letter upon which they are placed. This makes it possible to remember in a natural manner all the commands which manipulate the text. The special function keys on the terminal keyboard are not usually used for another reason: it is not possible to use these keys without having the typist's hands leave the standard keyboard position. This choice means that it takes a little longer to learn the command set, but the time made up later by command touch-typing is well worth the effort.

In most editors, since there is a differentiation between text insertion mode and editing mode, it is easy to type the characters which make up the editor command set into the text. Since Mince commands are all control characters it is unlikely that you will want to type them into your text. If you do,

however, there is a command to "quote" the next character and insert it into the text directly rather than interpreting it as a command. (See C-Q in the Command List.)

3.3 The Two-Keystroke Editing Commands

As you will see, there is a progression in command names. The simplest and most-used commands are the easiest to type. Text itself, for example, is very easy to type. Those one-character commands which are textual characters self-insert into the document you are editing. Commands used to edit single characters or lines are quite frequently used, and are placed on the simple control characters. The more complicated and lesser-used commands are slightly harder to type and require two keystrokes.

3.3.1 Meta-commands

Some of two-character commands are called "meta-commands". These are generated by typing the ESCAPE key, then typing a character. Typing the escape key causes the following character to be treated as a Meta-command. If sufficient time elapses after typing the <ESC> and the following character, the message "Meta:" will appear in the echo area to indicate that Mince is waiting for a character to make up one of the Meta-commands. A C-G will cancel the Meta prefix and leave the text unchanged. Again, case makes no difference; the second character typed may be either upper or lower case. For example, typing ESCAPE F (M-F) moves the cursor forward a word, and ESCAPE B (M-B) moves backward a word. Commands which operate on words, paragraphs, or the like are usually two-keystroke meta-commands.

3.3.2 Meta-control-commands

There is another set of two-keystroke commands, which is a combination of the control commands and the meta-commands. These are the "meta-control-commands", which are given by typing an ESCAPE, then the appropriate control character.

3.3.3 Control X commands

Yet another (but the last!) set of two-keystroke commands is the set of "Control X commands." These are nothing more than an ordinary one-keystroke command prefixed by the one-keystroke C-X command. Similarly to the Meta prefix character (<ESC>), typing the C-X will display "Control X:" in the echo area while waiting for the second character in the sequence, and typing C-G will cancel the C-X prefix.

3.3.4 Relation between simple and two-keystroke commands

There is a relation between the control commands, the

meta-commands, and the meta-control-commands. Usually, the character upon which the commands is based has similar effects in both the control and meta-commands. For example, C-A moves to the beginning of a line and C-E moves to the end. M-A and M-E move to the beginning and end, respectively, of sentences. Occasionally, the control and meta-commands may be direct opposites instead. For example, C-V views the next screen of text, whereas M-V views the previous screenful. In almost all cases, though, they are related in some way.

3.4 Arguments to Commands

Most Mince commands are self-contained. Some of them, however, need parameters, or arguments, to accomplish their task. For example, the string search command needs to know what string to search for. The command which sets tab spacing needs to know what column increment to use. Or perhaps Mince wants to know if it should destroy a buffer of text to read in a new file. These arguments give the commands any extra information they need to perform properly. In particular, numeric arguments are useful, as they allow repetition of the other Mince commands.

3.4.1 Numeric arguments (repeat counts)

Commands may be given numeric parameters as arguments. These arguments usually are used as repeat counts. For example, an argument of 100 to the C-F command would cause the cursor to move forward 100 characters forward rather than the usual 1 character. Occasionally, arguments are used differently; for example, an argument to the command which sets the right margin specifies the column number to set it at, rather than the number of times to reset the right margin! Since all ordinary textual characters are actually commands too, they can be given arguments in order to insert several of themselves at once. For example, an argument of 20 to "*" will insert twenty stars into the text at the Point.

Of course, the manner of giving commands numeric arguments is itself a command! The C-U command specifies the numeric argument for the command which follows it. For example, if we wished to give the argument of 100 to the C-F command, we would type the C-U, then the number 100, then the C-F command. The number typed will be displayed in the echo line as it is being entered. Only positive numbers or zero may be entered; negative arguments do not exist.

3.4.2 String Arguments

Some commands will explicitly ask for character strings to be used as parameters for their execution. Examples of these are the string search commands and the file reading or writing commands. If such a string is needed a prompt will appear at

the bottom of the screen, in the echo area. The character string to be supplied to the command as an argument should be typed in.

Part of the echo-line display will be the prompt itself, which will describe what the string to be given to the command will be used for. Following this prompt will be a string specifying the termination character, either a <CR> or <ESC>. This character, when typed, will signal the end of the string argument and allow the command to process it.

If any typing mistakes occur while entering this string, the <DEL> key will erase them, as it does in ordinary text. However, the rest of the editing commands will just insert themselves into the string argument, rather than editing it. This means that the command which usually quotes the command characters to insert them into the text (the C-Q command) is almost never necessary when entering a string argument. However, if the termination character is to be inserted into the string argument, it must be quoted.

The C-G command will abort execution of the command which is asking for the string argument. Naturally, it will thus terminate the string entry as well. If a C-G must be entered into the string argument, it too can be quoted as normal.

The <CR> key has a different effect than usual when typed as part of a string argument. It will display as "<NL>" rather than going to a new line. This both assures you that you have indeed inserted a Newline into your string (for example, to search for the end of a line followed by a word) and prevents the cursor from going off the screen.

Typing the termination character without typing any string first (called entering a "null string" in the Command List) causes the command to use an appropriate default for the character string. For example, the string search commands will use whatever string was last searched for, the file I/O commands will use whatever file name was last used in the text buffer, and the buffer commands will use whatever buffer name was last switched from.

3.4.3 Yes/No arguments

Some commands will ask Yes/No questions to get information. These questions, like the prompts for string arguments, will appear in the echo line, but they will not have any string termination character specified. Typing a "Y" or "y" or a space will answer yes to the question, and typing "N" or "n" or the <DEL> key will answer no. Typing C-G will abort the command which asking the question; this is equivalent to the no answer to the question. The yes/no questions are usually asked if it is possible that some user command is about to destroy an entire buffer of text or make other large, irreversible

Section 4: Input/Output

4.1 Terminal I/O

Terminal input/output is fairly straightforward. Mince has several parameters which tell it how to do appropriate cursor positioning, character insertion, and line erasing on most standard terminals. The configuration program sets these up during your system installation. Character-at-a-time input is done from the keyboard; typeahead is automatically available, even during screen redisplay. Note that typeahead DOES NOT work during the other two forms of I/O. This is due to using the particular small computer (8080/Z80) and operating system which Mince is running on. Versions of Mince for slightly larger machines (e.g. PDP-11 or VAX) or other operating systems (e.g. UNIX or its look-alikes) do not have this restriction.

4.2 Text Buffer I/O

Text buffer I/O occurs because Mince implements a virtual memory system for storing text. Text files which are many times larger than available free memory space may be edited. Mince maintains a page-swapping file (called "MINCE.SWP") which it uses to page sections of text in and out of main memory on a least-recently-used basis. This paging affects the user in three ways:

(1) Typing on the terminal keyboard is ignored during paging.
(2) Screen redisplay may occasionally be interrupted for a short period if a page not in main memory is referenced.
(3) If the keyboard remains idle for a certain length of time (settable in the configuration program) Mince will write pages which have been modified in main memory back to the disc swap file, in order to make swapping a little faster when the terminal keyboard is in use again. Typing at the keyboard causes this "housecleaning" operation to cease.

A warning message tells you that a text buffer I/O interruption is about to occur. The message "Swapping..." will appear in the error message area of the echo line.

It is possible to get the error message "Swap File Full" to appear on the mode line during file read operations, large text deletions, or during text entry (although the last is highly unlikely). This means that the total of all the text which Mince is storing is larger than the swapping file size, and that the file read or text entry operation did not

successfully complete because it was unable to find more room to create new pages for text storage. This condition is not fatal; editing may still be continued, although the operation which caused the swap file overflow almost certainly did not complete properly. There are two solutions for this condition: either remove some of the text buffers (via the "C-X K" command) which Mince is using and are no longer needed (if there is more than one) (you can, of course, write them out since the editor will still operate) or exit the editor and increase the swap file size using the configuration program. (It is, of course, possible to gain space in the swap file by deleting some of the text in one of the buffers rather than removing the buffer entirely, but this alternative is rarely used.)

4.3 File System I/O

File system I/O occurs only when it is requested. As with buffer I/O, terminal keyboard input is ignored during disc access. As this is not unexpected I/O, as is buffer page swapping, this limitation is not very important. File system names for the CP/M versions of Mince are of the following form:

X:FIRSTNAM.LST

"X:" is the optional disc drive name. "FIRSTNAM" is the one-through-eight character first component of the file name. ".LST" is the optional one-through-three character second component of the file name. (This second name must be preceded by a period if it is used.) It is probably wisest to use alphanumeric filenames, as different conventions are used when different operating system versions and programs try to parse these names later. When a file name is not specified to the file I/O commands, the file name which was associated with the text buffer in which the file read command was given (as shown at the right edge of the mode line) is used.

Section 5: Text Buffers

5.1 What is a Buffer?

Mince does not edit text directly on the files in the file system. Instead, it copies the text from whatever file you wish to edit into a "buffer" of text. Changes are made to this buffer, and are then saved back into the file if you ask for it, by giving the "write file" command.

5.2 Why Use Buffers?

This method has quite a few advantages. First, if you decide that the changes you have been making are not quite the right thing to do, you can exit the editor without having damaged your original file. Secondly, if the computer crashes while you are editing, your file will not be left open with incorrect information in the middle of it. Thirdly, you can read data from one file and write it out to a different one, leaving the original text file intact. Finally, you can have several buffers of text active at once while running Mince. This capability allows you to edit one document based upon information contained in another without constantly entering and exiting the editor. Additionally, with the use of the kill buffer (explained in Section 5.4 below), text may be moved from one document to another as well.

There are commands which allow you to switch from buffer to buffer, automatically create new buffers and select names for them while reading in new files. There is also a command to delete buffers, if necessary. Finally, there is a command to list all the buffers which currently exist, should you forget what files you have and have not read in to be edited.

5.3 What Other Information the Buffer Contains

In addition to the text in each buffer, several other pieces of information are stored with the buffer. A file name is associated with each buffer. It is the name of the file which has last been read from or written to while in this buffer. The default for this name (for example, if new text is typed into a buffer in which no read or write file commands have been issued) is "DELETE.ME". A Point and a Mark are also associated with each buffer. This means that when switching back and forth between buffers, the Mark and the Point in that buffer will not have moved, regardless of where you have moved any other buffers' Points or Marks, and that the display will

reappear in the same state it was when you left the buffer. Each buffer also has a "mode" associated with it. Usually this mode is "Normal" mode, but it can be changed. Modes are described in Section 6.

5.4 The Kill Buffer

There is one special buffer, called the "kill buffer". This buffer is used to save any deleted text which is deleted in clumps of more than one character at a time. For example, if you delete a line or two, they will be saved in this kill buffer. There is a command to retrieve what is saved in the kill buffer, so that if you make any large mistakes in deleting, you can undo what you have done. This feature is also used to move text from one place to another, as it acts as a temporary storage buffer.

The kill buffer stores only text deleted by commands which delete words, sentences, lines, or regions. The commands which delete characters are not considered "dangerous" enough to make mistakes with, nor good vehicles for deleting large quantities of text. Also, the kill buffer tries to store text in the proper order when deletions occur. Thus, deleting words backwards and deleting words forwards will append text to the correct end of the kill buffer, such that, if the text is yanked back out of the kill buffer again, all the words will be in the proper order.

The kill buffer will store and merge together consecutive text deletions. This means that if several line-kills are done in a row, they are all merged together. (You may have noticed that this deletion-merging was implied above by the discussion of word deletion order.) One kill buffer retrieval command will yank all the text which is in the kill buffer back at once. The kill buffer will only store one set of deletions at once, however. The first kill-buffer-saving text deletion which is done "opens up" the kill buffer. Thereafter, any successive text deletion commands given which save the deleted text in the kill buffer will merge it with whatever is already in the buffer. If any non-deletion commands are given thereafter, the kill buffer is "closed off". Following kill-buffer-saving text deletion commands will throw away whatever text is then in the kill buffer and open up a new set of killed text in the kill buffer. This behavior is reflected on the screen display. The plus-sign on the right side of the mode line indicates whether or not any text to be killed will be merged onto the kill buffer or not. If the plus is turned on, the text deleted will be added to what is already in the kill buffer. If it is off, the new deleted text will start a new kill buffer (and turn the plus-sign on again until a non-deletion command is given). There is a command which does nothing but "turn on the plus sign", so that text may, if desired, be deleted from several places and yet still merged together onto the kill buffer so that a single kill-buffer-retrieval command will yank it all

back.

The kill buffer is the mechanism used to copy or move text from one part of a document to another. A region of text is deleted (which saves it in the kill buffer). If it is to be copied, it is yanked back out of the kill buffer again, at the same spot, so that the text has not "really" been deleted. The copy of the deleted text is still stored in the kill buffer, however. Thus, to make another copy of it, the retrieval command can be used to yank it out again and again. Thus, it can be copied at any other place in the buffer, merely by issuing the command again. If the text is to be moved rather than copied, the text is merely not yanked back the first time at its original position, but only at the desired new position in the document. The kill buffer is constant across all buffers. Text which is deleted and saved in the kill buffer while editing one buffer can be yanked back into a different buffer. Thus, pieces of text can be moved from file to file by using multiple buffers, one to hold each file used in the move or copying operation, and by using the kill-buffer-saving text deletion commands and the buffer-retrieval yanking command.

Section 6: Mince Modes

Mince's "modes" are used to implement fundamentally differing strategies for interaction with text in the buffer. In general, if a large group of commands is to be changed in some consistent fashion or if entering text is to have a new meaning, the changes are combined into a mode.

6.1 Normal Mode

Mince begins execution in Normal Mode. This mode causes all ordinary textual characters typed to be inserted into the buffer at the Point. Other text in the buffer is shifted over to make room for the new characters entered. Normal Mode is the "starting point" for the Mince command set: Other modes may be added (on a per-buffer basis), but those modes only add functions or rebind commands to keys; unless the documentation for a mode specifically mentions Normal Mode commands which have been deleted, all commands which have not been assigned new functions are left with their Normal Mode meaning.

6.2 Fill Mode

Fill mode was designed to incrementally fill paragraphs. The auto-filling action makes text entry neater and prevents having to look at the display to keep track of margins. In Fill Mode, the space key's command checks to see if the previous word typed extends past the preset "fill column" (the same column used by the "Fill Paragraph" command), and if so, automatically inserts a Newline before the word. Then a space is entered into the text, as usual. This behavior means that typists need only hit the carriage-return key when they mean to enter blank lines or intentionally break text to begin on a new line. In order to insert a space past the fill column setting, the space command must be quoted (by using the C-Q command).

All other Normal Mode commands remain unchanged by Fill Mode. In particular, the Fill Paragraph command still works, and text entry and deletion is the same as usual. Thus, while the space command performs the auto-fill once, it does not keep the paragraph filled. Going back and inserting or deleting text can make the right margin more ragged again.

Note that this mode, like the Fill Paragraph command, does not justify the right margin by inserting spaces in the line. It leaves the right margin ragged, but fits as many words

as possible on a line, then automatically inserts a Newline.

6.3 Page Mode

Page Mode was implemented in Mince primarily as an aid for those who have had previous experience with other types of screen-oriented editors. In Page Mode, text insertion and movement commands treat the text buffer as if it were a two-dimensional grid of characters the width of the terminal screen (a "page" of text). This differs from Normal Mode, in which text is treated as a one-dimensional string of characters in which the Newline characters define lines of differing lengths.

6.3.1 Page Mode text entry

Unlike Normal Mode, text insertion while in Page Mode overwrites previously existing text. In order to actually insert characters without overwriting the character in the same position on the screen, the character must be quoted (via the C-Q command). Consistent with this, the <BS> key, rather than merely deleting the previous character, backs up and overwrites the character with a space. (Note that C-B does not overwrite, and <DEL> actually removes the previous character from the line.) The carriage-return key inserts a Newline character and moves to the next line, exactly as it does in Normal Mode.

Frequently, you may desire to enter text in the middle of a line as in Normal Mode (i.e. insert rather than overwrite) for a short period of time without all the keystrokes involved in switching from Page Mode to Normal Mode and then back again or the overhead of quoting each character to be inserted. The easiest way to do this is to position the point in the buffer as you normally would, type the C-O command to split the right half of the line to a new one, type the text, then type a C-D command to delete the extra Newline and bring the right-hand part of the text back to the current line.

6.3.2 Page Mode line lengths

More differences from Normal Mode arise in the way Page Mode handles the end of a line and motion around Newline characters. In Page Mode, the horizontal character movement commands do not go past Newlines, as they do in Normal Mode. Instead, if the Forward Character command is given when the Point is just in front of a Newline, the line is extended by a character. On the screen, this means that the cursor will keep moving to the right on the same line, regardless of where the end of the text or the Newline was.

Similarly, the vertical line movement commands do not obey the same rules as in Normal Mode. In Normal Mode, if the Previous Line command (for example) is given at the end of a very long line and moves to a shorter line, the cursor will

move to the left on the screen, so as to be at the end of the shorter line. There was, after all, no text on the shorter line that far over. In Page Mode, however, the cursor will move directly up one line, not changing horizontal position. The short line would have been extended to the right in order to allow the cursor to rest in that position on the screen. This feature is very useful for adding or modifying columns of data as opposed to paragraphs of text.

Unfortunately, this uniform policy of extending lines whenever the cursor is positioned to a farther right column leads to some text storage inefficiencies. When writing the text buffer out to a file, any lines which were extended with spaces will not be trimmed back to the last nonblank character. Thus, as much of the page as the user has accessed by moving the Point will be stored when the text is written out. To counteract this waste, a command was included in Page Mode which deletes trailing whitespace on each line in the buffer (the C-X \ command).

These idiosyncracies are implemented in Page Mode to allow the user to consider the screen as a page of text, with screen-sized line boundaries and a consistent line length rather than lines which extend only as far as they were originally typed. Some other commands have been modified to give suitable effects for this mode. For example, the Beginning and End of Line commands have been changed to go to the first and last, respectively, nonblank characters on a line. See the Mince Command List for a description of the particular commands and their new properties.

Section 7: Two Windows

The Mince window system is a facility which augments the use of multiple buffers. By allowing the single screen window to be split into two separate windows, not only can many buffers of text be in use during one Mince session, but more than one of them may be displayed on the screen at once. This capability makes it easier than ever to modify one document based upon another or to move text from one buffer to another.

7.1 Creating a Second Window

The text window, which displays part of the document being edited, normally occupies almost all of the screen. When a second window is created (via the Two Windows command, "C-X 2"), each window occupies a little less than half of the screen. The windows are separated by a row of dashes the width of the screen.

It is important to distinguish between windows and buffers when using the system. A window may occupy all or part of the screen, and it merely shows a portion of what is in the text buffer currently being edited in it. The buffer stores the text, and the window displays the buffer. Thus, when the second window is initially created, since it displays the same buffer which was in the original (full-screen) window, the screen displays two copies of the same text, separated by a line of dashes.

7.2 Editing in a Window

Any editing requests affect the text buffer which is displayed in the window which has the cursor in it (the "current window"). This is just the same as when only a single window is displayed -- editing occurs at the cursor, because the cursor is attached to the Point in the buffer being displayed. Usually, different buffers are displayed in each of the windows, since a duplicate display is not too useful. (It is possible to display different parts of the same buffer in each window, however.) The buffer commands work for each window just as they did with the single window; the Select Buffer command (C-X B) may be used to show a different buffer in the current window. When switching from one window to the other (via the Other Window command, "C-X O"), the mode line will change to reflect the mode, file, and buffer names of whatever buffer is being displayed in the current window.

Using two windows does not affect the operation of the buffer system, since the Other Window command automatically switches to whatever buffer is displayed in the window. In particular, the window system does not affect the Kill Buffer. This makes it very easy to copy text (paragraphs, subroutines, whatever...) from one buffer to another. If the buffer into which the text is to be copied is displayed in one window, and the buffer from which text is to be taken is displayed in the other, there is immediate visual feedback on the successful completion of the Yank Killed Text command (C-Y).

7.3 Manipulating the Windows

If the number of lines displayed in one of the windows is too small, it can be increased by using the Grow Window command (C-X ^). Note that growing the current window shrinks the other window, however. A window may not be shrunk to display fewer than three lines; therefore, the other window can only be grown to a certain size.

Frequently, it is useful to scroll the text in the other window while making changes to the text in the current window. (Modifying source code based upon a compiler error listing file is a good example.) Going to the other window, scrolling it, and returning to the first window is rather tedious. Therefore, the View Next/Previous Screen Other Window commands (C-X C-V and C-X C-Z) allow you to view the next or previous screenful of text in the buffer displayed in the other window.

Finally, note that the two-window display is not permanent. The One Window command (C-X 1) causes the current window to become the only window and thus grow to occupy the entire display area again.

Command Cross-Reference Index

Copy	
Region	M-W
see also Wipe	
see also Yank	
Delete	
Buffer	C-X K
Character	C-D, <DEL>
Line (Kill)	C-K, M-C-K
Mode	C-X C-M
Region (Wipe)	C-W
Sentence	M-K
Whitespace	M-\
Word	M-D, M-<DEL>
see also Yank	
Display	
see Screen	
see also Buffer	
see also Windows	
Down	
see Forward	
End	
Buffer	M->
Line	C-E
see also Forward	
Exit	C-X C-C
Files	
Find	C-X C-F
Read	C-X C-R
Save	C-X C-S
Write	C-X C-W
Fill	
Paragraph	M-Q
Find	
see Search	
see also File	
Forward	
Character	C-F
Line	C-N
Page	C-V
Paragraph	M-]
Screen	C-V
Screen, Other Window	C-X C-V
Sentence	M-E
Word	M-F

Go To
see Buffer
see also Move
see also Other Window

Indent
see Center Line
see also Margins
see also Tabs
Newline and Auto-Indent C-J

Insert
see beginning of Commands List
see also Quote

Kill
see Delete

Line

Backward, Previous	C-P
Beginning	C-A
Center	M-S
Delete, Kill	C-K, M-C-K
Delete Whitespace on	M-\
End	C-E
Forward, Next	C-N

Lowercase

Word	M-L

Margins

Set Fill Column	C-X F
Set Indent Column	C-X .

Mark

Exchange Point and	C-X C-X
Set to Point	C-@ or M-<SPACE>
Whole Paragraph	M-H

Modes

Add	C-X M
Delete	C-X C-M

Move
see Backward
see also Beginning
see also Copy
see also End
see also Forward
see also Kill
see also Yank

Next
see Forward

Search	
Forward Search	C-S
Reverse Search	C-R
see also Replace	
Sentence	
Backward, Beginning	M-A
Delete, Kill	M-K
Forward, End	M-E
Set	
see Margins	
see also Mark	
see also Tabs	
Tabs	
Insert	<TAB> or C-I
Set	C-X <TAB>
Transpose	C-T
Undelete	
see Yank	
Universal	
see Argument	
Up	
see Backward	
Uppercase	
Word	M-U
Windows	
Grow	C-X ^
One	C-X 1
Other	C-X O
Two	C-X 2
View Next Screen Other	C-X C-V
View Previous Screen Other	C-X C-Z
Wipe	
see Delete	
Word	
Backward	M-B
Capitalize	M-C
Delete	M-D, M-<DEL>
Forward	M-F
Lowercase	M-L
Uppercase	M-U
Write	

The Mince Command List

All printing characters: a-z, A-Z, 0-9, space, and !"#$%&'()*+,-./:;<=>?@[]^_{|}~`
Self-insert
These characters are commands which insert themselves into the buffer. The characters are inserted at the Point, that is, they are inserted just in front of where the cursor is on the screen. The Point is then left after the new character. This means that on the screen, the cursor will move over one character position. Since characters are inserted at the Point, if the Point happens to be in the middle of some line of text, the rest of the line is moved over to make room for the new text. If any of these characters is given a numeric argument, that number of them is inserted into the buffer at the Point.

<LF> see C-J
(The linefeed key sends ASCII ^J, decimal 10.)

<CR> Newline Insert
This character is self-inserting as well, but causes the cursor to go the beginning of the next line on the screen. If a <CR> is typed when the Point is in the middle of a line, the line is split in two, with the portion of the line after the Point being moved down and turned into the next line. This is a little non-intuitive, but if you consider the Newline to be a character just like the other self-inserting ones, it makes sense. It inserts a Newline character at the Point, and moves the Point past the Newline character.

<TAB> Tab Insert
This character is also self-inserting (It inserts an ASCII ^I (decimal 9).), but causes the cursor to move over to the next tab stop. (To set tab increments for the screen display, see the "C-X <TAB>" command.)

<DEL> Delete Character Backward
Typing the delete key causes the last character typed to be removed from the text. Actually, what happens is that the character before the Point is deleted, so that the delete key, if used when the cursor is somewhere in a block of text, will always delete the character which is just to the left of the cursor. Since a Newline is treated as an ordinary character, typing a <DEL> at the beginning of a line causes the current line and the previous line to be joined. A <DEL> typed at the beginning of the buffer has no effect. This command does not save the deleted text

in the kill buffer.

<BS> Delete Character Backward
The backspace key is equivalent to the delete key, as described above.

<ESC> Meta-command Prefix
Typing the escape key causes the following character to be treated as a Meta-command. If sufficient time elapses after typing the <ESC> and the following character, the message "Meta:" will appear in the echo line to indicate that Mince is waiting for a character to make up one of the Meta-commands (or the Meta-Control-commands). The Meta-commands are explained in the second section below. A C-G will cancel the Meta prefix and leave the text unchanged.

C-@ Set Mark
This command sets the invisible Mark to the position where the Point is currently. On some terminals, You may have to type a Control-<SPACE> to get a C-@ command (it sends ASCII Null (^@), decimal 0). Others may be physically unable to generate this character. In that case, use the M-<SPACE> command instead. The message "Mark Set" is displayed in the echo line. (Remember this command because it sets the mark AT the Point.)

C-A Beginning of Line
This command moves the Point to just after the first Newline character preceding the Point. This has the effect of moving the cursor to the first character on the current line. Thus, repeated C-A's leave the cursor on the same line at the left edge of the screen. (Remember this command by either being at the beginning of the alphabet -> beginning of line, or by being at the left-hand edge of your keyboard -> left-hand edge of the screen.)

C-B Backward Character
This command moves the Point backward a character in the buffer. At the beginning of the buffer, C-B has no effect. Given a numeric argument, C-B moves back that many characters. Since Newline and <TAB> are treated as single characters, C-B skips over them as well. This means that a C-B issued at the beginning of a line will move the cursor to just after the last character on the previous line.

C-D Delete Character Forward
This command deletes the character which is after the Point. In other words, this deletes the character which the cursor is on. If the Point is before a Newline, the Newline is deleted; on the screen, if the cursor is after the last character on a line, a C-D causes the next line to be joined to the current line. Given a numeric argument, C-D deletes that many characters. A C-D typed at the end of the buffer has no effect. Note that the C-D command does <u>not</u> save the text it deletes in the kill buffer.

C-E End of Line
This command moves the Point to just before the first Newline character following the Point. This has the effect of moving the cursor to the last character on the current line. Thus, repeated C-E's leave the cursor on the same line at the right edge of the screen.

C-F Forward Character
This command moves the Point forward a character in the buffer. At the end of the buffer, C-F has no effect. With an argument, C-F moves forward that many characters. Since Newline and <TAB> are treated as single characters, C-F

skips over each of them as well. This means that a C-F issued at the end of a line will move the cursor to the first character on the next line.

C-G Abort/Cancel Prefix
This command aborts out of any other command which is requesting a string argument. It will also nullify any prefix characters typed to make Meta-commands or C-X commands. In other words, C-X C-G does nothing and M-C-G does nothing. C-G ignores any numeric argument it gets; therefore, this is a way to cancel numeric arguments as well. C-G also rings the terminal bell. (Think of this one as "beeping out" of command prefixes.)

C-H see <DEL>
(This is the backspace key (ASCII backspace, decimal 8) on most terminals.)

C-I see <TAB>
(<TAB> sends ASCII ^I (decimal 9) on most terminals.)

C-J Newline and Indent
This command inserts a Newline character then inserts enough whitespace (tabs and spaces) to move the cursor horizontally so that the next character typed on the new line is in the same column as the first non-whitespace character on the previous line. This is frequently useful when writing programs in a block-structured computer language. It is functionally equivalent to a a <CR> and enough tabs and spaces to position the cursor properly.

C-K Kill Line
This command deletes text from the Point to the following Newline, unless the Point is just in front of a Newline, in which case it deletes the Newline itself. A positive argument to C-K repeats the command that many times; a zero argument to C-K deletes from the Point to the beginning of the current line. This means that a single C-K at the beginning of a line will "clear" that line, and typing it again will actually remove the Newline character and move all the others on the screen up a line. The C-K command stores text which it deletes in the kill buffer.

C-L Redisplay Screen
A C-L causes a Mince screen redisplay to occur. The current line will be centered on the screen (or placed on another screen line selected while running the configuration program). (Remember this one because ^L is the ASCII character for "form feed" (decimal 12), interpreted as page feed on printers and screen clear on some display terminals.)

C-M see <CR>
(The carriage-return key sends ASCII ^M, decimal 13.)

C-N Next Line
This command moves the Point to the next line in the buffer. This has the effect of moveing the cursor to the next line on the screen. It tries to maintain the same horizontal position as it had when vertical movement was begun, that is, C-N tries to move the cursor to the spot on the screen directly below it. If the line is too short, it moves the cursor to the rightmost position on that line (i.e., just before the Newline character). Repeated use of C-N (or C-P, the Previous Line vertical motion command) uses the original horizontal position. Given a numeric argument, C-N moves down that many lines. At the end of the buffer, C-N has no effect.

C-O Open Line
This command inserts a Newline character but leaves the Point in front of the inserted character rather than after it, as is the case with <CR>. With a numeric argument, C-O inserts that many lines. This command is functionally equivalent to a <CR> followed by a C-B. It is primarily useful for splitting a line in half and inserting text immediately at the end of the first of the two resulting lines.

C-P Previous Line
This command moves the Point to the previous line in the buffer. This has the effect of moving the cursor to the previous line on the screen. It tries to maintain the same horizontal position as it had when vertical movement was begun, that is, C-P tries to move the cursor to the spot on the screen directly above it. If the line is too short, it moves the cursor to the rightmost position on that line (i.e., just before the Newline character). Repeated use of C-P (or C-N, the Next Line vertical motion command) uses the original horizontal position. Given a numeric argument, C-P moves up that many lines. At the beginning of the buffer, C-P has no effect.

C-Q Quote Next Character
This command is used to insert special characters into the text buffer which might otherwise be interpreted as Mince commands. It is also used to insert a string argument termination character into the string argument. If sufficient time elapses after typing the C-Q and the following character, the message "Quote:" will appear in the echo line to indicate that the C-Q command has been typed and is waiting for the next character. If C-Q is used while entering characters for a string argument to another command rather than while inserting text, the echo line is already in use and this message will not appear. The C-G command cannot abort out of a C-Q command; if typed, the C-G will be inserted into the buffer or string argument. Given a numeric argument, the C-Q command will

insert the following character into the buffer that many times, just as do the self-inserting (ordinary textual) characters.

C-R Reverse String Search
This command is very similar to the Forward String Search command, C-S, which is explained below. The string prompt message is "Reverse Search <ESC>:" instead, and the Point is left BEFORE the string if it is found between the current Point and the beginning of the buffer.

C-S Forward String Search
This command is used to search for particular strings of text. It displays the message "Forward Search <ESC>:" in the echo line of the screen display and awaits a string argument to search for as a string argument. If no string is entered (i.e., if the escape key is typed immediately after the C-S), whatever string was last given to a C-S or C-R string search command is used. Thus, repeated searches for the same string do not require retyping it each time. The string search is performed from the Point to the end of the buffer. If the string is found, the Point is left just after it. If not, the message "Not Found" is displayed in the error area of the echo line and the Point is left in its original position. The search is partially case-independent; that is, a lower case letter in the search string will match either a lower or an upper case letter in the buffer; however, an upper case letter will match only the upper case letter in the buffer. Given a numeric argument, the search is performed that many times. If the string is found that many times, the Point is left after that occurrence of the string. If not, the Point is not moved, and the "Not Found" message is displayed.

C-T Transpose Characters
This command transposes the characters before and after the Point (i.e., switches the character which the cursor is on with the one before it). It leaves the Point after the second one (i.e., moves the cursor to the character after the two just switched). Thus, successive C-T's will "drag" the previous character toward the end of the line. If the Point is at the end of a line, the two characters before the Point are transposed. This different behavior is useful when typing in new text; transposition typographical errors can be undone, since text insertion leaves the Point after the character just inserted. At the beginning of the buffer, the two characters following the Point are transposed since there is no character before the Point. Given a numeric argument, C-T will repeat whatever operation it would normally have done that many times. For example, a very large repeat count given to C-T in the middle of a line would first drag the character before the Point to the end of the line, then continuously transpose the last two characters on the line until the

repeat count was depleted.

C-U Universal Argument
This command is used to give numeric arguments to other commands. A number may be typed after typing C-U. If so, this number is the argument which is given to the next command typed. If not, the number 4 is automatically used. If sufficient time elapses after typing the C-U and any following character, the message "Arg: 4" will appear in the echo line to indicate that the C-U command may be given a typed number. If numbers are typed after the C-U, the "4" will be repaced with the number given. Note that <DEL> will not remove the last digit typed to the C-U command; rather, the sequence of C-U followed by any number and a <DEL> will cause repeated character deletion backwards. C-U's are multiplicative, that is, two C-U's typed in a row will cause the numeric argument given to the next command typed to be 16 rather than 4, and "C-U 7 C-U C-F" will move the Point forward 7x4=28 characters. If the C-U command is not used, the argument to any command will automatically be 1 instead. C-U may not be used to enter negative number arguments ("C-U -3" will enter "----3" into the text!) but may be used to enter zero as an argument (for example, "C-U 0 C-K").

C-V View Next Screen
This command moves the window so that it views the next screenful of text in the buffer. Incidental to this, the Point is moved down in the text as well, since the cursor always appears in the window display. There is an overlap of a few lines at the top and bottom of the screen so that repeated C-V's will have some continuity from screenful to screenful of text. Given a numeric argument, C-V will move the window down that many screens of text.

C-W Wipe Region
This command deletes ("wipes") the region of text between the Mark (see the C-@ command) and the Point. Wiped text is saved in the kill buffer.

C-X C-X Command Prefix
The next character typed is interpreted as one of the two-character C-X commands. See the list below. If sufficient time elapses after typing the C-X and any following character, the message "Control-X:" will appear in the echo line to indicate that the C-X command is waiting for another character to complete the command.

C-Y Yank Killed Text
This command inserts the contents of the kill buffer at the Point. It does not destroy the kill buffer, so that several C-Y's may be done to get several copies of the previously killed text. Given a numeric argument, C-Y yanks back the killed text that many times.

C-[see <ESC>
(The escape key generates the ASCII ^[(decimal 27) character.)

C-\ Delete Indentation
This command deletes the leading whitespace on the current line. It does not move the Point from its current position, however. This command does not save the whitespace it deletes in the kill buffer.

C-X <TAB> Set Tab Spacing

This command sets the tab increments for the display screen. These affect how far each tab character indents the following text. This command may be used in one of two ways: Given a numeric argument, it sets the tab spacing to that number. Given no argument, it uses the column number which the Point is at as the argument. Thus the tab spacing may be set "by eye".

C-X C-B List Buffers

This command lists all the buffers created in this editing session. Each element of the list has the following form:

```
buf1 * 1234 X:FIRSTNAM.LST
```

where "buf1" is the buffer name, the asterisk indicates that the buffer has not been written out since it has been modified, "1234" is the length of the buffer in characters, and "X:FIRSTNAM.LST" is the name of the file which was either last read from or written to in this buffer "buf1". As many of these lines as there are buffers are displayed at the top of the screen. (The kill buffer is not displayed in this list, since it cannot be used in the same manner as the others.) This command is useful if you forget which buffer a certain file has been read into. (Note its similarity to the "C-X B" command.) In order to remove the temporary display at the top of the screen, type any command (for example, C-L or C-G may be used).

C-X C-C Exit to Command Level

This command exits Mince, returning the user to the operating system. (CP/M users may remember it by thinking of it as an augmented C-C, the "standard" program interrupt character.) If any buffers have been modified since being written out, Mince asks the yes/no question "Abandon Modified Buffer(s)?" in the echo line.

C-X C-F Find File

This command puts you in a buffer editing a particular file, regardless of whether you have already read it into Mince or not. It displays the prompt "Find File <CR>:" in the echo line and waits for you to type in a file name as a string argument. If that file name is associated with any buffer available in Mince, it effectively does a "C-X B" command to display and allows you to edit that buffer. If there is no such file in a Mince buffer at the moment, Mince tries to create a buffer whose name is the same as the first component of the file name. If such a buffer is already in use, Mince displays the message "Buffer Exists!" and asks "Buffer to Use <CR>:" in the echo line and waits for you to supply a different name for the buffer. If you choose to re-use the buffer whose name is the first component of the file name (i.e. the one which Mince had originally chosen), just enter a carriage-return and the previous contents of the buffer will be lost. If

you choose to use a different buffer, enter its name. In any case, the filename given will be read in to the buffer selected, just as if the "C-X C-R" command had been given. (See its description for yet more ramifications.) In general, this command is functionally equivalent to a C-X B command, possibly followed by a C-X C-R command.

C-X C-I (same as C-X <TAB>)

C-X C-M Delete Mode
This command is used to delete modes from the current buffer's mode list. (Think of it as the inverse of the "C-X M" command.) It displays the message "Delete Mode <CR>:" in the echo line and waits for a mode name as a string argument. If no such mode exists, the message "Unknown Mode" is displayed at the right of the mode line. If the mode exists but has not been added to the buffer's mode list, this command does nothing. If it does exist and is on the buffer's list, it is removed and the mode line is updated to reflect the change in modes.

C-X C-R Read File
This command is used to read the contents of a file into the current buffer. It displays the message "Read File <CR>:" in the echo line and waits for the user to supply the name of the file to be read as a string argument. If a null string is entered, the file name currently associated with the buffer (i.e., whatever file name was last used in a file read or write command or "DELETE.ME" if there had been none) is used. If no such file is found on disc the message "New File" appears in the echo line, and the buffer is made empty. Since the Read File command always overwrites the current contents of the buffer, it checks to see if it has been previously modified without being written out. If it has, the Read File command queries the user with the yes/no question Clobber modified buffer?" in the echo line. The Mark and the Point are set to the beginning of the text buffer when a file is read in.

C-X C-S Save File
This command is equivalent to typing "C-X C-W <CR>". See the Write File command description below.

C-X C-V View Next Screen Other Window
This command is used to scroll forward (or "down") the text in the window which the cursor is not currently in. Thus, it is functionally equivalent to the sequence "C-X O C-V C-X O". Given a numeric argument, the text is scrolled down that many times. (The actual number of lines passed depends upon the size of the other window.) If there is only one window being displayed, this command has no effect.

C-X C-W Write File

This command is used to write the contents of the current buffer to a disc file. It displays the message "File to Write <CR>:" in the echo line and waits for the user to supply a file name as a string argument. If the null string is entered, the file name currently associated with the buffer (i.e., whatever file name was last used in a file read or write command or "DELETE.ME" if there had been none) is used. If no file of the name given exists, one is created. If one exists, it is overwritten with the contents of the buffer.

C-X C-X Exchange Point and Mark
This command switches the Point and the invisible Mark in the current buffer. It is useful for determining the edges of a region which is about to be wiped with the C-W command. (Remember this command as standing for "eXchange".)

C-X C-Z View Previous Screen Other Window
This command is used to scroll backward (or "up") the text in the window which the cursor is not currently in. Thus, it is functionally equivalent to the sequence "C-X O M-V C-X O". Given a numeric argument, the text is scrolled up that many times. (The actual number of lines passed depends upon the size of the other window.) If there is only one window being displayed, this command has no effect.

C-X . Set Indent Column
This command is used to set the number of spaces which should be left blank at the left margin for the M-Q command, the M-S command, or while using Fill mode. Given a numeric argument, it sets the indent column to that number. If not given an argument, it sets the indent column to whatever column the Point is at in the text. The indent column is the first column in which text may appear when filling paragraphs or centering lines. (The fill column is the first column in which text may not appear.) Note that columns are numbered from zero. A default setting for the indent column (usually zero) is selected when the configuration program is run. This command displays "Indent Column is n" in the echo line.

C-X 1 One Window
This command makes whatever window in which editing is currently being done the only window on the screen. Effectively, this undoes the effect of the "C-X 2" command. If two windows are not being displayed on the screen, this command has no effect.

C-X 2 Two Windows
This command splits the display screen in half and turns the one window into two. This allows a buffer to be displayed in either the upper or lower window area (or

both). The window boundary is indicated by a line of dashes across the screen, separating the upper from the lower window. Initially, the second window will display the same buffer as the first window; that is, both windows will display the same buffer which was being edited when the Two Windows command was given. The cursor will be left in the upper window, and editing may be continued there. If there are already two windows on the screen, this command will have no effect.

C-X = Where Am I

This command displays (in the echo line) the location of the Point and Mark measured in characters from the beginning of the buffer and the size of the buffer measured in characters. It also displays the the Point's current horizontal position (column number).

C-X B Select Buffer

This command displays "Switch to Buffer <CR>:" in the echo line and waits for the user to supply in a buffer name to go to as a string argument. If the user picks a buffer name which is already in use, that buffer becomes the current buffer, it is displayed on the screen, and all subsequent editing operations are performed on it. If the user picks a new buffer name, Mince asks the yes/no question "Create New Buffer?". A no answer aborts the command execution, and a yes answer creates a new empty buffer. If no buffer name is supplied (i.e., the user enters a null string) whatever buffer was last switched from is switched to. Thus, repeated executions of the "C-X B" command cause the editor to switch back and forth between two buffers (once the "other" buffer is first set by actually giving C-X B a buffer name or by using the C-X C-F command, which automatically performs a C-X B).

C-X F Set Fill Column

This command is used to set the right margin used for filling text during the operation of the M-Q command or while using Fill mode or the line centering command, M-S. Given a numeric argument, it sets the fill column to that number. If not given an argument, it sets the fill column to whatever horizontal column the Point is at in the text. The fill column is the first column in which text may not appear when filling paragraphs or centering lines. (The indent column is the first column in which text <u>may</u> appear when filling paragraphs or centering lines.) Note that columns are numbered from zero. A default setting for the fill column (usually 65) is selected when the configuration program is run. This command displays "Fill Column is n" in the echo line.

C-X K Kill Buffer

This command is used to remove buffers from the Mince working set. It may be used to free up space in the swap

file if a "Swap File Full" error is encountered or to remove some of the visual (and mental) clutter which occurs with many buffers to keep track of. It displays "Delete Buffer <CR>:" in the echo line and waits for the user to supply the name of the buffer to be deleted as a string argument. If the null string is entered, the buffer name used is the last one switched from, as with the Select Buffer command, C-X B. If the buffer selected to be killed is the current buffer, Mince displays "Switch To Buffer <CR>:" in the echo line and waits for a new buffer name string argument. (Mince will not let you switch to the buffer you are about to delete, and displays an error message if you try to.) After performing the buffer switch (which is identical in function to the "C-X B" command), if any, Mince tries to delete the requested buffer. If it has not been written out since being modified, Mince asks the yes/no question "Delete Modified Buffer?" in the echo line.

C-X M Add Mode

This command is used to add modes to the current buffer's mode list. It displays the message "Mode Name <CR>:" in the echo line and waits for a mode name as a string argument. If there is no such mode, the error message "Unknown Mode" is displayed. Adding a mode to buffer in which it is already on the buffer's mode list has no effect.

C-X O Other Window

When two windows are displayed on the screen, this command switches from one window to the other. It automatically selects the buffer which is being displayed in the window being switched to. If only one window is being displayed on the screen, this command has no effect.

C-X ^ Grow Window

This command increases the number of lines used to display the window in which editing is currently being done. (Thus it decreases the number of screen lines available to display the other window.) Given a numeric argument, this command enlarges the window by that many lines. Windows cannot be smaller than three lines, thus a window cannot be grown such that it will force the other one to be smaller than that limit. If this command is given when only one window is being displayed on the screen, it has no effect. (Remember this command by thinking of the arrow "pushing" the window boundary up or down.)

M-<DEL> Delete Word Backward
This command deletes text backward until it finds the beginning of a word. This means that if the Point is in the middle of a word, the first part of the word will be deleted. If the Point is at the end of a word, the entire word will be deleted. If the Point is after a word, all the intervening characters between the Point and the last character of the word will be deleted as well. Deleted text is saved in the kill buffer. Given a numeric argument, M-<DEL> deletes that many words.

M-<SPACE> see C-@
(This command is implemented for those terminals physically unable to generate the character for the C-@ command.)

M-< Beginning of Buffer
This command moves the Point back to the beginning of the text buffer. Before doing this, it sets the Mark to the place where the Point currently is. This allows a C-X C-X command to get you back to where you started. (Remember this one because the less-than symbol Points in the direction you want to move.)

M-> End of Buffer
This command moves the Point to the end of the buffer. Before doing this, it sets the invisible Mark to the place where the Point currently is. This allows the C-X C-X command to get you back to where you started. (Remember this one because the greater-than symbol Points in the direction you want to move.)

M-A Backward Sentence
This command moves the Point to just before the sentence which it is currently in. If the Point is not in some sentence, it will be moved to the beginning of the previous sentence. Given a numeric argument, this command will move the Point backward that many sentences.

M-B Backward Word
This command moves the Point backward to just before the word which it is currently in. If the Point is not in some word, it will be moved to the beginning of the previous word. Given a numeric argument, this command will move backward that many words. Like all word commands, this one will skip over any intervening whitespace or punctuation while looking for the beginning of a word.

M-C Capitalize Word
This command capitalizes the current word. If the Point is in the middle of a word rather than in front of it, the letter which the cursor is on will be capitalized rather than the first letter of the word. Note that the numbers 0

through 9 are considered to be parts of words and have no capital form. The rest of the word is lowercased. This command leaves the Point just after the word which has been capitalized. Therefore, given a numeric argument, this command will capitalize and move forward past that many words. Like all word commands, this one will skip over any intervening whitespace or punctuation while looking for the beginning of a word.

M-D Delete Word Forward
This command deletes text until it finds the end of a word. This means that if the Point is in the middle of a word, the rest of the word will be deleted. If the Point is at the beginning of a word, the entire word will be deleted. If the Point is before a word, all the intervening characters before the word will be deleted as well. Deleted text is saved in the kill buffer. Given a numeric argument, M-D deletes that many words.

M-E Forward Sentence
This command moves the Point to the end of the sentence which it is currently in. If the Point is not in some sentence, it will be moved to the end of the following sentence. Given a numeric argument, this command will move the Point forward that many sentences.

M-F Forward Word
This command moves the Point forward to just after the word which it is currently in. If the Point is not in some word, it will be moved to the end of the next word. Given a numeric argument, this command will move forward that many words. Like all word commands, this one will skip over any intervening whitespace or punctuation while looking for the end of a word.

M-H Mark Whole Paragraph
This command Marks the paragraph which the Point is in (or which the Point is just before). It moves the Point to the beginning of the paragraph and sets the Mark to the end. This command is functionally equivalent to the sequence M-], C-@, then M-[. It is used for convenience with C-W and C-Y for copying or moving paragraphs. (Remember it by the "H" standing for "wHole".)

M-K Kill Sentence Forward
This command deletes text until it finds the end of a sentence. This means that if the Point is in the middle of a sentence, the rest of the sentence will be deleted. If the Point is at the beginning of a sentence, the entire sentence will be deleted. If the Point is before a sentence, all the intervening characters before the sentence will be deleted as well. Deleted text is saved in the kill buffer. Given a numeric argument, M-K deletes that many sentences. If the argument is zero, M-K deletes

backward from the Point to the beginning of the sentence.

M-L Lowercase Word
This command lowercases the current word. If the Point is in the middle of a word rather than in front of it, the letter which the cursor is on will be the first to be changed from upper to lower case rather than the first letter of the word. Note that the numbers 0 through 9 are considered to be parts of words and have no special lowercase form. This command leaves the Point just after the word which has been recased. Therefore, given a numeric argument, this command will lowercase and move forward past that many words. Like all word commands, this one will skip over any intervening whitespace or punctuation while looking for the beginning of a word.

M-Q Fill Paragraph
This command fills text in the current paragraph so that each line of text does not go past the right margin (set by the Set Fill Column command, "C-X F"), and so that each line after the first begins at the left margin (set by the Set Indent Column command, "C-X ."). Given a numeric argument, it sets the fill column to that number (i.e., executes an automatic "C-X F") and then performs the text filling operation. The text filling operation will move words between lines and insert or delete as many lines as are necessary to properly format the text.

M-R Replace String
This command replaces strings from the Point to the end of the buffer. It is functionally equivalent to the Query Replace (M-C-R) command, with the "!" option (to replace all following occurrences) specified. See the M-C-R command description for an explanation of the string arguments.

M-S Center Line
This command centers the current line, if possible, between the left margin (set by the Set Indent Column command, "C-X .") and the right margin (set by the Set Fill Column command, "C-X F"). If this is not possible, the line to be centered is left flush at the left edge of the screen. Given a numeric argument, it sets the fill column to that number (i.e., executes an automatic "C-X F") and then performs the centering operation. (Remember this command by the S-sound in "center".)

M-T Transpose Words
This command transposes the words before and after the Point. It leaves the Point after the second one. Thus, successive M-T's will "drag" the previous word toward the end of the line. If the Point is at the end of a line, the last word on the line is exchanged with the first word on the next line. If the Point is at the end of the buffer,

this command has no effect. If the Point is in the middle of a word, the two halves of the word are switched instead. Given a numeric argument, the word-switching is performed that many times.

M-U Uppercase Word
This command uppercases the current word. If the Point is in the middle of a word rather than in front of it, the letter which the cursor is on will be the first to be changed to upper case rather than the first letter of the word. Note that the numbers 0 through 9 are considered to be parts of words and have no special uppercase form. This command leaves the Point just after the word which has been recased. Therefore, given a numeric argument, this command will uppercase and move forward past that many words. Like all word commands, this one will skip over any intervening whitespace or punctuation while looking for the beginning of a word.

M-V View Previous Screen
This command moves the window so that it views the previous screenful of text in the buffer. Incidental to this, the cursor is moved backward in the text as well, since it must always appear in the window display. There is an overlap of a few lines at the top and bottom of the screen so that repeated M-V's will have some continuity from screenful to screenful of text. Given a numeric argument, M-V will move the window down that many screens of text.

M-W Copy Region
This command copies the region of text between the Mark (see the C-@ command) and the Point onto the kill buffer. (Remember this command by its relationship to the kill buffer and the C-W command.)

M-[Backward Paragraph
This command moves backward to the beginning of the current paragraph. If this command is given while the Point is not inside any paragraph, the Point will be moved to the beginning of the paragraph before the Point. Given a numeric argument, M-[will move the Point backward that many paragraphs.

M-\ Delete Surrounding Whitespace
This command deletes all spaces and tab characters on both sides of the Point. The whitespace deleted is not saved in the kill buffer.

M-] Forward Paragraph
This command moves forward to the end of the current paragraph. If this command is given while the Point is not inside any paragraph, the Point will be moved to the end of the paragraph after the Point. Given a numeric

M-C-H see M-<DEL>

M-C-K Kill Entire Line

This command deletes the entire current line. It is similar to the C-K command, except that it will kill any text from the beginning of the line to the Point as well, and will kill the Newline after killing the text on the line. The killed text is saved in the kill buffer.

M-C-R Query Replace String

This command does string replacement, asking for some sort of confirmation at each occurrence of the string to be replaced. The search/replace operation is performed starting at the Point, toward the end of the buffer. Old and new strings (the string to be repeatedly searched for and the string to replace it with) are requested as string arguments with the prompts "Query Replace <ESC>:" and "With <ESC>:". At each occurrence of the old string, the user has the following command choices:

C-G Abort

This character will cause the replacing operation to stop. The Point will be left where it was in the middle of the searching operation (i.e., just after the current occurrence of the old string).

! Replace Rest

This character will replace all the remaining occurrences of the old string with the new one, without stopping at each to ask.

. Exit

This character will cause the query-replace to stop without searching for any remaining occurrences of the old string. The Point will be left at the position where the query-replace command was given.

, Replace and Request Confirmation

This character will cause the replacement to occur, then ask a Yes/No question to determine whether or not to actually leave the old string replaced by the new. Thus, the user can see the results of a replacement and decide whether or not to keep it.

Y (or y or space) Replace and Find Next

This character causes the current occurrence of the old string to be replaced with the new and the next occurrence of the old string in the buffer to be found.

<anything else> Don't Replace, and Find Next

Any other character typed will leave the current occurrence of the old string unchanged and search for the next one.

M-C-W Append to Kill Buffer

This command causes the next group of text deleteion commands to append to the kill buffer if they otherwise would have started a new kill group. (This is effectively "turning on the plus sign" on the Mince screen display.)

(Remember this command by its relationship with the C-W command.)

Fill Mode Command List

space Auto Fill Space

This command inserts a space into the buffer and moves past the space. It also checks to see if the word just prior to the space extends past the right margin (set by the Set Fill Column command, C-X F), and if so, inserts a newline and inserts whitespace to the left margin (set by the Set Indent Column command, C-X .) before that word. Thus, a typist can type whole paragraphs without looking at the screen.

Page Mode Command List

All printing characters: a-z, A-Z, 0-9, space, and !"#$%&'()*+,-./:;<=>?@[]^_{|}~`
Self-overwrite
These characters are commands which overwrite the character at the Point with themselves. These characters will not overwrite a Newline character (instead, they extend the line), and if overwriting a tab character, will insert sufficient spaces to maintain the column position of the text following the tab.

C-A To First Non-White
This command moves the Point to just before the first non-whitespace character (i.e., non-space, non-tab) on the current line. This is similar to the Normal Mode C-A command, except that after moving back to just after the first preceding Newline, it moves forward until it encounters a non-white character in the line.

C-B Backward Character on Line
This command moves the Point backward a character in the buffer. It is similar to the Normal Mode C-B command, except that it will not skip over Newline characters (it always stays on the current line) and it treats tab characters as multiple spaces while moving rather than as single characters.

C-E To Last Non-White
This command moves the Point to just after the last non-whitespace character on the current line. This is similar to the Normal Mode C-E command, except that after moving forward to the first following Newline, it moves backward until it encounters a non-white character in the line.

C-F Forward Character on Line
This command moves the Point forward a character in the buffer. It is similar to the Normal Mode C-F command, except that it will not skip over Newline characters (it always stays on the current line) and that it treats tab characters as multiple spaces while moving rather than as single characters. Thus, moving to a Newline and typing C-F will extend the line by one more character to the right.

C-H Overwrite Character Backward
This command moves backward a character in the buffer and overwrites it with a space, leaving the Point before the

overwritten space. It is functionally equivalent to typing C-B <SPACE> C-B. (C-H is usually available by typing the <BS> key.)

C-N Next Line Forced
This command moves the Point to the next line in the buffer. It is similar to the Normal Mode C-N command, except that rather than trying to preserve the horizontal column position when moving from line to line, it forces the horizontal column position to be the same. Thus, if the line to be moved to has fewer columns than the current column position, it is extended the appropriate amount with whitespace before the cursor is moved. This has the effect of always moving the cursor directly vertically, rather than "hugging" the right margin of the text, as may occur in Normal Mode.

C-P Previous Line Forced
This command moves the Point to the previous line in the buffer. It is similar to the Normal Mode C-P command in all respects except those mentioned above for the C-N command.

C-Q Quote Next
This command "quotes" the next character typed; that is, it inserts the character literally into the buffer rather than performing the command associated with it. This command is identical in operation to the Normal Mode C-Q command. The only reason it is mentioned here is that when ordinary textual characters are quoted with C-Q, they are inserted into the buffer as in Normal Mode, rather than overwritten on the buffer as in Page Mode.

C-X \ Delete Trailing Whitespace
This command examines every line in the buffer and deletes trailing blanks from each line. This is useful to delete any trailing blanks which may have been created by the screen cursor positioning commands in Page Mode. (Remember this command by its similarity to M-\, the Delete Surrounding Whitespace command.)

The Mince Commands

All printing characters: a-z, A-Z, 0-9, space, and !"#$%&'()*+,-./:;<=>?@[]^_{|}~`

Self-insert

<LF> see C-J

<CR> Newline Insert

<TAB> Tab Insert

<DEL> Delete Character Backward

<ESC> Meta-command Prefix

C-@ Set Mark

C-A Beginning of Line

C-B Backward Character

C-D Delete Character Forward

C-E End of Line

C-F Forward Character

C-G Abort/Cancel Prefix

C-H see <DEL>

C-I see <TAB>

C-J Newline and Indent

C-K Kill Line

C-L Redisplay Screen

C-M see <CR>

C-N Next Line

C-O Open Line

C-P Previous Line

C-Q Quote Next Character

C-R Reverse String Search

C-S Forward String Search

C-T Transpose Characters

C-U Universal Argument

C-V View Next Screen

C-W Wipe Region

C-X C-X Command Prefix

C-Y Yank Killed Text

C-[see <ESC>

C-\ Delete Indentation

C-X <TAB> Set Tab Spacing

C-X C-B List Buffers

C-X C-C Exit to Command Level

C-X C-F Find File

C-X C-I (same as C-X <TAB>)

C-X C-M Delete Mode

C-X C-R Read File

C-X C-S Save File

C-X C-V View Next Screen Other Window

C-X C-W Write File

C-X C-X Exchange Point and Mark

C-X C-Z View Previous Screen Other Window

C-X . Set Indent Column

C-X = Where Am I

C-X 1 One Window

C-X 2 Two Windows

C-X B Select Buffer

C-X F Set Fill Column

C-X K Kill Buffer

C-X M Add Mode

C-X O Other Window

C-X ^ Grow Window

M-<DEL> Delete Word Backward

M-<SPACE> see C-@

M-< Beginning of Buffer

M-> End of Buffer

M-A Backward Sentence

M-B Backward Word

M-C Capitalize Word

M-D Delete Word Forward

M-E Forward Sentence

M-F Forward Word

M-H Mark Whole Paragraph

M-K Kill Sentence Forward

M-L Lowercase Word

M-Q Fill Paragraph

M-R Replace String

M-S Center Line

M-T Transpose Words

M-U Uppercase Word

M-V View Previous Screen

M-W Copy Region

M-[Backward Paragraph

M-\ Delete Surrounding Whitespace

M-] Forward Paragraph

M-C-H see M-<DEL>

M-C-K Kill Entire Line

M-C-R Query Replace String

M-C-W Append to Kill Buffer

Fill Mode Commands

space Auto Fill Space

Page Mode Commands

All printing characters: a-z, A-Z, 0-9, space, and !"#$%&'()*+,-./:;<=>?@[]^_{|}~`

Self-overwrite

C-A To First Non-White

C-B Backward Character on Line

C-E To Last Non-White

C-F Forward Character on Line

C-H Overwrite Character Backward

C-N Next Line Forced

C-P Previous Line Forced

C-Q Quote Next

C-X \ Delete Trailing Whitespace

USASCII Character Set

as modified for printing
and the inclusion of Meta characters

Decimal	Octal	Hex	Graphic	Name (Meaning) <English Text Reference>
0.	000	00	^@	NUL (used for padding) <NULL>
1.	001	01	^A	SOH (start of header)
2.	002	02	^B	STX (start of text)
3.	003	03	^C	ETX (end of text)
4.	004	04	^D	EOT (end of transmission)
5.	005	05	^E	ENQ (enquiry)
6.	006	06	^F	ACK (acknowledge)
7.	007	07	^G	BEL (bell or alarm) <BELL>
8.	010	08	^H	BS (backspace) <BS>
9.	011	09	^I	HT (horizontal tab) <TAB>
10.	012	0A	^J	LF (line feed) <LF>
11.	013	0B	^K	VT (vertical tab)
12.	014	0C	^L	FF (form feed, new page) <FF>
13.	015	0D	^M	CR (carriage return) <CR>
14.	016	0E	^N	SO (shift out)
15.	017	0F	^O	SI (shift in)
16.	020	10	^P	DLE (data link escape)
17.	021	11	^Q	DC1 (device control 1, XON)
18.	022	12	^R	DC2 (device control 2)
19.	023	13	^S	DC3 (device control 3, XOFF)
20.	024	14	^T	DC4 (device control 4)
21.	025	15	^U	NAK (negative acknowledge)
22.	026	16	^V	SYN (synchronous idle)
23.	027	17	^W	ETB (end transmission block)
24.	030	18	^X	CAN (cancel)
25.	031	19	^Y	EM (end of medium)
26.	032	1A	^Z	SUB (substitute)
27.	033	1B	^[	ESC (escape, alter mode, SEL) <ESC>
28.	034	1C	^\	FS (file separator)
29.	035	1D	^]	GS (group separator)
30.	036	1E	^^	RS (record separator)
31.	037	1F	^_	US (unit separator)
32.	040	20		space or blank <SP>
33.	041	21	!	exclamation mark
34.	042	22	"	double quote
35.	043	23	#	number sign (hash mark)
36.	044	24	$	dollar sign
37.	045	25	%	percent sign
38.	046	26	&	ampersand sign
39.	047	27	'	single quote (apostrophe)
40.	050	28	(	left parenthesis
41.	051	29	)	right parenthesis
42.	052	2A	*	asterisk (star)
43.	053	2B	+	plus sign
44.	054	2C	,	comma
45.	055	2D	-	minus sign (dash)
46.	056	2E	.	period (decimal point, dot)
47.	057	2F	/	(right) slash

48.	060	30	0	numeral zero
49.	061	31	1	numeral one
50.	062	32	2	numeral two
51.	063	33	3	numeral three
52.	064	34	4	numeral four
53.	065	35	5	numeral five
54.	066	36	6	numeral six
55.	067	37	7	numeral seven
56.	070	38	8	numeral eight
57.	071	39	9	numeral nine
58.	072	3A	:	colon
59.	073	3B	;	semi-colon
60.	074	3C	<	less-than sign
61.	075	3D	=	equal sign
62.	076	3E	>	greater-than sign
63.	077	3F	?	question mark
64.	100	40	@	atsign
65.	101	41	A	upper-case letter ALPHA
66.	102	42	B	upper-case letter BRAVO
67.	103	43	C	upper-case letter CHARLIE
68.	104	44	D	upper-case letter DELTA
69.	105	45	E	upper-case letter ECHO
70.	106	46	F	upper-case letter FOXTROT
71.	107	47	G	upper-case letter GOLF
72.	110	48	H	upper-case letter HOTEL
73.	111	49	I	upper-case letter INDIA
74.	112	4A	J	upper-case letter JERICHO
75.	113	4B	K	upper-case letter KAPPA
76.	114	4C	L	upper-case letter LIMA
77.	115	4D	M	upper-case letter MIKE
78.	116	4E	N	upper-case letter NOVEMBER
79.	117	4F	O	upper-case letter OSCAR
80.	120	50	P	upper-case letter PAPPA
81.	121	51	Q	upper-case letter QUEBEC
82.	122	52	R	upper-case letter ROMEO
83.	123	53	S	upper-case letter SIERRA
84.	124	54	T	upper-case letter TANGO
85.	125	55	U	upper-case letter UNICORN
86.	126	56	V	upper-case letter VICTOR
87.	127	57	W	upper-case letter WHISKEY
88.	130	58	X	upper-case letter XRAY
89.	131	59	Y	upper-case letter YANKEE
90.	132	5A	Z	upper-case letter ZEBRA
91.	133	5B	[	left square bracket
92.	134	5C	\	left slash (backslash)
93.	135	5D	]	right square bracket
94.	136	5E	^	uparrow (carat)
95.	137	5F	_	underscore

96.	140	60	`	(single) back quote (grave accent)
97.	141	61	a	lower-case letter alpha
98.	142	62	b	lower-case letter bravo
99.	143	63	c	lower-case letter charlie
100.	144	64	d	lower-case letter delta
101.	145	65	e	lower-case letter echo
102.	146	66	f	lower-case letter foxtrot
103.	147	67	g	lower-case letter golf
104.	150	68	h	lower-case letter hotel
105.	151	69	i	lower-case letter india
106.	152	6A	j	lower-case letter jericho
107.	153	6B	k	lower-case letter kappa
108.	154	6C	l	lower-case letter lima
109.	155	6D	m	lower-case letter mike
110.	156	6E	n	lower-case letter november
111.	157	6F	o	lower-case letter oscar
112.	160	70	p	lower-case letter pappa
113.	161	71	q	lower-case letter quebec
114.	162	72	r	lower-case letter romeo
115.	163	73	s	lower-case letter sierra
116.	164	74	t	lower-case letter tango
117.	165	75	u	lower-case letter unicorn
118.	166	76	v	lower-case letter victor
119.	167	77	w	lower-case letter whiskey
120.	170	78	x	lower-case letter xray
121.	171	79	y	lower-case letter yankee
122.	172	7A	z	lower-case letter zebra
123.	173	7B	{	left curly brace
124.	174	7C	|	vertical bar
125.	175	7D	}	right curly brace
126.	176	7E	~	tilde
127.	177	7F	^?	DEL (delete, rub out) <DEL>
128.	200	80	~^@	Meta NUL
129.	201	81	~^A	Meta SOH
130.	202	82	~^B	Meta STX
131.	203	83	~^C	Meta ETX
132.	204	84	~^D	Meta EOT
133.	205	85	~^E	Meta ENQ
134.	206	86	~^F	Meta ACK
135.	207	87	~^G	Meta BEL
...				
159.	237	9F	~^_	Meta US
160.	240	A0	~	Meta space
161.	241	A1	~!	Meta exclamation mark
...				
253.	375	FD	~}	Meta right curly brace
254.	376	FE	~~	Meta tilde
255.	377	FF	~^?	Meta DEL

Notes:

The "Meta" form of each character is created by adding 128 (decimal) to that character's ASCII value.

To prevent ambiguity, the following alternate forms can be used for printing:

94.	136	5E	^	can be printed as	^=
126.	176	7E	~	can be printed as	^~
222.	336	DE	~^	can be printed as	~^=
254.	376	FE	~~	can be printed as	~^~