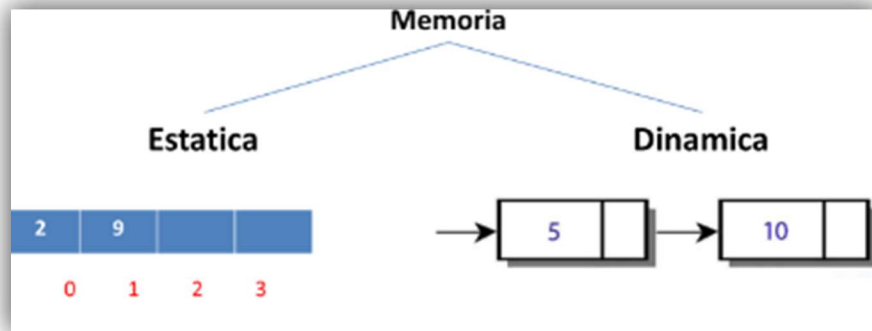


Universidad Autónoma de la Ciudad de México

Introducción a la ingeniería en software

Practica 3 Creación de un editor de texto básico, Uso de Arreglos/Buffer, Funciones, apuntadores y memoria dinámica



Nombre: Marcial Espinoza Joseline Ariana

Matricula: 22-003-0552

Maestro: Raúl Rolando Jara Montes

En esta práctica representa el mejoramiento del texto desarrollado en la práctica 2, incorporando ahora el uso de memoria dinámica para eliminar las limitaciones de tamaño fijo y optimizar el uso de recursos del sistema. El objetivo principal es el manejo de punteros, funciones y la gestión de memoria dinámica mediante las funciones `malloc()`, `realloc()` y `free()`.

A diferencia de la versión anterior, que utilizaba un buffer estático de tamaño fijo, este nuevo editor emplea un buffer dinámico capaz de crecer automáticamente conforme el usuario ingresa más texto. De esta manera, se logra una mayor flexibilidad y eficiencia al manejar cadenas de longitud variable.

El programa se estructura en varias funciones:

`inicializar()`: asigna memoria inicial al texto.

`insertarTexto()`: permite ingresar texto y redimensiona la memoria según sea necesario.

`mostrarTexto()`: muestra el contenido actual del texto almacenado.

`buscarYReemplazar()`: realiza la búsqueda de una palabra o frase y su sustitución por otra, administrando la memoria de forma automática.

El menú principal brinda al usuario opciones básicas para insertar texto, visualizarlo, buscar y reemplazar fragmentos, o salir del programa. Cada operación demuestra la aplicación práctica de la gestión dinámica de memoria, asegurando que la cantidad de espacio utilizado se ajuste a las necesidades del usuario sin desperdiciar recursos.

Durante el desarrollo de esta práctica se implementó un editor de texto ,basado en el uso de punteros y memoria dinámica para gestionar de forma eficiente el contenido ingresado por el usuario. El programa utiliza un puntero principal (`char *texto`) que apunta a un bloque de memoria asignado dinámicamente con `malloc()` y redimensionado con `realloc()` conforme se agrega o modifica información, permitiendo eliminar las limitaciones del tamaño fijo. Se diseñaron funciones específicas como `inicializar()` para reservar memoria, `insertarTexto()` para añadir

nuevo contenido, mostrarTexto() para visualizar el texto almacenado y buscarYReemplazar() para localizar y sustituir cadenas dentro del buffer. Cada función hace uso directo de punteros para acceder, recorrer y modificar los datos en memoria, reforzando el manejo de direcciones y desplazamientos en el lenguaje C. Finalmente, la memoria utilizada se libera correctamente con free(), asegurando un uso responsable de los recursos del sistema. Este enfoque permitió construir un editor funcional, dinámico y escalable, que demuestra el dominio de los conceptos fundamentales de punteros, funciones y administración de memoria dinámica.

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

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

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

```
char *texto = NULL;
```

```
size_t tam = 100;
```

```
size_t longitud = 0;
```

```
void inicializar();
```

```
void insertarTexto();
```

```
void mostrarTexto();
```

```
void buscarYReemplazar();
```

```
int main() {
```

```
    int opcion;
```

```
    inicializar();
```

```
    do {
```

```
        printf("\n--Menu Principal--\n");
```

```
        printf("1. Insertar texto\n");
```

```
        printf("2. Mostrar texto\n");
```

```
printf("3. Buscar y reemplazar\n");

printf("4. Salir\n");

printf("Seleccione una opcion: ");

scanf("%d", &opcion);

getchar(); // limpia el buffer

switch (opcion) {

    case 1:

        insertarTexto();

        break;

    case 2:

        mostrarTexto();

        break;

    case 3:

        buscarYReemplazar();

        break;

    case 4:

        printf("Saliendo del programa.\n");

        break;

    default:

        printf("Opcion no valida.\n");

}

} while (opcion != 4);

free(texto);

return 0;
```

```

}

void inicializar() {

    texto = (char *)malloc(tam);

    if (!texto) {

        printf("Error al asignar memoria.\n");

        exit(1);

    }

    texto[0] = '\0';

}

void insertarTexto() {

    char entrada[200];

    printf("Ingrese el texto a insertar:\n");

    fgets(entrada, sizeof(entrada), stdin);

    entrada[strcspn(entrada, "\n")] = '\0'; // eliminar salto de línea

    size_t len = strlen(entrada);

    if (longitud + len + 1 >= tam) {

        tam *= 2;

        texto = (char *)realloc(texto, tam);

        if (!texto) {

            printf("Error al redimensionar memoria.\n");

            exit(1);

        }

        printf("Memoria aumentada a %zu bytes.\n", tam);

    }

}

```

```
    strcat(texto, entrada);

    longitud += len;

    printf("Texto insertado (%zu caracteres).\n", longitud);
}
```

```
void mostrarTexto() {

    printf("\n--- Texto actual ---\n");

    if (longitud == 0)

        printf("(Vacío)\n");

    else

        printf("%s\n", texto);

}
```

```
void buscarYReemplazar() {

    char buscar[50], reemplazar[50];

    printf("Ingrese el texto a buscar: ");

    fgets(buscar, sizeof(buscar), stdin);

    buscar[strcspn(buscar, "\n")] = '\0';

    printf("Ingrese el texto de reemplazo: ");

    fgets(reemplazar, sizeof(reemplazar), stdin);

    reemplazar[strcspn(reemplazar, "\n")] = '\0';
```

```
char *pos, *resultado;

int ocurrencias = 0;

size_t lenBuscar = strlen(buscar);

size_t lenReemplazar = strlen(reemplazar);


// Contar ocurrencias

pos = texto;

while ((pos = strstr(pos, buscar)) != NULL) {

    ocurrencias++;

    pos += lenBuscar;

}

if (ocurrencias == 0) {

    printf("No se encontró el texto '%s'.\n", buscar);

    return;

}


// Calcular tamaño nuevo y reservar memoria

resultado = (char *)malloc(strlen(texto) + ocurrencias * (lenReemplazar - lenBuscar) + 1);

if (!resultado) {

    printf("Error al asignar memoria para el texto nuevo.\n");

    exit(1);

}

// Reemplazar texto
```

```
char *ptrResultado = resultado;

pos = texto;

while (*pos) {

    if (strstr(pos, buscar) == pos) {

        strcpy(ptrResultado, reemplazar);

        ptrResultado += lenReemplazar;

        pos += lenBuscar;

    } else {

        *ptrResultado++ = *pos++;

    }

}

*ptrResultado = '\0';

// Actualizar texto original

free(texto);

texto = resultado;

longitud = strlen(texto);

printf("Reemplazo completado (%d ocurrencias).\n", ocurrencias);

printf("Texto final: %s\n", texto);

}
```

Pruebas:


```
C:\Users\arian\OneDrive\Doc... X + v
-- MENU PRINCIPAL --
1. Insertar texto
2. Mostrar texto
3. Buscar y reemplazar
4. Mostrar uso de memoria
5. Salir
Seleccione una opcion: 1
Ingrese el texto a insertar:
arbol rosa
Texto insertado (10 caracteres).
```

```
-- MENU PRINCIPAL --
1. Insertar texto
2. Mostrar texto
3. Buscar y reemplazar
4. Mostrar uso de memoria
5. Salir
Seleccione una opcion: 3
Ingrese el texto a buscar: arbol
Ingrese el texto de reemplazo: rosa
Reemplazo completado (1 ocurrencias).
Texto final: rosa rosa
```

```
C:\Users\arian\OneDrive\Doc... X + v
Texto final: rosa rosa

-- MENU PRINCIPAL --
1. Insertar texto
2. Mostrar texto
3. Buscar y reemplazar
4. Mostrar uso de memoria
5. Salir
Seleccione una opcion: 2

--- TEXTO ACTUAL ---
rosa rosa
```

```
-- MENU PRINCIPAL --
1. Insertar texto
2. Mostrar texto
3. Buscar y reemplazar
4. Mostrar uso de memoria
5. Salir
Seleccione una opcion: 4

--- USO DE MEMORIA ---
Tam total asignado: 100 bytes
Texto utilizado: 9 caracteres
Espacio libre: 91 bytes

-- MENU PRINCIPAL --
1. Insertar texto
2. Mostrar texto
3. Buscar y reemplazar
4. Mostrar uso de memoria
5. Salir
Seleccione una opcion: |
```

```
--- USO DE MEMORIA ---
Tam total asignado: 100 bytes
Texto utilizado: 9 caracteres
Espacio libre: 91 bytes

-- MENU PRINCIPAL --
1. Insertar texto
2. Mostrar texto
3. Buscar y reemplazar
4. Mostrar uso de memoria
5. Salir
Seleccione una opcion: 5
Saliendo del programa.

-----
Process exited after 151.5 seconds with re
Presione una tecla para continuar . . . |
```

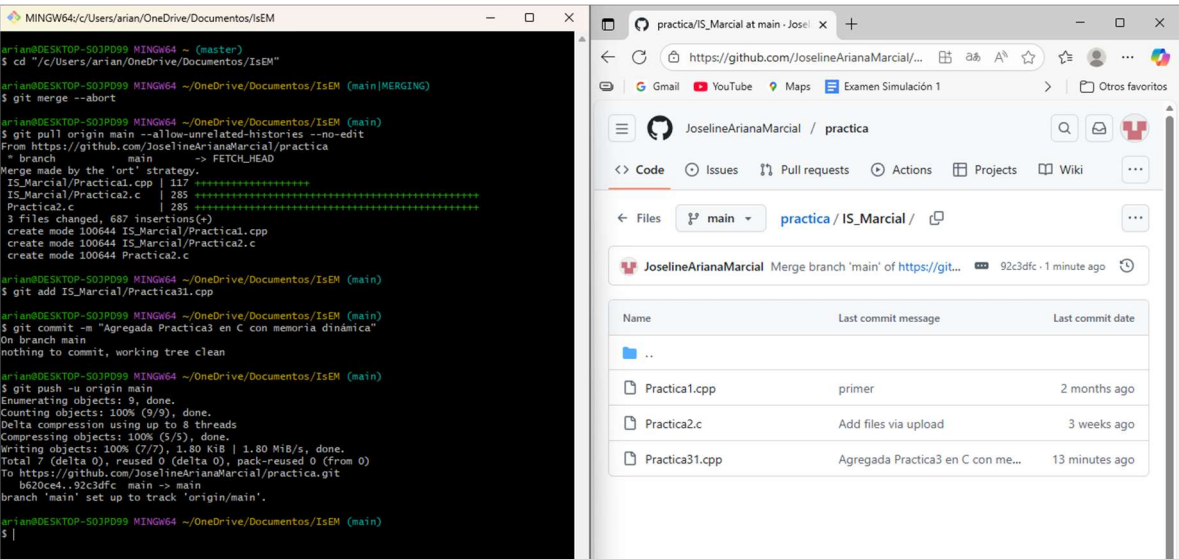
En esta práctica se logró desarrollar un editor de texto mejorado que supera las limitaciones de la versión anterior mediante la implementación de memoria dinámica y el uso de punteros en el lenguaje C. Se consiguió una gestión eficiente de memoria

utilizando las funciones malloc(), realloc() y free() para asignar, ampliar y liberar memoria según las necesidades del texto, eliminando las restricciones de tamaño fijo y evitando el desperdicio de espacio. El programa amplía su capacidad de manera automática cuando el usuario ingresa más texto, asegurando un funcionamiento continuo sin interrupciones ni pérdida de datos. Además, se mantuvieron las funciones de búsqueda y reemplazo.

Registro de tiempos:

Fecha	Actividad	inicio	Fin	Duración	Interrupciones	T.E	T.R	Comentarios
7/nov/25	Revisión de practica 2 ,para iniciar la 3	9:00	1:00	240min		160min	210min	Repaso de memoria y codificación para adaptarla y mejorarla.
8/Nov25	Pruebas	12:00	13:30	90min		50min	90min	Pruebas
11/Nov/25	Documentación	4:00	6:00	30min		60 min	90min	Documentación ,repositorio.

Evidencia de GitHub



JoselineArianaMarcial / practica

Q Type [Z] to search

+

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

main

practica / IS_Marcial /

Q Go to file

Add file

JoselineArianaMarcial

Merge branch 'main' of https://github.com/JoselineArianaMarcial/practica

92c3dfc · 3 minutes ago

History

Name	Last commit message	Last commit date
..		
Practica1.cpp	primer	2 months ago
Practica2.c	Add files via upload	3 weeks ago
Practica31.cpp	Agregada Practica3 en C con memoria dinamica	16 minutes ago

<https://github.com/JoselineArianaMarcial/practica>