Práctica 9
Un sencillo editor de línea

Introducción

Los editores de texto son programas que nos permiten escribir y modificar archivos digitales compuestos únicamente por texto sin formato, conocidos como archivos de texto. Un archivo de texto puede modelarse como una secuencia (un conjunto ordenado de caracteres). El proceso de edición es un dialogo interactivo entre el usuario y la computadora diseñado para completar 4 tareas: i*)* Seleccionar la parte del documento de texto a desplegar y manipular, *ii*) Determinar como formatear la vista y como desplegarla, *iii*) Especificar y ejecutar operaciones para modificar el documento (inserción, borrado, encontrar texto, entre otras) y *iv*) Actualizar la vista adecuadamente.

Los tipos de editores son: *a*) De línea, las tareas de edición se realizan línea a línea, *b*) De pantalla, las tareas de edición se realizan en cualquier parte del contenido desplegado en la ventana de edición y *c*) Gráficos, del tipo WYSIWYG (what you see is what you get), más útiles para procesadores de texto que para editores de texto.

Descripción de un sencillo editor de línea

El editor de línea usado en esta práctica puede manejar hasta 25 líneas de texto. Cada línea puede contener hasta 80 caracteres y termina con el Retorno de Carro (CR, Carriage Return). Utiliza un arreglo de caracteres para guardar el texto. Los comandos del editor son los siguientes:

Comando	F u n c i ó n
`P`	Despliega las 25 líneas del texto
`Px`	Despliega la línea `x` (`x` debe ser un número entre 1 y 25)
`Sx,y`	Intercambia las líneas `x` y `y`. `x` y `y` deben ser números entre 1 y 25)
`Nx`	Crea una nueva línea `x` (`x` es un número entre 1 y 25)
`Dx`	Borra la línea `x`
`F%cadena%`	Busca la ocurrencia de la `cadena` en las 25 líneas de texto. Despliega el número de línea y el texto de la línea donde encuentra las ocurrencias.
`Q`	Sale del editor.

1.- Actividades

1.1.- Captura el código fuente del editor de línea `edlin` y genera su ejecutable.

```
/* Archivo: edlin.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char car1,texto[27][80],car2[30],n1[5],n2[5];
int i,line1,line2;

main()
{
    printf("-");
    car1=getchar();
    gets(car2);

    while( (car1 != 'q') || (strcmp(car2,"") != 0) )
    {
         switch (car1)
         {
              case 'p':
```

```
     if (strcmp(car2,"") == 0) {
          for(i=1;i<26;i++)
               printf("\n%i:%s",i,texto[i]);
    printf("\n");
 }
     else {
          line1=atoi(car2);
          if (line1 < 1 || line1 > 25)
               printf("Linea fuera de rango\n");
          else
               printf("%d:%s\n",line1,texto[line1]);
     }
     break;

case 'd':
     line1=atoi(car2);

     if(line1 < 1 || line1 > 25)
          printf("Linea fuera de rango\n");
     else {
          strcpy(texto[line1],"");
          printf("Linea borrada\n");
     }
     break;

case 'n':
     line1=atoi(car2);

     if(line1 < 1 || line1 > 25)
          printf("Linea fuera de rango\n");
     else {
     printf("%d:",line1);
     gets(texto[26]);
     strcpy(texto[line1],texto[26]);
     strcpy(texto[26],"");
     }
     break;

case 's':
     strcpy(n1,"");
     strcpy(n2,"");
     strncat(n1,&car2[0],1);
     strncat(n1,&car2[1],1);
     line1=atoi(n1);
     strncat(n2,&car2[3],1);
     strncat(n2,&car2[4],1);
     line2=atoi(n2);
     if(line1 < 1) {
          printf("Primera linea fuera de rango\n");
          break;
     }
     if(line2 > 25) {
          printf("Seguna linea fuera de rango\n");
          break;
     }
     else {
          strcpy(texto[26],texto[line1]);
          strcpy(texto[line1],texto[line2]);
          strcpy(texto[line2],texto[26]);
        printf("Intercambiadas las lineas %d y %d\n",line1,line2);
     }
     break;

case 'f':
```

```
                    strcpy(texto[26],"");
                    line2=strlen(car2);
                    if(car2[0] == '%' && car2[line2-1] == '%') {
                         for(i=1;i < line2-1;i++)
                              strncat(texto[26],&car2[i],1);
                         for(i=1;i<26;i++)
                              if( strstr(texto[i],texto[26]) )
                                   printf("%i:%s\n",i,texto[i]);
                    }
                    else
                         printf("Error en los (%)\n");
                    break;

               default:
                    printf("Comando invalido\n");
          }
          printf("-");
          car1=getchar();
          gets(car2);
     }
     return 0;
}
```

1.2.- Captura OK: _____ Compilación: OK ___ Tamaño `edlin.exe`: _____

2.- Verificar funcionamiento

2.1.- Desde una ventana de DOS (Símbolo del Sistema) prueba el funcionamiento de `edlin` ingresando las siguientes líneas: (Nota: para ejecutar el editor sólo tecleas `edlin<enter>`)

N1: Hoy es viernes.
N2: Esta es una prueba del uso de este sencillo editor.
N3: Esta es la tercera línea.
N4: Esta es la línea N4.
N5: Este es la última línea de este bloque.

2.2.- Ahora verificamos el funcionamiento de las siguientes opciones:

Opción	Verificada	
`P`	_____	
`P3`	_____	
`D4`	_____	(Después de ejecutar esta opción despliega el archivo completo para verificar)
`S01,05`	_____	(Después de ejecutar esta opción despliega el archivo completo para verificar)
`F%linea%`	_____	
`Q`	_____	

3.- Análisis del editor

3.1.- Funciones de texto usadas en el editor.

Apoyándote en la ayuda del IDE de Borland, investiga y describe brevemente que hace cada una de las siguientes funciones usadas en `edlin`:

Función	Descripción
`getchar()`	_____
	_____
`gets()`	_____
	_____

`strcmp()` _____

_____

`strcpy()` _____

_____

`strncat()` _____

_____

`strlen()` _____

_____

`strstr()` _____

_____

4.- Agregados al editor.

4.1.- Este editor carece del manejo de archivos. Las funciones faltantes son lectura y escritura de archivos, que consisten en copiar al arreglo desde un archivo y guardar el arreglo en un archivo, respectivamente. Usando la ayuda del Borland IDE investiga el uso de las siguientes funciones para el manejo de archivos:

Función Descripción

`open()` _____

_____

`close()` _____

_____

`read()` _____

_____

`write()` _____

_____

4.2.- Utilizando las funciones anteriores, agrega las siguientes funciones a `edlin`:

Comando	Función
`Rarchivo`	Lee el `archivo`, esto es, copia de `archivo` al arreglo `texto[]`
`Warchivo`	Escribe el arreglo `texto[]` completo a `archivo`.

4.3.- Código agregado para implementar las funciones `Rarchivo` y `Warchivo:`

_____

_____

_____

_____

_____

_____

5.- Comentarios y conclusiones

_____

_____

_____

_____

_____

_____

_____

_____