

UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA, SOLOLA

Ingeniería En Sistemas

Tercer Semestre

Curso: Programación 1

Catedrático: Nefalí García

Estudiante: Pascual Evelio Yaxon Tziquin

Selvin Geovany Tzunún Balán

No. Carné: 2290-24-26037

2290-24-44



Tema

Tarea Semana 8

LISTAS DOBLEMENTE ENLAZADAS

- Código comentado:

```
Tarea semana 8.cpp* -> X
Tarea semana 8 (Ámbito global)
1 // Tarea semana 8.cpp : Este archivo contiene la función "main". La ejecución del programa comienza y termina ahí.
2 //
3
4 #include <iostream> // Incluye la biblioteca para entrada y salida de datos
5
6 using namespace std; // Permite usar los elementos del espacio de nombres estándar sin especificarlo cada vez
7
8 // Definición de la estructura NodoDoble para la lista doblemente enlazada
9 struct NodoDoble {
10     int dato; // Dato almacenado en el nodo
11     NodoDoble* siguiente; // Puntero al siguiente nodo
12     NodoDoble* anterior; // Puntero al nodo anterior
13 };
14
15 // Definición de la clase ListaDoble
16 class ListaDoble {
17 private:
18     NodoDoble* cabeza; // Puntero al primer nodo de la lista
19 public:
20     // Constructor que inicializa la cabeza de la lista en nullptr (lista vacía)
21     ListaDoble() : cabeza(nullptr) {}
22
23     // Método para insertar un nuevo nodo al inicio de la lista
24     void insertarInicio(int valor) {
25         NodoDoble* nuevo = new NodoDoble; // Crea un nuevo nodo en memoria dinámica
26         nuevo->dato = valor; // Asigna el valor al nodo
27         nuevo->siguiente = cabeza; // El nuevo nodo apunta al antiguo primer nodo
28         nuevo->anterior = nullptr; // Como es el primer nodo, no tiene anterior
29
30         // Si la lista no está vacía, se actualiza el puntero anterior del nodo cabeza
31         if (cabeza != nullptr) {
32             cabeza->anterior = nuevo;
33         }
34         cabeza = nuevo; // Se actualiza la cabeza de la lista al nuevo nodo
35     }
36
37     // Método para imprimir la lista desde el inicio hasta el final
38     void imprimirAdelante() {
39         NodoDoble* actual = cabeza; // Se inicializa un puntero en la cabeza
40         cout << "Lista (adelante): ";
41         while (actual != nullptr) { // Se recorre la lista mientras el nodo actual no sea nulo
42             cout << actual->dato << " <-> "; // Imprime el valor del nodo
43             actual = actual->siguiente; // Avanza al siguiente nodo
44         }
45         cout << "NULL" << endl; // Indica el final de la lista
46     }
47
48     // Método para imprimir la lista desde el final hasta el inicio
49     void imprimirAtras() {
50         NodoDoble* actual = cabeza; // Se inicializa el puntero en la cabeza
51         if (actual == nullptr) return; // Si la lista está vacía, se sale del método
52     }
53 }
```

```
45         cout << "NULL" << endl; // Indica el final de la lista
46     }
47
48     // Método para imprimir la lista desde el final hasta el inicio
49     void imprimirAtras() {
50         NodoDoble* actual = cabeza; // Se inicializa el puntero en la cabeza
51         if (actual == nullptr) return; // Si la lista está vacía, se sale del método
52
53         // Se mueve el puntero hasta el último nodo
54         while (actual->siguiente != nullptr) {
55             actual = actual->siguiente;
56         }
57
58         cout << "Lista (atras): ";
59         while (actual != nullptr) { // Se recorre la lista hacia atrás
60             cout << actual->dato << " <-> "; // Imprime el valor del nodo
61             actual = actual->anterior; // Retrocede al nodo anterior
62         }
63         cout << "NULL" << endl; // Indica el inicio de la lista
64     }
65
66     // Método para eliminar el primer nodo que contenga el valor dado
67     void eliminar(int valor) {
68         NodoDoble* actual = cabeza; // Se inicializa un puntero en la cabeza
69
70         // Se busca el nodo que contiene el valor
71         while (actual != nullptr && actual->dato != valor) {
72             actual = actual->siguiente;
73         }
74
75         // Si no se encontró el valor, se informa al usuario
76         if (actual == nullptr) {
77             cout << "Valor no encontrado." << endl;
78             return;
79         }
80
81         // Se ajusta el puntero del nodo anterior al nodo siguiente
82         if (actual->anterior != nullptr) {
83             actual->anterior->siguiente = actual->siguiente;
84         }
85         else {
86             // Si se elimina la cabeza, se actualiza la cabeza de la lista
87             cabeza = actual->siguiente;
88         }
89
90         // Se ajusta el puntero del nodo siguiente al nodo anterior
91         if (actual->siguiente != nullptr) {
92             actual->siguiente->anterior = actual->anterior;
93         }
94
95         delete actual; // Se libera la memoria del nodo eliminado
96         cout << "Valor eliminado: " << valor << endl;
```

```
Tarea semana 8.cpp*  X
Tarea semana 8  ListaDoble

92         actual->siguiente->anterior = actual->anterior;
93     }
94
95     delete actual; // Se libera la memoria del nodo eliminado
96     cout << "Valor eliminado: " << valor << endl;
97 }
98
99 // Destructor para liberar la memoria de todos los nodos de la lista
100 ~ListaDoble() {
101     NodoDoble* actual = cabeza; // Se inicializa el puntero en la cabeza
102     while (actual != nullptr) { // Se recorre la lista
103         NodoDoble* siguiente = actual->siguiente; // Se guarda el siguiente nodo antes de eliminar
104         delete actual; // Se elimina el nodo actual
105         actual = siguiente; // Se mueve al siguiente nodo
106     }
107 }
108 };
109
110 // Función principal del programa
111 int main() {
112     ListaDoble lista; // Se crea una instancia de la lista doblemente enlazada
113
114     // Se insertan elementos en la lista
115     lista.insertarInicio(200);
116     lista.insertarInicio(300);
117     lista.insertarInicio(400);
118
119     // Se imprime la lista en ambas direcciones
120     cout << "Lista doblemente encadenada:" << endl;
121     lista.imprimirAdelante();
122     lista.imprimirAtras();
123
124     // Se elimina un nodo con el valor 30
125     lista.eliminar(30);
126     cout << "Despues de eliminar 30:" << endl;
127
128     // Se vuelve a imprimir la lista
129     lista.imprimirAdelante();
130     lista.imprimirAtras();
131
132     return 0; // Fin del programa
133 }
134
```

- Compilación:

```
Consola de depuración de Microsoft Visual Studio

Lista doblemente encadenada:
Lista (adelante): 400 <-> 300 <-> 200 <-> NULL
Lista (atras): 200 <-> 300 <-> 400 <-> NULL
Valor no encontrado.
Despues de eliminar 30:
Lista (adelante): 400 <-> 300 <-> 200 <-> NULL
Lista (atras): 200 <-> 300 <-> 400 <-> NULL

D:\Desktop\Programacion trabajo 2\Tarea semana 8\x64\Debug\Tarea semana 8.exe (proceso 34628) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

LISTA SIMPLE

- Código comentado:

```
Lista Simple (Ámbito global) main()

1 #include <iostream> // Incluye la biblioteca estándar para entrada/salida
2 using namespace std; // Permite usar los nombres de la biblioteca estándar directamente
3
4 // Estructura que define un nodo de la lista enlazada
5 struct Nodo {
6     int dato; // Almacena el valor del nodo
7     Nodo* siguiente; // Puntero al siguiente nodo en la lista
8 };
9
10 // Clase que implementa una lista enlazada simple
11 class ListaSimple {
12 private:
13     Nodo* cabeza; // Puntero al primer nodo de la lista
14
15 public:
16     // Constructor: inicializa la lista vacía (cabeza apunta a nullptr)
17     ListaSimple() : cabeza(nullptr) {}
18
19     // Método para insertar un nuevo nodo al inicio de la lista
20     void insertarInicio(int valor) {
21         Nodo* nuevo = new Nodo; // Crea un nuevo nodo
22         nuevo->dato = valor; // Asigna el valor al nuevo nodo
23         nuevo->siguiente = cabeza; // El nuevo nodo apunta al antiguo primer nodo
24         cabeza = nuevo; // La cabeza ahora apunta al nuevo nodo
25     }
26
27     // Método para imprimir todos los elementos de la lista
28     void imprimir() {
29         Nodo* actual = cabeza; // Comienza desde el primer nodo
30         while (actual != nullptr) { // Mientras no llegue al final
31             cout << actual->dato; // Imprime el valor del nodo actual
32             if (actual->siguiente != nullptr) {
33                 cout << " -> "; // Imprime flecha si hay siguiente nodo
34             }
35             actual = actual->siguiente; // Avanza al siguiente nodo
36         }
37         cout << " -> NULL" << endl; // Indica el final de la lista
38     }
39 }
```

```
Lista Simple (Ambito global) main()
40 // Método para eliminar el primer nodo que contenga un valor específico
41 void eliminar(int valor) {
42     Nodo* actual = cabeza; // Comienza desde el primer nodo
43     Nodo* anterior = nullptr; // Mantiene referencia al nodo anterior
44
45     // Busca el nodo con el valor a eliminar
46     while (actual != nullptr && actual->dato != valor) {
47         anterior = actual;
48         actual = actual->siguiente;
49     }
50
51     // Si no encontró el valor
52     if (actual == nullptr) {
53         cout << "Valor no encontrado." << endl;
54         return;
55     }
56
57     // Reorganiza los punteros para eliminar el nodo
58     if (anterior == nullptr) {
59         // Si es el primer nodo, actualiza la cabeza
60         cabeza = actual->siguiente;
61     }
62     else {
63         // Si está en medio, el anterior salta al siguiente
64         anterior->siguiente = actual->siguiente;
65     }
66
67     delete actual; // Libera la memoria del nodo eliminado
68     cout << "Valor eliminado: " << valor << endl;
69 }
70
71 // Destructor: libera toda la memoria asignada a los nodos
72 ~ListaSimple() {
73     Nodo* actual = cabeza; // Comienza desde el primer nodo
74     while (actual != nullptr) { // Mientras no llegue al final
75         Nodo* siguiente = actual->siguiente; // Guarda referencia al siguiente
76         delete actual; // Libera el nodo actual
77         actual = siguiente; // Avanza al siguiente nodo
78     }
79 }
80
81 };
```

```
Lista Simple (Ambito global) main()
82 // Función principal para probar la lista
83 int main() {
84     ListaSimple lista; // Crea una lista vacía
85
86     // Inserta tres valores al inicio
87     lista.insertarInicio(5);
88     lista.insertarInicio(10);
89     lista.insertarInicio(15);
90
91     // Imprime la lista después de las inserciones
92     cout << "Lista después de inserciones:" << endl;
93     lista.imprimir();
94
95     // Elimina un valor específico
96     lista.eliminar(10);
97     cout << "Lista después de eliminar 10:" << endl;
98     lista.imprimir();
99
100     return 0; // Fin del programa
101 }
```

• Compilación:

```
Consola de depuración de Mi x + -
Lista despues de inserciones:
15 -> 10 -> 5 -> NULL
Valor eliminado: 10
Lista despues de eliminar 10:
15 -> 5 -> NULL

C:\Users\pasev\source\repos\Códigos De Clase\x64\Debug\Lista Simple.exe (proceso 18256) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

LINK DEL REPOSITORIO PARA ESTA TAREA:

<https://github.com/E-lev10/Tarea-Semana-8-Progra.git>