Nombre:Diego Alejandro Angarita Arboleda

Ejercicio 1:
Problema: Dado un arreglo de números enteros, implemente un algoritmo que genere un nuevo arreglo en el que los elementos aparezcan en orden inverso. Además, se debe desarrollar una función para mostrar los elementos del arreglo.

Requisitos:

A. Definir una función InvertirArreglo que reciba un arreglo y retorne otro arreglo con los elementos invertidos.
B. Definir una función MostrarArreglo que recorra y muestre los elementos de un arreglo en pantalla.
C. Demostración:

• Cree un arreglo de al menos 5 elementos.
• Muestre el arreglo original.
• Aplique la función de inversión y muestre el arreglo resultante.

Solución

Para realizar este ejercicio quise realizar un algoritmo en el cual el usuario ingresara por sí mismo el número de elementos del arreglo y el código no termine de ejecutarse por si acaso el usuario ingresa un dato incorrecto (ej.  caracteres o números negativos), como se puede observar en el código.

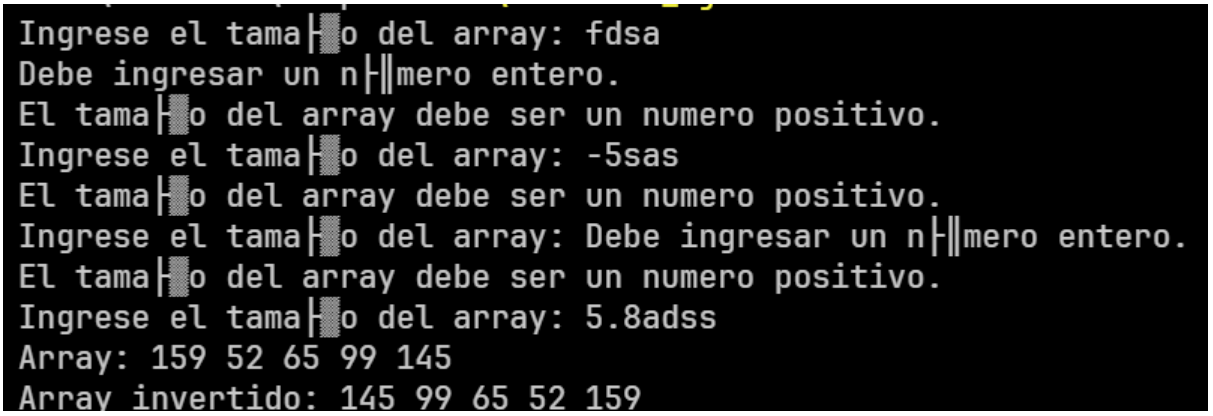
```
int n;
while (true)//ciclo para validar la entrada del usuario
{
    cout << "Ingrese el tamaño del array: ";
    cin >> n;

    if (cin.fail()) {
        // Si la entrada no es un número
        cin.clear(); // Limpia el estado de error
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Descarta la entrada incorrecta
        cout << "Debe ingresar un número entero." << endl;
    }

    if (n > 0) {
        int arr[n];
        llenarArray(arr, n);
        imprimirArray(arr, n);
        cout << endl;
        invertirArreglo(arr, n);
        break;
    } else {
        cout << "El tamaño del array debe ser un numero positivo." << endl;
    }
```

el código contenido en el primer condicional fue generado por el LLM chat GPT 4.1 mediante github copilot

Cabe aclarar que el algoritmo si acepta entradas como 8oiujhbn o 4dsvc, ya que comienza con un número entero positivo el cual será tomado como el tamaño del arreglo.




debido al formato que se usa, la terminal no muestra la letra ñ de forma correcta

Ejercicio 2:
Problema:  Dada una lista enlazada simple (ya ordenada en forma ascendente), implemente un algoritmo que permita insertar nuevo nodo de manera que se mantenga el orden de la lista. Además, desarrolle una función para mostrar los elementos de la lista

Requisitos:
A. Definir la estructura del nodo: Cada nodo debe contener un valor entero y una referencia al siguiente nodo.
B. Implementar la función InsertarOrdenado que reciba la cabeza de la lista y el valor a insertar, ubicando el nuevo nodo en la posición correcta para mantener el orden.
C. Implementar la función MostrarLista para recorrer y mostrar los elementos de la lista enlazada.
D. Demostración:

• Cree un arreglo de al menos 5 elementos.
• Muestre el arreglo original.
• Aplique la función de inversión y muestre el arreglo resultante.

Solución

Lo primero que hice fue crear la estructura de nodo y la función para imprimir la lista ya que eran cosas que había realizado anteriormente

```
struct Nodo{
    int dato;
    Nodo *siguiente;
```

```
void imprimirLista(Nodo *lista){
    Nodo *actual = lista;
    while(actual ≠ nullptr){
        cout<<actual→dato<<" → ";
        actual = actual→siguiente;
    }
    if (actual == nullptr) {
        cout << "null" << endl;
    }
}
```

Luego, al momento de crear la función insertarOrdenado me di cuenta que necesitaba una forma de crear un nodo sin la necesidad de tener un puntero al próximo dato desde que se crea así que añadí un constructor al al struct Nodo

```
struct Nodo{
    int dato;
    Nodo *siguiente;

    Nodo(int valor, Nodo *sig = nullptr): dato(valor), siguiente(sig){}// Constructor para inicializar el nodo
};
```

para crear la funcion de insertarOrdenado cree un nodo que en un principio solo tiene un dato, luego existen casos 2 posibles:
Caso 1: el valor del nodo a insertar es menor a todos los demás o la lista está vacía, en este caso el valor debe ser insertado al inicio de la lista.

```
void insertarOrdenado(Nodo*& inicio, int valor) {
    Nodo* nuevo = new Nodo(valor);
    if (inicio == nullptr || valor ≤ inicio→dato) {
        nuevo→siguiente = inicio;
        inicio = nuevo;
    }
```

para esta función se deba pasar una lista y un valor, luego crea un nodo con el valor que se quiere insertar, luego, pone ese nodo al inicio de la lista si el primer elemento de la lista apunta a nullptr (la lista está vacía), o si el valor que se quiere insertar es menor o igual al primer dato de la lista

Caso 2: el valor debe ser insertado en cualquier otra posición distinta del inicio

```
else {
    // Buscar la posición correcta para insertar
    Nodo* actual = inicio;
    while (actual->siguiente != nullptr && valor > actual->siguiente->dato) {
        actual = actual->siguiente;
    }
    nuevo->siguiente = actual->siguiente; // Enlazar el nuevo nodo
    actual->siguiente = nuevo; // Enlazar el nodo anterior al nuevo
}
```

se crea un nodo nuevo el cual va a servir para recorrer todos los nodos mediante el ciclo while que recorre cada nodo de la lista mientras que el elemento actual no apunte a nullptr (que no sea el último elemento de la lista) y mientras que el valor que se quiere insertar sea mayor al valor del dato del nodo siguiente, luego, en el momento en el que el nodo actual sea el último en la lista o el valor a insertar deje de ser mayor al valor del dato siguiente, el ciclo se acaba y para insertar el valor en la lista, el nodo va a apuntar hacia el mismo dato que apuntaba el nodo actual, y el nodo actual ahora a va a apuntar al valor nuevo que se va a ingresar.
Viéndose así:

```
39 int main() {
40     Nodo* inicio = nullptr; // Inicializar la lista vacía
41     imprimirLista(inicio); // Imprimir lista vacía
42     insertarOrdenado(inicio, 5);
43     imprimirLista(inicio);
44     insertarOrdenado(inicio, 3);
45     imprimirLista(inicio);
46     insertarOrdenado(inicio, 8);
47     imprimirLista(inicio);
48     insertarOrdenado(inicio, 1);
49     imprimirLista(inicio);
50     insertarOrdenado(inicio, 7);
51     imprimirLista(inicio);
52
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  COMMENTS

PS C:\Users\diego\Estruc.Datos y Algoritmos\Estructura-de-Datos-y-Al
ructura-de-Datos-y-Algoritmos\Parcial1\output'
PS C:\Users\diego\Estruc.Datos y Algoritmos\Estructura-de-Datos-y-Al
null
5 -> null
3 -> 5 -> null
3 -> 5 -> 8 -> null
1 -> 3 -> 5 -> 8 -> null
1 -> 3 -> 5 -> 7 -> 8 -> null
```

Por último, debido a que las listas enlazadas se almacenan en el heap, es necesario borrar la lista manualmente después de usarla.

```
Nodo* actual = inicio;
while (actual != nullptr) {
    Nodo* temp = actual;
    actual = actual->siguiente;
    delete temp; // Liberar memoria del nodo actual
}
return 0;
```

Este algoritmo recorre cada nodo de la lista borrándolo.