UACM

Universidad Autónoma
de la Ciudad de México

NADA HUMANO ME ES AJENO

Practica 3: creación de un editor de texto básico, uso de arreglos/buffer, funciones, apuntadores y memoria dinámica.

Introducción a la ingeniería de Software.

Profesor: Raúl Jara.

Zavala Pérez Velia.

Semestre 2025-II.

Introducción.

En esta práctica se desarrolla un editor de texto básico en lenguaje C que permite al usuario:

- Ingresar texto libremente (sin límite fijo de caracteres),
- Buscar una palabra o frase dentro del texto,
- Reemplazarla por otra palabra o frase,
- Mostrar los resultados intermedios y finales,
- Observar cómo cambia el uso de memoria al modificar el texto.

El objetivo es aplicar el uso de memoria dinámica mediante las funciones malloc, calloc, realloc y free, así como el manejo de apuntadores y funciones para modularizar el programa.

Código:

```
#include <stdio.h>

#include <stdlib.h>


#define TAM_INICIAL 64


// --- Prototipos ---

char* leer_texto_dinamico(void);

void mostrar(const char *titulo, const char *texto);

int longitud(const char *cadena);

char* buscar_y_reemplazar(const char *original, const char *buscar, const char *reemplazar);

void liberar_memoria(char **ptr);


int main(void) {

    char *texto_original = NULL;

    char *texto_buscar = NULL;
```

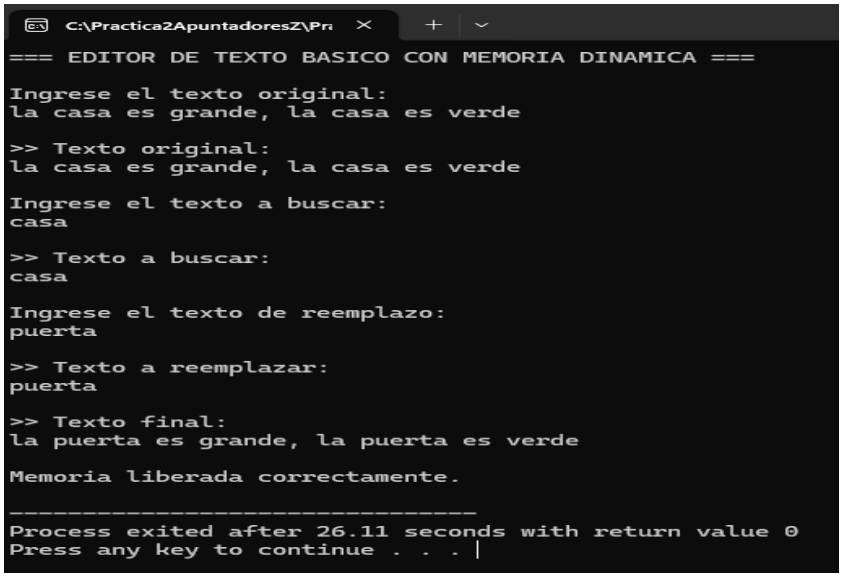
```
    char *texto_reemplazar = NULL;
    char *resultado = NULL;

    printf("=== EDITOR DE TEXTO BASICO CON MEMORIA DINAMICA ===\n");

    // Leer textos
    printf("\nIngrese el texto original:\n");
    texto_original = leer_texto_dinamico();
    mostrar("Texto original", texto_original);

    printf("\nIngrese el texto a buscar:\n");
    texto_buscar = leer_texto_dinamico();
    mostrar("Texto a buscar", texto_buscar);

    printf("\nIngrese el texto de reemplazo:\n");
    texto_reemplazar = leer_texto_dinamico();
    mostrar("Texto a reemplazar", texto_reemplazar);

    // Buscar y reemplazar
    resultado = buscar_y_reemplazar(texto_original, texto_buscar,
texto_reemplazar);
    mostrar("Texto final", resultado);

    // Liberar memoria
    liberar_memoria(&texto_original);
    liberar_memoria(&texto_buscar);
    liberar_memoria(&texto_reemplazar);
    liberar_memoria(&resultado);
```

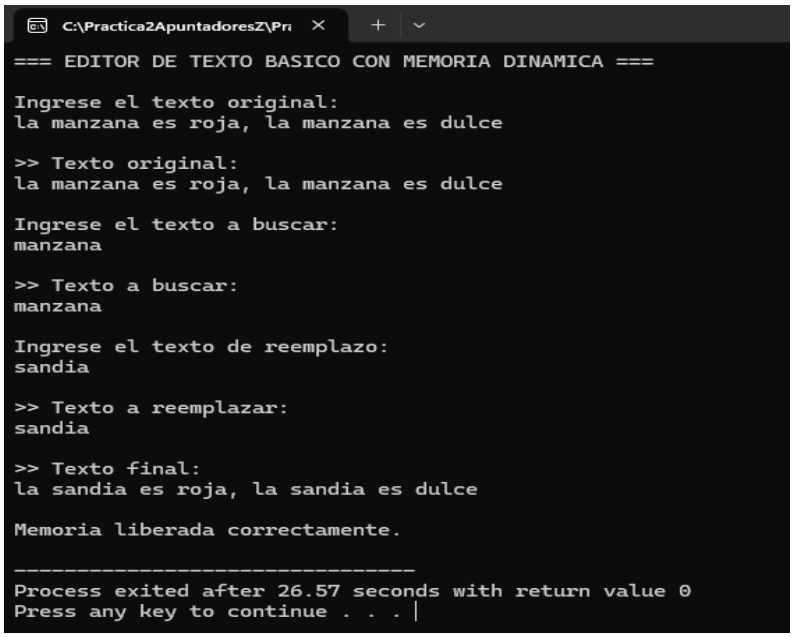
```
    printf("\nMemoria liberada correctamente.\n");

    return 0;
}

// --- Función para leer texto dinámicamente ---
char* leer_texto_dinamico(void) {
    char *buffer = (char *)malloc(TAM_INICIAL * sizeof(char));
    if (!buffer) {
        printf("Error de memoria\n");
        exit(1);
    }

    int capacidad = TAM_INICIAL;
    int longitud = 0;
    int c;

    while ((c = getchar()) != '\n' && c != EOF) {
        if (longitud >= capacidad - 1) {
            capacidad *= 2;
            char *temp = (char *)realloc(buffer, capacidad * sizeof(char));
            if (!temp) { free(buffer); printf("Error de memoria\n"); exit(1); }
            buffer = temp;
            printf("[DEBUG] Memoria expandida a %d bytes\n", capacidad);
        }
        buffer[longitud++] = (char)c;
    }
    buffer[longitud] = '\0';
```

```
    return buffer;
}

// --- Mostrar texto ---
void mostrar(const char *titulo, const char *texto) {
    printf("\n>> %s:\n%s\n", titulo, texto);
}

// --- Longitud manual ---
int longitud(const char *cadena) {
    int len = 0;
    while (cadena[len] != '\0') len++;
    return len;
}

// --- Buscar y reemplazar ---
char* buscar_y_reemplazar(const char *original, const char *buscar, const char *reemplazar) {
    int len_buscar = longitud(buscar);
    int len_reemplazar = longitud(reemplazar);
    int capacidad = 128;
    char *resultado = (char *)malloc(capacidad * sizeof(char));
    if (!resultado) { printf("Error de memoria\n"); exit(1); }

    int idx = 0;
    const char *p = original;

    while (*p) {
        int coincide = 1;
```

```
for (int i = 0; i < len_buscar && p[i]; i++) {
    if (p[i] != buscar[i]) { coincide = 0; break; }
}

if (coincide && len_buscar > 0) {
    // Asegurar espacio
    while (idx + len_reemplazar >= capacidad) {
        capacidad *= 2;
        char *temp = (char *)realloc(resultado, capacidad * sizeof(char));
        if (!temp) { free(resultado); printf("Error de memoria\n"); exit(1); }
        resultado = temp;
        printf("[DEBUG] Memoria expandida a %d bytes\n", capacidad);
    }
    // Copiar reemplazo
    for (int j = 0; j < len_reemplazar; j++)
        resultado[idx++] = reemplazar[j];
    p += len_buscar;
} else {
    // Asegurar espacio
    if (idx + 1 >= capacidad) {
        capacidad *= 2;
        char *temp = (char *)realloc(resultado, capacidad * sizeof(char));
        if (!temp) { free(resultado); printf("Error de memoria\n"); exit(1); }
        resultado = temp;
        printf("[DEBUG] Memoria expandida a %d bytes\n", capacidad);
    }
    resultado[idx++] = *p++;
}
```

```
    }
    resultado[idx] = '\0';
    return resultado;
}

// --- Liberar memoria ---
void liberar_memoria(char **ptr) {
    if (*ptr) { free(*ptr); *ptr = NULL; }
}
```





Acción.	Tiempo estimado.	Tiempo real.
Leer texto original	**5 minutos**	**10 minutos**
Leer texto buscar	**5 minutos**	**5 minutos**
Leer texto reemplazo	**5 minutos**	**6 minutos**
Implementar buscar/reemplazar	**30 minutos**	**20 minutos**
Pruebas y depuración	**15 minutos**	**10 minutos**

Defectos encontrados y correcciones:

- Olvido de liberar memoria → agregado free y función liberar_memoria.
- Overflow de buffer al ingresar texto largo → corregido con realloc.
- Uso de malloc sin cast → se agregó (char *) para compiladores estrictos.

Conclusiones.

Se logró implementar un editor de texto básico que utiliza memoria dinámica, mostrando el crecimiento del buffer.

El uso de funciones modulares y apuntadores mejora la claridad del código.

La práctica permite extender funcionalidades para proyectos futuros (como varias líneas, guardar en archivo, etc.).