

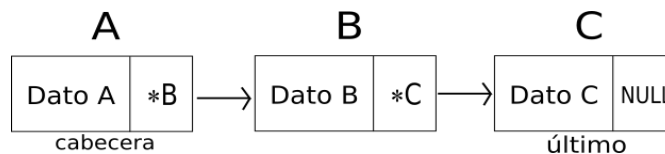
Instructor: Ernesto Enrique García Ramos

Contacto: egarcia97.r@gmail.com

gr15i04001@usonsonate.edu.sv

## Guía 2: Listas simplemente enlazadas

Las listas enlazadas son estructuras de datos semejantes a los arreglos salvo que el acceso a un elemento no se hace mediante un índice sino mediante nodos que hacen uso de punteros para sus direccionamientos.

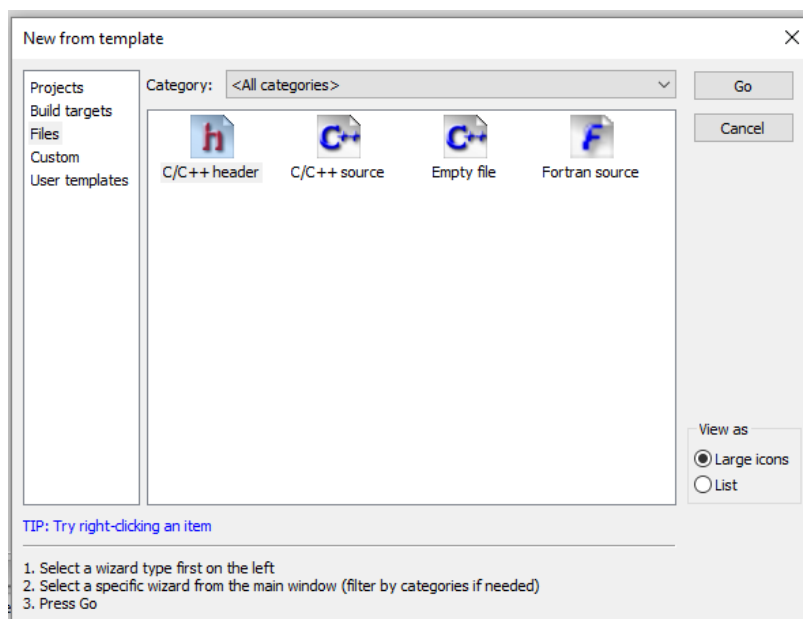


### Objetivos:

- Facilitar a los estudiantes el aprendizaje de listas enlazadas.
- Implementar listas enlazadas en el desarrollo de aplicaciones prácticas.

### Conceptos que debes conocer:

- **Nodo:** es un punto de intersección, conexión o unión de varios elementos.
- **Friend Class:** En C++ permite acceder a los miembros privados y protegidos de la clase en la cual se ha hecho una declaración de amistad.
- **Instancia:** Es la creación de un objeto.
- **Objeto:** es una unidad dentro de un programa de computadora que consta de un estado y de un comportamiento.
- **Puntero:** es un objeto del lenguaje de programación, cuyo valor se refiere a (o "apunta a").



# Programación 3

Facultad de Ingeniería y Ciencias Naturales

## Pasos para crear listas simplemente enlazadas:

1. Abrimos CodeBlocks, creamos una nueva aplicación en consola C++.
2. Agregamos una clase, para ello nos vamos a "Archivo-Nuevo-Archivo-y seleccionamos Cabecera
3. La crearemos con el nombre "Nodo".

**Definiendo la clase Nodo:** Contiene una variable que definirá el tipo con el que trabajaremos la lista, y también un puntero cuya dirección es al nodo siguiente, con un solo puntero se puede construir una lista simplemente enlazada

El código de implementación es el siguiente:

```
class Nodo
{
    private:
        int variable;
        Nodo *Siguiete;
    public:
        Nodo(int valor,Nodo *Sig=NULL) {
            this->variable = valor;
            this->Siguiete = Sig;
        }
        int getVariable(){
            return this->variable;
        }
        void setVariable(int SetVariable){
            this->variable = SetVariable;
        }
        virtual ~Nodo() {}
        friend class Lista;

};typedef Nodo *pNodo;
```

4. Creamos la clase Lista, de la misma manera que creamos la clase Nodo

## Definiendo la clase Lista:

En la lista simple los nodos se organizan de modo que cada uno apunta al siguiente, y el último a nulo. Cuando la lista se encuentra vacía y se inserta el primer elemento, este apunta a nulo, cuando se inserta un segundo, el primero apunta al segundo, y el segundo a nulo...

A diferencia de la clase nodo, la clase lista cuenta con varios métodos y funciones, desde inserción hasta búsqueda.

```
using namespace std;
#include "Nodo.h" //Incluimos la clase nodo, ya que trabajaremos
con elementos de esta

class Lista{
```

# Programación 3

Facultad de Ingeniería y Ciencias Naturales

```
private:
    pNodo primero;
    pNodo actual;    //el nodo que definara la posicion
public:

    Lista(void) {
        this->primero = actual = NULL; //al momento de crear la lista
        se inicializara el primero y ultimo nodo como NULL
    }

    virtual ~Lista() { //destructor de la clase lista
        pNodo aux;
        while ( this->primero ) //mientras exista algun nodo
        {
            aux = this->primero;
            this->primero = this->primero->Siguiente;
            delete aux;
        }
    }

    bool ListaVacia() {
        return (this->primero==NULL); //comprueba si esta vacia la
        lista , solo observando el primer nodo
    }

    void Primero() {
        this->actual = this->primero; //reiniciamos el orden para que
        actual este en la posicion del primero
    }

    void Siguiente() {
        if(this->actual->Siguiente!=NULL) {
            this->actual = this->actual->Siguiente; //nos movemos una
            posicion a la posicion siguiente
        }
    }

    void Final() {
        this->Primero();
        if(ListaVacia()!=true) {
            while(this->actual->Siguiente!=NULL) { //mientras exista
            un nodo mas que recorrer
                this->Siguiente(); //nos movemos una posicion
            }
        }
    }

    void Insertar(int valor) { //para insertar le pasamos el valor,
    que debe ser del tipo declarado en la clase nodo
        if(this->ListaVacia()) //si la lista está vacía
        {
            this->primero = new Nodo(valor); //insertamos el elemento
            en el primero
        }
        else
        {
            //cuando ya contiene valores
            this->Final(); //nos posicionamos al final de la lista
            //el ultimo nodo en la posicion siguiente apuntaba a
            null ahora apuntara al nuevo valor
            this->actual->Siguiente = new Nodo(valor);
        }
    }
}
```

```
    }

    }

    void Mostrar() {
        if(this->ListaVacia() != true) {
            this->Primero(); //Comenzamos a recorrer la lista
            desde el inicio
            while(this->actual) { //mientras todavia exista un nodo
            que recorrer
                cout << " " << this->actual->getVariable()
                << endl; //hacemos un llamado al metodo getVariable
                this->actual = this->actual->Siguiente;
            }
        }
        else {
            cout << "No hay datos que mostrar" << endl;
        }
    }

    pNodo Buscar(int valor) {
        this->Primero();
        while(this->actual != NULL) { //mientras todavia exista un nodo
        que recorrer
            if(this->actual->getVariable() == valor) {
                return this->actual;
            }
            this->actual = this->actual->Siguiente; //nos movemos al
            elemento siguiente
        }
        return NULL;
    }
}
```

## Continuando con la clase lista (Función eliminar)

La función eliminar de la clase lista devuelve un valor booleano dependiendo del estado de la eliminación. Utiliza variables auxiliares, y métodos definidos anteriormente, pudiendo así eliminar al principio, en medio y al final.

```
bool Eliminar(int valor) {
    /** Primero se comprueba que la lista no esté vacía y que el
    valor se encuentre */
    if(this->ListaVacia() == true || this->Buscar(valor) == NULL ) {
        return false;
    }
    else //si paso es porque hay datos y se encontro el elemento
    {
        this->Primero();
        pNodo aux; //se crea un nodo auxiliar que ayuda a las
        eliminaciones

        if(this->primero->getVariable() == valor ) //comprueba que
        el elemento se encuentra en la primera posicion
        {
            aux = this->primero;
            this->primero = aux->Siguiente; // primero tiene la
            posicion de auxiliar en la posicion siguiente
            delete aux; //se elimina auxiliar
        }
    }
}
```

# Programación 3

Facultad de Ingeniería y Ciencias Naturales

```
        return true;
    }
    else//si el elemento se encuentra ya sea en medio o al
final de la lista
    {
        //mientras exista un nodo en la posicion siguiente
        while(this->actual->Siguiete!=NULL)
        {
            //si nuestro dato esta en el nodo siguiente
            if(this->actual->Siguiete-
>getVariable()==valor)
            {
                //se le asigna a auxiliar el nodo que
contiene el valor buscado
                aux = this->actual->Siguiete;

                //actual en la posicion siguiente ahora
apunta un nodo mas adelante
                this->actual->Siguiete = this->actual-
>Siguiete->Siguiete;

                delete aux;
                return true;
            }
            this->Siguiete();
        }
    }
}
}
};
```

## Implementación del código

```
#include <iostream>
#include <stdlib.h>
using namespace std;

#include "Lista.h"

void insercion();
void buscar();
void eliminar();

Lista lstNumeros;
int main(){

    int opcion;
    do{
        system("cls");
        cout << "Operaciones con listas simples, utilizando enteros" <<
endl;
        cout << "1-Insertar"<< endl;
        cout << "2-Buscar"<<endl;
        cout << "3-Mostrar valores almacenados"<< endl;
        cout << "4-Eliminar"<< endl;
        cout << "5-Salir"<<endl;
        cout << "Digita una opcion: ";
```

# Programación 3

Facultad de Ingeniería y Ciencias Naturales

```
cin >> opcion;
system("cls");

switch(opcion){
    case 1:
        insercion();
        break;
    case 2:
        buscar();
        break;
    case 3:
        cout << "Numeros almacenados:"<< endl;
        lstNumeros.Mostrar();
        break;
    case 4:
        eliminar();
        break;
    case 5:
        cout << "Saliendo..." << endl;
        break;
    default:
        cout << "Opcion no valida"<<endl;
        break;
}
system("pause");

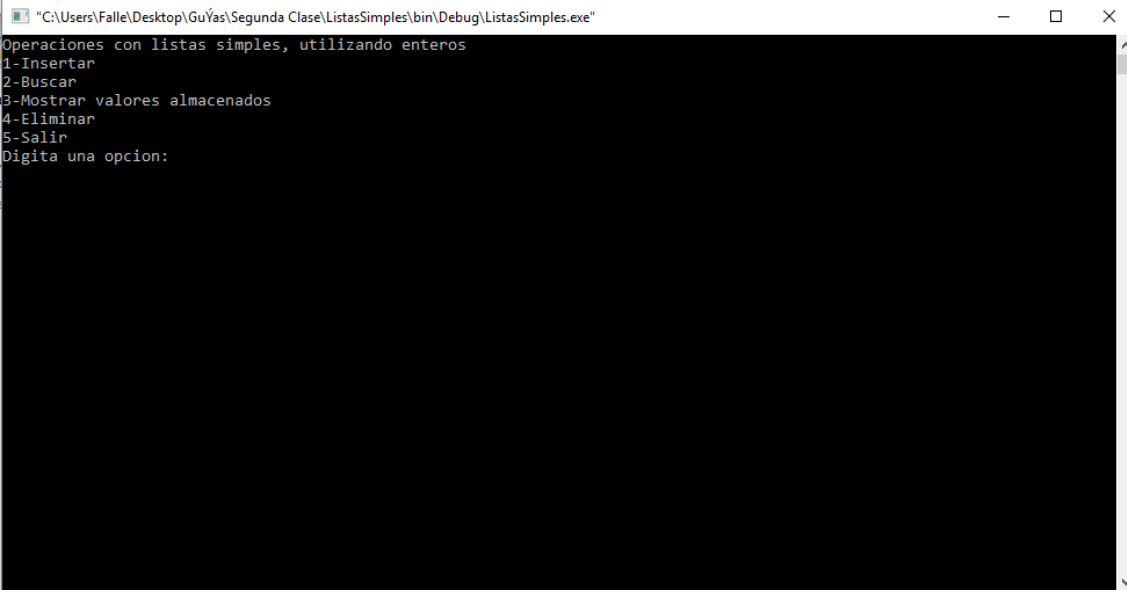
}while(opcion!=5);
return 0;
}

void insercion(){
    int variable;
    cout << "Digite el numero a insertar : ";
    cin >> variable;
    lstNumeros.Insertar(variable);
    cout << "Insertado"<<endl;
}

void eliminar(){
    int variable;
    cout << "Eliminar numero " << endl;
    cout << "Digita el numero a eliminar : ";
    cin >> variable;
    if(lstNumeros.Eliminar(variable)){
        cout << "Eliminado con exito" << endl;
    }else{
        cout << "No se pudo eliminar" << endl;
    }
}

void buscar(){
    int variable;
    cout << "Digita el numero a buscar : ";
    cin >> variable;
    if(lstNumeros.Buscar(variable)!=NULL)
    {
        cout << "Variable : " << lstNumeros.Buscar(variable)-
>getVariable() << endl;
    }else{
        cout << "No se encontro el numero" << endl;
    }
}
```

Al ejecutarlo les quedará de la siguiente manera:



```
"C:\Users\Falle\Desktop\Guías\Segunda Clase>ListasSimples\bin\Debug>ListasSimples.exe"
Operaciones con listas simples, utilizando enteros
1-Insertar
2-Buscar
3-Mostrar valores almacenados
4-Eliminar
5-Salir
Digita una opcion:
```

## Ejercicio

Dado el siguiente diagrama de clase, sustituir el tipo primitivo "int" en la clase nodo por el objeto "Paciente".

Paciente
<b>-Codigo int</b> <b>-Nombre String</b> <b>-Apellido String</b> <b>-Departamento String</b> <b>-Telefono String</b>
<b>+Paciente()</b> <b>+Paciente(int, string, string, string, string)</b> <b>+getCodigo int</b> <b>+setCodigo(int) void</b> <b>+getNombre String</b> <b>+setNombre void</b> <b>+getApellido String</b> <b>+setApellido void</b> <b>+getDepartamento String</b> <b>+setDepartamento void</b> <b>+getTelefono String</b> <b>+setDepartamento void</b>