UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE CIENCIAS DE LA COMPUTACIÓN

LABORATORIO 11 – Punteros, POO y Pilas.

DOCENTE:

Enzo Edir Velásquez Lobatón

ALUMNO:

Owen Haziel Roque Sosa.

FECHA:

17/06/2022

Arequipa – Perú

```cpp
#ifndef PILA_H
#define PILA_H
#include <iostream>
#include <iterator>
#include <vector>

template <class T>
class Pila;

template <class T>
class Nodo {
public:
    Nodo<T>(T, Nodo* =nullptr);
    friend class Pila<T>;
private:
    T value;
    Nodo* sig;
};

template <class T>
Nodo<T>::Nodo(T _value, Nodo* _sig) {
    sig = _sig;
    value = _value;
}

template <class T>
class Pila {
public:
    Pila<T>();
    ~Pila<T>();
    void insertar(T);
    void eliminar(T);
    void search(T);
    void imprimir();
private:
    Nodo<T> * first = nullptr;
    unsigned int size = 0;
};
```

Clase Nodo y Clase Pila

```cpp
//constructor
template <class T>
Pila<T>::Pila(){}

//destructor
template <class T>
Pila<T>::~Pila() {
    Nodo<T> *aux = first;
    Nodo<T> *delete_node;
    while(aux != nullptr) {
        delete_node = aux;
        aux = aux->sig;
        delete delete_node;
    }
    first = nullptr;
    size = 0;
}
```

Constructor y destructor de Pila

```cpp
template <class T>
void Pila<T>::imprimir(){
    Nodo<T> *aux = first;
    while(aux != nullptr) {
        std::cout << aux->value << " -> ";
        aux = aux->sig;
    }
    std::cout << '\n';
}
```

Método para mostrar la Pila

1. Defina una Pila que permita insertar elementos utilizando clases.

```
template <class T>
void Pila<T>::insertar(T val){
    Nodo<T> * nuevo = new Nodo<T>(val);
    if (first == nullptr){
        nuevo->sig=nullptr;
        first = nuevo;
    } else {
        nuevo->sig=first;
        first = nuevo;
    }
    size++;
}
```

```
#include<iostream>
#include "Pila.h"
using namespace std;

int main (int argc, char *argv[]) {
    Pila<int> A;
    A.insertar(1);
    A.imprimir();
    A.insertar(2);
    A.imprimir();
    A.insertar(7);
    A.imprimir();
    A.insertar(6);
    A.insertar(1);
    A.imprimir();

    return 0;
}
```

```
1 ->
2 -> 1 ->
7 -> 2 -> 1 ->
1 -> 6 -> 7 -> 2 -> 1 ->
```

2. Sobre el ejercicio anterior, adecue el programa para eliminar elementos de una Pila.

```
template <class T>
void Pila<T>::eliminar(T val){
    if (first == nullptr) {
        std::cout << "No existen elementos en la lista.\n";
        return;
    } // si es el primer elemento a eliminar a la lista
    else {
        Nodo<T> * delete_node = first; // asigno delete a first
        first = first->sig; // first recorre al siguiente
        delete delete_node;
    }
    size--;
}
```

```
#include<iostream>
#include "Pila.h"
using namespace std;

int main (int argc, char *argv[]) {
    Pila<int> A;
    A.insertar(1);
    A.insertar(2);
    A.insertar(7);
    A.insertar(6);
    A.insertar(1);
    A.imprimir();
    A.eliminar();
    A.imprimir();
    A.eliminar();
    A.imprimir();
    return 0;
}
```

```
1 -> 6 -> 7 -> 2 -> 1 ->
6 -> 7 -> 2 -> 1 ->
7 -> 2 -> 1 ->
```

3. Implemente un algoritmo para buscar elementos de la Pila.

```cpp
template <class T>
void Pila<T>::search(T val){
    // lista de posiciones
    std::vector<int> res;
    unsigned int idx=0;
    Nodo<T> * aux = first;
    for (;idx < size; idx++){
        if (aux->value == val){
            res.push_back(idx);
        }
        // recorrer al siguiente nodo
        aux = aux->sig;
    }
    if (res.empty())
        std::cout << "No se encontró el valor en la pila.\n";
    else  {
        for (unsigned int i=0; i<res.size(); i++){
            std::cout << val << " encontrado en posicion ";
            std::cout << res.at(i) << "\n";
        }
        std::cout << std::endl;
    }
}
```

```cpp
#include<iostream>
#include "Pila.h"
using namespace std;

int main (int argc, char *argv[]) {
    Pila<int> A;
    A.insertar(1);
    A.insertar(2);
    A.insertar(1);
    A.insertar(1);
    A.insertar(5);
    A.insertar(6);
    A.insertar(8);
    A.insertar(8);
    A.insertar(2);
    A.imprimir();
    A.search(1);
    A.search(2);
    return 0;
}
```

```
2 -> 8 -> 8 -> 6 -> 5 -> 1 -> 1 -> 2 -> 1 ->
1 encontrado en posicion 5
1 encontrado en posicion 6
1 encontrado en posicion 8

2 encontrado en posicion 0
2 encontrado en posicion 7
```

4. Escribir un programa que dé la solución al problema de las Torres de Hanoi para N discos, utilizando pilas, las cuales representen cada uno de los postes:

```
template <class T>
void imprimirHanoi(Pila<T> &o, Pila<T> &d, Pila<T> &a);

template <class T>
void hanoi(unsigned int size,Pila<T> &o,Pila<T> &d,Pila<T> &a);
```

definiciones de funciones amigas de Pila

```
unsigned int getSize() const {
    return size;
}
Nodo<T>* getfirst(){
    return first;
}
friend void imprimirHanoi<>(Pila<T> &o, Pila<T> &d, Pila<T> &a);
friend void hanoi<>(unsigned int size,Pila<T> &o,Pila<T> &d,Pila<T> &a);
```

funciones amigas en Clase Pila

```
template <class T>
void imprimirHanoi(Pila<T> &o, Pila<T> &d, Pila<T> &a){
    std::cout << "Estado de torres \n";
    std::cout << std::setw(20) << std::left << "Origen";
    o.imprimir();
    std::cout << std::setw(20) << std::left << "Auxiliar";
    a.imprimir();
    std::cout << std::setw(20) << std::left << "Destino";
    d.imprimir();
    std::cout<<'\n';
}

template <class T>
void hanoi(unsigned int size, Pila<T> &o, Pila<T> &d, Pila<T> &a){
    if (size == 1){
        Nodo<T> *origin_first = o.getfirst();
        T temp = origin_first->getValue();
        d.insertar(temp);
        o.eliminar();
        delete origin_first;
        return;
    }
    else {
        hanoi(size-1,o,a,d);
        Nodo<T> *origin_first = o.getfirst();
        T temp = origin_first->getValue();
        d.insertar(temp);
        o.eliminar();
        delete origin_first;
        hanoi(size-1,a,d,o);
    }

}
```

```cpp
#include<iostream>
#include<iomanip>
#include "Pila.h"
using namespace std;
int main (int argc, char *argv[]) {
    Pila<int> org; // origen
    Pila<int> aux; // auxiliar
    Pila<int> dest; // destino
    int n;
    cout<<"Ingresa el nro de discos: ";
    cin>>n;
    for (int v=1;v<=n;v++){
        org.insertar(v);
    }
    // funciona con la pila ya ordenada, con el
    // puntero first siendo el menor value de
    // la pila (torre), con recursividad
    imprimirHanoi(org,dest,aux);
    hanoi(org.getSize(),org,dest,aux);
    imprimirHanoi(org,dest,aux);
    return 0;
}
```

```
Ingresa el nro de discos: 6
Estado de torres
Origen              BaseTorre -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 ->
Auxiliar            BaseTorre ->
Destino             BaseTorre ->

Estado de torres
Origen              BaseTorre ->
Auxiliar            BaseTorre ->
Destino             BaseTorre -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 ->
```

```
Ingresa el nro de discos: 4
Estado de torres
Origen              BaseTorre -> 4 -> 3 -> 2 -> 1 ->
Auxiliar            BaseTorre ->
Destino             BaseTorre ->

Estado de torres
Origen              BaseTorre ->
Auxiliar            BaseTorre ->
Destino             BaseTorre -> 4 -> 3 -> 2 -> 1 ->
```