PRÁCTICA/LABORATORIO Nº 03

LISTAS SIMPLEMENTE ENLAZADAS

Objetivos:

- Utilizar Dev-C++.
- Implementar estructuras (**nodo**) y clases (**lista_simple, lista_simple_ordenada**).

Instrucciones:

- Analizar y **probar** el comportamiento del código presentado en la sección 0 (disponible en el aula virtual).
- Para el envío de la resolución de la práctica, adjuntar el/los archivo **lista_simple.hpp** y/o **lista_simple_ordenada.hpp** dentro de un **zip**.

0. Ejercicios con listas simplemente enlazadas

A continuación, se presentan y describen fragmentos del archivo **lista_simple.hpp**:

Código	Descripción
Estructura "nodo":	
5 struct nodo 6 { 7 int key; 8 nodo *next; 9 };	La estructura **nodo** solo contiene la clave (**key**) y el **puntero** al siguiente elemento. El tipo de dato de **key** puede ser diferente (por ejemplo *string*). El atributo **key** representa de manera **única** a cada nodo y que usaremos para los métodos a implementarse. La estructura **nodo** puede contener **más atributos**.
Clase "lista_simple"	

11 class lista_simple
12 {
13 public:
14 nodo *head;
15
16 lista_simple()
17 {
18 this->head=NULL;
19 }
20
21 void insertar_inicio(int k);
22 void insertar_fin(int k);
23 void imprimir();
24 void eliminar_key(int k);
25 int buscar_key(int k);
26 };
	La clase **lista_simple**, es en la mantiene la lista enlazada y donde implementaremos los **métodos**.
	
	Como atributos de la **lista_simple**, solo tenemos el **nodo cabeza (head)** de la lista simplemente enlazada.

Función "insertar_inicio" de la clase "lista_simple"	
28 void lista_simple::insertar_inicio(int k) 29 { 30 nodo *x = new nodo; 31 x->key = k; 32 x->next=this->head; 33 this->head=x; 34 }	Es un **método** de la clase **lista_simple** que permite **crear un nodo, cargarlo de información** e **insertarlo al inicio** de la lista simplemente enlazada.

Función "insertar_fin" de la clase "lista_simple"

```
36  void lista_simple::insertar_fin(int k)
37  {
38      if(this->head == NULL)
39      {
40          this->insertar_inicio(k);
41      }
42      else
43      {
44          nodo *x = this->head;
45          while(x->next != NULL)
46          {
47              x = x->next;
48          }
49          nodo *y = new nodo;
50          y->key = k;
51          y->next = x->next;
52          x->next = y;
53      }
54  }
```

Es un **método** de la clase **lista_simple** que permite **crear un nodo, cargarlo de información** e **insertarlo al final** de la lista simplemente enlazada.

Función "imprimir" de la clase "lista_simple"

```
56  void lista_simple::imprimir()
57  {
58      nodo *x=this->head;
59      while(x!=NULL)
60      {
61          cout<<x->key<<' ';
62          x=x->next;
63      }
64      cout<<endl;
65  }
```

Es un **método** de la clase **lista_simple** que permite **imprimir** los elementos de la lista simplemente enlazada separados por un espacio.

Función "eliminar_key" de la clase "lista_simple"

```
69  void lista_simple::eliminar_key(int k)
70  {
71      if(this->head != NULL)
72      {
73          nodo *x = this->head;
74          if(x->key == k)
75          {
76              this->head = this->head->next;
77              delete x;
78          }
79          else
80          {
81              while(x->next != NULL and x->next->key != k)
82              {
83                  x = x->next;
84              }
85              if(x->next != NULL)
86              {
87                  nodo *y = x->next;
88                  x->next = x->next->next;
89                  delete y;
90              }
91          }
92      }
93  }
```

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

Función "buscar_key" de la clase "lista_simple"

```
95  int lista_simple::buscar_key(int k)
96  {
97      nodo *x = this->head;
98      while(x != NULL and x->key != k)
99      {
100         x = x->next;
101     }
102     if(x != NULL)
103     {
104         return 1;
105     }
106     else
107     {
108         return 0;
109     }
110 }
```

Es un **método** de la clase **lista_simple** que permite **buscar** un nodo con key = **k** de la lista simplemente enlazada.

Devuelve **1** si lo encuentra, caso contrario devuelve **0**.

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

Función main()

```cpp
#include<iostream>
#include "lista_simple.hpp"

using namespace std;

int main()
{
    lista_simple lista;
    lista.insertar_inicio(55);
    lista.imprimir(); //55
    lista.insertar_inicio(44);
    lista.imprimir(); //44 55
    lista.insertar_inicio(33);
    lista.imprimir(); //33 44 55
    lista.insertar_inicio(22);
    lista.imprimir(); //22 33 44 55
    lista.insertar_inicio(11);
    lista.imprimir(); //11 22 33 44 55

    lista.eliminar_key(11);
    lista.imprimir(); //22 33 44 55
    lista.eliminar_key(33);
    lista.imprimir(); //22 44 55
    lista.eliminar_key(55);
    lista.imprimir(); //22 44

    cout<<lista.buscar_key(22)<<endl; //1
    cout<<lista.buscar_key(33)<<endl; //0

    lista.insertar_fin(66);
    lista.imprimir(); //22 44 66
}
```

Es la función del archivo **main.cpp** en la que podemos **probar** nuestra clase **lista_simple**.

1. Implementar en el archivo lista_simple.hpp los siguientes métodos: (1 punto c/u)

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

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

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

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

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

- elimina el primer elemento de la lista.

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

- elimina el último elemento de la lista.

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

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

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

- elimina el nodo posterior a **k**.

`imprimir_posicion()` **(1 puntos)**

- imprime el atributo key del elemento ubicado en la posición p.

`sumar()` **(1 punto)**

- retorna (**`return`**) la suma de todos los nodos.

`sumar_posiciones_par()` **(1 punto)**

- retorna (**`return`**) la suma de todos los nodos en posición par.

`ultimo_primero()` **(2 puntos)**

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

`invertir()` **(2 puntos)**

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

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

`imprimir()` **(1 puntos)**

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

`insertar(int k)` **(2 puntos)**

- considerar que se debe mantener el orden ascendente.

`eliminar(int k)` **(2 puntos)**

- considerar que puede ser el primer nodo.

`buscar(int k)` **(2 puntos)**

- retorna (**`return`**) **1** si encuentra el elemento **k**, caso contrario retorna **0**.
- considerar que la lista está ordenada.