# Capítulo 2 Pilas:

## 2.1 Definición

Una pila es un tipo especial de lista abierta en la que sólo se pueden insertar y eliminar nodos en uno de los extremos de la lista. Estas operaciones se conocen como "push" y "pop", respectivamente "empujar" y "tirar". Además, las escrituras de datos siempre son inserciones de nodos, y las lecturas siempre eliminan el nodo leído.

Estas características implican un comportamiento de lista LIFO (Last In First Out), el último en entrar es el primero en salir.

El símil del que deriva el nombre de la estructura es una pila de platos. Sólo es posible añadir platos en la parte superior de la pila, y sólo pueden tomarse del mismo extremo.

El nodo típico para construir pilas es el mismo que vimos en el capítulo anterior para la construcción de listas:

```
struct nodo {
   int dato;
   struct nodo *siguiente;
};
```

## 2.2 Declaraciones de tipos para manejar pilas en C:

Los tipos que definiremos normalmente para manejar pilas serán casi los mismos que para manejar listas, tan sólo cambiaremos algunos nombres:

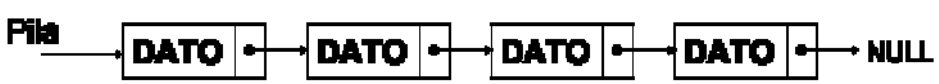
```
typedef struct _nodo {
   int dato;
   struct _nodo *siguiente;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Pila;
```

tipoNodo es el tipo para declarar nodos, evidentemente.

pNodo es el tipo para declarar punteros a un nodo.

Pila es el tipo para declarar pilas.



Es evidente, a la vista del gráfico, que una pila es una lista abierta. Así que sigue siendo muy importante que nuestro programa nunca pierda el valor del puntero al primer elemento, igual que pasa con las listas abiertas.

Teniendo en cuenta que las inserciones y borrados en una pila se hacen siempre en un extremo, lo que consideramos como el primer elemento de la lista es en realidad el último elemento de la pila.

## 2.3 Operaciones básicas con pilas:

Las pilas tienen un conjunto de operaciones muy limitado, sólo permiten las operaciones de "push" y "pop":

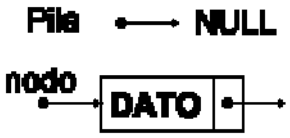
- Push: Añadir un elemento al final de la pila.
- Pop: Leer y eliminar un elemento del final de la pila.

## 2.4 Push, insertar elemento:

Las operaciones con pilas son muy simples, no hay casos especiales, salvo que la pila esté vacía.

### Push en una pila vacía:

Partiremos de que ya tenemos el nodo a insertar y, por supuesto un puntero que apunte a él, además el puntero a la pila valdrá NULL:



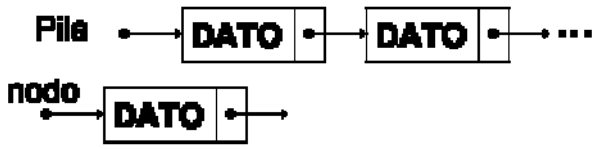

El proceso es muy simple, bastará con que:

1. nodo->siguiente apunte a NULL.
2. Pilaa apunte a nodo.

### Push en una pila no vacía:

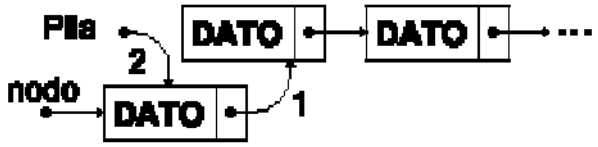
Podemos considerar el caso anterior como un caso particular de éste, la única diferencia es que podemos y debemos trabajar con una pila vacía como con una pila normal.

De nuevo partiremos de un nodo a insertar, con un puntero que apunte a él, y de una pila, en este caso no vacía:



El proceso sigue siendo muy sencillo:

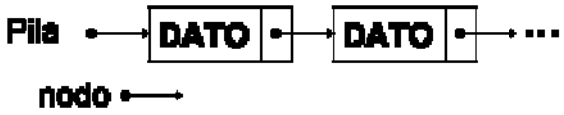
1. Hacemos que nodo->siguiente apunte a Pila.
2. Hacemos que Pila apunte a nodo.



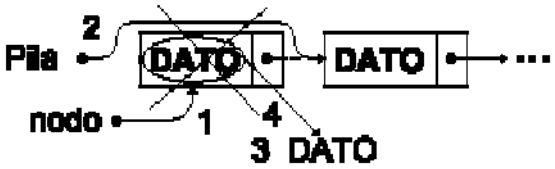

## 2.5 Pop, leer y eliminar un elemento:

Ahora sólo existe un caso posible, ya que sólo podemos leer desde un extremo de la pila.

Partiremos de una pila con uno o más nodos, y usaremos un puntero auxiliar, nodo:




1. Hacemos que nodo apunte al primer elemento de la pila, es decir a Pila.
2. Asignamos a Pila la dirección del segundo nodo de la pila: Pila->siguiente.
3. Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación pop equivale a leer y borrar.
4. Liberamos la memoria asignada al primer nodo, el que queremos eliminar.




Si la pila sólo tiene un nodo, el proceso sigue siendo válido, ya que el valor de Pila->siguiente es NULL, y después de eliminar el último nodo la pila quedará vacía, y el valor de Pila será NULL.

## 2.6 Ejemplo de pila en C:

Supongamos que queremos construir una pila para almacenar números enteros. Haremos pruebas intercalando varios "push" y "pop", y comprobando el resultado.

### Algoritmo de la función "push":

1. Creamos un nodo para el valor que colocaremos en la pila.
2. Hacemos que nodo->siguiente apunte a Pila.
3. Hacemos que Pila apunte a nodo.

```
void Push(Pila *pila, int v)
```

```
{
   pNodo nuevo;

   /* Crear un nodo nuevo */
   nuevo = (pNodo)malloc(sizeof(tipoNodo));
   nuevo->valor = v;

   /* Añadimos la pila a continuación del nuevo nodo */
   nuevo->siguiente = *pila;
   /* Ahora, el comienzo de nuestra pila es en nuevo nodo */
   *pila = nuevo;
}
```

## Algoritmo de la función "pop":

1. Hacemos que nodo apunte al primer elemento de la pila, es decir a Pila.
2. Asignamos a Pila la dirección del segundo nodo de la pila: Pila->siguiente.
3. Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación pop equivale a leer y borrar.
4. Liberamos la memoria asignada al primer nodo, el que queremos eliminar.

```
int Pop(Pila *pila)
{
   pNodo nodo; /* variable auxiliar para manipular nodo */
   int v;      /* variable auxiliar para retorno */

   /* Nodo apunta al primer elemento de la pila */
   nodo = *pila;
   if(!nodo) return 0; /* Si no hay nodos en pila retornamos 0 */
   /* Asignamos a pila toda la pila menos el primer elemento */
   *pila = nodo->siguiente;
   /* Guardamos el valor de retorno */
   v = nodo->valor;
   /* Borrar el nodo */
   free(nodo);
   return v;
}
```

## Código del ejemplo completo:

```
#include <stdlib.h>
#include <stdio.h>

typedef struct _nodo {
   int valor;
   struct _nodo *siguiente;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Pila;

/* Funciones con pilas: */
void Push(Pila *l, int v);
int Pop(Pila *l);

int main()
{
   Pila pila = NULL;

   Push(&pila, 20);
```

```
   Push(&pila, 10);
   printf("%d, ", Pop(&pila));
   Push(&pila, 40);
   Push(&pila, 30);

   printf("%d, ", Pop(&pila));
   printf("%d, ", Pop(&pila));
   Push(&pila, 90);
   printf("%d, ", Pop(&pila));
   printf("%d\n", Pop(&pila));

   system("PAUSE");
   return 0;
}

void Push(Pila *pila, int v)
{
   pNodo nuevo;

   /* Crear un nodo nuevo */
   nuevo = (pNodo)malloc(sizeof(tipoNodo));
   nuevo->valor = v;

   /* Añadimos la pila a continuación del nuevo nodo */
   nuevo->siguiente = *pila;
   /* Ahora, el comienzo de nuestra pila es en nuevo nodo */
   *pila = nuevo;
}

int Pop(Pila *pila)
{
   pNodo nodo; /* variable auxiliar para manipular nodo */
   int v;      /* variable auxiliar para retorno */

   /* Nodo apunta al primer elemento de la pila */
   nodo = *pila;
   if(!nodo) return 0; /* Si no hay nodos en la pila retornamos 0
*/
   /* Asignamos a pila toda la pila menos el primer elemento */
   *pila = nodo->siguiente;
   /* Guardamos el valor de retorno */
   v = nodo->valor;
   /* Borrar el nodo */
   free(nodo);
   return v;
}
```