



Algoritmos y Estructura de datos

Clase 09 Pilas

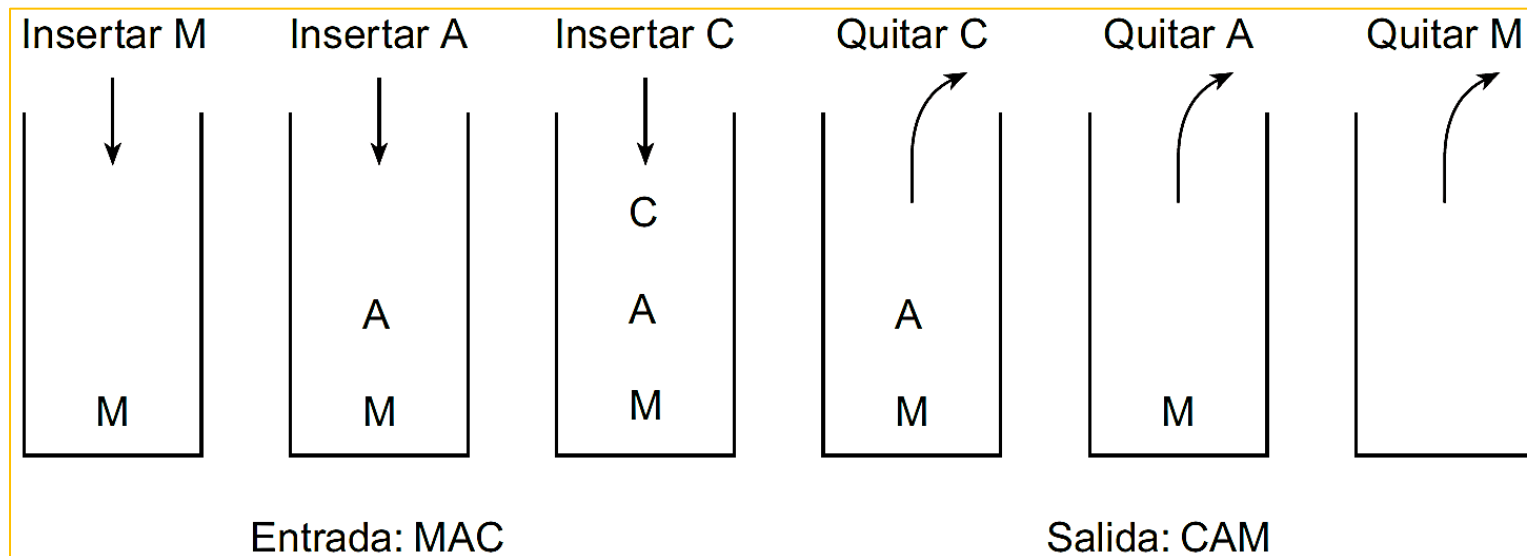
Profesor: Carlos Diaz

Contenido

- Concepto de Pila
- Implementación de una Pila
- Operaciones en una Pila
- Implementando la Pila con arrays
- Declaración de la clase Pila
- Implementación las funciones de una pila
- Implementando la pila con listas enlazadas
- Ejercicios

Concepto de Pila

- Una pila es una estructura de datos tal que sólo se puede introducir o eliminar elementos por un extremo (llamado **cima**).
- Las pilas se conocen también como estructuras LIFO (Last-In, First-Out, último en entrar primero en salir).
- Las operaciones usuales en la pila son **Insertar** (**push**) y **Quitar** (**pop**). La operación Insertar añade un elemento en la cima de la pila y la operación Quitar elimina o saca un elemento de la cima de la pila. Es la única forma de acceder a la pila.



Implementación de una Pila

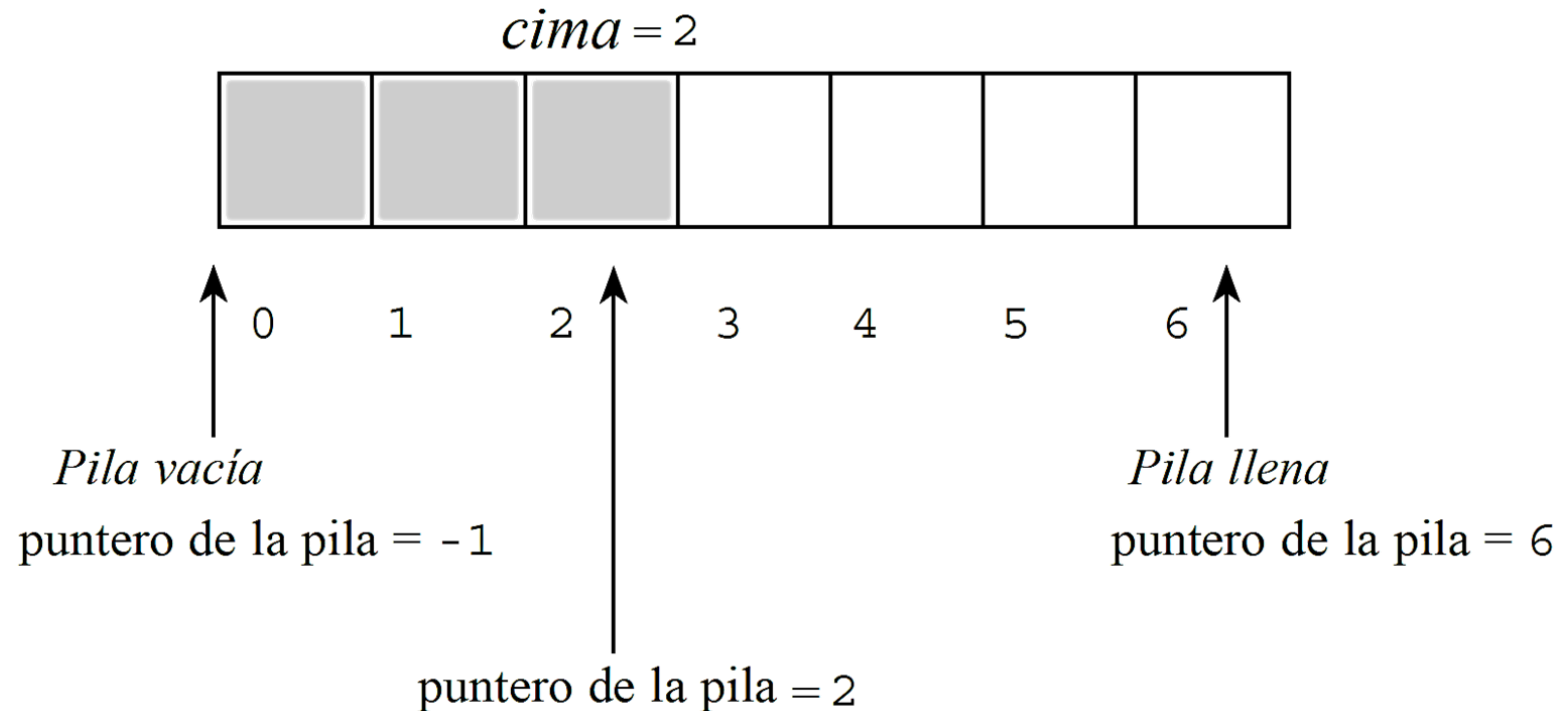
- La pila se puede implementar guardando los elementos en un array, en cuyo caso su dimensión o longitud es fija.
- Otra forma de implementación consiste en construir una lista enlazada, cada elemento de la pila forma un nodo de la lista; la lista crece o decrece según se añaden o se extraen elementos de la pila; ésta es una representación dinámica y no existe limitación en su tamaño excepto la memoria de la computadora.
- Una pila puede estar *vacía* (sin elementos) o *llena* (en la representación con un array, si se ha llegado al último elemento; en la representación de lista, si se llenó la memoria).
- Si un programa intenta sacar un elemento de una pila vacía, se producirá un error; esta situación se denomina **desbordamiento negativo** (*underflow*).
- Por el contrario, si un programa intenta poner un elemento en una pila *llena* se produce un error de **desbordamiento** (*overflow*).
- Para evitar estas situaciones se diseñan funciones, que comprueban si la pila está llena o vacía.

Operaciones en una Pila

- Las operaciones que sirven para definir una pila y poder manipular su contenido son las siguientes.
 - **CrearPila**: Inicia la pila.
 - **Insertar** (push): Pone un dato en la pila.
 - **Quitar** (pop): Retira (saca) un dato de la pila.
 - **Pilavacía**: Comprobar si la pila no tiene elementos.
 - **Pilallena**: Comprobar si la pila está llena de elementos.
 - **LimpiarPila**: Quita todos sus elementos y dejar la pila vacía.
 - **CimaPila**: Obtiene el elemento cima de la pila.
 - **MostrarCima**: Devuelve el valor de la variable cima.
 - **Tamaño de la pila**: Número de elementos máximo que puede contener la pila.
- Nota**: La operación **Pilallena** sólo se implementa cuando se utiliza un array para almacenar los elementos. Una pila puede crecer indefinidamente si se implementa con una estructura dinámica, por ejemplo una lista simplemente enlazada.

Implementando la Pila con arrays

- El rango de elementos que puede tener una pila varía de 0 a TAMPILA-1, donde TAMPILA es la cantidad de elementos. De modo que *en una pila llena* el apuntador (índice del array) de la pila tiene el valor TAMPILA-1, y *en una pila vacía* tendrá -1 (el valor 0 es el índice del primer elemento).
- Ejemplo:** Representación gráfica de una pila de 7 elementos. TAMPILA=7.



Declaración de la clase Pila

```
typedef int TipoDato; // tipo de los elementos de la pila
const int TAMPILA = 6; //en este caso int, para otro tipo de dato, solo cámbielo
#include <iostream>
using namespace std;
class Pila
{
    private:
        int cima; //Es la índice del último elemento ingresado en la pila
        TipoDato listaPila[TAMPILA]; //El tamaño de la lista esta definido al inicio
    public:
        Pila() //Constructor que crea la pila
        {
            cima = -1; // Condición de pila vacía
        }
        void insertar(TipoDato); //push
        TipoDato quitar(); //pop
        bool pilaVacía(); //Comprueba si la pila esta vacía
        bool pilaLlena(); //Comprueba si la pila esta llena
        void limpiarPila(); //Quita todos los elementos de la pila
        TipoDato cimaPila(); //Obtiene elemento cima de la pila
        int mostrarCima(); //Obtiene elemento cima de la pila
        int tamanoPila(); //Número de elementos de la pila
};
```

La función insertar()

- En primer lugar la función `insertar()` verifica si hay espacio en la pila llamando a la función `pilaLlena()`.
- La pila es un array de tamaño `TAMPILA`, los índices van desde `0` hasta `TAMPILA-1`. Por tanto, la pila esta llena cuando la `cima` es `TAMPILA-1`.
- Si hay espacio, la función `insertar()` incrementa el atributo `cima` en `1` y coloca el elemento en el array.
- Si no hay espacio muestra un mensaje de desbordamiento.

```
bool Pila::pilaLlena()
{
    return cima == TAMPILA - 1;
}
```

```
void Pila::insertar(TipoDato elemento)
{
    if (pilaLlena())
    {
        cout<<"Desbordamiento (overflow)\n";
    }
    else{
        cima++; //incrementar puntero cima y copia elemento
        listaPila[cima] = elemento;
    }
}
```


La función quitar()

- En primer lugar la función `quitar()` verifica si la pila está vacía llamando a la función `pilaVacía()`.
- La pila esta vacía cuando el array no contiene elementos, es decir que `cima` es `-1`.
- Si no esta vacía, la función `quitar()` obtiene el elemento de la `cima` del array y **disminuye** en **1** el valor de la `cima`. Luego, devuelve el elemento obtenido
- Si la pila esta vacía muestra un mensaje de desbordamiento negativo.

```
bool Pila::pilaVacía()  
{  
    return cima == -1;  
}
```

```
TipoDato Pila::quitar()  
{  
    TipoDato x;  
    if (pilaVacía())  
    {  
        cout<<"Desbordamiento negativo (underflow)\n";  
        return NULL;  
    }  
    else  
    {  
        x = listaPila[cima]; // obtiene el elemento de la cima  
        cima--; // decrementa cima  
        return x; //devuelve elemento quitado  
    }  
}
```

Las función mostrarPila()

- La función `mostrarPila()` llama a la función `quitar()` para mostrar los elementos de la pila, el inconveniente es que también los “`retira`” de la pila.
- La función `mostrarCima()` devuelve el valor de la `cima`. La `cima` va desde `-1` (Sin elementos) hasta `TAMPILA-1` (Pila llena)
- Adicionalmente también muestra como se mueve la variable `cima` mediante la función `mostrarCima()`.

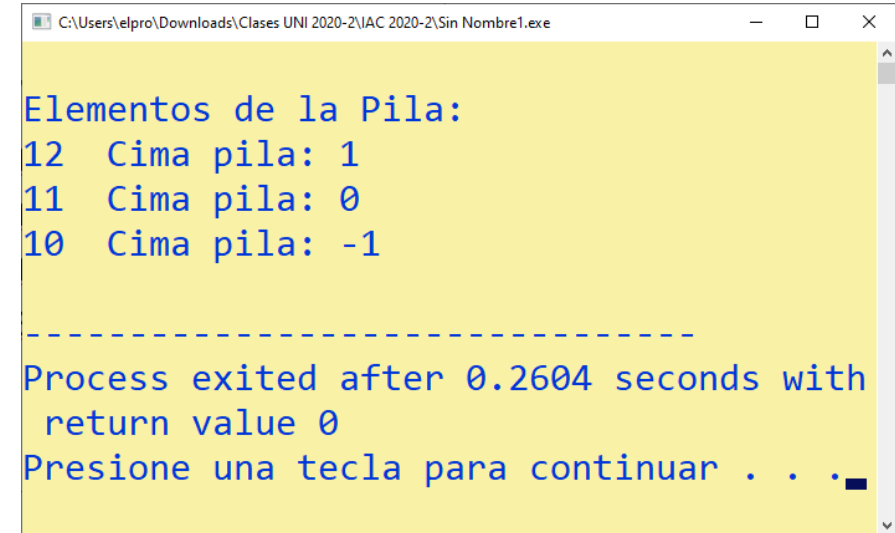
```
int Pila::mostrarCima()
{
    return cima;
}
```

```
void mostrarPila(Pila &pila)
{
    TipoDato x;
    cout << "\nElementos de la Pila:"<<endl;
    while (!pila.pilaVacía())
    {
        x = pila.quitar();
        cout << x << " ";
        cout<<"Cima pila: "<<pila.mostrarCima()<<endl;
    }
}
```

Probando el código

```
int main()
{
    Pila pila; //Crea una pila vacía
    pila.insertar(10);
    pila.insertar(11);
    pila.insertar(12);
    mostrarPila(pila); //Muestra los elementos de la pila
    return 0;
}
```

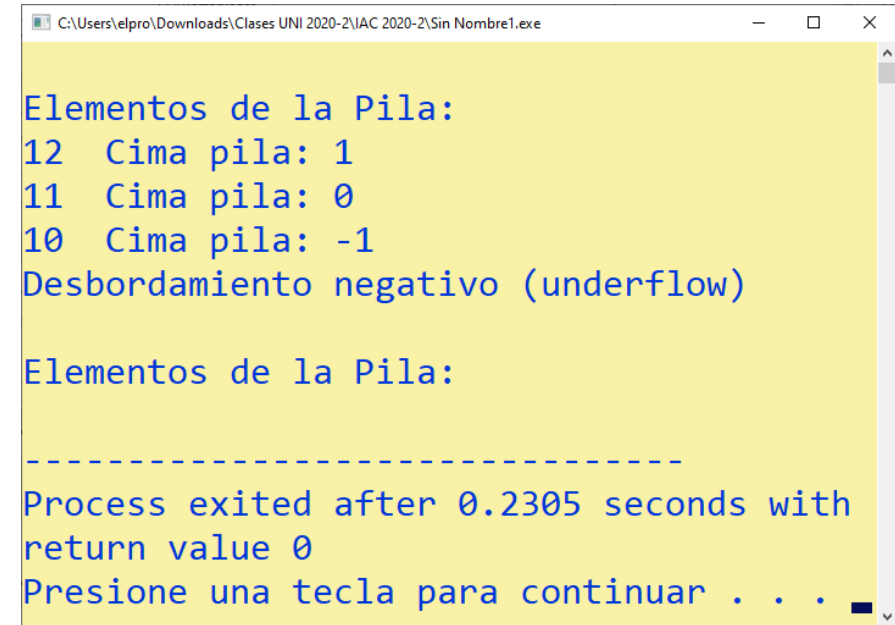
```
int main()
{
    Pila pila; //Crea una pila vacía
    pila.insertar(10);
    pila.insertar(11);
    pila.insertar(12);
    mostrarPila(pila); //Muestra los elementos de la pila
    pila.quitar(); //Trata de quitar pero la pila está vacía
    mostrarPila(pila); //La pila está vacía
    return 0;
}
```



```
C:\Users\elpro\Downloads\Clases UNI 2020-2\IAC 2020-2\Sin Nombre1.exe

Elementos de la Pila:
12 Cima pila: 1
11 Cima pila: 0
10 Cima pila: -1

-----
Process exited after 0.2604 seconds with
return value 0
Presione una tecla para continuar . . .
```



```
C:\Users\elpro\Downloads\Clases UNI 2020-2\IAC 2020-2\Sin Nombre1.exe

Elementos de la Pila:
12 Cima pila: 1
11 Cima pila: 0
10 Cima pila: -1
Desbordamiento negativo (underflow)

Elementos de la Pila:

-----
Process exited after 0.2305 seconds with
return value 0
Presione una tecla para continuar . . .
```

Las función completarPila()

- La función `completarPila()` llama a la función `insertar()` para insertar elementos a la pila.
- Opcionalmente se muestra como se mueve la variable `cima`, esta variable indica la posición (`índice`) del elemento insertado.

```
void completarPila(Pila &pila)
{
    TipoDato x;
    int n, i;
    cout<<"Cuantos elementos ingresara a la pila (max. 6): ";
    cin>>n;
    cout<<"Digite el elemento y luego pulse Enter\n";
    for (i = 0; i < n ; i++)
    {
        cout<<"índice "<<i<<": ";
        cin >> x;
        pila.insertar(x);
        cout<<"Cima pila: "<<pila.mostrarCima()<<endl;
    }
}
```

Creación de una pila

- Para crear la pila, el usuario debe usar las funciones definidas en la clase Pila.
- En la función main() creamos un objeto **pila** y llámanos a dos funciones:
- **completarPila()**: Se encargará de insertar elementos a la pila.
- **mostrarPila()**: Se encargará de mostrar los elementos de la pila.

```
int main()
{
    Pila pila; //Crea una pila vacía
    completarPila(pila);
    mostrarPila(pila);
    return 0;
}
```

Funciones tamañoPila(), cimaPila() y limpiarPila()

- La función **tamañoPila()** devuelve el tamaño fijo del array, que en este caso representa la pila.
- La función **cimaPila()** devuelve el elemento que esta en la cima de la pila. Si la pila esta vacía muestra un mensaje.
- La función **limpiarPila()** reinicia la **cima** al valor **-1**. Los elementos aun están en el array.

```
int Pila::tamañoPila()
{
    return TAMPILA;
}
```

```
TipoDato Pila::cimaPila()
{
    if (pilaVacía())
    {
        cout<<"Pila vacía, no hay elementos\n";
        return NULL;
    }
    else
        return listaPila[cima];
}
```

```
void Pila::limpiarPila()
{
    cima = -1;
}
```

Ejercicio 1

- Escriba en el `main()` las instrucciones necesarias, que utilice la pila que hemos programado, para que compruebe si una palabra o frase es un palíndromo.

Ejemplos de palabras o frase palíndromas:

- Arenera
- Rallar
- Rotomotor
- Anita lava la tina
- Eva usaba rimel y le miraba suave
- Luz azul

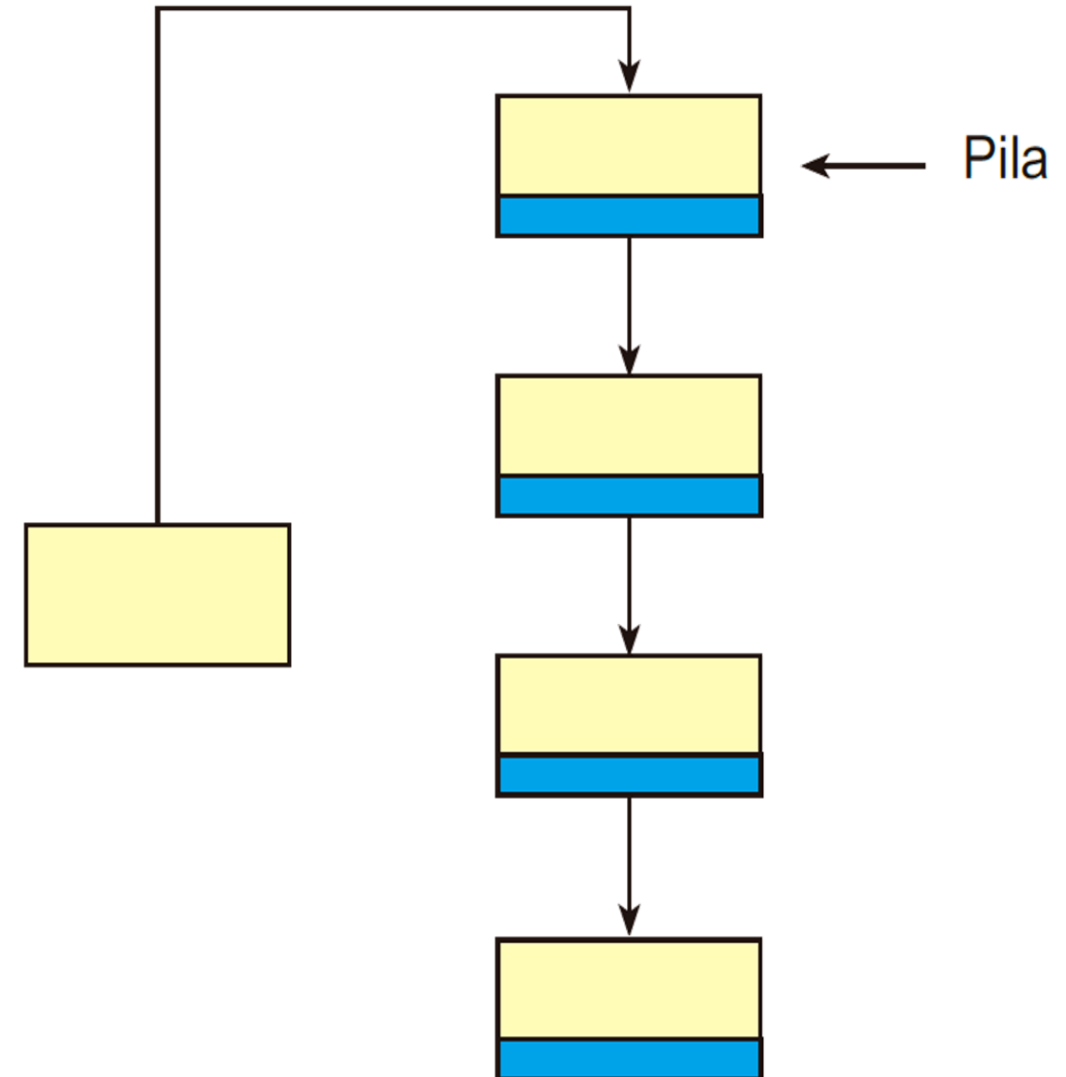
Solucion 1

```
#include <iostream>
using namespace std;
#include <string>
int main()
{
    Pila pilaChar; // crea pila vacía
    TipoDato ch; //Cantidad de caracteres de la palabra
    bool esPal; //Devuelve true si es palíndromo
    string palabra;
    string palabraSE;
    cout << "Escriba la palabra verificar si es palindromo:\n";
    getline(cin,palabra);
    for (int i = 0; i < palabra.length();i++)
    {
        char c;
        c = toupper(palabra[i]);
        if (c!=' ')
        {
            pilaChar.insertar(c);
            palabraSE+=c;
        }
    }
}
```

```
// se comprueba si es palíndromo
esPal = true;
for (int i = 0; esPal && !pilaChar.pilaVacia(); i++)
{
    char c;
    c = pilaChar.quitar();
    esPal = palabraSE[i] == c;
}
if (esPal)
    cout << "La palabra " << palabra << "\nes un palindromo \n";
else
    cout << "La palabra " << palabra << "\nno es un palindromo \n";
return 0;
}
```


Representación dinámica de una Pila

- La realización **dinámica** de una pila se logra con una **lista enlazada**, almacenando cada elemento de la pila como un nodo de la lista.
- Como las operaciones de insertar y extraer en la pila se realizan por el mismo extremo (**cima de la pila**), las acciones correspondientes con la lista se realizarán siempre por el mismo extremo de la lista, generalmente por la **cabeza**.
- Esta realización tiene la ventaja de que el tamaño se ajusta exactamente al número de elementos de la pila.
- En tiempo de ejecución, se reserva memoria según se ponen elementos a la pila y se libera memoria según se extraen elementos de la pila.



La clase NodoPila

- La clase **NodoPila** representa un nodo de la lista enlazada (**Pila**), tiene dos atributos:
 - **elemento**: guarda el elemento del nodo.
 - **siguiente**: contiene la dirección del siguiente nodo de la lista (**Pila**).
- Inicialmente el nodo apunta a **NULL**, pero luego apuntará a la **cima** y se convertirá en la nueva cima.

```
class NodoPila
{
    public:
    int elemento;
    NodoPila* siguiente;
    NodoPila(int x) //Constructor
    {
        elemento = x;
        siguiente = NULL;
    }
};
```

La clase PilaDina

- El constructor inicializa la cima a NULL (pila vacía).
- Contiene funciones que permitirán administrar la pila: insertar(), quitar(), cimaPila(), pilaVacía() y limpiarPila().

```
class PilaDina
{
    public:
        NodoPila* cima;
        PilaDina () //Constructor
        {
            cima = NULL;
        }
        void insertar(int);
        int quitar();
        int cimaPila();
        bool pilaVacía();
        void limpiarPila();
};
```

La función insertar()

- En primer lugar la función **insertar()** crea un nodo y le asigna el elemento.
- Luego hace que este nuevo elemento apunte a la **cima**. Al iniciar la pila, esta vacía y **cima=NULL**.
- Finalmente hace que el **nuevo** elemento sea la nueva cima.

```
void PilaDina::insertar(int elemento)
{
    NodoPila* nuevo;
    nuevo = new NodoPila(elemento);
    nuevo -> siguiente = cima;
    cima = nuevo;
}
```

La función mostrarPila()

- La función `mostrarPila()` primero llama `pilaVacia()` para verificar si hay elementos en la pila.
- Si hay elementos llama la función `quitar()` para mostrar los elementos de la pila, el inconveniente es que también los “`retira`” de la pila.
- También muestra como se mueve la variable `cima` y el elemento de la cima.

```
bool PilaDina::pilaVacia()
{
    return cima == NULL;
}
```

```
int PilaDina::cimaPila()
{
    if (pilaVacia())
        cout<<"Pila vacia\n";
    else
        return cima->elemento;
}
```

```
void mostrarPila(PilaDina &pila)
{
    int x;
    cout <<"\nElementos de la Pila:"<<endl;
    while (!pila.pilaVacia())
    {
        cout<<"Direccion cima: "<<pila.cima<<" . Elemento en la cima: "<<pila.cimaPila()<<endl;
        x = pila.quitar();
        cout << x << endl;
    }
    cout<<"Al terminar queda:\n "
    cout<<"Direccion cima: "<<pila.cima<<" . Elemento en la cima: "<<pila.cimaPila()<<endl;
}
```

La función quitar()

- En primer lugar la función `quitar()` verifica si la pila está vacía llamando a la función `pilaVacía()`.
- Si la pila esta vacía muestra un mensaje.
- Si no está vacía, la función `quitar()` obtiene el elemento de la cima.
- Luego procede a cambiar la cima por el segundo elemento de la pila, mediante `cima=(cima->siguiente)`.
- Finalmente elimina el nodo de la memoria

```
int PilaDina::quitar()
{
    NodoPila* n;
    if (pilaVacía())
    {
        cout<<"Pila vacia, no se puede extraer.\n";
    }
    else
    {
        n = cima;
        int x = cima -> elemento;
        cima = cima -> siguiente;
        delete n;
        return x;
    }
}
```

Probando el código

```
int main()
{
    PilaDina pila;
    pila.insertar(7);
    pila.insertar(6);
    pila.insertar(5);
    pila.insertar(4);
    pila.insertar(3);
    mostrarPila(pila);
    return 0;
}
```

C:\Users\elpro\Downloads\Clases UNI 2020-2\IAC 2020-2\Sin Nombre1.exe

Elementos de la Pila:

Direccion cima: 0xb61a70 . Elemento en la cima: 3
3

Direccion cima: 0xb61a50 . Elemento en la cima: 4
4

Direccion cima: 0xb61590 . Elemento en la cima: 5
5

Direccion cima: 0xb61570 . Elemento en la cima: 6
6

Direccion cima: 0xb61550 . Elemento en la cima: 7
7

Al terminar:

Pila vacia

Direccion cima: 0 . Elemento en la cima: 4745728

Process exited after 0.13 seconds with return value 0

Presione una tecla para continuar . . . █

La función completarPila()

- La función `completarPila()` llama a la función `insertar()` para insertar elementos a la pila. Opcionalmente se muestra como se mueve la variable `cima` y el elemento que esta en la cima.

```
void completarPila(PilaDina &pila)
{
    int x, n;
    cout<<"Cuantos elementos ingresara a la pila: ";
    cin>>n;
    cout<<"Digite el elemento y luego pulse Enter\n";
    for (int i = 0; i < n ; i++)
    {
        cout<<"Direccion cima: "<<pila.cima<<" . Elemento en la cima: "<<pila.cimaPila()<<endl;
        cout<<"indice "<<i<<": ";
        cin >> x;
        pila.insertar(x);
    }
    cout<<"Direccion cima: "<<pila.cima<<" . Elemento en la cima: "<<pila.cimaPila()<<endl;
}
```


Creación de una pila

- Para crear la pila, el usuario debe usar las funciones definidas en la clase **PilaDina**.
- En la función `main()` creamos un objeto **pila** y llámanos a dos funciones:
- **`completarPila()`**: Se encargará de insertar elementos a la pila.
- **`mostrarPila()`**: Se encargará de mostrar los elementos de la pila.

```
int main()
{
    PilaDina pila;
    completarPila(pila);
    mostrarPila(pila);
    return 0;
}
```

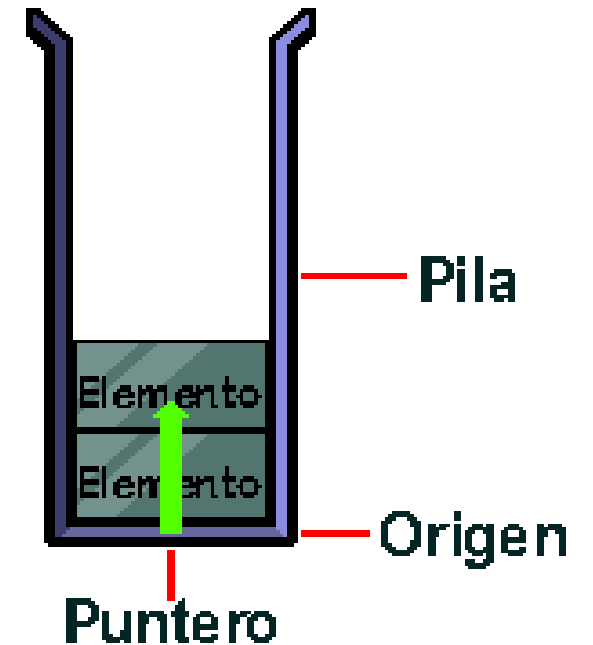
La función limpiarPila()

- La función `limpiarPila()` libera la memoria de los nodos.
- Antes de borrar verifica si hay elementos en la pila.
- Si hay elementos, mueve la cima al nodo siguiente y luego elimina el nodo de la memoria.

```
void PilaDina::limpiarPila()
{
    NodoPila* n;
    cout<<"Limpiar Pila: "<<endl;
    while(!pilaVacía())
    {
        cout<<"Direccion cima: "<<cima<<" . Elemento en la cima: "<<cimaPila()<<endl;
        n = cima;
        cima = cima -> siguiente;
        delete n;
    }
}
```

Ejercicio 2

- Utilizando la implementación dinámica de una pila, programe en un formulario que muestre visualmente (animación) las operaciones insertar, quitar, limpiar, etc.



Ejercicio 3

- Las operaciones en la pila se realizan por el mismo extremo de la lista enlazada (**cima de la pila**), en esta clase se realizó por la **cabeza**.
- Ahora usted implementar una **pila** pero utilizando la **cola** de la lista enlazada, es decir que la cima siempre es la cola.

FIN