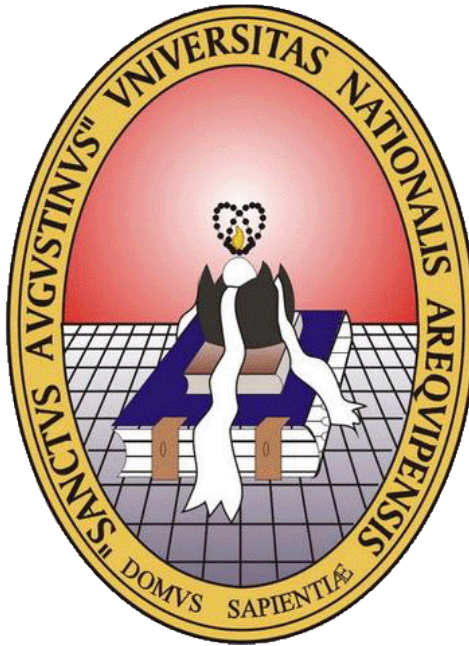


UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA
FACULTAD DE PRODUCCIÓN Y SERVICIOS
ESCUELA DE CIENCIAS DE LA COMPUTACIÓN



PRÁCTICA DE LABORATORIO 10
CURSO DE CIENCIAS DE LA COMPUTACIÓN II

ESTUDIANTE:
RUIZ MAMANI, EDUARDO GERMÁN

EMAIL: eruizm@unsa.edu.pe

CUI: 20193061

TURNO:

C

AREQUIPA- PERÚ

2021

LINK DEL REPOSITORIO: https://github.com/EGRM23/CCII_20193061.git

1. EJERCICIO

1. Defina una lista enlazada que permita insertar elementos al final de todos los elementos que ya se hayan ingresado. Por el momento no es necesario preservar un orden, simplemente los elementos nuevos deben de ingresar como el último elemento.

- **CÓDIGO**

- **Nodo.h**

```
#ifndef NODO_H
#define NODO_H
#include <iostream>

//EDUARDO GERMAN RUIZ MAMANI
//CUI: 20193061

using namespace std;

template <typename R> class Lista;
template <typename R> class Nodo {
public:
    Nodo(R val, Nodo* = NULL);
    ~Nodo();
    friend class Lista<R>;
private:
    Nodo* sig;
    R valor;
};

template <typename R>
Nodo<R> :: Nodo(R val, Nodo* s) {
    valor = val;
    sig = s;
}

template <typename R>
Nodo<R> :: ~Nodo() {}
#endif
```

- **Nodo.cpp**

```
#include "Nodo.h"

//EDUARDO GERMAN RUIZ MAMANI
//CUI: 20193061
```

- **Lista.h**

```
#ifndef LISTA_H
#define LISTA_H
#include <iostream>
#include "Nodo.h"

//EDUARDO GERMAN RUIZ MAMANI
//CUI: 20193061
```

```

using namespace std;

template <typename R> class Nodo;
template <typename R> class Lista {
public:
    Lista();
    ~Lista();
    void insertarfinal(const R);
    void insertarcomienzo(const R);
    void insertarmedio(int,const R);
    void eliminarfinal();
    void eliminarcomienzo();
    void eliminarpos(int);
    void ordenar();
    void mostrarlista();
private:
    Nodo<R>* cabeza;
    Nodo<R>* cola;
    int cant;
};

template <typename R>
Lista<R> :: Lista() {
    cabeza = NULL;
    cola = NULL;
    cant = 0;
}

template <typename R>
Lista<R> :: ~Lista() {
    Nodo<R>* temp = cabeza;
    Nodo<R>* borrar;
    while(temp != NULL){
        borrar = temp;
        temp = temp->sig;
        delete borrar;
    }
    cabeza = NULL;
    cola = NULL;
}

template <typename R>
void Lista<R> :: insertarfinal(const R val) {
    Nodo<R>* nuevo = new Nodo<R>(val);

    if (cabeza == NULL)
        cabeza = nuevo;
    else
        cola->sig = nuevo;

    nuevo->sig = NULL;
    cola = nuevo;
    cant++;
}

```

```

template <typename R>
void Lista<R> :: mostrarlista() {
    cout << "LISTA: ";
    Nodo<R>* temp = cabeza;
    while (temp != NULL) {
        cout << temp->valor << " ";
        temp = temp->sig;
    }
    cout << endl;
}
#endif

```

- **Lista.cpp**

```

#include "Lista.h"

//EDUARDO GERMAN RUIZ MAMANI
//CUI: 20193061

```

- **main.cpp**

```

#include<iostream>
#include "Nodo.h"
#include "Lista.h"

//EDUARDO GERMAN RUIZ MAMANI
//CUI: 20193061

using namespace std;

int main (int argc, char *argv[]) {
    Lista<int> l1;

    l1.insertarfinal(3);
    l1.mostrarlista();

    l1.insertarfinal(4);
    l1.insertarfinal(5);
    l1.mostrarlista();

    l1.insertarcomienzo(7);
    l1.insertarcomienzo(8);
    l1.mostrarlista();

    l1.insertarmedio(2,6);
    l1.mostrarlista();

    l1.eliminarfinal();
    l1.mostrarlista();

    l1.eliminarcomienzo();
    l1.mostrarlista();

    l1.eliminarpos(3);
    l1.mostrarlista();

    l1.ordenar();
}

```

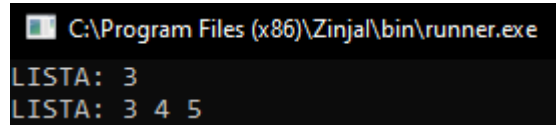
```

        l1.mostrarlista();

        l1.~Lista();
        return 0;
    }

```

- **CAPTURAS**



```

C:\Program Files (x86)\Zinjal\bin\runner.exe
LISTA: 3
LISTA: 3 4 5

```

2. EJERCICIO

2. Con la implementación de la lista enlazada anterior, desarrolle una función que permita ingresar los elementos al inicio de todos los demás elementos. Tendrá que modificar el comportamiento del puntero que tiene referencia al primer elemento para que sea redireccionado al nuevo elemento por ingresar.

- **CÓDIGO**

```

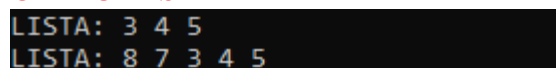
template <typename R>
void Lista<R> :: insertarcomienzo(const R val) {
    Nodo<R>* nuevo = new Nodo<R>(val);

    if (cabeza == NULL) {
        cola = nuevo;
        nuevo->sig = NULL;
    } else {
        nuevo->sig = cabeza;
    }

    cabeza = nuevo;
    cant++;
}

```

- **CAPTURAS**



```

LISTA: 3 4 5
LISTA: 8 7 3 4 5

```

3. EJERCICIO

3. Desarrolle una función que permita ingresar elementos en el medio de dos elementos de la lista enlazada, como se muestra en la siguiente imagen. Solicite que se ingrese una posición válida dentro de la lista y permita que el valor ingresado se pueda anexar en esa posición.

- **CÓDIGO**

```

template <typename R>
void Lista<R> :: insertarmedio(int pos, const R val) {
    Nodo<R>* nuevo = new Nodo<R>(val);
    Nodo<R>* temp = cabeza;
    pos--;

    if (cabeza == NULL) {
        nuevo->sig = cabeza;
        cabeza = nuevo;
    }
}

```

```

    } else {
        while(pos != 1) {
            temp = temp->sig;
            pos--;
        }

        nuevo->sig = temp->sig;
        temp->sig = nuevo;
    }

    cant++;
}

```

- **CAPTURAS**

```

LISTA: 8 7 3 4 5
LISTA: 8 6 7 3 4 5

```

4. EJERCICIO

4. Elabore una función que permita eliminar el último elemento de una lista enlazada. (Evite copiar los elementos en una nueva lista para completar la eliminación del elemento)

- **CÓDIGO:**

```

template <typename R>
void Lista<R> :: eliminarfinal() {
    Nodo<R>* temp = cabeza;
    int cont = cant;

    if (cabeza->sig == NULL) {
        delete temp;
        cabeza = NULL;
        cola = NULL;
    } else {
        while (cont != 2) {
            temp = temp->sig;
            cont--;
        }
        delete temp->sig;
        temp->sig = NULL;
        cola = temp;
    }

    cant--;
}

```

- **CAPTURAS**

```

LISTA: 8 6 7 3 4 5
LISTA: 8 6 7 3 4

```

5. EJERCICIO

5. Desarrolle una función que permita eliminar el primer elemento de una lista sin perder referencia de los demás elementos que ya se encuentran almacenados en la estructura .(Evite copiar los elementos en una nueva lista para completar la eliminación de los elementos)

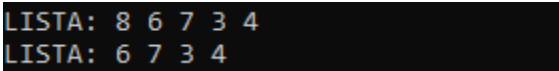
- **CÓDIGO**

```
template <typename R>
void Lista<R> :: eliminarcomienzo() {
    Nodo<R>* temp = cabeza;

    if (cabeza->sig == NULL) {
        cabeza = NULL;
        cola = NULL;
    } else
        cabeza = temp->sig;

    delete temp;
    cant--;
}
```

- **CAPTURAS**



```
LISTA: 8 6 7 3 4
LISTA: 6 7 3 4
```

6. EJERCICIO

6. Dado una posición válida dentro de la lista, permita al usuario eliminar un elemento de cualquier posición sin perder referencia de los demás elementos.

- **CÓDIGO:**

```
template <typename R>
void Lista<R> :: eliminarpos(int pos) {

    if (pos == 1)
        eliminarcomienzo();
    else if (pos == cant)
        eliminarfinal();
    else {
        Nodo<R>* temp = cabeza;
        Nodo<R>* borrar;
        pos--;

        while(pos != 1) {
            temp = temp->sig;
            pos--;
        }

        borrar = temp->sig;
        temp->sig = borrar->sig;

        delete borrar;
        cant--;
    }
}
```

- **CAPTURAS**

```
LISTA: 6 7 3 4
LISTA: 6 7 4
```

7. EJERCICIO

7. Desarrolle un algoritmo de ordenamiento que permita ordenar los elementos de forma ascendente y descendente de acuerdo a la elección del usuario. Se debe poder simular el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el menor tiempo posible (< 2 seg).

- **CÓDIGO (ERROR! Se me dificultó hacerlo funcionar, tengo que mejorarlo)**

```
template <typename R>
void Lista<R> :: ordenar() {
    Nodo<R>* temp1 = cabeza;
    Nodo<R>* temp2 = cabeza->sig;

    while(temp2 != NULL){
        if((temp1->valor) > (temp2->valor)) {
            temp1->sig = temp2->sig;
            if (temp2->sig != NULL) {
                temp2->sig = temp1;

                if((temp1->sig != NULL) && ((temp1->sig->valor) > ((temp1->sig)->valor))) {
                    temp2->sig = temp1->sig;
                }
                if(temp1 == cabeza) {
                    cabeza = temp2;
                }
            }
        } else {
            temp1 = temp2;
        }
        temp2 = temp1->sig;
    }

    cola = temp1;
}
```

- **CAPTURA (ERROR! Por el momento no funciona)**

```
LISTA: 6 7 4
LISTA: 6 7
```