Laboratorio 3
Listas enlazadas simples

Objetivo:

El objetivo de este laboratorio es aplicar el concepto de listas enlazadas simples para completar la implementación vista en clase y ampliar las operaciones como borrar, traer elementos, buscar, entre otros.

PARTE 1 Creación de una clase llamada List

Paso 1: Creación de la Clase "List" en `list.h`

1. En Visual Studio Code, dentro de la carpeta de tu proyecto llamada "Laboratorio 3", crea un nuevo archivo llamado `list.h`.
2. Abre `list.h` y define la clase "List" con sus atributos y métodos. Por ejemplo:

```
C/C++
#ifndef LIST_H
#define LIST_H

#include <iostream>
#include <cassert>

using namespace std;

template<typename T>
struct Node{
  T data;
  Node<T>* next;
};

template<typename T>
class List {
private:
  Node<T>* begin;
```

```
  int count;
  Node<T>* makeNode(const T& value);
public:
  List();
  ~List();
  void insert(int pos, const T& value);
  int size() const;
};

#endif //LIST_H
```

Paso 2: Implementación de la Clase en `list.cpp`

3. Crea un archivo llamado `list.cpp` en la misma ubicación que `list.h`.
4. Abre `list.cpp` y define los métodos de la clase "List" que has declarado en `list.h`. Por ejemplo:

```
C/C++
#ifndef LIST_CPP
#define LIST_CPP

#include "list.h"

template<typename T>
List<T>::List():begin(0), count(0){

}
template<typename T>
List<T>::~List(){
  Node<T>* del = begin;
  while (begin){
        begin = begin->next;
        delete del;
        del = begin;
  }
}
```

```
template<typename T>
Node<T>* List<T>::makeNode(const T &value) {
  Node<T>* temp = new Node<T>;
  temp->data = value;
  temp->next = 0;
  return temp;
}
template<typename T>
void List<T>::insert(int pos, const T &value) {
  if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
  }
  Node<T>* add = makeNode(value);
  if(pos == 0){
        add->next = begin;
        begin = add;
  }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
        cur = cur->next;
        }
        add->next = cur->next;
        cur->next = add;
  }
  count++;
}

template<typename T>
int List<T>::size() const {
  return count;
}


#endif
```

Paso 3: Incluye la clase List a main.cpp

5. Ahora se debe incluir la clase List al archivo de programa main.cpp para poder utilizarlo como una librería. El nombre del .h debe coincidir con el nombre del archivo.

```C/C++
#include "list.h"
using namespace std;

int main() {

  List<string> list;
  list.insert(0, "nestor");
  list.insert(1, "victor");
  list.insert(2, "Maria");
  list.insert(3, "juan");
  list.insert(4, "pedro");

  cout << "\nSize: " << list.size() << endl;
  return 0;
}
```

6. Antes de compilar por separado se debe aclarar de manera explícita al compilador de C++ sobre qué clases plantilla debe crear. Aquí agregamos un paso extra distinto a como solemos compilar clases normales. Para esto debemos agregar al final una sentencia en la implementación de `list.cpp`.

```C/C++
template class List<string>;
```

Le pedimos al compilador generar una clase List con la implementación de string, ya que en el `main.cpp` estamos creando una lista de nombres.

Paso 4: Compilar y ejecutar los Archivos

7. Abre la terminal en Visual Studio Code (si no está abierta).
8. Navega a la ubicación de tu proyecto "Laboratorio 4" si no lo has hecho aún.

9. Compila los archivos `main.cpp` y `list.cpp` juntos utilizando el compilador de C++. Por ejemplo:

```
Unset
g++ -o laboratorio3 main.cpp list.cpp

./laboratorio3
```

10. Ejercicio 1. Crear una lista de tipos de dato double con 3 elementos e imprimir su tamaño.
11. Ejercicio 2. Crear una lista de objetos de una clase llamada Punto con 4 elementos e imprimir su tamaño. La clase Punto tiene dos atributos x e y. Debe crear definición de la clase .h y definición de los método de la clase en .cpp.

PARTE 2 Operación Borrar.

Paso 5: Creación de la operación borrar en `list.h`

1. En el archivo `list.h`, debes definir un método que permita borrar cualquier nodo de la lista. Agrega este método a la definición de la clase "List" como sigue:

```
C/C++
template<typename T>
class List {
...
public:
   ...
   void erase(int pos);
   ...
};
```

Paso 6: Implementación del método borrar en `list.cpp`

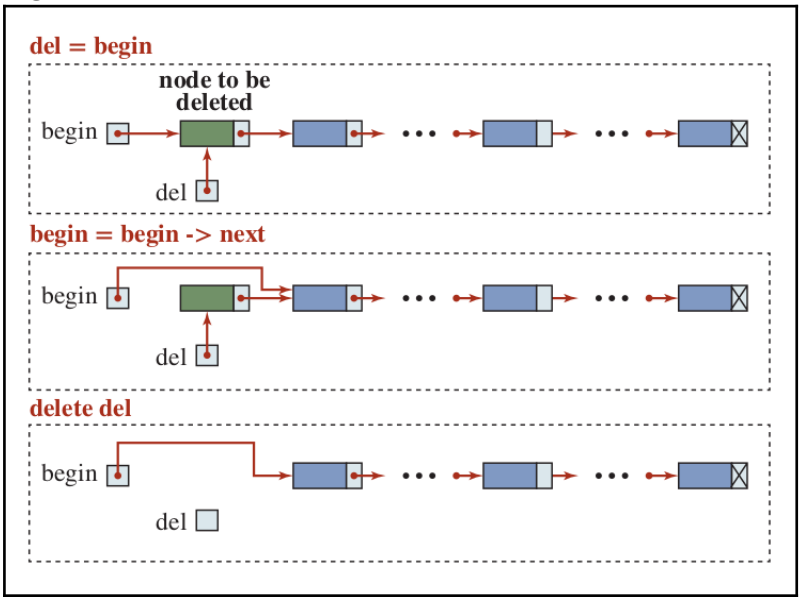
2. Ahora, en el archivo `list.cpp`, implementa el método que has declarado en `list.h`. Aquí hay un ejemplo de cómo implementarlos:

```C/C++
template<typename T>
void List<T>::erase(int pos) {
	// Verificar si la posición es válida
	if(pos < 0 || pos>count){
		cout << "Error! The position is out of range." << endl;
		return;
	}

	// Verificar si la posición es la primera (posición 0)
	if(pos == 0){

		// Eliminar el primer nodo
		Node<T>* del = begin;
		begin = begin->next;
		delete del;
	}else{
		// Encontrar el nodo anterior al que se quiere eliminar
		Node<T>* cur = begin;
		for(int i=0; i<pos-1; i++){
			cur = cur->next;
		}
		// Eliminar el nodo en la posición dada
		// Comienzar ejercicio 3
		// Agregar tu código aquí
		// Fin del ejercicio 3
	}

	// Reducir el contador de elementos en la lista
	count--;
}
```

Borrar un nodo al principio de la lista, como se muestra en la **figura 1**, requiere los siguientes pasos:

- Se crea un puntero apuntando al primer nodo de la lista (begin).
- Luego, se actualiza el puntero begin para que apunte al siguiente nodo de la lista, es decir, al segundo elemento.
- Finalmente, se elimina el nodo original al que apuntaba del, liberando así la memoria ocupada por ese nodo.

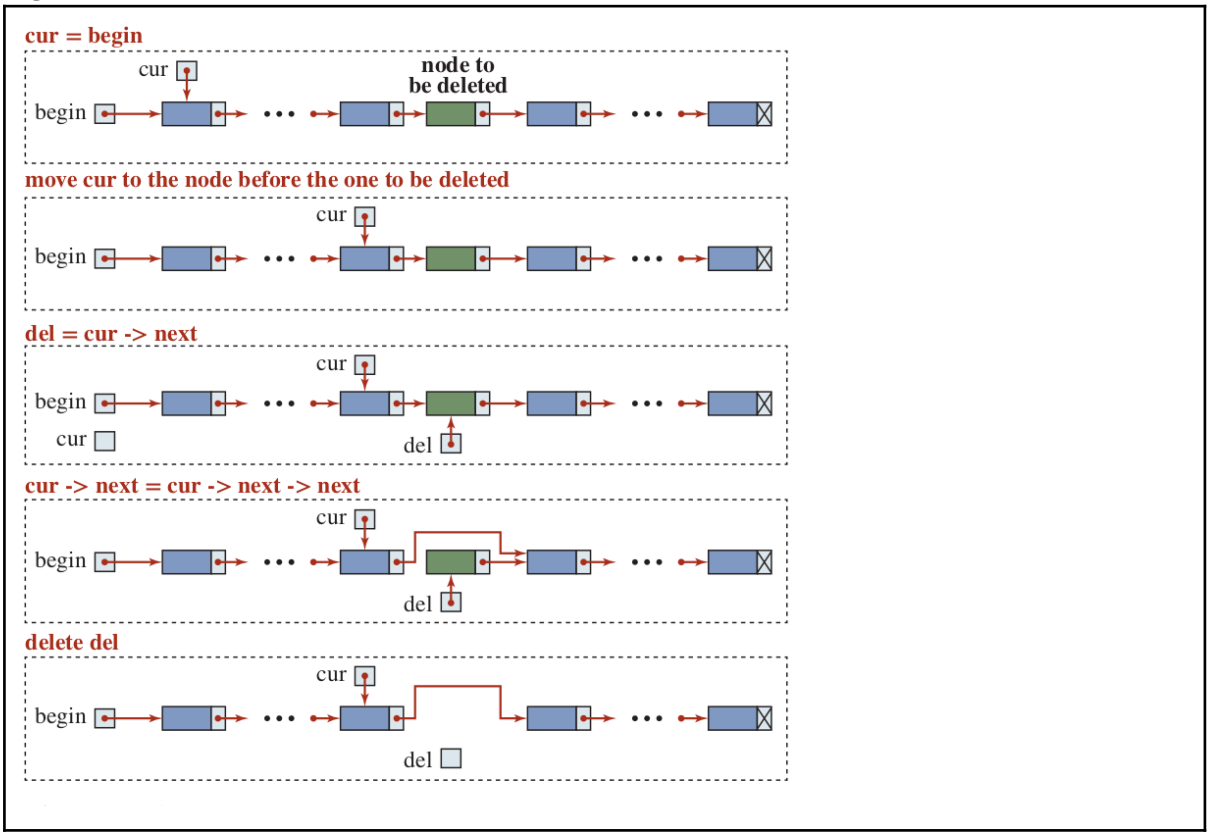
Figura 1. *Borrar el primer nodo de una lista.*



Borrar un nodo en una posición distinta al primer elemento de la lista, como se muestra en la **figura 2**, requiere los siguientes pasos:

- Se crea un puntero cur que se inicializa apuntando al primer nodo de la lista (begin).
- Luego, se utiliza un bucle for para mover el puntero cur hacia el nodo que está justo antes del que se desea eliminar. El bucle se ejecuta pos - 1 veces, lo que coloca cur en la posición anterior a la que se quiere eliminar.
- Se crea un puntero del que apunta al nodo que se desea eliminar, es decir, el nodo en la posición pos.
- Se actualiza el puntero cur->next para que apunte al nodo que está después del nodo del, saltando así el nodo que se va a eliminar.
- Finalmente, se elimina el nodo del, liberando la memoria ocupada por ese nodo.

Figura 2. *Borrar un nodo en cualquier posición.*



Paso 7: Ejemplos de Uso en `main.cpp`

3. En el archivo `main.cpp`, puedes eliminar un elemento de la lista. Aquí hay ejemplos:

```C/C++
#include "list.h"
using namespace std;

int main() {
```

```
    List<string> list;
    list.insert(0, "nestor");
    list.insert(1, "victor");
    list.insert(2, "Maria");
    list.insert(3, "juan");
    list.insert(4, "pedro");
    list.print();
    cout << "\nSize: " << list.size() << endl;

    list.erase(2);
    list.erase(3);
    cout << "\nSize: " << list.size() << endl;
    return 0;
}
```

Paso 8: Ejercicio 3.

4. Siguiendo ejemplo del paso a paso y la figura dos sobre eliminar un nodo en cualquier posición. Completar el código cpp anterior para esta operación.

PARTE 3 Traer un elemento de la lista

Paso 9: Método get para traer elementos.

1. En el archivo `list.h`, vamos agregar a la definición de la clase una operación para traer elementos de la lista.

```C/C++
template<typename T>
class List {
...
```

```
public:
    ...
    T& get(int pos) const;
    ...
};
```

2. En el archivo `list.cpp`, vamos a agregar la definición de este método de la siguiente manera.

```
C/C++
template<typename T>
T& List<T>::get(int pos) const{
       // Verificar si la posición es válida (dentro del rango permitido)
       if(pos < 0 || pos>count-1){
              cout << "Error! The position is out of range." << endl;
              assert(false);
       }
       // Si la posición es 0, devolver el valor del primer elemento
       if(pos == 0){
              return begin->data;
       }else{
              // Comienzar ejercicio 4
              // Agregar tu código aquí
              // Fin del ejercicio 4
   }
}
```

4. En el archivo `main.cpp`, puedes traer un elemento de la lista. Aquí hay ejemplos:

```
C/C++
#include "list.h"
using namespace std;
```

```
int main() {

   List<string> list;
   list.insert(0, "nestor");
   list.insert(1, "victor");
   list.insert(2, "Maria");
   list.insert(3, "juan");
   list.insert(4, "pedro");

   cout << "\nSize: " << list.size() << endl;
   cout << "Element (Maria): " << list.get(2) << endl;

   return 0;
}
```

Paso 10: Ejercicio 4

4. Completar la implementación anterior usando la misma lógica para movernos a la posición deseada (como en insert y erase) para traer el elemento (data) del nodo de la posición diferente a la primer elemento. Consejo incluir también un puntero cur.

PARTE 4 Imprimir los elementos de lista.

Paso 11: Entender cómo imprimir los elementos de una lista enlazada simple.

1. En el archivo `list.h`, vamos a crear un método que nos permita imprimir cada uno de los elementos de la lista.

```C/C++
template<typename T>

class List {

...

public:

   ...

   void print() const;

   ...

};
```

2. Ahora, en el archivo `list.cpp` se agregará la definición de la siguiente manera.

```C/C++
template<typename T>

void List<T>::print() const{

      // Verificar si la lista está vacía

      if(count == 0){

      cout << "List is empty." << endl;

      return;

      }


      // Si la lista no está vacía, comenzar desde el primer
nodo
```

```
    Node<T>* cur = begin;


    // Usar un bucle while para recorrer la lista y mostrar sus elementos. Se ejecuta mientras cur apunte a un nodo válido en la lista.
    while(cur){
    cout << cur->data << " "; // Imprimir el valor del nodo actual
    cur = cur->next; // Avanzar al siguiente nodo en la lista
    }
}
```

Paso 12: Probar

4. Probar en el archivo main.cpp para imprimir todos los elementos contenidos en la lista. Aquí hay ejemplos:

```
C/C++
#include "list.h"
#include "point.h"
using namespace std;

int main() {

   List<string> list;
   list.insert(0, "nestor");
   list.insert(1, "victor");
   list.insert(2, "Maria");
   list.insert(3, "juan");
```

```
    list.insert(4, "pedro");
    list.print();

    return 0;
}
```

Paso 13: Ejercicio 5

5. Agregar una método a la definición de clase (.h) y su implementación (.cpp) para imprimir una lista en el orden inverso.

PARTE FINAL ENVIAR SOLUCIÓN

Paso 14: Guardar el Proyecto en un Archivo `.zip`

Paso 15: Generar un Informe en PDF

3. Los estudiantes deben crear un informe en PDF que describa la solución de cada ejercicio realizado en la guía de laboratorio. Pueden utilizar herramientas como Microsoft Word o LaTeX para crear el informe. Asegúrate de que el informe contenga:
    - Título del laboratorio.
    - Nombres de los estudiantes o grupo.
    - Introducción al problema.
    - Descripción de las soluciones para cada ejercicio con evidencia (captura del ejercicios compilando).
    - Código fuente relevante (pueden incluir fragmentos de código en el informe).
    - Resultados y conclusiones.
4. Exporten el informe como un archivo PDF y cargarlo junto con el comprimido en .zip. El nombre del archivo se debe renombrar como el prefijo informe seguido de los 4 últimos dígitos de los estudiantes, usar el guión como separador.