# PDP8 CONTEXT EDITOR (MARK 4)

26th February 1968

A. Freeman,
COMPUTER SCIENCE DEPT.,
EDINBURGH UNIVERSITY

TABLE OF CONTENTS

## A1. CONTEXT EDITOR

### A1.1. SUMMARY

The program is written for the PDP-8 computer with teletype, high-speed reader and punch. Tapes may be read in sections on the high-speed reader under teletype control, and edited at the teletype before being punched out. Up to 4,000(8) characters may be held in store.

Editing Commands specify a position in text by quoting the context, and then allow deletions or insertions of specified text, or the inter- changing of specified sections of text.

If desired, series of editing commands may be stored and executed in sequence. This 'programmed editing' facility permits the synthesis of powerful editing commands, without detracting from the simplicity or flexibility of the basic commands.

The punching of text is executed independently of the rest of the program if possible, so that the program can function at maximum speed.

## A2. DETAILED OPERATING INSTRUCTIONS

### A2.1. LOADING AND CALLING

The program is loaded in the usual way through the binary loader, occupying locations 0 - 2000 (8) approximately. The tables and text buffers occupy locations 6000 upwards, and the main file occupies 2000 - 6000.

The program starting address is 200.

### A2.2. BASIC EDITING PROCEDURE

The program edits in a single block through which all the text passes. Text may be added to the end of the block from the input tape by the instructions G (see below).

The user may punch out the block: this does not destroy the text in the machine; a separate command (K) must be used for this purpose. While the text is in the machine it may be edited using the commands described below.

### A2.3. TEXT POINTER

Editing is carried out with reference to a 'text pointer', which may be moved around within the block, or set the head of the block, by the appropiate instructions.

### A2.4. LINE POINTER

A subsidiary pointer is used by the program for the use of listing routines, etc. This need not concern the user for most purposes, but this document will refer to it occasionally, so its existance should be noted. It is set whenever a search is made, by AFTER, or by DELETE, etc. (See below), to the start of the line containing/

containing the text.

In certain cases, the line pointer may be separated by more than one line from the text pointer; in particular, insertions do not move the line pointer, so that the line pointer and text pointer become separated, searches involving more than one line of text have a similar effect.

If the command is followed by a prime, then the program does not read a number, but uses the same number as was typed in last time the command was used.

## A2.5. EDITING INSTRUCTIONS

### A2.5.1. DEFINITION OF AN INSTRUCTION

An instruction comprises a string of BASIC COMMAND SYMBOLS, terminated by any non-command symbol.

Basic command symbols are: all alphabetic characters together with the symbols:-

' () CTRL(T) "

Carriage return & linefeed are also accepted in commands, but have no effect.

### A2.5.2. TYPING AN INSTRUCTION

In the idling state the program awaits for an instruction before taking any action, typing a newline and a colon to indicate when it reaches this state. Instructions may then be typed in, and will be executed when the non-command terminating symbol is typed. This last is obligatory: in future a space sign (_) will be used to indicate its presence.

If 'rubout' is depressed whilst typing an instruction, the last/

last command symbol is erased and a star typed. If 'rubout' is pressed at the beginning of an instruction, then the last instructions typed is reactivated.

If 'Alt Mode' is depressed at any stage whilst the program is typing or reading, the entire current instruction is scrapped and the program reverts to the idling state.

A2.5.5. SIMPLE AND COMPOUND INSTRUCTIONS

Editing Instructions may be simple or compound. Compound instructions will be described in sections 2 (Programmed Editing). The user need not know about these in order to use the edit program.

All simple instructions comprise a single letter, which may be followed by a prime. They fall into three main categories:

i) Those associated with a number specified by the user.
ii) Those associated with a piece of text specified by the user.
iii) Others.

A2.6. COMMANDS ASSOCIATED WITH A NUMBER

The program types a number sign (#) and waits for a number. The first numeric character typed starts the reading of the number; the first subsequent non-numeric character terminates it. Typing 'Rubout' causes the program to restart reading the number, ignoring all previous numeric charaters ( not cancelling the instruction). Typing 'Alt Mode' cancels the instruction, as stated previously. The Commands are:

A2.6.1. G_g where g is any +ve integer

Get g lines of text from the tape to be edited and append them/

them to the block; set the pointer to the front of the appended text, and list the first line. (See: Optional Listing).

Reading terminates at the end of a line and 'FULL' is typed when the buffer is half full. If, however, a very long 'line' (e.g. with a lot of runout in it) is read in, the buffer may be further filled. In this case reading terminates anyway when the buffer is full, 'FULL' is typed and the current command is scrapped.

Erase characters are ignored, but not runout. However, runout is not listed on the teletype.

A2.6.2. L_g

List g lines, starting from that containing the line pointer. An up-arrow (↑) is printed to indicate the position of the text pointer. This arrow is NOT punched out and ONLY appears in teletype listings.

The pointers are not affected by these instructions.

A2.6.3. M_g

Move the text on line pointers down g lines. If there are less than g lines, the pointers move to the end of the block.

A2.6.4. N_g

Move the pointers up g lines. N_0 is taken, conventionally, to mean ''set the text pointer to the line pointer''. In all other cases using ∅ as a parameter has the conventional meaning of ''an arbitrarily large number'': e.g. G_0 means ''get as many lines as possible'', and M_0 means ''move

the/

the pointers to the end of the block''.

## A2.7. COMMANDS ASSOCIATED WITH A PIECE OF TEXT

These type out a brief caption and a double quote ('') to indicate that text is expected form the user. Everything (EVERYTHING) up to the next double quote is read into the relevant text buffer.

Typing a 'rubout' during this causes the program to erase the last character of assembled text, typing it back. If the text is erased as far back as the beginning, further rubouts cause a fresh quote to be typed. Typing 'Alt Mode' cancels the entire instruction.

PARAMETERS IN THE COMMAND

As with number commands, if any of the text commands is followed by a prime, then the program uses the same buffer as was typed in last time, and omits to read any fresh text.

Parameters may also be introduced into the command by typing a control ''T'' (Hold CTRL Key and depress ''T''). The program types a space and a double quote and reads in text, which is stored in a dictionary and used by the command immediately preceding it. When the terminating double quote has been typed, further commands may be added after the text parameter. The commands are:

### A2.7.1. A_

The program types AFTER ''. When the text has been specified it is located and the pointer is set at the end of it.

The lines containing it are listed.

The search begins at the current position of the text pointer,/

pointer, continues to the end of the block, and then goes through the whole block. The first occurrence of the specified text in this order of search is used.

Failure to find the text results in a diagnostic, and the pointer is set to the start of the block. (But see the command H_ )

A2.7.2. F_

The program types FROM ''. This command is the same as AFTER, except the pointer is set to the beginning of the specified text.

A2.7.3. D_

The program types DELETE ''. When the text is assembled, it is located and deleted. The text pointer moves to the next position following the deleted text. The search begins at the next pointer and finishes at the end of the block.

It is not possible to delete an occurrence of the specified text PRECEDING the pointer.

Failure to locate the text causes a diagnostic, and the pointer is set to the head of the block.

A2.7.4. I_

The program types INSERT ''. The text is inserted at the current position of the text pointer. The pointer moves to the end of the inserted text. An attempt to insert too many characters (i.e. overfill the buffer) is not carried out, and causes a diagnostic.

A2.7.5. U_

The program types UNTIL ". Everything from the current position of the text pointer to the BEGINNING of the specified text is deleted. The same restrictions apply as for DELETE.

## A2.8. OTHER COMMANDS

A2.8.1. P_

Punch out the current block (all of it) on the high-speed punch, without destroying it, and set the pointers to the head of it.

See: Section [[A2.9]] : Punching and Buffering

A2.8.2. K_

Kill the current block; i.e. delete it. Normally this command is combined with P_
(See:Programmed Editing)

A2.8.3. B_

Set both pointers to the head of the block

A2.8.4. R_

Punch 200 Runout characters.

A2.8.5. C_

Change terminator. The next character typed in is used in place of the double quote as a text delimiter.

A2.8.6. H_

This command may be used, after any search has been made, to restore the positions occupied by the pointers before the search. Its main use is in the event of an error (H stands for HELP!) in specifying text, in which case the user can ´go back'and try again´.

A2.8.7. LEFT AND RIGHT SQUARE BRACKETS [ AND ]

These two commands are used to temporarily limit the scope of the block. The first moves the head of the block to the position now occupied by the text pointer. The modified form, ´ , restores the old position. The right square bracket has the same effect on the end of the block.

Care must be taken to remove temporary limits placed on the block before punching, as the command P punches only between the temporary limits.

A2.8.8. S_ AND S´_

The command S_ destroys all the text outside the temporary limits. The command S´_ destroys all the text inside.

A2.8.9. C_

This command swaps round the piece of text between the head of the block and the text pointer, with that between the text pointer and the end of the block. By using this in conjunction with the scope limiting commands, any permutation of the text may be effected.

## A2.9. PUNCHING AND BUFFERING

The buffering system used is arranged so that the user may punch out and read in simultaneously, provided he does not make large insertions. This is explained below in greater detail: it is based on the assumption that the user either will be processing text fast with few, if any alterations, using the Programming facility; in which case I/O speed is all important; or he will be making considerable alterations in which case it is more important to have plenty of space available (the program does not update the text it holds).

### A2.9.1. READING IN

The program attempts to terminate reading at the end of a line, before the buffer is half-full. The diagnostic ´FULL´ is typed if the number of lines requested do not fit completely in less than half the buffer.

### A2.9.2. INSERTING AND DELETING

The whole buffer is available for insertions. The buffer used to store text during ASSEMBLY, however, is limited in length. The program types ´FULL´ if a piece of text being assembled exceeds this length, and terminates assembly, exactly as if a double quote had been typed. The text is then used in the normal way.

If an attempt is made to insert a piece of text which would overfill the whole main buffer, ´FULL´ is typed and the program reverts to the idling state. (The text may be retrieved by punching and destroying the block and typing I´ ).

It should be noted that no space is ever recovered after a deletion./

deletion. Only K releases fresh space for insertions.

### A2.9.3. PUNCHING

If the buffer is less than half full, it is copied into the upper half of the buffer and a punching process is launched which 'steals' time from the other processes, appearing to function independently. Any attempt by other processes to access the top half of the buffer is held up until punching is complete. If the buffer is more than half full, it is punched out directly and no other processes are launched until this is complete.

### A2.10. LISTING AND DIAGNOSTICS

The instructions involving the text pointer, excluding the instruction B , all list the lines containing the line & text pointers after successful execution. This listing may be suppressed in two ways.

(i)

If Bit . of the Computer SR is set to 1 (in the 'up' position), listing will be suppressed. This does not affect listing from the L instruction.

(ii)

Either the 'Alt Mode' or the 'Rubout' Key may be depressed during listing, interrupting it. In the first case the program reverts to the idling state; in the second the program continues to the next command of a compound command (if there is one). (See Section 2 on programmed editing.). The program also types a number of diagnostics and captions from time to time, which may be similarly suppresse using bit 1 of the SR.

However, the first letter of the text command captions is always typed out, as well as the double quote.

## A2.11. SUMMARY OF COMMANDS

| Nature of Command | Command | Modified form | Effect | Notes |
|---|---|---|---|---|
| Input | G N | G' | Input N lines | G 0 means 'Input as many lines as possible' |
| Output | P | | Punch the Block | Functions independently of other operations if possible |
| | R | | Punch 200 Runout characters | |
| | L N | L' | List N lines | |
| Pointer Control | B | | Set TP&LP to head of block | |
| | M N | M' | More TP&LP down N lines | M 0 means 'Move TP&LP to end of block' |
| | H N | H' | Move TP&LP up N lines | H 0 means 'Move TP to LP' |
| Searching | A Text | A' | Set TP after Text | |
| | F Text | F' | Set TP before Text | |
| Deletions | D Text | D' | Delete Text | Search extends from TP to end of block only |
| | U Text | U' | Delete from TP until Text | |
| Insertion | I Text | I' | Insert Text at TP | |
| Scope Control | [ | | Temporarily more head of block to TP | |
| | | [' | Restore old block head | |
| | ] | | Temporarily move end of block to TP | |
| | | ]' | Restore old block end | |
| Deletion | K | | Delete whole block | |
| | S | | Delete all outside temporary block | |
| | S' | | Delete all inside | |
| | C | | Interchange first & second half of block | |
| | Q | | Change Terminator | |
| | M | | Restore old TP and LP. | |

## B1. PROGRAMMED EDITING

### B1.1. Introduction

Using single, simple editing commands, in any system, has two disadvantages.

Firstly, editing which is a highly repetitive nature can be tedious.

Secondly, it is almost impossible to devise a set of editing commands which will cover all editing tasks, unless the set of commands is so big that both user and programmer get throughly confused.

To try and overcome this difficulty, the program facility has been introduced.

The editing commands have been kept few and simple, but have been chosen to give greatest flexibility and generality in the creation of complex commands.

### B1.2. Use of the facility

An instruction, as stated previously, is a set of command symbols terminated by any non-command character. All instructions so far described have been simple, comprising a single letter (which may be modified by a prime) only.

The program facility allows simple commands to be added to each other and executed in sequence, exactly as if each had been typed in singly. This is achieved by typing the simple commands one after another, omitting the terminating symbol until all the required symbols are typed.

e.g.

| | |
|---|---|
| : PK_ | Punch and destroy block. |
| : DI_ | Type DELETE´´ |
| | Read in text, locate and delete it. |
| | Type INSERT´´ |
| | Read in text and insert it. |
| : D´I´L | Locate and delete last delete |
| | parameter typed in. |
| | Insert last insert parameter. |
| | Type # |
| | Read in a number(n) |
| | List n lines. |

Typing a rubout during this Process erases one character of the command, typing a star to indicate this.

## B1.3. Loop Control and Branching

During the execution of a command sequence, it may be necessary to repeat a given portion, or to branch depending on whether or not a command has been successful. For this purpose a command may be sectioned using left and right brackets and the dash(´).

### B1.3.1. Branching

This occurs when an ´exit condition´ arises.

Exit conditions are as follows:

(i) Failure to locate text during a search by AFTER, DELETE, FROM or UNTIL.

(ii) Use of the command E (see below).

(iii) Attempting to read when the end of the tape has been reached.

When such a condition occurs, the program exits from the bracket pair enclosing the sequence and recommences execution from the right bracket of the pair. If no enclosing right bracket is found the program reverts to the idling state.

B1.3.2. Repitition

A command sequence is executed repetitively if enclosed in a bracket pair of which the right-hand bracket is followed by a dash. Exit conditions still cause an exit from the bracket pair.

Example. Consider the command (G´(D´I´)´PK)´

Assume that the GET parameter last typed in was 5, and that the DELETE and INSERT parameters were ´´*´´ and ´´/´´ respectively.

The program will read in five lines. It will then attempt to locate an occurrence of ´´*´´. If successful, it will replace it by ´´/´´. It will then repeat from the second left bracket, attempting to find another ´´*´´. It will continue deleting asterisks and inserting slashes until there are no more asterisks, when it will type ´EH?´, set the pointers to the head of the block, and exit from the loop, punch the block and destroy it, and repeat from the first left bracket reading in a fresh five lines. This will continue until the end of the tape is reached, when it will type ´EH?´ and exit completely, typing a newline and a colon.

The diagnostics may, as previously stated, be suppressed by setting bit 1 of the SD to 1.

B1.3.3. The double quote (2+Shift)

This is ignored the first time it is executed, but replaced by a dash; so that it subsequently behaves as if it were a dash. This permits many instructions to be written more concisely:- for example instructions like (D´´I´´)´, which obtains DELETE and INSERT parameters from the user and then substitutes the second for the first everywhere in the block.

B1.3.4. The Command E

This command allows the user some measure of control over his ´mini-program´.

It types out X? and waits for the user to specify what he wants to do. If he types ´Alt Mode´ the program reverts to the idling state: if he types ´Rubout´, then an ´exit condition´ is created. Anything else typed in causes the program to resume execution at the next command after the ´E´.

B1.3.5. The command X

This causes a complete exit from the command sequence. This is useful for commands like:

(G´´(A´´X)PK)´

Which copies the tape until an occurrence of the AFTER parameter is located, and then exits.