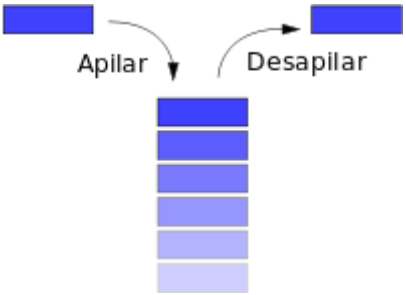
¿Qué es una pila?



Una pila (*stack* en inglés) es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés *LastIn FirstOut*, último en *entrar*, primero en *salir*) que permite almacenar y recuperar datos. Esta estructura se aplica en multitud de ocasiones en el área de informática debido a su simplicidad y ordenación implícita de la propia estructura.

Para el manejo de los datos se cuenta con dos operaciones básicas:

- **apilar** (*push*), que coloca un objeto en la pila.
- **desapilar** (*pop*), que retira el último elemento apilado.

En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado **tope de pila**). La operación **desapilar** permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo **tope de pila**.

Por analogía con objetos cotidianos, una operación **apilar** equivaldría a colocar un plato sobre una pila de platos, y una operación **retirar** a retirarlo.

Una metáfora que se utiliza con frecuencia es la idea de una pila de platos. En esa pila, sólo el primer plato es visible y accesible para el usuario, todos los demás platos permanecen ocultos. A medida que se añaden nuevos platos, cada nuevo plato se convierte en la parte superior de la pila. A medida que el plato superior se elimina de la pila, el segundo plato se convierte en la parte superior de la pila. Dos principios importantes son ilustrados por esta metáfora: En primer lugar la última salida es un principio, la segunda es que el contenido de la pila está oculto. Sólo el plato de la parte superior es visible, por lo que para ver lo que hay en el tercer plato, el primer y segundo plato tendrán que ser retirados.

Operaciones

Una pila cuenta con 2 operaciones imprescindibles: apilar y desapilar, a las que en las implementaciones modernas de las pilas se suelen añadir más de uso habitual.

- **Crear:** se crea la pila. (constructor)
- **Inicializar:** se deja la pila vacía, lista para ser utilizada.
- **Apilar:** se añade un elemento a la pila.
- **Leer:** agrega al tope de la pila un elemento ingresado desde el teclado por el usuario.
- **Desapilar:** se elimina el elemento superior de la pila.
- **Tope:** devuelve el elemento que está en la cima de la pila. No lo elimina.
- **PilaVacía:** devuelve cierto si la pila está vacía o falso en caso contrario.
- **Mostrar:** muestra por pantalla el contenido de la pila.

Empezando a programar

A continuación vamos a ver el código de las operaciones descriptas con anterioridad:

- **Crear una pila:**

Básicamente deben realizarse dos acciones, primero crear la pila y posteriormente inicializarla.

```
Pila idPila;
```

Donde pila es el tipo de dato

Todas las instrucciones que manipulan las pilas reciben como parámetro (variables y constantes que se escriben entre paréntesis) entre otros al identificador de la variable precedido por el operador & (ampersand), que permite que la variable sea modificada por la función.

- **Inicializar una pila:**

```
inicpila(&idPila);
```

Inicializa a la pila con cero elementos. Lo que se pone entre paréntesis es la dirección de memoria de la variable de tipo Pila que se quiere inicializar. Es necesario agregar el modificador & para que la instrucción cause efecto sobre la pila (esto se explicará más adelante en la materia).

Teniendo en cuenta que en un mismo programa podemos disponer de múltiples pilas, debemos darle un nombre particular a cada una, a efectos de poder identificarlas.

Existen dos maneras de apilar elementos en una pila:

- **Apilar:**

Apilar por código: el valor es ingresado por el sistema.

```
apilar(&idPila, <número>);
```

Pone el elemento <número> al tope de la pila.

Si se llama a apilar con una pila llena se producirá un error y se abortará el programa. De todas formas no nos ocuparemos por ahora de esa posibilidad.

El valor <número> puede ser un valor constante expresado directamente en la instrucción o el resultado de la función desapilar: por ejemplo

```
apilar(&miPila, 4);
apilar(&miPila, desapilar(&tuPila));
```

En el primer ejemplo se agrega la constante numérica 4 a la pila llamada miPila, en el segundo ejemplo se agrega a la pila miPila el valor que está en el tope de la pila tuPila (además ese valor se retira de la pila tuPila).

- **Leer:**

Apilar por teclado: el valor es ingresado por el usuario.

```
leer(&idPila);
```

Agrega al tope de la pila un elemento ingresado desde el teclado por el usuario. La ejecución del programa se suspende hasta que el usuario ingresa un valor y presiona luego ENTER.

- **Desapilar:**

```
desapilar(&idPila); // → <número>
```

Devuelve el elemento que está en el tope de la pila y lo elimina de la misma.

Luego de llamar a desapilar, el elemento que estaba después del elemento extraído pasará al tope o la pila se vaciará si éste era el último.

Si se llama a desapilar con una pila vacía como parámetro se producirá un error y se abortará el programa.

- **Tope:**

```
tope(&idPila); // → <numero>
```

Devuelve el valor del elemento que está en el tope de la pila. No lo elimina de la pila. Con el valor devuelto se puede, por ejemplo, agregarlo a otra pila, mostrarlo por pantalla o evaluarlo como condición de un if.

- **PilaVacia:**

```
pilavacia(&idPila); // → <Verdadero / Falso>
```

Esta instrucción responde indicando si es verdadero o falso que la pila esté vacía. En lenguaje C el valor de verdad “verdadero” se simboliza con un número distinto de cero y el valor de verdad “falso” con un cero.

Esta instrucción se utiliza como parte de una condición, por ejemplo:

```
if(pilavacia(idPila))
{
   printf("la pila está vacía");
}
else
{
   printf("la pila no está vacía");
}
```

- **Mostrar:**

```
mostrar(&idPila);
```

Muestra el contenido de la pila por pantalla.

Ejemplo 1:

En el siguiente ejemplo, se definen dos pilas, en la pila ORIGEN se agregan 3 elementos por

teclado y posteriormente los mismos se desapilan y apilan en la pila DESTINO, para luego mostrarlos.

```
#include "pila.h"
// ver archivo explicativo de como agregar librerias
// el mismo se encuentra dentro de la librería de pilas compartida vía campus
int main() {
  Pila origen;
  Pila destino;
  inicpila(&origen);
  inicpila(&destino);
  leer (&origen);
  leer (&origen);
  leer (&origen);
  mostrar(&origen);
  apilar (&destino, desapilar(&origen));
  apilar (&destino, desapilar(&origen));
  apilar (&destino, desapilar(&origen));
  mostrar (&destino);
  return 0;
}
```

Cuando trabajamos con múltiples pilas en un mismo programa, podemos optar por hacer una simplificación en la definición de las mismas, de la siguiente manera:

```
Pila origen, destino;
```

lo cual equivale a:

```
Pila origen;
Pila destino;
```

Aplicación de estructura condicional simple if

La porción de pseudocódigo muestra los elementos de una pila si la misma no está vacía.

```
if(!pilavacia(&nombre_pila)) {
      mostrar(&nombre_pila)
}
```

Aplicación de estructura condicional doble if

La porción de pseudocódigo muestra los elementos de una pila si la misma no está vacía y en caso de que esté vacía muestra el mensaje “La pila está vacía”

```
if(!pilavacia(&nombre_pila)) {
   mostrar(&nombre_pila)
}else{
```

```
   printf("La pila está vacía");
}
```

Aplicación de la estructura repetitiva while

La porción de pseudocódigo desapila los elementos en otra pila, mientras la primera no esté vacía.

```
while (! pilavacia (&nombre_pila_origen)) {
      apilar (&nombre_pila_destino, desapilar (&nombre_pila_origen));
}
```