PRÁCTICA/LABORATORIO Nº 04

LISTAS DOBLEMENTE ENLAZADAS Y

LISTAS CIRCULARES DOBLEMENTE ENLAZADAS

Objetivos:

- Utilizar Dev-C++.
- Implementar listas doblemente enlazadas y listas circulares doblemente enlazadas.

Instrucciones:

- Analizar y comprender la sección 0.
- Descargar las librerías **base** del aula virtual: **nodo.hpp**, **lista_doble.hpp**.
- Analizar y probar el comportamiento de las librerías base usando el archivo **main.cpp** disponible en el aula virtual.
- Se recomienda pruebe implementar desde 0 los métodos presentados en la sección 0, a fin de asegurarse que puede programar las funciones básicas de una lista doblemente enlazada.
- Para el envío de la resolución de la práctica, adjuntar el/los archivo/s:
  - **nodo.hpp (obligatorio)** y/o
  - **lista_doble.hpp (si ha resuelto la sección 1)** y/o
  - **lista_doble_ordenada.hpp** (**si ha resuelto la sección 2**) y/o
  - **ruleta.cpp (si ha resuelto la sección 3)**.

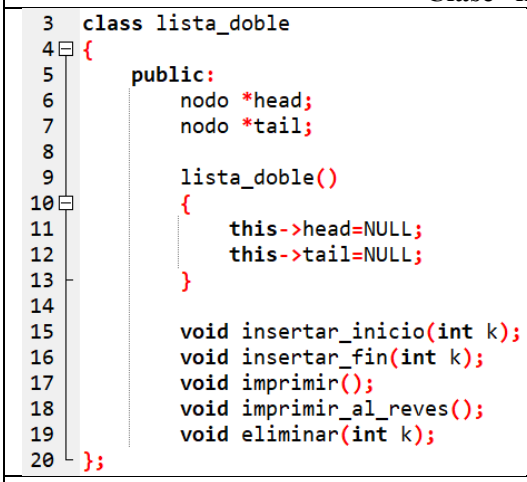
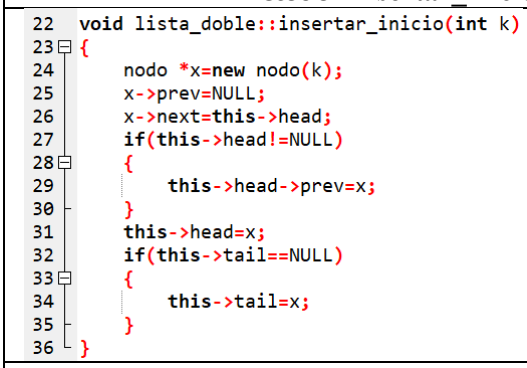
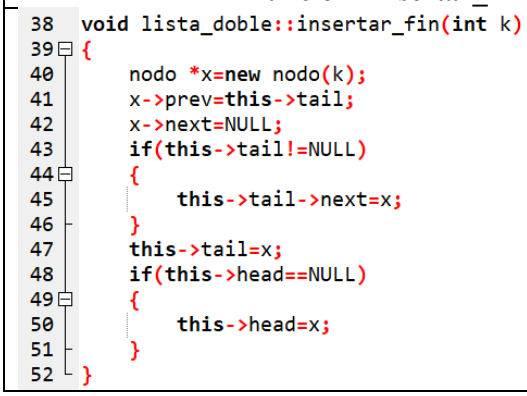
0. Ejercicios con listas doblemente enlazadas

A continuación, se presenta el archivo base **nodo.hpp**:

Código	Descripción
Estructura "nodo":	

`struct nodo`
`{`
`  int key;`
`  nodo *prev;`
`  nodo *next;`
`  nodo(int k)`
`  {`
`  this->key=k;`
`  }`
`};`
	La estructura **nodo** solo contiene la clave (**key**) y dos **punteros**: uno al anterior nodo (**prev**) y uno al siguiente nodo (**next**).
	
	Esto permite hacer algunas operaciones de forma más **sencilla**, pero tiene el **costo** de tener que mantener ambos punteros.
	
	La estructura **nodo** puede contener **más atributos**.

A continuación, se presenta el archivo base **lista_doble.hpp**:

Código	Descripción
Clase "lista_doble"	
	La clase **lista_doble** es en la mantiene la lista doblemente enlazada y donde implementaremos los **métodos** (operaciones de la lista). Como atributos de la **lista_doble**, tenemos el puntero a **nodo cabeza (head) y nodo cola (tail)** de la lista doblemente enlazada.
Método "insertar_inicio" de la clase "lista_doble"	
	Es un **método** de la clase **lista_doble** que permite **crear un nodo, cargarlo de información** e **insertarlo al inicio** de la lista doblemente enlazada. Para ello deberemos **mantener** correctamente los enlaces a **head** y **tail**.
Función "insertar_fin" de la clase "lista_doble"	
	Es un **método** de la clase **lista_doble** que permite **crear un nodo, cargarlo de información** e **insertarlo al final** de la lista doblemente enlazada. Para ello deberemos **mantener** correctamente los enlaces a **head** y **tail**.

Función "imprimir" de la clase "lista_doble"

```
54  void lista_doble::imprimir()
55  {
56      nodo *x=this->head;
57      while(x!=NULL)
58      {
59          cout<<x->key<<" ";
60          x=x->next;
61      }
62      cout<<endl;
63  }
```

Es un **método** de la clase **lista_doble** que permite **imprimir,** desde la cabeza (**head**) a través del puntero **next**, los elementos de la lista doblemente enlazada separados por un espacio.

Función "imprimir_al_reves" de la clase "lista_doble"

```
65  void lista_doble::imprimir_al_reves()
66  {
67      nodo *x=this->tail;
68      while(x!=NULL)
69      {
70          cout<<x->key<<" ";
71          x=x->prev;
72      }
73      cout<<endl;
74  }
```

Es un **método** de la clase **lista_doble** que permite **imprimir,** desde la cola (**tail**) a través del puntero **prev**, los elementos de la lista doblemente enlazada separados por un espacio.

Nota: Es importante comprobar en funcionamiento de los métodos implementados y a implementarse utilizando **imprimir** e **imprimir_al_reves** para asegurarnos de que mantenemos todos los enlaces de forma correcta.

Función "eliminar" de la clase "lista_doble"

```
76  void lista_doble::eliminar(int k)
77  {
78      nodo *x=this->head;
79      while(x!=NULL and x->key!=k)
80      {
81          x=x->next;
82      }
83      if(x!=NULL)
84      {
85          if(x->prev!=NULL)
86          {
87              x->prev->next=x->next;
88          }
89          else
90          {
91              this->head=x->next;
92          }
93          if(x->next!=NULL)
94          {
95              x->next->prev=x->prev;
96          }
97          else
98          {
99              this->tail=x->prev;
100         }
101         delete x;
102     }
103 }
```

Es un **método** de la clase **lista_doble** que permite **eliminar** la primera aparición de un nodo con key = **k** de la lista doblemente enlazada.

Casos a considerarse en eliminación:

- Si no hay nodos.
- Si el nodo a eliminar es **cabeza**.
- Si el nodo a eliminar es **cola**.
- Si el nodo a eliminar **no es cabeza ni cola**.

Es importante señalar que **pueden existir otras implementaciones** del método eliminar. Son todas las implementaciones **válidas** siempre y cuando mantengan los enlaces **prev** y **next** de cada nodo de forma correcta, así como los enlaces **head** y **tail** de la **lista_doble**.

A continuación, se presenta y describe el código del archivo **main.cpp**:

Código	Descripción
Función main()	

`#include<bits/stdc++.h>`
`#include"nodo.hpp"`
`#include"lista_doble.hpp"`
 	
`using namespace std;`
 	
`int main()`
`{`
`  lista_doble x;`
`  x.insertar_inicio(2); // 2`
`  x.insertar_fin(3); // 2 3`
`  x.insertar_inicio(1); // 1 2 3`
`  x.insertar_fin(4); // 1 2 3 4`
 	
`  x.imprimir(); // 1 2 3 4`
`  x.imprimir_al_reves(); // 4 3 2`
 	
`  x.eliminar(2);`
`  x.imprimir(); // 1 3 4`
`  x.imprimir_al_reves(); // 4 3 1`
`  x.eliminar(3);`
`  x.imprimir(); // 1 4`
`  x.imprimir_al_reves(); // 4 1`
`  x.eliminar(1);`
`  x.imprimir(); //4`
`  x.imprimir_al_reves(); //4`
`  x.eliminar(4);`
`  x.imprimir(); //`
`  x.imprimir_al_reves(); //`
`  return 0;`
`}`
	Usaremos el archivo **main.cpp** para **probar (sandbox)** los métodos implementados en nuestra clase **lista_doble**.
	
	Es importante tomar en cuenta el orden de las librerías que incluimos, primero la librería que contiene la estructura nodo (**nodo.hpp**) y luego la librería que contiene la lista doblemente enlazada (**lista_doble.hpp**) que depende de la estructura nodo para funcionar. De incluirlas al revés (primero **lista_doble.hpp** y luego **nodo.hpp**) obtendremos un error.
	
	Nuevamente, es importante probar nuestras implementaciones con **imprimir()** e **imprimir_al_reves()** para asegurarnos que todos los punteros han sido **mantenidos** de forma correcta a través de las operaciones.

Para todas las implementaciones, verificar todos los casos.

1. Implementar en el archivo lista_doble.hpp los siguientes métodos: (12 puntos)

`insertar_antes_de_key(int k, int kk)` **(1 punto) O(n)**

- inserta el valor **kk** justo antes de la primera aparición de **k**.

`insertar_despues_de_key(int k, int kk)` **(1 punto) O(n)**

- inserta el valor **kk** justo después de la primera aparición de **k**.

`eliminar_inicio()` **(1 punto) O(1)**

- elimina el primer elemento de la lista.

`eliminar_fin()` **(1 punto) O(1)**

- elimina el último elemento de la lista.

`eliminar_antes_de_key(int k)` **(1 punto) O(n)**

- elimina el nodo anterior a la primera aparición de **k**.

`eliminar_despues_de_key(int k)` **(1 punto) O(n)**

- elimina el nodo posterior a la primera aparición de **k**.

`ultimo_primero()` **(2 puntos) O(1)**

- mueve el último nodo a la primera posición.
- usar solo punteros y sin crear nuevos nodos.

`intercambiar(int p1, int p2)` **(2 puntos) O(n)**

- intercambia los nodos en las posiciones p1 y p2.
- usar solo punteros y sin crear nuevos nodos.

`invertir()` **(2 puntos) O(n)**

- invierte el orden de los nodos.
- usando solo punteros y sin crear nuevos nodos.

2. Implementar en el archivo lista_doble_ordenada.hpp la clase lista_doble_ordenada que mantiene en una lista doblemente enlazada el orden ascendente de nodos que almacenan un entero key:

`imprimir()` **(0 puntos, OBLIGATORIO)**

- imprime el atributo key de los nodos separados por un espacio desde **head**.

`imprimir_al_reves()` **(0 puntos, OBLIGATORIO)**

- imprime el atributo key de los nodos separados por un espacio desde **tail**.

`insertar(int k)` **(1.5 puntos) O(n)**

- inserta el valor k en la lista manteniendo el orden ascendente.

`eliminar(int k)` **(1.5 puntos) O(n)**

- considerar que puede ser el primer nodo.

`eliminar_duplicados()` **(2 puntos) O(n)**

- elimina los nodos duplicados en una lista enlazada.
- considerar que la lista está ordenada.

3. Ruleta de la Fortuna (3 puntos)

Implementar en el archivo ruleta.cpp la estructura nodo y clase lista_circular_doble y la solución al siguiente problema.

Descripción

Tienes una ruleta con **N** casillas (la imagen superior corresponde a una ruleta de 12 casillas) en donde se encuentran los montos a premiarse $P_0,P_1,P_2,\ldots,P_{N-1}$ ubicados en forma horaria $P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_{N-1} \rightarrow P_0 \rightarrow P_1 \rightarrow \ldots$. Antes de girar la ruleta, la flecha que indica el premio a ganar se ubica justo entre P_0 y P_{N-1} pero sin estar en ninguna de ambas casillas (como se puede ver en la imagen). Al girar la ruleta con una fuerza **F**, la flecha avanza o retrocede **F** casillas, dependiendo si el valor de **F** es positivo (movimiento horario de la ruleta) o negativo (movimiento anti horario de la ruleta), pudiendo dar una o varias vueltas a la ruleta, hasta detenerse. Se debe mostrar cuál es premio de la casilla en la cual se detiene la flecha de la ruleta.

Entrada

Un entero **N**, seguido de **N** enteros $P_0,P_1,P_2,\ldots,P_{N-1}$, finalmente un entero **F.** Puedes suponer que $0<P_i,N<1{,}000{,}000$, que $-1{,}000{,}000<F<1{,}000{,}000$, adicionalmente $F \neq 0$.

Salida

El premio ganador.

Ejemplo

Entrada	Salida
5 10 100 20 50 200 6	10
5 10 100 20 50 200 -6	200

Las entradas y salidas son tal cual el ejemplo, no existen mensajes adicionales como "ingrese …". Para cin>> es igual ingresar un ENTER o un ESPACIO como fin de ingreso de dato.