	Práctica 3
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 5

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.

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

// Estructura Contenedor (en lugar de usar simplemente un entero como valor)
struct Contenedor {
    int numeroIdentificacion;
    char tipoProducto[50];
    char paisOrigen[50];
    float pesoKg;
};

// Estructura Nodo
struct Nodo {
    struct Contenedor contenedor;
    struct Nodo* aptSigNodo;
};

// Estructura Pila
struct Pila {
    struct Nodo* aptNodoSuperior;
    int tamanioMaximo;
    int tamanioActual;
};

// Prototipos de las funciones de la pila
void inicializarPila(struct Pila* laPila, int capacidadMaxima);
void push(struct Pila* laPila, struct Contenedor contenedorAIngresar);
struct Contenedor pop(struct Pila* laPila);
int estaVacia(struct Pila* laPila);
int estaLlena(struct Pila* laPila);
void vaciarPila(struct Pila* laPila);
void mostrarMenuOpciones(void);
void obtenerDatosContenedor(struct Contenedor* nuevoContenedor);
void mostrarDatosContenedor(struct Contenedor contenedor);

// Función principal
int main() {
    struct Pila pilaContenedores;
    struct Contenedor contenedorTemporal;
    int opcion;
    char respuesta;

    // Inicializar la pila con capacidad máxima de 10
    inicializarPila(&pilaContenedores, 10);

    do {
        mostrarMenuOpciones();
        scanf("%d", &opcion);
        getchar(); // Limpiar el buffer

        switch(opcion) {
            case 1: // Apilar contenedor
                obtenerDatosContenedor(&contenedorTemporal);
                push(&pilaContenedores, contenedorTemporal);
                break;

            case 2: // Desapilar contenedor
                if (!estaVacia(&pilaContenedores)) {
                    contenedorTemporal = pop(&pilaContenedores);
                    printf("\n--- Datos del contenedor desapilado ---\n");
                    mostrarDatosContenedor(contenedorTemporal);
                }
                break;

            default:
                printf("Opción no válida. Por favor intente nuevamente.\n");
        }
```

```c
        printf("\n¿Desea realizar otra operación? (s/n): ");
        scanf(" %c", &respuesta);
        getchar(); // Limpiar el buffer

    } while (respuesta == 's' || respuesta == 'S');

    // Vaciar la pila al finalizar
    printf("\nVaciando la pila para despachar todos los contenedores restantes...\n");
    vaciarPila(&pilaContenedores);

    return 0;
}

void mostrarMenuOpciones() {
    printf("\n--- CONTROL DE BODEGA DE CONTENEDORES ---\n");
    printf("1. Apilar un contenedor\n");
    printf("2. Desapilar un contenedor\n");
    printf("Seleccione una opción: ");
}

void obtenerDatosContenedor(struct Contenedor* nuevoContenedor) {
    printf("\n--- Ingreso de nuevo contenedor ---\n");

    printf("Número de identificación: ");
    scanf("%d", &nuevoContenedor->numeroIdentificacion);
    getchar(); // Limpiar el buffer

    printf("Tipo de producto: ");
    fgets(nuevoContenedor->tipoProducto, 50, stdin);
    nuevoContenedor->tipoProducto[strcspn(nuevoContenedor->tipoProducto, "\n")] = 0; // Eliminar salto de línea

    printf("País de origen: ");
    fgets(nuevoContenedor->paisOrigen, 50, stdin);
    nuevoContenedor->paisOrigen[strcspn(nuevoContenedor->paisOrigen, "\n")] = 0; // Eliminar salto de línea

    printf("Peso en kilogramos: ");
    scanf("%f", &nuevoContenedor->pesoKg);
    getchar(); // Limpiar el buffer
}

void mostrarDatosContenedor(struct Contenedor contenedor) {
    printf("Número de identificación: %d\n", contenedor.numeroIdentificacion);
    printf("Tipo de producto: %s\n", contenedor.tipoProducto);
    printf("País de origen: %s\n", contenedor.paisOrigen);
    printf("Peso en kilogramos: %.2f kg\n", contenedor.pesoKg);
}

void inicializarPila(struct Pila* laPila, int capacidadMaxima) {
    laPila->aptNodoSuperior = NULL;
    laPila->tamanioMaximo = capacidadMaxima;
    laPila->tamanioActual = 0;
}

void push(struct Pila* laPila, struct Contenedor contenedorAIngresar) {
    if (estaLlena(laPila) == 1) {
        printf("\nAviso: La pila está llena. No se puede apilar más contenedores (máximo 10).\n");
    }
    else {
        struct Nodo* aptNuevoNodo = (struct Nodo*) calloc(1, sizeof(struct Nodo));
        if (aptNuevoNodo != NULL) {
            aptNuevoNodo->contenedor = contenedorAIngresar;
            aptNuevoNodo->aptSigNodo = laPila->aptNodoSuperior;
            laPila->aptNodoSuperior = aptNuevoNodo;
```

```
            laPila->tamanioActual = (laPila->tamanioActual) + 1;
            printf("\nContenedor #%d apilado exitosamente. Tamaño actual de la pila: %d\n",
                    contenedorAIngresar.numeroIdentificacion, laPila->tamanioActual);
        }
        else {
            printf("\nAviso: No se pudo reservar memoria para el nuevo contenedor\n");
        }
    }
}

struct Contenedor pop(struct Pila* laPila) {
    struct Contenedor contenedorVacio = {0, "", "", 0.0}; // Contenedor vacío para retornar en caso de error

    if (estaVacia(laPila) == 1) {
        printf("\nAviso: No se puede desapilar porque la pila está vacía\n");
        return contenedorVacio;
    }
    else {
        struct Nodo* aptAlNodoAEliminar = laPila->aptNodoSuperior;
        struct Contenedor contenedorDesapilado = aptAlNodoAEliminar->contenedor;
        laPila->aptNodoSuperior = aptAlNodoAEliminar->aptSigNodo;
        free(aptAlNodoAEliminar);
        laPila->tamanioActual = laPila->tamanioActual - 1;
        printf("\nContenedor desapilado exitosamente. Tamaño actual de la pila: %d\n", laPila->tamanioActual);
        return contenedorDesapilado;
    }
}

void vaciarPila(struct Pila* laPila) {
    int contadorContenedores = 0;

    while (estaVacia(laPila) != 1) {
        struct Contenedor contenedorDesapilado = pop(laPila);
        printf("\n--- Despachando contenedor #%d ---\n", contenedorDesapilado.numeroIdentificacion);
        mostrarDatosContenedor(contenedorDesapilado);
        contadorContenedores++;
    }

    printf("\nSe han despachado %d contenedores. La bodega está vacía.\n", contadorContenedores);
}

int estaLlena(struct Pila* laPila) {
    int bandera = 0;
    if (laPila->tamanioActual == laPila->tamanioMaximo) {
        bandera = 1;
    }
    else {
        bandera = 0;
    }
    return bandera;
}

int estaVacia(struct Pila* laPila) {
    int bandera = 0;
    if (laPila->aptNodoSuperior == NULL) {
        bandera = 1;
    }
    else {
        bandera = 0;
    }
    return bandera;
}
```

```
--- CONTROL DE BODEGA DE CONTENEDORES ---
1. Apilar un contenedor
2. Desapilar un contenedor
Seleccione una opción: 1

--- Ingreso de nuevo contenedor ---
Número de identificación: 201
Tipo de producto: piezas de aeronave
País de origen: México
Peso en kilogramos: 160.45

Contenedor #201 apilado exitosamente. Tamaño actual de la pila: 1

¿Desea realizar otra operación? (s/n): s

--- CONTROL DE BODEGA DE CONTENEDORES ---
1. Apilar un contenedor
2. Desapilar un contenedor
Seleccione una opción: 1

--- Ingreso de nuevo contenedor ---
Número de identificación: 305
Tipo de producto: Maíz
País de origen: México
Peso en kilogramos: 100.02

Contenedor #305 apilado exitosamente. Tamaño actual de la pila: 2

¿Desea realizar otra operación? (s/n): s

--- CONTROL DE BODEGA DE CONTENEDORES ---
1. Apilar un contenedor
2. Desapilar un contenedor
Seleccione una opción: 2

Contenedor desapilado exitosamente. Tamaño actual de la pila: 1
```

```
--- Datos del contenedor desapilado ---
Número de identificación: 305
Tipo de producto: Maíz
País de origen: México
Peso en kilogramos: 100.02 kg

¿Desea realizar otra operación? (s/n): s

--- CONTROL DE BODEGA DE CONTENEDORES ---
1. Apilar un contenedor
2. Desapilar un contenedor
Seleccione una opción: 2

Contenedor desapilado exitosamente. Tamaño actual de la pila: 0

--- Datos del contenedor desapilado ---
Número de identificación: 201
Tipo de producto: piezas de aeronave
País de origen: México
Peso en kilogramos: 160.45 kg

¿Desea realizar otra operación? (s/n): n

Vaciando la pila para despachar todos los contenedores restantes...

Se han despachado 0 contenedores. La bodega está vacía.
```

Ejercicio 2. Programa que implemente una estructura de datos Cola

En la cocina de un restaurante se lleva el control de los platillos de comida que piden los clientes. Para identificar cada platillo se recabará el nombre del cliente, su número de mesa y el nombre de platillo que quiere. El programa deberá preguntar qué se quiere hacer entre las siguientes dos opciones:

1. Indicar un platillo a preparar

2. Recoger un platillo

Si se indica opción 1, el programa deberá solicitar el nombre del cliente, el número de su mesa y el nombre del platillo que quiere. Estos datos conformarán un nuevo nodo que se agregará a la Cola de pedidos. Por el contrario, si se indica la opción 2, el programa deberá quitar de la cola el platillo que tiene el primer turno y mostrar en pantalla los datos de esa orden que se recoge. Ya sea que se haya indicado la opción 1 o 2 del menú de acciones, el programa deberá preguntar si hay más pedidos de platillos. En caso afirmativo, deberá repetir las acciones desde que se presenta en pantalla el menú de acciones. El comportamiento descrito se repetirá mientras se indique al programa que sí hay más clientes. Cuando ya no haya clientes, el programa vaciará la Cola para asegurarse que todas las órdenes pendientes se despachen.

Nota. No olvide revisar que la Cola esté llena o vacía según se requiera. Como máximo se podrán indicar 12 platillos para que se preparen.

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

#define MAX_PEDIDOS 12
#define LONG_NOMBRE 50

// Estructura del pedido
typedef struct Pedido {
    char nombre[LONG_NOMBRE];
    int mesa;
    char platillo[LONG_NOMBRE];
    struct Pedido* siguiente;
} Pedido;

// Estructura de la cola
typedef struct {
    Pedido* frente;
    Pedido* final;
    int tamaño;
} Cola;

// Inicializar la cola
void inicializarCola(Cola* cola) {
    cola->frente = cola->final = NULL;
    cola->tamaño = 0;
}

// Verificar si la cola está vacía
int estaVacia(Cola* cola) {
    return cola->frente == NULL;
}

// Añadir un pedido a la cola
void agregarPedido(Cola* cola, char nombre[], int mesa, char platillo[]) {
    if (cola->tamaño >= MAX_PEDIDOS) {
        printf("⚠ La cola está llena. No se pueden agregar más pedidos.\n");
        return;
    }

    Pedido* nuevo = (Pedido*)malloc(sizeof(Pedido));
    strcpy(nuevo->nombre, nombre);
    nuevo->mesa = mesa;
    strcpy(nuevo->platillo, platillo);
    nuevo->siguiente = NULL;
```

```c
    if (estaVacia(cola)) {
        cola->frente = cola->final = nuevo;
    } else {
        cola->final->siguiente = nuevo;
        cola->final = nuevo;
    }
    cola->tamaño++;
    printf("☑ Pedido agregado: %s, Mesa: %d, Platillo: %s\n", nombre, mesa, pl
}

// Retirar un pedido de la cola
void recogerPedido(Cola* cola) {
    if (estaVacia(cola)) {
        printf("⚠ No hay pedidos para recoger.\n");
        return;
    }

    Pedido* temp = cola->frente;
    printf("🍽 Recogiendo pedido: %s, Mesa: %d, Platillo: %s\n", temp->nombre,

    cola->frente = cola->frente->siguiente;
    free(temp);
    cola->tamaño--;

    if (cola->frente == NULL) {
        cola->final = NULL;
    }
}

// Vaciar la cola para despachar los pedidos pendientes
void vaciarCola(Cola* cola) {
    while (!estaVacia(cola)) {
        recogerPedido(cola);
    }
}

// Menú principal
int main() {
    Cola colaPedidos;
    inicializarCola(&colaPedidos);

    int opcion, mesa;
```

```c
void vaciarCola(Cola* cola) {
    while (!estaVacia(cola)) {
        recogerPedido(cola);
    }
}

// Menú principal
int main() {
    Cola colaPedidos;
    inicializarCola(&colaPedidos);

    int opcion, mesa;
    char nombre[LONG_NOMBRE], platillo[LONG_NOMBRE];

    do {
        printf("\nMenú:\n");
        printf("1. Indicar un platillo a preparar\n");
        printf("2. Recoger un platillo\n");
        printf("3. Salir\n");
        printf("Elige una opción: ");
        scanf("%d", &opcion);
        getchar(); // Limpiar buffer

        switch (opcion) {
            case 1:
                if (colaPedidos.tamaño < MAX_PEDIDOS) {
                    printf("Nombre del cliente: ");
                    fgets(nombre, LONG_NOMBRE, stdin);
                    nombre[strcspn(nombre, "\n")] = 0;

                    printf("Número de mesa: ");
                    scanf("%d", &mesa);
                    getchar();

                    printf("Nombre del platillo: ");
                    fgets(platillo, LONG_NOMBRE, stdin);
                    platillo[strcspn(platillo, "\n")] = 0;

                    agregarPedido(&colaPedidos, nombre, mesa, platillo);
                } else {
                    printf("⚠ La cola está llena. No se pueden añadir más pedid
                }
                break;

            case 2:
```

```c
        printf("2. Recoger un platillo\n");
        printf("3. Salir\n");
        printf("Elige una opción: ");
        scanf("%d", &opcion);
        getchar(); // Limpiar buffer

        switch (opcion) {
            case 1:
                if (colaPedidos.tamaño < MAX_PEDIDOS) {
                    printf("Nombre del cliente: ");
                    fgets(nombre, LONG_NOMBRE, stdin);
                    nombre[strcspn(nombre, "\n")] = 0;

                    printf("Número de mesa: ");
                    scanf("%d", &mesa);
                    getchar();

                    printf("Nombre del platillo: ");
                    fgets(platillo, LONG_NOMBRE, stdin);
                    platillo[strcspn(platillo, "\n")] = 0;

                    agregarPedido(&colaPedidos, nombre, mesa, platillo);
                } else {
                    printf("⚠ La cola está llena. No se pueden añadir más pedid
                }
                break;

            case 2:
                recogerPedido(&colaPedidos);
                break;

            case 3:
                printf("🧹 Vaciando cola antes de salir...\n");
                vaciarCola(&colaPedidos);
                printf("☑ Todos los pedidos fueron despachados.\n");
                break;

            default:
                printf("✖ Opción no válida. Intente de nuevo.\n");
        }
    } while (opcion != 3);

    return 0;
}
```

```
rexta@DESKTOP-J5AEL8D ~
$ ./a.exe

Menú:
1. Indicar un platillo a preparar
2. Recoger un platillo
3. Salir
Elige una opción: 1
Nombre del cliente: Marco Fuentes
Número de mesa: 2
Nombre del platillo: arroz
☑ Pedido agregado: Marco Fuentes, Mesa: 2, Platillo: arroz

Menú:
1. Indicar un platillo a preparar
2. Recoger un platillo
3. Salir
Elige una opción: 3
🧹 Vaciando cola antes de salir...
🍽 Recogiendo pedido: Marco Fuentes, Mesa: 2, Platillo: arroz
☑ Todos los pedidos fueron despachados.
```

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.