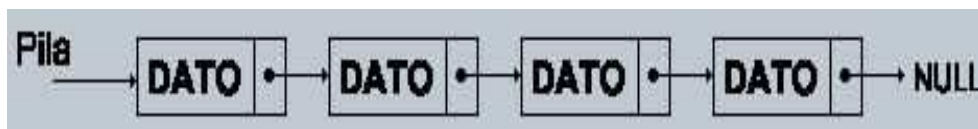


# 2012

UNAN – LEON  
Departamento de Computación  
Ing. En Sistemas de Información

Asignatura:  
Algoritmo y Estructura de Datos

## ESTRUCTURAS DINÁMICAS DE DATOS (PILAS, COLAS)



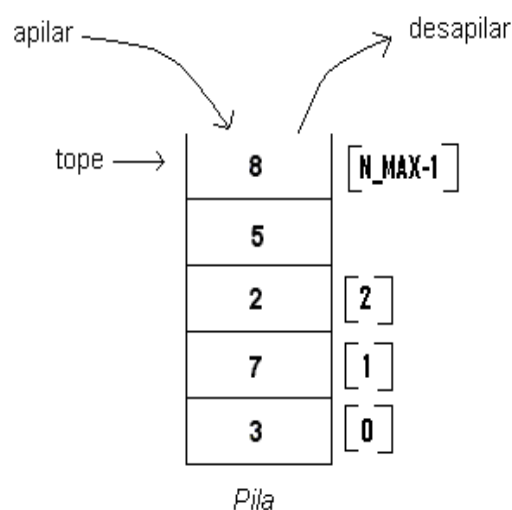
### TEMA 3: ESTRUCTURAS DINÁMICAS DE DATOS (PILAS)

#### 3.1 INTRODUCCIÓN:

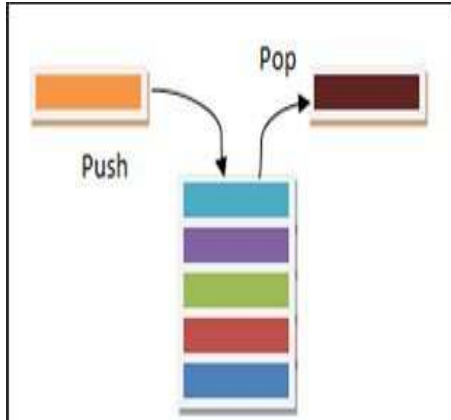
En términos de listas lineales, una pila puede ser definida como, una lista lineal, en la que las inserciones y supresiones se hacen en un extremo de la lista. Por esto, las pilas son estructuras de datos de tipo **LIFO (Last In, First Out): último en entrar, primero en salir**. Esto sugiere la idea de una pila de platos, en la que se van apilando platos unos encima de otros. En un momento determinado, si nos decidiéramos a quitar un plato, el plato que quitaríamos será el último que pusimos en la pila, es decir, el plato que está más arriba; de ahí, el nombre de estructuras tipo LIFO. 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".

Es evidente, que una pila es una lista lineal o 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.

Para empezar a comprender cómo funciona una pila, haremos uso de una estructura que incluya como campos, un array de elementos enteros y un variable tipo entero, que nos diga a cada momento, cuál es el tope de la pila. En este contexto, entenderemos por tope de la pila, la posición (índice) del arreglo en la que se encuentra el último elemento introducido a la pila. Dicho de otra forma, el tope de la pila incrementado en una unidad (recordar que los arreglos en C comienzan en la posición 0), nos indica el tamaño de la pila o la cantidad de elementos introducidos en la pila.



### 3.2 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.

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

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;
```

Donde:

- 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.

#### 3.2.1 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:

- **nodo->siguiente apunte a NULL.**
- **Pila apunte a nodo.**

### 3.2.2 **PUSH EN UNA PILA NO VACÍA:**

Podemos considerar el caso anterior como un caso particular de éste, la única diferencia es que 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:

- **Hacemos que nodo->siguiente apunte a Pila.**
- **Hacemos que Pila apunte a nodo.**

### 3.2.3 **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:
- Hacemos que nodo apunte al primer elemento de la pila, es decir a Pila.
- Asignamos a Pila la dirección del segundo nodo de la pila: Pila->siguiente.
- Guardamos el contenido del nodo para devolverlo como retorno de la función (recordemos que la operación pop equivale a leer y borrar.)
- 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.

### 3.3 **EJEMPLO #1 DE PILAS EN C:**

#### **Algoritmo de la función "push":**

Creamos un nodo para el valor que colocaremos en la pila.

Hacemos que nodo->siguiente apunte a Pila.

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":**

Hacemos que nodo apunte al primer elemento de la pila, es decir a Pila.  
Asignamos a Pila la dirección del segundo nodo de la pila: Pila->siguiente.  
Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación pop equivale a leer y borrar.  
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 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;  
}
```

**Código completo del ejemplo:**

```
#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;
}

```

### 3.4 EJEMPLO #2 DE PILAS EN C: CALCULADORA CON NOTACIÓN POSTFIJA APLICANDO PILAS.

Las operaciones de insertar y suprimir en una pila, son conocidas en los lenguajes ensambladores como **push** y **pop** respectivamente.

La operación de recuperación de un elemento de la pila, lo elimina de la misma.

Como **ejemplo** de utilización de una pila, vamos a simular una calculadora capaz de realizar las operaciones de +, -, \* y /. La mayoría de las calculadoras aceptan la notación **infija** y unas pocas la notación postfija. Su principio es el de evaluar los datos directamente cuando se introducen y manejarlos dentro de una estructura LIFO (Last In First Out), lo que optimiza los procesos a la hora de programar. Básicamente la diferencias con el método algebraico o notación de infijo es que, al evaluar los datos directamente al introducirlos, no es necesario ordenar la evaluación de los mismos, y que para ejecutar un comando, primero se deben introducir todos sus argumentos, así, para hacer una suma 'a+b=c' el RPN lo manejaría a b +, dejando el resultado 'c' directamente. En estas últimas para sumar 10 y 20 introduciríamos primero 10, después 20 y por último el +. Cuando se introducen los operandos, se colocan en una pila y cuando se introduce el operador, se sacan dos operandos de la pila. La ventaja de la notación postfija es que expresiones complejas pueden evaluarse fácilmente sin mucho código.

**El programa realiza las siguientes operaciones:**

1. Leer un dato, operando u operador, y lo almacena en la variable op.
2. Analiza op; si se trata de un operando lo mete en la pila utilizando la **función push()**; y si se trata de un operador extrae, utilizando la **función pop()** los dos últimos operandos de la pila, realiza la operación indicada por dicho operador e introduce el resultado en la pila para encadenarlo con otra posible operación.

La **función push()**, introduce un elemento al inicio de la pila. El método que sigue es simple y sencillo:

1. Crear un nuevo elemento y referenciarlo por q.
2. Apuntar con el puntero siguiente de q, a la cima de la pila.
3. Reasignar la cima de la pila, para que ahora apunte a q.

La **función pop()**, sigue estos pasos para obtener un elemento de la pila:

1. Guarda en una variable auxiliar la cima de la pila.
2. Verifica el estado de la pila, y si está vacía, regresa 0.
3. Si la pila no está vacía, entonces
4. Obtiene en una variable x el dato que se encuentra en el elemento cima.
5. Reasigna la cima de la pila, para que ahora pase a apuntar adonde apunta su puntero siguiente. Luego de esta operación, la cima de la pila puede ser NULL, lo cual quiere decir que sólo había un elemento, o la cima es diferente de NULL, lo cual indica que había al menos dos elementos.
6. Libera el espacio de memoria de la cima antigua, referenciado por la variable auxiliar.
7. Regresa el valor de la variable x.

### **Código en C de la calculadora con notación postfija**

```
//calculadorapostfija
#include <stdio.h>
#include <stdlib.h>
#define PilaVacía (cima==NULL)      /* ¿Pila Vacía? */

typedef struct datos /*Estructura de un elemento de la pila*/
{
    float dato;
    struct datos *siguiente;
}elemento;

/*Definición de Funciones*/
void error(void)
{
    perror("Error: insuficiente espacio en memoria.");
    exit(1);
}

elemento *NuevoElementoPila()
{
    elemento *q;
    q = (elemento *)malloc(sizeof(elemento));
    if(q == NULL)
        error();
    return(q);
}
```



**/\*Añadir un dato a la pila\*/**

```
void push(elemento **p, float x)
{
    elemento *q, *cima;
    cima = *p;    /*cima de la pila*/
    q=NuevoElementoPila();
    q->dato=x;
    q->siguiente=cima;
    cima=q;
    *p=cima;
}
```

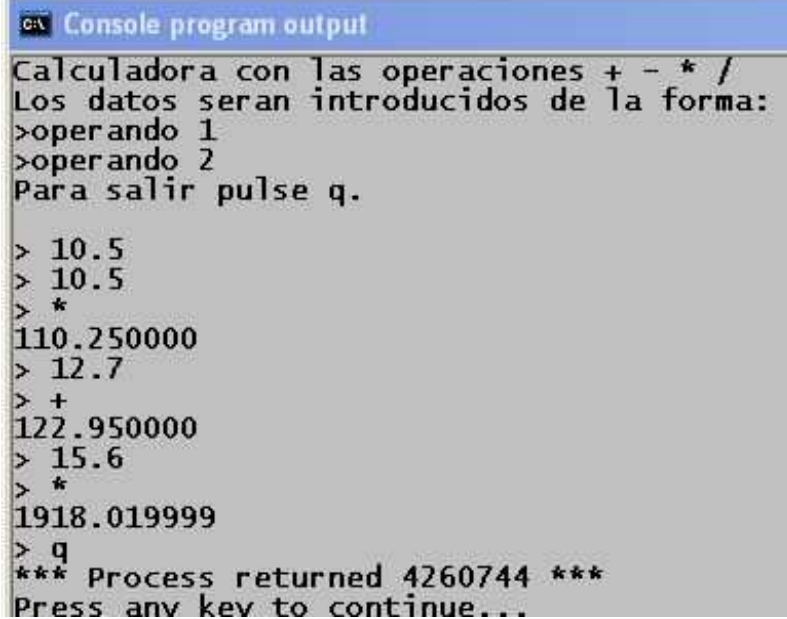
**/\*Recuperar un dato de la cima de la pila\*/**

```
float pop(elemento **p)
{
    elemento *cima;
    float x;
    cima = *p;    /*cima o tope de la pila*/
    if(PilaVacía)
    {
        printf("\nError: pop de una pila vacía");
        return 0;
    }
    else
    {
        x = cima->dato;
        *p = cima->siguiente;
        free (cima);
        return (x);
    }
}
```

void main()

```
{
    elemento *cima = NULL; /*Pila Vacía*/
    float a,b,res;
    char op[2];
    system("cls");
    printf("Calculadora con las operaciones + - * ^\n");
    printf("Los datos serán introducidos de la forma: (operando1 operando 2\n");
    printf("operador)\n");
    printf(">operando 1:\n");
    printf(">operando 2:\n");
    printf("Para salir pulse q.\n\n");
}
```

```
do
{
    printf("> ");
    gets(op);
    switch(*op)
    {
        case '+':
            a=pop(&cima);
            b=pop(&cima);
            res=a+b;
            printf("%f\n",res);
            push(&cima,res);
            break;
        case '-':
            a=pop(&cima);
            b=pop(&cima);
            res=a-b;
            printf("%f\n",res);
            push(&cima,res);
            break;
        case '*':
            a=pop(&cima);
            b=pop(&cima);
            res=a*b;
            printf("%f\n",res);
            push(&cima,res);
            break;
        case '/':
            a=pop(&cima);
            b=pop(&cima);
            if(b==0)
            {
                printf("\n\naDivision por cero\n");
                break;
            }
            res=a/b;
            printf("%f\n",res);
            push(&cima,res);
            break;
        default:
            push(&cima,atof(op));
    }
}while(*op!='q');
}
```

**Ejemplo de Salida:**

```
C:\> Console program output
Calculadora con las operaciones + - * /
Los datos seran introducidos de la forma:
>operando 1
>operando 2
Para salir pulse q.

> 10.5
> 10.5
> *
110.250000
> 12.7
> +
122.950000
> 15.6
> *
1918.019999
> q
*** Process returned 4260744 ***
Press any key to continue...
```