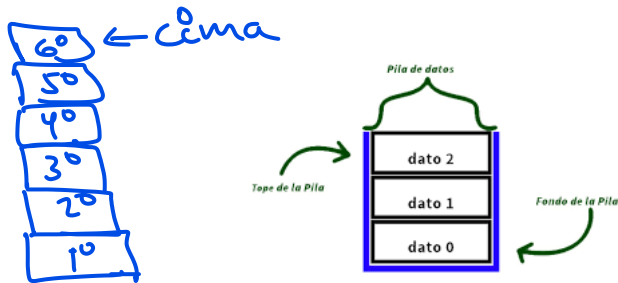


# Pilas



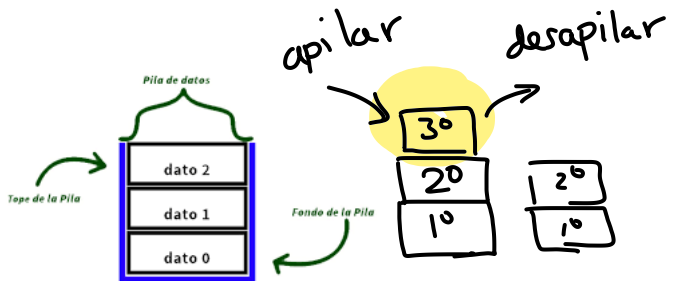
Dr. Jaime Osorio Ubaldo

# Estructura de dato Pila



La pila o stack, es una colección ordenada de elementos en la cual en un extremo (llamado parte superior) se pueden insertar nuevos elementos y de la cual se pueden retirar otros.

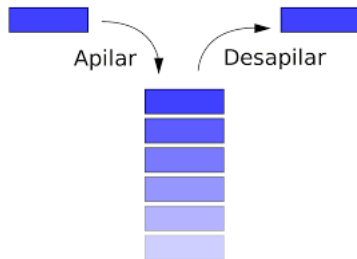
# Estructura de dato Pila



La pila o stack, es una colección ordenada de elementos en la cual en un extremo (llamado parte superior) se pueden insertar nuevos elementos y de la cual se pueden retirar otros.

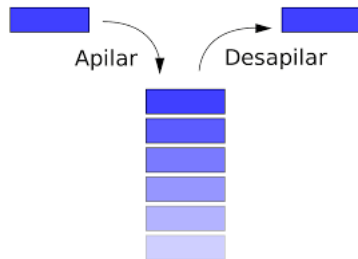
A la pila se le conoce como estructuras de datos de tipo LIFO (el último en entrar primero en salir).

# Operaciones



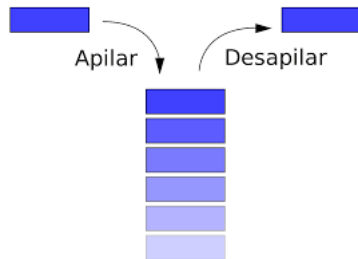
- Apilar un elemento a la pila.

# Operaciones



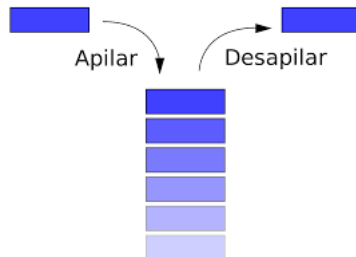
- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.

# Operaciones



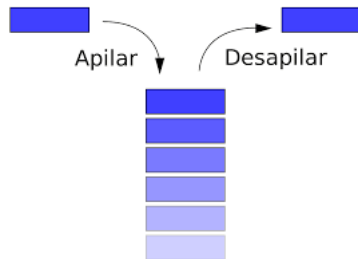
- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.
- Determinar si una pila está vacía.

# Operaciones



- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.
- Determinar si una pila está vacía.
- Determinar el tope o cima de la pila.

# Operaciones

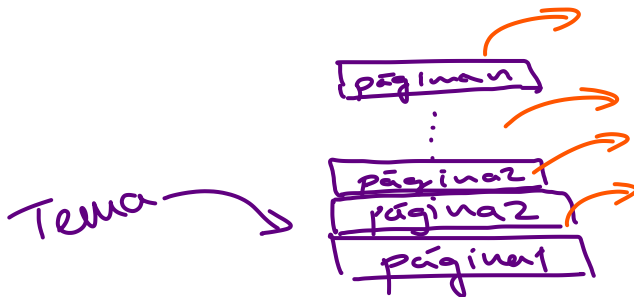


- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.
- Determinar si una pila está vacía.
- Determinar el tope o cima de la pila.
- Vaciar pila.



- 1 Reconocedores de análisis sintácticos de lenguajes de libre contexto.

- 1 Reconocedores de análisis sintácticos de lenguajes de libre contexto.
- 2 LLamadas a subprogramas (por ejemplo en los navegadores).



- 1 Reconocedores de análisis sintácticos de lenguajes de libre contexto.
- 2 LLamadas a subprogramas(por ejemplo en los navegadores).
- 3 En programas que requiere aplicar recursividad (factorial de un número, torres de Hanoi, algoritmos de ordenamiento).

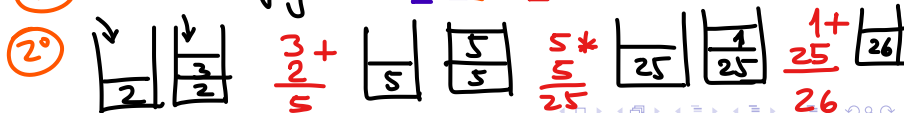
- 1 Reconocedores de análisis sintácticos de lenguajes de libre contexto.
- 2 Llamadas a subprogramas (por ejemplo en los navegadores).
- 3 En programas que requiere aplicar recursividad (factorial de un número, torres de Hanoi, algoritmos de ordenamiento).
- 4 Tratamiento de expresiones aritméticas (Convertir de infija a prefija o postfija).

Notación

Infija	Prefija	Postfija
$4 + 5$	$45 +$	$+ 45$

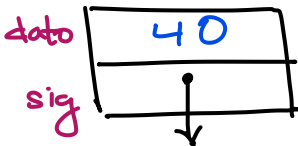
- 1 Reconocedores de análisis sintácticos de lenguajes de libre contexto.
- 2 Llamadas a subprogramas (por ejemplo en los navegadores).
- 3 En programas que requiere aplicar recursividad (factorial de un número, torres de Hanoi, algoritmos de ordenamiento).
- 4 Tratamiento de expresiones aritméticas (Convertir de infija a prefija o postfija).
- 5 Las calculadoras usan pilas para el cálculo de expresiones aritméticas como  $(2 + 3) * 5 + 1$ .

1º Notación Infija: 23+5\*1+



# Definimos la estructura de datos

```
struct nodopila{  
    int dato;  
    nodopila *sig;  
};  
typedef struct nodopila *pnodopila;  
pnodopila pcima;
```



↑ variable tipo puntero a nodopila.

# Clase pila

```
class pila{
```

```
private:
```

```
    pnodopila pcima;
```

```
public:
```

```
    pila();
```

```
    ~pila();
```

```
    void apilar(int x);
```

```
    int desapilar();
```

```
    void imprimir();
```

```
};
```

constructor de la clase  
destructor de la clase

clase función  
↓ ↓  
`pila::pila(){`  
    `pcima = NULL;`  
`}`

Pila vacía

pcima →



# Destructor

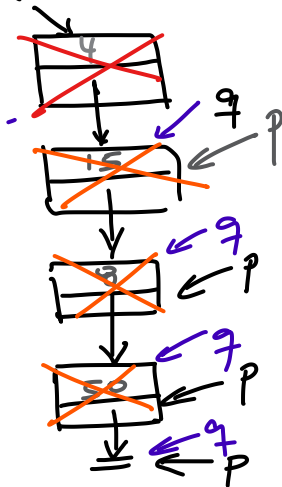
date

función

```
pila::~pila(){  
  pnodopila p,q;  
  if(pcima!=NULL){  
    p = pcima;  
    while(p!=NULL){  
      q=(*p).sig;  
      delete p;  
      p=q;  
    }  
  }  
}
```

si la pila  
no está  
vacía

pcima



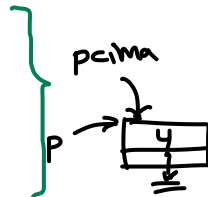
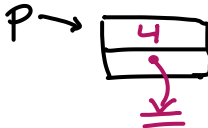
# Apilar

```
void pila::apilar(int x){  
    pnodopila p; ✓  
    p = new nodopila; ✓  
    (*p).dato=x; ✓  
    (*p).sig=pcima; ✓  
    pcima=p; ✓  
}
```

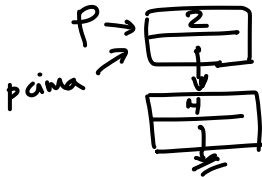
$p \rightarrow \text{dato} = x$

$pcima$

$apilar(4):$



$apilar(2):$

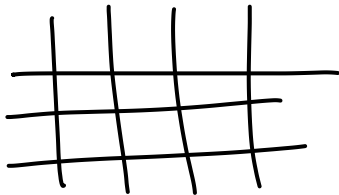
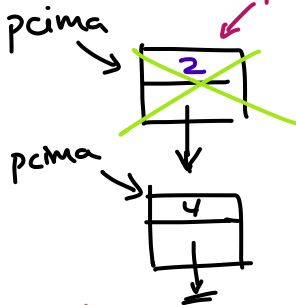


# Desapilar

void

```
int pila::desapilar(){  
    pnodopila p; ✓  
    int x; ✓  
    p = pcima; ✓  
    pcima = (*p).sig;  
    x = (*p).dato;  
    delete p; // Liberar memoria  
    return x;  
}
```

desapilar()



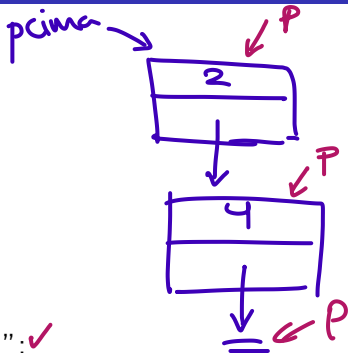
delete P

La variable  
tipo puntero

# Imprimir Pila

```
void pila::imprimir(){
    pnodopila p;
    if(pcima==NULL)
        cout<<" Pila vacia" <<endl; ✓
    else{
        p=pcima;
        while(p!=NULL){
            cout<<(*p).dato<<" - > "; ✓
            p=(*p).sig;
        }
        cout<<" NULO" <<endl;
    }
}
```

pcima  
Pila Vacía



2 → 4 → NULO

```
#include <iostream>
```

```
#include "pila.h"
```

```
using namespace std;
```

```
int main(){
```

```
    pila stack;
```

```
    int x, i;
```

```
    cout<< "Apilando datos:" << endl;
```

```
    for(i=1;i<=5;i++){
```

```
        cin>>x;
```

```
        stack.apilar(x);
```

```
        stack.imprimir(); }
```

```
    cout<< endl<< "Desapilando datos" << endl;
```

```
    while(!stack.estavacia()){
```

```
        x=stack.desapilar();
```

```
        cout<<x<< endl;
```

```
        stack.imprimir();
```

```
    }
```

```
}
```

← crear un objeto de la clase pila

stack.apilar(8)  
stack.apilar(4)

stack.desapilar()

# Problema

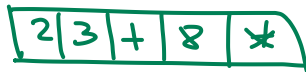
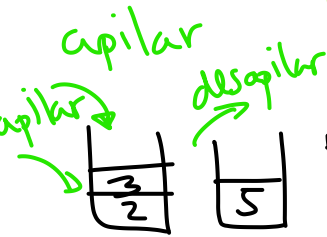
Entrada:  $(2+3)*8$

pila

Escriba un programa que permita calcular una expresión dada en postfijo.

Input :  $\underline{2} \ \underline{3} \ + \ \underline{8} \ *$   
Output : 40

pila



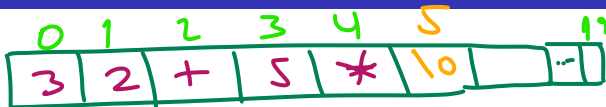
## Código Asscii

## TABLA DE CARACTERES DEL CÓDIGO ASCII

1	25	49	73	97	121	145	169	193	217	241
2	26	50	74	98	122	146	170	194	218	242
3	27	51	75	99	123	147	171	195	219	243
4	28	52	76	100	124	148	172	196	220	244
5	29	53	77	101	125	149	173	197	221	245
6	30	54	78	102	126	150	174	198	222	246
7	31	55	79	103	127	151	175	199	223	247
8	32	56	80	104	128	152	176	200	224	248
9	33	57	81	105	129	153	177	201	225	249
10	34	58	82	106	130	154	178	202	226	250
11	35	59	83	107	131	155	179	203	227	251
12	36	60	84	108	132	156	180	204	228	252
13	37	61	85	109	133	157	181	205	229	253
14	38	62	86	110	134	158	182	206	230	254
15	39	63	87	111	135	159	183	207	231	255
16	40	64	88	112	136	160	184	208	232	PRESIONA LA TECLA
17	41	65	89	113	137	161	185	209	233	Alt
18	42	66	90	114	138	162	186	210	234	MÁS EL NÚMERO
19	43	67	91	115	139	163	187	211	235	
20	44	68	92	116	140	164	188	212	236	
21	45	69	93	117	141	165	189	213	237	
22	46	70	94	118	142	166	190	214	238	
23	47	71	95	119	143	167	191	215	239	
24	48	72	96	120	144	168	192	216	240	

# Calculadora

```
int main() {  
    int x,y;  
    pila p;  
    char cadena[20];  
    cout<< "Ingrese la cadena en postfijo" <<endl;  
    fgets(cadena,20,stdin);  
    for(int i=0;cadena[i]!='\0';i++) {  
        if(cadena[i]>=48&&cadena[i]<=57)  
            p.apilar(cadena[i]-48);  
        else if(cadena[i]=='+' || cadena[i]=='*')  
            x=p.desapilar();  
            y=p.desapilar();  
            p.apilar(x+y);  
        else if(cadena[i]=='-' || cadena[i]=='/')  
            x=p.desapilar();  
            y=p.desapilar();  
            p.apilar(x-y);  
            p.apilar(x/y);  
    }  
    p.imprimir();  
}
```



Fin de cadena





Agregar el código necesario para que la calculadora realice las cuatro operaciones básicas (adición, sustracción, multiplicación y división).