Universidad de Panamá

Facultad de Informática, Electrónica Comunicación

Escuela: De ingeniería en informática

Asignatura: Programación II

Profesora: Aneyka Hurtado

Nombre: Carlos Correa 8-1007-1440

Fecha de Entrega: 07/06/2023

Análisis: Circular.

C: > Users > corre > Desktop > C++ circular.cpp > Insertar(int)

```cpp
// Incluye la biblioteca iostream para permitir la entrada y salida estándar
#include <iostream>
using namespace std;

// Clase que representa un nodo en la lista
class nodo {
   public:
    // Constructor que inicializa el valor y el puntero siguiente
    nodo(int v, nodo *sig = NULL)
    {
       valor = v;
       siguiente = sig;
    }

   private:
    int valor;
    nodo *siguiente;

   friend class lista; // Declara la clase lista como amiga para permitir el acceso a los miembros privados
};

typedef nodo *pnodo; // Define el alias pnodo para un puntero a nodo

// Clase que representa la lista circular enlazada
class lista {
   public:
    // Constructor que inicializa el puntero actual como NULL
    lista() { actual = NULL; }

    // Destructor que libera la memoria ocupada por los nodos de la lista
    ~lista();

    // Método para insertar un valor en la lista
    void Insertar(int v);

    // Método para borrar un valor de la lista
    void Borrar(int v);
```

```cpp
    void Borrar(int v);

    // Método que devuelve verdadero si la lista está vacía
    bool ListaVacia() { return actual == NULL; }

    // Método para mostrar los elementos de la lista
    void Mostrar();

    // Método para mover el puntero actual al siguiente nodo
    void Siguiente();

    // Método que devuelve verdadero si el puntero actual no es NULL
    bool Actual() { return actual != NULL; }

    // Método que devuelve el valor del nodo actual
    int ValorActual() { return actual->valor; }

   private:
    pnodo actual; // Puntero al nodo actual en la lista
};

// Destructor de la clase lista
lista::~lista()
{
   pnodo nodo;

   // Mientras la lista tenga más de un nodo
   while(actual->siguiente != actual) {
      // Borrar el nodo siguiente al apuntado por actual
      nodo = actual->siguiente;
      actual->siguiente = nodo->siguiente;
      delete nodo;
   }
   // Y borrar el último nodo
   delete actual;
   actual = NULL;
}
```

```
74
75  // Método para insertar un valor en la lista
76  void lista::Insertar(int v)
77  {
78     pnodo Nodo;
79
80     // Creamos un nodo para el nuevo valor a insertar
81     Nodo = new nodo(v);
82
83     // Si la lista está vacía, la lista será el nuevo nodo
84     // Si no lo está, insertamos el nuevo nodo a continuación del apuntado
85     // por actual
86     if(actual == NULL) actual = Nodo;
87     else Nodo->siguiente = actual->siguiente;
88     // En cualquier caso, cerramos la lista circular
89     actual->siguiente = Nodo;
90  }
91
92  // Método para borrar un valor de la lista
93  void lista::Borrar(int v)
94  {
95     pnodo nodo;
96
97     nodo = actual;
98
99     // Hacer que actual apunte al nodo anterior al de valor v
100    do {
101       if(actual->siguiente->valor != v) actual = actual->siguiente;
102    } while(actual->siguiente->valor != v && actual != nodo);
103    // Si existe un nodo con el valor v:
104    if(actual->siguiente->valor == v) {
105       // Y si la lista sólo tiene un nodo
106       if(actual == actual->siguiente) {
107          // Borrar toda la lista
108          delete actual;
109          actual = NULL;
110       }
```

```
111       else {
112          // Si la lista tiene más de un nodo, borrar el nodo de valor v
113          nodo = actual->siguiente;
114          actual->siguiente = nodo->siguiente;
115          delete nodo;
116       }
117    }
118 }
119
120 // Método para mostrar los elementos de la lista
121 void lista::Mostrar()
122 {
123    pnodo nodo = actual;
124
125    // Recorre la lista circular y muestra los valores de los nodos
126    do {
127       cout << nodo->valor << "-> ";
128       nodo = nodo->siguiente;
129    } while(nodo != actual);
130
131    cout << endl;
132 }
133
134 // Método para mover el puntero actual al siguiente nodo
135 void lista::Siguiente()
136 {
137    if(actual) actual = actual->siguiente;
138 }
139
140 // Función principal del programa
141 int main()
142 {
143    lista Lista;
144
145    // Insertar valores en la lista
146    Lista.Insertar(20);
147    Lista.Insertar(10);
```

```
    Lista.Insertar(40);
    Lista.Insertar(30);
    Lista.Insertar(60);

    // Mostrar los elementos de la lista
    Lista.Mostrar();

    cout << "Lista de elementos:" << endl;

    // Borrar algunos valores de la lista
    Lista.Borrar(10);
    Lista.Borrar(30);

    // Mostrar los elementos actualizados de la lista
    Lista.Mostrar();

    cin.get();
    return 0;
}
```

Análisis: doble sentido.

```
#include <iostream>
#include <CCadena.h> // Incluye el archivo de encabezado para la clase Cadena
using namespace std;

#define ASCENDENTE 1
#define DESCENDENTE 0

template<class TIPO> class lista; // Declaración adelantada de la clase lista

template<class TIPO>
class nodo {
public:
    // Constructor de la clase nodo
    nodo(TIPO v, nodo<TIPO> *sig = NULL, nodo<TIPO> *ant = NULL) :
       valor(v), siguiente(sig), anterior(ant) {}

private:
    TIPO valor; // Almacena el valor del nodo
    nodo<TIPO> *siguiente; // Puntero al siguiente nodo
    nodo<TIPO> *anterior; // Puntero al nodo anterior

    friend class lista<TIPO>; // Permite que la clase lista acceda a los miembros privados de la clase nodo
};

template<class TIPO>
class lista {
public:
    // Constructor de la clase lista
    lista() : plista(NULL) {}

    // Destructor de la clase lista
    ~lista();

    // Inserta un nuevo elemento en la lista
    void Insertar(TIPO v);

    // Elimina un elemento de la lista
```

```
    // Elimina un elemento de la lista
    void Borrar(TIPO v);

    // Comprueba si la lista está vacía
    bool ListaVacia() { return plista == NULL; }

    // Muestra los elementos de la lista en el orden especificado (ascendente o descendente)
    void Mostrar(int orden);

    // Avanza al siguiente nodo en la lista
    void Siguiente();

    // Retrocede al nodo anterior en la lista
    void Anterior();

    // Coloca el puntero plista en el primer nodo de la lista
    void Primero();

    // Coloca el puntero plista en el último nodo de la lista
    void Ultimo();

    // Comprueba si el puntero plista apunta a un nodo válido
    bool Actual() { return plista != NULL; }

    // Devuelve el valor almacenado en el nodo actual (plista)
    TIPO ValorActual() { return plista->valor; }

private:
    nodo<TIPO> *plista; // Puntero al primer nodo de la lista
};

// Implementación de destructor de la clase lista
template<class TIPO>
lista<TIPO>::~lista()
{
    nodo<TIPO> *aux;
```

```
    Primero();
    while (plista) {
        aux = plista;
        plista = plista->siguiente;
        delete aux;
    }
}

// Implementación de la función Insertar de la clase lista
template<class TIPO>
void lista<TIPO>::Insertar(TIPO v)
{
    nodo<TIPO> *nuevo;

    Primero();
    // Si la lista está vacía o el primer nodo tiene un valor mayor a v
    if (ListaVacia() || plista->valor > v) {
        // Asigna a lista un nuevo nodo de valor v y cuyo siguiente elemento es la lista actual
        nuevo = new nodo<TIPO>(v, plista);
        if (!plista) plista = nuevo;
        else plista->anterior = nuevo;
    }
    else {
        // Busca el nodo de valor menor a v
        // Avanza hasta el último elemento o hasta que el siguiente tenga un valor mayor que v
        while (plista

->siguiente && plista->siguiente->valor <= v)
            Siguiente();
        // Crea un nuevo nodo después del nodo actual con valor v
        nuevo = new nodo<TIPO>(v, plista->siguiente, plista);
        plista->siguiente = nuevo;
        if (nuevo->siguiente)
            nuevo->siguiente->anterior = nuevo;
    }
}
```

```cpp
// Implementación de la función Borrar de la clase lista
template<class TIPO>
void lista<TIPO>::Borrar(TIPO v)
{
    nodo<TIPO> *p = plista, *q = NULL;

    // Busca el nodo con el valor v
    while (p && p->valor < v) {
        q = p;
        p = p->siguiente;
    }
    if (!p || p->valor != v) return;
    else if (!q) {
        plista = p->siguiente;
        if (plista) plista->anterior = NULL;
    }
    else if (!p->siguiente) {
        q->siguiente = NULL;
    }
    else {
        q->siguiente = p->siguiente;
        p->siguiente->anterior = q;
    }
    delete p;
}

// Implementación de la función Mostrar de la clase lista
template<class TIPO>
void lista<TIPO>::Mostrar(int orden)
{
    nodo<TIPO> *p;

    if (orden == ASCENDENTE) {
        Primero();
        p = plista;
        while (p) {
            cout << p->valor << " ";
            p = p->siguiente;
        }
    }
    else {
        Ultimo();
        p = plista;
        while (p) {
            cout << p->valor << " ";
            p = p->anterior;
        }
    }
    cout << endl;
}

// Implementación de la función Siguiente de la clase lista
template<class TIPO>
void lista<TIPO>::Siguiente()
{
    if (plista) plista = plista->siguiente;
}

// Implementación de la función Anterior de la clase lista
template<class TIPO>
void lista<TIPO>::Anterior()
{
    if (plista) plista = plista->anterior;
}

// Implementación de la función Primero de la clase lista
template<class TIPO>
void lista<TIPO>::Primero()
{
    while (plista && plista->anterior)
        plista = plista->anterior;
}

// Implementación de la función Ultimo de la clase lista
```