



CICLO 1

[FORMACIÓN POR CICLOS]

Fundamentos de **PROGRAMACIÓN**



Ingeni@
Soluciones TIC



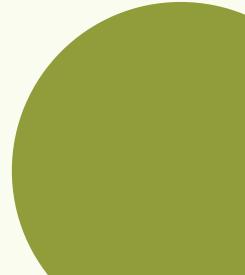
UNIVERSIDAD
DE ANTIOQUIA

Facultad de Ingeniería



Lectura

PILAS



Introducción

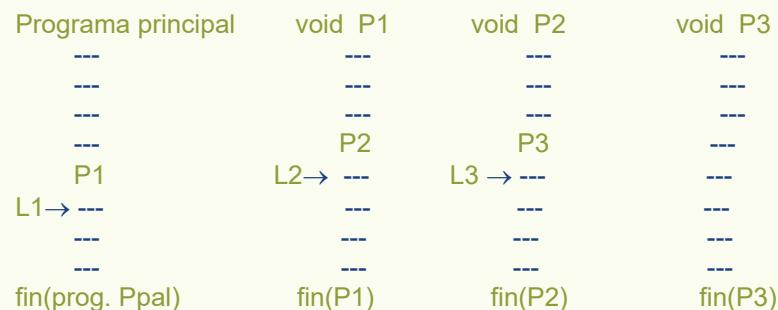
Hemos tratado hasta ahora estructuras que permiten almacenar colecciones de datos: arreglos y listas ligadas. En esto de almacenar colecciones de datos y efectuar operaciones de búsqueda, inserción y borrado, hay situaciones en las cuales se requiere guardar datos y recuperarlos en orden inverso al que fueron guardados. En este módulo veremos la estructura pila, que permite procesar los datos de esta manera.

Definición

Una pila es una lista ordenada en la cual todas las operaciones (inserción y borrado) se efectúan en un solo extremo llamado **tope**. Es una estructura **LIFO** (Last Input First Output), que son las iniciales de las palabras en inglés “último en entrar primero en salir”, debido a que los datos almacenados en ella se retiran en orden inverso al que fueron entrados.

Un ejemplo clásico de aplicación de pilas en computadores se presenta en el proceso de llamadas a subprogramas y sus retornos.

Supongamos que tenemos un programa principal y tres subprogramas, así:



Cuando se ejecuta el programa principal, se hace una llamada al subprograma P1, es decir, ocurre una interrupción a la ejecución del programa principal. Antes de iniciar la ejecución de este subprograma, se guarda



la dirección de la instrucción donde debe retornar a continuar la ejecución del programa principal cuando termine de ejecutar el subprograma. Llamemos L1 esta dirección. Cuando ejecuta el subprograma P1 existe una llamada al subprograma P2, hay una nueva interrupción, pero antes de ejecutar el subprograma P2 se guarda la dirección de la instrucción donde debe retornar a continuar la ejecución del subprograma P1, cuando termine de ejecutar el subprograma P2. Llamemos L2 a esta dirección.

Hasta el momento hay guardados dos direcciones de retorno:

L1, L2

Cuando ejecuta el subprograma P2 hay llamada a un subprograma P3, lo cual implica una nueva interrupción y, por ende, guardar una dirección de retorno al subprograma P2, que llamamos L3.

Tenemos entonces tres direcciones guardadas así:

L1, L2, L3

Al terminar la ejecución del subprograma P3, retorna a continuar ejecutando en la última dirección que guardó, es decir, extrae la dirección L3 y regresa a continuar ejecutando el subprograma P2 en dicha instrucción. Los datos guardados ya son:

L1, L2

Al terminar el subprograma P2, extrae la última dirección que tiene guardada, en este caso L2, y retorna a esta dirección a continuar la ejecución del subprograma P1.

En este momento los datos guardados son:

L1

Al terminar la ejecución del subprograma P1 retornará a la dirección que tiene guardada, o sea a L1.

Obsérvese que los datos fueron procesados en orden inverso al que fueron almacenados, es decir, último en entrar primero en salir. Es esta forma de procesamiento la que define una estructura PILA. Veamos cuáles son las operaciones que definiremos para la clase pila.

- **crear:** crea una pila vacía.
- **apilar:** incluye el dato d en la pila.
- **desapilar:** elimina el último elemento de la pila y deja una nueva pila la cual queda con un elemento menos.
- **cima:** retorna el dato que está de último en la pila, sin eliminarlo.
- **esVacia:** retorna verdadero si la pila está vacía, falso de lo contrario.
- **esLlena:** retorna verdadero si la pila está llena, falso de lo contrario.

Representación de pilas en un vector.

La forma más simple es definir una clase pila derivada de la clase vector. Definamos entonces nuestra clase pila:

```
class pila(vector):
    def __init__(self, n):
        vector.__init__(self, n)

    def apilar(self, d):
        self.agregarDato(d)

    def muestraPila(self):
        self.imprimeVector()

    def desapilar(self):
        if self.V[0] == 0:
            print("Pila vacía")
            return None
        d = self.V[self.V[0]]
        self.V[0] = self.V[0] - 1
        return d
```

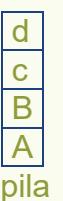
y un programa de ejemplo de uso de esta clase es:

```
from claseVector import pila
st = pila(10)
st.apilar("a")
st.apilar("b")
st.apilar(316)
st.muestraPila()
d = st.desapilar()
print(d)
```

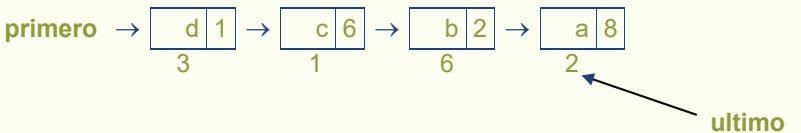
Se recomienda al estudiante comprobar el funcionamiento de este algoritmo y notar bien el uso de la herencia en programación orientada a objetos.

Representación de pilas como lista ligada.

Para representar una pila como lista ligada basta definir la clase pila derivada de la clase **LSL**. Nuevamente, como estamos definiendo la clase pila derivada de la clase lista simplemente ligada, podremos utilizar todos los métodos que hemos definido para la clase **LSL**. A modo de ejemplo, representemos como lista ligada la siguiente pila:



Dibujémosla de la forma como solemos hacerlo con listas ligadas:





Como hemos dicho, las operaciones sobre una pila son apilar y desapilar.

- **apilar:** consiste en insertar un registro al principio de una lista ligada.
- **desapilar:** consiste en eliminar el primer nodo de la lista ligada y retornar el dato que se hallaba en ese nodo.

Al definir la clase pila derivada de la clase **LSL**, el método para controlar pila llena ya no aplica. Los métodos para la clase pila son:

```
class pila(LSL):
    def __init__(self):
        LSL.__init__(self)

    def apilar(self, d):
        self.insertar(d)

    def muestraPila(self):
        self.recorrerLista()

    def desapilar(self):
        p = self.primerNodo()
        d = p.retornarDato()
        self.borrar(p)
        return d
```

Fíjese que en los métodos apilar y desapilar hemos usado los métodos insertar y borrar, los cuales fueron definidos para la clase **LSL**.

A continuación, presentamos un algoritmo usando la clase pila definida como derivada de la clase **LSL**.



Tenga en cuenta que en este ejemplo en la instrucción **from** (la primera instrucción) **nodoSimple** es el nombre del archivo donde está definida la clase **pila**:

```
from nodoSimple import pila
a = pila()
a.apilar("a")
a.apilar("e")
a.apilar("i")
a.apilar("o")
a.recorrerLista()
b = a.esVacia()
print("\n", b)
d = a.desapilar()
print("\ndato desapilado", d)
a.recorrerLista()
```

Nuevamente se recomienda al estudiante notar lo que es la herencia y el polimorfismo en dicho algoritmo.

Fíjese que **recorrerLista()** es un método de la clase base, la clase **LSL**, y lo estamos usando con el objeto **a**, el cual es un objeto de la clase **pila**. En otras palabras, los objetos de las clases **derivadas** pueden utilizar los métodos de la clase **base**. Eso en virtud de la herencia.