[**Descripción: 1**](#_o8n1va16r1m2)

[**Estructuras de Datos Implementadas: 1**](#_hhkj452yov7z)

[**Método de Ordenamiento Implementado: 1**](#_8sh5jdgh1cmj)

[**Funciones Auxiliares: 1**](#_x2cptm23fymo)

[**Funciones Principales: 2**](#_uxrobciaa9cl)

[**Estructuras de Datos: 2**](#_tj6jr9b78na9)

[1. Pilas: 2](#_joebkkt3unkq)

[1.1 Estructura de la Pila: 2](#_w2nc5kgrrurp)

[1.2 Funciones Relacionadas con Pilas: 3](#_qbm9fj8d1zwl)

[1.2.1. `inicializarPila`: 3](#_p3ig4oezkr9b)

[1.2.2. `push`: 3](#_ia2ypyazjqra)

[1.2.3. `pop`: 3](#_wnec6y4z3e2o)

[1.2.4. `mostrarPila`: 4](#_mud7hegglg3m)

[1.2.5. `modificarPila`: 4](#_y8dqgp157u5q)

[1.3 Menú de Pilas (`menuPilas`): 4](#_yb9lw44l5qx2)

[2. Colas: 4](#_4ln7z62nupm3)

[2.1 Estructura del Código: Implementación de Colas en C 5](#_betwejules01)

[2.1.1 Estructura de la Cola: 5](#_u46st9q5a31e)

[2.1.2. Funciones Relacionadas con Colas: 5](#_vg4zbafq2np6)

[2.2.1 `inicializarCola`: 5](#_674lc0mfygvp)

[2.2.2 `encolar`: 5](#_pzk17vbs5gl3)

[2.2.3 `desencolar`: 6](#_qftf0so88y7u)

[2.2.4 `mostrarCola`: 6](#_3039cochq7ez)

[2.3. Menú de Colas (`menuColas`): 7](#_cevdwzl4u5tr)

[3. Listas: 7](#_g2bh67j8rbk)

[3.1 Estructura del Código: Implementación de Listas en C 7](#_1ootwk90lhws)

[3.1.1. Estructura de la Lista: 7](#_97v8wcr1po17)

[3.2. Funciones Relacionadas con Listas: 7](#_w0gz7awnu5u1)

[3.2.1 `inicializarLista`: 7](#_65mm9oxiaz0)

[3.2.2 `insertarAlInicio`: 7](#_9x8szni5mx7d)

[3.2.3 `mostrarLista`: 8](#_vsz8rvod7hf8)

[3.2.4 `eliminarLista`: 8](#_834suggk3rir)

[3.3. Menú de Listas (`menuListas`): 9](#_f0ypbrlr81oh)

[4. Árboles: 9](#_k3e81oylyg9s)

[4.1 Estructura del Código: Implementación de Árboles en C 9](#_lah0l8nvuqyp)

[4.1.1. Estructuras Relacionadas con Árboles: 9](#_9wqlinfqjk80)

[4.2. Funciones Relacionadas con Árboles: 9](#_ylphvx86wphq)

[4.2.1 `crearNodoArbol`: 9](#_osflgtfk9t4j)

[4.2.2 `insertarEnArbol`: 10](#_dakdazh5ro7w)

[4.2.3 `mostrarArbolInOrden`: 10](#_m1n8h3hfp738)

[4.3. Menú de Árboles (`menuArboles`): 10](#_aunauo5b3i6k)

[5 Ordenamiento: 10](#_7azodwgxd6su)

[5.1 Estructura del Código: Ordenamiento por Inserción en C 11](#_4ra9vpsckl9f)

[5.1.1. Funciones Relacionadas con el Ordenamiento por Inserción: 11](#_ccrgtem3fhon)

[5.1.1.1 `ordenamientoInsercion`: 11](#_5j32t9wt7hju)

[5.1.1.2 `mostrarArreglo`: 11](#_o5elx5e30g3l)

[5.2. Menú de Ordenamiento por Inserción (`menuOrdenamientoInsercion`): 12](#_k3s5oel70fcu)

[Menú Principal: 12](#_ead3j7jvh57g)

[Estructura del Código: Menú Principal en C 12](#_ripgymij72lw)

[1. Función Principal (`main`): 12](#_7m57ahnyucgx)

[2. Menú Principal: 13](#_jy2xod1zg0gs)

[3. Llamadas a Funciones según la Opción Seleccionada: 13](#_ob7c5hvj7pxu)

[4. Mensajes al Usuario: 14](#_1i4xlkvq8qoe)

[5. Bucle `do-while`: 14](#_jq3kd2e8tr0p)

[**Menús Específicos: 14**](#_d1ccrtyolrb9)

[1. Menú de Pilas (`menuPilas`): 14](#_yh4xg3sj5rba)

[Descripción: 15](#_lvqxqmqhj33k)

[2. Menú de Colas (`menuColas`): 15](#_6tifbipp21wb)

[Descripción: 16](#_9lkjwp70dhi8)

[3. Menú de Listas (`menuListas`): 16](#_6cl22hax279b)

[Descripción: 17](#_8ldjr63cj28n)

[4. Menú de Árboles (`menuArboles`): 17](#_k17w6hech80g)

[Descripción: 18](#_js0kth15pt2e)

[5. Menú de Ordenamiento por Inserción (`menuOrdenamientoInsercion`): 18](#_jg7gnqecv1qc)

[Descripción: 18](#_1uxreigerrf5)

[Notas: 19](#_l2alu6go6718)

[Interacción con el Usuario: 19](#_gde6ojvdkrx3)

[Consideraciones Importantes: 19](#_29rozib04uw0)

# **Descripción:**

Este código realizado en lenguaje **C** implementa diversas estructuras de datos (pilas, colas, listas, árboles) y un método de ordenamiento (ordenamiento por inserción) para gestionar información de personas. El programa presenta un menú principal con opciones para operar con cada estructura y el método de ordenamiento.

# **Estructuras de Datos Implementadas:**

1. Pilas

2. Colas

3. Listas

4. Árboles

# **Método de Ordenamiento Implementado:**

- Ordenamiento por Inserción

# **Funciones Auxiliares:**

- inicializarPila

- push

- pop

- mostrarPila

- modificarPila

- inicializarCola

- encolar

- desencolar

- mostrarCola

- inicializarLista

- insertarAlInicio

- mostrarLista

- eliminarLista

- crearNodoArbol

- insertarEnArbol

- mostrarArbolInOrden

- ordenamientoInsercion

- mostrarArreglo

# 

# **Funciones Principales:**

- menuPilas

- menuColas

- menuListas

- menuArboles

- menuOrdenamientoInsercion

- main

# **Estructuras de Datos:**

## **1. Pilas:**

- La estructura de pila se define con un nodo que contiene información sobre una persona y un puntero al siguiente nodo.

- Funciones como `inicializarPila`, `push`, `pop`, `mostrarPila` y `modificarPila` permiten realizar operaciones en la pila.

En el código proporcionado, las pilas se implementan utilizando la estructura de datos de una pila enlazada. A continuación, se explica el funcionamiento de las pilas en el código:

### **1.1 Estructura de la Pila:**

// Estructura para la pila

struct Pila {

struct Nodo\* tope;

};

La estructura `Pila` tiene un único miembro llamado `tope`, que es un puntero a un nodo. Cada nodo contiene información sobre una persona (`struct Persona`) y un puntero al siguiente nodo en la pila.

// Definición de la estructura Nodo para listas y árboles

struct Nodo {

struct Persona datos;

struct Nodo\* siguiente;

};

### **1.2 Funciones Relacionadas con Pilas:**

#### **1.2.1. `inicializarPila`:**

#### 

void inicializarPila(struct Pila\* pila) {

pila->tope = NULL;

}

Esta función inicializa una pila vacía estableciendo el puntero `tope` en `NULL`.

#### **1.2.2. `push`:**

#### 

void push(struct Pila\* pila, struct Persona datos) {

struct Nodo\* nuevoNodo = (struct Nodo\*)malloc(sizeof(struct Nodo));

// ... (verificación de la asignación de memoria)

nuevoNodo->datos = datos;

nuevoNodo->siguiente = pila->tope;

pila->tope = nuevoNodo;

}

La función `push` agrega una nueva persona a la pila. Crea un nuevo nodo, asigna los datos de la persona y ajusta el puntero `tope` para apuntar al nuevo nodo.

#### **1.2.3. `pop`:**

void pop(struct Pila\* pila) {

if (pila->tope == NULL) {

printf("La pila está vacía. No se puede eliminar.\n");

return;

}

struct Nodo\* temp = pila->tope;

pila->tope = temp->siguiente;

free(temp);

}

La función `pop` elimina la persona en la cima de la pila. Verifica si la pila está vacía, ajusta el puntero `tope` y libera la memoria del nodo eliminado.

#### **1.2.4. `mostrarPila`:**

void mostrarPila(struct Pila pila) {

if (pila.tope == NULL) {

printf("La pila está vacía.\n");

return;

}

printf("Contenido de la pila:\n");

struct Nodo\* actual = pila.tope;

while (actual != NULL) {

// ... (imprimir los datos de la persona en el nodo)

actual = actual->siguiente;

}

}

La función `mostrarPila` imprime en la consola el contenido de la pila, recorriendo los nodos y mostrando los datos de cada persona.

#### **1.2.5. `modificarPila`:**

void modificarPila(struct Pila\* pila) {

// ... (verificar si la persona a modificar está en la pila y realizar la modificación)

}

La función `modificarPila` permite al usuario modificar la información de una persona en la pila.

### **1.3 Menú de Pilas (`menuPilas`):**

El menú específico para las pilas (`menuPilas`) utiliza estas funciones para realizar operaciones como agregar, modificar, eliminar y mostrar personas en la pila. Los mensajes en el menú guían al usuario a través de estas operaciones.

## **2. Colas:**

- Similar a las pilas, la estructura de cola también se define con un nodo y un puntero al siguiente nodo.

- Funciones como `inicializarCola`, `encolar`, `desencolar` y `mostrarCola` facilitan las operaciones en la cola.

### **2.1 Estructura del Código: Implementación de Colas en C**

#### **2.1.1 Estructura de la Cola:**

struct Cola {

struct Nodo\* frente;

struct Nodo\* final;

};

La estructura `Cola` consta de dos punteros, `frente` y `final`, que indican el principio y el final de la cola, respectivamente. Cada `Nodo` almacena los datos de una persona y apunta al siguiente nodo en la cola.

### **2.1.2. Funciones Relacionadas con Colas:**

#### **2.2.1 `inicializarCola`:**

void inicializarCola(struct Cola\* cola) {

cola->frente = NULL;

cola->final = NULL;

}

La función `inicializarCola` establece una cola vacía inicializando los punteros `frente` y `final` en `NULL`.

#### **2.2.2 `encolar`:**

void encolar(struct Cola\* cola, struct Persona datos) {

struct Nodo\* nuevoNodo = (struct Nodo\*)malloc(sizeof(struct Nodo));

// ... (verificación de la asignación de memoria)

nuevoNodo->datos = datos;

nuevoNodo->siguiente = NULL;

if (cola->final == NULL) {

cola->frente = nuevoNodo;

cola->final = nuevoNodo;

} else {

cola->final->siguiente = nuevoNodo;

cola->final = nuevoNodo;

}

}

La función `encolar` agrega una nueva persona al final de la cola, creando un nuevo nodo y ajustando los punteros `frente` y `final` según sea necesario.

#### **2.2.3 `desencolar`:**

void desencolar(struct Cola\* cola) {

if (cola->frente == NULL) {

printf("La cola está vacía. No se puede desencolar.\n");

return;

}

struct Nodo\* temp = cola->frente;

cola->frente = temp->siguiente;

if (cola->frente == NULL) {

// Si la cola está vacía, también actualiza el puntero al final

cola->final = NULL;

}

free(temp);

}

La función `desencolar` elimina la persona en el frente de la cola, ajustando el puntero `frente` y liberando la memoria del nodo eliminado. Si la cola queda vacía, también actualiza el puntero `final`.

#### **2.2.4 `mostrarCola`:**

void mostrarCola(struct Cola cola) {

if (cola.frente == NULL) {

printf("La cola está vacía.\n");

return;

}

printf("Contenido de la cola:\n");

struct Nodo\* actual = cola.frente;

while (actual != NULL) {

// ... (imprimir los datos de la persona en el nodo)

actual = actual->siguiente;

}

}

La función `mostrarCola` imprime el contenido de la cola, recorriendo los nodos y mostrando los datos de cada persona.

### **2.3. Menú de Colas (`menuColas`):**

El menú para las colas utiliza estas funciones para operaciones como agregar, eliminar y mostrar personas en la cola. Los mensajes guían al usuario a través de estas operaciones.

## **3. Listas:**

- La lista enlazada se implementa con nodos que contienen información de personas y un puntero al siguiente nodo.

- Operaciones básicas como `inicializarLista`, `insertarAlInicio`, `mostrarLista` y `eliminarLista` están disponibles.

### **3.1 Estructura del Código: Implementación de Listas en C**

#### **3.1.1. Estructura de la Lista:**

struct Lista {

struct Nodo\* inicio;

};

La estructura `Lista` contiene un puntero `inicio` que señala al primer nodo de la lista. Cada nodo almacena los datos de una persona y apunta al siguiente nodo en la lista.

### **3.2. Funciones Relacionadas con Listas:**

#### **3.2.1 `inicializarLista`:**

void inicializarLista(struct Lista\* lista) {

lista->inicio = NULL;

}

La función `inicializarLista` establece una lista vacía inicializando el puntero `inicio` en `NULL`.

#### **3.2.2 `insertarAlInicio`:**

void insertarAlInicio(struct Lista\* lista, struct Persona datos) {

struct Nodo\* nuevoNodo = (struct Nodo\*)malloc(sizeof(struct Nodo));

// ... (verificación de la asignación de memoria)

nuevoNodo->datos = datos;

nuevoNodo->siguiente = lista->inicio;

lista->inicio = nuevoNodo;

}

La función `insertarAlInicio` agrega una nueva persona al inicio de la lista, creando un nuevo nodo y ajustando el puntero `inicio`.

#### **3.2.3 `mostrarLista`:**

void mostrarLista(struct Lista lista) {

if (lista.inicio == NULL) {

printf("La lista está vacía.\n");

return;

}

printf("Contenido de la lista:\n");

struct Nodo\* actual = lista.inicio;

while (actual != NULL) {

// ... (imprimir los datos de la persona en el nodo)

actual = actual->siguiente;

}

}

La función `mostrarLista` imprime el contenido de la lista, recorriendo los nodos y mostrando los datos de cada persona.

#### **3.2.4 `eliminarLista`:**

void eliminarLista(struct Lista\* lista) {

while (lista->inicio != NULL) {

struct Nodo\* temp = lista->inicio;

lista->inicio = temp->siguiente;

free(temp);

}

}

La función `eliminarLista` elimina todos los nodos de la lista, liberando la memoria asignada a cada nodo.

### **3.3. Menú de Listas (`menuListas`):**

El menú para las listas utiliza estas funciones para operaciones como agregar personas al inicio, mostrar la lista y eliminar la lista completa. Los mensajes guían al usuario a través de estas operaciones.

## **4. Árboles:**

- Un árbol binario de búsqueda se utiliza para almacenar información de personas ordenadas por sus nombres.

- Funciones como `crearNodoArbol`, `insertarEnArbol` y `mostrarArbolInOrden` permiten la manipulación y visualización del árbol.

### **4.1 Estructura del Código: Implementación de Árboles en C**

#### **4.1.1. Estructuras Relacionadas con Árboles:**

// Estructura para el árbol binario de búsqueda

struct NodoArbol {

struct Persona datos;

struct NodoArbol\* izquierda;

struct NodoArbol\* derecha;

};

// Estructura para el árbol

struct Arbol {

struct NodoArbol\* raiz;

};

Se utiliza una estructura de nodo `NodoArbol` para almacenar los datos de una persona y apuntar a los nodos izquierdo y derecho. La estructura `Arbol` contiene un puntero `raiz` que señala al nodo raíz del árbol.

### **4.2. Funciones Relacionadas con Árboles:**

#### **4.2.1 `crearNodoArbol`:**

struct NodoArbol\* crearNodoArbol(struct Persona datos) {

struct NodoArbol\* nuevoNodo = (struct NodoArbol\*)malloc(sizeof(struct NodoArbol));

// ... (verificación de la asignación de memoria)

nuevoNodo->datos = datos;

nuevoNodo->izquierda = NULL;

nuevoNodo->derecha = NULL;

return nuevoNodo;

}

La función `crearNodoArbol` crea un nuevo nodo para el árbol, asignando memoria y estableciendo los datos y punteros izquierdo y derecho en `NULL`.

#### **4.2.2 `insertarEnArbol`:**

void insertarEnArbol(struct Arbol\* arbol, struct Persona datos) {

if (arbol->raiz == NULL) {

arbol->raiz = crearNodoArbol(datos);

} else {

// ... (verificación y recorrido para insertar el nuevo nodo en la posición adecuada)

}

}

La función `insertarEnArbol` agrega una nueva persona al árbol. Si el árbol está vacío, crea un nuevo nodo en la raíz; de lo contrario, recorre el árbol para encontrar la posición adecuada e inserta el nuevo nodo.

#### **4.2.3 `mostrarArbolInOrden`:**

void mostrarArbolInOrden(struct NodoArbol\* nodo) {

if (nodo != NULL) {

mostrarArbolInOrden(nodo->izquierda);

// ... (imprimir los datos de la persona en el nodo)

mostrarArbolInOrden(nodo->derecha);

}

}

La función `mostrarArbolInOrden` realiza un recorrido inorden del árbol, mostrando los datos de las personas en el orden correcto.

### **4.3. Menú de Árboles (`menuArboles`):**

El menú para los árboles utiliza estas funciones para operaciones como agregar personas al árbol y mostrar las personas en orden. Los mensajes guían al usuario a través de estas operaciones.

## **5 Ordenamiento:**

- El método de ordenamiento por inserción se implementa en la función `ordenamientoInsercion`.

- También se proporciona una función `mostrarArreglo` para visualizar el contenido antes y después del ordenamiento.

### **5.1 Estructura del Código: Ordenamiento por Inserción en C**

#### **5.1.1. Funciones Relacionadas con el Ordenamiento por Inserción:**

##### **5.1.1.1 `ordenamientoInsercion`:**

void ordenamientoInsercion(struct Persona arreglo[], int n) {

int i, j;

struct Persona actual;

for (i = 1; i < n; ++i) {

actual = arreglo[i];

j = i - 1;

while (j >= 0 && strcmp(arreglo[j].nombre, actual.nombre) > 0) {

arreglo[j + 1] = arreglo[j];

--j;

}

arreglo[j + 1] = actual;

}

}

La función `ordenamientoInsercion` implementa el algoritmo de ordenamiento por inserción. Toma un arreglo de personas y su tamaño como entrada.

- Comienza desde el segundo elemento del arreglo (`i = 1`) y compara cada elemento con los elementos anteriores.

- Si el nombre de la persona actual (`actual.nombre`) es menor que el nombre de la persona en la posición `j`, se realiza un desplazamiento a la derecha para hacer espacio para la persona actual.

- El bucle interno continúa moviéndose hacia la izquierda hasta encontrar la posición adecuada para la persona actual.

- Finalmente, se coloca la persona actual en su posición ordenada.

##### **5.1.1.2 `mostrarArreglo`:**

void mostrarArreglo(struct Persona arreglo[], int n) {

// ... (imprimir el contenido del arreglo)

}

La función `mostrarArreglo` se utiliza para imprimir el contenido del arreglo antes y después de aplicar el ordenamiento por inserción.

### **5.2. Menú de Ordenamiento por Inserción (`menuOrdenamientoInsercion`):**

El menú para el ordenamiento por inserción utiliza estas funciones para:

- Solicitar al usuario la cantidad de personas y sus datos.

- Mostrar las personas antes de la ordenación.

- Aplicar el ordenamiento por inserción.

- Mostrar las personas después de la ordenación.

Los mensajes en el menú proporcionan instrucciones claras y guían al usuario a través del proceso.

# Menú Principal:

- En la función `main`, se presenta un menú principal que permite al usuario elegir entre las diferentes estructuras de datos y el método de ordenamiento.

- Cada opción del menú llama a funciones específicas que gestionan las operaciones relacionadas con la estructura de datos seleccionada.

## Estructura del Código: Menú Principal en C

### 1. Función Principal (`main`):

int main() {

int opcion;

do {

printf("\nMenú Principal:\n");

printf("1. Pilas\n");

printf("2. Colas\n");

printf("3. Listas\n");

printf("4. Árboles\n");

printf("5. Método de Ordenamiento por Inserción\n");

printf("0. Salir\n");

printf("\n\nIngrese su opción: ");

scanf("%d", &opcion);

switch (opcion) {

case 1:

menuPilas();

break;

case 2:

menuColas();

break;

case 3:

menuListas();

break;

case 4:

menuArboles();

break;

case 5:

menuOrdenamientoInsercion();

break;

case 0:

printf("\n\nSaliendo del programa. ¡Hasta luego!\n");

break;

default:

printf("\nOpción no válida. Por favor, ingrese una opción válida.\n");

}

} while (opcion != 0);

return 0;

}

### 2. Menú Principal:

- El programa comienza ejecutando la función `main`, que representa el menú principal del programa.

- Muestra un menú con varias opciones numeradas.

- Utiliza un bucle `do-while` para permitir que el usuario realice varias operaciones antes de salir del programa (opción `0`).

- Lee la opción ingresada por el usuario.

- Un bloque `switch` dirige la ejecución a la función correspondiente según la opción seleccionada.

- Si la opción es `0`, el programa se cierra con un mensaje de despedida. Si es una opción no válida, se muestra un mensaje de advertencia.

### 3. Llamadas a Funciones según la Opción Seleccionada:

- `menuPilas()`: Llama al menú para la manipulación de pilas.

- `menuColas()`: Llama al menú para la manipulación de colas.

- `menuListas()`: Llama al menú para la manipulación de listas.

- `menuArboles()`: Llama al menú para la manipulación de árboles.

- `menuOrdenamientoInsercion()`: Llama al menú para el ordenamiento por inserción.

### 4. Mensajes al Usuario:

- Mensajes claros indican las opciones disponibles y guían al usuario a través del menú.

- Se manejan situaciones de opción no válida con un mensaje informativo.

### **5. Bucle `do-while`:**

- Permite que el usuario realice operaciones hasta que decide salir del programa (`opcion = 0`).

- Garantiza que el menú se muestre al menos una vez, incluso si el usuario elige salir inmediatamente.

# **Menús Específicos:**

- Se han creado menús específicos para cada estructura de datos (`menuPilas`, `menuColas`, `menuListas`, `menuArboles`, `menuOrdenamientoInsercion`) para proporcionar una interfaz más amigable al usuario.

Menús Específicos en el Programa: Pilas, Colas, Listas, Árboles, Ordenamiento por Inserción.

## **1. Menú de Pilas (`menuPilas`):**

void menuPilas() {

// ... (Inicialización y declaraciones)

do {

printf("\nMenú de Pilas:\n");

printf("1. Agregar persona\n");

printf("2. Modificar persona\n");

printf("3. Eliminar persona\n");

printf("4. Mostrar pilas\n");

printf("0. Volver al menú principal\n");

printf("Ingrese su opción: ");

scanf("%d", &opcion);

switch (opcion) {

case 1:

// ... (Agregar persona a la pila)

break;

case 2:

// ... (Modificar persona en la pila)

break;

case 3:

// ... (Eliminar persona de la pila)

break;

case 4:

// ... (Mostrar contenido de la pila)

break;

case 0:

printf("\n\nVolviendo al menú principal.\n");

break;

default:

printf("\nOpción no válida. Por favor, ingrese una opción válida.\n");

}

} while (opcion != 0);

}

### **Descripción:**

- Menú específico para la manipulación de pilas.

- Opciones para agregar, modificar, eliminar y mostrar personas en la pila.

- La opción `0` permite regresar al menú principal.

- El bucle `do-while` permite al usuario realizar operaciones hasta que decide volver al menú principal.

## **2. Menú de Colas (`menuColas`):**

void menuColas() {

// ... (Inicialización y declaraciones)

do {

printf("\nMenú de Colas:\n");

printf("1. Agregar persona a la cola\n");

printf("2. Eliminar persona de la cola\n");

printf("3. Mostrar cola\n");

printf("0. Volver al menú principal\n");

printf("\n\nIngrese su opción: ");

scanf("%d", &opcion);

switch (opcion) {

case 1:

// ... (Agregar persona a la cola)

break;

case 2:

// ... (Eliminar persona de la cola)

break;

case 3:

// ... (Mostrar contenido de la cola)

break;

case 0:

printf("\n\nVolviendo al menú principal.\n");

break;

default:

printf("\nOpción no válida. Por favor, ingrese una opción válida.\n");

}

} while (opcion != 0);

}

### **Descripción:**

- Menú específico para la manipulación de colas.

- Opciones para agregar, eliminar y mostrar personas en la cola.

- La opción `0` permite regresar al menú principal.

- El bucle `do-while` permite al usuario realizar operaciones hasta que decide volver al menú principal.

## **3. Menú de Listas (`menuListas`):**

void menuListas() {

// ... (Inicialización y declaraciones)

do {

printf("\nMenú de Listas:\n");

printf("1. Agregar persona al inicio de la lista\n");

printf("2. Mostrar lista\n");

printf("3. Eliminar lista\n");

printf("0. Volver al menú principal\n");

printf("\n\nIngrese su opción: ");

scanf("%d", &opcion);

switch (opcion) {

case 1:

// ... (Agregar persona al inicio de la lista)

break;

case 2:

// ... (Mostrar contenido de la lista)

break;

case 3:

// ... (Eliminar lista)

break;

case 0:

printf("\n\nVolviendo al menú principal.\n");

break;

default:

printf("\nOpción no válida. Por favor, ingrese una opción válida.\n");

}

} while (opcion != 0);

}

### **Descripción:**

- Menú específico para la manipulación de listas.

- Opciones para agregar personas al inicio de la lista, mostrar la lista y eliminar la lista.

- La opción `0` permite regresar al menú principal.

- El bucle `do-while` permite al usuario realizar operaciones hasta que decide volver al menú principal.

## **4. Menú de Árboles (`menuArboles`):**

void menuArboles() {

// ... (Inicialización y declaraciones)

do {

printf("\nMenú de Árboles:\n");

printf("1. Agregar persona al árbol\n");

printf("2. Mostrar personas en orden\n");

printf("0. Volver al menú principal\n");

printf("\nIngrese su opción: ");

scanf("%d", &opcion);

switch (opcion) {

case 1:

// ... (Agregar persona al árbol)

break;

case 2:

// ... (Mostrar personas en orden)

break;

case 0:

printf("\n\nVolviendo al menú principal.\n");

break;

default:

printf("\nOpción no válida. Por favor, ingrese una opción válida.\n");

}

} while (opcion != 0);

}

### **Descripción:**

- Menú específico para la manipulación de árboles.

- Opciones para agregar personas al árbol y mostrar las personas en orden.

- La opción `0` permite regresar al menú principal.

- El bucle `do-while` permite al usuario realizar operaciones hasta que decide volver al menú principal.

## **5. Menú de Ordenamiento por Inserción (`menuOrdenamientoInsercion`):**

void menuOrdenamientoInsercion() {

// ... (Inicialización y declaraciones)

printf("\nIngrese la cantidad de personas para el ordenamiento: ");

scanf("%d", &n);

struct Persona personas[n];

for (int i = 0; i < n; ++i) {

// ... (Solicitar datos de personas)

}

printf("\n\nPersonas antes del ordenamiento:\n");

mostrarArreglo(personas, n);

ordenamientoInsercion(personas, n);

printf("\n\nPersonas después del ordenamiento por inserción:\n");

mostrarArreglo(personas, n);

}

### **Descripción:**

- Menú específico para el ordenamiento por inserción.

- Solicita al usuario la cantidad de personas y sus datos para realizar el ordenamiento.

- Muestra las personas antes y después del proceso de ordenamiento por inserción.

# **Notas:**

- Cada menú se ejecuta en un bucle

`do-while`, permitiendo que el usuario realice operaciones hasta que decide volver al menú principal.

- Cada opción dentro de un menú realiza operaciones específicas relacionadas con la estructura de datos correspondiente.

- Se han omitido detalles de implementación específicos para simplificar la explicación y centrarse en la estructura general del código.

# **Interacción con el Usuario:**

- El programa solicita al usuario ingresar información sobre personas según la estructura de datos o el método de ordenamiento seleccionado.

- Se incluyen mensajes informativos y de error para guiar al usuario a través de las opciones disponibles.

# **Consideraciones Importantes:**

- El código evita alterar funciones ya implementadas al agregar nuevas funcionalidades, siguiendo buenas prácticas de desarrollo.

- Las funciones están diseñadas para ser modulares y reutilizables, facilitando su mantenimiento y ampliación.