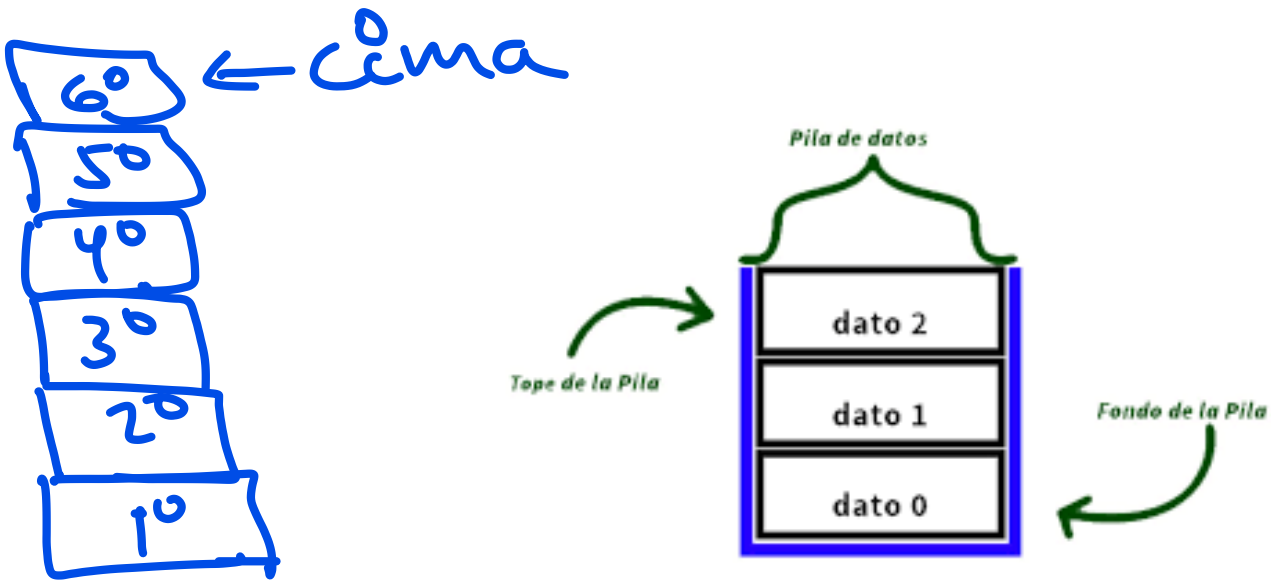
# Pilas

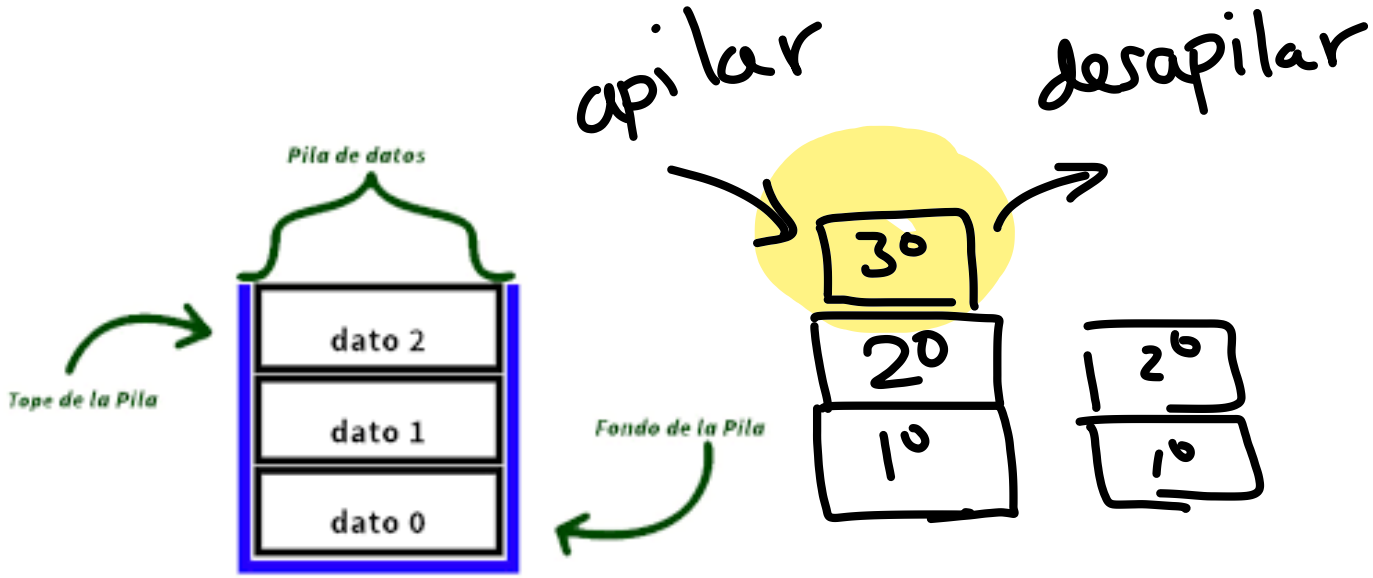

Dr. Jaime Osorio Ubaldo

# Estructura de dato Pila



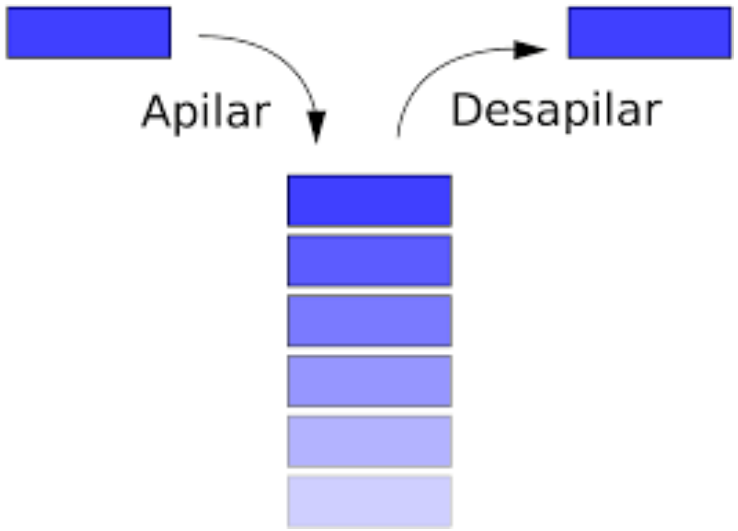

La pila o stack, es una colección ordenada de elementos en la cual en un extremo(llamado parte superior) se pueden insertar nuevos elementos y de la cual se pueden retirar otros.

# Estructura de dato Pila



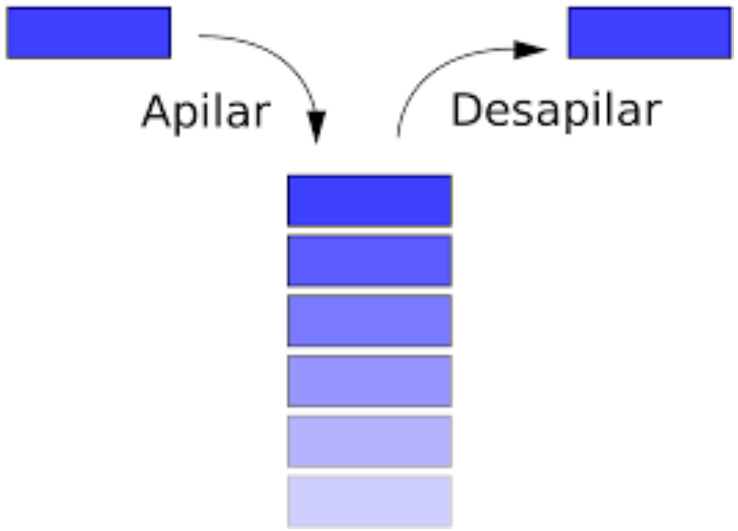

La pila o stack, es una colección ordenada de elementos en la cual en un extremo(llamado parte superior) se pueden insertar nuevos elementos y de la cual se pueden retirar otros.
A la pila se le conoce como estructuras de datos de tipo LIFO( el último en entrar primero en salir).

# Operaciones



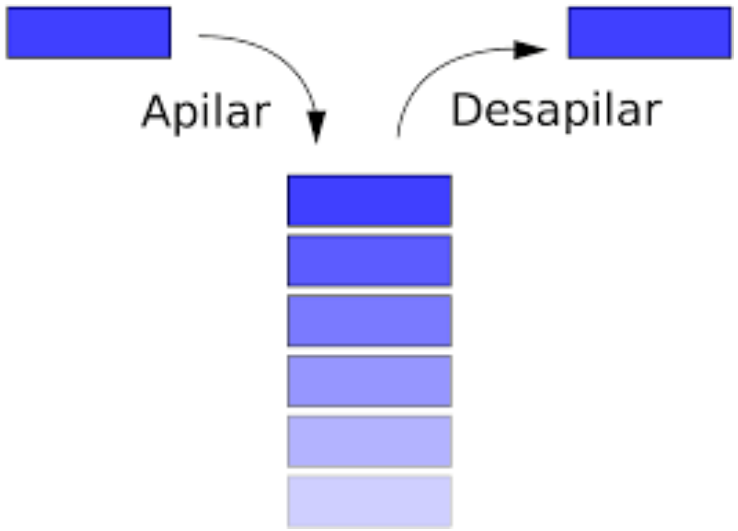

- Apilar un elemento a la pila.

# Operaciones



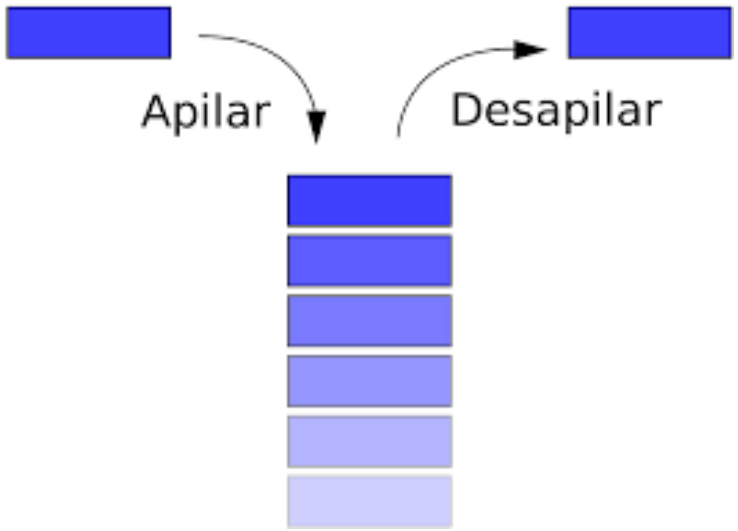

- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.

# Operaciones



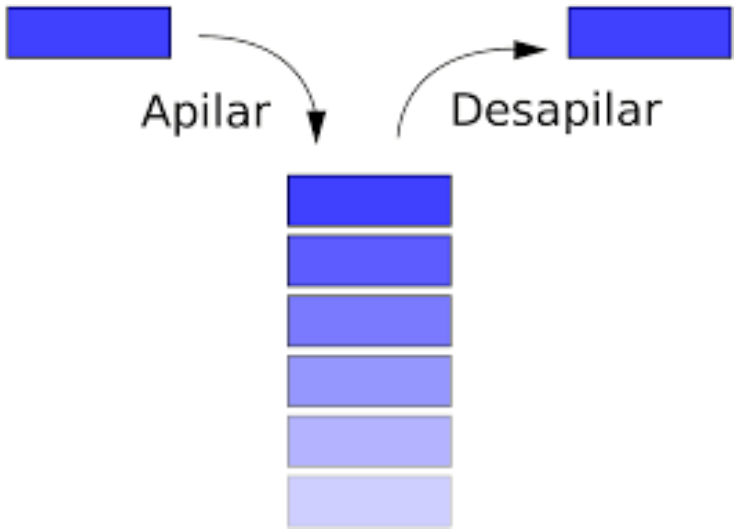

- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.
- Determinar si una pila está vacía.

# 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



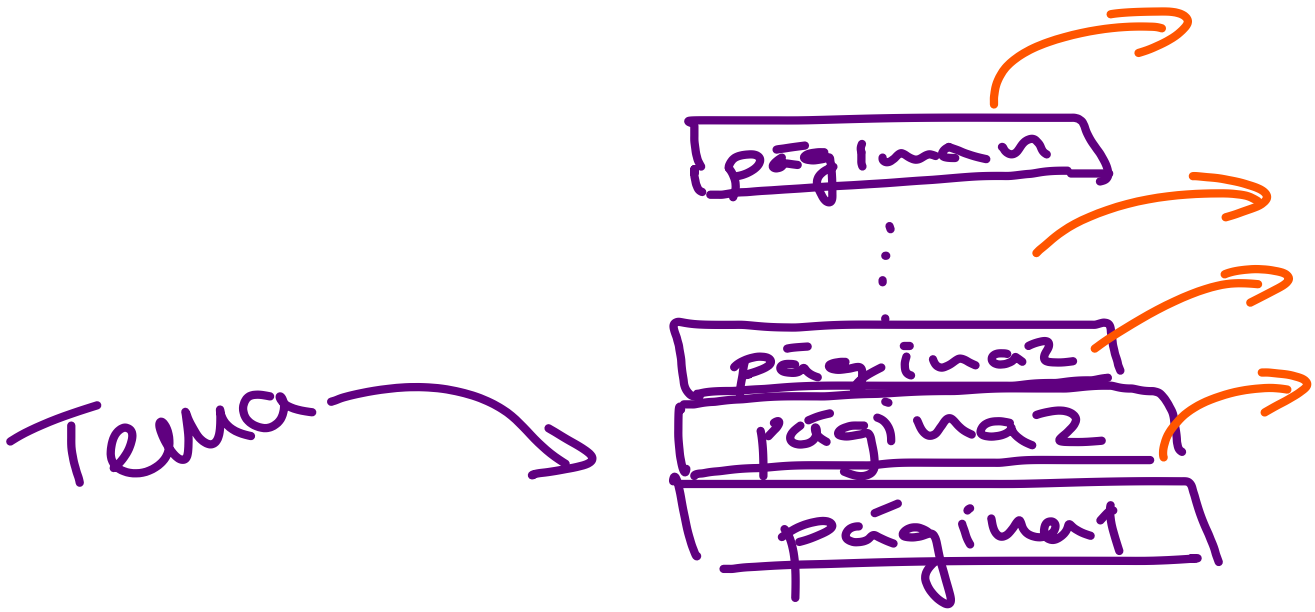

- Apilar un elemento a la pila.
- Desapilar un elemento de la pila.
- Determinar si una pila está vacía.
- Determinar el tope o cima de la pila.
- Vaciar pila.

# Aplicaciones

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

# Aplicaciones

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



# Aplicaciones

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 Hannoi, algoritmos de ordenamiento).

# Aplicaciones

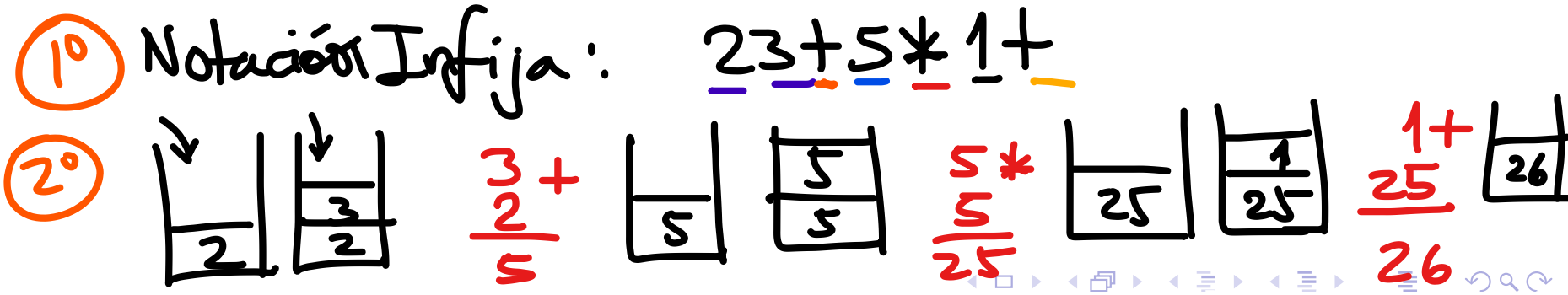
1. Reconocedores de análisis sintácticos de lenguajes de libre contexto.
2. LLamadas a subprogramas(por ejemplo en los navegadores).
3. En programas que requiere aplicar recursividad (factorial de un número, torres de Hannoi, algoritmos de ordenamiento).
4. Tratamiento de expresiones aritméticas (Convertir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

Notación

| Infija | Prefija | Postfija |
|---|---|---|
| 4+5 | 45+ | +45 |

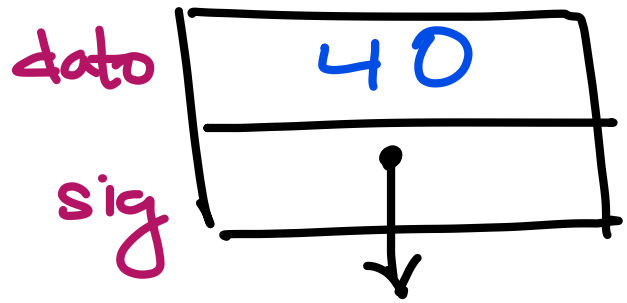
# Aplicaciones

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 Hannoi, 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 $(2 + 3) * 5 + 1$.



# Definimos la estructura de datos

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




↑ variable tipo puntero a nodopila.

# Clase pila

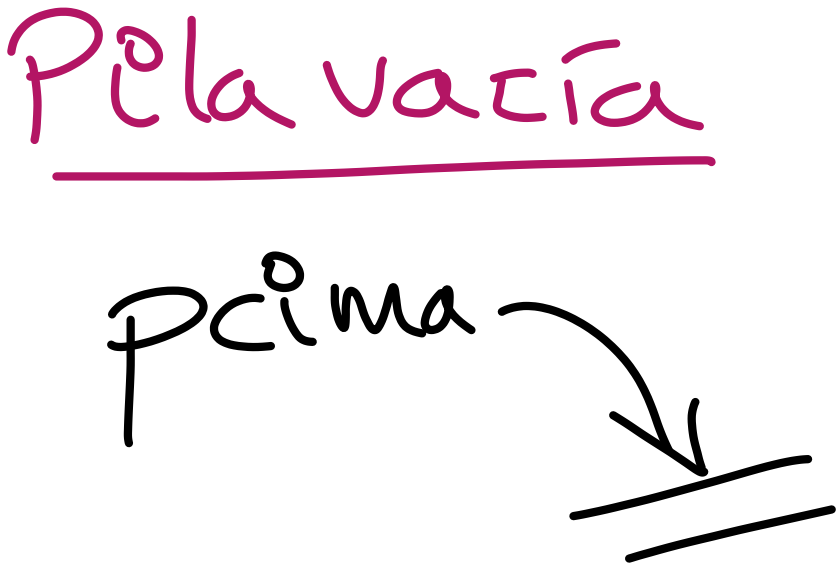
```
class pila{
private:
      pnodopila pcima;
public:
      pila();
   ~pila();
      void apilar(int x);
      int desapilar();
      void imprimir();
};
```

constructor de la clase
destructor de la clase

# Constructor

clase función

```
pila::pila(){
        pcima = NULL;
}
```



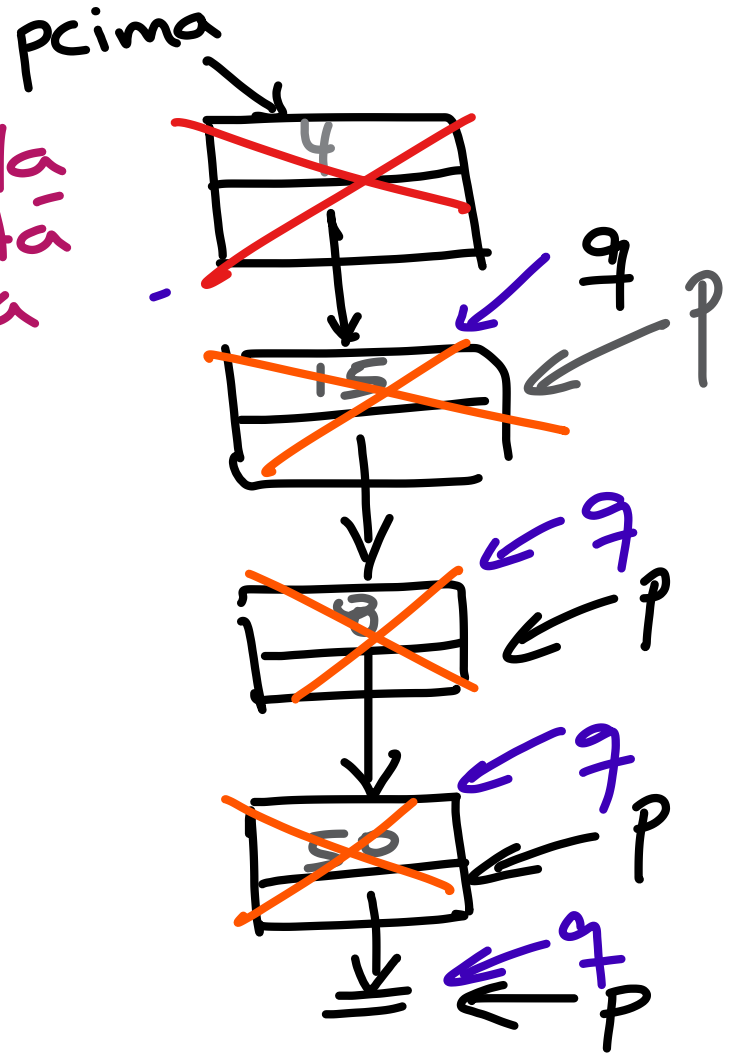
# Destructor

clase

función

si la pila no está vacía

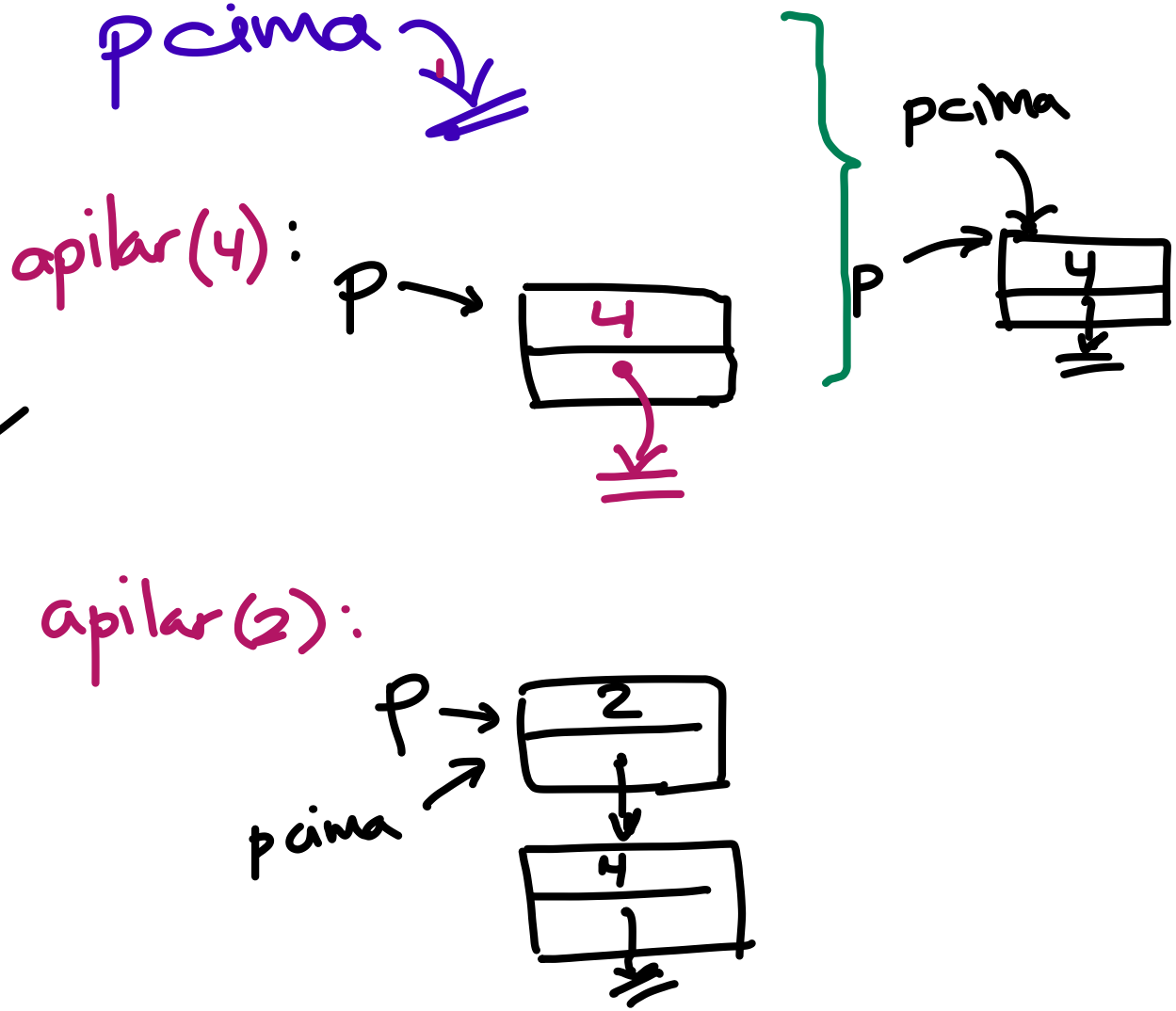
```
pila::~pila(){
pnodopila p,q;
if(pcima!=NULL){
      p = pcima;
      while(p!=NULL){
            q=(*p).sig;
            delete p;
            p=q;
      }
}
}
```



# Apilar

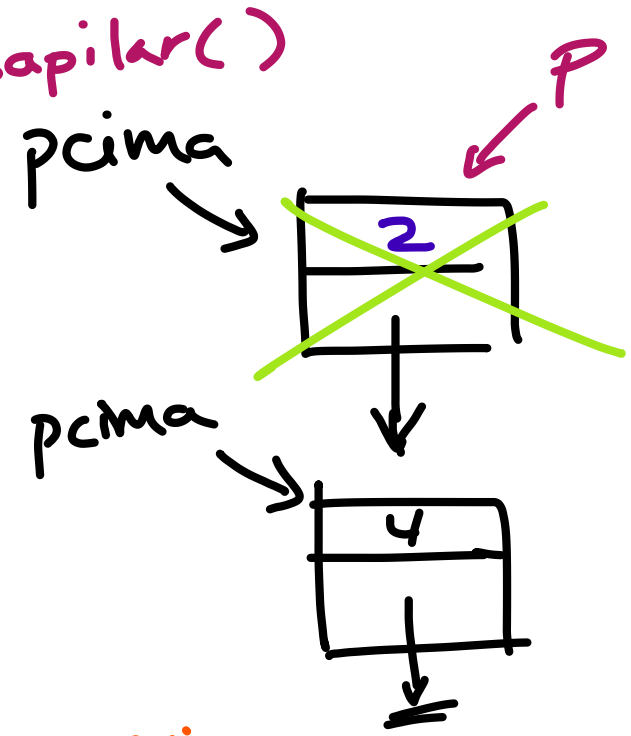
```
void pila::apilar(int x){
      pnodopila p;
      p = new nodopila
      (*p).dato=x;
      (*p).sig=pcima;
      pcima=p;
}
```

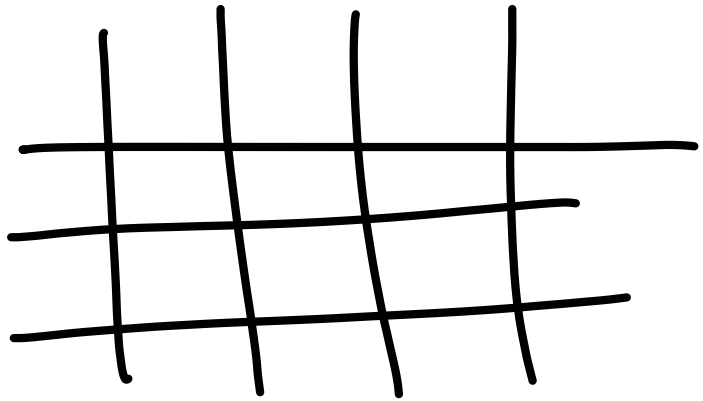
p->dato = x



# Desapilar

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



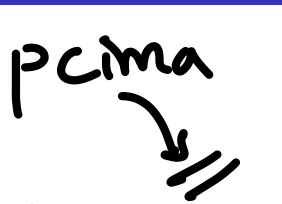
delete p

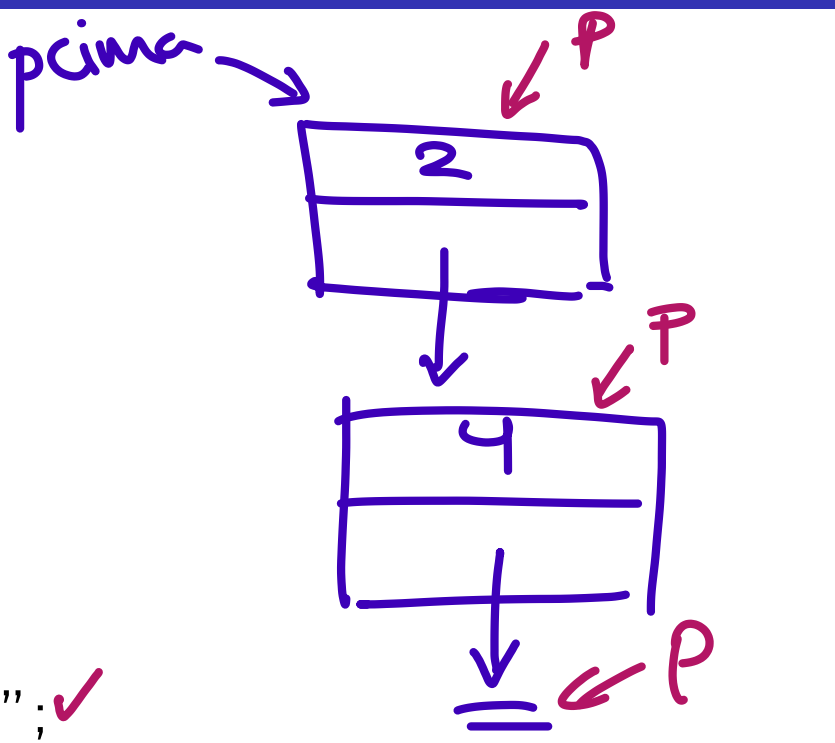
↑

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;
}
}
```



# Ejecución

pila.cpp

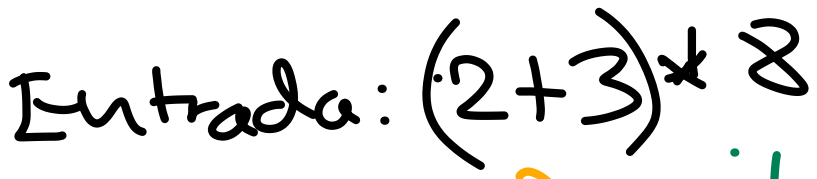
```
#include <iostream>
#include "pila.h"
using namespace std;
int main(){
pila stack;
int x, i;
cout<< "Apilando datos:"<<endl;
for(i=1;i<=5;i++){
cin>>x;
stack.apilar(x);
stack.imprimir(); }
cout<<endl<<"Desapilando datos"<<endl;
while(!stack.estavacia()){
x=stack.desapilar();
cout<<x<<endl;
stack.imprimir();
}
}
```

← crear un objeto de la clase pila

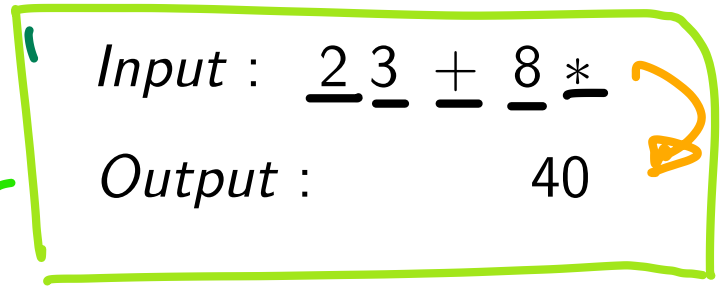
stack.apilar(8)
stack.apilar(4)

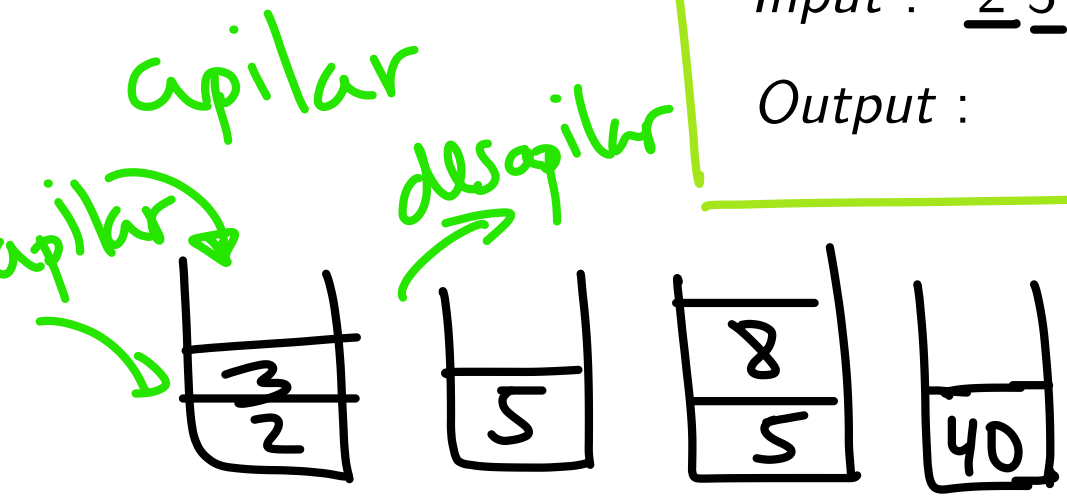
stack.desapilar()

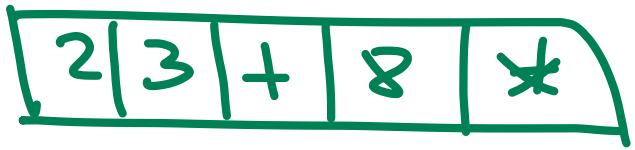
# Problema




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



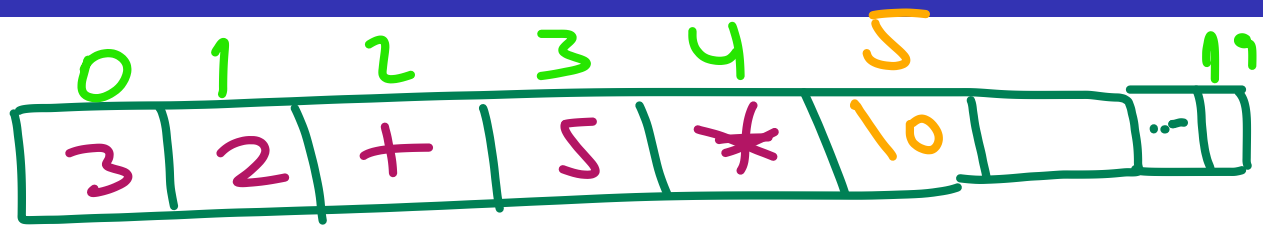



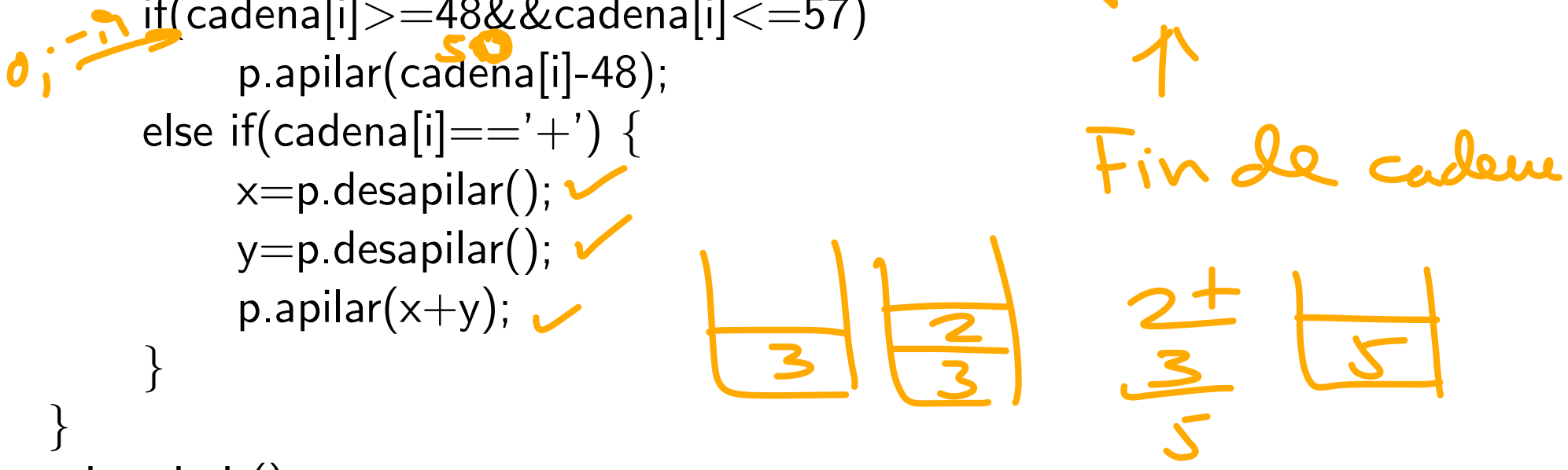



# Código Asscii

# 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]=='+') {
            x=p.desapilar();
            y=p.desapilar();
            p.apilar(x+y);
    }
}
p.imprimir();
}
```



# Tarea

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