LINE EDITOR

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct node
{
 char line[80];
 struct node* next;
}node;

char fname[20];
FILE* fp;
int cnt,modified;
node *first,*last;

void create()
{
 char str[80];
 node* temp;
 do
   {
      fgets(str,80,fp);

      if(feof(fp))          //precaution for last line which prints twice
         break;

      temp=(node*)malloc(sizeof(node));
      strcpy(temp->line,str);
      temp->next=NULL;

      if(first==NULL)
         first= temp;
      else
         last->next=temp;

      last=temp;
      cnt++;
   }while(!feof(fp));
}
```

```
void append()
{
 char str[80];
 node* temp;

 printf("Enter (. to stop)\n");
 while(1)
       {
           fgets(str,80,stdin); //gets is dangerous to use.so use fgets. here stdin
                                                                  means keyboard
                                        //gets cant read \n. fgets reads \n also.
           if(strcasecmp(str,".\n")==0)  //on turbo use stricmp()
              break;

           temp=(node*)malloc(sizeof(node));
           strcpy(temp->line,str);
           temp->next=NULL;

           if(first==NULL)
                 first = temp;
           else
                 last->next = temp;

           last = temp;
           cnt++;
        }
 modified=1;
}
```



```
void save()
{
 node* p;
 fp=fopen(fname,"w");

 p=first;
 while(p!=NULL)
      {
       fprintf(fp,"%s",p->line);
       p=p->next;
       }
 printf("%s saved successfully...\n",fname);
 fclose(fp);
 modified=0;
}

node* findnode(int pos)
{
 node* p;
 int i=1;

 p=first;
 while(p!=NULL && i<pos)
      {
       p=p->next;
       i++;
      }
 return p;
}

void printnodes(int m,int n)
{
 node* p;
 int i=0;

 p=findnode(m);
 while(p!=NULL && i<=n-m)
      {
       printf("%d: %s",i+m,p->line);
       i++;
       p=p->next;
      }
}
```



```
void deletenodes(int m,int n)
{
 node *temp,*p;
 if(m>n || m<=0 || n<=0)
   {
    printf("\nInvalid");
     return;
   }

 if(n>cnt)
    n=cnt;

 if(m==1)
   {
    while(n>0)
          {
            temp=first;
            first=temp->next;
            temp->next=NULL;
            free(temp);
            cnt--;
            n--;
          }
   }
 else
    {
     p=findnode(m-1);
     while(n>=m)
            {
             temp=p->next;
             p->next=temp->next;
             temp->next=NULL;
             free(temp);
             cnt--;
             n--;
            }
      }
 p=first;              //set last pointer to last node(if last node get deleted)
 while(p->next!=NULL)
      p=p->next;
 last=p;
 modified=1;
}
```



```
void insertnodes(char str[80],int no)
{
 node *temp,*p;

 temp = (node*)malloc(sizeof(node));
 strcpy(temp->line,str);
 temp->next=NULL;

 if(first==NULL)  // if linkedlist is empty then can't be inserted
     printf("\nThere should be at least one node.So Append first");
 else if(no>cnt)  //if position is beyond no of nodes then insert at the end
           {
            last->next=temp;
            last=temp;
           }
       else if(no>0)   //insert at specific position
                 {
                     if(no==1)
                        {
                           temp->next=first;
                           first=temp;
                        }
                     else
                        {
                           p=findnode(no-1);
                           temp->next=p->next;
                           p->next=temp;
                        }
                 }
 cnt++;
 modified=1;
}
```



```
void copynodes(int x,int y,int z)
{
 node *p,*temp;

 if(x>cnt || y>cnt || z>cnt)
   {
    printf("\nInvalid position to be copied");
     return;
   }

 p=findnode(x);
 while(x<=y)
       {
           insertnodes(p->line,z);
           p=p->next;
           z++;
           x++;
      }
 modified=1;
}

void movenodes(int x,int y,int z)
{
 node *p;

 if(x>cnt || y>cnt || z>cnt)
   {
    printf("\nInvalid position to be moved");
     return;
   }

 copynodes(x,y,z);
 deletenodes(x,y);
}
```



```
void findpattern()
{
 node* p;
 int i=1,flag=0;
 char str[80];
 printf("\nEnter pattern");
 fgets(str,80,stdin);

 p=first;
 while(p!=NULL)
      {
           if(strstr(p->line,str))
             {
              printf("%d: %s",i,p->line);
              flag=1;
             }

           p=p->next;
           i++;
       }
 if(flag==0)
    printf("\nPattern not found");
}

void help()
{
 printf("\na for append");
 printf("\np for print all lines");
 printf("\np [source pos] [destination pos] for printing range of lines");
 printf("\ni [pos] for insert a new line to specific position");
 printf("\nd [pos] for deleting a specific position line");
 printf("\nd [source pos] [destination pos] for deleting a range of lines");
 printf("\nf for find a pattern");
 printf("\nc [from pos] [to pos] for copying a line from one pos to another pos");
 printf("\nc [src pos] [dest pos] [to pos] for copying range of lines to another pos");
 printf("\nm [from pos] [to pos] for moving a line from one position to another pos");
 printf("\nm [src pos] [dest pos] [to pos] for moving range of lines to another pos");
 printf("\ns for save file");
 printf("\nh for help");
}
```



```
int main(int argc, char* argv[])
{
 char cmd[20],t1,str[80],ch;
 int n,t2,t3,t4;
 if(argc==1)
   {
    printf("File name not given \n");
    printf("\nPlease enter file name:");
    fgets(fname,80,stdin);
    append();
   }
 else
   {
    strcpy(fname,argv[1]);
    fp=fopen(fname,"r");
    if(fp==NULL)
      {
       printf("File doesn't exist\n");
       append();
      }
    else
      {
       printf("\nFile exist");
       create();
       fclose(fp);
      }
   }
 printf("\nNumer of lines is %d",cnt);
 while(1)
      {
        printf("\n$ ");
        fgets(cmd,80,stdin);  //gets is dangerous to use

        n=sscanf(cmd,"%c%d%d%d",&t1,&t2,&t3,&t4);
        //split by space
        //inside any command we are not reading string
        //so no need to read in the form of string
        //first token is character and others are integers
        switch(t1)
          {
             case 'f': findpattern();  break;
             case 'h': help();         break;
             case 'a': append();       break;
```

```
            case 'p': if(n==1)
                        printnodes(1,cnt);  //print all lines
                      else
                        printnodes(t2,t3);  //print range of lines
                      break;
            case 's': if(modified)
                        save();
                      else
                        printf("\nFile is already saved:");
                      break;
            case 'i': if(n==2)
                       {
                        printf("\nEnter text:");
                        fgets(str,80,stdin);
                        insertnodes(str,t2);
                       }
                      break;
            case 'd': if(n==2)
                        deletenodes(t2,t2);  //delete single line
                      else
                        deletenodes(t2,t3);  //delete range of lines
                      break;
            case 'c': if(n==3)
                        copynodes(t2,t2,t3);   //copy one line to another pos
                      else
                        copynodes(t2,t3,t4);   //copy range of lines to another pos
                      break;
            case 'm': if(n==3)
                        movenodes(t2,t2,t3);  //move one line to another pos
                      else
                        movenodes(t2,t3,t4);  //move range of lines to another pos
                      break;
            case 'q': if(modified)
                       {
                        printf("\n%s not saved. Save y/n?",fname);
                        ch=getchar();
                        if(ch=='y' || ch=='Y')
                          save();
                       }  exit(0);
            default : printf("\nInvalid choice");
        }
    }
}
```

