Programación con Memoria Dinámica

Navegador de Archivos de Texto

I. Objetivo de la actividad

La evaluación extra-aúlica del parcial, en la materia de Programación con Memoria Dinámica (PMD), tiene por objetivo medir su capacidad de resolución de problemas computacionales, incluyendo algunos o todos los temas que se han abordado en el curso; además, se pretende evidenciar su aprendizaje en programación utilizando el lenguaje C, así como sus habilidades de investigación y capacidad del trabajo en equipo.

II. Descripción del problema

En esta actividad, su objetivo es crear un programa que permita al usuario consultar el contenido de archivos de texto y navegar entre el histórico de archivos que ha consultado (reciente, anterior, siguiente). Para implementar la navegación entre archivos del histórico deberá hacer uso de los contenedores que hemos abordado hasta este momento en el curso de programación con memoria dinámica.

Antes de iniciar con el análisis de los requerimientos funcionales, la construcción de la propuesta de solución y su implementación, es conveniente conocer sobre el manejo de archivos en el lenguaje C.

2.1 Manejo de archivos de texto en C

Hay dos tipos de archivos, archivos de texto y archivos binarios. Un archivo de texto es una secuencia de caracteres organizadas en líneas finalizadas por un carácter de nueva línea. Los archivos de texto se caracterizan por ser planos, es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho. Un archivo de texto puede ser creado con editores como el bloc de notas de windows, VS Code, nano, gedit, etc.

En C podemos realizar operaciones para crear, abrir, leer y escribir archivos. Veamos cómo realizar cada una de las operaciones sobre archivos de texto.

2.1.1 Apertura de un archivo de texto

Antes de poder leer o escribir texto en un archivo, es necesario abrir el archivo o crearlo. Para esto, en C podemos utilizar la función **fopen()**. La sintaxis es la siguiente:

```
FILE* pf;
pf = fopen (“nombre del archivo”, "modo de apertura");
```

Donde pf es el apuntador de tipo FILE, “nombre del archivo” es el nombre del archivo que queremos abrir o crear. Este nombre debe ir encerrado entre comillas. También podemos especificar la ruta donde se encuentra o utilizar una cadena de caracteres que contenga el nombre del archivo. Veamos algunos ejemplos:

```
pf=fopen("datos.txt","r");
```

```
pf=fopen("c:\\txt\\saludo.txt","w");
```

El Segundo parámetro “modo de apertura” que recibe la función **fopen()** permite indicar en qué modo queremos abrir el archivo. Para archivos de texto tenemos los siguientes modos de apertura:

Modo	Descripción
w	crea un archivo para escritura. Si ya existe lo crea de nuevo
w+	crea un archivo para lectura y escritura. Si ya existe lo crea de nuevo
a	abre o crea un archivo para añadir datos al final del mismo
a+	abre o crea un archivo para leer y añadir datos al final del mismo
r	abre un archivo para lectura
r+	abre un archivo para lectura y escritura

La función **fopen()** retorna, un apuntador de tipo FILE. Si al intentar abrir el archivo se genera un error (por ejemplo si no existe el archivo y lo tratamos de abrir en modo lectura), la función fopen() devolvería NULL. Por esta razón es mejor controlar las posibles causas de error a la hora de programar. Por ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
	FILE *pf;
	pf=fopen("datos.txt","r");
	if (pf == NULL)
		printf("Error al abrir el fichero");
	return 0;
}
```

2.1.2 Cierre de un archivo

Una vez que hemos acabado de leer o escribir en un archivo es recomendable cerrarlo. Para cerrar los ficheros utilizaremos la función **fclose( )**. Esta función cierra el archivo, cuyo apuntador indicamos como parámetro. Por ejemplo:

```
fclose(pf);
```

Si el archivo se cierra con éxito retorna 0.

Veamos un ejemplo en el que abrimos un archivo de texto, en caso de generarse un error al abrir el archivo se imprime un mensaje. Si el archivo fue abierto con éxito, simplemente cerramos el archivo utilizando la función **fclose()**.

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

int main(){
	FILE *pf;
	pf=fopen("GameOfThrones.txt","r");
	if(pf==NULL)
		printf ("Error al abrir el archivo");
	else
		fclose(pf);
	return 0;
}
```

2.1.2 Escritura y lectura de un archivo

Para escribir caracteres en un archivo de texto, podemos utilizar la función **fputc()**, su sintaxis es:

```
fputc(caracter , apuntador_archivo);
```

Veamos un ejemplo, en el que abrimos un archivo de texto en modo escritura y escribimos una letra en el archivo:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
	FILE *pf;
	char letra='a';
	pf=fopen("datos.txt","w");
	if(pf==NULL){
		printf("Error al abrir el archivo");
		return 0;
	}
	else{
		fputc(letra, pf);
		fclose(pf);
	}
	return 0;
}
```

Para leer un carácter de un archivo (abierto en modo lectura). Deberemos guardarlo en una variable. Veamos un ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
	FILE *pf;
	char letra;
	pf=fopen("datos.txt","r");
	if (pf==NULL){
		printf("Error al abrir el fichero");
```

```
                return 0;
        }
        else {
                letra=fgetc(pf);
                printf("%c",letra);
                fclose(pf);
        }
        return 0;
}
```

Si queremos leer todos los caracteres que se encuentran almacenados en un archivo de texto, es necesario poder identificar cuando finaliza el archivo. Para esto, podemos utilizar la función **feof()**. Su sintaxis es:

```
feof(apuntador_archivo);
```

Esta función retorna 0 si no ha llegado al final del archivo, y un valor diferente de 0 si lo ha alcanzado. Veamos un ejemplo de cómo podemos leer todo el contenido de un archivo de texto e imprimirlo en pantalla:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        FILE *pf;
        char letra;
        pf=fopen("datos.txt","r");
        if (pf==NULL){
                printf("Error al abrir el fichero");
                return 0;
        }
        else {
                while(1){
                        letra=fgetc(pf);
                        if (feof(pf)) break;
                        printf("%c",letra);
                }
                fclose(pf);
        }
        return 0;
}
```

Si no queremos leer y escribir a nivel carácter en los archivos, podemos utilizar las funciones **fputs()** y **fgets()** que permiten escribir y leer cadenas de texto. Veamos un ejemplo del uso de la función **fgets()** para leer el contenido de un archivo de texto.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        setbuf(stdin, NULL);
        setbuf(stdout, NULL);
        FILE *pf;
        char buffer[300];
        pf=fopen("datos.txt","r");
        if (pf==NULL){
                printf("Error al abrir el fichero");
                return 0;
        }
        else {
                while(!feof(pf)){
```

```
                if (fgets(buffer, 256, pf) != NULL)
                        printf("%s",buffer);
            }
            fclose(pf);
        }
        return 0;
}
```

2.2 Requisitos funcionales

Recordemos que nuestro objetivo es crear un programa que permita al usuario consultar el contenido de archivos de texto y navegar entre el histórico de archivos que ha consultado. Para el diseño de la solución y su posterior implementación, consideremos los siguientes escenarios.

a) Escenario 1 (inicio del programa): No existen archivos en el histórico

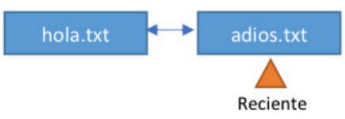
```
[1] Abrir archivo
[2] Salir
Seleccione opción: 1
Nombre del archivo: hola.txt
   Este es el archivo hola.txt
   Ya se acabó el archivo hola.txt
```

b) Escenario 2: Existe un elemento en el histórico

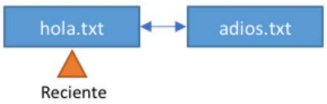
```
[1] Abrir archivo
[2] Salir
[3] Mostrar reciente        Reciente es hola.txt
[4] Mostrar anterior        No hay anterior
[5] Mostrar siguiente       No hay siguiente
Seleccione opción: 1
Nombre del archivo: adiós.txt
   Este es el archivo adiós.txt
   Ya se acabó el archivo adiós.txt
```

c) Escenario 3: Existen dos elementos en el histórico

```
[1] Abrir archivo
[2] Salir
[3] Mostrar reciente        Reciente es adiós.txt
[4] Mostrar anterior        Anterior es hola.txt
[5] Mostrar siguiente       No hay siguiente
Seleccione opción: 3
   Archivo: adiós.txt
   Este es el archivo adiós.txt
   Ya se acabó el archivo adiós.txt
```

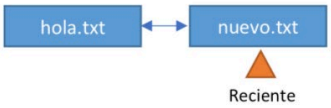


```
[1] Abrir archivo
[2] Salir
[3] Mostrar reciente          Reciente es adiós.txt
[4] Mostrar anterior          Anterior es hola.txt
[5] Mostrar siguiente         No hay siguiente
Seleccione opción: 4
   Archivo: hola.txt
   Este es el archivo hola.txt
   Ya se acabó el archivo hola.txt
```



d) Escenario 4: Existen tres o más elementos en el histórico

```
[1] Abrir archivo
[2] Salir
[3] Mostrar reciente          Reciente es hola.txt
[4] Mostrar anterior          No hay anterior
[5] Mostrar siguiente         El siguiente es adiós.txt
Seleccione opción: 1          Estábamos en hola y abrimos nuevo, se perdió adiós
   Archivo: nuevo.txt
   Este es el archivo nuevo.txt
   Ya se acabó el archivo nuevo.txt
```




```
[1] Abrir archivo
[2] Salir
[3] Mostrar reciente          Reciente es nuevo.txt
[4] Mostrar anterior          Anterior es hola.txt
[5] Mostrar siguiente         No hay siguiente
Seleccione opción: 1          El archivo abierto es igual a Anterior.
   Archivo: hola.txt
   Este es el archivo nuevo.txt
   Ya se acabó el archivo nuevo.txt
```


```
[1] Abrir archivo
[2] Salir
[3] Mostrar reciente          Reciente es hola.txt
[4] Mostrar anterior          Anterior es nuevo.txt
[5] Mostrar siguiente         No hay siguiente
Seleccione opción: 1          Como el reciente es igual al archivo abierto, no hay cambios.
   Archivo: hola.txt
   Este es el archivo nuevo.txt
   Ya se acabó el archivo nuevo.txt

Incluir estas opciones (con significado obvio)
[6] Mostrar primero   (hola.txt)
[7] Mostrar último    (de nuevo, hola.txt)
```

3. Evaluación

Los productos que debe entregar en CANVAS son:

- Código fuente de sus contenedores y solución del problema (*.c y *.h)
- Reporte de la actividad (consultar plantilla)

Los criterios para considerar para evaluar su actividad son los siguientes:

Requerimientos funcionales (80%)

Funcionalidades	10	5	0
Abrir archivo	Funciona de manera correcta	Tiene errores de sintaxis	Tiene errores de lógica o no fue implementada
Mostrar reciente	Funciona de manera correcta	Tiene errores de sintaxis	Tiene errores de lógica o no fue implementada
Mostrar anterior	Funciona de manera correcta	Tiene errores de sintaxis	Tiene errores de lógica o no fue implementada
Mostrar siguiente	Funciona de manera correcta	Tiene errores de sintaxis	Tiene errores de lógica o no fue implementada
Mostrar primero	Funciona de manera correcta	Tiene errores de sintaxis	Tiene errores de lógica o no fue implementada
Mostrar último	Funciona de manera correcta	Tiene errores de sintaxis	Tiene errores de lógica o no fue implementada

Nota: para tener derecho a calificación, es requisito indispensable utilizar contenedores genéricos, memoria dinámica y manejo de archivos.

Reporte (30%)

Criterios	10	5	0
Portada	Incluye los elementos necesarios para identificar la asignatura, el trabajo realizado, la institución educativa, los integrantes del equipo y fecha. Adicionalmente, tiene una buena presentación.	Incluye los elementos necesarios para identificar la asignatura, el trabajo realizado, la institución educativa, los integrantes del equipo y fecha. Sin embargo, tiene una mala presentación.	No se incluye o no cumple con los requisitos para una calificación mayor

Introducción	Brinda un resumen de su trabajo que permite conocer al lector de manera **breve** cuál es el contenido de su trabajo: problema, propuesta de solución y resultados.	Brinda un resumen de su trabajo, pero no permite conocer al lector de manera breve cuál es el contenido de su trabajo: problema, propuesta de solución y resultados.	No se incluye o no cumple con los requisitos para una calificación mayor
Análisis del problema	Brinda evidencia del análisis realizado sobre el problema: incluye redacción y diagramas.	Brinda evidencia del análisis realizado sobre el problema: incluye redacción, pero no incluye diagramas.	No se incluye o no cumple con los requisitos para una calificación mayor
Diseño de la solución	Brinda evidencia del diseño de la solución que se realizó en base aun análisis previo del problema: incluye redacción y diagramas.	Brinda evidencia del diseño de la solución que se realizó en base aun análisis previo del problema: incluye redacción y, pero no incluye diagramas.	No se incluye o no cumple con los requisitos para una calificación mayor
Pruebas	Muestra evidencia de las pruebas que fueron realizadas para validar cada requerimiento funcional.	Muestra evidencia de las pruebas que fueron realizadas, pero no son suficientes para validar cada requerimiento funcional.	No se incluye
Conclusiones	Brinda evidencia de una reflexión y discusión realizada entre los autores del trabajo respecto a: problemas, aprendizajes, resultados y oportunidades de mejora.	Brinda evidencia de una reflexión y discusión realizada entre los autores del trabajo, sin embargo no aborda alguno de los siguientes aspectos: problemas, aprendizajes, resultados o oportunidades de mejora.	No se incluye