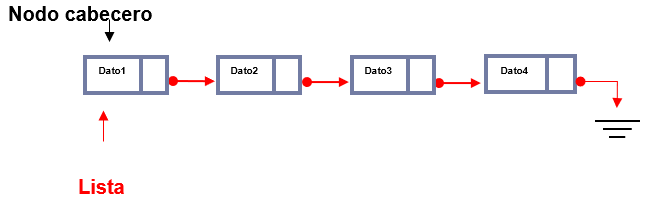
**Tipos de listas**

1. **Listas Simples**

Una lista simple “es una estructura de datos lineal, dinámica, formada por una colección de elementos llamados nodos. Cada nodo está formado por dos partes: la primera de ellas se utiliza para almacenar la información (razón de ser de la estructura de datos), y la segunda se usa para guardar la dirección del siguiente nodo” (Silvia Guardati, 2017,).

Cada nodo (elemento) contiene un enlace que conecta el nodo con el siguiente o el siguiente nodo. Esta lista es eficaz para el recorrido directo ("hacia adelante").



**Imagen 1** Lista Simple

Ejemplo:

***main.cpp***

#include <iostream>  
#include "ListaSimple.h"  
#include "Validacion.h"  
**int** **main**(){  
 Lista lista;  
 **do** {  
 system("cls");  
 std::cout << "**\n\tLISTA SIMPLE\n\n**" <<  
 "1. Insertar al inicio**\n**" <<  
 "2. Eliminar**\n**" <<  
 "3. Mostrar**\n**" <<  
 "4. Salir**\n\n**";  
 **switch** (ingresar\_enteros("Ingresar la opcion: ")) {  
 **case** **1**:  
 lista.insertarAlInicio(ingresar\_enteros("**\n\n**Ingresar un numero: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **2**:  
 lista.eliminar(ingresar\_enteros("**\n\n**Ingresar el numero a eliminar: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **3**:  
 lista.imprimir();  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **4**:  
 **return** **0**;  
 **default:**  
 **break**;  
 }  
 } **while** (true);  
}

***ListaSimple.h***

#pragma once  
#include <iostream>  
  
// Definición de la clase Nodo  
**class** **Nodo** {

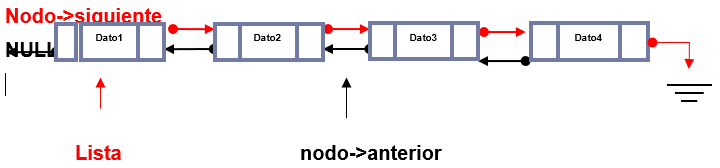
**public:**  
 **int** dato;  
 Nodo\* siguiente;  
  
 // Constructor  
 Nodo(**int** valor) : dato(valor), siguiente(nullptr) {}  
};  
  
// Clase Lista que contiene operaciones básicas  
**class** **Lista** {  
**private:**  
 Nodo\* cabeza;  
  
**public:**  
 // Constructor  
 Lista() {  
 cabeza = nullptr;  
 }  
  
 // Método para insertar un elemento al inicio de la lista  
 **void** insertarAlInicio(**int** valor) {  
 Nodo\* nuevoNodo = **new** Nodo(valor);  
 nuevoNodo->siguiente = cabeza;  
 cabeza = nuevoNodo;  
 }  
  
 // Método para imprimir la lista  
 **void** imprimir() {  
 Nodo\* temp = cabeza;  
 **while** (temp != nullptr) {  
 std::cout << temp->dato << " -> ";  
 temp = temp->siguiente;  
 }  
 std::cout << "NULL" << std::endl;  
 }  
  
 // Método para eliminar un elemento de la lista dada una clave (valor)  
 **void** eliminar(**int** clave) {  
 Nodo\* temp = cabeza;  
 Nodo\* prev = nullptr;  
  
 // Buscar el nodo con la clave dada  
 **while** (temp != nullptr && temp->dato != clave) {  
 prev = temp;  
 temp = temp->siguiente;  
 }  
  
 // Si la clave no se encuentra en la lista, no hay nada que eliminar  
 **if** (temp == nullptr) {  
 std::cout << "**\n**El elemento " << clave << " no esta en la lista." << std::endl;  
 **return**;  
 }  
  
 // Eliminar el nodo encontrado  
 **if** (prev != nullptr)  
 prev->siguiente = temp->siguiente;  
 **else**  
 cabeza = temp->siguiente;  
  
 **delete** temp;  
 }  
  
 // Destructor para liberar la memoria asignada a los nodos de la lista  
 ~Lista() {  
 Nodo\* temp = cabeza;  
 **while** (temp != nullptr) {  
 Nodo\* siguiente = temp->siguiente;  
 **delete** temp;  
 temp = siguiente;  
 }  
 cabeza = nullptr;  
 }  
};

***Validacion.h***

#pragma once  
#include <conio.h>  
#include <iostream>  
  
**int** **borrar**(**char**\* datos, **int**& i) {  
 **if** (i > **0**) {  
 printf("**\b** **\b**");  
 i--;  
 datos[i] = '\0';  
 **return** i;  
 }  
 **return** **0**;  
}  
  
**int** **ingresar\_enteros**(std::string msj) {  
 **char**\* datos = **new** **char**[**10**];  
 **char** c;  
 **int** i = **0**;  
  
 std::cout << msj;  
 **while** ((c = \_getch()) != **13** && i < **9**) {  
 **if** ((c >= '0' && c <= '9') || c == **8**) {  
 **if** (c == **8**) {  
 i = borrar(datos, i);  
 }  
 **else** {  
 printf("%c", c);  
 datos[i++] = c;  
 }  
  
 }  
 }  
 datos[i] = '\0';  
 **return** atoi(datos);  
}

1. **Listas Dobles Enlazadas.**

Una lista enlazada es una lista mundial con dos enlaces, un enlace al nodo siguiente y otro enlace al nodo anterior, además; se recorrido lo podemos hacer en ambos sentidos a partir de cualquier nodo, por esta razón es posible alcanzar cualquier nodo de la lista, hasta llegar a uno de sus extremos. Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directo (“adelante”) como en recorrido inverso (“atrás”). (Silvia Guardati, 2017)



**Imagen 2** Lista Doble

Ejemplo:

***main.cpp***

#include <iostream>  
#include "ListaDoble.h"  
#include "Validacion.h"  
  
**int** **main**(){  
 ListaDoble lista;  
 **do** {  
 system("cls");  
 std::cout << "**\n\tLISTA DOBLE ENLAZADA\n\n**" <<  
 "1. Insertar al inicio**\n**" <<  
 "2. Eliminar**\n**" <<  
 "3. Mostrar cabeza**\n**" <<  
 "4. Mostrar cola**\n**" <<  
 "5. Salir**\n\n**";  
 **switch** (ingresar\_enteros("Ingresar la opcion: ")) {  
 **case** **1**:  
 lista.insertarAlInicio(ingresar\_enteros("**\n\n**Ingresar un numero: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **2**:  
 lista.eliminar(ingresar\_enteros("**\n\n**Ingresar el numero a eliminar: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **3**:  
 lista.imprimirAdelante();  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **4**:  
 lista.imprimirAtras();  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **5**:  
 **return** **0**;  
 **default:**  
 **break**;  
 }  
 } **while** (true);  
}

***ListaDoble.cpp***

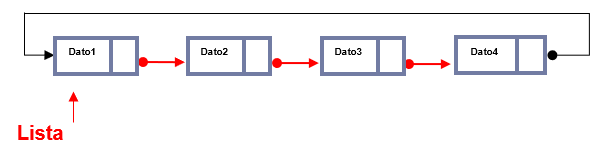
#pragma once  
#include <iostream>  
  
// Definición de la clase NodoDoble  
**class** **NodoDoble** {  
**public:**  
 **int** dato;  
 NodoDoble\* siguiente;  
 NodoDoble\* anterior;  
  
 // Constructor  
 NodoDoble(**int** valor) : dato(valor), siguiente(nullptr), anterior(nullptr) {}  
};  
  
// Clase ListaDoble que contiene operaciones básicas  
**class** **ListaDoble** {  
**private:**  
 NodoDoble\* cabeza;  
  
**public:**  
 // Constructor  
 ListaDoble() {  
 cabeza = nullptr;  
 }  
  
 // Método para insertar un elemento al inicio de la lista  
 **void** insertarAlInicio(**int** valor) {  
 NodoDoble\* nuevoNodo = **new** NodoDoble(valor);  
 **if** (cabeza == nullptr) {  
 cabeza = nuevoNodo;  
 }  
 **else** {  
 nuevoNodo->siguiente = cabeza;  
 cabeza->anterior = nuevoNodo;  
 cabeza = nuevoNodo;  
 }  
 }  
  
 // Método para imprimir la lista de principio a fin  
 **void** imprimirAdelante() {  
 NodoDoble\* temp = cabeza;  
 **while** (temp != nullptr) {  
 std::cout << temp->dato << " <-> ";  
 temp = temp->siguiente;  
 }  
 std::cout << "NULL" << std::endl;  
 }  
  
 // Método para imprimir la lista de fin a principio  
 **void** imprimirAtras() {  
 NodoDoble\* temp = cabeza;  
 **while** (temp->siguiente != nullptr) {  
 temp = temp->siguiente;  
 }  
 **while** (temp != nullptr) {  
 std::cout << temp->dato << " <-> ";  
 temp = temp->anterior;  
 }  
 std::cout << "NULL" << std::endl;  
 }  
  
 **void** eliminar(**int** valor) {  
 NodoDoble\* temp = cabeza;  
 NodoDoble\* prev = nullptr;  
  
 // Buscar el nodo con el valor dado  
 **while** (temp != nullptr && temp->dato != valor) {  
 prev = temp;  
 temp = temp->siguiente;  
 }  
  
 // Si el valor no se encuentra en la lista, no hay nada que eliminar  
 **if** (temp == nullptr) {  
 std::cout << "El elemento " << valor << " no esta en la lista." << std::endl;  
 **return**;  
 }  
  
 // Eliminar el nodo encontrado  
 **if** (prev != nullptr)  
 prev->siguiente = temp->siguiente;  
 **else**  
 cabeza = temp->siguiente;  
  
 **if** (temp->siguiente != nullptr)  
 temp->siguiente->anterior = prev;  
  
 **delete** temp;  
 }  
  
 // Destructor para liberar la memoria asignada a los nodos de la lista  
 ~ListaDoble() {  
 NodoDoble\* temp = cabeza;  
 **while** (temp != nullptr) {  
 NodoDoble\* siguiente = temp->siguiente;  
 **delete** temp;  
 temp = siguiente;  
 }  
 cabeza = nullptr;  
 }  
};

***Validacion.h***

#pragma once  
#include <conio.h>  
#include <iostream>  
  
**int** **borrar**(**char**\* datos, **int**& i) {  
 **if** (i > **0**) {  
 printf("**\b** **\b**");  
 i--;  
 datos[i] = '\0';  
 **return** i;  
 }  
 **return** **0**;  
}  
  
**int** **ingresar\_enteros**(std::string msj) {  
 **char**\* datos = **new** **char**[**10**];  
 **char** c;  
 **int** i = **0**;  
  
 std::cout << msj;  
 **while** ((c = \_getch()) != **13** && i < **9**) {  
 **if** ((c >= '0' && c <= '9') || c == **8**) {  
 **if** (c == **8**) {  
 i = borrar(datos, i);  
 }  
 **else** {  
 printf("%c", c);  
 datos[i++] = c;  
 }  
  
 }  
 }  
 datos[i] = '\0';  
 **return** atoi(datos);  
}

1. **Listas Simples Circulares**

Una lista circular simple es una lista en la cual el nodo que sigue al último es el primero. Es decir, el último nodo tiene como sucesor al primero de la lista, logrando con ello tener acceso nuevamente a todos los miembros de la lista. Esta característica permite que desde cualquier nodo de esta estructura de datos se tenga acceso a cualquiera de los otros nodos de esta (Silvia Guardati, 2017).



**Imagen 3** Lista simple circular

Ejemplo:

***Main.cpp***

#include <iostream>  
#include "ListaCircular.h"  
#include "Validacion.h"  
  
**int** **main**(){  
 ListaCircular lista;  
 **do** {  
 system("cls");  
 std::cout << "**\n\t**LISTA SIMPLE CIRCULAR**\n\n**" <<  
 "1. Insertar al inicio**\n**" <<  
 "2. Eliminar**\n**" <<  
 "3. Mostrar**\n**" <<  
 "4. Salir**\n\n**";  
 **switch** (ingresar\_enteros("Ingresar la opcion: ")) {  
 **case** **1**:  
 lista.insertarAlInicio(ingresar\_enteros("**\n\n**Ingresar un numero: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **2**:  
 lista.eliminar(ingresar\_enteros("**\n\n**Ingresar el numero a eliminar: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **3**:  
 lista.imprimir();  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **4**:  
 **return** **0**;  
 **default:**  
 **break**;  
 }  
 } **while** (true);  
}

***ListaCircular.h***

#pragma once  
#include <iostream>  
  
// Definición de la clase Nodo  
**class** **Nodo** {

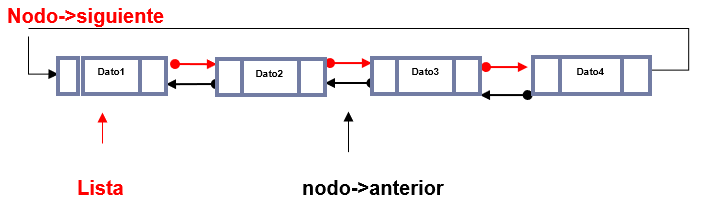
**public:**  
 **int** dato;  
 Nodo\* siguiente;  
  
 // Constructor  
 Nodo(**int** valor) : dato(valor), siguiente(nullptr) {}  
};  
  
// Clase ListaCircular que contiene operaciones básicas  
**class** **ListaCircular** {  
**private:**  
 Nodo\* cabeza;  
  
**public:**  
 // Constructor  
 ListaCircular() : cabeza(nullptr) {}  
  
 // Método para verificar si la lista está vacía  
 **bool** estaVacia() {  
 **return** cabeza == nullptr;  
 }  
  
 // Método para insertar un elemento al inicio de la lista  
 **void** insertarAlInicio(**int** valor) {  
 Nodo\* nuevoNodo = **new** Nodo(valor);  
 **if** (estaVacia()) {  
 nuevoNodo->siguiente = nuevoNodo;  
 cabeza = nuevoNodo;  
 }  
 **else** {  
 nuevoNodo->siguiente = cabeza->siguiente;  
 cabeza->siguiente = nuevoNodo;  
 }  
 }  
  
 // Método para imprimir la lista  
 **void** imprimir() {  
 **if** (estaVacia()) {  
 std::cout << "La lista está vacía." << std::endl;  
 **return**;  
 }  
  
 Nodo\* temp = cabeza->siguiente;  
 std::cout << "Lista: ";  
 **do** {  
 std::cout << temp->dato << " -> ";  
 temp = temp->siguiente;  
 } **while** (temp != cabeza->siguiente);  
 std::cout << "(Cabeza)" << std::endl;  
 }  
  
 // Método para eliminar un elemento de la lista dada una clave (valor)  
 **void** eliminar(**int** clave) {  
 **if** (estaVacia()) {  
 std::cout << "La lista está vacía." << std::endl;  
 **return**;  
 }  
  
 Nodo\* temp = cabeza;  
 Nodo\* prev = nullptr;  
  
 // Buscar el nodo con la clave dada  
 **do** {  
 prev = temp;  
 temp = temp->siguiente;  
 **if** (temp->dato == clave) {  
 prev->siguiente = temp->siguiente;  
 **if** (temp == cabeza)  
 cabeza = prev;  
 **delete** temp;  
 std::cout << "Elemento " << clave << " eliminado." << std::endl;  
 **return**;  
 }  
 } **while** (temp != cabeza->siguiente);  
  
 std::cout << "Elemento " << clave << " no encontrado en la lista." << std::endl;  
 }  
  
 // Destructor para liberar la memoria asignada a los nodos de la lista  
 ~ListaCircular() {  
 **if** (!estaVacia()) {  
 Nodo\* temp = cabeza->siguiente;  
 **while** (temp != cabeza) {  
 Nodo\* siguiente = temp->siguiente;  
 **delete** temp;  
 temp = siguiente;  
 }  
 **delete** cabeza;  
 cabeza = nullptr;  
 }  
 }  
};

***Validacion.h***

#pragma once  
#include <conio.h>  
#include <iostream>  
  
**int** **borrar**(**char**\* datos, **int**& i) {  
 **if** (i > **0**) {  
 printf("**\b** **\b**");  
 i--;  
 datos[i] = '\0';  
 **return** i;  
 }  
 **return** **0**;  
}  
  
**int** **ingresar\_enteros**(std::string msj) {  
 **char**\* datos = **new** **char**[**10**];  
 **char** c;  
 **int** i = **0**;  
  
 std::cout << msj;  
 **while** ((c = \_getch()) != **13** && i < **9**) {  
 **if** ((c >= '0' && c <= '9') || c == **8**) {  
 **if** (c == **8**) {  
 i = borrar(datos, i);  
 }  
 **else** {  
 printf("%c", c);  
 datos[i++] = c;  
 }  
  
 }  
 }  
 datos[i] = '\0';  
 **return** atoi(datos);  
}

1. **Listas Dobles Circulares.**

Las listas doblemente ligadas son otra variante de las estructuras vistas en las secciones previas. En las listas simplemente ligadas cada nodo conoce solamente la dirección de su nodo sucesor. De ahí la importancia de no perder el puntero al primer nodo de esta. Por su parte, en las listas doblemente ligadas, cada nodo conoce la dirección de su predecesor y de su sucesor. La excepción es el primer nodo de la lista que no cuenta con predecesor, y el último que no tiene sucesor. Debido a esta característica, se puede visitar a todos los componentes de la lista a partir de cualquiera de ellos (Silvia Guardati, 2017).



**Imagen 4** Lista doble circular

Ejemplo:

***Main.h***

#include <iostream>  
#include "ListaDobleCircular.h"  
#include "Validacion.h"  
  
**int** **main**(){  
 ListaDobleCircular lista;  
 **do** {  
 system("cls");  
 std::cout << "**\n\t**LISTA DOBLE CIRCULAR**\n\n**" <<  
 "1. Insertar al inicio**\n**" <<  
 "2. Eliminar**\n**" <<  
 "3. Mostrar**\n**" <<  
 "4. Salir**\n\n**";  
 **switch** (ingresar\_enteros("Ingresar la opcion: ")) {  
 **case** **1**:  
 lista.insertarAlInicio(ingresar\_enteros("**\n\n**Ingresar un numero: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **2**:  
 lista.eliminar(ingresar\_enteros("**\n\n**Ingresar el numero a eliminar: "));  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **3**:  
 lista.imprimir();  
 std::cout << std::endl;  
 system("pause");  
 **break**;  
 **case** **4**:  
 **return** **0**;  
 **default:**  
 **break**;  
 }  
 } **while** (true);  
}

***ListaDobleCircular.h***

#pragma once  
#include <iostream>  
  
// Definición de la clase Nodo  
**class** **Nodo** {  
**public:**

**int** dato;  
 Nodo\* siguiente;  
 Nodo\* anterior;  
  
 // Constructor  
 Nodo(**int** valor) : dato(valor), siguiente(nullptr), anterior(nullptr) {}  
};  
  
// Clase ListaDobleCircular que contiene operaciones básicas  
**class** **ListaDobleCircular** {  
**private:**  
 Nodo\* cabeza;  
  
**public:**  
 // Constructor  
 ListaDobleCircular() : cabeza(nullptr) {}  
  
 // Método para verificar si la lista está vacía  
 **bool** estaVacia() {  
 **return** cabeza == nullptr;  
 }  
  
 // Método para insertar un elemento al inicio de la lista  
 **void** insertarAlInicio(**int** valor) {  
 Nodo\* nuevoNodo = **new** Nodo(valor);  
 **if** (estaVacia()) {  
 nuevoNodo->siguiente = nuevoNodo;  
 nuevoNodo->anterior = nuevoNodo;  
 cabeza = nuevoNodo;  
 }  
 **else** {  
 nuevoNodo->siguiente = cabeza;  
 nuevoNodo->anterior = cabeza->anterior;  
 cabeza->anterior->siguiente = nuevoNodo;  
 cabeza->anterior = nuevoNodo;  
 cabeza = nuevoNodo;  
 }  
 }  
  
 // Método para imprimir la lista  
 **void** imprimir() {  
 **if** (estaVacia()) {  
 std::cout << "La lista está vacía." << std::endl;  
 **return**;  
 }  
  
 Nodo\* temp = cabeza;  
 std::cout << "Lista hacia adelante: ";  
 **do** {  
 std::cout << temp->dato << " <-> ";  
 temp = temp->siguiente;  
 } **while** (temp != cabeza);  
 std::cout << "(Cabeza)" << std::endl;  
  
 temp = cabeza->anterior;  
 std::cout << "Lista hacia atrás: ";  
 **do** {  
 std::cout << temp->dato << " <-> ";  
 temp = temp->anterior;  
 } **while** (temp != cabeza->anterior);  
 std::cout << "(Cabeza)" << std::endl;  
 }  
  
 // Método para eliminar un elemento de la lista dada una clave (valor)  
 **void** eliminar(**int** clave) {  
 **if** (estaVacia()) {  
 std::cout << "La lista está vacía." << std::endl;  
 **return**;  
 }  
  
 Nodo\* temp = cabeza;  
 **do** {  
 **if** (temp->dato == clave) {  
 **if** (temp == cabeza) {  
 cabeza = temp->siguiente;  
 }  
 temp->anterior->siguiente = temp->siguiente;  
 temp->siguiente->anterior = temp->anterior;  
 **delete** temp;  
 std::cout << "Elemento " << clave << " eliminado." << std::endl;  
 **return**;  
 }  
 temp = temp->siguiente;  
 } **while** (temp != cabeza);  
 std::cout << "Elemento " << clave << " no encontrado en la lista." << std::endl;  
 }  
  
 // Destructor para liberar la memoria asignada a los nodos de la lista  
 ~ListaDobleCircular() {  
 **if** (!estaVacia()) {  
 Nodo\* temp = cabeza->siguiente;  
 **while** (temp != cabeza) {  
 Nodo\* siguiente = temp->siguiente;  
 **delete** temp;  
 temp = siguiente;  
 }  
 **delete** cabeza;  
 cabeza = nullptr;  
 }  
 }  
};

***Validacion.h***

#pragma once  
#include <conio.h>  
#include <iostream>  
  
**int** **borrar**(**char**\* datos, **int**& i) {  
 **if** (i > **0**) {  
 printf("**\b** **\b**");  
 i--;  
 datos[i] = '\0';  
 **return** i;  
 }  
 **return** **0**;  
}  
  
**int** **ingresar\_enteros**(std::string msj) {  
 **char**\* datos = **new** **char**[**10**];  
 **char** c;  
 **int** i = **0**;  
  
 std::cout << msj;  
 **while** ((c = \_getch()) != **13** && i < **9**) {  
 **if** ((c >= '0' && c <= '9') || c == **8**) {  
 **if** (c == **8**) {  
 i = borrar(datos, i);  
 }  
 **else** {  
 printf("%c", c);  
 datos[i++] = c;  
 }  
  
 }  
 }  
 datos[i] = '\0';  
 **return** atoi(datos);  
}

1. **Referencias**

[1] Guardati, S. G. (2017). Algoritmos con C+ 1ra. Castillo, Bernardino Gutiérrez Hernández Supervisor de producción: Rodrigo Romero Villalobo.