SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 11, 467–477 (1981)

CHEF: a Versatile Portable Text Editor

M. A. MACLEAN
Computer Science Department, University of Canterbury, Christchurch 1, New Zealand

AND

J. E. L. PECK
Computer Science Department, University of British Columbia, Vancouver V6T 1W5, Canada

SUMMARY

CHEF is an interactive text editor for use with both printing and display terminals. Its prime field of application is computer source program editing but it has some word processing capabilities that make it useful for documentation and general text editing work. There is a comprehensive set of whole-line operations, including block moves and the insertion of text from external files, together with substring replacement and line segmentation based on a flexible pattern-matching algorithm. There is a set of one-line buffers for temporary storage of lines or command strings and complex command sequences can be built up by macro substitution. Considerable effort has been made to design a command syntax that is flexible and consistent and, at the same time, minimizes effort during the editing process. CHEF copies the user's file into an internal work-space so that the original is not disturbed until the user is satisfied with the results of the editing session. A virtual memory technique is used to provide a work-space that can be almost any desired size, with random access to any part and efficient editing operations. CHEF is written in BCPL and has already been implemented on four different machines.

KEY WORDS Text editor Portability BCPL

INTRODUCTION

In 1977 one of us collaborated (J. E. L. P. and W. E. Webb) in the writing of a BCPL version of the UNIX text editor ED.[1] This was designed to be portable to a variety of machines, both large and small, possessing BCPL compilers. The design was quite close to that of ED, although there were several new features and a few omissions.

Since then we have had other thoughts about many aspects of the original design and its implementation, culminating in the redesign and rewriting of the entire editor. The differences from ED in the commands and facilities are now such that we feel that the new editor CHEF should stand on its own feet. Nonetheless we would like to acknowledge the influence of its worthy predecessor.

DESIGN AIMS

As its users know, ED is characterized by its terse commands. We have retained this brevity, and in some cases enhanced it, believing that the novice user is better helped in ways other than by providing a wordy command syntax.

0038–0644/81/050467–11$01.10

Received 30 May 1980

We have applied the principle of orthogonality, reducing the number of independent concepts to a minimum and increasing the power of each as much as is reasonable. We feel that this helps the novice and the experienced user alike, by minimizing the number of concepts that must be learned and making it more likely that the user can achieve what he wants in a simple or obvious way. The application of this principle has increased the range of things that can be done with a single command, without sacrificing brevity.

We have aimed for consistency in the meanings of symbols and the default values of command elements so that commands are more likely to mean what the user imagines them to mean.

Some additional features have been included that seem to us to increase the usefulness of CHEF. In doing this, we have tried to maintain the simplicity and consistency of the command syntax and not to compromise the straightforward use of the editor, particularly for the novice user. Implementors with small machines may wish to sacrifice this feature or that and the code is constructed in such a way that this is easily done.

Much effort has been put into writing a portable system suitable for both large and small machines. Moreover, the code is carefully constructed in the hope that it may serve as a model for students of system programming and portability.

The name of the editor, CHEF, stands for the Christchurch Edit Facility. The final product is a mixture of many flavours and we hope that the reader will agree that we have concocted a harmonious blend.

THE WORK-SPACE

Like ED, CHEF works on a copy of the user's file and the original is not disturbed or overwritten until this is specifically requested. From the user's point of view, the work-space consists of lines numbered from 1 to some arbitrarily large number and can expand or contract as changes are made to the stored text. The user can access any line in the work-space randomly at any time. Normally the work-space is written to a file when editing is complete.

The user may refer to a line in the work-space by its line number or by requesting CHEF to search for a line containing a specified text pattern. There is an invisible pointer, the 'current line', that simplifies some commands by making it unnecessary to give complete location information. Certain commands change the current line, others leave it unchanged.

It is possible to 'tag' one or more lines with a single character so that they can be picked out later. This tag can be used as a target in search operations and may be changed or removed at will. Tags are not preserved when the work-space is written to a file, but do remain with a text line if lines are shuffled.

If a single work-space is not enough, the present one can be stacked and temporarily replaced by a new work-space. The full gamut of editing operations may be carried out within this new work-space and then, in due course, the original work-space can be reinstated, complete with its stored text.

Although all storage has its physical limitations, we have tried to keep the work-space free of arbitrary size restrictions. The actual limit is implementation-dependent and is quite large; one minicomputer version permits files of up to 128K bytes. However, the

implementor is free to redefine certain constants and thus increase the limit. The details of the virtual memory technique used to achieve this will be described elsewhere.

CHEF COMMAND FORMAT

The general format of a CHEF command is:

⟨*location*⟩ ⟨*operator*⟩ ⟨*modifier*⟩ ⟨*operand(s)*⟩

The ⟨*location*⟩ specifies the text lines that are the target of the command, or the position of an insertion or viewing operation. For many operators a range of lines may be specified, for some a single line, and a number of operators do not permit a location to be specified at all. With operators that specify a location, the location may always be omitted and the current line will be assumed.

The ⟨*operator*⟩ is a letter, except in the case of '?' which is used to query the latest error message or the values of various CHEF parameters. The ⟨*modifier*⟩ is a letter or other character that specifies a variant of the basic operator. For example, the modifier '*f*' indicates that the following operand is a file name.

The ⟨*operand*⟩ may take a variety of forms, depending on the particular command. This includes a range of lines to be inserted into the work-space at some point, a file name, pattern and replacement strings, an integer, and so on. Once the ⟨*operator*⟩ and the ⟨*modifier*⟩ have been parsed, the number of possibilities is sharply reduced so that command analysis is straightforward.

Several commands may be concatenated in one line, separated by ';'. For example, a simple use of this feature is the command pair

r/old/new/; p

which replaces the string '*old*' by the string '*new*' in the current line and then prints the line for verification. More complex applications will be described later.

We have avoided the use of unusual characters such as ESC or CTRL Z, since a command syntax based on common visible characters is more likely to be usable with a range of machines and terminals. Some characters that we use, e.g. '∧' and '\', are not found on some terminals and for these we use instead '@' and '|' respectively. The input of commands is in free format, in the sense that one or more spaces may precede each of the four constituents of a command, or may follow a command.

TEXT BUFFERS AND MACROS

There are 27 one-line buffers called 'controls'. These are used to store frequently used text patterns, portions of commands, or lines containing strings of complete commands. They are stored by the same mechanism used for the work-space but are independent of the current work-space and do not participate in the stacking process.

The controls are addressed by the letters *A* to *Z*, which may be upper or lower case, and the character +. Control *N*, for example, can be referenced as a location by the form '@*N*' so that its contents may be edited as desired with the ordinary editing commands.

If the form '%*N*' is used anywhere in a command it is replaced by the contents of the control *N*. A common use for this feature is to store a complex pattern to save repetitive

typing. For example, the command

r/%A/%B/

performs a string replacement in the current line, obtaining the pattern and its replacement from controls *A* and *B* respectively. The substitution mechanism is disabled by preceding the character '%' with '#'. Thus the combination '°₀N' becomes simply '%*N*'.

This macro mechanism permits the nesting of macros to arbitrary depth. Moreover, substitution is delayed until the last possible moment in command analysis. Thus the command and string

@Ac@B; %*A*

assigns the value of control *B* to control *A* and then executes the commands newly stored in *A*, if meaningful.

The macro capability has been further extended to allow the substitution of text from the console. Just as the symbol '%*A*' is replaced by text from *A*, the symbol '°₀°₀' is replaced by a complete line from the console. For example, if the command

r/old/%%/

is stored in control R, so that it can be invoked with a simple '°₀*R*', CHEF obtains a new replacement string from the console each time the command is used.

LOCATION SPECIFICATIONS

A 'location' may be a single line of the current work-space, a range of lines, or a control. The use of the form '@*A*' to specify control A has already been described.

A work-space line is specified by a 'finder', which is an arithmetic combination (using + and − only) of terms such as the following:

10	line number 10
.	the current line,
$	the last line of the work-space
/abc/	The first line containing '*abc*' found by a forward search from the current line (*abc*\ specifies a backward search),
'*A*	the first line tagged with '*A*' found by a forward search ("*A* specifies a backward search),
~*/abc/*,	~*\abc*\ ,~'*A* and~"*A* which require no matching pattern or tag.

A 'range' of lines is specified with two finders separated by a comma or colon. The difference between the comma and colon is only significant when a search has to be made to evaluate the finders. If the colon is used, the current line pointer is set by the first finder before the second is evaluated.

A finder is an expression that evaluates to a work-space line number. We will use the term 'region' to denote either a single line or a range of lines (but not a control). Thus

10 .+1 $−5 */abc/*+1 are finders
10, 15 */abc/* */abc/*, */def/* are regions

To reduce typing, various components of finders and regions may be elided. The rules are as follows:

i. in a finder, if no term precedes a leading '+' or '−' then an elided '.' is assumed,
ii. in a finder, if no term follows a '+' or '−', then a '1' is assumed,
iii. in a range, either or both finders may be elided. The default value for the first is '1', and for the second '$',
iv. if the complete location specification is elided, the default value is '.'.

These rules have been carefully chosen to yield abbreviated forms that are intuitively reasonable and practically useful.
For example,

,	signifies the entire work-space,
,5	signifies the first five lines,
+	signifies the next line,
++	signifies the line after next,
−*i*	is a command to insert text before the current line,
−, +	is a comand to print the current line and one line on either side.

In choosing the default values for elided finders within a range, we decided to break with the ED tradition wherein the default region is sometimes '1,$' and sometimes '.'. With CHEF the default region is always '.' and the effect of '1,$' can be obtained simply by typing ','.

TEXT PATTERNS

Text patterns may be specified as search targets or as templates for substring selection and we have generally followed the ED model, with some exceptions. Patterns are made up of elements such as the following:

A	matches the letter '*A*',
.	matches any character at all,
[*A*-*Z*_]	matches any single letter in the range '*A*' to '*Z*' or the character '_',
~[*A*-*Z*]	matches anything but an upper-case letter,
~*A*	matches any character except 'A',
#101	matches the character having octal code 101 (#41 in hexadecimal machines).

Any of these elements may be followed by '*' to signify that a string (the longest) of zero or more such characters will be matched.

In addition, the character '∧' matches the beginning of the line and '$' the end of the line. To allow for pattern matching within restricted column ranges, the left and right margins may be set by the user to values between 1 and the line length limit. When this is done, text patterns are matched only within these limits and '∧' and '$' behave as the left and right margins respectively. The novice user need not be aware of this facility since the margins are set to their widest limits when CHEF is started.

The special meaning of any of the characters '[', ']', '~', '∧', '#' and so on may be disabled by the escape character '#'.

WHOLE WORK-SPACE COMMANDS

These commands are for loading and unloading the work-space. None of them accepts a location specifier.

e	clears the current work-space,
efxxx	clears the work-space and reads the file '*xxx*', storing this name as the 'current file',
wf.	writes the work-space to the current file,
n	stacks the work-space and creates an empty new one,
nfzzz	stacks the work-space, creates a new one and reads the file '*zzz*',
q	quits the current work-space for the stacked one (or terminates the session),
qq	unstacks all work-spaces and terminates the session,
qs	quits temporarily to the host system, executing the operand, if there is one, as a system command.

The name of the 'current file' may be set and queried like other CHEF parameters and can be referred to by the symbol '.' in any file name context. We resisted the temptation to let a simple '*w*' mean 'write to the current file' because of a feeling that writing to a file is a serious matter that should not be made too simple, or too likely to be done by mistake. Moreover we feel that it is important to maintain the consistency of the command syntax.

LINE MANIPULATION COMMANDS

The basic editing operations pertinent to whole lines are insertion and deletion. We have chosen to include one more, the wholesale changing of one or more lines, because it is frequently needed and somewhat awkward to produce by a combination of the other two. So we have the '*i*' operator which inserts text lines into the work-space after the location specified, the '*c*' operator which changes text lines at the specified location to any amount of new text, and the '*d*' operator which deletes text at the location.

The source of text for the '*i*' and '*c*' operators may be the console, a file, a region of the work-space, or a control. An '*f*' modifier is used to denote a file and a '*d*' modifier indicates that the source text is to be deleted. The possibilities are best shown by examples:

10, 15*c*	changes lines 10 to 15 to new text entered from the console. The end of text entry is signalled by a line with only a '.' in column one.
10*id*50, 60	moves the block of text in lines 50 to 60 to a new position following line 10.
10, 15*cfxxx*	changes lines 10 to 15 to the contents of the file '*xxx*'.
10, 15*c*50, 60	changes lines 10 to 15 to a copy of the text in lines 50 to 60. (Note that this increases the size of the work-space by 5 lines.)
10, 15*cd*50, 60	changes lines 10 to 15 to the text in lines 50 to 60 and deletes the source lines. (Note that the work-space shrinks by 6 lines.)
10, 15*c@A*	changes lines 10 to 15 to the contents of control *A*. (Note that the work-space shrinks by 5 lines.)

Controls may be specified as locations in many cases, though not all. It is not meaningful to insert new text after a control so the form '*@A*' may never appear as a location to the left of the '*i*' operator. It did not seem reasonable to permit the command '*@Acfxxx*' because a control can only accept one line of a file and there is no concept of a

range of controls. With these two exceptions, controls may be operands of '*i*', '*c*' and '*d*' in the same way as work-space lines.

LINE VIEWING COMMANDS

There are two commands used for the inspection of text. They are '*p*' (print) and '*v*' (view). The first of these is the default operator that is assumed whenever an operator is elided. For example the command '1,7' is equivalent to '1,7*p*', which prints the first seven lines at the console. The operator '*p*' may take several modifiers: '*pn*' prints line numbers, '*pc*' (*count*) prints the number of characters at the location, '*pa*' (*all*) prints the line number, the tag, and the text of the lines, '*pl*' (*lucid*) prints the text with unprintable characters displayed in octal and preceded by a '#', and '*pfxxx*' prints the lines of the location to the file '*xxx*'. The operator '*p*' accepts, as location, either a control or a region.

The operator '*v*' which accepts only a finder as location, is useful for viewing lines at a display terminal. It displays a portion of the work-space centred on the location, together with the line numbers, and the current line is distinguished by a '*'. The half width of this viewing window is controlled by an integer which may be given as an operand, e.g. '*v*10' will display 21 lines. When '*v*' is used without an operand, the viewing window remains at the size last set.

There is an automatic verification of any simple (non-concatenated) command that modifies the work-space. CHEF does this by executing the contents of control + as a command following the command in question. The user may set this control to contain a '*p*' or '*v*' or any other command and may toggle the verification mode on and off with an '*lv*' command.

OPERATIONS WITHIN LINES

The '*r*' operator carries out the replacement of a substring by a new string, either for the first occurrence in the line (no modifier) or for all occurrences ('*a*' modifier). The way of specifying the pattern and replacement strings is the conventional one using an arbitrary delimiter. In both cases the replacement is done for all lines of the location specified by the left operand. Examples are '10,15*ra/old/new/*' and '*@Ar/abc/def/*'.

Sometimes it is useful to be able to split a line and store the segments for future use. This is done with the '*s*' (segment) operator. The left operand of '*s*' is either a finder or control specifier that designates the line to be segmented, while the right operand is a pattern followed by a list of controls. The action of the command is to segment the line according to the specified pattern and to store the segments in order in the controls. The original line is not modified. If the '*s*' operator has no modifier, there are three segments, the lead, the match segment, and the tail. With an '*a*' modifier, the line is segmented by all occurrences of the pattern, p occurrences giving rise to (2p+1) segments. For example, if line 10 is the string '1,2,3',

10*s/, /ABC* stores '1' in control *A*, ',' in *B* and '2,3' in *C*,
10*sa/, /ABCDE* stores '1' in control *A*, ',' in *B*, '2' in *C*, ',' in *D* and '3' in *E*.

In some cases the user will not wish to store all the segments and can achieve this by including dummy characters in the list of controls. For example, if the current line is

$I = I+1;\quad J := J+2;\quad K+3;$

then the command

s/*J*.*2;/–*X*–

extracts the assignment statement '*J* : = *J*+2;' and stores it in control *X*.

With the use of '*s*', quite intricate operations become possible. For example, if the current line is the same as the previous example, the commands

s/*J*.*2;/–*X*–; *r*///; *i*@*X*; *v*

extract the middle assignment statement, delete it from the current line, and then insert it into the work-space as a new line. At the end there is a '*v*' for verification. The result is

```
I : = I+1;  K : = K+3;
J : = J+2;
```

Two aspects of the last example require further explanation. The use of a semicolon in a pattern or replacement is not taken to be a command terminator because of the manner in which the command analysis is done. Secondly, a null pattern in an '*r*' or '*s*' command, or in a finder, is taken to be the last pattern specified, but a null replacement is indeed null.

WORD PROCESSING

We have given CHEF some simple word processing facilities to help in documentation. These consist of the centering of titles, the justification of text into paragraphs with adjustable left and right limits, and the freezing of displays.

These capabilities do not make use of commands embedded in the text like the usual text formatter. The '*j*' (*justify*) operator processes a selected region of the work-space according to simple built-in rules and then replaces the original text with the processed version. It is a simple matter to edit this section and re-justify it if changes are necessary.

Three CHEF parameters concerned with the '*j*' operator are the left and right '*verges*' and the '*threshold*'. Text is picked up starting at the left margin of each line (initially column 1) and justified into lines between the limits of the left and right verges. When a blank or indented line is encountered, this is taken to mean the end of the previous paragraph and its last line is not justified. Any indentation at the beginning of a line is preserved in the processed text. The threshold is a small number, initially 6. If the indentation at the beginning of the line is greater than the threshold, the whole line is 'frozen', in other words it is included in the processed text without change of any sort. This is useful to preserve tabular displays. If sentences are originally separated by at least two blanks or by a new line, then a separation of at least two blanks is preserved.

Any line that starts with the centring symbol (initially ':'), and whose indentation is greater than the threshold, is centred in the processed text. The centring symbol itself is deleted.

REPETITIVE COMMANDS, SCRIPTS AND COMMAND PROGRAMS

CHEF provides three methods for executing complex command sequences. There is a 'global' command of the type provided in ED, a facility to execute command scripts from a file, and the ability to write command programs with the aid of macro substitution.

The first two methods are combined in the '*x*' (*execute*) operator. When the modifier is '*f*', the named file is opened and the editor executes the commands found there. A useful application of this is the loading of control lines with predetermined values using CHEF commands stored in a file such as the following:

@Ac
⟨*text for control A*⟩
@Bc
⟨*text for control B*⟩

By this means various libraries of controls can be maintained.

When there is no modifier, and the '*x*' is followed by a pattern, the remaining commands on the same line are executed repeatedly for each work-space line within the location that contains the pattern. Each such line is then temporarily considered as the current line. For example,

,100 ~*x*/∧ *C*/;*p*

prints the first 100 lines of a Fortran program excluding the comment lines. Note that the pattern can be negated, as in this example, and it could have been enclosed in reverse slashes to indicate that execution should proceed from larger line numbers to smaller. Instead of a pattern to select the lines, a tag may be used (forms '*A*,"*A*,~'*A*, ~"*A*). If the operand following '*x*' is elided, then the pattern '/∧/', which matches every line, is assumed.

Some simple uses of the macro facility have been demonstrated earlier. More complex uses derive from the fact that macro calls can be recursive, but for these to be feasible we need some means of escaping from an indefinite loop. The '*k*' (*kill*) operator provides a simple decision-making facility. If the line specified by its left operand is empty, the remainder of the command line is abandoned. If the right operand is a finder or a control the contents of that line become the new CHEF command line, otherwise the user is prompted for further commands. For example:

@Ak	kills the current command line if *A* is empty,
@Ak@B	kills the current command line if *A* is empty and takes the contents of *B* as the command line.

The following example of a command program causes the contents of selected controls to be printed:

control	*contents*
A	Enter list of controls (form *GHI . . .*)
B	*@Dk; @Ds/[A-Z]/-ED; @Fc@%E; @Fr/∧/%E − /; @F; %B*
C	*@A; @Dc; %B*
D,E,F	(*working space*)

The program is invoked by the macro call '%*C*'. First control *A* is printed, a message requesting the user to specify the lines to be displayed, then the list of lines is read from the console and stored in *D*. Finally, on this line, *B* is invoked as a command string. Note that *B* calls itself recursively so that we have an indefinite loop to display the specified lines. In each cycle of the loop, the '*s*' operator is used to extract the first letter of the string stored in control *D* and then this letter is printed followed by the contents of the corresponding control line. The first command in *B* uses the '*k*' operator to terminate the loop as soon as the list of letters in *D* is exhausted.

UNDOING CHEF COMMANDS

It is desirable to be able to recover quickly from errors in an editing session. For this CHEF provides an undoing operator '*u*' requiring neither a location nor a right operand. The action of the undo operator is to restore the work-space to the state existing before the last command that altered it. Commands that do not change the work-space, such as '*p*' (*print*) or '*v*' (*view*), may have intervened. In order to keep the code for this feature to a reasonable size, the operators '*e*' (*edit*), '*n*' (*new*), '*u*' (*undo*) and '*x*' (*execute*) cannot be undone. Of course the wise user will write out the work-space frequently.

CHEF PARAMETERS

A number of CHEF parameters can be set and queried by the '*l*' (load) and '?' operators with suitable modifiers and operands. For example:

lfxxx	sets the name of the current file to be '*xxx*',
?f	queries the name of the current file,
l∧5	sets the left margin to column 5,
?∧	queries the left margin setting.

The '?' operator without a modifier prints the latest error message. CHEF prints a simple '?' when an error is detected and experienced users often find this sufficient. However, a full error message is available if required.

ON-LINE DOCUMENTATION

Extensive on-line documentation is available to the CHEF user with the '*h*' (*help*) operator which can be used with various operands to print helpful messages describing aspects of the editor. Operands '0' to '9' provide a general introduction to the command syntax and the rules for constructing patterns and location specifications. Operands '*a*' to '*z*' give a description of the operators and modifiers, while the various special characters all have their own descriptions. The descriptive material is contained in a file so only a small amount of code is needed to provide this facility and the messages themselves can be modified easily as experience dictates.

CONCLUSION

CHEF has been implemented on four machines, an Amdahl 470 under MTS, a Data General Eclipse under AOS, a Data General Eclipse under RDOS, and a Digital Equipment PDP-11 under UNIX. Development work has been shared between MTS and RDOS so it has been most important that the code be readily portable between machines of different architecture, and operating systems with diverse philosophies. In the course of this development, and in the writing of all the system documentation, including this paper, CHEF itself has been a day to day working tool that has proven very satisfactory to its users.

The techniques used to implement CHEF and ensure its portability are described in a companion paper.[2] A user's manual is available.[3]

ACKNOWLEDGEMENT

We would like to thank Doug Dyment for his contributions to the text justification facility and for much helpful comment and criticism during the development of CHEF.

REFERENCES

1. B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, Reading, Mass., 1976.
2. J. E. L. Peck and M. A. Maclean, 'The construction of a portable editor', *Software—Practice and Experience*, **11**, 479–489 (1981).
3. M. A. Maclean and J. E. L. Peck, 'The CHEF editor', *TM80—1, Computer Science*, University of British Columbia, Vancouver, B.C., 1980.