	Práctica 6
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): M.I. Heriberto García Ledezma

Asignatura: Estructura de datos y algoritmos I

Grupo: 15

No de Práctica(s): Práctica 6

Integrante(s): Fuentes Llantada Marco Antronio

Rojas Contreras Aaron

No. de lista o brigada: 11 y 31

Semestre: 2025-2

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

Objetivos de la práctica

Revisar las definiciones, características, procedimientos y ejemplos de las estructuras lineales Pila y Cola, con la finalidad de comprender sus estructuras y poder implementarlas.

Ejercicios de la práctica

Ejercicio 1.

Hacer el código para estructura cola circular

Continuar el código realizado en clase para una estructura cola circular. Ahora implementar

las funciones:

- desencolar
- vaciar
- mostrarValores

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

//Se diseña la estructura del nodo.
struct Nodo{
    int valor;
    struct Nodo* aptSigNodo;
};

//Se diseña la estructura de la cola.
struct ColaCircular{
    struct Nodo* aptFrente;
    struct Nodo* aptAtras;
    int tamanioMaximo;
    int tamanioActual;
};


void inicializar(struct ColaCircular* laColaCircular, int capacidadMaxima); //Recibe un apuntador a la cola
void encolar(struct ColaCircular* laColaCircular, int valorAAgregar); //Recibe un apuntador a la cola y el
int desencolar(struct ColaCircular* laColaCircular); //Recibe un apuntador a la cola circular de la que se
int estaLlena(struct ColaCircular* laColaCircular); //Recibe un apuntador a la cola circular que se revisar
int estaVacia(struct ColaCircular* laColaCircular); //Recibe un apuntador a la cola circular que se revisar
void vaciar(struct ColaCircular* laColaCircular); //Recibe un apuntador a la cola circular que se vaciará
void mostrarValores(struct ColaCircular* laColaCircular);

int main(){
    struct ColaCircular laColaCircular;
    inicializar(&laColaCircular, 5);
    encolar(&laColaCircular, 1);
    encolar(&laColaCircular, 2);
    encolar(&laColaCircular, 3);
    encolar(&laColaCircular, 4);
    desencolar(&laColaCircular);
    encolar(&laColaCircular, 5);
    encolar(&laColaCircular, 6);
    encolar(&laColaCircular, 7);
    encolar(&laColaCircular, 8);
    encolar(&laColaCircular, 9);
    mostrarValores(&laColaCircular);
    vaciar(&laColaCircular);
    mostrarValores(&laColaCircular);
    return 1;
}

//Recibe un apuntador a la cola que se va a inicializar y su capacidad máxima.
void inicializar(struct ColaCircular* laColaCircular, int capacidadMaxima)
{
    laColaCircular->aptFrente = NULL;
    laColaCircular->aptAtras = NULL;
    laColaCircular->tamanioMaximo = capacidadMaxima;
    laColaCircular->tamanioActual = 0;
}

//Recibe un apuntador a la cola que se revisará si está llena.
//Regresa un 1 si está llena y un 0 si no está llena.
int estaLlena(struct ColaCircular* laColaCircular)
{
    int bandera=0;
    if(laColaCircular->tamanioActual == laColaCircular->tamanioMaximo )
    {
        bandera=1;
    }
    else
    {
        bandera=0;
    }
```

```
{
    else
    {
        bandera=0;
    }
    return bandera;
}


//Recibe un apuntador a la cola que se revisará si está vacía.
//Regresa un 1 si está vacía y un 0 si no está vacía.
int estaVacia(struct ColaCircular* laColaCircular)
{
    int bandera=0;
    if(laColaCircular->aptAtras == NULL && laColaCircular->aptFrente==NULL)
    {
        bandera=1;
    }
    else
    {
        bandera=0;
    }
    return bandera;
}

//Recibe un apuntador a la cola y el valor para el nodo a agregar.
void encolar(struct ColaCircular* laColaCircular, int valorAAgregar)
{
    if( estaLlena(laColaCircular) == 1   )
    {
        printf("\nLa cola esta llena. No se pueden agregar elementos\n");
    }
    else
    {
        struct Nodo* aptNuevoNodo= (struct Nodo*) calloc(1, sizeof(struct Nodo) ); //Se reserva memoria par
        if(aptNuevoNodo != NULL)    //Si se pudo reservar memoria dinámicamente para ese nodo, entonces:
        {
            aptNuevoNodo->valor = valorAAgregar;    //Se asigna el valor indicado al nodo que se agregará a
            aptNuevoNodo->aptSigNodo = NULL;    //De inicio el nuevo nodo apunta a NULL

            if( estaVacia(laColaCircular) == 1  )   //Si la cola está vacía, significa que el nodo a agrega
            {
                laColaCircular->aptFrente = aptNuevoNodo;   //Se hace que el aptFrente apunte al nuevo nodo
                laColaCircular->aptAtras = aptNuevoNodo;    //Se hace que el aptAtras apunte al nuevo nodo
            }
            else    //Si la cola NO está vacía,
            {
                laColaCircular->aptAtras->aptSigNodo = aptNuevoNodo;    //Se hace que el nodo que actualmen
                                                                        //atrás apunte al nuevo nodo, que s
                laColaCircular->aptAtras = aptNuevoNodo;                //Se hace que el aptAtras apunte al


            }


            /*La siguiente linea permite la circularidad*/
            //Se hace que el nodo de hasta atrás apunte al nodo que está hasta enfrente.
            //IMPORTANTE: Esto debe ser común a si se inserta un nuevo nodo o si ya había nodos
            laColaCircular->aptAtras->aptSigNodo = laColaCircular->aptFrente;

            laColaCircular->tamanioActual++;    //Se incrementa el tamanio de la cola en uno
            printf("\nSe agregó el valor: %i\n", laColaCircular->aptAtras->valor);  //Se imprime el valor d
        }
        else
        {
            printf("\nNo se pudo reservar memoria para el nuevo nodo\n");
        }
```

```c
{
    {
        {
        }

    }
}

//Recibe un apuntador a la cola circular de la que se extrae un nodo
//Regresa el valor del nodo desencolado
int desencolar(struct ColaCircular* laColaCircular)
{
    int valorExtraido = -1;  // Valor por defecto si la cola está vacía

    if(estaVacia(laColaCircular) == 1)
    {
        printf("\nLa cola está vacía. No se pueden extraer elementos\n");
    }
    else
    {
        struct Nodo* aptNodoAEliminar = laColaCircular->aptFrente;  // Se guarda una referencia al nodo a eliminar
        valorExtraido = aptNodoAEliminar->valor;  // Se guarda el valor del nodo a eliminar

        if(laColaCircular->aptFrente == laColaCircular->aptAtras)  // Si solo hay un nodo en la cola
        {
            laColaCircular->aptFrente = NULL;
            laColaCircular->aptAtras = NULL;
        }
        else
        {
            laColaCircular->aptFrente = aptNodoAEliminar->aptSigNodo;  // El frente apunta al siguiente nodo
            laColaCircular->aptAtras->aptSigNodo = laColaCircular->aptFrente;  // Se actualiza la referencia circu
        }

        free(aptNodoAEliminar);  // Se libera la memoria del nodo eliminado
        laColaCircular->tamanioActual--;  // Se decrementa el tamaño actual de la cola
        printf("\nSe extrajo el valor: %i\n", valorExtraido);
    }

    return valorExtraido;
}

//Muestra los valores de todos los nodos de la cola circular
void mostrarValores(struct ColaCircular* laColaCircular)
{
    if(estaVacia(laColaCircular) == 1)
    {
        printf("\nLa cola está vacía. No hay elementos que mostrar\n");
    }
    else
    {
        struct Nodo* aptNodoActual = laColaCircular->aptFrente;
        int contador = 0;

        printf("\nValores en la cola circular:\n");
        do
        {
            printf("Posición %d: %d\n", contador, aptNodoActual->valor);
            aptNodoActual = aptNodoActual->aptSigNodo;
            contador++;
        } while(aptNodoActual != laColaCircular->aptFrente);

        printf("Tamaño actual de la cola: %d\n", laColaCircular->tamanioActual);
        printf("Tamaño máximo de la cola: %d\n", laColaCircular->tamanioMaximo);
    }
}
```

```
Se agregó el valor: 1

Se agregó el valor: 2

Se agregó el valor: 3

Se agregó el valor: 4

Se extrajo el valor: 1

Se agregó el valor: 5

Se agregó el valor: 6

La cola esta llena. No se pueden agregar elementos

La cola esta llena. No se pueden agregar elementos

La cola esta llena. No se pueden agregar elementos

Valores en la cola circular:
Posición 0: 2
Posición 1: 3
Posición 2: 4
Posición 3: 5
Posición 4: 6
Tamaño actual de la cola: 5
Tamaño máximo de la cola: 5

Vaciando la cola circular...

Se extrajo el valor: 2

Se extrajo el valor: 3

Se extrajo el valor: 4

Se extrajo el valor: 5

Se extrajo el valor: 6
La cola ha sido vaciada completamente
```

Ejercicio 2.

Programa que almacene los nombres de las canciones que contendrá una playlist. Primero deberá pedir el nombre de la playlist y cuántas va a tener. Después presentará al usuario el siguiente menú de opciones:

1. Agregar una canción a la playlist

2. Quitar una canción de la playlist

3. Reproducir la playlist una vez

4. Reproducir la playlist un determinado número de veces

5. Vaciar la playlist

Si el usuario selecciona la opción 1, el programa preguntará el nombre de la canción, su género y el nombre del o de la cantante. Si no hay espacio en la lista de canciones el programa le dirá al usuario que la playlist ya está llena, en caso contrario indicará que la canción se agregó y presentará sus datos en pantalla. Si el usuario escoge la opción 2, el programa quitará de la playlist la canción que se registró más antiguamente en la playlist. El programa mostrará los datos de la canción que se eliminó. Si el usuario indica la opción 3, el programa reproducirá la playlist una vez. Para simular que se reproduce cada canción de la lista, se mostrará en pantalla para cada canción la palabra “Reproduciendo” seguida de los datos de la canción y se hará una pausa de tres segundos antes de pasar con la siguiente canción.Si el usuario indica la opción 4, el programa preguntará cuántas veces debe reproducirse la playlist. Enseguida la reproducirá ese número de veces. Para simular que se reproduce cada canción de la lista, se mostrará en pantalla para cada canción la palabra “Reproduciendo” seguida de los datos de la canción y se hará una pausa de tres segundos antes de pasar con la siguiente canción.Si el usuario elige la opción 5, el programa vaciará la lista

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

#define MAX_LENGTH 50

// Estructura para almacenar la información de la canción
typedef struct {
    char nombre[MAX_LENGTH];
    char genero[MAX_LENGTH];
    char cantante[MAX_LENGTH];
} Cancion;

// Estructura de la cola circular
typedef struct {
    Cancion *canciones;
    int frente, final, capacidad, tamaño;
} ColaCircular;

// Inicializar la cola
void inicializarCola(ColaCircular *cola, int capacidad) {
    cola->capacidad = capacidad;
    cola->frente = 0;
    cola->final = -1;
    cola->tamaño = 0;
    cola->canciones = (Cancion *)malloc(capacidad * sizeof(Cancion));
}


int estaLlena(ColaCircular *cola) {
    return cola->tamaño == cola->capacidad;
}


int estaVacia(ColaCircular *cola) {
    return cola->tamaño == 0;
}


void agregarCancion(ColaCircular *cola) {
    if (estaLlena(cola)) {
        printf("La playlist está llena. No se puede agregar más canciones.\n");
        return;
    }
    Cancion nuevaCancion;
```

```c
    Cancion nuevaCancion;
    printf("Nombre de la canción: ");
    scanf(" %[^\n]", nuevaCancion.nombre);
    printf("Género de la canción: ");
    scanf(" %[^\n]", nuevaCancion.genero);
    printf("Nombre del o la cantante: ");
    scanf(" %[^\n]", nuevaCancion.cantante);

    cola->final = (cola->final + 1) % cola->capacidad;
    cola->canciones[cola->final] = nuevaCancion;
    cola->tamaño++;
    printf("Canción agregada: %s - %s (%s)\n", nuevaCancion.nombre, nuevaCancion.cantante, nuevaCancion.genero);
}


void quitarCancion(ColaCircular *cola) {
    if (estaVacia(cola)) {
        printf("La playlist está vacía. No hay canciones para quitar.\n");
        return;
    }
    Cancion cancionEliminada = cola->canciones[cola->frente];
    cola->frente = (cola->frente + 1) % cola->capacidad;
    cola->tamaño--;
    printf("Canción eliminada: %s - %s (%s)\n", cancionEliminada.nombre, cancionEliminada.cantante, cancionEliminada.genero);
}


void reproducirUnaVez(ColaCircular *cola) {
    if (estaVacia(cola)) {
        printf("La playlist está vacía. No hay canciones para reproducir.\n");
        return;
    }
    for (int i = 0; i < cola->tamaño; i++) {
        int indice = (cola->frente + i) % cola->capacidad;
        printf("Reproduciendo: %s - %s (%s)\n", cola->canciones[indice].nombre, cola->canciones[indice].cantante, cola->canciones[indice].genero);
    }
}


void reproducirVariasVeces(ColaCircular *cola, int veces) {
    if (estaVacia(cola)) {
        printf("La playlist está vacía. No hay canciones para reproducir.\n");
        return;
    }
    for (int v = 0; v < veces; v++) {
```

```c
void reproducirVariasVeces(ColaCircular *cola, int veces) {
    if (estaVacia(cola)) {
        printf("La playlist está vacía. No hay canciones para reproducir.\n");
        return;
    }
    for (int v = 0; v < veces; v++) {
        printf("Reproducción #%d:\n", v + 1);
        reproducirUnaVez(cola);
    }
}


void vaciarPlaylist(ColaCircular *cola) {
    cola->frente = 0;
    cola->final = -1;
    cola->tamaño = 0;
    printf("La playlist ha sido vaciada.\n");
}


void mostrarMenu() {
    printf("\nMenú de opciones:\n");
    printf("1. Agregar una canción a la playlist\n");
    printf("2. Quitar una canción de la playlist\n");
    printf("3. Reproducir la playlist una vez\n");
    printf("4. Reproducir la playlist un determinado número de veces\n");
    printf("5. Vaciar la playlist\n");
    printf("6. Salir\n");
}

int main() {
    ColaCircular playlist;
    char nombrePlaylist[MAX_LENGTH];
    int capacidad, opcion, veces;


    printf("Nombre de la playlist: ");
    scanf(" %[^\n]", nombrePlaylist);
    printf("¿Cuántas canciones puede tener la playlist? ");
    scanf("%d", &capacidad);

    inicializarCola(&playlist, capacidad);

    do {
        mostrarMenu();
```

```c
    do {
        mostrarMenu();
        printf("Elige una opción: ");
        scanf("%d", &opcion);

        switch (opcion) {
            case 1:
                agregarCancion(&playlist);
                break;
            case 2:
                quitarCancion(&playlist);
                break;
            case 3:
                reproducirUnaVez(&playlist);
                break;
            case 4:
                printf("¿Cuántas veces deseas reproducir la playlist? ");
                scanf("%d", &veces);
                reproducirVariasVeces(&playlist, veces);
                break;
            case 5:
                vaciarPlaylist(&playlist);
                break;
            case 6:
                printf("Saliendo del programa...\n");
                break;
            default:
                printf("Opción no válida. Intenta de nuevo.\n");
        }
    } while (opcion != 6);


    free(playlist.canciones);
    return 0;
}
```

```
Menú de opciones:
1. Agregar una canción a la playlist
2. Quitar una canción de la playlist
3. Reproducir la playlist una vez
4. Reproducir la playlist un determinado número de veces
5. Vaciar la playlist
6. Salir
Elige una opción:
1
Nombre de la canción: peligro
Género de la canción: corridos
Nombre del o la cantante: pesu pluma
Canción agregada: peligro - pesu pluma (corridos)

Menú de opciones:
1. Agregar una canción a la playlist
2. Quitar una canción de la playlist
3. Reproducir la playlist una vez
4. Reproducir la playlist un determinado número de veces
5. Vaciar la playlist
6. Salir
Elige una opción: 6
Saliendo del programa...
```

Conclusiones

El uso de funciones en lenguaje C para reservar y almacenar información de manera dinámica en tiempo de ejecución es una herramienta fundamental en la programación eficiente y flexible. A través de funciones como `malloc()`, calloc(), realloc() y free(), es posible gestionar la memoria de forma dinámica, optimizando el uso de los recursos del sistema y permitiendo la creación de estructuras de datos de tamaño variable. Esto resulta especialmente útil en aplicaciones que requieren un manejo eficiente de la memoria, como bases de datos, simulaciones o algoritmos que procesan grandes volúmenes de información. Sin embargo, un uso incorrecto de estas funciones puede generar errores como fugas de memoria o accesos indebidos, por lo que es crucial liberar correctamente la memoria asignada para garantizar un programa seguro y estable.