

ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

# Aplicaciones de pilas

Manuel Montenegro Montes  
Departamento de Sistemas Informáticos y Computación  
Facultad de Informática – Universidad Complutense de Madrid

Cuando debemos usar las pilas.

Sabemos qué son las pilas, cómo se implementan, y ahora vamos a ver para qué sirven.

# Expresiones en forma postfija

# Expresiones en forma postfija

- Expresión en forma infija: operandos situados a ambos lados de cada operador.

$$(3 \text{ * } (5 \text{ + } 2)) \text{ - } 6$$

- La misma expresión en forma postfija:

$$(3 \text{ (} 5 \text{ 2 +) * ) 6 -}$$
$$3 \text{ 5 2 + * 6 -}$$

- Objetivo:** evaluar expresión en forma postfija.
  - Por ejemplo,  $3 \text{ 5 2 + * 6 -}$  se evalúa al valor 15.

# Expresiones en forma postfija

¿Qué ventajas tiene esto?

- Las expresiones en forma postfija no contienen ambigüedades, no necesitan paréntesis o reglas de precedencia entre operadores, al contrario que las expresiones en forma infija.

$$2 + 4 * 5$$

Por lo que nos han enseñado sabemos que primero se multiplica y que después se suma, pero si no cómo lo sabríamos.

En los postfijos no hay esa ambigüedad que nos enseñaron el cole.



# Otros ejemplos

- 2 3 + 6 + 1 + = 2+3= 5+6= 11 +1= 12
- 3 1 - 6 5 \* + = 3-1= 2, 6\*5=30, 30+2=32



# Evaluando las expresiones mediante una pila

- Comenzamos con una pila vacía.
- Recorremos de izquierda a derecha los caracteres de la expresión.
- Si el carácter actual es un número:
  - Insertar el número en la pila.
- Si el carácter actual es un operador:
  - Desapilar los dos operandos y realizar la operación con ellos.
  - Apilar el resultado. Este es un uso de las pilas.
- Al finalizar el recorrido, el elemento que quede en la pila es el resultado de evaluar la expresión.

El único valor de la pila

# Ejemplo de ejecución

3 5 2 + \* 6 -  
↑



# Ejemplo de ejecución

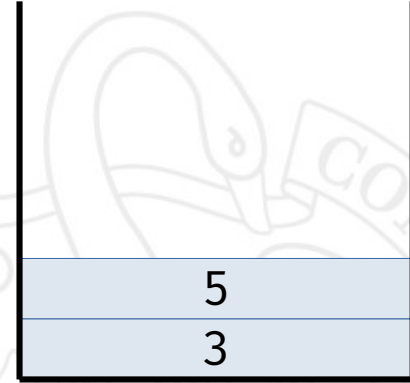

3 5 2 + \* 6 -  
↑





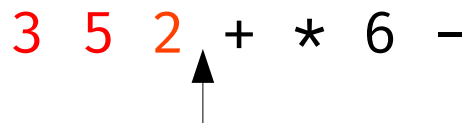
# Ejemplo de ejecución

3 5 2 + \* 6 -



# Ejemplo de ejecución

3 5 2 + \* 6 -

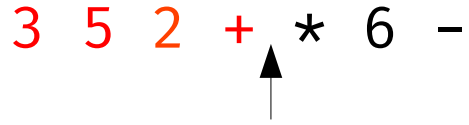


desapilamos  
y hacemos suma  
y apilamos resul

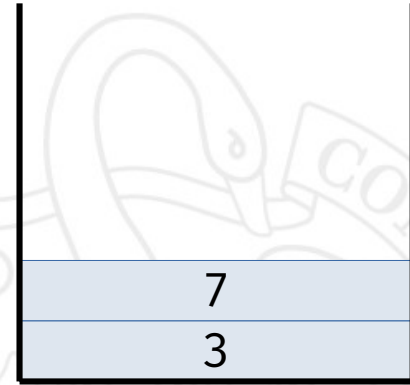
2
5
3

# Ejemplo de ejecución

3 5 2 + \* 6 -

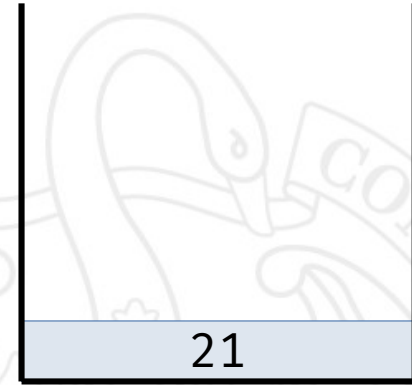


desapilamos y multi  
apilamos result



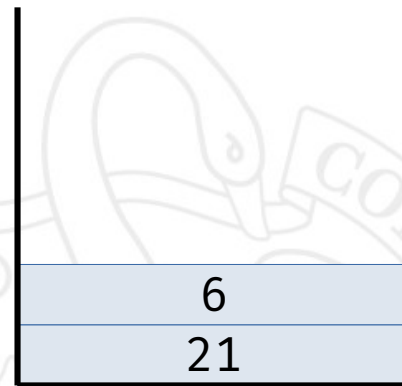
# Ejemplo de ejecución

3 5 2 + \* 6 -  
↑



# Ejemplo de ejecución

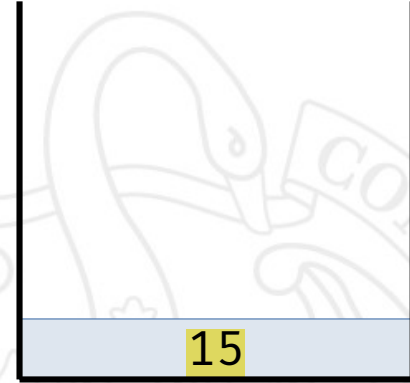
3 5 2 + \* 6 -  
↑



6
21

# Ejemplo de ejecución

3 5 2 + \* 6 -



# Pila de llamadas a funciones

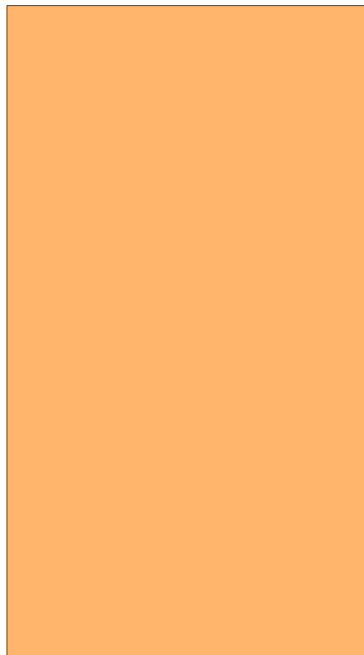
Se utiliza en la ejecución de programas en C++ y en la mayoría de lenguajes de programación

# Ejemplo de ejecución

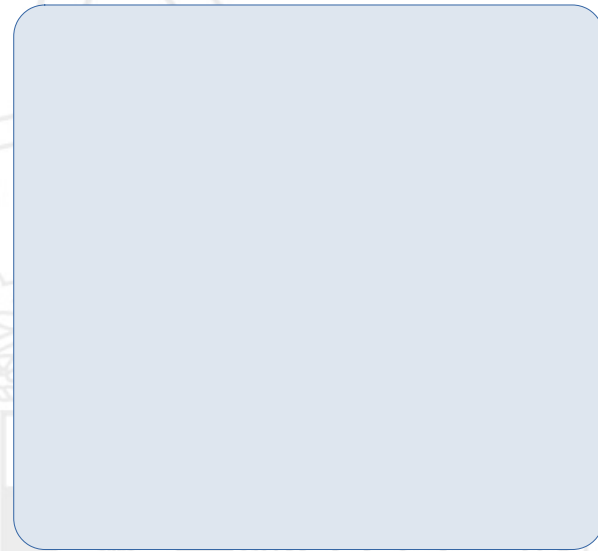
- La región de memoria en la que se almacenan las variables locales y parámetros de un programa funciona como una pila.

Al principio habíamos dicho que nos olvidáramos de estas pilas, que no tenían nada que ver. Pero en realidad sí que tienen algo que ver. Vamos a ver por qué se le llama pila.

**Pila**



**Heap**





# Ejemplo de ejecución

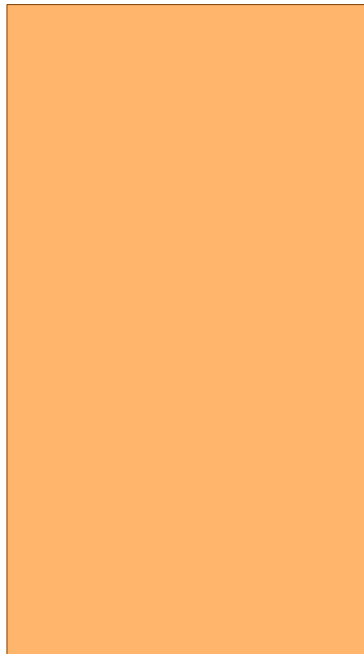
Funciones que se llaman unas a otras.

```
void f(int x) {  
    int y = 40;  
}
```

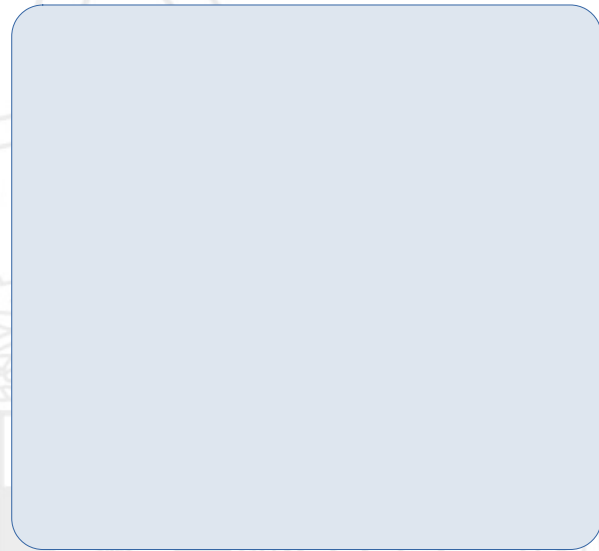
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila



Heap



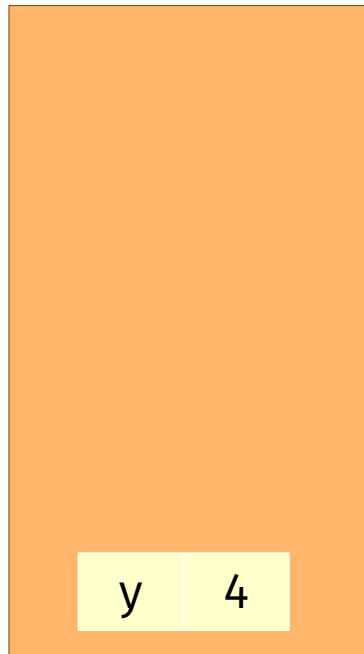
# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

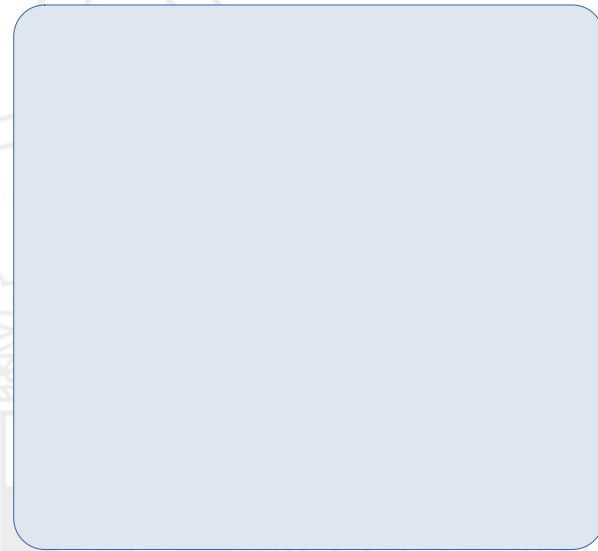
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila



Heap



# Ejemplo de ejecución

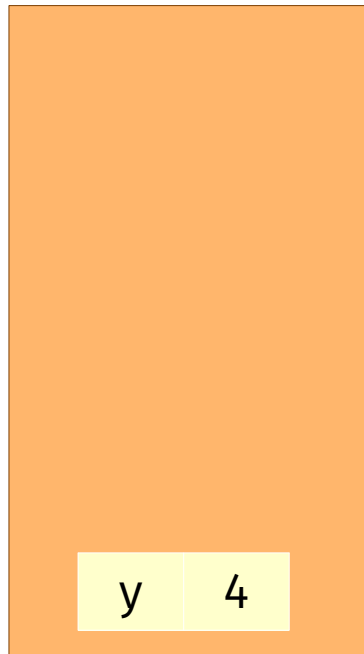
```
void f(int x) {  
    int y = 40;  
}
```

```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

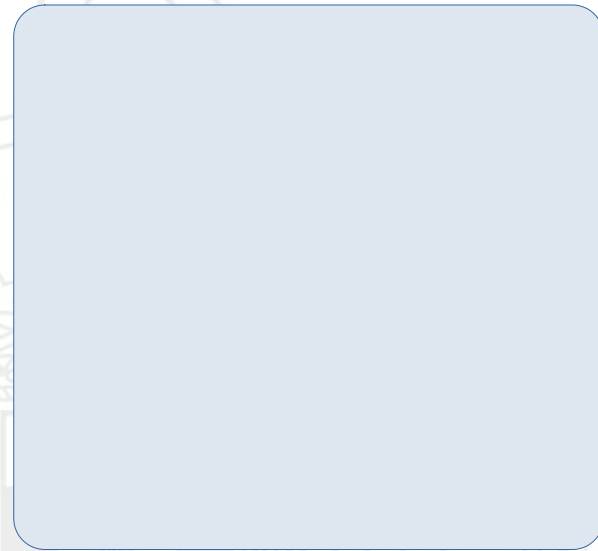
```
int main() {  
    int y = 4;  
    g();  
}
```

es un entorno

Pila



Heap

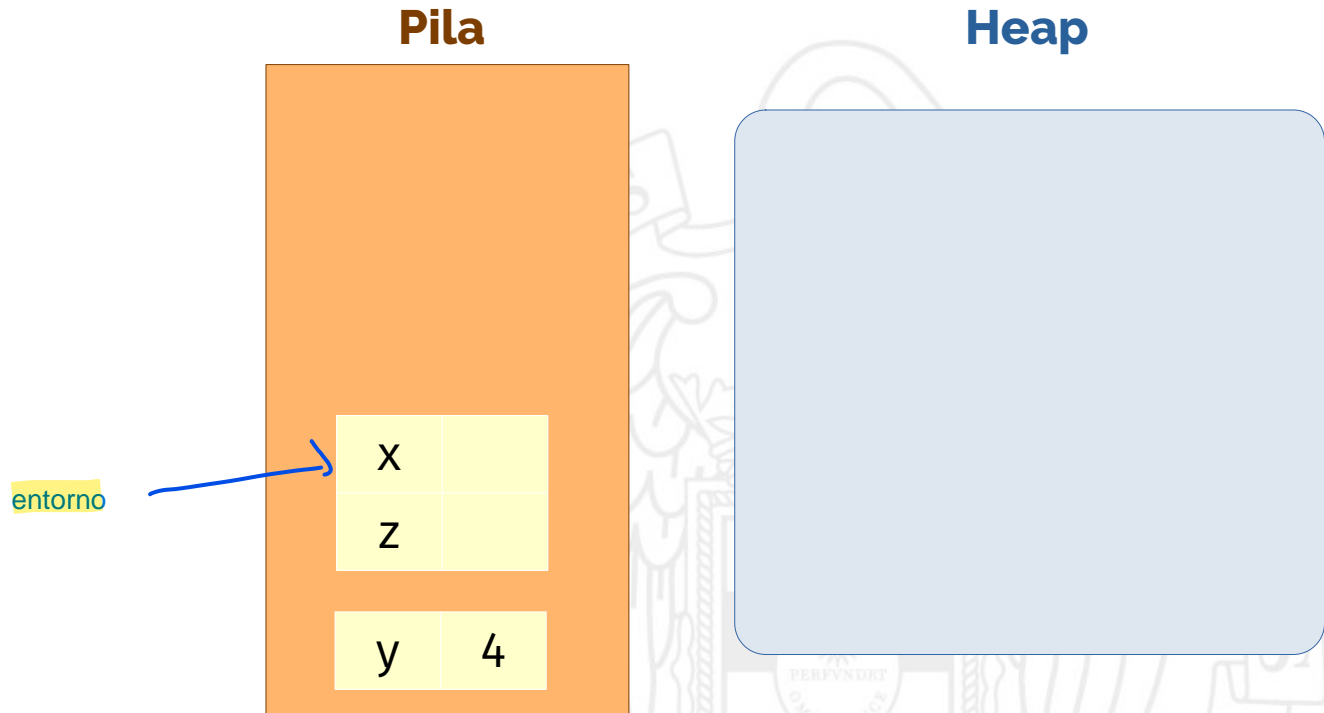


# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```



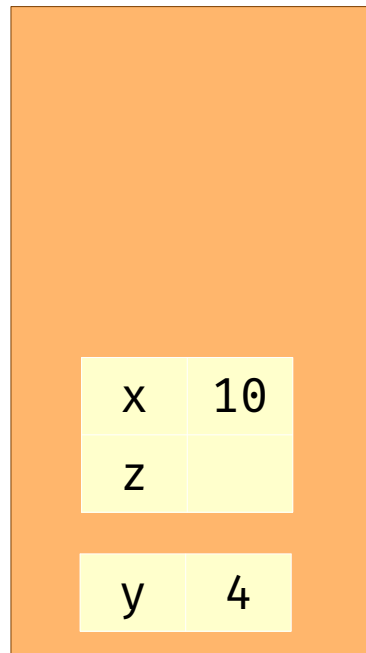
# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

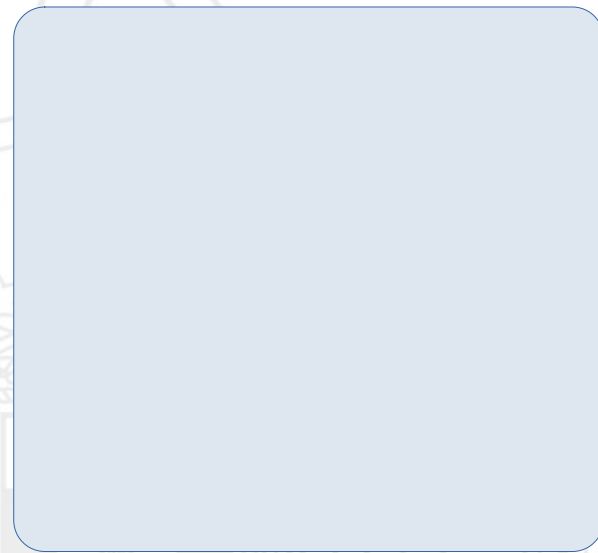
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

**Pila**



**Heap**



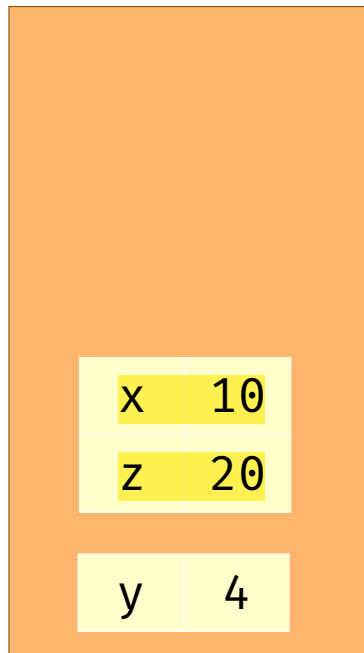
# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

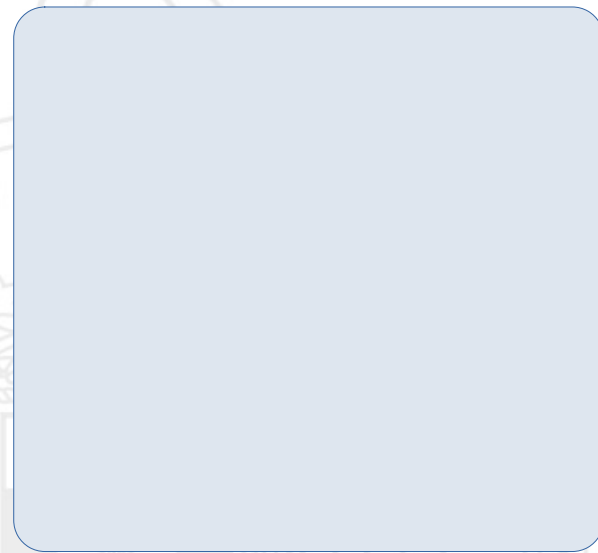
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila



Heap



# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

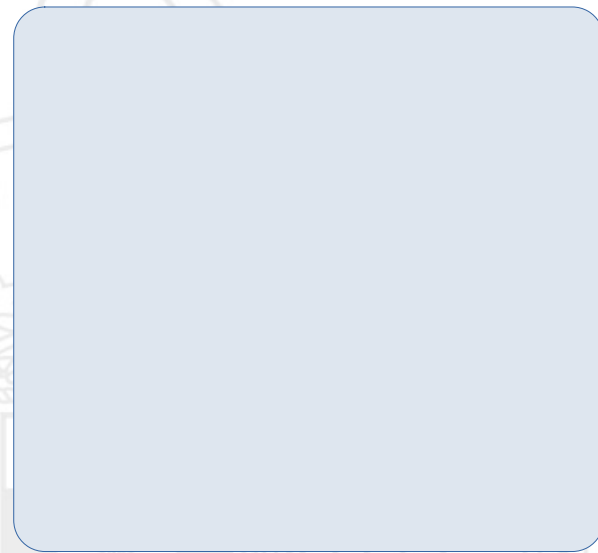
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

**Pila**

x	10
z	20
y	4

**Heap**



# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

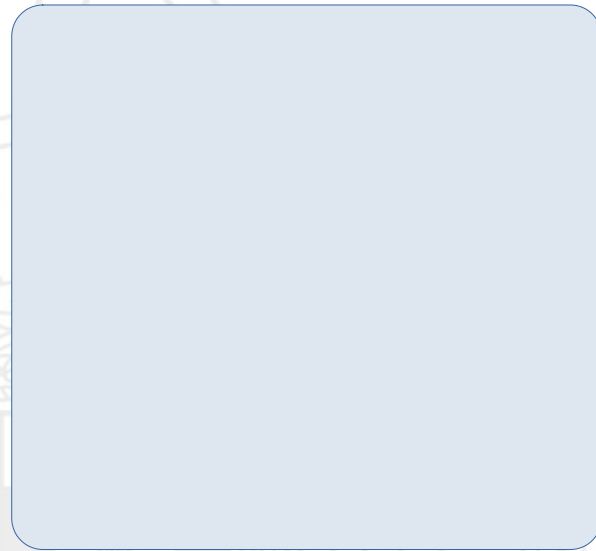
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila

x	30
y	
x	10
z	20
y	4

Heap





# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

otro entorno

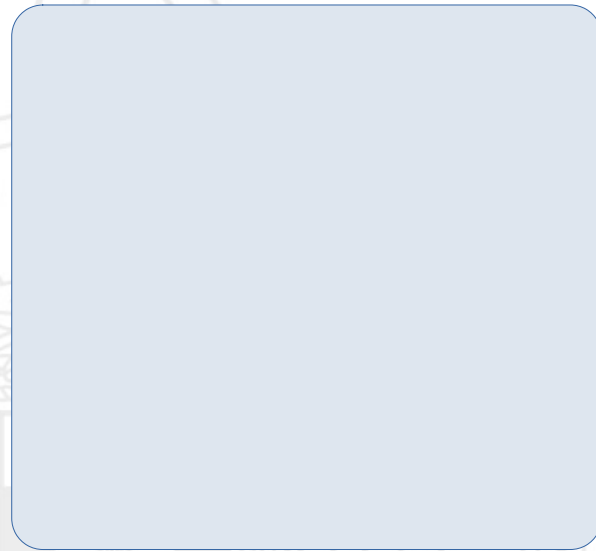
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila

x	30
y	40
x	10
z	20
y	4

Heap



# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

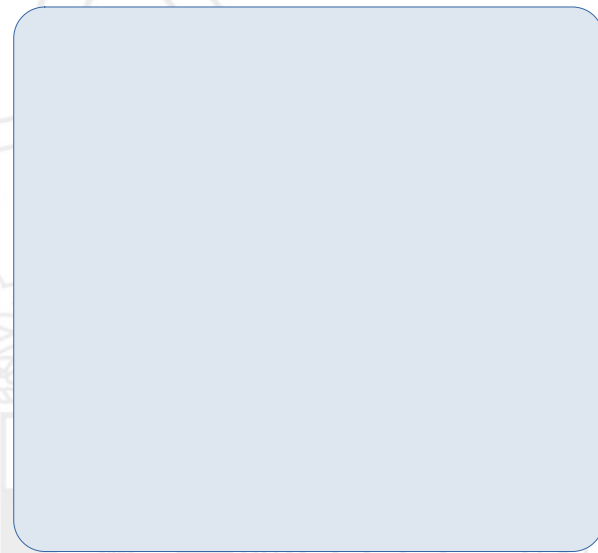
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila

x	30
y	40
x	10
z	20
y	4

Heap



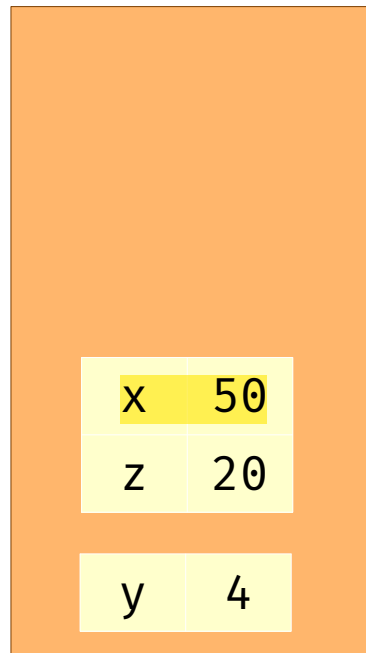
# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

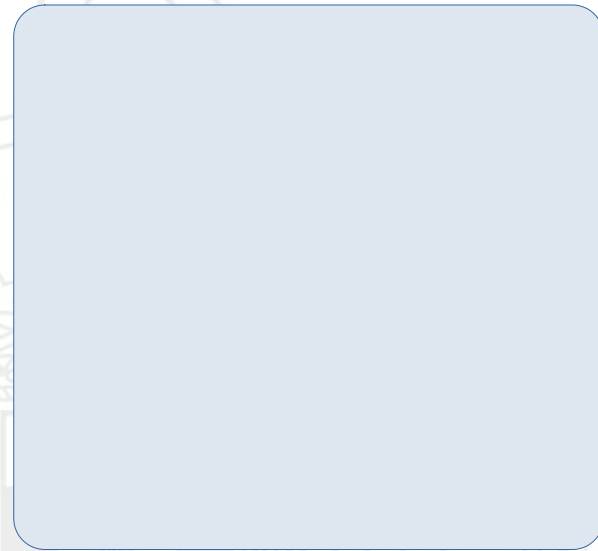
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila



Heap



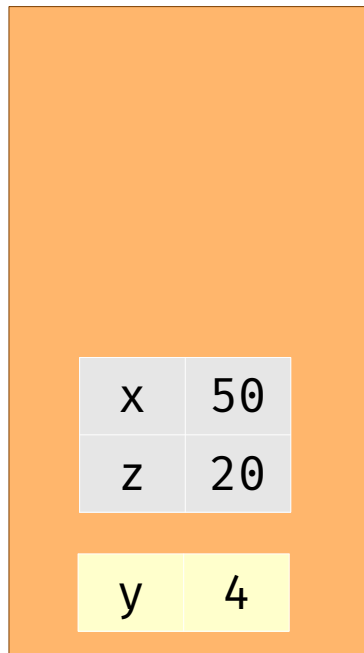
# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

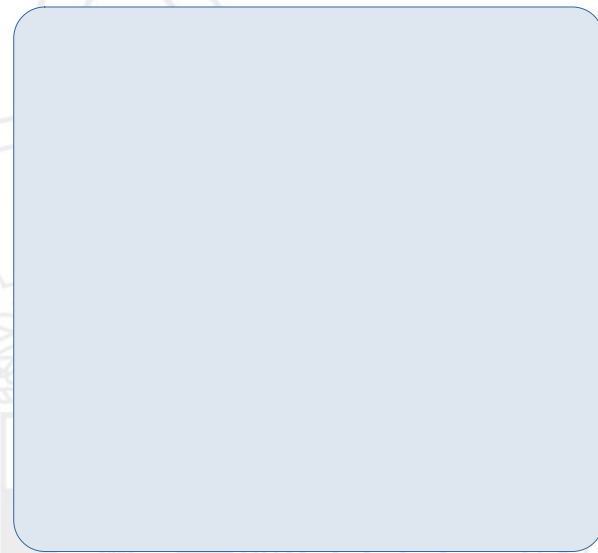
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

**Pila**



**Heap**



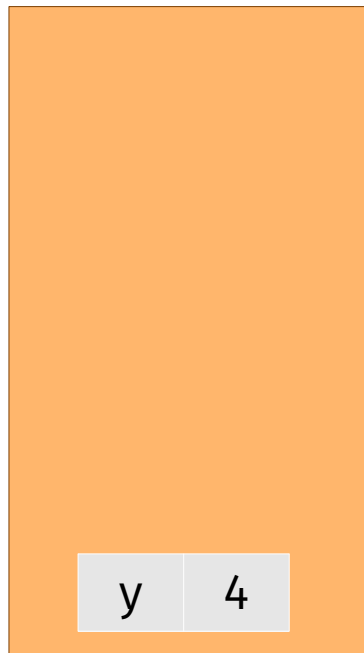
# Ejemplo de ejecución

```
void f(int x) {  
    int y = 40;  
}
```

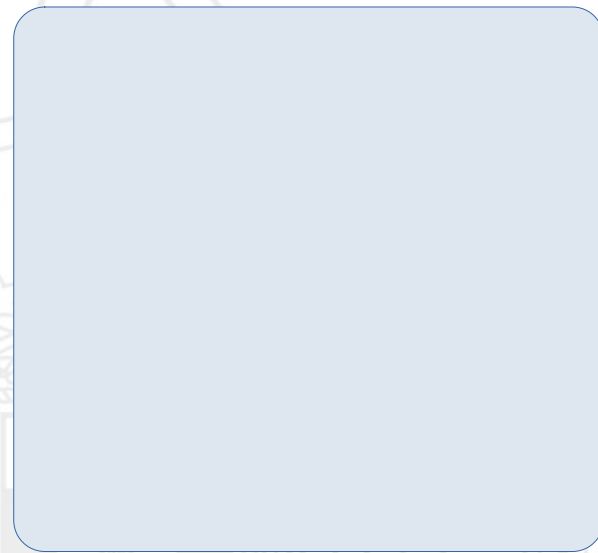
```
void g() {  
    int x = 10;  
    int z = 20;  
    f(30);  
    x = 50;  
}
```

```
int main() {  
    int y = 4;  
    g();  
}
```

Pila



Heap



Los entornos de abajo son funciones que no han acabado aun

Entorno de la cima corresponde con la función que ahora mismo se está ejecutando

En resumen, el anidamiento de funciones se representan con los entornos que se van apilando