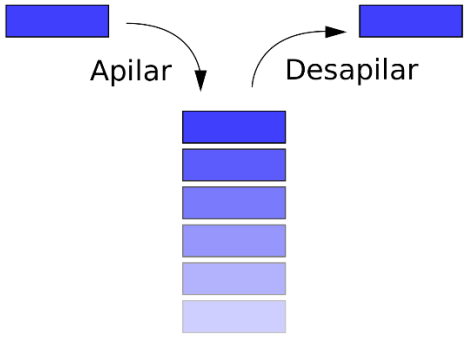
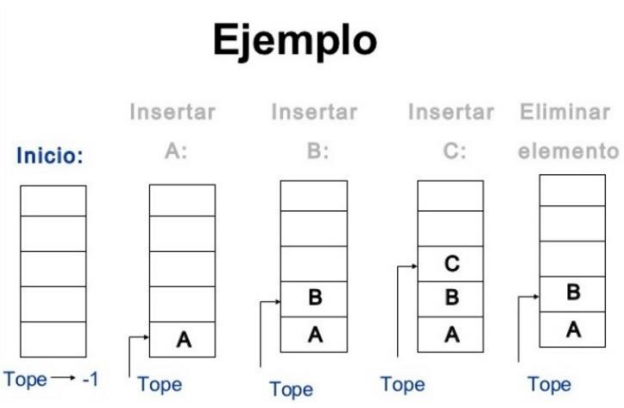
Nava Reyes Carlos – 17212163 TAREA #2 UNIDAD-2

Pilas

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. Se aplica en multitud de ocasiones en informática debido a su simplicidad y ordenación implícita en la propia estructura.



La administración de una pila se puede hacer con muy pocas operaciones: una constructora (permite crear pilas vacías), dos modificadoras (para agregar y eliminar elementos) y dos analizadoras (retornar el elemento del tope, e informar si la pila está vacía). Se incluye, además, una destructora para retornar el espacio ocupado por la pila. Por simplicidad, no se contemplan operaciones de persistencia. El formalismo escogido para referirse al objeto abstracto pila se muestra a continuación. Allí se da un nombre a cada uno de los elementos que hacen parte de la estructura y se marca claramente el tope y el fondo.



Si la pila no tiene ningún elemento se dice que se encuentra vacía y no tiene sentido referirse a su tope ni a su fondo. Una pila vacía se representa con el símbolo Ø. Por último, se define

la longitud de una pila como el número de elementos que la conforman. Una pila tiene las siguientes operaciones básicas:

1. **Crear**: se crea la pila vacía
2. **Apilar**: (push), que coloca un elemento en la pila
3. **Retirar** (o desapilar, pop), es la operación inversa que retira el último elemento apilado
4. **Cima**: devuelve el elemento que está en la cima de la pila (top o peek).
5. **Vacía:** devuelve true si la pila está vacía o falso en caso contrario

La implementación en lenguaje C++ sería de la siguiente forma:

```
// pila.h
template <typename T>
class Pila
{
    class Nodo
    {
        private: T valor;
        Nodo *pSiguiente;
        public:
          Nodo(T _valor, Nodo *pSiguiente = NULL){
          {
            setValor(_valor);
            setSiguiente(_pSiguiente);
          }
          T getValor() {        return valor;        }
          void setValor(T &_valor) { valor = _valor;                }
          Nodo *getSiguiente() {        return pSiguiente;        }
          void setSiguiente(Nodo *_pSiguiente) { pSiguiente = _pSiguiente;
                    }
        };

        private:
          Nodo *pCabeza;
          int tamanio;
        public:
          Pila(): pCabeza(NULL), tamanio(0)        {        }
          int getTamanio() {        return tamanio;                }
          boolean esVacia(){        return (pCabeza == NULL);        }
```

```
        void apilar(T valor)
        {
          new Nodo(valor, pCabeza);
          tamanio++;
        }
        void retirar()
        {
          if (esVacia())
              throw Pila vac a ;
          Nodo *pNodo = pCabeza;
          pCabeza = pCabeza->getSiguiente();
          delete pNodo;
          tamanio--;
        }

        T cima()
        {
           if (esVacia())
               return null;
            return pCabeza->getValor();
        }
    };
```

```
// main.cpp
#include <iostream.h>
#include pila . h
void main()
{
    Pila<int> pila;
    pila2.apilar(2);
    pila2.apilar(5);
    pila2.apilar(7);
    cout << pila2.cima();
    pila2.retirar();
    cout << pila2.cima();
    pila2.retirar();
    cout << pila2.cima();
    pila2.retirar();
    cout << pila2.cima();

    //Probar con otra pila, donde se almacenen objetos tipo Persona o Contacto o Libro, etc.
    //Algo así;
    //Pila <Contacto *>pila3;
    //pila3.apilar(new Contacto(2,"Juan Perez", "31245434","juanito@hotmail.com");
    //pila3.retirar();
    //...
}
```

Aguilar, Luis, and Ignacio Zahonero. *Algoritmos y estructuras de datos: una perspectiva en C*. Madrid: McGraw-Hill, Interamericana de España, 2004. Print.