707

EDJNU — A WORDPROCESSING PACKAGE IN 'C' LANGUAGE

675

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for
the award of the Degree of

MASTER OF TECHNOLOGY
IN COMPUTER SCIENCE

28p

ANUPAM GOVIL

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI—110067

1988

DISSERTATION

TOPIC : EDJNU - A WORDPROCESSING PACKAGE IN 'C' LANGUAGE

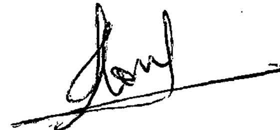
BY

ANUPAM GOVIL
M. TECH
COMPUTER SCIENCE
JAWAHARLAL NEHRU UNIV.
NEW DELHI

<u>DECLARATION</u>

The work embodied in this Dissertation contains the result of the research work carried out under the supervision of Dr. P. C. Saxena , SCSS , JNU , New Delhi . The work is original and has not been submitted , in part or full , to any other University for the award of any other degree or diploma .

PROF. KARMESHU
DEAN



ANUPAM GOVIL
STUDENT

DR. P. C. SAXENA
SUPERVISOR

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-67

ACKNOWLEDGEMENTS

I am indebted to my supervisor Dr. P. C. Saxena for his guidance and the encouragement provided by him in carrying out this work. His affection and personal interest have always been an added stimulus.

My thanks are also due to the faculty members for their kind cooperation and encouragement.

I am thankful to the computer laboratory staff for their cooperation in letting me use the computer equipment.

Finally I express my gratitude to the Jawaharlal Nehru University for the financial assistance.

ANUPAM GOVIL
M. TECH
COMPUTER SCIENCE
JAWAHARLAL NEHRU UNIV.
NEW DELHI

PREFACE

This Dissertation deals with an indigeneously developed Word Processor - EDJNU . This package has a standard set of commands with a few customised features . The main advantage of this Word Processor is its compactness , its flexibility and its portability . The language C has been chosen for development of this software due ot its various inherent advantages . This software has been developed on a IBM PC -AT compatible using a Turbo C compiler . However this Wordprocessor can run on almost any machine due to its compact code and device independence .

All important aspects of development of this software have been exhaustively discussed in the Dissertation . We start from a brief description of programming techniques in C , the advantages and applications of C in Software Enginnering . Then we give an introduction to the Wordprocessor and discuss the various considerations that have gone into designing the word processor . Thereafter we provide a detailed structural analysis of the software itself . Here each and every routine is discussed seperately together with the special techniques used .Finally we link each command with these routines and hence elucidate the execution path of each command . Care has been taken to make the Dissertation as informative and detailed as possible . Even an average user of C would be able to understand the flow and structure of the complete program .

CONTENTS

SECTION I : Advantages and Techniques of 'C' in Software Engineering

1.1.1 Role of C in Software Engineering
1.1.2 C Language Constructs
1.1.3 Program Structures
1.1.4 Simple Data Types
1.1.5 Control Structures
1.1.6 if Statement
1.1.7 ? Statement
1.1.8 switch Statement
1.1.9 Structured Data Types
1.1.10 Functions
1.1.11 Pointer Variables
1.1.12 Recursion in C
1.1.13 Applications of C
1.2 Conclusion

SECTION II : Introduction and Salient Features of the Package

2.1 Introduction to A Word Processor
2.1.1 What is a Wordprocessor
2.1.2 Developing a Word Processor
2.1.3 Choosing a Language for the Editor
2.1.4 Why Use C ?
2.1.5 Why to write a Text Editor
2.2.1 Salient Features of Word Processor EDJNU
2.2.2 Algorithm for Designing a Text Editor
2.2.3 Implementation of Functions
2.2.4 A Few Salient Features

SECTION III : Structural Analysis of the Text Editor Program

3.1 Header Files - A Preview
3.2 Header Files Used in this Package
3.3 Structural Analysis of Text Editor Program
3.3.1 main()
3.3.2 searchcomd()
3.3.3 executecomd()
3.3.4 store()

3.3.5 getime()
3.3.6 gethelp()
3.3.7 savefile()
3.3.8 loadfile()
3.3.9 ldfile()
3.3.10 exeseq()
3.3.11 displine()
3.3.12 display()
3.3.13 atoi()
3.3.14 listall()
3.3.15 deletel()
3.3.16 skipblank()
3.3.17 delall()
3.3.18 delline()
3.3.19 lowercase()
3.3.20 instext()
3.3.21 movtext()
3.3.22 findstring()
3.3.23 nfindstring()
3.3.24 nchng_string()
3.3.25 nfind_replace()
3.3.26 chng_string()
3.3.27 find_replace()
3.3.28 strn_cmp()
3.3.29 search()
3.3.30 append()
3.3.31 duptext()
3.3.32 mergfile()
3.3.33 nldfile()
3.4 Execution Path of the Commands

SECTION IV : Conclusion and Appendices

4.1 Conclusion
4.2 Appendix A : List of Commands
4.3 Appendix B : List of Routines written for this Editor
4.4 Appendix C : Bibliography
4.5 Appendix D : Program Listing

SECTION I

ADVANTAGES AND TECHNIQUES OF 'C' IN SOFTWARE ENGINEERING

THE ROLE OF C IN SOFTWARE ENGINEERING

The Programming language C was developed by Dennis M Ritchie in 1972. C was an off-spring of Martin Richard's BPCL, by way of a language called B written by Ken Thompson. Most languages in common use to-day were developed by committees. But C was developed by Ritchie and Thomson, who were undoubtedly excellent system programmers. So C was developed by programmers for the programmers. It is relatively a pure language, unrestricted by compromise. As a result of this it is capable of doing anything that you can expect a language to do.

The C was made available to public in 1979. Dennis Ritchie calls this language as "a programmming language which features economy of expression, modern control flow and data structures, and a rich set of operations".

A programming language is judged from its support for the design and production of correct, reliable, maintainable software. So in discussing C, we should consider the extent to which it meets these criteria.

The major problem faced by designers and implementors of realistically large software systems is that of complexity and the principal tools to deal with complexity are the process of abstraction and refinement. Consequently, programming languages should support expression of abstract concepts but must also provide means of ealizing the concepts. To allow the programmer to express abstract concepts, C provides a hierarchical program structure and user definition of data types; realization is achieved by performing a series of transforamtions, or refinements, on the original procedures and data structures until an executable form is obtained.

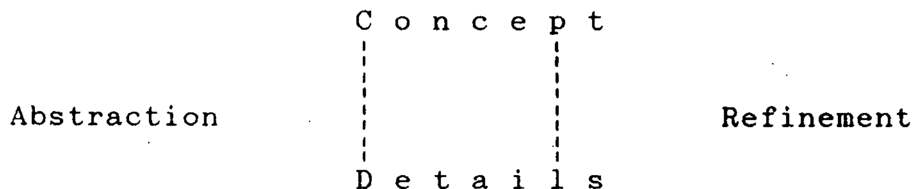
It is assumed that the programmer has expressed the original model of the solutionj correctly - and the likely hood of this being so will be influenced by how closely the language being used matches the programmer's cconept of the problem then the correctness and reliability of the final program will depend upon the accuracy of the refinements made to the original solution. As the refinements represent only nformal (as opposed to mathematically formal) transformations, their preservation of correctness cally formal) transformations, their preservation of correctness will depend primarily on the clarity and ease with which they can be made.

Aspects of C which facilitates this process are, for example, unambigous control structures with a single entry point and single exit point, and a hierarchical data types with compiler enforced type checking. Meaningful choice of datanames (and appropriate comments in routines) also help the programmer to implement ideas and are a primary means of communication when it comes to program maintenance. Software development by successive refinement allows the maintenance programmer to follow each explicit design decision which have been made. If subsequently there are changes to software requirements or environment, it is possible to see clearly which levels of abstraction are affected, and the extent of amendment is required.

This section is intended to present the characteristics, level scope and use of C, and to indicate the support it can offer in software engineering practice. Then we also discuss the major language constructs. After this discussion , the various applications of C have been discused. In order to appreciate the potential of C and its application, the algorithms and development of a full-fledged text editor is mentioned. This is given as an illustration of the refinement of an abstract concept (in this case, a text editor) to an executable C program.

C language Constructs :

The Concepts of abstraction and refinement are the programmer's principal aids in developing computer programs. Presented with the details of a complex problem, the programmer can abstract to a higher level at which the concept to be realized is isolated from details of its implementation and environment.



Once the concept has been isolated, a model for a solution can be developed. The solution will be implemented by refining the model until it is realizable language. With this approach programes are developed by successive refinements of (active) algorithems and the (passive) data structures on which they operate.

The basic mechanisms provided by C to support refinements are the functions for algorithms, and the user defined data types for data structures. Underlying the functions are C control statements and underlying user defined data types are C data types and structures.

The language constructs provided by C can be summarized under the heading of program structures, and data types and structures.

Program Structures

'A C program has the following form :-

Constant definitions,

external files to be included

external variables and data types declarations

global variable and data types declarations'

main function declarations ;

other functions declarations;

You can include declarations given in some other files if you want to use some external functions. For example

#include <stdio.h> is used for standard input output functions like scanf ( ), printf ( ), getchar ( ), Putchar ( ), etc.

If you want to access a variable declared in some other files you have to define them as external like e.g.

```
extern      int     i;
```

This allows the programer to access the variable i declared in some other file. C allows data abstraction and data hiding efficiently like in Modula - 2.

Control passes to the main function on the commencement of execution. The main function can in turn call other functions and other functions can call some other functions. One thing has to be noted here is that function (s) cannot be defined inside other function in C as being done in Pascal or Modula - 2.

Simple Data Types :

'int', 'short', 'long', 'float', 'double', 'char' are standard C data types. some examples are given below.

```
#define        TRUE      1

#define        FALSE     0

Main (     )

               int                 i;
               short               is;
               long                l;
               float               total;
               char                c;
               double              d;
```

The programmer can define further scalar types (user define types) by specifiying an existing file i.e.

```
typedef              short                byte;
byte                              xl;
```

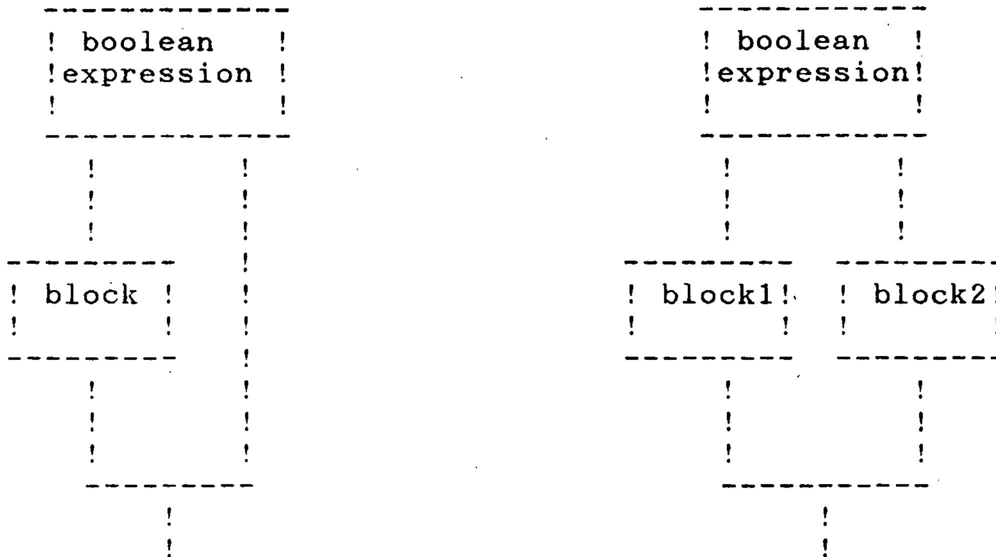
So xl is an integer of 8 -bits.

Control Structures ;

C control structures fall into two main categories; Conditional (if, switch, ?) and repetitive (do, while, for,).

```
if (x > y)
{
     some code
}
else
{
     more code
}
```

```
if (x > y)
{
     some code
}
```



if .. statement **if ....else....statement**

if statement can be nested e.g.

```
if (red)
     stop();
          else
               if green()
                    go();
                         else
                              reverse();
```

One should note that the statemenmt following else corresponding to the statement following else corresponding to the previous if ... statement .

? Statement :

This function evaluates the expression . If the expression is True , then it executes the statement immediately after it , else it executes the statement after the symbol ":" . e.g.

```
int lower(int c)    /*converts a char to lowercase*/
{
     c = (isupper(c) ? tolower(c) : c ) ;
     return(c);
}
```

switch Statement :

This statement allows you to take different actions on the value of a variable . This is analogous to the **case** statement in Pascal . Let us consider a device driver for a Printer =>

```
switch(c)
{
     case FF :
               ffflag = 1;
               break;

     case '\n' :
               putlpr('\r');
               putlpr('\n');
               printno(lineno++);
               break;
          .
          .
          .
          .
     default :
               putlpr(c);
               colno++;
}
```

One point to be noted here is that C statement falls through in the switch statement . For example , if C = FF then all the staements below Case FF and till } will be executed . So a break statement is necessary to exit the switch statement .

C's repetitive structures "for" , "do" and "while" have essentially the same constituent parts :- A boolean expression to be evaluated and a Statement or sequence of statements to be executed . They only differ in relative placements .

The structure of "for" loop is :-

```
for (initial value ; condition ; increment etc.)
        {
                <some statements>
        }
```

The structure of "while" statement is :-

```
while (condition)
        {
                <do something>
        }
```

The structure of "do .. while" is :-

```
do
        {
                <some statements>
        }
while (condition)
```

There can also be infinite loops . For e.g. :-

```
1.   for ( ; ; )
        {
                <do junk>
        }

2.   while(1)
        {
                <do some other junk>
        }
```

The above loops will be executed forever , unless a break , return or exit statement is encountered .

Structured Data Types

- **arrays**
- **structures**
- **unions**
- **files**

Arrays : Arrays are not declared explicitly in C as in Pascal . Instead they form the part of a char or int declaration as in the following example :-

```
char  a[10] ;    /* This declares an array a of maximum
                    10 characters */
```

Structures : Structures are used to keep different data items under one common name . This is similar to Record type in Pascal but much more versatile . For example :-

```
typedef  struct  employee
          {
          int   empno;
          char  *name;
          int   birth;
          char  desg[20];
          }  EMPLOYEE ;
```

Here EMPLOYEE is declared to be a structure of type "employee" which has got integer variable empno , char string name , int variable birth and char string desg . Hence this struct has 4 elements . These 4 elements can be modified individually as in next example :-

```
EMPLOYEE  empmast;
     .
     .
     .
empmast.name = "MARILYN";
     .
     .
```

So MARILYN is stored in the variable name in the struct empmast .

Unions : Union in C allows you to select a particular data type among the various ones . For e.g. :-

```
union  regs
       {
        int kk;
        char *x;
        float fl;
       } REGS ;
```

Only one variable can be accessed at any time which cvan be kk or x or fl .

Files : Files are handled in C through file pointers . A typical file handling example is given below :-

```
#include  <stdio.h>

main()
     {
          int c;
          cahr filename[30];
          FILE *fp1;
          puts("** Enter a filename **");
          gets(filename);
          fp1 = fopen(filename,"r");
          while ((c = fgetc(fp1) != EOLN )
               putc(c);
     }
```

This reads a file and prints the file on the VDU one character at a time .

Functions

A function can be a procedure or a subroutine in C . It may or may not return a value . A function can be called with or without parameters . The return value of a function can be predefined . Let us take an example :-

```
main()
    {
        int a,b,c;
        a = b = 5;
        c = add(a,b);
        printf("c =%d\n",c);
    }

int add(int x , int y)  /* A function that adds 2 int*/
        {
            return(x+y);
        }
```

A function can return a integer , float , character or a pointer to any of the above . It can also return pointer to structures and functions .

Functions that return nothing are declared as void :-

```
void functionanme()
```

Functions are the most important aspect of C's modularity and flexibility . Since functions for small tasks can be easily written and attached to the main program , huge programs become more easy to understand , decode and modify .

Pointer Variables

Pointers are the most powerful feature of C language . Pointers allow the creation of dynamic data structures while size can change during execution of a program . A pointer variable allows you to return the space allocated back to the operating system . The pointer variables can point to anything like integer , float , struct or even functions ! A pointer can literally point to any part of the memory . In addition to these C allows pointer arithmetic to facilitate greater flexibility in accessing different types of data structures . Let us take an example of a linked list which stores different numbers .

```
typedef   struct    link
     {
          int  number;
          link *next;
     }  LINK ;

LINK first;
first = {0,NULL};
LINK *p;

main()
     {
          p = first.next;
          p = malloc(sizeof(LINK));
          p->number = 10;
          p->next = NULL;
          .....
          .....
     }
```

The space dynamically allocated by the function malloc() can be returned by function mfree() .

You can define structures , inside structures , inside pointers and mix them up . So the programmers should be careful in using pointers and should know their implications . Also since pointers can point to any part of memory , they shopuld be manipulated with utmost care , else they can corrupt your own operating system . As they say too much of power in the wrong hands can be very dangerous !

Recursion in C

C allows recursion routines . i.e. the routines can call themselves . Let us give an example :-

```
#include <stdio.h>

main()     /* This finds the factorial of a number */
     {
          int n;
          puts("** Enter a digit **");
          scanf("%d",&n);
          printf("** Factorial of %d is = %d\n",n,
                         factor(n));
     }

int factor(int x)
     {
          if (x == 0 || x == 1)
               return(x) ;
          return(x*factor(x-1));
     }
```

Here the function factor() calls on itself recursively until x is reduced to 1 . This recursive property of C is one source behind the power of C .

Applications of C

THere are various areas in which C has found its usefulness However we shall concentrate on three areas - namely System software ; Scientific applications and Real-time software .

C in System Software :

C is a medium level language . Hence it has the user friendliness of a high level language and the power and efficiency of a low level language . These properties make C the ideal language to develop System Siftware in .

The applicability of C in system software has been clearly illustrated in the next Section where we discuss why C was chosen to develop this Wordprocessing package .

C in Scientific Applications :

C has got integer , floating point and double precision data types to do numerical computations . The usual storage allocated for these data types is as given below :-

int a ; 16 bit single word is allocated to the integer variable a .

short x; 8 bit byte is allocated to the integer variable x .

float f1 ; 32 bit two words are allocated to the floating point variable f1 .

double d ; 64 bit 4 words are allocated to the variable d .

So C has the necessary storage allocation for handling complex arithmetic . Using these data types various complex scientific functions can be built in C .

C also allows data conversion . You can assign integer to a floating point number , double to an integer etc. These features can be used to their advantage by clever programmers .

C in Real Time Systems :

With the development of Concurrent C it has become possible to design real-time software in C . A language should have the following properties for real-time programming :-

1) Execution speed : Fast code generation by the compiler .

2) Should possess the ability to react to external events within a specified response time .

3) Should be able to handle multiple tasks (either by illusion or multiprocessors).

4) Should facilitate inter-process communication .

5) Since software has to run continously , it should have sufficient error recovery methods in built .

The Concurrent C has these advantages and more . Features like semaphores , monitors , messages , critical sections , concurrent statements etc . are ideally suited for programming real-time software .

Conclusion

This section has presented the major features of C and illustrated their use in program design and development . What programming languages offer however is support for , and not a guarantee of , good software production . The constructs provided by a language must therefore underpin a more comprehensive philosophy of software engineering .

C's undoubted success as a programming language is a result of its provision of basic language constructs which support the techniques of abstraction and refinement . C is very flexible language and it does not restrict the programmer from using it freely . However C expects the programmer to write their own error routines rather than the compiler giving the error messages This can be used to their advantages by skilled C programmers .

A lot of C compilers are available in the market today . C has been made very versatile by some software vendors by developing huge libraries and providing low level interfaces with the hardware . This has made C very flexible . Many critical projects which were previosly done in assembly languages are now being done in C as it almost provides the power and performance of an assembler .

C is now available on most of the computers and runs on every popular operating system . Of late C is also being used in diversified areas like scientific applications , real time systems and artificial intelligence . Hence C can truly be called a Universal language which is finally coming of age and making its impact on the Software Engineering world .

SECTION II

INTRODUCTION AND SALIENT FEATURES OF THE PACKAGE

INTRODUCTION TO A WORDPROCESSOR

What is a Word Processor ?

A word processor is basically a software package which allows you to enter, retrieve and manipulate text. It is hence a Text Editor with more advanced functions . A minimal text editor should provide the following functions:-

- Creation of Files
- modification on the contents of the files
- File manpulation like loading, saving and merging operations on the files.
- Insertion/Deletion of text which can refer to a character, a word or a block
- Mainpulation of text (like moving a block to some other part of the file, duplicating a block in different places
- String searching and string replacement

Developing a Word Processor:

Keeping all requirements in mind (as mentioned above) algorithm for development of a Word Processor can be evolved.

The editor should have a library of commands. All the commands are stored in this library. This feature allows you to add more ccommands in future. The existing commands can also be manipulated.

The editor works in two modes : -

a) **Command mode**

b) **Text insertion mode**

In the command mode, the input string is analyzed for valied ccommand. If a valid command is found, the command is executed.

In the text insertion mode, the input text is stored in the memory.

Choosing a language for the Editor :

An editor can be written in a high level or a low level language. The low level language can be an assembler. Assemble language programming is exhaustive, errorprone and difficult to debug eventhough it may be faster than high level languages. Moreover, a programl written in an assembly language cannot be run on a kdifferent machine which is possible in case of a high level language.

With the development of optimized compliers, the object codes of high level language have become efficient and nearly as fast as assemblers.

The following points need to be considered before selecting a language for writing a Text Editor.

- The language should be portable (standardized)
- The language should fecilitate efficient string manipulaltion like searching, sorting and concatenation operations on the strings
- Dynamic memory allocation and deallocation for strong, deleting text in memory.
- The language should encourage modular programming
- The language should support rich data types and allow efficient usage of them (e.g. record type in Pascal, struct type in C).

The present text editor has been developed usijng 100% portable C. The sofware is portable and can be run on any ccomputer (mainframes, minis or micros) which has got a C compiler.

TH-2907

Disseratation

681.3.06C
G747
ed.

Why use C ? :

C is very popular high level language which has esablished its usefulness in the programming environment. In spite of being a high level language, C has got many low level features like that of an assembler. It allows dynamic memory allocation and deallocation. C is a highly modular language with a rich set of data type declarations. The string handling is extremely efficient in C due to its very powerful compiler. The biggest advantage of C is its Pointers. Using pointers, the programmer can access any variable or data type or even any memory locations. In addition to this, C allows pointer arithmetic. So one can practically access any part of the memory.

During present time, lot of software development is being done in C. This has been used for writing sophisticated Operating systems, Complex graphic pacckages, Optimized Compilers, Database design, data communications, artificial intelligence etc. The most popular operating system UNIX has bee written almost 97% in C. The breathtaking animations shown in the sci-fi movie "Return of the Jedi" has been written in portable C. In USA majority of the system software development is being done in C.

Why to write a Text Editor ? :

One may ask the question "Why should I write an editor when so many efficient editors and word processor are available "?

My answer to this question is simple and straight forward. Though a lot of editors may be available in the market, you are never given the source code of these packages. So no modifications can be carried out in these packages. Moreover, these editors are not portable i.e. they cannotv be moved from one machine to the other.

When you write a text editor yourself you can eliminate all these problems. A text editor available in the market is very general in nature. But you can tailormade your editor to suit your requirements when you are developing an editor yourself. This will reduce the code size and increase the efficiency of the editor. The code can be modified later to suit your requirements and lot of additional features can be implemented depending on the needs.

SALIENT FEATURES OF WORD PROCESSOR 'EDJNU'

EDJNU is a portable word processor which has been developed using a Turbo C compiler. The software has got a command set which are given below. When any of these command is typed in the Command Mode the appropriate routine is executed.

The commands are :-

LIST	**EXIT**	**NFIND**
LOAD	**TIME**	**MERGE**
RESEQ	**CHANGE**	**MOVE**
DELETE	**NCHANGE**	**INSERT**
SAVE	**FIND**	**DUPLICATE**
HELP	**APPEND**	

For explanations of these commands see appendix A .

The data structure for storing text has been defined as :

```
typedef     struct     text

                inst         lineno;

                char         string [MAX-CHAR];

                fp           *text;

        }    TEXT ;
```

Here 'lineno' represents the actual line number of the text
'string' stores the text in it
'fp' is a pointer to the next structure

Algorithm for designing the Text Editor

The command or input is accepted a line at a time. If the input is a text it is stored in the linked list in ascending order of line number. New spaces is alloted whenever a new line of test is entered. The new text points to the text with the next higher line number. The link from the immediate lower line number packed is conenected to this list.

If the input is a command it is separated and compared with the existing command library.. If a match is found, the control passes to the appropriate action routine. If some error occurs during the execution appropriate exception is done. So the editor never loses control over text.

We shall discuss some of the implementation of various functions in the editor in brief . A more detailed explanation can be found later on in Section III .

Implementation of Functions :

The editor is invoked by typing the command EDJNU. this loads the editor into memory. When the editor is ready to accept input from you it displays a prompt *.

A file can be loaded from secondary storage by typing LOAD filename. The required file is loaded into the memory with line numbers starting from 10 with an increment of 10.

Inserting a Line :

The input text along with the line number is stored in the buffer 'line'. The line number is decoded and stored in variable n. The linked list is searched for the line number n. This is done by traversing from structure first. If it is found the text is overwritten. If not, a new packet is created using the system call malloc after which it is connected to the packet with the next higher line number. Similarly a link from the next lower line number packet is established with the new packet. The flow of operations for this action can be seen from the diagram.

Deletion of a line :

The line number which is to be deleted is stored in a variable and passed onto the routine for deletion.

The packet containing the desired line number is searched in the same way. If found, the packet is freed and returned to the operating system using the system call mfree. The link from the previous packet is now connected to the packet next to the deleted one.

A Few Salient Features :

Listing of line numbers can be of a single number or a range of number or the entire memory . For e.g. :-

LIST [* displays all lines *)

LIST 15 (*displays the line 15*)

LIST 10-100 (*displays all lines between 10 and 100*)

This is also true for the command DELETE.

A lot of routines have been written into the editor to make the job of the user smooth and enjoyable. Please see Section III for more details.

The input text can be entered a line at a time with the line number or in block mode ie. by using the commands insert and append. You can terminate block mode by just pressing <RETURN> KEY ALONE.

A string can be searched in any range of line numbers. Nth occurence of a string in a range of line numbers can also be done. Similarly a string can be substituted for another by the CHANGE command within a range of numbers. The Nth occurence of a string within a line number can also be changed to something else by the NCHANGE command. Similarly a string can be searched forr its Nth occurence in within a range of line numbers.

Line numbers can be entered in any order. However, if the user wants to sequence the line numbers in some order she can do so by command reseq m+n, where m is the starting line number and n is the increment between two adjacent lines.

The lines in the memory can be stored into a file by giving the command save filename. If the file already exists, the editor asks you whether to delete the same. Depending on the answer appropriate action is taken.

A range of line numbers ca be moved from one location to another by typing the command move m-n before/after p where m & n represent the range and p is the destination.

A range of lines can duplicated at some location by issuing the command dupliacte m-n before/after p where m & n represent the range and p is the destination.

You cannot load a file into the memory if some lines are existing in the memory. This can be overridden by issuing the command merge filename bfore/after lineno, where lineno is the destination line number.

You can use the help command if you want to see the list of commands.

You can see the data and time by typing the command time.

SECTION III

STRUCTURAL ANALYSIS OF THE TEXT EDITOR

HEADER FILES - A PREVIEW

C provides a library of routines for input/output , file manipulation , memory allocation/deallocation , string handling , DOS interfacing etc. These files can be viewed as a collection of predefined symbols and values which help to provide the various useful macros . This is the "header files" library .

These files are the backbone of C . These files contain the various functions which endow C with its reputed power , ease and flexibility . These files have a .h extension . The **.h** informs the compiler that the files are **header** files that contain definitions to be placed prior to "main()" .

The include files are provided along with the C package . However the user himself can create any Function which he feels is used often and can store it in his own "include" file . The include files are generally stored in a seperate Directory .

The syntax for accessing an "include" file is ->

#include <filename>

This tells the Preprocessor to load the contents of the text file "filename" as though it formed part of your .C file at that point . Once the file has been included , all of the routines it contains can be accessed throughout the code .

The various "include" files used in this package are discussed on the next page .

HEADER FILES USED IN THIS PACKAGE

<stdio.h>

This is the standard Input/Output include file for character I/O , stream handling and other I/O functions . It contains several definitions with which we must provide the compiler when we perform character I/O . If this file is not included , several of the definitions reqd. for character I/O cannot be resolved by the compiler , resulting in syntax errors .

<bios.h>

This include file acts as an interface between C and the BIOS of the PC . Many of its functions return useful BIOS information about the Memory , I/O Ports , Communications etc.

<dos.h>

This include file links C with the MS DOS . Using functions in this file we can make System Calls to DOS , allocate DOS memory segments , include command line arguments etc.

<string.h>

This very useful include file makes string handling relatively simple for C . The family of string manipulation functions made available by this file is invaluable . Functions like strcmp , strcpy & strlen make the otherwise rigid C very flexible .

<alloc.h>

This include file deals almost exclusively with Memory Mangement Functions . A good share of C's power comes from the functions in this file . Allocation of memory heaps , their management and handling are done by these functions .

<ctype.h>

This include file deals with characters . The functions in this file work on characters to convert them to ASCII or Uppercase or Lowercase etc.

<mem.h>

This file includes the important functions which load bytes (data) into/from the memory segments , manipulate memory arrays and make memory handling so simple .

STRUCTURAL ANALYSIS OF THE TEXT EDITOR PROGRAM

main()

This routine reads the input and decides the course of action .

First of all the Text Editor screen is set up . Then the "*" prompt is provided for the user to enter a command . The command is accepted into a character array "line[MAX_CHAR]" . It is then tested for suitable format and if found suitable , the next routine "searchcomd()" is called .

searchcomd()

This routine searches for a valid command .

Here the input string is first converted to Uppercase (C is reknowned for its Case sensitivity) . Then it is compared with the Table of available commands to see whether it is a valid command . Once the validity is established , the command is processed by the next routine "executecomd" , else "Command not recognised " is echoed onto the screen .

executecomd()

This module initiates the execution of the command .

This is the block where the input string is compared with individual commands . When a match is met , the corresponding action is initiated . The string function "strcmp" is used in this routine for comparison . The relevant command is executed by calling on the corresponding module . More details about this routine will be revealed when individual commands are discussed later on .

store()

This module stores the entered Text in the memory .

The number of lines of text inputted are kept track of by a variable "counter" . If the number of lines entered exceeds the limit "MAXLINE" (predefined) , then a warning is given that no further lines can be stored . If the number of lines is less than the maximum limit and available memory (out of the allocated memory) is sufficient , then the line is stored in the memory . Memory resident text is very essential for rapid processing and editing .

We have to first allocate memory for the text to be stored . Here we use the Dynamic Memory Allocation technique for keeping track of the text in the memory . We create a linked list to keep track of the memory segments . Initially a struct called "TEXT" is created . This has three elements =>

1. A pointer "fp" which is itself a struct of same type and which points to another struct of the same type . Here we use the concept of Self-referential structures .

2. An int variable "lineno" which will contain the line number of the input line .

3. A char array "string[MAX_CHAR]" which will contain the input line itself .

Hence we see that the Text is stored as individual lines with an associated lione number .

Initially we test to see whether memory has been allocated or not Incidently the struct variable "first" is of type "struct text" and it always points to the first line in the memory .

Case I : TOP == NULL =>

Memory is allocated by "malloc(sizeof(TEXT))" where sizeof(TEXT) gives the number of bytes to be allocated . If NULL is returned then we are out of memory and the user is suitably warned .

Once memory is allocated from the heap , we enter our line into the struct type "TEXT" . The member "p->string[j]" is the char array which stores each line . The line number "n" is stored in the member "p->lineno" . (The notation "p->lineno" is equivalent to "*p.lineno") .

Case II : TOP == NULL =>

This means we already have some text in the memory and hence we have to insert the new line into the existing text at the proper place corresponding to its line number . Now we test the input line number .

Case II.I : n < lineno of first line in memory

We have to insert this line before the first line currently in the memory . Hence we allocate memory and then attach this new line to the linked list as its first element . Now "first" points to this new line and this new line points to what was originally the first line .

Case II.II : n > or = lineno of first line

We scan the linked list to find out the position in which the line is to be inserted . A simple "for" loop increments the pointer "p" (by "p = p->fp") and we test each time to see either of the 3 cases =>

Case II.II.I : p -> lineno == null

This means we have reached the end of the text in the memory and hence we have to attach the new line at the end of this linked list . This is easily done by allocating memory and manipulating the pointers accordingly .

Case II.II.II : p->lineno == n

Here we are rewriting an existing line and hence no pointer manipulation is reqd.

Case II.II.III : p->lineno > n

Here we have to insert the new line just before the present line manipulation .

Note that at any stage if we are out of memory , thid routine returns the value "0" to the calling routine . Else it returns a value "1" or in the case of overwriting a line , a value "3" .

getime()

This module prints the date and time on the screen .

The two routines "getdate()" & "gettime()" are called by this module to do the actual execution .

gethelp()

This module displays the Help menu and provides various Help faclities .

We initially set up the Help Menu . This is done by opening the Help file "edhelp.hp" and simply echoing its contents onto the screen . Subsequently if more help is reqd. about the use and syntax of each command , then that is handled by jumping to the corresponding statements in the "executecomd()" routine .

savefile()

This saves the input text file onto the disk .

If the pointer TOP = NULL then we have no lines to be saved .

The name of the file to be saved is enterd into the variable array "filename[j]" Then we open this file for "read" mode and test to see if there already exists a file with the same name . Once this is verified then we open the file for "write" mode and the whole file is written onto the disk by the "fprintf" command included in a "for" loop which increments the pointer "p->fp" at each step .

loadfile()

This initiates loading of the requested file into the memry from the disk .

We first check to see whether there are any lines already in the memory . If there are they are to be deleted .

We then enter the name of the file to be loaded , which is stored in "filename[j]" . Then the next routine "ldfile[xxx]" is called

ldfile(xxx)

This routine actually does the loading of the file from the disk.

As we see the filename is passed to this rputine as a parameter from the previous routine "loadfile" . The requested file is opened by "fopen" and if valid , each line from the file is stored in the memory . This is achieved by a "while" loop where each time one line is entered into the variable "line" . An EOLN char is added to each line and then a line number (with an increment of ~ 10) is allotted to it . Subsequently the routine "store()" is called to store each line in the memory .

exeseq(int m1 , int n1)

This routine resequences the line numbers of all the lines in memory according to user's specifications .

Here the 2 parameters passed to the routine are : m1 = the initial line number ; n1 = the increment .

First we test to see that the increment is not more than 100 . Then we test to see that the final line number doesn't exceed 32000 (which is the line number limit) . Once these 2 checks are okay we proceed with the resequencing .

Starting from the first line in the linked list ; i.e. p = first.fp ; we increment p->lineno by the increment "n1" till we reach the end of the linked list (i.e. p->fp == NULL) or line number exceeds 32000 .

Finally we check to see whether the resequencing has been successful by testing if p->fp == NULL .

displine(int a , int b)

This displays text in the specified range of line numbers .

Here "a" and "b" specify the first and the last line number to be displayed .

We first test to see that a>= b is not true and also that there exists lines in memory (TOP == NULL not true) .Then we increment the linked list pointer "p" to point to the line with the line number "a" . Once the pointer is correctly positioned , we output the requisite number of lines onto the screen by "printf" command

display(int m)

This displays a specific line from the text .

Here "m" specifies the required line . If lines exist in the memory , the line pointer "p" is incremented till it points to the reqd. line . This line is then echoed onto the screen .

atoi()

This converts from ascii to integer .

If the input character is a digit between '0' and '9' then it is converted from its ascii code to integer value by the simple formula =>

integer value = ascii code of the number - ascii code for '0' .

listall()

This lists all the lines in the memory .

This is done by equating the pointer "p" to the "TOP" of the list and sequentially printing all the lines till we come to the last element (i.e. p->fp == NULL) .

deletel()

This routine deletes a line in the memory .

Case I :

If the line to be deleted (lineno == n) is the first member in the linked list , then we equate "r" to point to the second member . Then we call the routine "mfree(p)" to free that node from the linked list . Subsequently we decrease the "counter" by 1 and also change the pointer "first.fp" to point to the second element "r" .

Case II :
Here we have to scan the linked list to reach the reqd. line . Hence "p" is incremented till we have either of the 2 cases =>

Case II.I : p->lineno == n
Here we again call the routine "mfree(p)" to free the node "p' and adjust the pointers accordingly . Here we keeep two pointers => "l" to point to the previous member and "r" to point to the next .

Case II.II : p->lineno > n or p->fp == NULL
Here it is the case of a wrong line number and the user is suitably warned .

skipblank()

This ruotine deletes the blank spaces within a line .

delall()

This routine deletes all lines in the memory .

Here the pointer "p" is incremented in a "for" loop and each time the routine "mfree(p)" is called to free the node "p" . This procedure is carried on till we reach the end of the list . At the end we reset the "TOP" pointer and the "counter" .

delline(int a , int b)

This deletes lines in the given range .

Here "a" and "b" specify the first and last lines to be deleted .

The pointer "p" is incremented till the desired line is reached . Then a "while" loop is used to increment the pointer "p" from lineno "a" to "b" and each time call routine "mfree(p)" to free that node .

lowercase(char *s , int k)

This converts the string to uppercase .

Here we pass 2 parameters to the routine - "s" is a pointer to the string and "k" is the length of the string . Then we use a "for" loop and increment the pointer "s" each time converting one char to upper case using "toupper(c1)" .

instext(int baflag , int m1)

This allows the user to insert text without line number .

Here the parameter "baflag" tells us whether insertion is to be done before or after the line number given by "m1" .

First we test to see if "baflag == 0" (i.e. insertion to be before or after the specified line) and "m1" corresponds to the first line . If this is so , then an insertion is not possible .

Now we proceeed with the main block . There are 2 cases :

Case I : baflag == 0
Here we have to insert the text before the specified line number . Hence we advance the pointer "p" till it points to the line before line "m1" .Now "mid2" points to the line "m1" , "mid1" points to the previous line and "lastnum" contains the lineno of the line "mid1" .

Case II : baflag == 0
Here we have to insert the text after the specified line . Hence here pointer is advancedd till "mid1" points to line "m1" and "mid2" to next line .

Now we proceed with the insertion . Since we are entering lines without line numbers , we store these lines with temporary lineno and then at the end we resequence all the line numbers .

We get the line from the screen . Thenm we initialize the pointer "middle.fp" and allocate memory . Our first line to be inserted is given the line no "10" . The variable "curline" keeps track of the inserted lineno since it is incremented by 10 in each iteration .

Now we enter each line into the char array "p->string[i]" using an infinite "for" loop . We "break" from this loop only when we come to the end of the lines to be inserted . Care is taken to increment the "counter" and "p->lineno" is given the latest value of "curline" on each iteration .

At the end we have to adjust the pointers to include the inserted text at its proper place in the linked list . Now "middle.fp" points to the first inserted line . Hence we change the pointer "midl-> fp" to "midle.fp" . The pointer "l" points to the last inserted line . Hence we change "l->fp" to point to "mid2" .

Once this is over our insertion is complete . Now we call "exeseq(10,10)" to resequence all the line numbers .

movtext(int baflag , int ml , int nl , int pl)

This routine moves the text freom one location to the other according to user specifications .

Here "ml" & "nl" are the range of linenumbers to be moved and "pl" is the destination line number . "baflag" denotes the status "after" or "before" .

First we test the validity of the 3 parameters ml, nl, & pl . THen we call on routine "search()" to see whether the 3 line numbers exist . Now we maintain 3 flags :

1. tflag => this is set to TRUE if ml is the first line in the linked list . (i.e. TOP->lineno == ml)

2. iflag => this is set to TRUE if pl is the line just after nl . (i.e. p->lineno == nl ; p = p->fp ; p->lineno == pl)

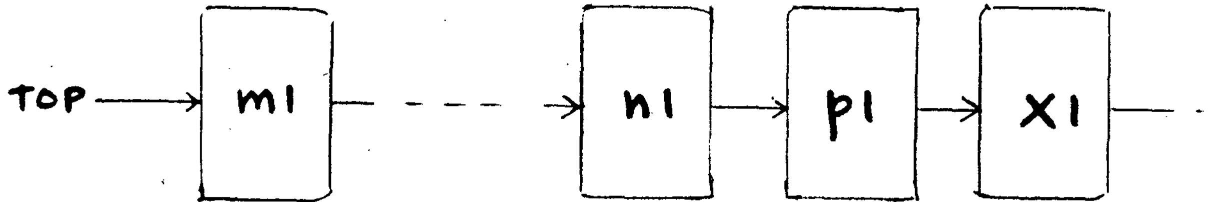
3. eflag => this is set to TRUE if nl is the last line in te linked list . (i.e. p->fp =NULL ; p->lineno == nl)

The 4th flag "baflag" is passed to this routine from the calling routine . However if we find that "iflag = TRUE " or pl is the first line in the linked list then we set this flag to "1" .

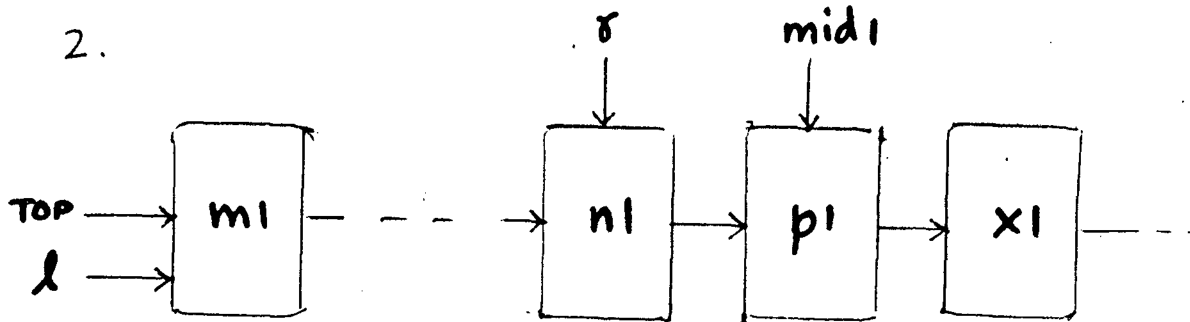
The variuos cases and how they are tackled are discussed using linked list representation diagrams in the following pages .

Ⓐ <u>IF tflag = TRUE ; iflag = TRUE</u>

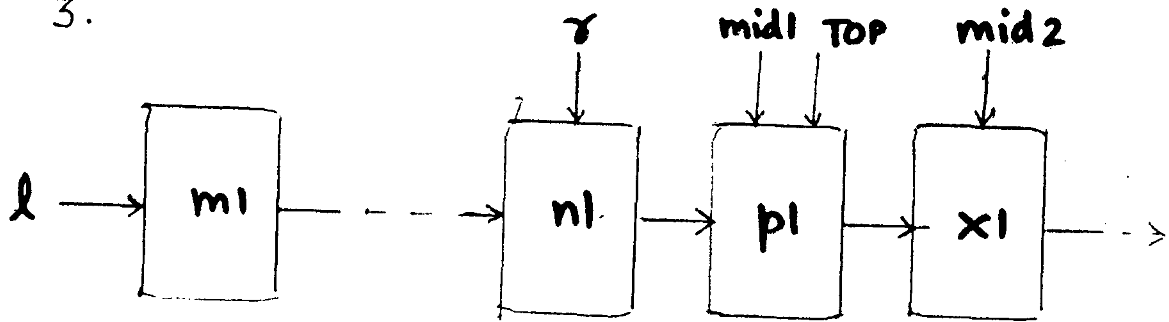
1. INITIAL



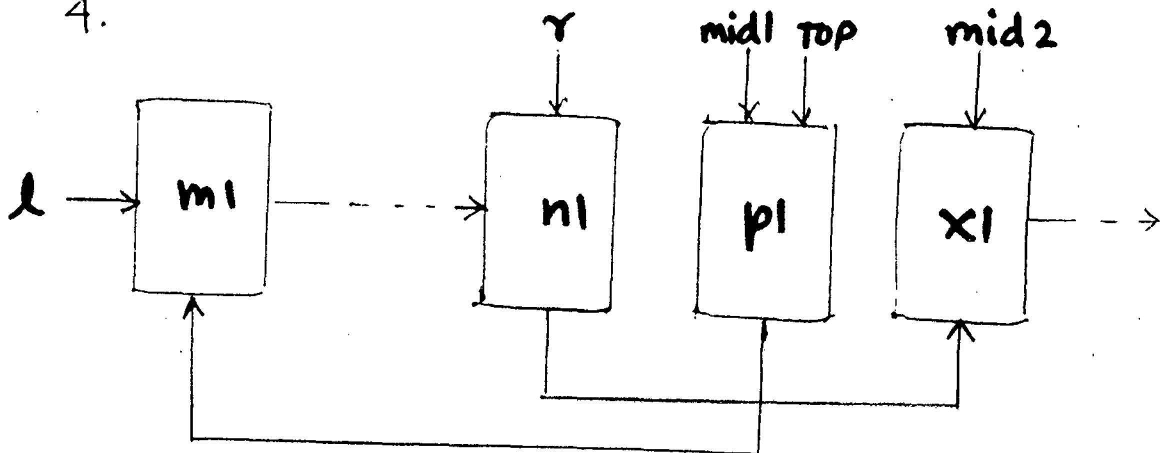
2.



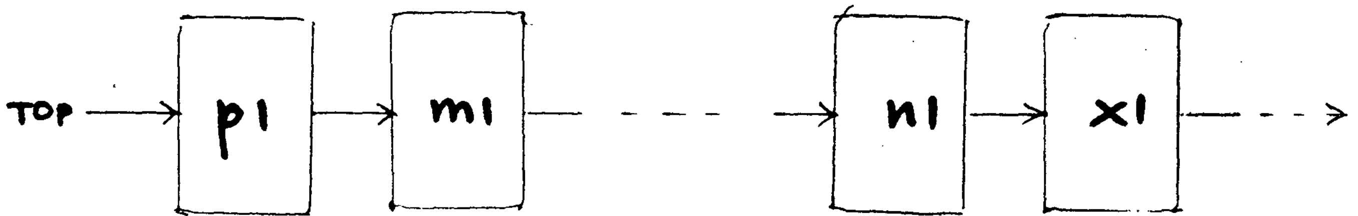
3.



4.

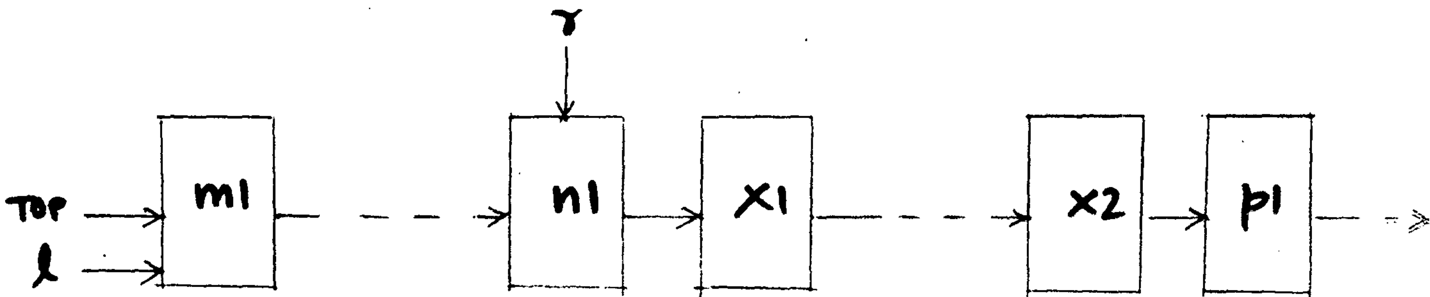


5. FINAL

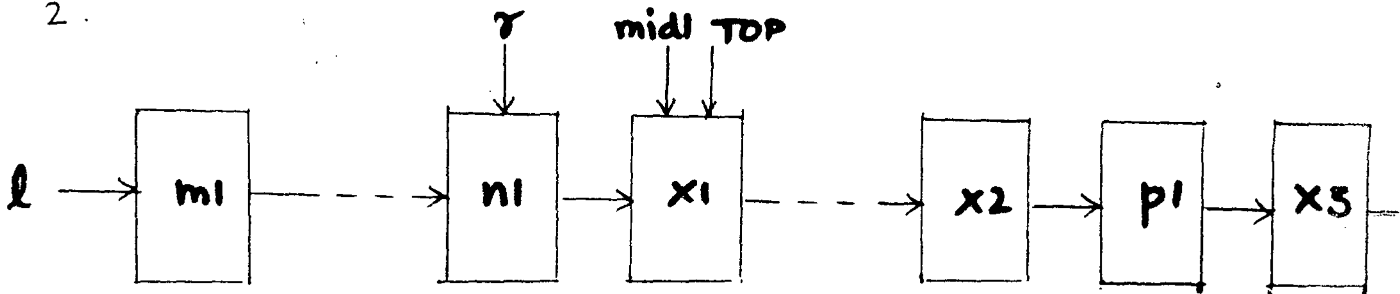


Ⓑ <u>IF $t_f lag$ = TRUE ; $i_f lag$ = FALSE</u>

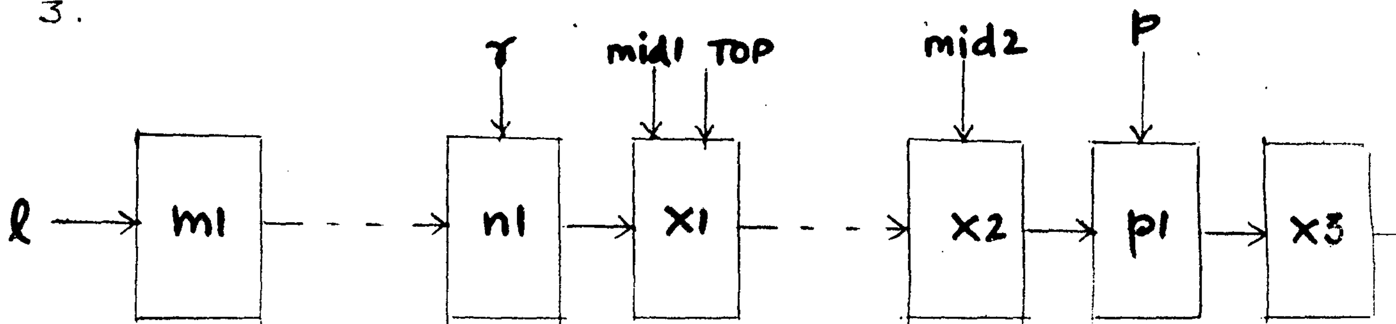
1. INITIAL



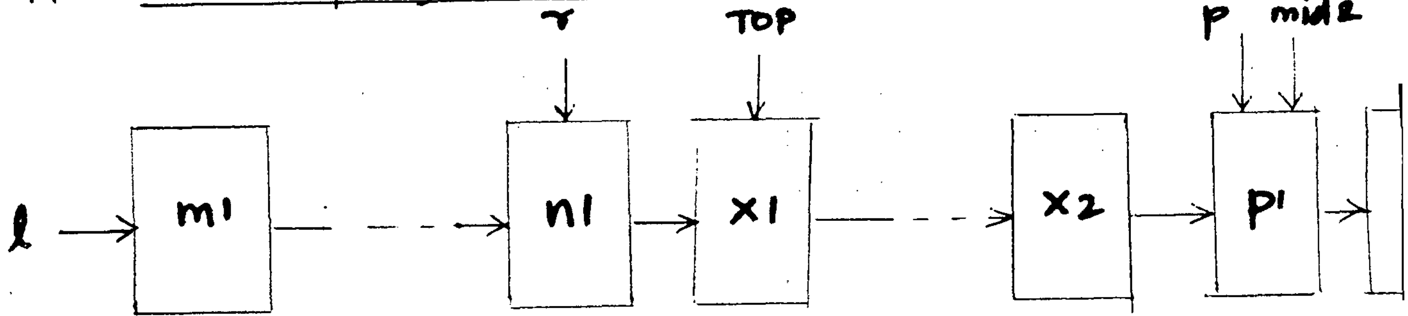
2.



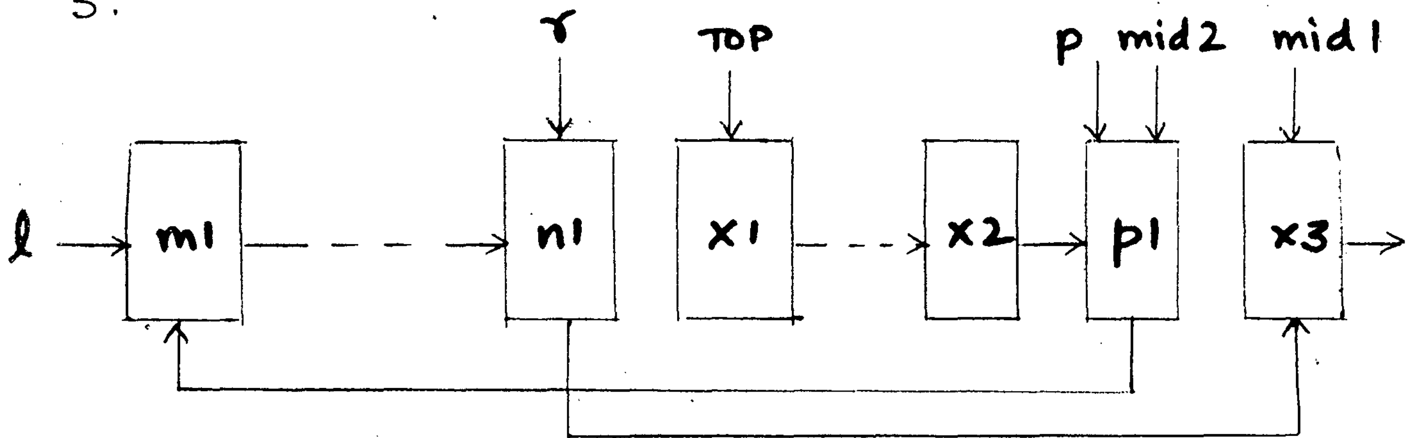
3.



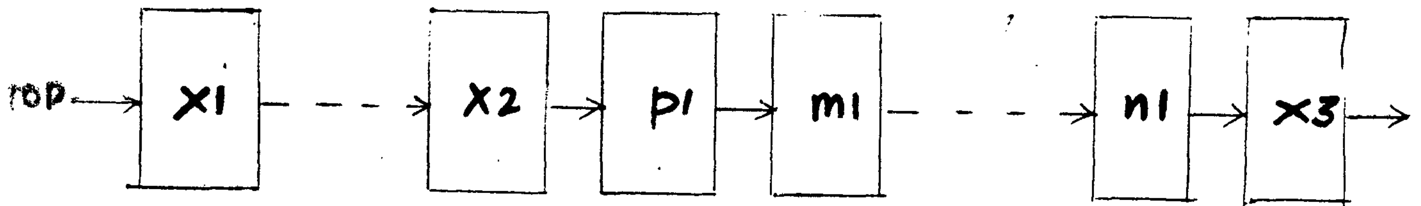
4. IF baflag = TRUE



5.

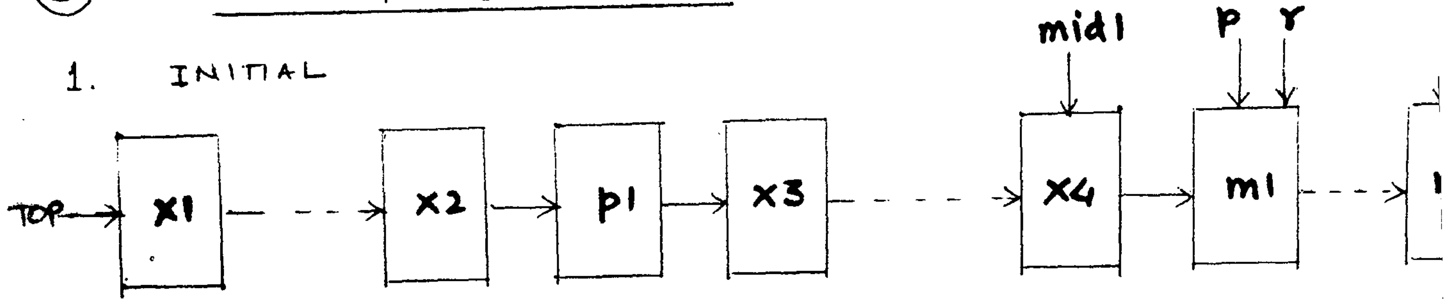


6. FINAL

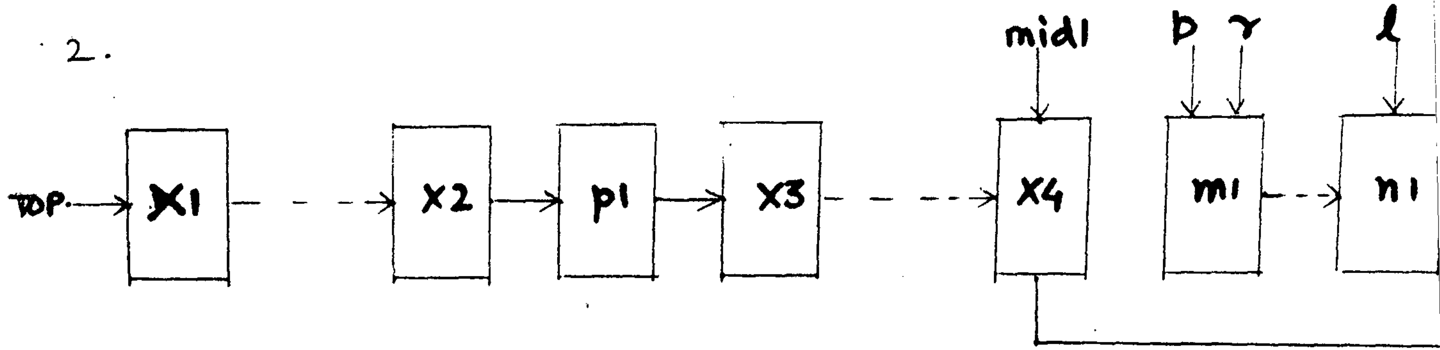


Ⓒ IF eflag = TRUE

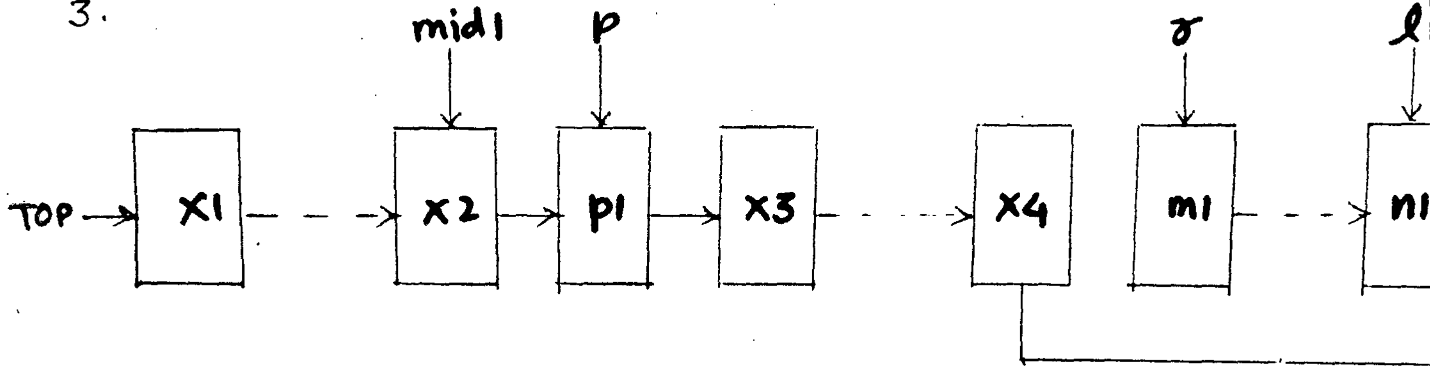
1. INITIAL



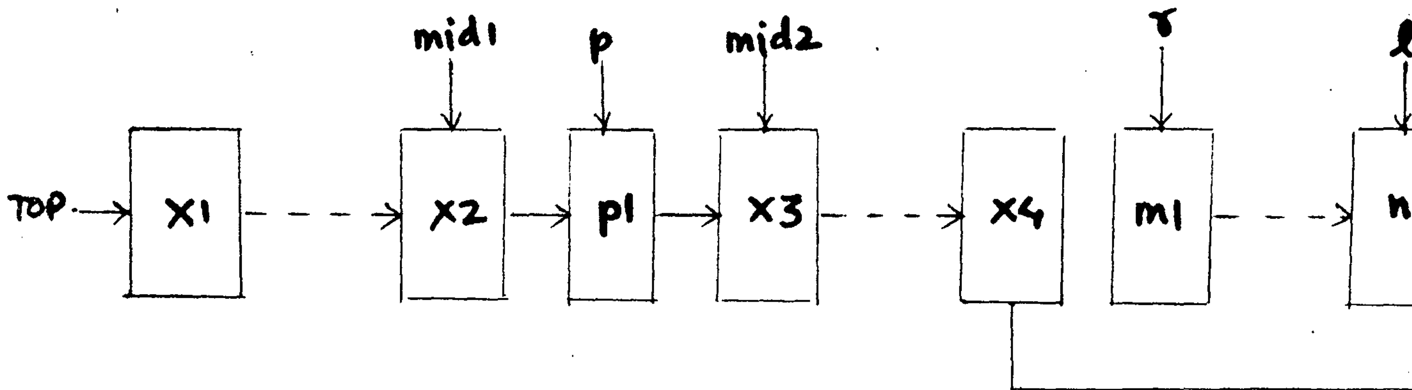
2.



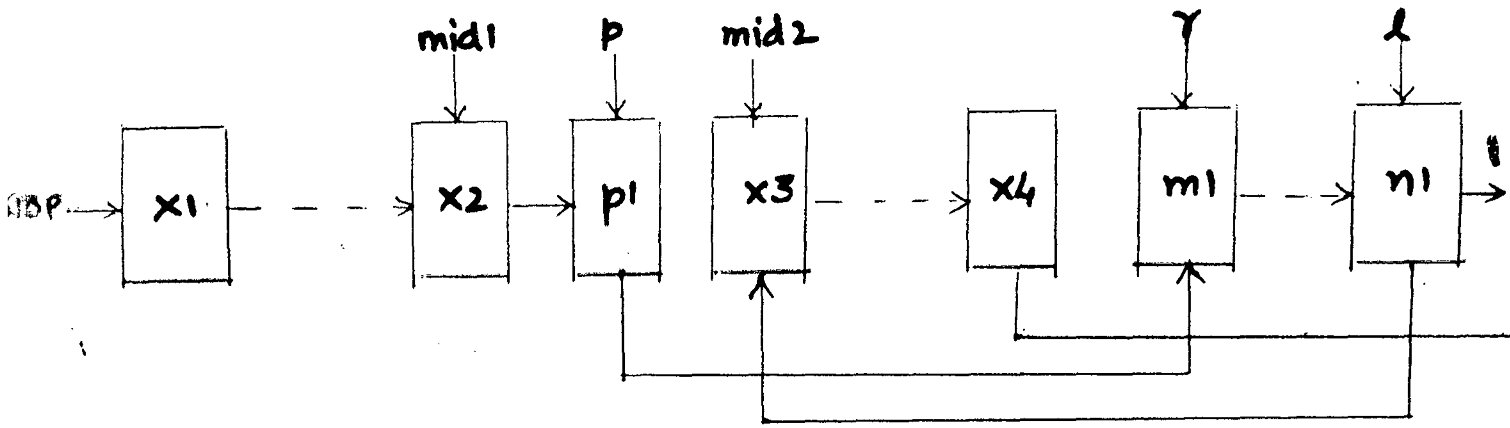
3.



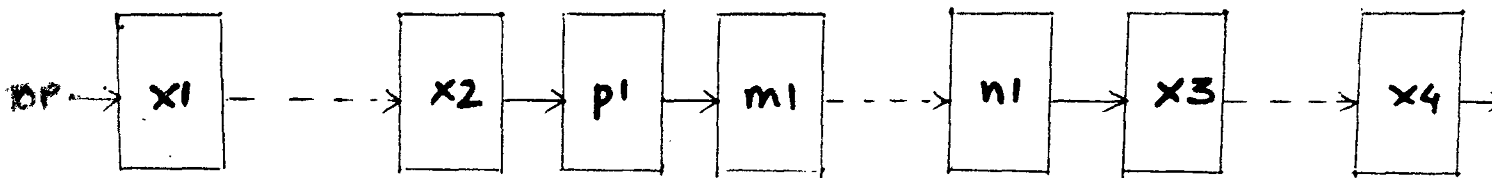
4. IF baflag = TRUE



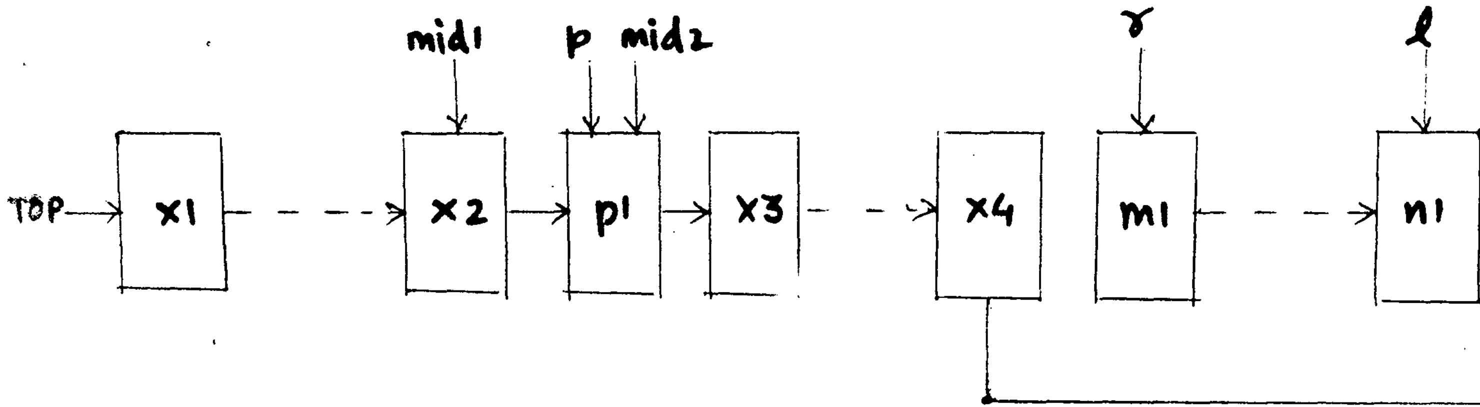
5.



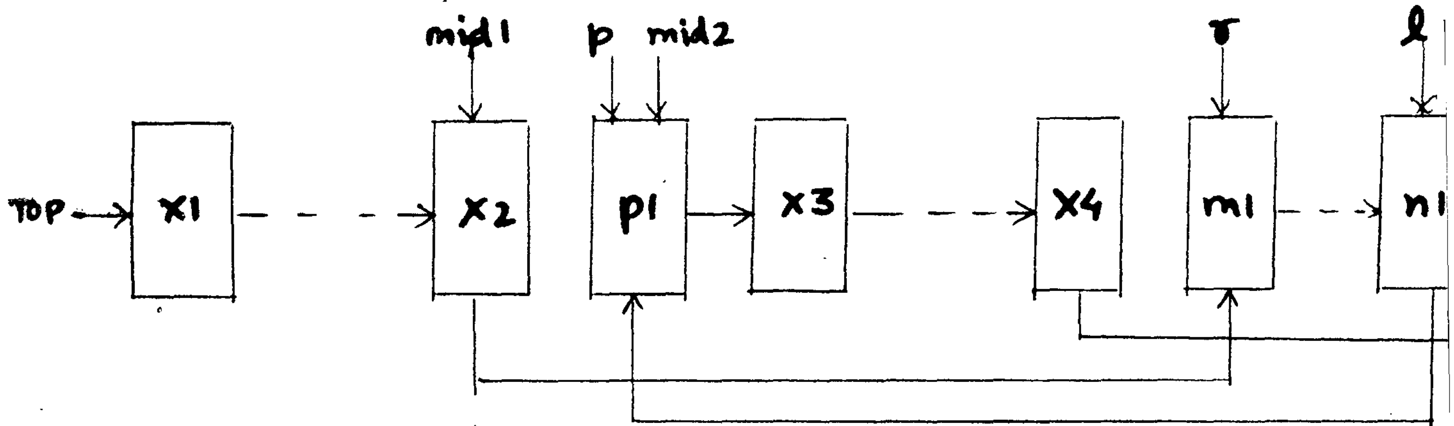
6. FINAL



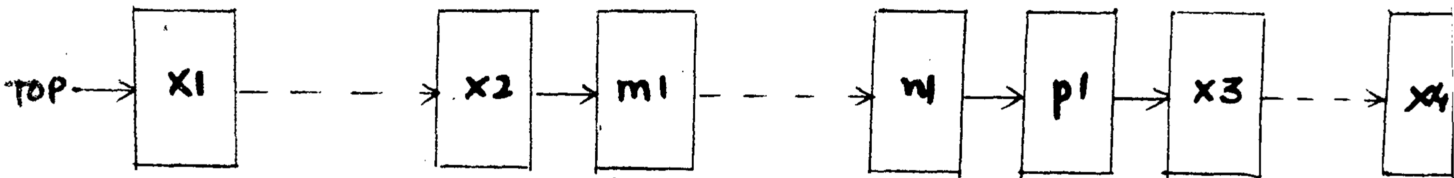
4. IF bafl ag = FALSE



5.

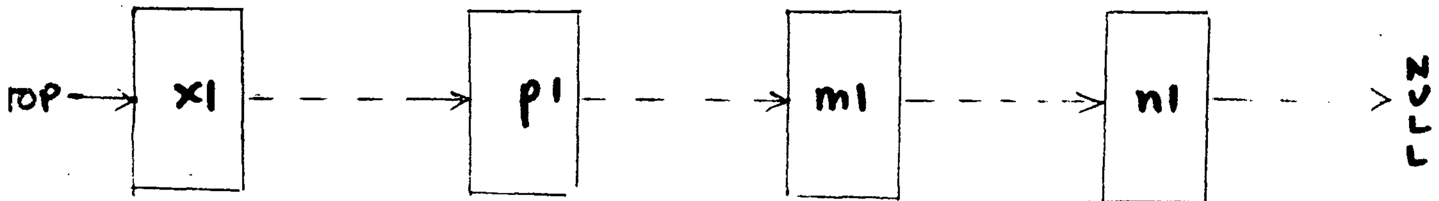


6. FINAL

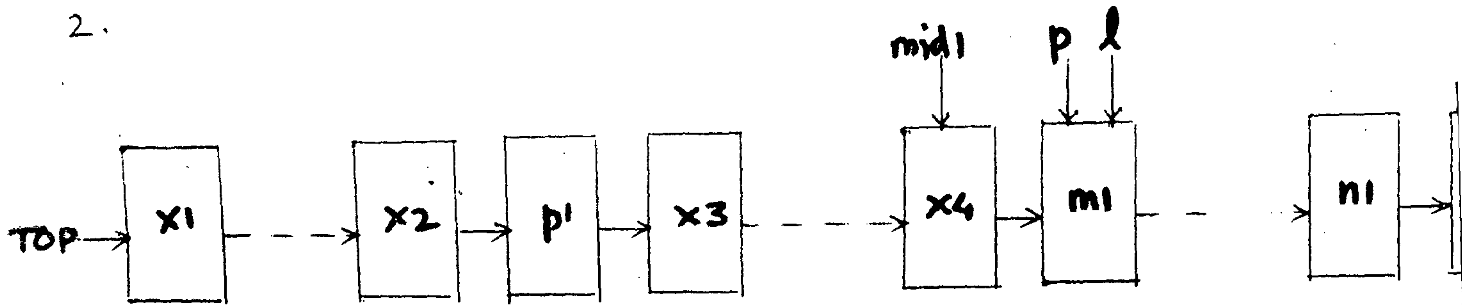


(D) MISCELLANEOUS CASE

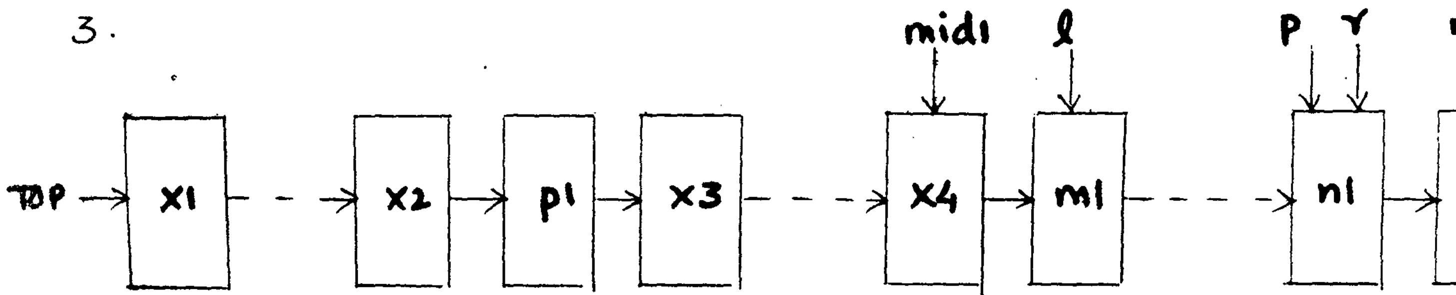
1. INITIAL



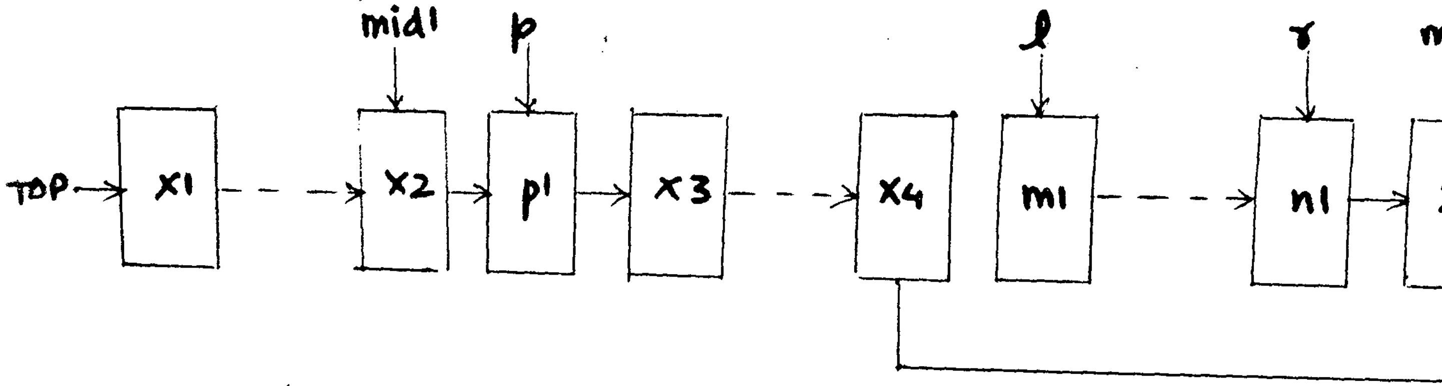
2.



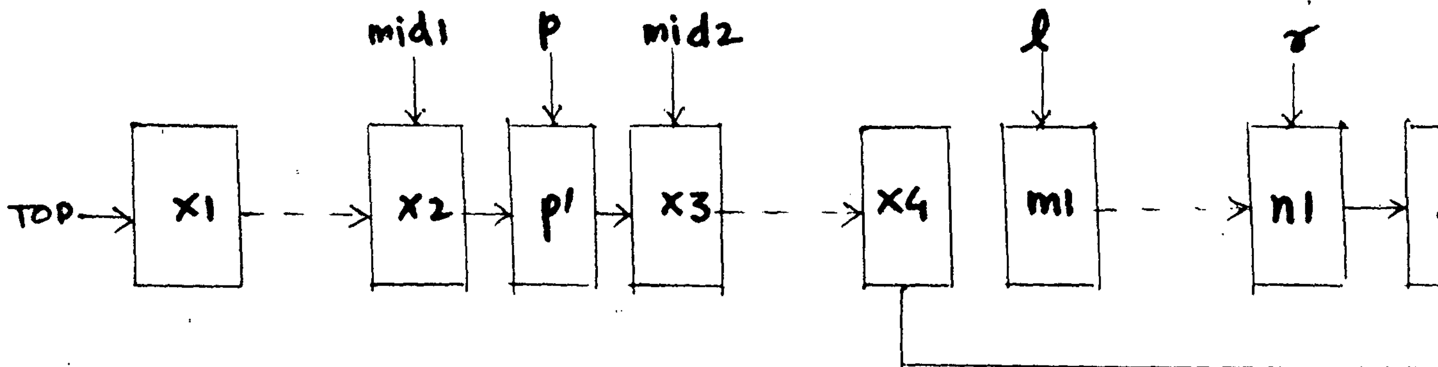
3.



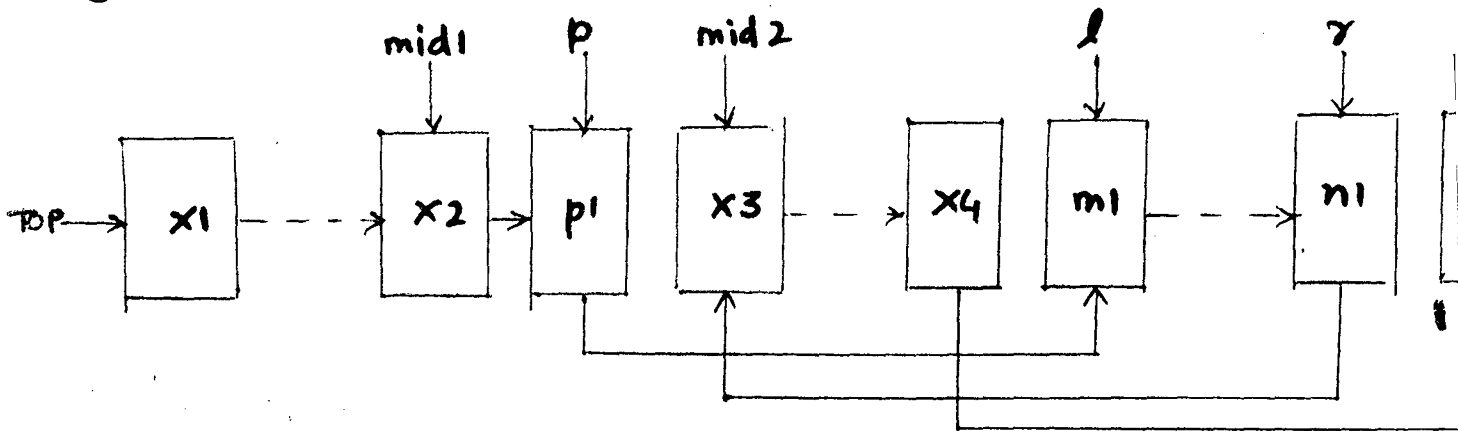
4. IF baflag = TRUE



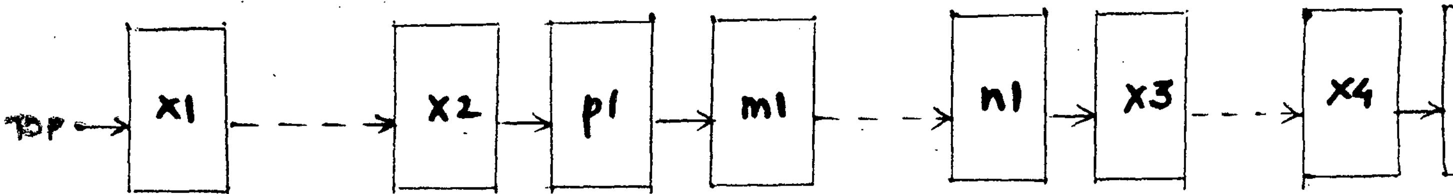
5.



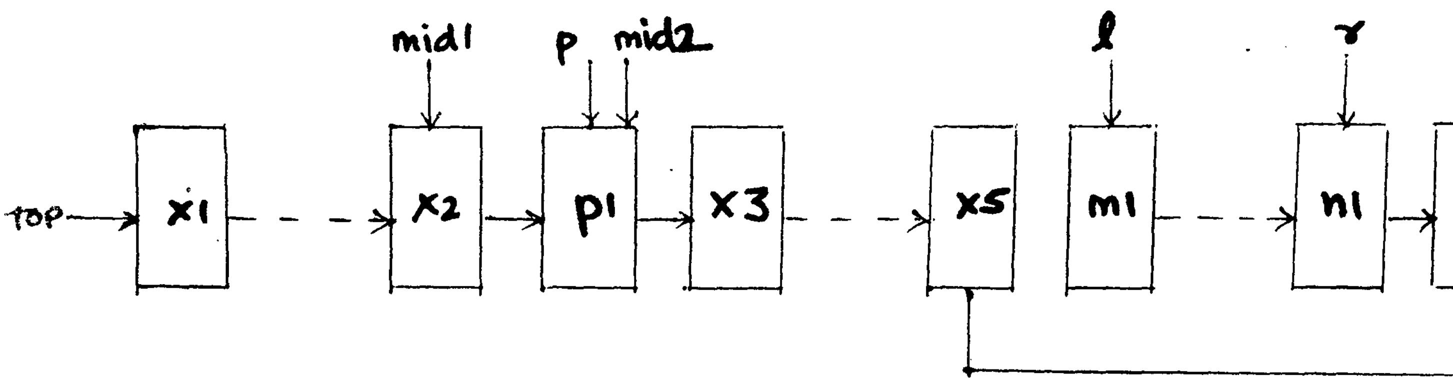
6.



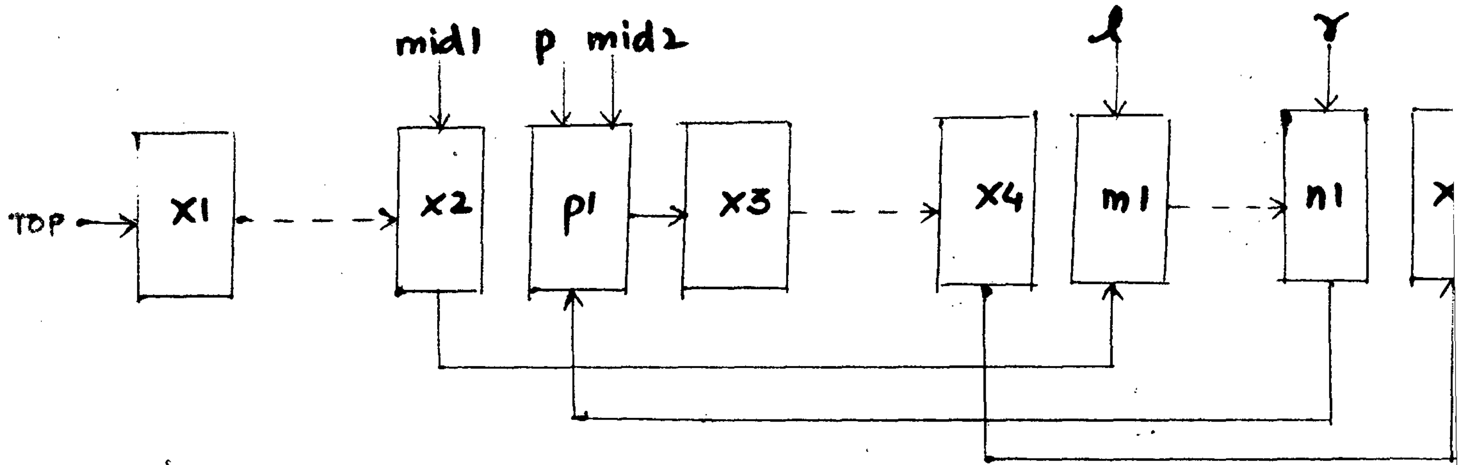
7. FINAL



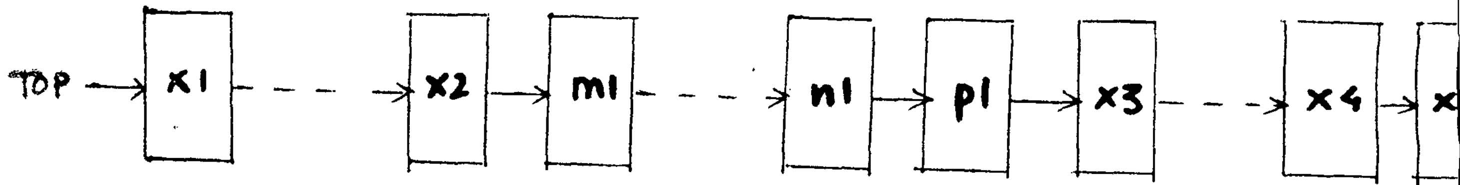
5. IF baflag = FALSE



6.



7. FINAL



findstring()

This finds a given string from the text and displays that particular line .

We first read the string to be searched in to the array "str2[x1]" . We then initialize the pointer "p" to TOP and scan all the lines by advancing the pointer by "p = p->fp" . On each iteration the string is compared using "strn_cmp()" and on successful comparison we display that particular line by calling routine "display(p->lineno)" .

nfindstring(int a1)

This routine searches for the nth occurrence of the specified string in the text and then displays that line .

This routine is similar to the previous one , except that here on first successful comparison we start a counter "y1" and increment it on each successful comparison . When "y1 == a1" we have found the nth occurrence of the string and hence call "display(p->lineno) to display that line .

nchng_string(int rept , int a1)

This replaces nth occurrence of a string ; within a range of line numbers ; with another string .

Here "rept" is the number of occurrences and "a1" is the line number .

We first read the string to be changed into the array "str1[x1]" and the string to be the replacement in the array "str2[x1]" . Now we scan the lines till we come to the line number a1 . Then we call on the next routine "nfind_replace()" to do the actual job. If the returned value of "rep_flag" is FALSE then we haven't found our string .

nfind_replace(int rept)

This finds the nth occurrence of the string and replaces it with another string .

We start scanning the line "a1" and use "strn_cmp()" function for comparison of the 2 strings . Once we find the string being searched , we start a counter "loop1" which counts the number of times that the comparison is successful . When we have "n" successful comparisons , we set "rep_flag" to TRUE . Then we exchange the searched string with the other string using the "strcat()" function .

chng_string(int a1 , int b1)

This replaces a string with another , within a range of line numbers .

Here the 2 parameters "a1" and "b1" specify the range of line numbers .

We first verify that the 2 line numbers are in proper sequence . Then we read the 2 strings into the arrays "str1" and "str2" . Now we advance the pointer "p" till we reach the line number a1 .

We now scan each line from "a1" to "b1" and each time call on the routine "find_replace()" to do the actual job of finding and replacing the specified string . If the returned value of "rep_flag" is FALSE , then our search has been unsuccessful .

find_replace()

This routine actually finds the specified string and replaces it with the other string .

Now this string is called to scan one line at a time . Hence here we scan the line "p->string[x1]" and use "strn_cmp()" for comparison of strings . If comparison is successful , we replace the found string with the other string using "strcat()" function and the intermediate array variable "reset(x3)" . This function returns "rep_flag" equal to TRUE if search and replacement is successful .

strn_cmp(out , in , count)

This compares the strings passed to it and returns the value "0" if successful .

Here "out" is the pointer to one string and "in" to the other . "count" contains the length of the first string . Now we set "flag1" to TRUE and compare the two strings "*out" and "*in" one cahr at a time . This is done by a "for" loop which increments as long as "a1 < count" . If all the characters of the 2 strings have matched (i.e. flag1 == TRUE) , then we return a value of "0" ; else we return a non zero value .

search(int a1)

This checks for the existence of the line number in the text .

Here the parameter "a1" is the line number which is to be searched . This is done by a simple "for" loop which searches for "p->lineno == a1" each time . A value TRUE is returned if search is successful .

append()

This routine lets you append lines at the bottom of the text .

We first advance the pointer "p" to the last line in the linked list and the variable "m1" is given the value of the last line number . Then using an infinite "while" loop we "gets(line)" and allot the line number "n = m1" to it . On each iteration "m1" is incremented by 10 and the line is appended to the linked list by calling the "store()" routine . This loop is exited by a "break" statement when there are no more lines to be appended .

duptext(int baflag, int m1, int n1, int p1)

This duplicates the text from one location to another .

This routine is similar to "movtext()" in most aspects except that here the text is copied and not moved . THe parameters passed here are the same as "movtext()" routine .

Here we maintain 2 flags :

1. tflag => This is set to TRUE if "p1" is the first line in the linked list .

2. eflag => This is set to TRUE if "p1" is the last line in the list .

Now we advance pointer "p1" till it points to the line with lineno equal to "m1" . The we allocate memory which is pointed to by the pointer "middle.fp" and "l" . This node is given the same line number as "m1" . We then use a "for" loop to copy the string from line "m1" one char at a time to the node "l" .

This procedure is then repeated for all line still line number "n1" . At the end "l" points to the last copied line (since each time we advance "l" by "l = l->fp") .

Now all we have to do is manipulate the pointers so that this copied text is placed at the right position . i.e. before or after line "p1" . This is accomplished by the same procedure as the one in routine "movtext()" .

mergfile()

This routine merges a file from disk with text in memory .

The input filename is read into the char array "filename[j]" (after "skipblank()") . Next we set the baflag to "1" if merging is to be done after and "0" if before the specified line number . Lastly we read the specified line number with the variable "k" .

Now we call on the next routine "nldfile()" to load the specified file from the disk and merge it with the text . After ,merging is successful we call on "exeseq()" to resequence the line numbers .

nldfile(baflag , xxx , lin)

This routine loads the specified file from the disk and merges it with the text .

The 3 parameters passed to this routine are "baflag" , "xxx" (filename) , and "lin" (line number) .

We first "fopen" the file xxx in read mode . Then we set "tflag" to TRUE if the specified line is the first line and set "eflag" to TRUE if the line is the last in the linked list .

Now we "fgets" each line from the open file using a "while" loop associated line number ("l->lineno = j") . The line is stored in the cahr array "l->string[i]" .

At the end when the whole file has been read , we manipulate the pointers to "merge" the file at the proper place .

EXECUTION PATH OF THE COMMANDS

EXIT => This is done by the function "exit(0)" .

LIST => First "skipblank()" is called to skip a blank . Then execution is according to 1 of the 3 cases .

LIST n : Here "display(j)" is called to list all lines after line number n .

LIST n-m : Here "displine(j,k)" is called to list all lines in range n to m .

LIST : Here "listall()" is called list all the lines .

TIME => Here routine "getime()" is called to display date and time .

FIND => Here the routine "findstring()" is called to find the specified string .

NFIND => Here the routine "nfindstring(j)" is called to find the nth occurrence of a string after line number "j" .

DEL => Here execution is according to 1 of 2 cases .

DEL n : Here we call "deletel()" to delete the line "n" .

DEL n-m : Here we call "delline(j,k)" to delete the lines from "n" to "m" .

SAVE => Here we call the routine "savefile()" to save the specified file in the disk .

RESEQ => Here we call the routine "exeseq(j,k)" to resequence the line numbers from line "j" to "k" .

INSERT => Here execution is according to 1 of 2 cases .

INSERT a n : Here set baflag to "1" and call on routine "instext(baflag,j)" to insert lines <u>after</u> line "j" .

INSERT n : Here baflag is kept as "0" and "instext(baflag,j)" is called to insert lines <u>before</u> line "j"

APPEND => Here the routine "append()" is called to append lines after the text in memory .

MOVE => Here we call the routine "movtext(baflag,j,k,rept)" to move the text between line numbers "j" and "k" to location before or after line number "rept" . There are 2 cases .

MOVE j k a n : Here baflag is set to "1" for <u>after</u>

MOVE j k n : Here we leave baflag as "0" for <u>before</u>

At the end "exeseq(10,10)" is called to resequence all line numbers .

DUPLICATE => THis command works in similar fashion , escept that here "duptext(baflag,j,k,rept)" routine is called to duplicate the text .

HELP =>Here the routine "gethelp()" is called to display the Help menu .

ERASE => Here the routine "delall()" is called to delete all the lines in the memory.

LOAD => Here the routine "loadfile()" is called to load the specified file from the disk into the memory .

MERGE => Here we call the routine "mergfile()" to merge the specified disk file with the text in memory .

NCHANGE => Here the routine "nchng_string(rept,j)" is called to replace the nth occurrence of a string after line "j" with another string .

CHANGE => Here the routine "chng_string(j,k)" is called to replace the specified string within a line range "j" to "k" with another string .

SECTION IV

CONCLUSION AND APPENDICES

CONCLUSION

The entire software of the word processor has been written in portable C. The total number of lines of code amounts to nearly 1500. One point to be noted here is that it is a line editor. It takes as input a line at a time. This is not a screen editor in which you take the cursor to the te text and do the modificatin. The idea behind writing in the line editing mode was to ensure portability. All screen based editors are actually device dependent as they use device specific instructions for the implementation. So a screen editor written for one machine may not run on some other machine. This is not the case with the text editor EDJNU . The coding has been made compact, device independent and flexible for future modifications. Hence this Wordprocessor can be easily ported from one machine to another with minimal changes . The compactness of the code also enables this editor to run on the smallest of machines . Special care has been taken to provide a very exhaustive Help facility for the ease of a new user .

All the best to the Users !

APPENDIX A

LIST OF COMMANDS

LIST : This commands lists the file which has been loaded into the memory .

EXIT : This command lets you exit to DOS .

TIME : This command displays the current date and time .

FIND : This command finds the specified string from your text and displays that particular line .

NFIND : This command finds the nth occurrence of the specified string and displays that line on the screen .

DEL : This command can delete a specific line ; or lines within a specific range .

SAVE : This command saves the text in memory in the specified disk file .

RESEQ : This command resequences all the line numbers according to user's specifications .

INSERT : This command allows you to insert lines without line numbers at a specified place in your text .

APPEND : This command allows you to append lines at the end of your current text .

MOVE : This command moves the text within a specified range of line numbers to another specified location in your text .

DUPLICATE : This command duplicates text within a specified range of line numbers in another specified location in the text .

HELP : This command displays the HELP Menu and prompts you for more help . If asked , it can display the list of all commands , their use and their syntax .

ERASE : This command deletes all the lines in the memory .

LOAD : This command loads the specified file from the disk into the memory .

MERGE : This command merges the specified disk (external) file with the text in memory at a specified location .

CHANGE : This command finds and replaces a specified string from the text , within a range of line numbers , with another string . Hence this combines the 2 commands FIND and REPLACE .

NCHANGE : This command finds the nth occurrence of a specified string , starting from a specified line number , and replaces it with another string . Hence this combines the commands NFIND and REPLACE .

APPENDIX B

LIST OF THE ROUTINES WRITTEN FOR THIS TEXT EDITOR

main ()

searchcomd()

executecomd()

store()

getime()

gethelp()

savefile()

loadfile()

ldfile()

exeseq()

displine()

display()

atoi()

listall()

deletel()

delline()

delall()

skipblank()

lowercase()

instext()

```
movtext()

nchng_string()

nfind_replace()

chng_string()

find_replace()

strn_cmp()

find_string()

nfind_string()

search()

append()

duptext()

mergfile()

nldfile()
```

APPENDIX C

BIBLIOGRAPHY

1. THE C PROGRAMMING LANGUAGE - DENNIS M. RITCHIE & BRIAN W. KERNIGHAN

2. PROGRAMMING IN C - KRIS A. JAMSA

3. UNDERSTANDING C - HUNTER

4. MASTERING TURBO C - STAN KELLY BOOTLE

5. MICROSOFT C COMPILER REFERNCE MANUAL - MICROSOFT CORPORATION

6. UNIX PRIMER PLUS - MICHAEL WAITE , DONALD MARTIN & STEPHEN PRATA

7. UNIX PROGRAMMER'S MANUAL - MASSACHUSETTS COMPUTER CORP.

8. FEATURES OF HIGH LEVEL LANG -UAGES FOR MICROPROCESSORS - A. C. DAVIES (MICROPROCESSORS MICROSYSTEM VOL II MAR 1987)

APPENDIX D

PROGRAM LISTING

```
/*  EDJNU - A PORTABLE WORDPROCESSOR  */

/** This software is being developed by  ANUPAM GOVIL
    as a part of his  MAJOR PROJECT for  fulfillment
    of Masters of Technology Degree in Computer Science
    at Jawaharlal Nehru University  , New Delhi **/

#include        <stdio.h>
#include        <bios.h>
#include        <dos.h>
#include        <string.h>
#include        <alloc.h>
#include        <ctype.h>
#include        <mem.h>
#define         MAXLINE         5000
#define         MAX_CHAR        132
#define         MIDDLE          middle.fp
#define         TOP             first.fp
#define         mfree           free
#define         TABLEN          19
#define         STRING_LIMIT    80
#define         TRUE            1
#define         EOLN            '\0'
#define         FALSE           0

/*              -----------------------------------------------------

                This Software acts as a  TEXT EDITOR. This has got a standard
                set of Commands. When any command is typed, appropriate
                action is taken.

                -----------------------------------------------------
**/

typedef struct  text    {
                struct  text    *fp;
                int     lineno;
                char    string[MAX_CHAR];
                }       TEXT  ;
TEXT    *p;
TEXT    first =  {NULL,0,'\0'};
TEXT    *l,*r,*midl,*mid2;
TEXT    middle = {NULL,0,'\0'};
char    *indblock[]     =     {
                "LIST",
                "DEL" ,
                "EXIT",
                "SAVE",
                "HELP",
```

```
                "CHANGE" ,
                "TIME" ,
                "LOAD"  ,
                "ERASE" ,
                "RESEQ" ,
                "NCHANGE" ,
                "FIND" ,
                "NFIND" ,
                "MOVE" ,
                "DUPLICATE" ,
                "INSERT" ,
                "APPEND" ,
                "MERGE" ,
                "\0" ,
                };
char    line[MAX_CHAR], line2[MAX_CHAR];
char    reset[MAX_CHAR];
int     i,n,y,counter;
int     rep_flag, set_quote_flag, set_help_flag ;
int     oldline[MAXLINE],newline[MAXLINE];
char    str1[STRING_LIMIT],str2[STRING_LIMIT];

main( )       /* This routine reads the input and decides the course of action */
{
        putchar(12);
        printf("\n\n  ***********************************************************************
        printf("\n\n\t\t    WELCOME TO THE JNU TEXT EDITOR\n");
        printf("\n\t\tDeveloped Exclusively for use at JNU , DELHI\n");
        printf("\n\n  **********************************************************************
        sleep(2);
        printf("     LIST OF COMMANDS IN THE REPERTOIRE \n");
        printf("  ------------------------------------\n ");
        printf("   LIST                            DEL    \n ");
        printf("   EXIT                            SAVE   \n ");
        printf("   HELP                            MOVE   \n ");
        printf("   TIME                            LOAD   \n ");
        printf("   ERASE                           RESEQ  \n ");
        printf("   CHANGE                          NCHANGE\n ");
        printf("   FIND                            NFIND  \n ");
        printf("   DUPLICATE                       INSERT \n ");
        printf("   APPEND                          MERGE  \n ");
        printf("  ------------------------------------\n ");
        printf("\n");
        counter = 0;
        set_quote_flag = FALSE ;
        set_help_flag = FALSE ;
        for ( ; ; )
        {
                printf("*");
                gets(line);
                for ( i = 0 ; line[i] == ' ' || line[i] == '\t' && line[i] != '\0'; i++)
                      ;
                if (line[i] != '\0' )
                  {  if(isdigit(line[i]))
                     {  n = atoi();
```

```
                        store();
                    }
                else
                        searchcomd();
                    ;
            }
}
searchcomd()  /* searches for the command */
{
                int     x;
                for (x=0; line[i] != '\t' && line[i] != '\0' && line[i] != '/'
                        && line[i] != '\x22' && line[i] != ' ';i++,x++)

                        ;
                lowercase(&line[i-x],x);
                for (y=0; y < TABLEN ; y++)
                        if (!strncmp(&line[i-x],indblock[y],x))
                        break;
                if (y >= TABLEN )
                        {
                                puts("** Command not Recognized **");
                                return;
                        }
                executecomd();
}

executecomd() /* This module executes the commands */
{
        int     j,k,rept,baflag;
        j = k = rept = baflag = 0;
        if (strcmp(indblock[y],"EXIT") == 0 )
        {
                if (set_help_flag == TRUE)
                   { printf("\n This lets you exit to DOS \n") ;
                     printf("\n Syntax => *EXIT \n");
                     set_help_flag = FALSE;
                     return;
                   };
                printf("** Thank You **\n");
                printf("\n\nJNU  TEXT  EDITOR  SYSTEM\n");
                sleep(1);
                exit(0);
        }
        if (strcmp(indblock[y],"LIST") == 0)
                   {    if (set_help_flag == TRUE )
                    { printf("\n This command lists the file which has been LOADed into the memo
                      printf("\n Syntax =>  *LIST  \n");
                      set_help_flag = FALSE ;
                      return ;
                    } ;
                skipblank();
                if (isdigit(line[i]))
                   {    j = atoi();
                        if (line[i] == ' ' || line[i] == EOLN )
                           {    display(j);
```

```
                      return;
                   }
               if (line[i] == '-')
                  { ++i;
                    k = atoi();
                    displine(j,k);
                    return;
                  }
           }
        listall();
        return;
}
if (strcmp(indblock[y],"TIME") == 0 )
   {    if (set_help_flag == TRUE)
           { printf("\n This gives you the time \n") ;
             printf("\n Syntax  =>  *TIME  \n");
             set_help_flag = FALSE;
             return;
           };
        getime();
        return;
   }
if (strcmp(indblock[y],"FIND") == 0 )
   {    if (set_help_flag == TRUE)
           { printf("\n This finds the specified string from your text \n") ;
             printf("\n Syntax  =>  *FIND xxx  \n");
             set_help_flag = FALSE;
             return;
           };
        skipblank();
        if (line[i] == '\x22')
           set_quote_flag = TRUE;
        ++i;
        findstring();
   }
if (strcmp(indblock[y],"NFIND") == 0 )
   {    if (set_help_flag == TRUE)
           { printf("\n This finds the nth occurrence of a specified string from your text \
             printf("\n Syntax  =>  *NFIND n xxx  \n");
             set_help_flag = FALSE;
             return;
           };
        skipblank();
        if (isdigit(line[i]))
           j = atoi();
        if (line[i] == '\x22')
           set_quote_flag = TRUE;
        ++i;
        nfindstring(j);
   }
if (strcmp(indblock[y],"DEL") == 0 )
   {    if (set_help_flag == TRUE)
           { printf("\n This can delete a specific line ; or lines within a specific range \
             printf(" or all the lines in your text \n");
             printf("\n Syntax  =>  * DEL n m  \n");
```

```
            set_help_flag = FALSE;
            return;
          };
        skipblank();
        if (isdigit(line[i]))
          {     j = atoi();
                if (line[i] == ' ' || line[i] == EOLN )
                  {     n=j;
                        deletel();
                        return;
                  }
                if (line[i] == '-')
                  {     ++i;
                        k = atoi();
                        delline(j,k);
                        return;
                  }
          }
         puts("** Error : Line numbers expected **");
   }
if (strcmp(indblock[y],"SAVE") == 0 )
   {    if (set_help_flag == TRUE)
          { printf("\n This saves the specified file in the disk  \n") ;
            printf("\n Syntax  =>  *SAVE filename  \n");
            set_help_flag = FALSE;
            return;
          };
        savefile();
        return;
   }
if (strcmp(indblock[y],"RESEQ") == 0 )
   {    if (set_help_flag == TRUE)
          { printf("\n This resequences all the line numbers in your text  \n") ;
            printf("\n Syntax  =>  *RESEQ n m    \n");
            set_help_flag = FALSE;
            return;
          };
        skipblank();
        j = atoi();
        i++;
        k = atoi();
        if ( j == 0 || k == 0 )
          {     puts("** Error : Command Ignored **");
                return;
          }
        exeseq(j,k);
        return;
   }
if (strcmp(indblock[y],"INSERT") == 0 )
   {    if (set_help_flag == TRUE)
          { printf("\n This allows the user to insert text without line numbers \n"
            printf("\n Syntax  =>  *INSERT a n   \n");
            set_help_flag = FALSE;
            return;
          };
```

```
        skipblank();
        if (line[i++] == 'a' || line[i++] == 'A')
           baflag = 1;
        while (!isdigit(line[i]))
              { if (line[i] == EOLN )
                  {     puts("** Line number expected **");
                        return;
                  }
                else
                ++i;
              }
        j = atoi();
        if ( j == 0 )
           {    puts("** Error : Command Ignored **");
                return;
           }
        instext(baflag,j);
        return;
    }
if (strcmp(indblock[y],"APPEND") == 0 )
    {   if (set_help_flag == TRUE)
           { printf("\n This allows the user to append lines at the end of the text\n
             printf("\n Syntax  =>  *APPEND   \n");
             set_help_flag = FALSE;
             return;
           };
        append();
    }
if (strcmp(indblock[y],"MOVE") == 0 )
    {   if (set_help_flag == TRUE)
           { printf("\n This moves the text from one location to other \n");
             printf("\n Syntax  =>  *MOVE n m a p \n");
             set_help_flag = FALSE;
             return;
           };
        skipblank();
        j = atoi();
        ++i;
        k = atoi();
        if ( j == 0 || k == 0)
           {    puts("** Error : Command Ignored **");
                return;
           }
        skipblank();
        if (line[i] == 'a' || line[i] == 'A')
           baflag = 1;
        while (!isdigit(line[++i]))
              if (line[i] == EOLN )
                 {      puts("** Command Ignored **");
                        return;
                 }
        rept = atoi();
        if (rept == 0 )
           {    puts("** Line number expected **");
                return;
```

```
            }
        movtext(baflag,j,k,rept);
        exeseq(10,10);
        return;
    }
if (strcmp(indblock[y],"DUPLICATE") == 0 )
    {   if (set_help_flag == TRUE)
            { printf("\n This duplicates text from one location to other \n");
              printf("\n Syntax  =>  *DUPLICATE n m a p \n");
              set_help_flag = FALSE;
              return;
            };
        skipblank();
        j = atoi();
        ++i;
        k = atoi();
        if ( j == 0 || k == 0)
            {    puts("** Error : Command Ignored **");
                 return;
            }
        skipblank();
        if (line[i] == 'a' || line[i] == 'A')
           baflag = 1;
        while (!isdigit(line[++i]))
              if (line[i] == EOLN  )
                 {       puts("** Command Ignored **");
                         return;
                 }
        rept = atoi();
        if (rept == 0 )
            {    puts("** Line number expected **");
                 return;
            }
        duptext(baflag,j,k,rept);
        exeseq(10,10);
        return;
    }
if (strcmp(indblock[y],"HELP") == 0 )
    {   gethelp();
        return;
    }
if (strcmp(indblock[y],"ERASE") == 0 )
    {   if (set_help_flag == TRUE)
            { printf("\n This erases all the lines in the memory \n");
              printf("\n Syntax  =>  *ERASE \n");
              set_help_flag = FALSE;
              return;
            };
        delall();
        return;
    }
if (strcmp(indblock[y],"LOAD") == 0 )
    {   if (set_help_flag == TRUE)
            { printf("\n This loads the specified file from the disk \n");
              printf("\n Syntax  =>  *LOAD filename \n");
```

```
            set_help_flag = FALSE;
            return;
            };
        loadfile();
        return;
    }
if (strcmp(indblock[y],"MERGE") == 0 )
    {   if (set_help_flag == TRUE)
            { printf("\n This merges a file from disk with text in memory \n");
              printf("\n Syntax  =>  *MERGE filename a n \n");
              set_help_flag = FALSE;
              return;
            };
        mergfile();
        return;
    }
if (strcmp(indblock[y],"NCHANGE") == 0)
    {   if (set_help_flag == TRUE)
            { printf("\n This changes the nth occurrence of a string \n");
              printf("\n Syntax  =>  *NCHANGE n m 'xxx' 'yyy' \n");
              set_help_flag = FALSE;
              return;
            };
        skipblank();
        if (isdigit(line[i]))
            rept = atoi();
        else
            return;
        if (line[i] == '\x2F' || line[i] == '\x22')
            i++;
        else
            return;
        if (isdigit(line[i]))
            j = atoi();
        if (line[i] == '\x22')
            set_quote_flag = TRUE;
        if (line[i] == '\x22' || line[i] == '/')
            i++;
        if ( j == 0 )
            { puts("** Cannot change String **");
              return;
            }
        nchng_string(rept,j);
        return;
    }

if (strcmp(indblock[y],"CHANGE") == 0 )
    {   if (set_help_flag == TRUE)
            { printf("\n This changes the specified string \n");
              printf("\n Syntax  =>  *CHANGE 'xxx' 'yyy' \n");
              set_help_flag = FALSE;
              return;
            };
        if (line[i] == '\x2F' || line[i] == '\x22')
            i++;
```

```
            if (isdigit(line[i]))
               j = atoi();
            if (line[i] == ',')
               {     i++;
                     if (isdigit(line[i]))
                        k = atoi();
               }
            else
               k = 0;
            if (line[i] == '\x22')
               set_quote_flag = TRUE;
            if (line[i] == '/' || line[i] == '\x22' )
               ++i;
            if (j == 0)
               {     puts("** Error : Cannot Change String **");
                     return;
               }
            chng_string(j,k);
            return;
         }

}

int     store() /*  This stores the text in memory */
{
        int j , flag;
        if (counter > MAXLINE )
           {     puts("** No of lines exceeding Limit : Cannot store");
                 return(2);
           }
        counter++;
        skipblank();
        if (TOP == NULL )
           {
                 first.fp = malloc(sizeof(TEXT));
                 if (first.fp == NULL )
                    {     printf("** Sorry : No Memory **");
                          return(1);
                    }
                 p = first.fp;
                 p->fp = NULL;
                 p->lineno = n;
                 for (j=0 ;line[i] != '\0';i++,j++)
                     p->string[j] = line[i];
                 p->string[j] = '\0';
                 l = first.fp;
           }
        else
           {
                 p = first.fp;
                 if (p->lineno > n )
                    {     r = p;
                          first.fp = malloc(sizeof(TEXT));
                          if (first.fp == NULL )
                             {     puts("** Sorry : No memory **");
```

```
                        return(1);

                p = first.fp;
                p->fp = r;
                p->lineno = n;
                for ( j=0 ; line[i] != EOLN ; i++,j++)
                    p->string[j] = line[i];
                p->string[j] = EOLN;
                return(0);
            }
        for (p=first.fp ; p->lineno < n ; p = p->fp )
            {   l = p;
                if (p->fp == NULL )
                    {   p->fp = malloc(sizeof(TEXT));
                        if (p->fp == NULL )
                            {   puts("** Sorry : NO memory **");
                                return(1);
                            }
                        p = p->fp;
                        p->fp = NULL;
                        p->lineno = n;
                        for (j=0;line[i] != EOLN ; i++,j++)
                            p->string[j] = line[i];
                        p->string[j] = EOLN;
                        return(0);
                    }
            }
        if (p->lineno == n )
            {   for (j=0;line[i] != EOLN; i++,j++)
                    p->string[j] = line[i];
                p->string[j] = EOLN;
                counter--;
                return(3);
            }
        if (p->lineno > n)
            {   r = p;
                l->fp = malloc(sizeof(TEXT));
                if (l->fp == NULL )
                    {   puts("** Sorry : No memory **");
                        return(1);
                    }
                p = l->fp;
                p->fp = r;
                p->lineno = n;
                for (j=0;line[i] != EOLN; i++,j++)
                    p->string[j] = line[i];
                p->string[j] = EOLN;
                return(0);
            }
    }
    return(0);
}

int getime()  /* This prints the data and time */
{
```

```
        struct  date    today;
        struct  time    now;
        getdate(&today);
        gettime(&now);
        printf("\tDATE : %d/%d/%d \tTIME = %02d:%02d:%02d.%02d\n",
                today.da_day,today.da_mon,today.da_year,now.ti_hour,now.ti_min,
                now.ti_sec,now.ti_hund);
        return;
}
int     gethelp()  /* displays the help menu  */
{
        FILE            *fp1;
        char            ch, yn2;
        fp1 = fopen("edhelp.hp","r");
        if (fp1 == NULL )
           {    puts("** File EDHELP.HP not found **");
                return;
           }
        puts("--------------------------------------------");
        while ((ch=getc(fp1)) != EOF )
                putchar(ch);
        puts("\n--------------------------------------------");
        fclose(fp1);
        puts("Do you want to know more about any Command ?");
        puts("Type Y or N");
        printf("\n_");
        yn2 = getchar();
        if ( yn2 == 'Y'|| yn2 == 'y')
           { set_help_flag = TRUE ;
             printf("\n#");
             gets(line);
             for (i = 0 ; line[i] == ' ' || line[i] == '\t' && line[i] != '\0' ; i++);
             if (line[i] != '\0' )
                { if (isdigit(line[i]))
                     { n = atoi();
                       store();
                     }
                  else
                     searchcomd();
                }
           }
        return;

}

int  savefile()  /*  This saves the file in the disk */
{
        char     filename[30];
        FILE     *fp1;
        char     ans;
        int     j,status;
        skipblank();
        if (TOP == NULL )
           {    puts("** No input line in memory **");
                return;
```

```
        for (j=0;line[i] != ' ' && line[i] != EOLN ; j++,i++)
            filename[j] = line[i];
        filename[j] = EOLN;
        status = FALSE;
        while (status == FALSE )
              { fp1 = fopen(&filename,"r");
                if (fp1 == NULL)
                   break;
                printf("File %s already exists : delete it ? (Y/N) : ",filename);
                ans = getche();
                if (ans == 'Y'|| ans == 'y')
                   break;
                printf("Enter the new name : ");
                gets(filename);
              }
        fp1 = fopen(filename,"w");
        if ( fp1 == NULL )
           {    printf("** Unable to open/create file : %s\n",filename);
                return;
           }
        for (p = first.fp ; p->fp != NULL ; p = p->fp )
            fprintf(fp1,"%s\n",p->string);
        fprintf(fp1,"%s\n",p->string);
        printf("\n** Text saved into file %s **\n",filename);
        printf("## Total numer of lines saved = %d ##\n",counter);
        fclose(fp1);
        return;
}

int  loadfile()    /* This loads the file into the memory */
{
        char      filename[30];
        int       j;
        skipblank();
        if (TOP != NULL )
           {    puts("** Delete all lines before loading the file **");
                return;
           }
        counter = 0;
        for (j=0;line[i] != ' ' && line[i] != EOLN ; i++,j++)
            filename[j] = line[i];
        filename[j] = EOLN;
        counter = 0;
        j = ldfile(filename);
        if ( j == 0 )
            return;
        printf("** O.K.\x07\x07 **\n");
        printf("\n## %d lines copied##\n",counter);
        return;
}

int     ldfile(xxx)  /* This loads a file from  disk */
char     xxx[];
{
```

```
        FILE      *fp1;
        int       j,m;
        fp1 = fopen(xxx,"r");
        if (fp1 == NULL )
            {     printf("** File %s not found\x07\x07**\n",xxx);
                  return(0);
            }
        j = 10;
        while (fgets(line,MAX_CHAR,fp1) != NULL )
              { i = 0;
                m = strlen(line);
                m--;
                line[m] = EOLN;
                skipblank();
                n = j;
                j += 10;
                store();
              }
        fclose(fp1);
        return(5);
}

int      exeseq(int m1,int n1)  /* resequencing line numbers */
{
        int       count1,count2;
        if (n1 > 100 )
            {     puts("** Increment cannot be more than 100 **");
                  return(0);
            }
        for (count1 = m1,count2=0 ;count2 < counter ; count1 += n1,count2++ )
            if (count1 > 32000 )
                {  puts("** New line numbers exceeding limit **");
                   return(0);
                }
        for (count1 = m1,p = first.fp ; count1 < 32100 && p->fp != NULL ;
                                  p = p->fp , count1 += n1 )
                p->lineno = count1;
        p->lineno =  count1;
        if (p->fp == NULL )
            {     printf("** Resequencing Over \x07\x07**\n");
                  return(0);
            }
        printf("** Resequencing Fails **\x07\x07**\n");
        return(0);
}

int    displine(int a ,int b)  /* displays text in the specified range */
{
        if (a >= b)
            {     puts("** Lines out of range : Try again **");
                  return;
            }
        if (TOP == NULL )
            {     puts("** No lines in memory **");
                  return;
```

```
            }
        p = TOP;
        for ( ;p->lineno < a ; p = p->fp)
            {   if (p->fp == NULL)
                    {   puts("** Line out of Limit **");
                        return;
                    }
            }
        for ( ; p->lineno >= a && p->lineno <= b ; p = p->fp)
            {   printf("%d %s\n",p->lineno,p->string);
                if (p->fp == NULL )
                    {   puts("** End of Storage **");
                        return;
                    }
            }
        if (p->fp != NULL )
           puts("** O K **");
        return;
}

int   display(int m)  /* diplays a line */
{
        if (TOP == NULL )
           {    puts("** No lines available **");
                return;
           }
        for (p = TOP ; p->lineno < m ; p = p->fp )
            {   if (p->fp == NULL )
                    {   puts("** Line not found *");
                        return;
                    }
            }
        if (p->lineno == m )
            printf("%d %s\n",p->lineno,p->string);
        else
            puts("** Line not found **");
        return;
}

int   atoi()   /* converts from ascii to integer */
{
        int   k;
        for (k=0;line[i] >= '0' && line[i] <= '9' ; ++i)
            k = 10*k + line[i] - '0';
        return(k);
}

int  listall()   /* Lists all line in the memory */
{
        if (TOP == NULL )
                puts("** No lines in memory **");
        else
            {
                p = TOP;
                while (p->fp != NULL )
```

```
                          { printf("%d %s\n",p->lineno,p->string);
                            p = p->fp ;
                          }
                    printf("%d %s\n",p->lineno,p->string);
                    puts("** O K **");
                }
        return;
}

int    deletel()  /* This deletes a line in the memory  */
{
        if (TOP == NULL )
            puts("** No lines in memory **");
        else
            {
              p = TOP;
              if (p->lineno == n)
                 { r = p->fp;
                   mfree(p);
                   counter--;
                   first.fp = r;
                   return;
                 }
               for (p=TOP; p->lineno != n ; p = p->fp )
                   { l = p;
                     if (p->lineno > n || p->fp == NULL )
                        { puts("** Line not found *");
                          return;
                        }
                   }
               if (p->lineno == n )
                  { r = p->fp;
                    mfree(p);
                    counter--;
                    l->fp = r;
                    puts("** O K **");
                  }
            }
        return;
}

int  skipblank()  /* deletes the white spaces */
{
        for ( ; line[i] == ' ' && line[i] != EOLN || line[i] == '\t';
                ++i)
                ;

}

int  delall()  /* This deletes all lines in the memory */
{
        int x1,x2;
        p = TOP;
        if (p->fp == NULL )
           {    mfree(p);
```

```
                TOP = NULL;
                puts("** O K **");
                return;
            }
        for ( p = TOP ; p->fp != NULL ; )
            {   l = p->fp;
                mfree(p);
                p = l;
            }
        mfree(p);
        TOP = NULL;
        counter = 0;
        puts("** O K **");
}

int  delline(int a , int b)     /* This deletes lines in the given range */
{
        int flagl;
        flagl = FALSE;
        if ( a >= b )
            {   puts("** Lines not in sequence **");
                return;
            }
        if (TOP == NULL )
            {   puts("** No lines in memory **");
                return;
            }
        l = TOP ;
        if (l->lineno >= a)
                flagl = TRUE;
        for (p = TOP ; p->lineno < a ; l = p , p = p->fp )
            { if (p->fp == NULL )
                  { puts("** Line out of range **");
                    return;
                  }
            }
        while (  p->lineno >= a && p->lineno <= b && p->fp != NULL)
              { r = p->fp;
                mfree(p);
                counter--;
                p = r;
              }
        if (p->lineno <= b)
            {   mfree(p);
                counter--;
                if (flagl == TRUE )
                        TOP = NULL ;
                else
                        l->fp = NULL ;
                puts("** O K **");
                return;
            }
        if (flagl == TRUE )
                TOP = p;
        else
```

```
                l->fp = p;
        puts("** O K **");
        return;
}

int     nchng_string(int rept,int a1)  /* Replaces n th occurence of a string */
{
        int x1;
        rep_flag = FALSE;
        if (set_quote_flag == TRUE )
                for (x1=0; line[i] != EOLN && line[i] != '\x22';i++,x1++ )
                        str1[x1] = line[i];
        else
                for(x1=0; line[i] != EOLN && line[i] != '/' ; i++,x1++)
                        str1[x1] = line[i];
        str1[x1] = EOLN;
        i++;
        if (set_quote_flag == TRUE )
                for (x1=0; line[i] != EOLN && line[i] != '\x22'; i++,x1++ )
                        str2[x1] = line[i];
        else
                for (x1 = 0; line[i] != EOLN && line[i] != '/'; i++,x1++)
                        str2[x1] = line[i];
        str2[x1] = EOLN;
        set_quote_flag = FALSE ;
        for ( p =  TOP ; p->fp != NULL && p->lineno != a1 ; p = p->fp )
                ;
        if (p->lineno != a1 )
           {    puts("** Line Number not found **");
                return;
           }
        nfind_replace(rept);
        if (rep_flag == FALSE )
                puts("** String not found **");
        return;
}

int nfind_replace(int rept)    /* Finds and replaces the string */
{       int  x1,x2,x3,a1,loop1;
        a1 = strlen(str1);
        for (x1=0,loop1=0; p->string[x1] != EOLN ; x1++ )
                if (strn_cmp(&p->string[x1],&str1,a1) == 0 )
                   { loop1++;
                     if (loop1 != rept)
                        continue;
                     rep_flag = TRUE ;
                     for (x3 = 0, x2 = x1+a1 ; p->string[x2] != EOLN ;  x2++,x3++ )
                          reset[x3] = p->string[x2];
                     reset[x3] = EOLN;
                     p->string[x1] = EOLN ;
                     strcat(p->string,str2);
                     strcat(p->string,reset);
                     return;
                   }
}
```

```
int    chng_string(int a1,int b1)  /* Replaces a string with another */
{
        int x1;
        if (a1    b1 && b1 != 0 )
           {     puts("** Lines not in sequence **");
                 return;
           }
        rep_flag = FALSE;
        if (set_quote_flag == TRUE )
                for (x1=0; line[i] != EOLN && line[i] != '\x22';i++,x1++ )
                        str1[x1] = line[i];
        else
                for(x1=0; line[i] != EOLN && line[i] != '/' ; i++,x1++)
                        str1[x1] = line[i];
        str1[x1] = EOLN;
        i++;
        if (set_quote_flag == TRUE )
                for (x1=0; line[i] != EOLN && line[i] != '\x22'; i++,x1++ )
                        str2[x1] = line[i];
        else
                for (x1 = 0; line[i] != EOLN && line[i] != '/'; i++,x1++)
                        str2[x1] = line[i];
        str2[x1] = EOLN;
        set_quote_flag = FALSE ;
        for(p = TOP ; p->fp != NULL && p->lineno < a1 ; p = p->fp )
                ;
        if (p->fp == NULL )
           {     puts("** Lines out of range **");
                 return;
           }
        if (b1 == 0 )
           { find_replace();
             if (rep_flag == FALSE )
                 puts("** String not found **");
             return;
           }
        for ( ; p->fp != NULL && p->lineno <= b1 ; p = p->fp )
                find_replace();
        if (p->lineno <= b1 )
                find_replace();
        if (rep_flag == FALSE )
                puts("** String not found **");
        return;
}

int  find_replace()   /* Finds and replaces the string */
{
        int  x1,x2,x3,a1;
        a1 = strlen(str1);
        for (x1=0; p->string[x1] != EOLN ; x1++ )
            if (strn_cmp(&p->string[x1],&str1,a1) == 0 )
                {        rep_flag = TRUE ;
                         for (x3 = 0, x2 = x1+a1 ; p->string[x2] != EOLN ; x2++,x3++ )
                             reset[x3] = p->string[x2];
```

```
                                reset[x3] = EOLN;
                                p->string[x1] = EOLN ;
                                strcat(p->string,str2);
                                strcat(p->string,reset);
                                break;
                        }
}

int     strn_cmp(out,in,count)    /* Compares two strings */
char *in,*out;
int  count;
{
        int     a1,flag1;
        flag1 = TRUE;
        for (a1=0 ; a1 < count ; a1++,out++,in++ )
                if (*out != *in )
                        flag1 = FALSE;
                if (flag1 ==  TRUE )
                        return(0);
                return(4);
}

int lowercase(char *s,int k)  /* converts the string to upper case */
{
        int x1,c1;
        for (x1=0; x1 < k ; x1++,s++ )
            {   c1 = *s;
                *s = toupper(c1);
            }
}

int   findstring()  /* finds the given string */
{
        int x1;
        if (set_quote_flag == TRUE )
                for (x1=0; line[i] != EOLN && line[i] != '\x22'; i++,x1++ )
                        str2[x1] = line[i];
        else
                for (x1 = 0; line[i] != EOLN && line[i] != '/'; i++,x1++)
                        str2[x1] = line[i];
        str2[x1] = EOLN;
        set_quote_flag = FALSE;
        for ( p = TOP ; p->fp != NULL ; p = p->fp )
            for (x1=0; p->string[x1] != EOLN ; x1++ )
                if (strn_cmp(&p->string[x1],&str2,strlen(str2)) == 0 )
                    {   display(p->lineno);
                        return;
                    }
            for (x1=0; p->string[x1] != EOLN ; x1++ )
                if (strn_cmp(&p->string[x1],&str2,strlen(str2)) == 0 )
                    {   display(p->lineno);
                        return;
                    }
            puts("** String not Found **");
        return;
```

```
.nt  nfindstring(int a1)   /* finds the given string */

        int x1,y1;
        if (set_quote_flag == TRUE )
                for (x1=0; line[i] != EOLN && line[i] != '\x22'; i++,x1++ )
                        str2[x1] = line[i];
        else
                for (x1 = 0; line[i] != EOLN && line[i] != '/'; i++,x1++)
                        str2[x1] = line[i];
        str2[x1] = EOLN;
        set_quote_flag = FALSE;
        y1 = 0;
        for ( p = TOP ; p->fp != NULL ; p = p->fp )
            for (x1=0; p->string[x1] != EOLN ; x1++ )
                if (strn_cmp(&p->string[x1],&str2,strlen(str2)) == 0 )
                    {       y1++;
                        if (y1 == a1)
                           { display(p->lineno);
                             return;
                           }
                    }
            for (x1=0; p->string[x1] != EOLN ; x1++ )
                if (strn_cmp(&p->string[x1],&str2,strlen(str2)) == 0 )
                    { y1++;
                      if (y1 == a1 )
                         { display(p->lineno);
                           return;
                         }
                    }
            puts("** String not Found **");
        return;

nt instext(int baflag,int m1)     /* This allows the user to insert without line number */

        int curline,incr,lastnum,cnt;
        curline = lastnum = 0;
        incr = 10;
        p = TOP;
        if (p->lineno == m1 && baflag == 0 )
           {     puts("** Insertion not before the beginning **");
                 return;
           }
        if (baflag == 0 )
           for ( p = TOP ; p->lineno < m1 && p->fp != NULL ; p = p->fp )
                { mid1 = p;
                  mid2 = p->fp ;
                  lastnum = p->lineno;
                }
        else
            for ( p = TOP ; p->lineno <= m1 && p->fp != NULL ; p = p->fp )
                { mid1 = p;
                  mid2 = p->fp ;
```

```
                lastnum = p->lineno;
            }
        curline = lastnum + incr;
        cnt = FALSE;
        middle.fp = NULL;
        gets(line);
        if (line[0] == 0 )
                return;
        middle.fp = malloc(sizeof(TEXT));
        p = middle.fp;
        p->lineno = curline;
        curline += incr;
        for ( i = 0; line[i] != EOLN ; i++ )
                p->string[i] = line[i];
        p->string[i] = EOLN;
        l = p;
        p->fp = NULL;
        ++counter;
        for ( ; cnt == FALSE ;  )
            {   gets(line);
                if (line[0] == EOLN )
                    break;
                p->fp = malloc(sizeof(TEXT));
                if (p->fp == NULL )
                    { puts("** Sorry : no memory **");
                      return;
                    }
                p = p->fp;
                p->lineno = curline;
                curline += incr;
                for ( i = 0; line[i] != EOLN ; i++ )
                    p->string[i] = line[i];
                p->string[i] = EOLN;
                p->fp = NULL;
                ++counter;
                l = p;
            }
        p = mid1;
        p->fp = middle.fp;
        p = p->fp;
        l->fp = mid2;
        exeseq(10,10);
        printf("\n\n** O. K. **\n");
}

int movtext(int baflag,int m1,int n1,int p1)  /* moves the text from one location to other */
{
        int tflag,eflag,iflag;
        tflag = iflag = eflag = FALSE;
        if (m1 >= n1 || p1 == n1 || p1 == m1 )
           {    puts("** Invalid Sequence **");
                return;
           }
        if (m1 <= p1 && n1 >= p1 )
           {    puts("** Invalid Sequence **");
```

```
        return;
      }
  if (search(m1) && search(n1) && search(p1))
        ;
  else
     {
        puts("** Line not found.**");
        return;
     }

  p = TOP;
  if (p->lineno == m1)
        tflag = TRUE;
  for(p=TOP;p->lineno != n1 ; p = p->fp )
        ;
  p = p->fp;
  if ( p->lineno == p1)
        iflag = TRUE;
  for (p = TOP ; p->fp != NULL ; p = p->fp )
        ;
  if (p->lineno == n1 )
        eflag = TRUE;
  if (iflag == TRUE )
        baflag = 1;
  p = TOP;
  if (p->lineno == p1 )
        baflag = 1;
  /* Extract the range */
  if (tflag == TRUE )
     {  l = TOP;
        for (p = TOP ;p->lineno != n1 ; p = p->fp )
            ;
        r = p;
        mid1 = r->fp;
        if (iflag == TRUE)
           {    TOP = r->fp;
                mid2 = mid1->fp;
                mid1->fp = l;
                r->fp = mid2;
                puts("** O K **");
                return;
           }
        else
           {
                TOP = mid1;
                for (p = TOP ; p->lineno != p1 ; p = p->fp )
                    mid2 = p;
                if (baflag == 1)
                   { mid2 = p;
                     mid1 =  mid2->fp;
                     mid2->fp = l;
                     r->fp = mid1;
                     puts("** O K **");
                     return;
                   }
```

```
                    else
                        {
                            mid1 = p->fp;
                            mid2->fp = l;
                            r->fp = p;
                            puts("** O K **");
                            return;
                        }
                }
    }

if (eflag == TRUE )
    {   for (p = TOP;p->lineno != m1 ; p = p->fp)
                mid1 = p;
        r = p;
        while ( p->fp != NULL )
                p = p->fp;
        l = p;
        mid1->fp = NULL;
        for (p = TOP ; p->lineno != p1 ; p = p->fp )
                mid1 = p;
        if (baflag == 1 )
            {   mid2 = p->fp;
                p->fp = r;
                l->fp = mid2;
                puts("** O K **");
                return;
            }
        else
            {
                mid2 = p;
                mid1->fp = r;
                l->fp = mid2;
                puts("** O K **");
                return;
            }
    }
mid1 = TOP;
if (p->lineno == p1 )
        baflag = 1;
for ( p = TOP; p->lineno != p1 ; p = p->fp )
    ;
if ( p->fp == NULL )
        baflag = FALSE;
for (p = TOP ; p->lineno != m1 ; p = p->fp )
        mid1 = p;
l = p;
for ( ; p->lineno != n1 ; p = p->fp )
        ;
r = p;
mid2 = p->fp;
mid1->fp = mid2;
for ( p = TOP ; p->lineno != p1 ; p = p->fp )
        mid1 = p;
if (baflag)
```

```
            {       mid2 = p->fp;
                    p->fp = 1;
                    r->fp = mid2;
                    puts("** O K **");
                    return;
            }
        else
            {
                    mid2 = p;
                    mid1->fp = 1;
                    r->fp = mid2;
                    puts("** O K **");
                    return;
            }
}

int search(int a1) /* This checks for the existence of the line number */
{
        for ( p = TOP; p->fp != NULL ; p = p->fp )
                if (p->lineno == a1)
                        return(TRUE);
        if (p->lineno == a1 )
                        return(TRUE);
        return(FALSE);
}

int append()  /* lets you add text */
{
        int m1,n1;
        m1 = n1 = 0;
        for ( p = TOP ; p->fp != NULL ; p = p->fp )
                ;
        m1 = p->lineno;
        n1 = 10;
        while (m1)
              { gets(line);
                if (line[0] == EOLN)
                        break;
                i = 0;
                m1 += n1;
                n = m1;
                store();
              }
        puts("** O K **");
}

int duptext(int baflag,int m1,int n1,int p1)  /* duplicates the text from one location to other */
{
        int tflag,eflag;
        tflag = eflag = FALSE;
        if (m1 >= n1 || p1 == n1 || p1 == m1 )
           {    puts("** Invalid Sequence **");
                return;
           }
        if (m1 <= p1 && n1 >= p1 )
```

```
    {   puts("** Invalid Sequence **");
        return;
    }
if (search(m1) && search(n1) && search(p1))
        ;
else
    {
        puts("** Line not found **");
        return;
    }

p = TOP;
if (p->lineno == p1)
        tflag = TRUE;
for (p = TOP ; p->fp != NULL ; p = p->fp )
        ;
if (p->lineno == p1 )
        eflag = TRUE;
if (tflag == TRUE )
        baflag = 1;
if (eflag == TRUE )
        baflag = FALSE;

for (p = TOP ; p->lineno != m1 ; p = p->fp )
        ;
middle.fp = malloc(sizeof(TEXT));
if (middle.fp == NULL )
    {   puts("** Sorry : No memory **");
        return;
    }
l = middle.fp;
l->lineno = p->lineno;
for ( i = 0 ; i < strlen(p->string) ; i++)
        l->string[i] = p->string[i];
l->string[i] = EOLN;
l->fp = NULL;
++counter;
for ( p = p->fp ; p->lineno <= n1 ; p = p->fp )
     {  l->fp = malloc(sizeof(TEXT));
        if (l->fp == NULL )
           { puts("** Sorry : No memory **");
             return;
           }
        l = l->fp;
        l->lineno = p->lineno;
        for ( i = 0 ; i < strlen(p->string) ; i++)
            l->string[i] = p->string[i];
        l->string[i] = EOLN;
        l->fp = NULL;
        ++counter;
     }
 for ( p = TOP ;  p->lineno != p1 ; p = p->fp )
        mid1 = p;
 if (baflag == 1)
     {  mid2 = p->fp;
```

```
                        mid1 = p;
                        mid1->fp = middle.fp;
                        l->fp = mid2;
                        puts("** O K **");
                        return;
                    }
                else
                    {
                        mid2 = p;
                        mid1->fp = middle.fp;
                        l->fp = mid2;
                        puts("** O K **");
                        return;
                    }
        }

int  mergfile()    /* This merges a file with text in memory */
{
        char      filename[30];
        int       j,k,baflag;
        baflag = k = 0;
        skipblank();
        for( j=0;line[i] != ' ' && line[i] != EOLN ; i++,j++)
                filename[j] = line[i];
        filename[j] = EOLN;
        skipblank();
        if (line[i] == 'A' || line[i] == 'a' )
                baflag = 1;
        while(!isdigit(line[++i]))
                if (line[i] == EOLN )
                    {     puts("** LIne number expected **");
                          return;
                    }
        k = atoi();
        j = nldfile(baflag,filename,k);
        if ( j == 0 )
            {     puts("** Could not merge**");
                  return;
            }
        exeseq(10,10);
        printf("\n** O.K.\x07\x07 **\n");
}

int     nldfile(baflag,xxx,lin)  /* This loads a file from       disk */
char     xxx[];
int     baflag,lin;
{
        FILE     *fp1;
        int      j,m,tflag,eflag;
        fp1 = fopen(xxx,"r");
        if (fp1 == NULL )
            {     printf("** File %s not found\x07\x07**\n",xxx);
                  return(0);
            }
        if (search(lin) == 0 )
```

```
    {     puts("** Line not found **");
          return(0);
    }
p = TOP;
if (p->lineno == lin)
          tflag = TRUE;
else
          tflag = FALSE;
for (p = TOP ; p->fp != NULL ; p = p->fp )
          ;
if (p->lineno == lin )
          eflag = TRUE;
else
          eflag = FALSE;
j = 10;
l = &middle;
while (fgets(line,MAX_CHAR,fp1) != NULL )
        { m = strlen(line);
          m--;
          line[m] = EOLN;
          l->fp = malloc(sizeof(TEXT));
          if (l->fp == NULL )
             {    puts("** Sorry : no memory **");
                  return(0);
             }
          l = l->fp;
          l->lineno = j;
          j += 10;
          for ( i = 0 ; line[i] != EOLN ; i++)
             l->string[i] = line[i];
          l->string[i] = EOLN;
          l->fp = NULL;
          ++counter;
        }
fclose(fp1);
if (tflag == TRUE && baflag == 0 )
    {     r = TOP;
          p = TOP =  middle.fp;
          while(p->fp != NULL)
                  p = p->fp;
          p->fp = r;
          return(5);
    }
if (eflag == TRUE && baflag == 1 )
    {     for (p=TOP;p->fp != NULL ; p = p->fp )
             ;
          p->fp = middle.fp;
          return(5);
     }
for (p=TOP; p->lineno != lin ; p = p->fp )
          mid1 = p;
if (baflag == 1)
    {     mid1 = p;
          mid2 = p->fp;
    }
```

```
else
    {
          mid2 = p;
    }
mid1->fp = middle.fp;
for (p=middle.fp ; p->fp != NULL ; p = p->fp )
          ;
p->fp = mid2;
return(5);
```