Metodología de la Programación

Tema 0. Revisitanto funciones. Paso por referencia ...

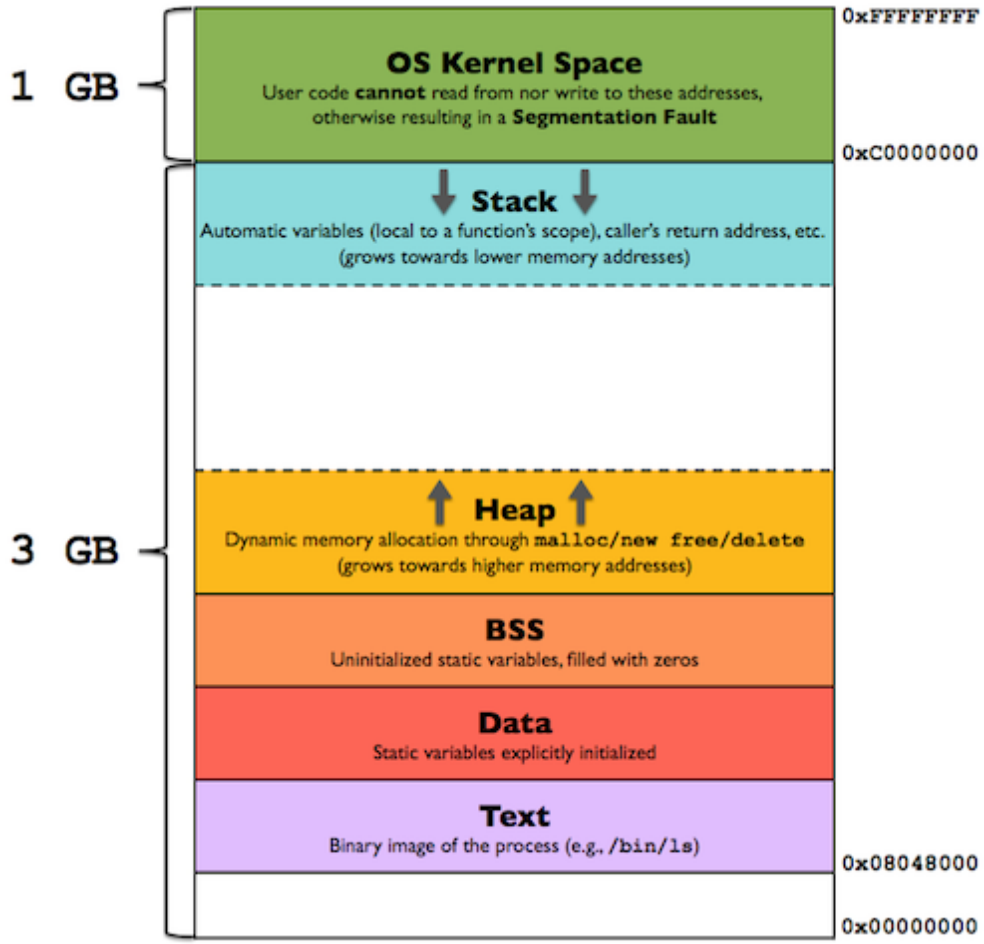
Sylvia Acid (acid@decsai.ugr.es)
derivado de la obra de Andrés Cano
Departamento de Ciencias de la Computación e I.A.

ETSIIT Universidad de Granada

Curso 2025-26

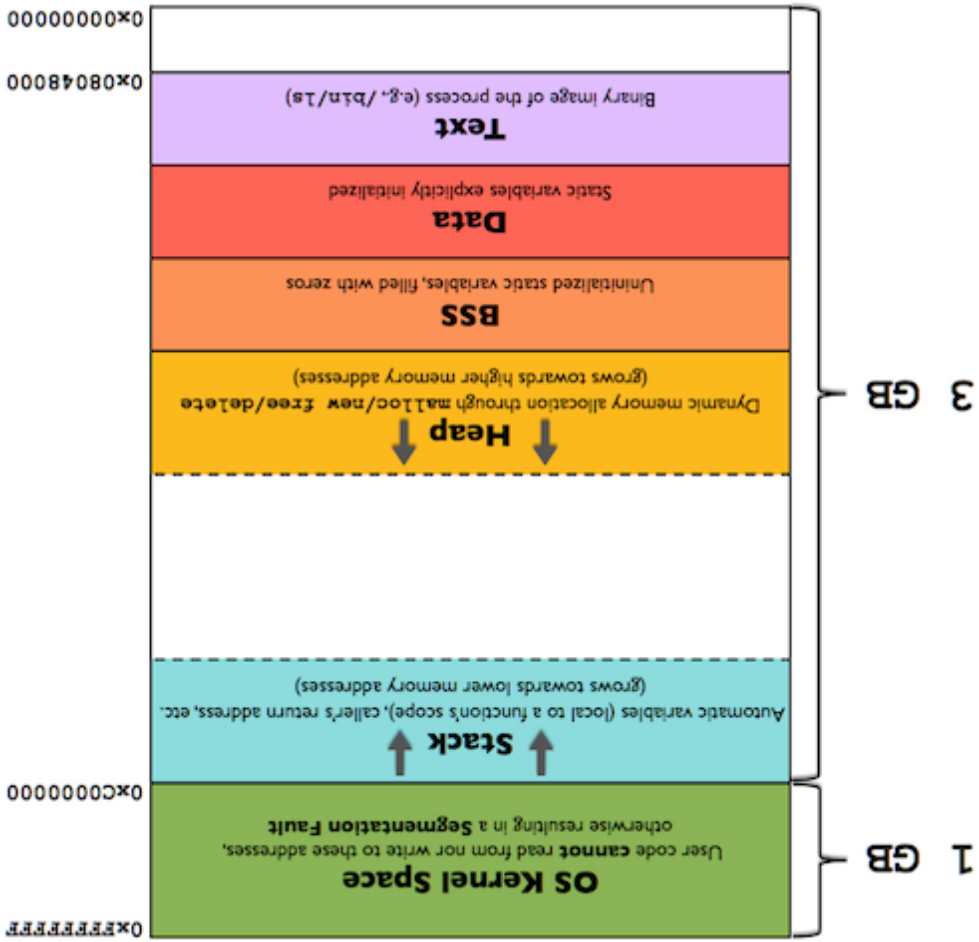
Contenido del tema

Contenido del tema



La ejecución de un programa se lleva a cabo mediante la **Pila (Stack)**.

Pila tiene una gestión LIFO (Last In First Out) de la memoria.



Los registros de las funciones activas se ubican en la **Pila** a modo de pila.

Atendiendo a la forma en que los procesos permanecen activos en memoria y se activan y desactivan, la gestión de la pila es **dinámica** y **automática**.

Contenido del tema

La pila

La pila (stack)

Zona de memoria que almacena información sobre las funciones activas de un programa.

La pila

La pila (stack)

Zona de memoria que almacena información sobre las funciones activas de un programa.

Funciones activas

Aquellas que han sido llamadas pero aún no han terminado su ejecución.

La pila

La pila (stack)

Zona de memoria que almacena información sobre las funciones activas de un programa.

Funciones activas

Aquellas que han sido llamadas pero aún no han terminado su ejecución.

Cuando se invoca a una función:

- se crea en la pila un entorno de programa que almacena la información del módulo:
  - la dirección de memoria de retorno,
  - las constantes y variables locales,
  - los parámetros formales, ...

La pila

La pila (stack)

Zona de memoria que almacena información sobre las funciones activas de un programa.

Funciones activas

Aquellas que han sido llamadas pero aún no han terminado su ejecución.

Cuando se invoca a una función:

- se crea en la pila un entorno de programa que almacena la información del módulo:
  - la dirección de memoria de retorno,
  - las constantes y variables locales,
  - los parámetros formales, ...
- Al terminar la ejecución del módulo se destruye su entorno de programa asociado.

Ejecución de un programa en C++

- La ejecución de un programa en C++ empieza creando un entorno de programa en el fondo de la pila para `main()`.

Ejecución de un programa en C++

- La ejecución de un programa en C++ empieza creando un entorno de programa en el fondo de la pila para `main()`.
- `main()`
  1. es una función que debe aparecer en todo programa ejecutable escrito en C++.

Ejecución de un programa en C++

- La ejecución de un programa en C++ empieza creando un entorno de programa en el fondo de la pila para `main()`.
- `main()`
  1. es una función que debe aparecer en todo programa ejecutable escrito en C++.
  2. presenta distintas versiones en cuanto a sus parámetros.

```
int main()                                    // de momento
int main(int argc, char *argv[])
```

Ejecución de un programa en C++

- La ejecución de un programa en C++ empieza creando un entorno de programa en el fondo de la pila para `main()`.
- `main()`
  1. es una función que debe aparecer en todo programa ejecutable escrito en C++.
  2. presenta distintas versiones en cuanto a sus parámetros.

     ```
     int main()                                    // de momento
     int main(int argc, char *argv[])
     ```
  3. devuelve un dato entero al sistema operativo.

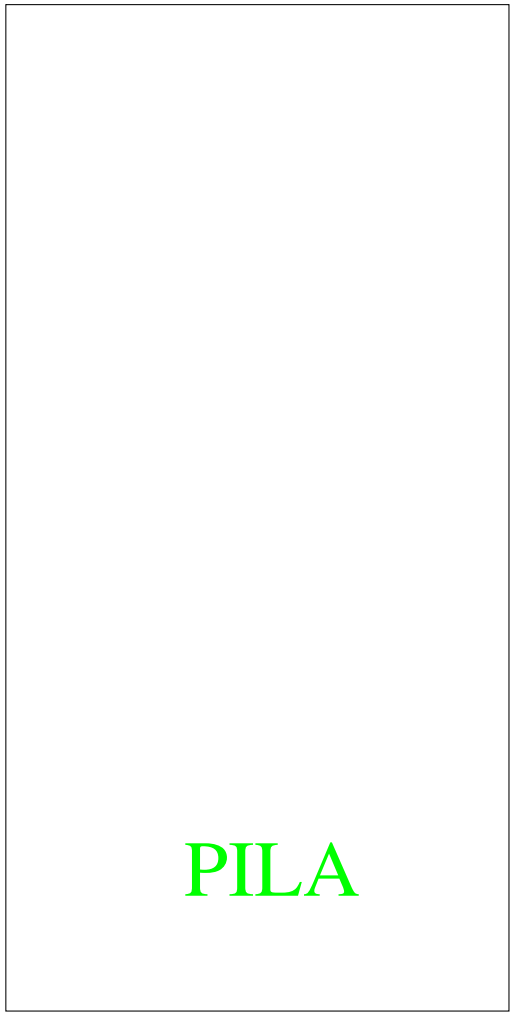
Ejecución de un programa en C++

- La ejecución de un programa en C++ empieza creando un entorno de programa en el fondo de la pila para `main()`.
- `main()`
  1. es una función que debe aparecer en todo programa ejecutable escrito en C++.
  2. presenta distintas versiones en cuanto a sus parámetros.

     ```
     int main()                                    // de momento
     int main(int argc, char *argv[])
     ```
  3. devuelve un dato entero al sistema operativo.
- Un programa termina cuando se desapila el entorno de programa asociado a `main()` de la pila.

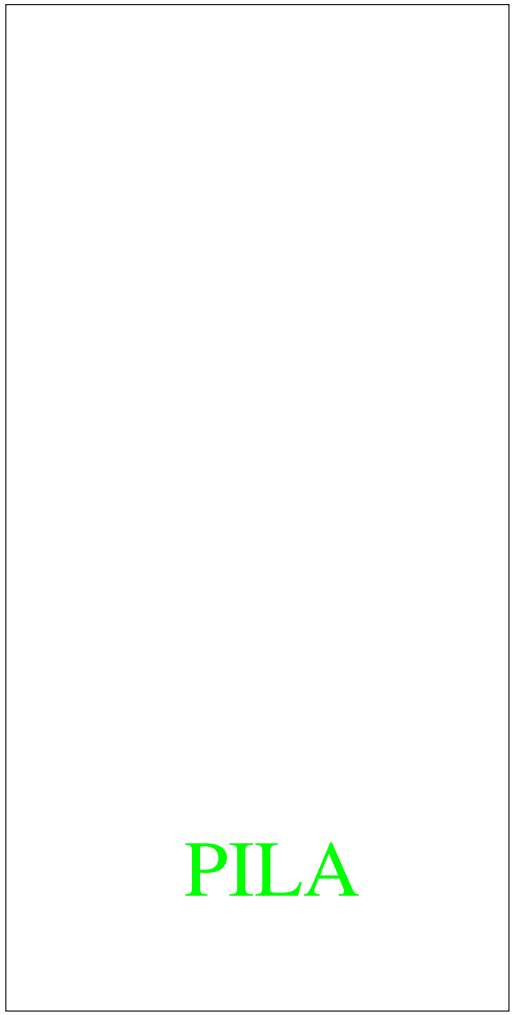
Ejemplo

```
1 int main(){
2   int valor;
3   cout << "\nIntroduce "
4   <<"un entero positivo: ";
5   cin >> valor;
6   ImprimeFactorial (valor);
7   Pausa();
8 }
```



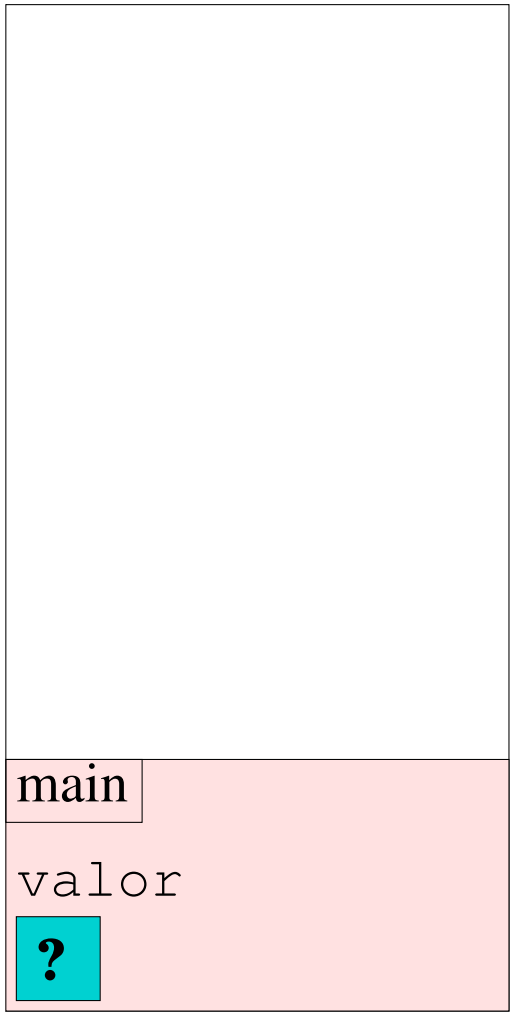
Ejemplo

```
1 int main(){
2   int valor;
3   cout << "\nIntroduce "
4   <<"un entero positivo: ";
5   cin >> valor;
6   ImprimeFactorial (valor);
7   Pausa();
8 }
```



Ejemplo

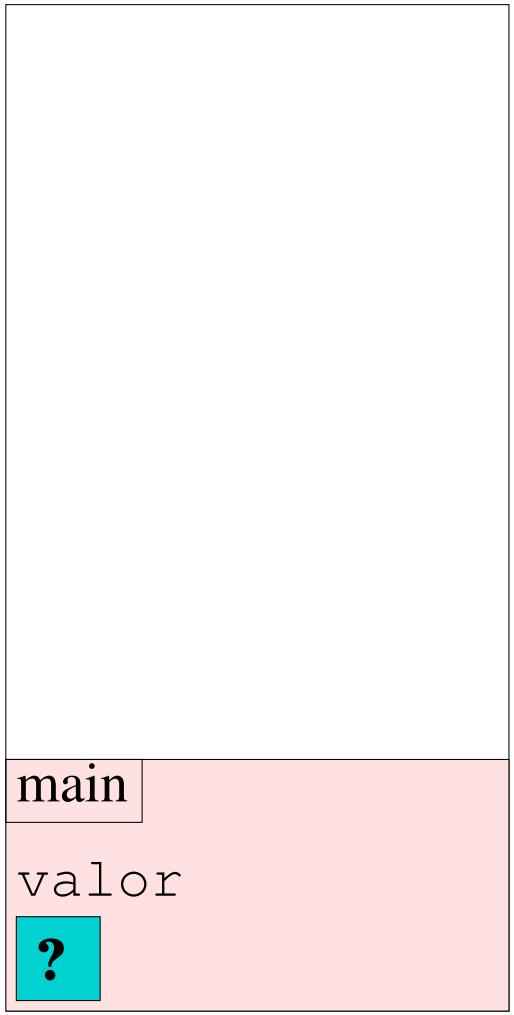
```
1 int main(){
2   int valor;
3   cout << "\nIntroduce "
4   <<"un entero positivo: ";
5   cin >> valor;
6   ImprimeFactorial (valor);
7   Pausa();
8 }
```



Ejemplo

```
1 int main(){
2   int valor;
3   cout << "\nIntroduce "
4   <<"un entero positivo: ";
5   cin >> valor;
6   ImprimeFactorial (valor);
7   Pausa();
8 }
```

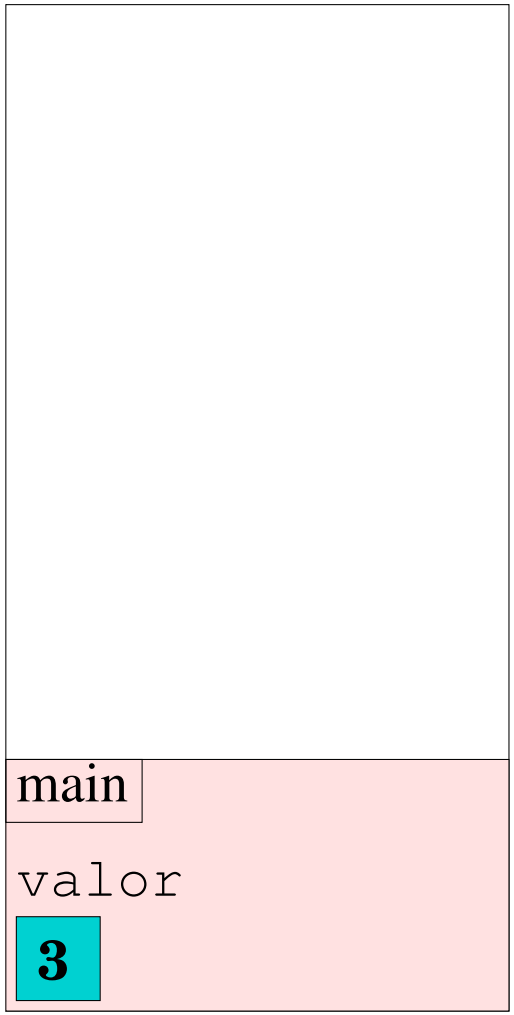
```
Introduce un entero positivo:
```



Ejemplo

```
1 int main(){
2   int valor;
3   cout << "\nIntroduce "
4   <<"un entero positivo: ";
5   cin >> valor;
6   ImprimeFactorial (valor);
7   Pausa();
8 }
```

```
Introduce un entero positivo: 3
```

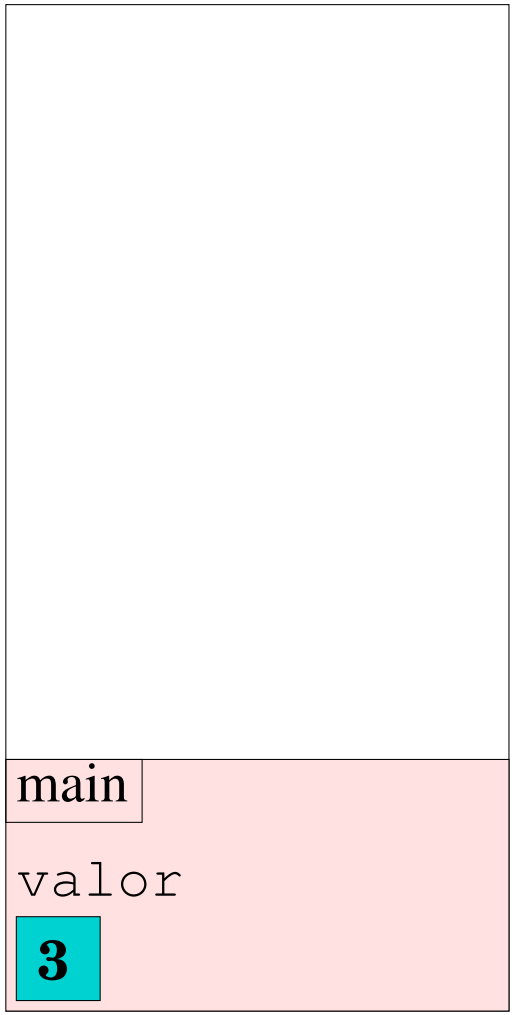


Ejemplo

ImprimeFactorial(valor)

```
1 void ImprimeFactorial (int n){
2   int resul;
3
4   resul = Factorial(n);
5   cout << "\nEl factorial de "
6        << n <<" es "<< resul
7        << endl;
8 }
```

```
Introduce un entero positivo: 3
```

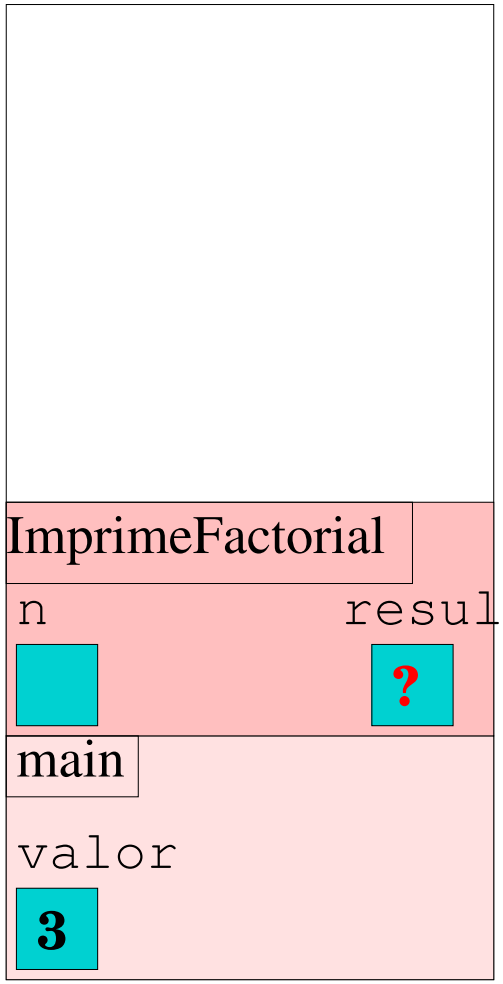


Ejemplo

`ImprimeFactorial(valor)`

```
1 void ImprimeFactorial (int n){
2   int resul;
3
4   resul = Factorial(n);
5   cout << "\nEl factorial de "
6        << n <<" es "<< resul
7        << endl;
8 }
```

```
Introduce un entero positivo: 3
```

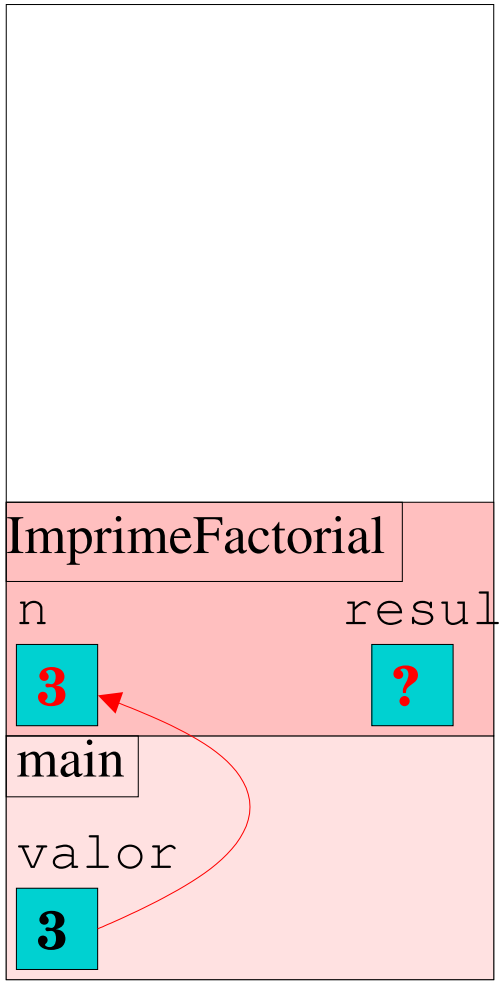


Ejemplo

`ImprimeFactorial(valor)`

```
1 void ImprimeFactorial (int n){
2   int resul;
3
4   resul = Factorial(n);
5   cout << "\nEl factorial de "
6        << n <<" es "<< resul
7        << endl;
8 }
```

```
Introduce un entero positivo: 3
```

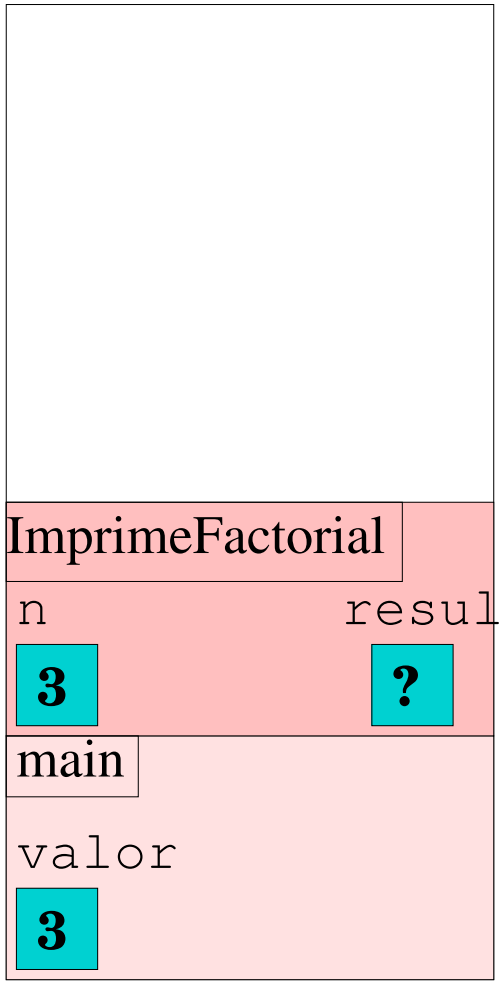


Ejemplo

`ImprimeFactorial(valor)`

```
1 void ImprimeFactorial (int n){
2   int resul;
3
4   resul = Factorial(n);
5   cout << "\nEl factorial de "
6        << n <<" es "<< resul
7        << endl;
8 }
```

```
Introduce un entero positivo: 3
```

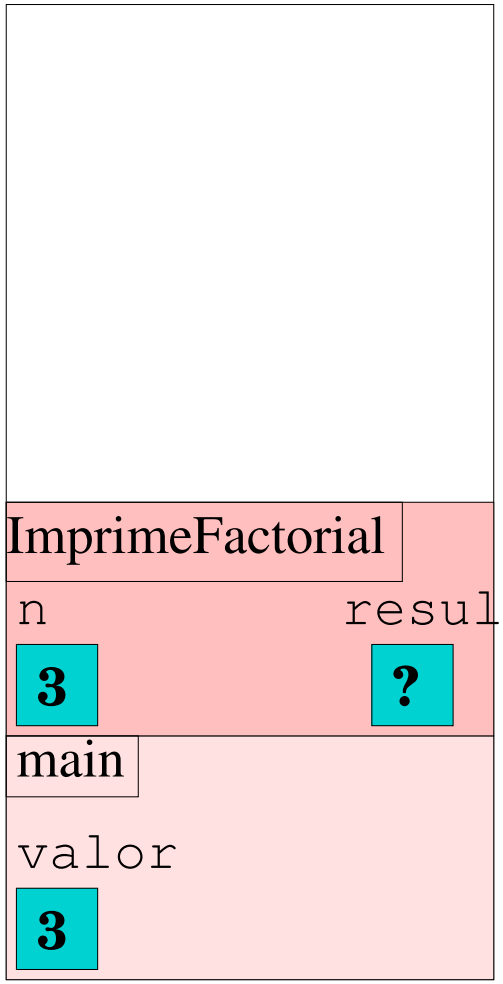


Ejemplo

```
result = Factorial (n)
```

```
1 int Factorial (int n){
2   int i, valor=1;
3
4   for (i=2; i<=n; i++)
5     valor=valor*i;
6
7   return valor;
8 }
```

```
Introduce un entero positivo: 3
```

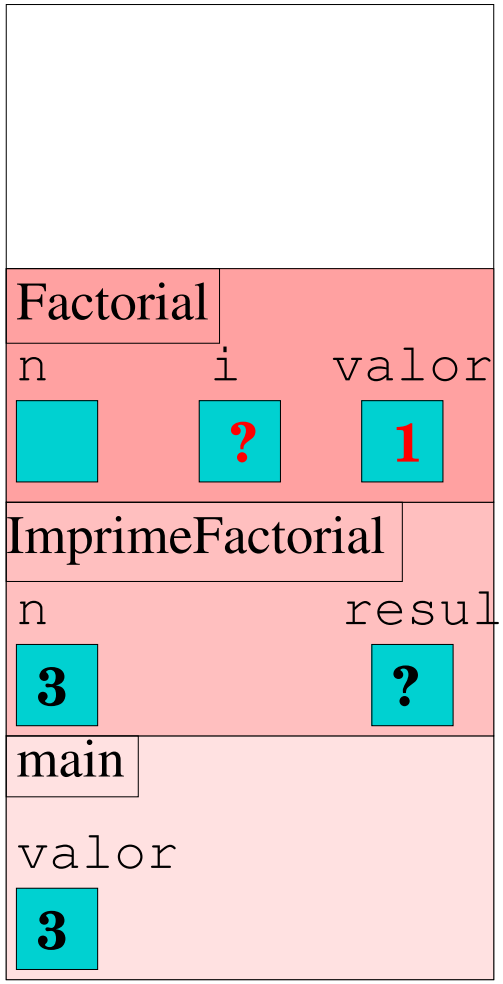


Ejemplo

```
result = Factorial (n)
```

```
1 int Factorial (int n){
2   int i, valor=1;
3
4   for (i=2; i<=n; i++)
5     valor=valor*i;
6
7   return valor;
8 }
```

```
Introduce un entero positivo: 3
```

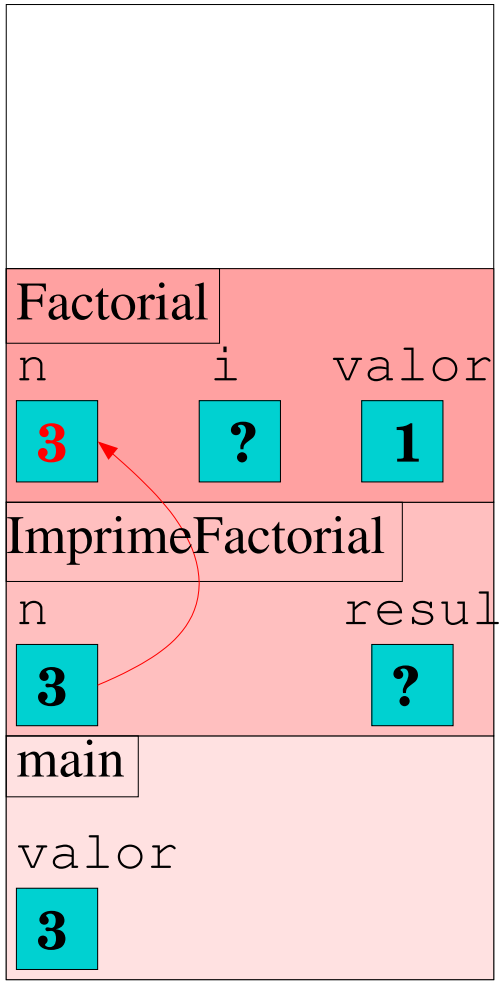


Ejemplo

```
result = Factorial (n)
```

```
1 int Factorial (int n){
2   int i, valor=1;
3
4   for (i=2; i<=n; i++)
5     valor=valor*i;
6
7   return valor;
8 }
```

```
Introduce un entero positivo: 3
```

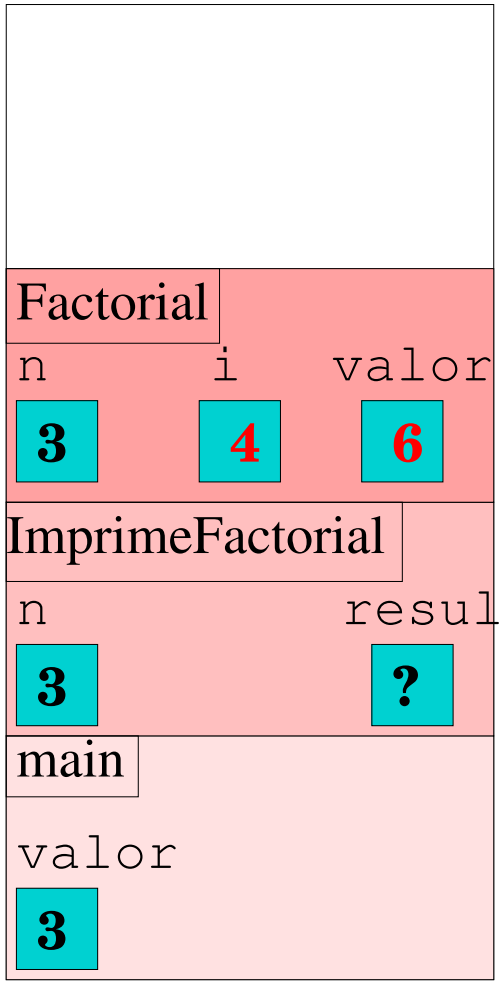


Ejemplo

```
result = Factorial (n)
```

```
1 int Factorial (int n){
2   int i, valor=1;
3
4   for (i=2; i<=n; i++)
5     valor=valor*i;
6
7   return valor;
8 }
```

```
Introduce un entero positivo: 3
```

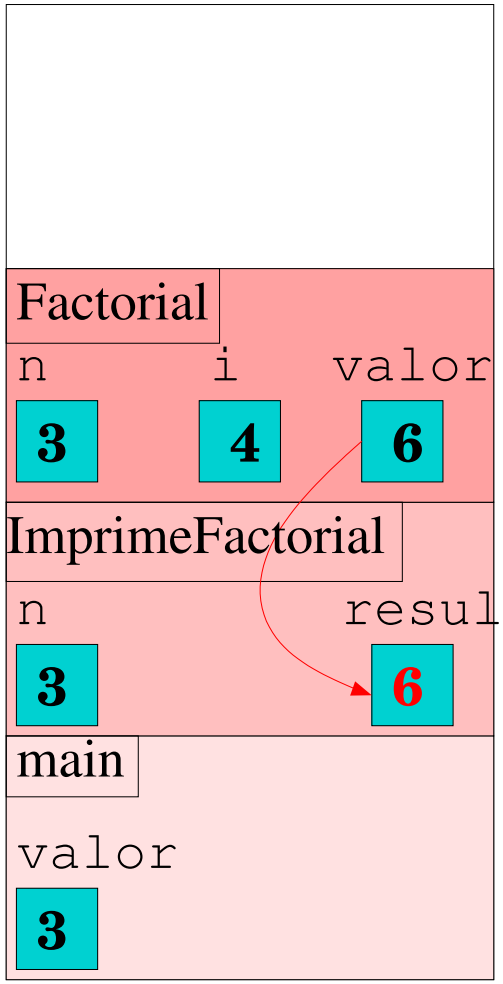


Ejemplo

```
result = Factorial (n)
```

```
1 int Factorial (int n){
2   int i, valor=1;
3
4   for (i=2; i<=n; i++)
5     valor=valor*i;
6
7   return valor;
8 }
```

```
Introduce un entero positivo: 3
```

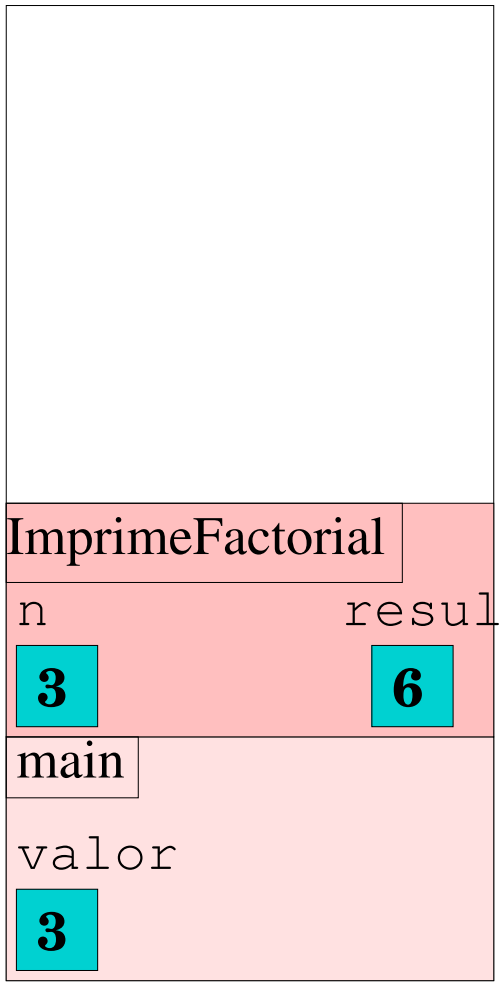


Ejemplo

```
result = Factorial (n)
```

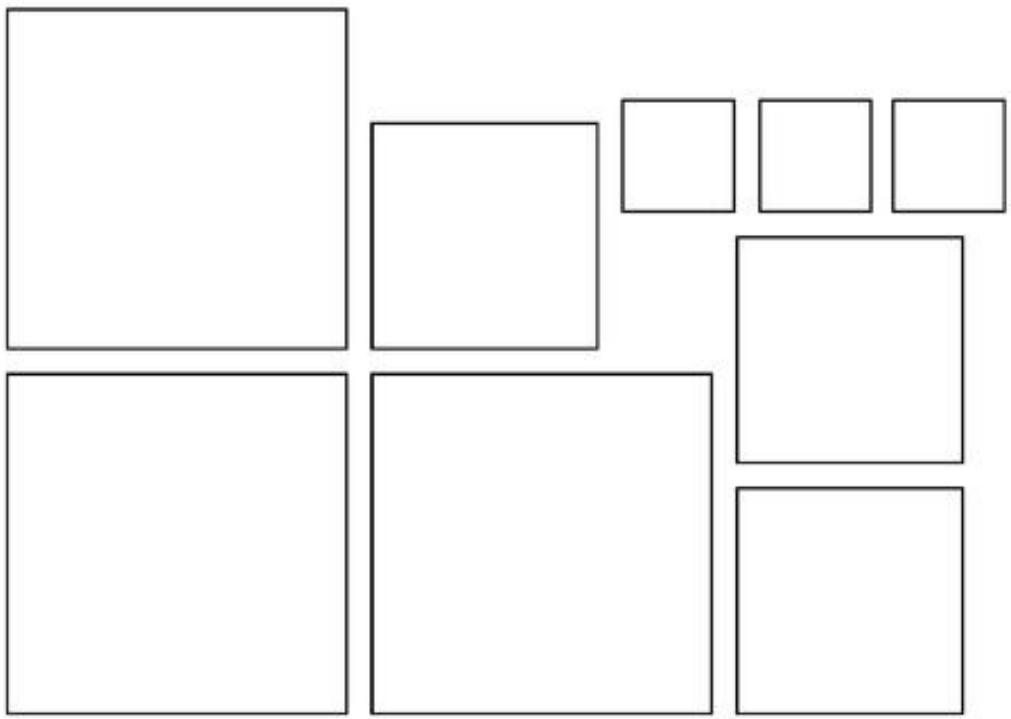
```
1 int Factorial (int n){
2   int i, valor=1;
3
4   for (i=2; i<=n; i++)
5     valor=valor*i;
6
7   return valor;
8 }
```

```
Introduce un entero positivo: 3
```



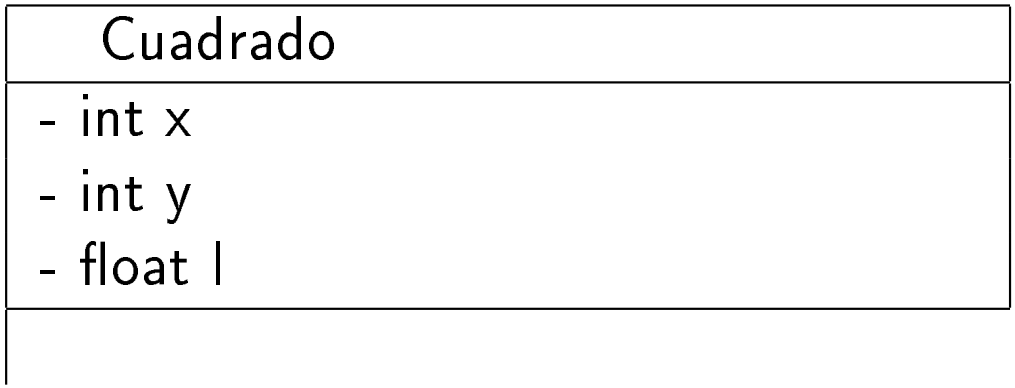
Ahora piensa en objetos

Para representar objetos de tipo cuadrado, situados en el espacio, de los que se quiere calcular área y perímetro, realizar comparaciones (relación de orden) entre 2 cuadrados, comprobar si uno está dentro de otro,...



Piensa en el diseño de la clase: Campos y métodos.

Diseño de la clase con UML



Diseño de la clase con UML

Cuadrado
- int x - int y - float l
- **bool correcto (float)()** + **Cuadrado()** + **Cuadrado(int, int, float)()** + float **area()** + float **perimetro()**

Diseño de la clase con UML

Cuadrado
- int x - int y - float l
- **bool correcto (float)**() + **Cuadrado**() + **Cuadrado(int, int, float)**() + float **area**() + float **perimetro**() + int **getX**() + int **getY**() + float **getL**() + void **setCuadrado**(int, int, float)

Diseño de la clase con UML

Cuadrado
- int x - int y - float l
- **bool correcto (float)**() + **Cuadrado**() + **Cuadrado(int, int, float)**() + float **area**() + float **perimetro**() + int **getX**() + int **getY**() + float **getL**() + void **setCuadrado**(int, int, float) + bool **mayorQ**(Cuadrado) + bool **contenidoEn**(Cuadrado)

Ejemplo: clase Cuadrado I

```
1  class Cuadrado {
2  private:
3    int x;
4    int y;
5    float l;
6  bool correcto( float _l){
7     return _l>0;
8   }
9  public:
10   Cuadrado()
11     :x(0), y(0), l(1){}
12   Cuadrado(int _x,int _y, float _l)
13     :x(_x), y(_y), l(1){
14     if (correcto(_l))
15         l = _l;
16   }
```

Ejemplo: clase Cuadrado II

```
17   float area(){
18    return l*l;
19   }
20   float perimetro(){
21    return 4*l;
22   }
23   int getX(){
24    return x;
25   }
26   int getY(){
27    return y;
28   }
29   float getL(){
30    return l;
31   }
```

Ejemplo: clase Cuadrado III

```
32
```

```
1   bool mayorQ(Cuadrado c){
2   bool mas = false;
3   if (area() > c.area())
4       mas=true;
5   return mas;
6   }
7   bool contenidoEn(Cuadrado c){
8   bool cabe = false;
9       if ((x <= c.getX()) && ((x+l) >= (c.getX()+ c.getL())) &&
10          (y >= c.getY()) && ((y-l) <= (c.getY()+ c.getL())))
11          cabe = true;
12   return cabe;
```

Ejemplo: clase Cuadrado IV

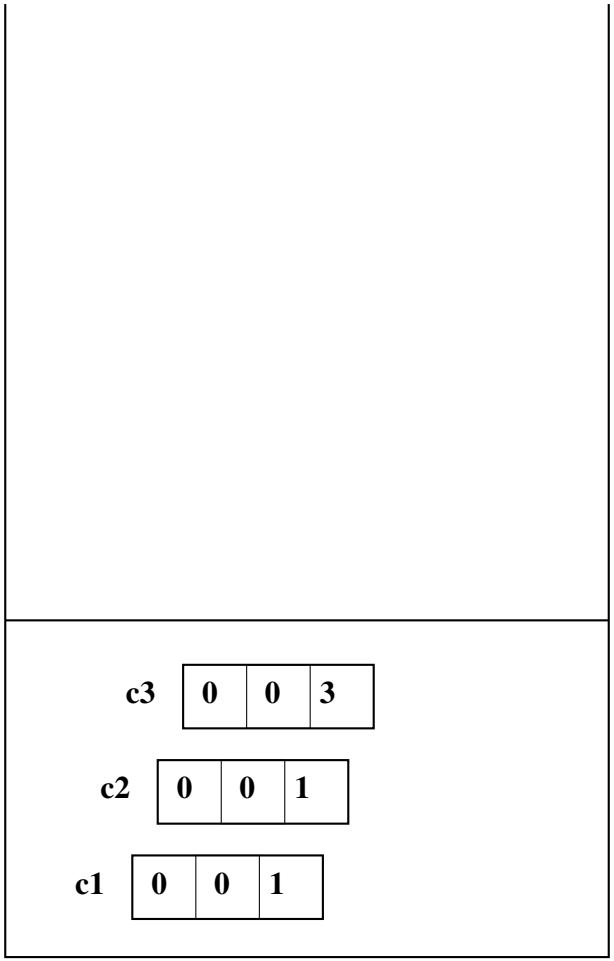
```
13   }
14 };
15
16 int main(){
17 Cuadrado c1;          // Ctor1 sin parametros
18 Cuadrado c2(0,0,3);// Ctor2 con parametros
19 Cuadrado c3(c2);     // Ctor de copia xDEFECTO
20 // Cuadrado c3;
21 // c3 = c2;           // asignacion entre objetos
22 c2.set(0,0,1);
23
24 if (c2.mayorQ(c3))
25     cout << "c2 > c3" << endl;
26 else cout << "c3 > c2" << endl;
27 cout << "c2.contenidoEn(c3) " << c2.contenidoEn(c3);
```

Ejemplo: clase Cuadrado V

```
28 cout << "c3.contenidoEn(c2) " << c3.contenidoEn(c2);
29 cout<< endl;
30  /*  ... */
31 cout<< endl;
32 }
```

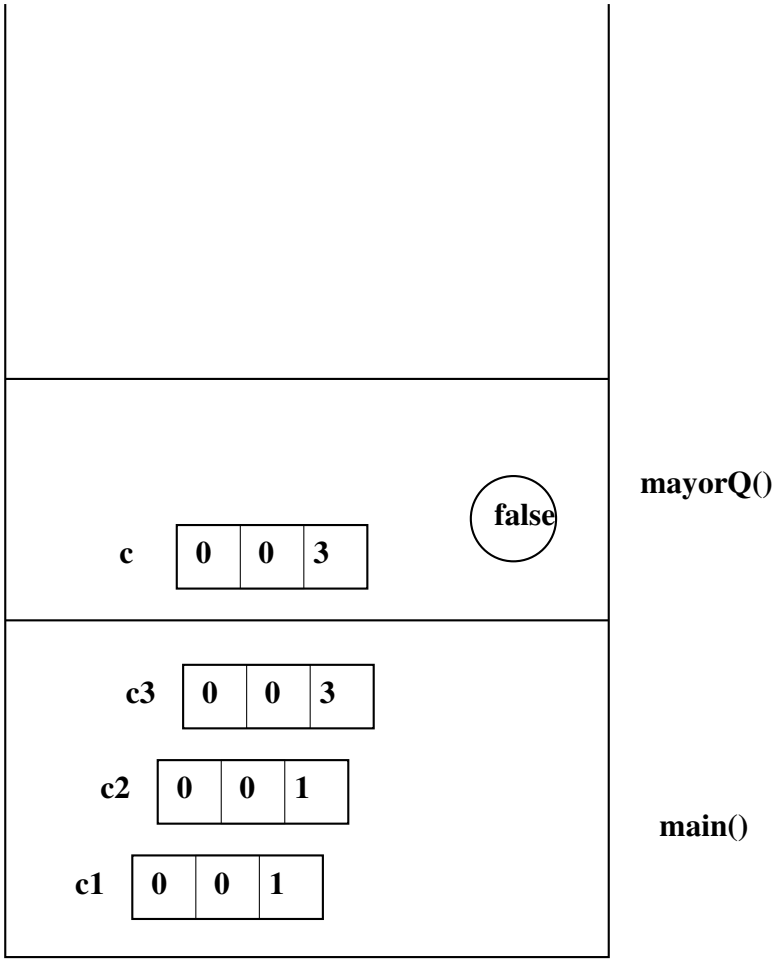
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



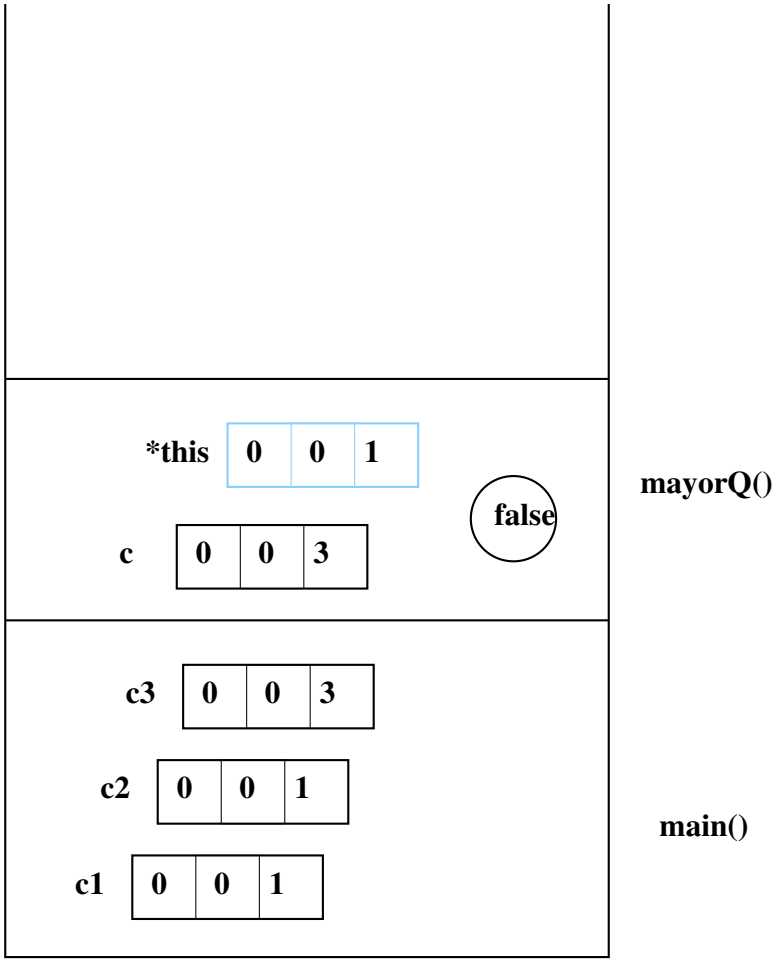
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



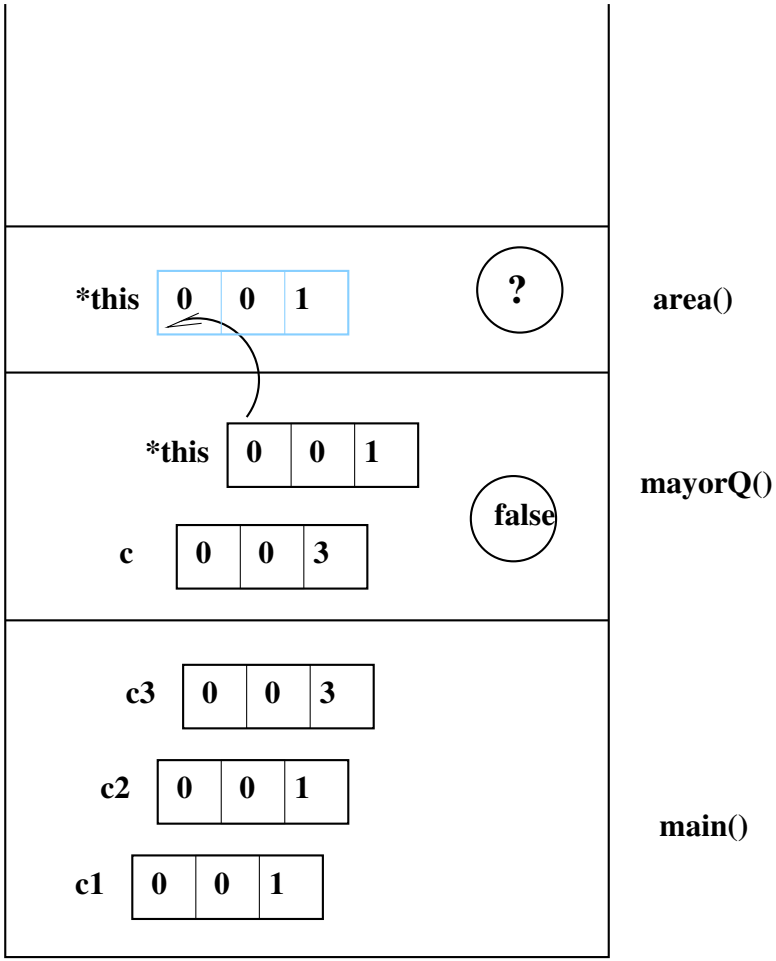
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



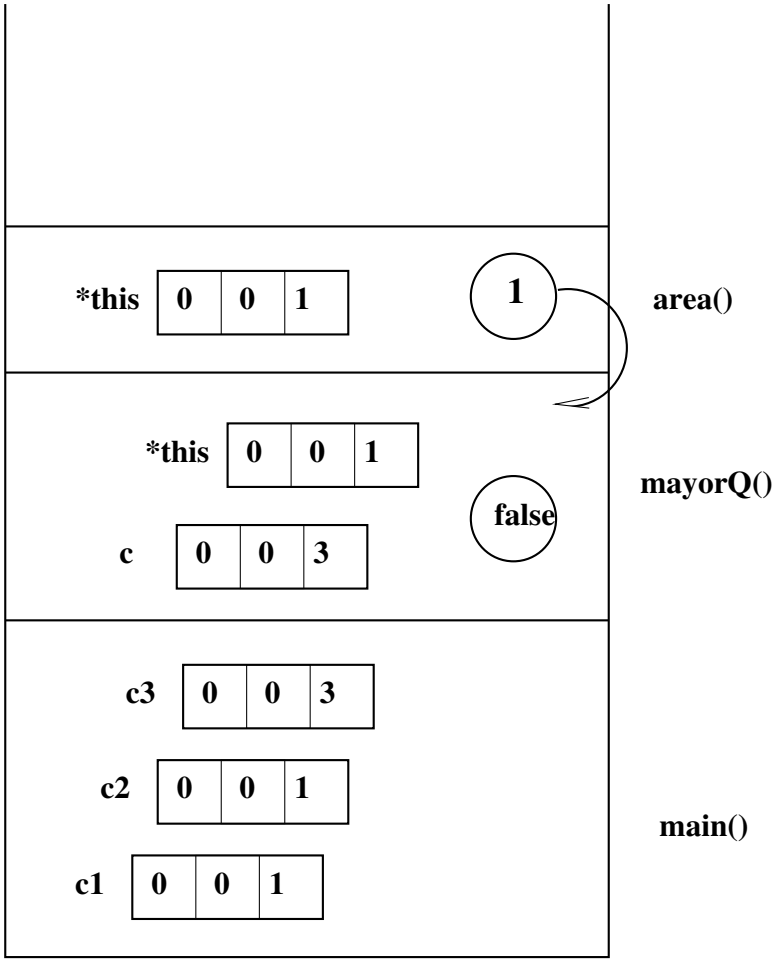
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



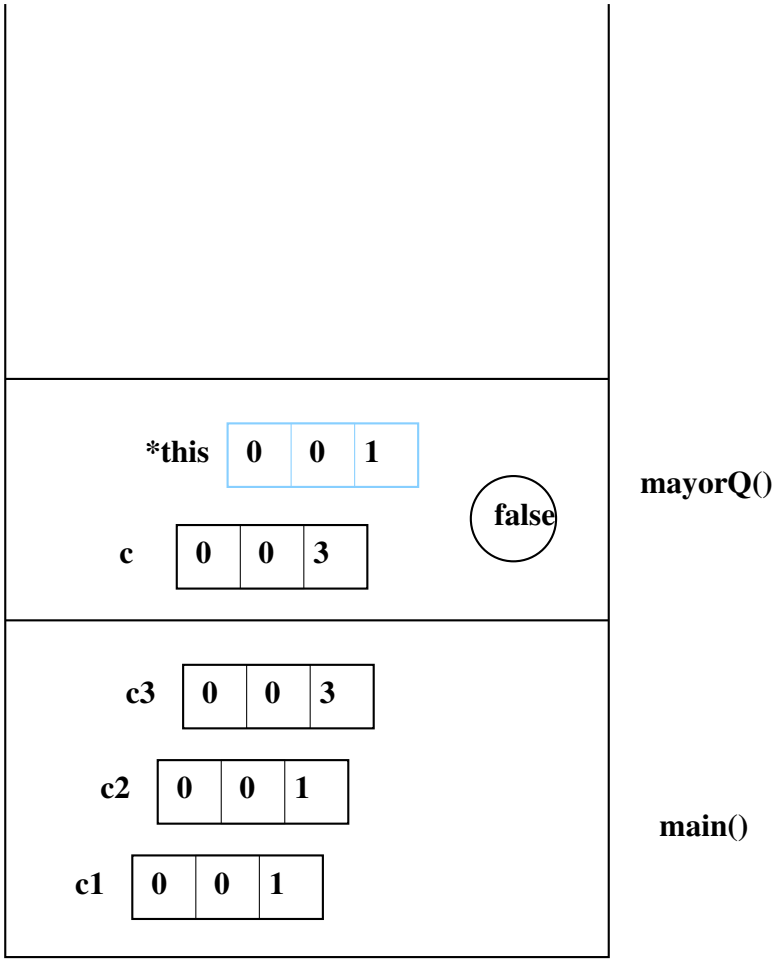
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



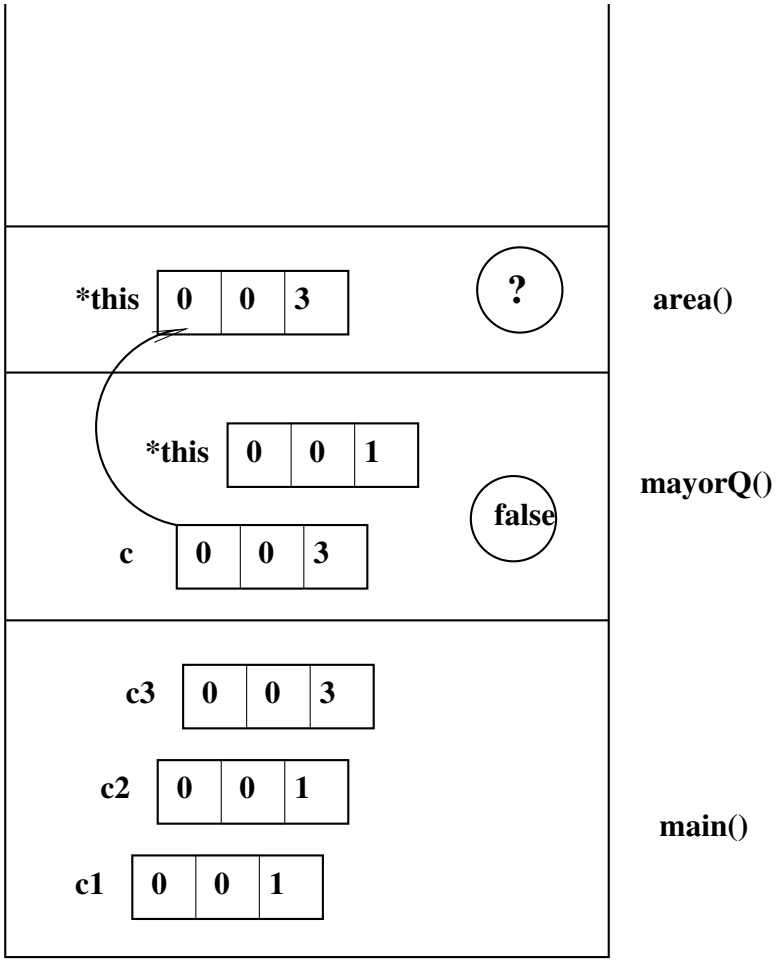
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



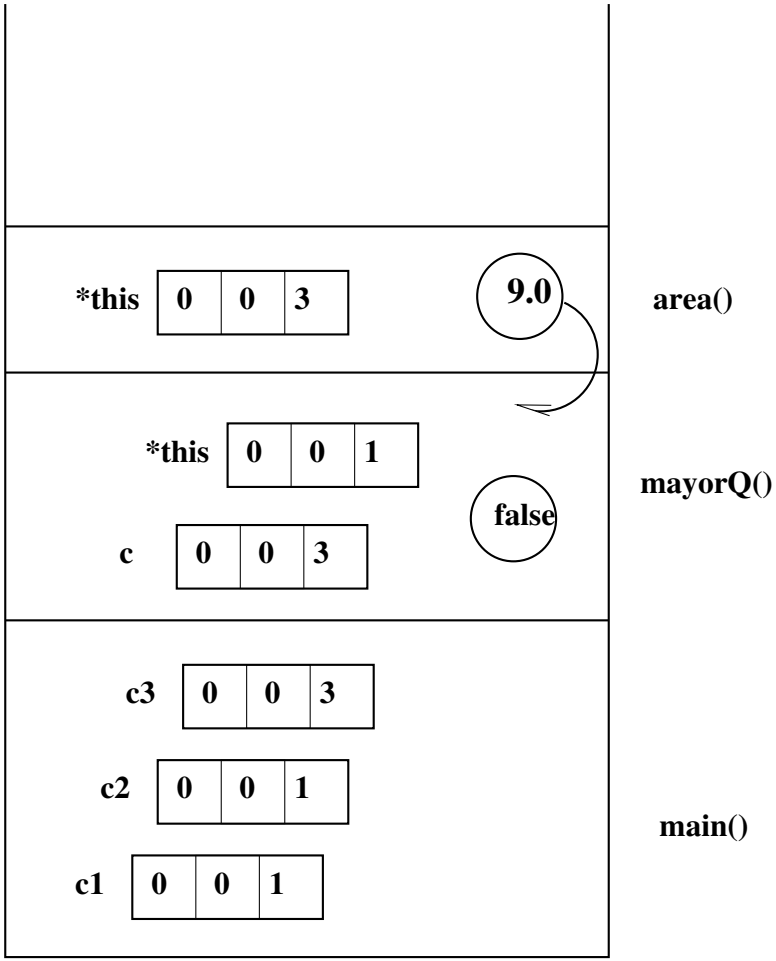
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



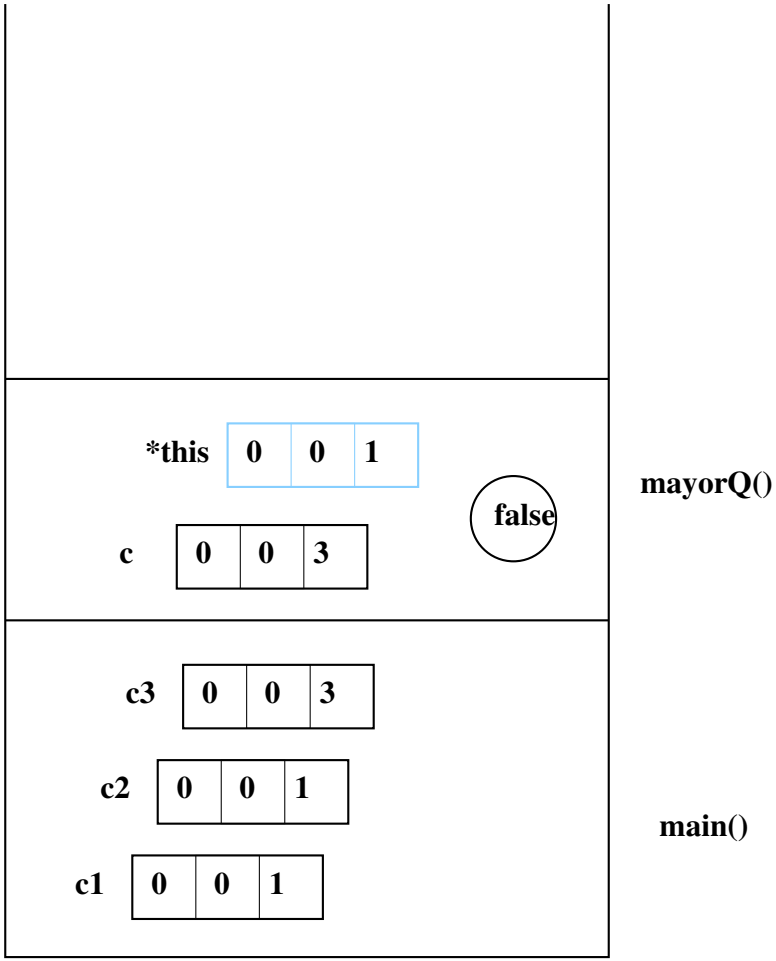
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



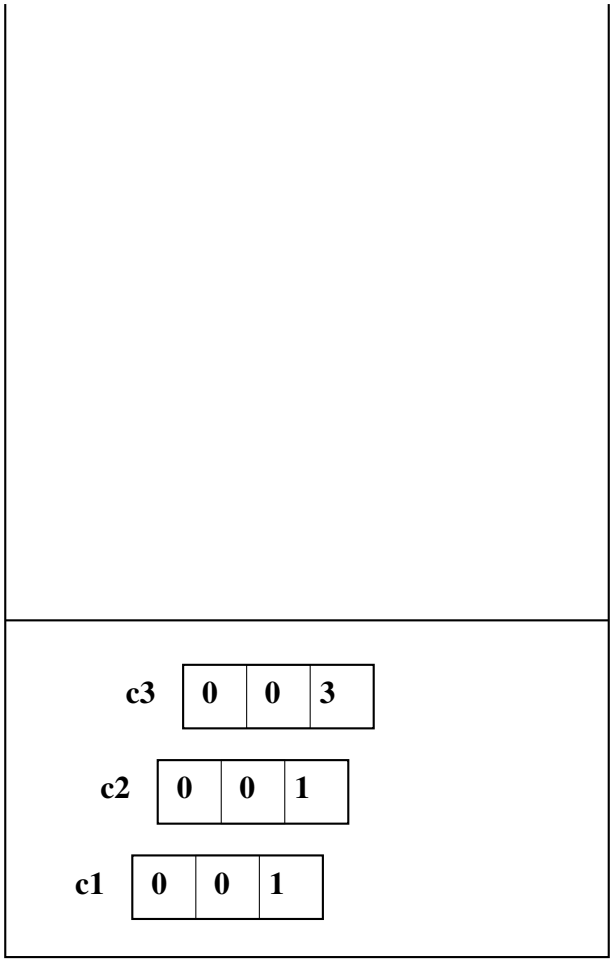
Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



Los métodos y la pila

Contenido de la pila en la instrucción *if*(*c2.mayorQ*(*c3*))... de *main*()



Contenido del tema

Ámbito de un dato

El ámbito de un dato

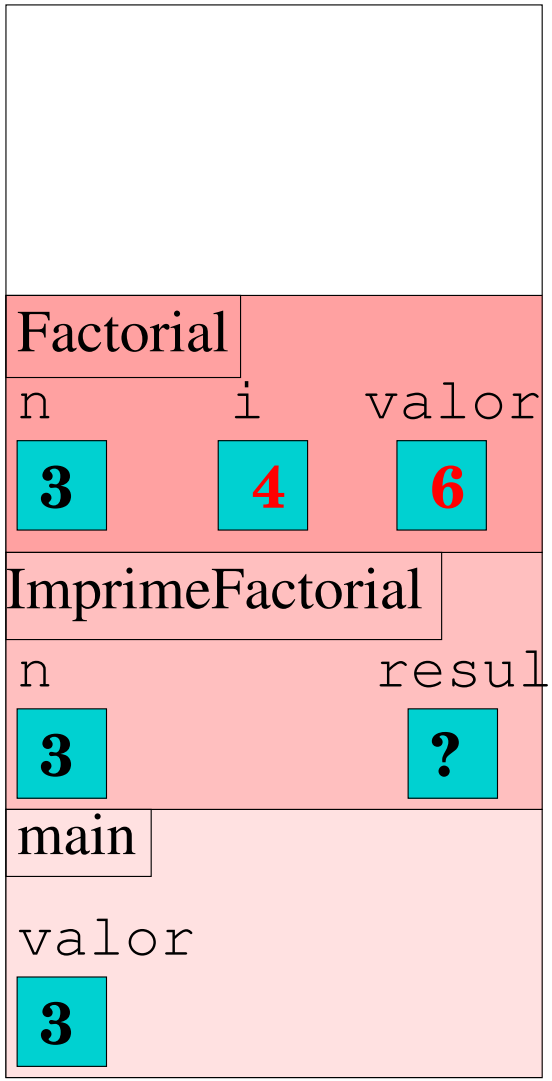
es el conjunto de todos aquellos módulos que lo pueden referenciar.

Ámbito de un dato

El ámbito de un dato

es el conjunto de todos aquellos módulos que lo pueden referenciar.

¿Cuál es el ámbito de cada uno de los datos que aparecen en la figura?



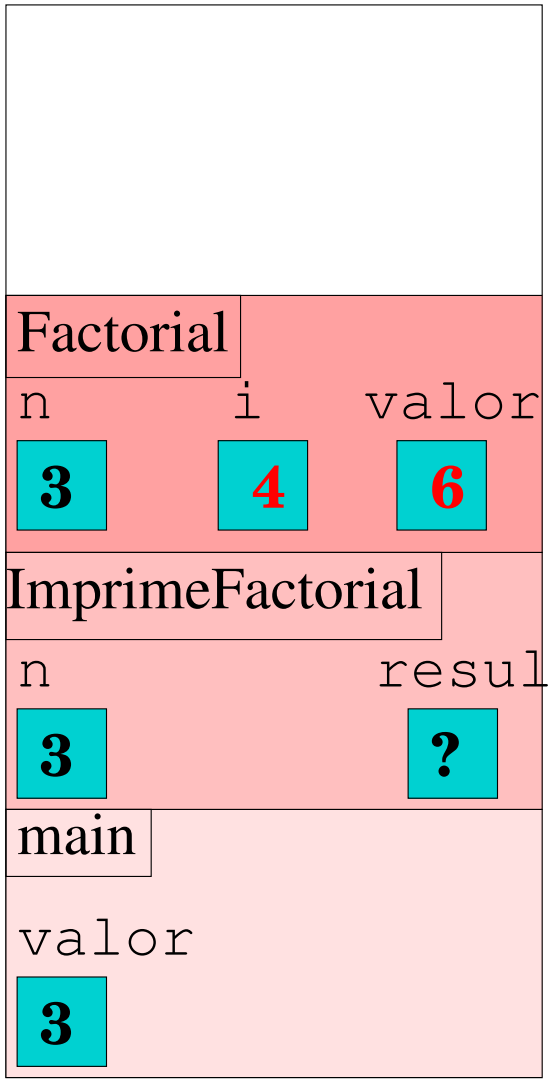
Ámbito de un dato

El ámbito de un dato

es el conjunto de todos aquellos módulos que lo pueden referenciar.

¿Cuál es el ámbito de cada uno de los datos que aparecen en la figura?

El ámbito de un dato está definido por el par de llaves que definen el bloque de código dónde se ha declarado el dato



Ámbito de un dato

El ámbito de un dato

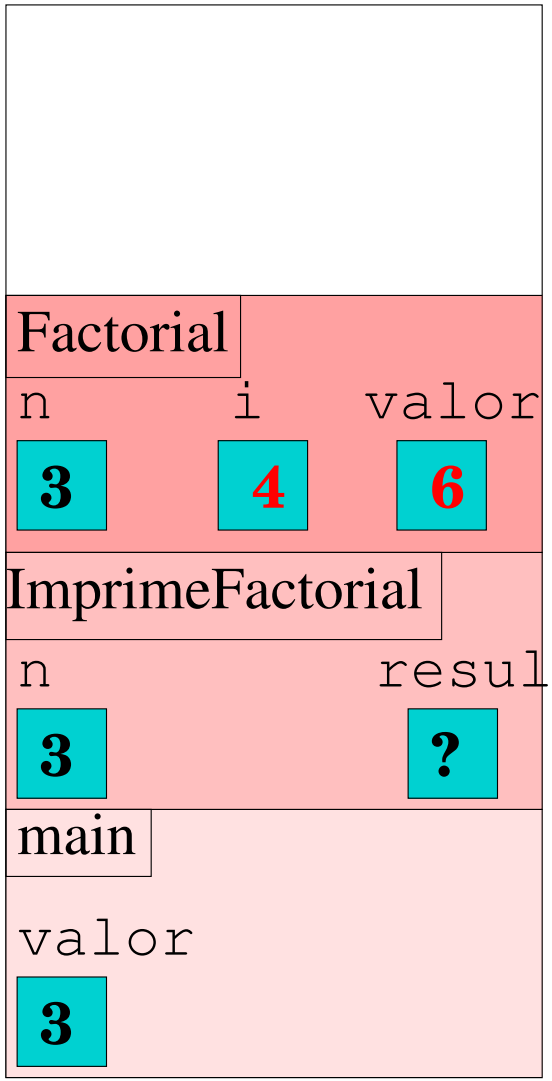
es el conjunto de todos aquellos módulos que lo pueden referenciar.

¿Cuál es el ámbito de cada uno de los datos que aparecen en la figura?

El ámbito de un dato está definido por el par de llaves que definen el bloque de código dónde se ha declarado el dato

Excepción

Datos globales



¿Cuál es el ámbito de los datos que aparecen en esta función?

```
1  double f1(double x, double y){
2     double x, j;
3
4     for (int i=x; i<y; i++){
5       double z;
6       z=(i-x);
7       j=z/(y-x);
8       cout << j <<endl;
9     }
10 }
```

¿Cuál es el ámbito de los datos que aparecen en esta función?

```
1  double f1(double x, double y){
2     double x, j;
3
4     for (int i=x; i<y; i++){
5       double z;
6       z=(i-x);
7       j=z/(y-x);
8       cout << j <<endl;
9     }
10 }
```

Solución

- x, y, j son globales a todo el módulo.
- i(línea 4), z son locales al cuerpo del **for**.
- qué x se está usando en el bucle ?

Contenido del tema

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float x){
...
}
int main(){
 float a;
 f1(a); ...
```

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float x){
...
}
int main(){
 float a;
 f1(a); ...
```

Por valor o copia

- Es el paso de argumentos por defecto.

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float x){
...
}
int main(){
 float a;
 f1(a); ...
```

Por valor o copia

- Es el paso de argumentos por defecto.
- Durante la llamada se realiza una copia del parámetro actual en el parámetro formal.

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float x){
...
}
int main(){
 float a;
 f1(a); ...
```

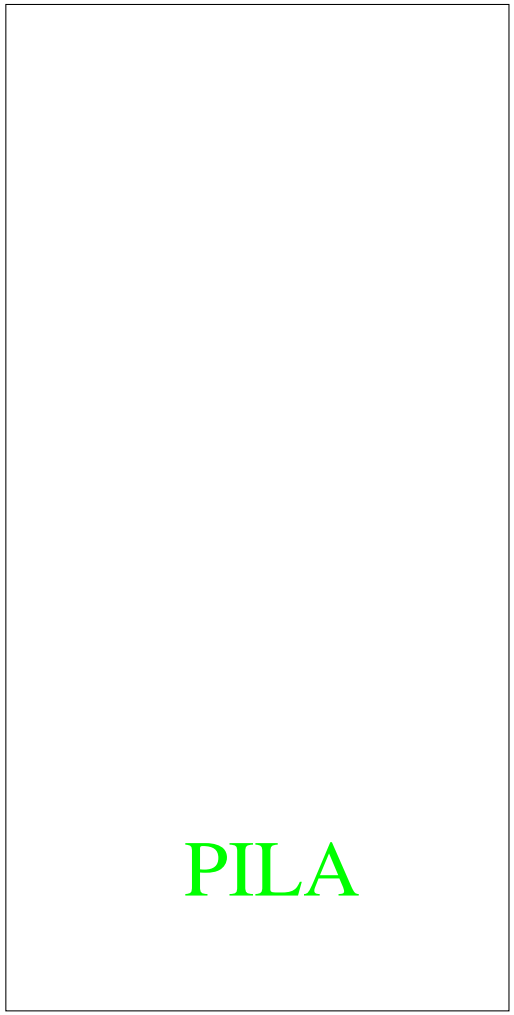
Por valor o copia

- Es el paso de argumentos por defecto.
- Durante la llamada se realiza una copia del parámetro actual en el parámetro formal.
- De esta forma, el módulo invocado trabaja con una copia y no con el valor original.

Problema:

Construir una función que intercambie el valor de dos variables

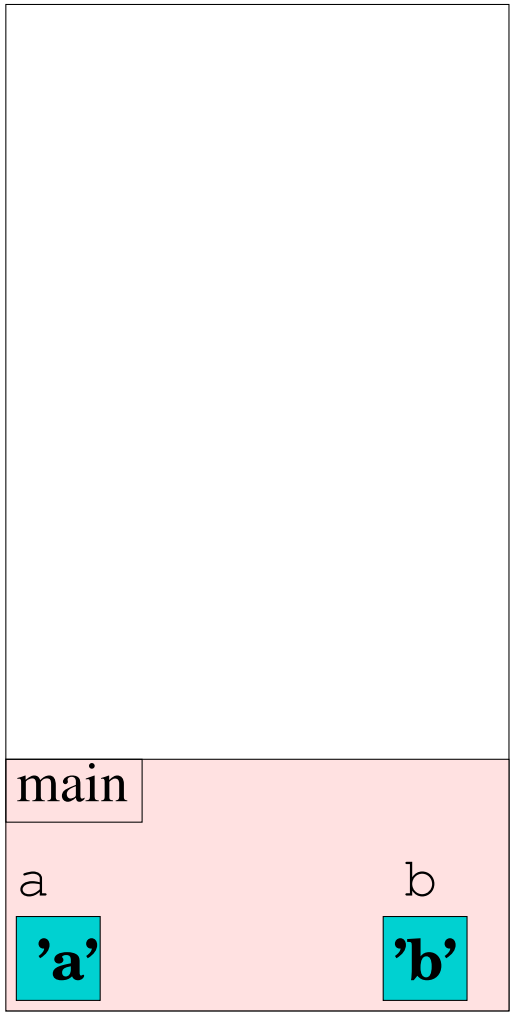
```
1  #include <iostream>
2  using namespace std;
3
4  void Swap(char c1, char c2){
5    char aux=c1;
6    c1=c2;
7    c2=aux;
8  }
9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



Problema:

Construir una función que intercambie el valor de dos variables

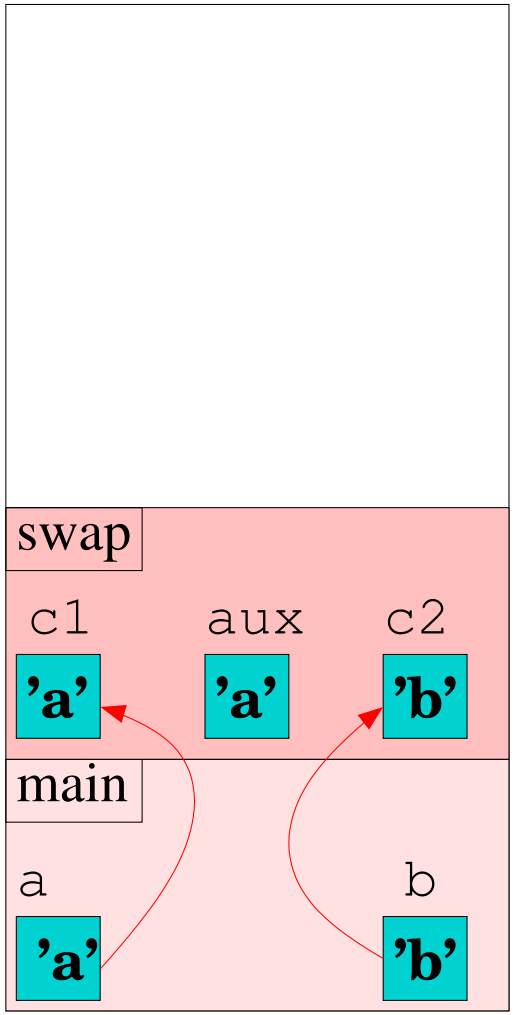
```
1  #include <iostream>
2  using namespace std;
3
4  void Swap(char c1, char c2){
5    char aux=c1;
6    c1=c2;
7    c2=aux;
8  }
9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



Problema:

Construir una función que intercambie el valor de dos variables

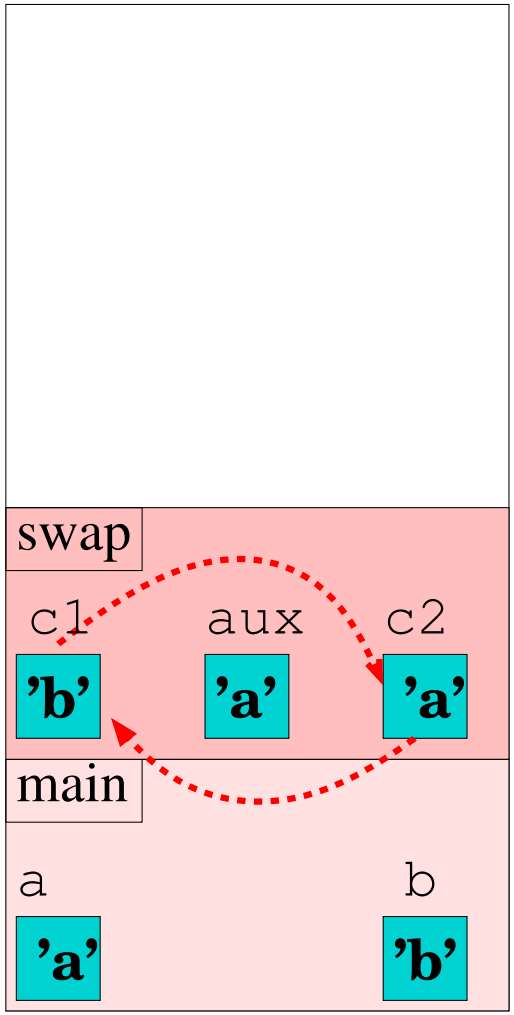
```
1  #include <iostream>
2  using namespace std;
3
4  void Swap(char c1, char c2){
5    char aux=c1;
6    c1=c2;
7    c2=aux;
8  }
9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



Problema:

Construir una función que intercambie el valor de dos variables

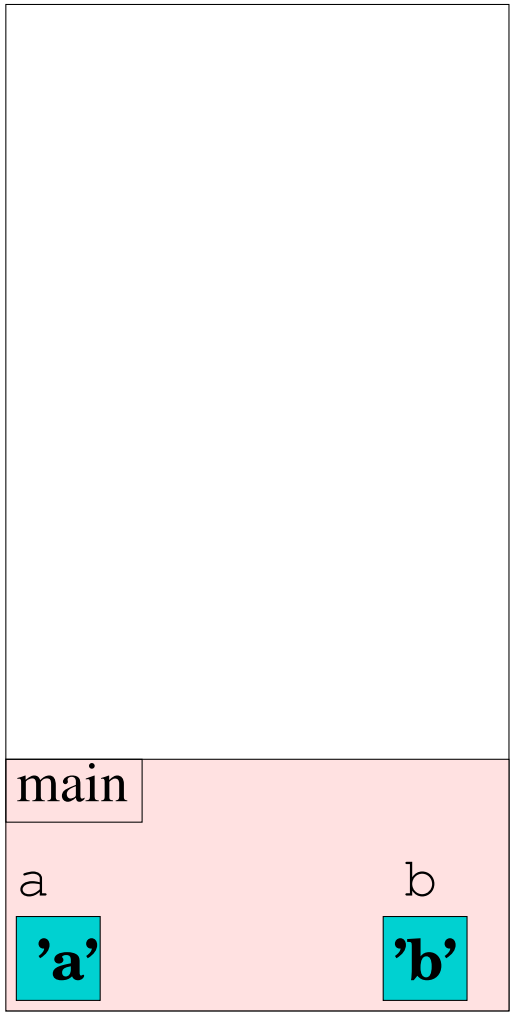
```
1  #include <iostream>
2  using namespace std;
3
4  void Swap(char c1, char c2){
5    char aux=c1;
6    c1=c2;
7    c2=aux;
8  }
9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



Problema:

Construir una función que intercambie el valor de dos variables

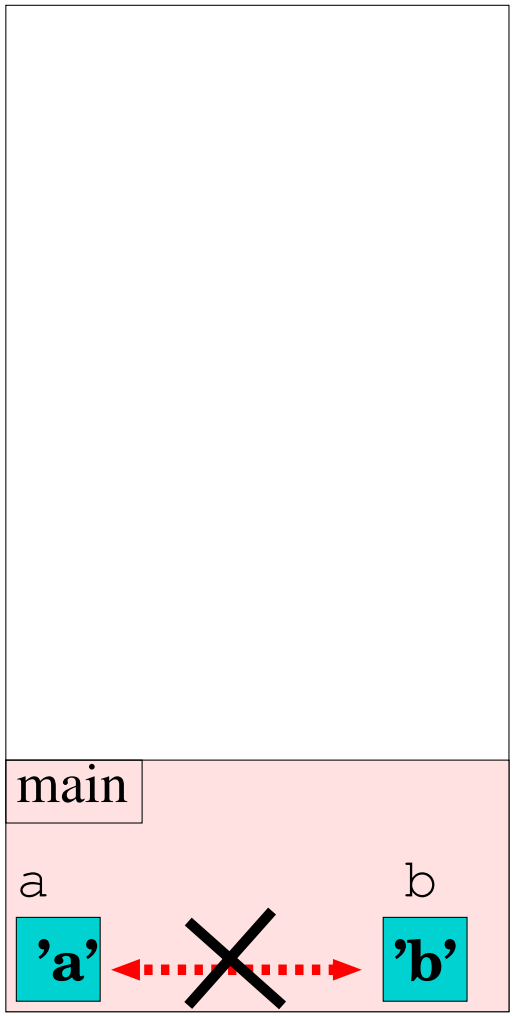
```
1  #include <iostream>
2  using namespace std;
3
4  void Swap(char c1, char c2){
5    char aux=c1;
6    c1=c2;
7    c2=aux;
8  }
9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



Problema:

Construir una función que intercambie el valor de dos variables

```
1  #include <iostream>
2  using namespace std;
3
4  void Swap(char c1, char c2){
5    char aux=c1;
6    c1=c2;
7    c2=aux;
8  }
9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



Análisis

- Los valores de las variables a y b no se han modificado.

Análisis

- Los valores de las variables a y b no se han modificado.
- Los que se intercambiaron fueron sus copias c1 y c2.

Análisis

- Los valores de las variables a y b no se han modificado.
- Los que se intercambiaron fueron sus copias c1 y c2.
- Para resolver el problema anterior tenemos que trabajar con los datos originales y no con las copias.

Análisis

- Los valores de las variables `a` y `b` no se han modificado.
- Los que se intercambiaron fueron sus copias `c1` y `c2`.
- Para resolver el problema anterior tenemos que trabajar con los datos originales y no con las copias.
- Esto es, extender el ámbito de `a` y `b` para que sean manipulables en el entorno de `Swap`.

Análisis

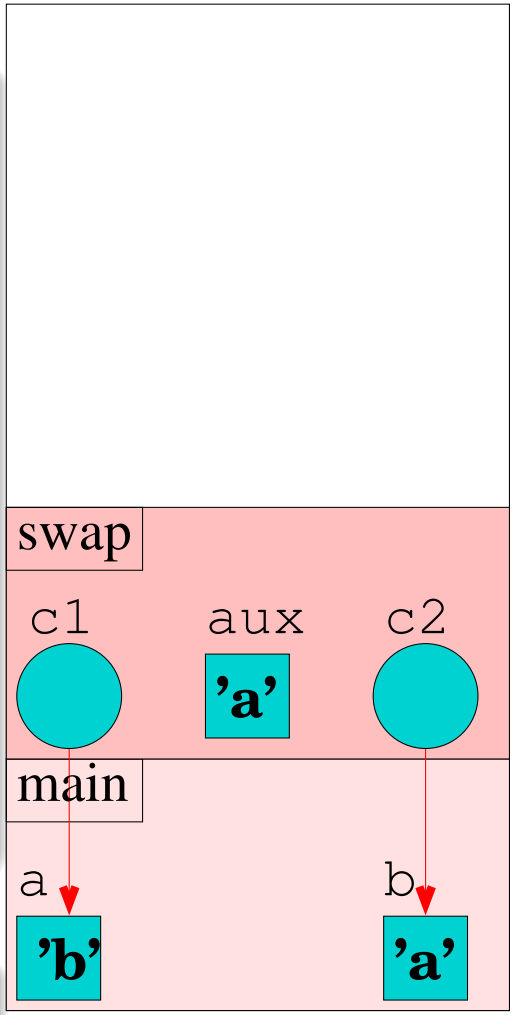
- Los valores de las variables `a` y `b` no se han modificado.
- Los que se intercambiaron fueron sus copias `c1` y `c2`.
- Para resolver el problema anterior tenemos que trabajar con los datos originales y no con las copias.
- Esto es, extender el ámbito de `a` y `b` para que sean manipulables en el entorno de `Swap`.

Solución

Paso de parámetros por referencia

Análisis

- Los valores de las variables a y b no se han modificado.
- Los que se intercambiaron fueron sus copias c1 y c2.
- Para resolver el problema anterior tenemos que trabajar con los datos originales y no con las copias.
- Esto es, extender el ámbito de a y b para que sean manipulables en el entorno de Swap.

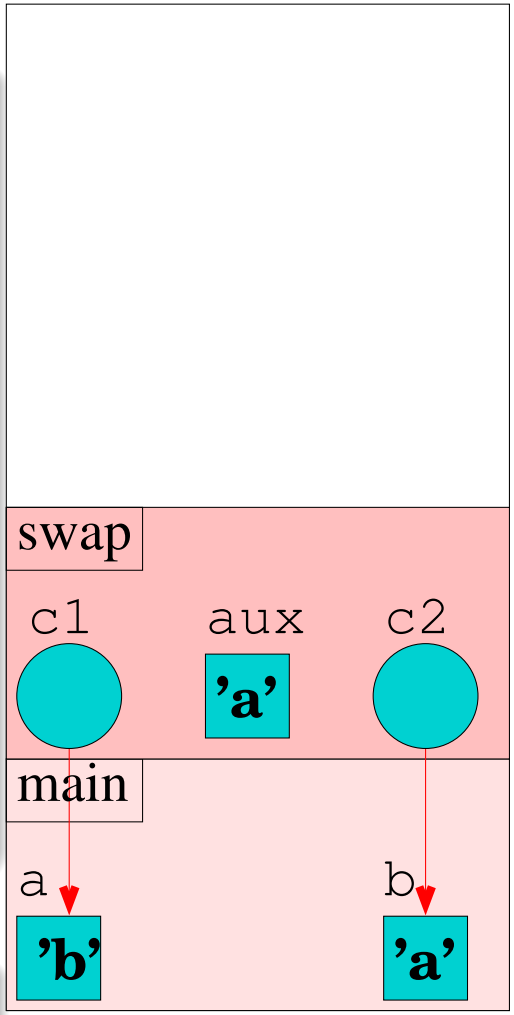


Solución

Paso de parámetros por referencia

Análisis

- Los valores de las variables a y b no se han modificado.
- Los que se intercambiaron fueron sus copias c1 y c2.
- Para resolver el problema anterior tenemos que trabajar con los datos originales y no con las copias.
- Esto es, extender el ámbito de a y b para que sean manipulables en el entorno de Swap.



Solución

Paso de parámetros por referencia

pero antes... Qué es una referencia?

Contenido del tema

Referencias

Referencia

Alias de un dato u objeto ya existente. Identificador al que **no** se le asocia nueva memoria. Se nota **&** entre el tipo y el alias-identificador.
Se usa en:

- Referencias como alias a otras variables
- Devolución por referencia desde una función
- Paso de parámetros por referencia en una función o método

Referencias como alias a otras variables I

Sintaxis

<tipo> & <identificador> = <iniciador> ;

Una referencia debe **inicializarse** en su declaración y **no puede reasignarse** como alias a otras variables.

- **Ejemplo 1**:

```
int lavariable_a = 0;
int &ref = lavariable_a; // alias
ref = 5;
cout << lavariable_a << endl;
```

- **Ejemplo 2**:

```
int v[5]={1,2,3,4,5};
int &pos3 = v[3];
pos3 = 0;
cout<< v[3] <<endl;    // muestra 0
```

Referencias como alias a otras variables II

- **Ejemplo 3. Con objetos básicos:**

```
Cuadrado origen(0,0,4), destino, &refCuadrado = origen;
destino = refCuadrado;  // en la Dcha de lasignacion
destino.setCuadrado(1,1,2);
refCuadrado = destino;  // en la Izda de lasignacion
// origen  ?
// destino ?
```

Devolución por referencia

Una función/método puede devolver una referencia a un dato o a un objeto

```
int& valor(int v[], int i){
    return v[i];  // devuelve el objeto no el valor del objeto
}
```

Devolución por referencia

Una función/método puede devolver una referencia a un dato o a un objeto

```
int& valor(int v[], int i){
    return v[i];  // devuelve el objeto no el valor del objeto
}
```

Se utilizará cuando no sea adecuado realizar una copia. `cin`, `cout`

Puede usarse tanto a la derecha como a la izquierda de una asignación.

```
int main(){
    int v[]={3,5,2,7,6};
    int a = valor(v,3);  // referencia a la dcha asignacion
}
```

Devolución por referencia

Una función/método puede devolver una referencia a un dato o a un objeto

```
int& valor(int v[], int i){
    return v[i];  // devuelve el objeto no el valor del objeto
}
```

Se utilizará cuando no sea adecuado realizar una copia. `cin`, `cout`

Puede usarse tanto a la derecha como a la izquierda de una asignación.

```
int main(){
    int v[]={3,5,2,7,6};
    int a = valor(v,3);  // referencia a la dcha asignacion
}
int main(){
    int v[]={3,5,2,7,6};
    valor(v,3) = 0; //referencia a izda asignacion
}
```

Devolución por referencia

Devolución de referencias a datos locales

La devolución de referencias a datos locales de una función es un error típico: Los datos locales se destruyen al terminar la función.

```
#include <iostream>
using namespace std;
int& funcion()
{
    int x=3;
    return x; // Error: devolucion referencia a variable local
}
int main()
{
    int y=funcion();
    cout << y << endl;
}
```

Contenido del tema

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float & x){
...
}
int main(){
 float a;
 f1(a); ...
```

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float & x){
...
}
int main(){
 float a;
 f1(a); ...
```

Por referencia o variable

- En la cabecera, se usa **&** entre el tipo y el identificador del argumento para indicar que el paso se realiza por referencia.
- No realiza una copia del parámetro actual en el formal, sino un **vínculo** entre ellos, de tal forma que una modificación en el parámetro formal, conlleva la misma modificación en el parámetro actual.

Paso de parámetros

¿Cuántas variables hay alojadas cuando se produce una llamada?

```
void f1(float & x){
...
}
int main(){
 float a;
 f1(a); ...
```

Por referencia o variable

- En la cabecera, se usa **&** entre el tipo y el identificador del argumento para indicar que el paso se realiza por referencia.
- No realiza una copia del parámetro actual en el formal, sino un **vínculo** entre ellos, de tal forma que una modificación en el parámetro formal, conlleva la misma modificación en el parámetro actual.

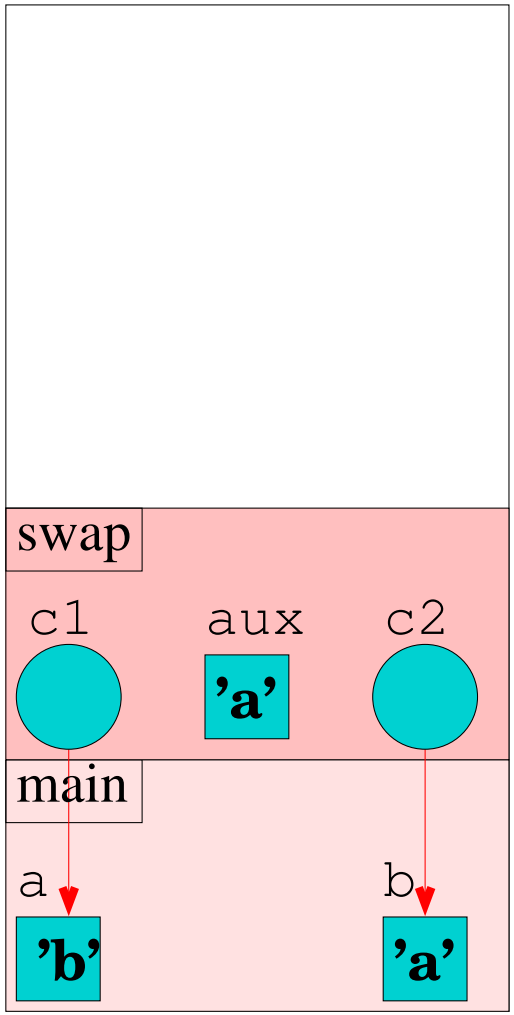
Ejemplos

```
1 void Swap (char &c1, char &c2);
2 void Division (int dividendo, int divisor,
3                int &coc, int &resto);
4 void ElegirOpcion (char &opcion);
```

Solución del problema

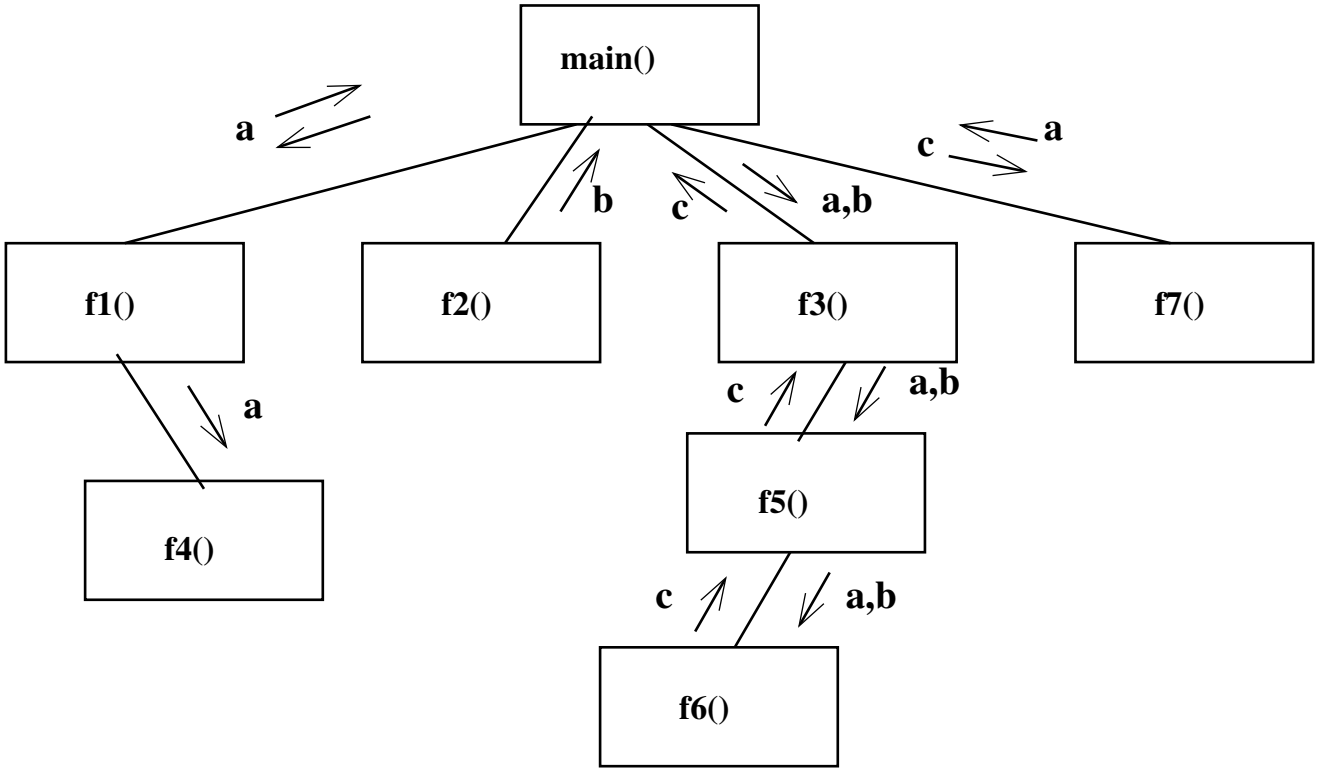
Construir una función que intercambie el valor de dos variables

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 void Swap(char &c1, char &c2){
 5   char aux=c1;
 6   c1=c2;
 7   c2=aux;
 8 }
 9
10 int main(){
11   char a='a', b='b';
12
13   Swap(a,b);
14   cout << "a=" << a
15        << " y b=" << b << endl;
16 }
```



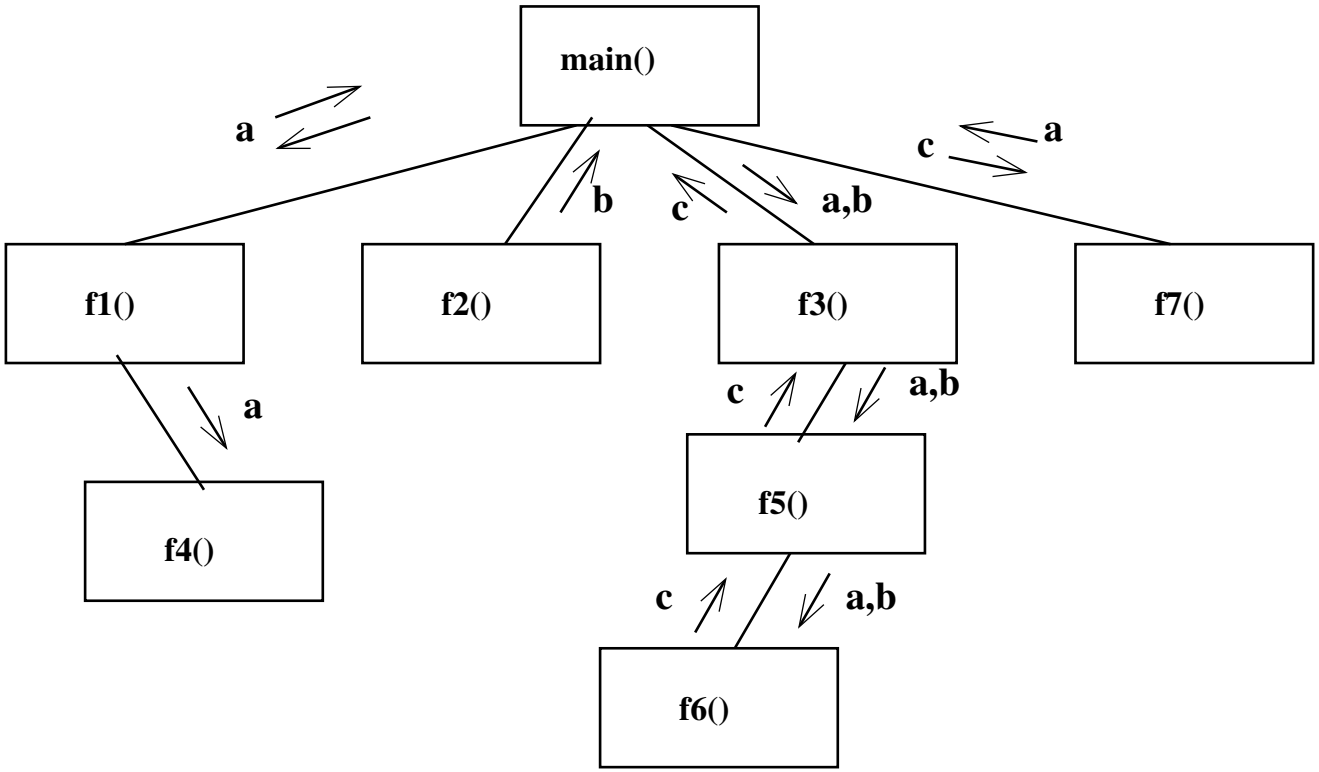
Contenido del tema

La comunicación se lleva a cabo mediante las cabeceras de los módulos, a través de los parámetros y en el tipo.



A cada módulo, los datos que necesita (ni menos, ni más).

El ocultamiento de información: una estrategia para la depuración de programas.



si error en **a**: se revisa **main(), f1(), f7()**
si error en **b**: se revisa **main(), f2()**

Contenido del tema

Valor/Referencia versus Entrada/Salida

Valor/Referencia versus Entrada/Salida

La identificación de la información que aporta un argumento en una función nos indica la forma en la que debe ser pasado dicho argumento.

Valor/Referencia versus Entrada/Salida

La identificación de la información que aporta un argumento en una función nos indica la forma en la que debe ser pasado dicho argumento.

- Si el argumento es usado como vehículo para obtener la solución, entonces nos encontramos ante un **parámetro de entrada**.

Valor/Referencia versus Entrada/Salida

La identificación de la información que aporta un argumento en una función nos indica la forma en la que debe ser pasado dicho argumento.

- Si el argumento es usado como vehículo para obtener la solución, entonces nos encontramos ante un **parámetro de entrada**.

PASO POR VALOR

Valor/Referencia versus Entrada/Salida

La identificación de la información que aporta un argumento en una función nos indica la forma en la que debe ser pasado dicho argumento.

- Si el argumento es usado como vehículo para obtener la solución, entonces nos encontramos ante un **parámetro de entrada**.

PASO POR VALOR

- Si el argumento es usado para almacenar la solución o parte de ella, entonces nos encontramos ante un **parámetro de salida**.

Valor/Referencia versus Entrada/Salida

La identificación de la información que aporta un argumento en una función nos indica la forma en la que debe ser pasado dicho argumento.

- Si el argumento es usado como vehículo para obtener la solución, entonces nos encontramos ante un **parámetro de entrada**.

PASO POR VALOR

- Si el argumento es usado para almacenar la solución o parte de ella, entonces nos encontramos ante un **parámetro de salida**.
- Si el argumento es tanto vehículo para obtener la solución como parte de la misma, entonces nos encontramos con un **parámetro de entrada/salida**.

Valor/Referencia versus Entrada/Salida

La identificación de la información que aporta un argumento en una función nos indica la forma en la que debe ser pasado dicho argumento.

- Si el argumento es usado como vehículo para obtener la solución, entonces nos encontramos ante un **parámetro de entrada**.

PASO POR VALOR

- Si el argumento es usado para almacenar la solución o parte de ella, entonces nos encontramos ante un **parámetro de salida**.
- Si el argumento es tanto vehículo para obtener la solución como parte de la misma, entonces nos encontramos con un **parámetro de entrada/salida**.

PASO POR REFERENCIA

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

```
bool Es_Primo (int);
```

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

```
int Num_Primos (int, int);
```

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

```
void MaxAndMix (double &, double &);
```

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

```
void presentaMenu ();
```

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

e) Calcular la suma de dos números complejos. Supóngase que representamos un número complejo usando dos números reales.

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

e) Calcular la suma de dos números complejos. Supóngase que representamos un número complejo usando dos números reales.

```
void SumaComplejos (double c1r, double c1i,
double c2r, double c2i, double &c3r, double &c3i);
```

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

e) Calcular la suma de dos números complejos. Supóngase que representamos un número complejo usando dos números reales.

f) Calcular la derivada de un polinomio de grado 3.

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

e) Calcular la suma de dos números complejos. Supóngase que representamos un número complejo usando dos números reales.

f) Calcular la derivada de un polinomio de grado 3.

```
void DerivadaPol (double a, double b, double c,
double d, double &ad, double &bd, double &cd);
```

Ejercicio: Escribir el prototipo de:

a) Determinar si un número es primo.

b) Calcular el número de primos existentes en un intervalo de valores.

c) Calcular el máximo y el mínimo de una secuencia de valores reales introducidos por la entrada estándar (los valores se leen dentro de la función).

d) Escribir por la salida estándar un menú.

e) Calcular la suma de dos números complejos. Supóngase que representamos un número complejo usando dos números reales.

f) Calcular la derivada de un polinomio de grado 3.

Valor/Referencia versus Entrada/Salida

En la llamada

cuando el paso de parámetros es por valor, el argumento actual puede ser **una expresión, una constante o una variable**.

```
void f (double a); // prototipo
f(1); f(v+10); // llamadas validas
f(v);
```

Valor/Referencia versus Entrada/Salida

En la llamada

cuando el paso de parámetros es por valor, el argumento actual puede ser **una expresión, una constante o una variable**.

```
void f (double a); // prototipo
f(1); f(v+10); // llamadas validas
f(v);
```

En la llamada

cuando el paso de parámetros es por referencia, el argumento actual debe ser obligatoriamente **una variable**.

```
void f (double & a); // prototipo
f(v); // llamada válida
```

Valor/Referencia según Criterio de espacio

Por Valor

Cuando el paso de parámetros es por valor, el parámetro es copiado enteramente, **gastando en la pila todo el espacio que ocupa el parámetro**.

Valor/Referencia según Criterio de espacio

Por Valor

Cuando el paso de parámetros es por valor, el parámetro es copiado enteramente, **gastando en la pila todo el espacio que ocupa el parámetro**.

Si el parámetro es un objeto, se reserva el espacio correspondiente, llevándose a cabo también la operación de copia del objeto.

```
Polinomio Suma(Polinomio sumando); // prototipo
poli = p1.Suma(p2); // llamada

void Concatenar(vectorCaracteres cadena2); // prototipo
cad.Concatenar(cad2); // llamada
```

Valor/Referencia según Criterio de espacio

Por Valor

Cuando el paso de parámetros es por valor, el parámetro es copiado enteramente, **gastando en la pila todo el espacio que ocupa el parámetro**.

Por referencia

Cuando el paso de parámetros es por referencia, se dispone de un sinónimo al objeto referenciado. **En la pila se ocupa siempre lo mismo** (el tamaño de una dirección).

Para ahorrar espacio (y esfuerzo de copia), **cuando se trate de objetos** el paso de parámetros se hace **por referencia**...

```
Polinomio Suma (Polinomio sumando); // prototipo vers1
poli = p1.Suma(p2); // llamada
```

```
Polinomio Suma (Polinomio & sumando); // prototipo vers2
poli = p1.Suma(p2); // llamada
```

Para ahorrar espacio (y esfuerzo de copia), **cuando se trate de objetos** el paso de parámetros se hace **por referencia**...

```
Polinomio Suma (Polinomio sumando); // prototipo vers1
poli = p1.Suma(p2); // llamada

Polinomio Suma (Polinomio & sumando); // prototipo vers2
poli = p1.Suma(p2); // llamada

void Concatenar (vectorCaracteres cadena2); // proto vers1
void Concatenar (vectorCaracteres &cadena2); // proto vers2

cad.Concatenar(cad2); // llamada
```

Para ahorrar espacio (y esfuerzo de copia), **cuando se trate de objetos** el paso de parámetros se hace **por referencia**...

Polinomio Suma (Polinomio sumando); `// prototipo vers1`

```
poli = p1.Suma(p2); // llamada
```

Polinomio Suma (Polinomio & sumando); `// prototipo vers2`

```
poli = p1.Suma(p2); // llamada
```

```
void Concatenar (vectorCaracteres cadena2); // proto vers1
void Concatenar (vectorCaracteres &cadena2); // proto vers2
```

```
cad.Concatenar(cad2); // llamada
```

¿Algún problema?

Para ahorrar espacio (y esfuerzo de copia), **cuando se trate de objetos** el paso de parámetros se hace **por referencia**...

Polinomio Suma (Polinomio sumando); `// prototipo vers1`

```
poli = p1.Suma(p2); // llamada
```

Polinomio Suma (Polinomio & sumando); `// prototipo vers2`

```
poli = p1.Suma(p2); // llamada
```

```
void Concatenar (vectorCaracteres cadena2); // proto vers1
void Concatenar (vectorCaracteres &cadena2); // proto vers2
```

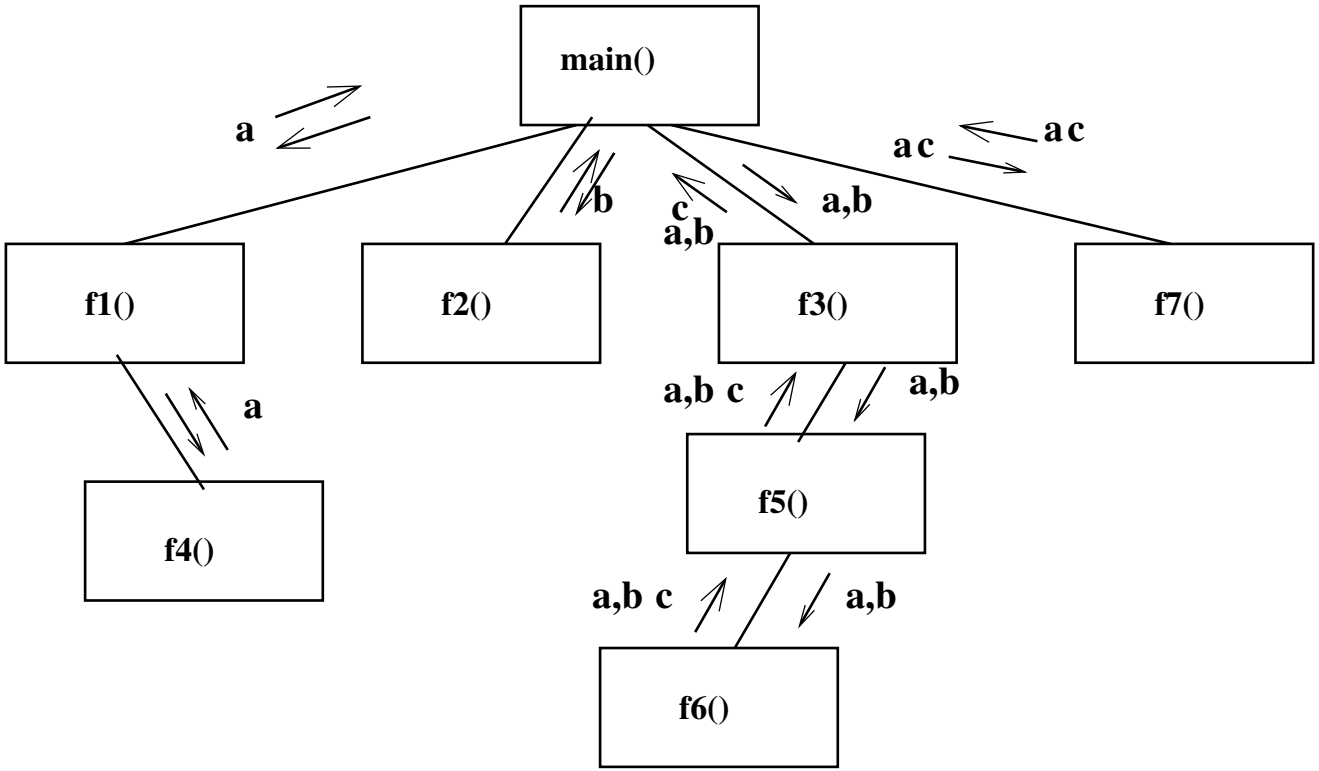
```
cad.Concatenar(cad2); // llamada
```

¿Algún problema?

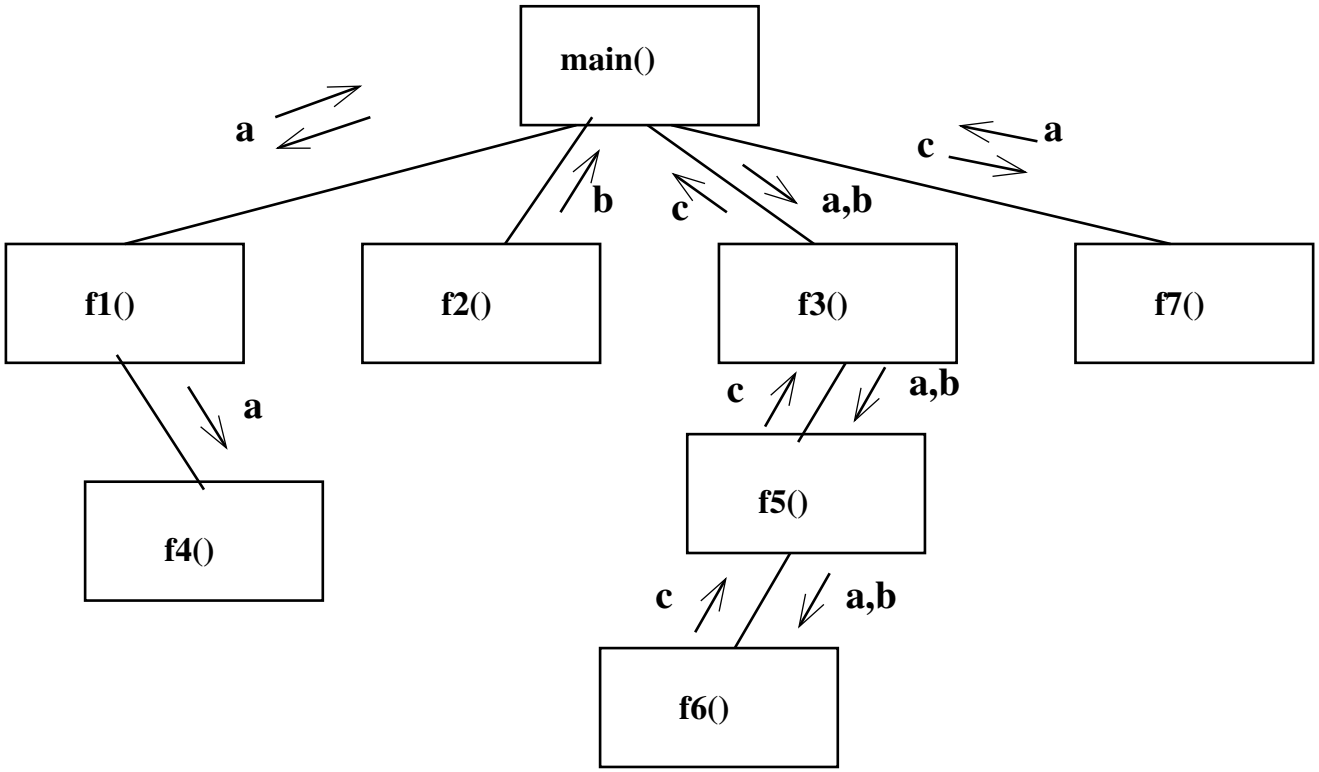
p2 (sumando) **en peligro**!!!

Con el paso de objetos por referencia hay un problema de comunicación entre módulos.

Cualquier módulo puede tener acceso a datos de entrada y los puede modificar **por error**.



¿Cómo restaurar la comunicación selectiva de entrada/salida de parámetros con objetos?



const

es un atributo preservativo que se utiliza asociado a variables, objetos y métodos. Declara el dato como **no modificable** (prohibido la escritura en memoria del valor).

Objetos protegidos. ¿Cómo?

```
 Polinomio Suma (const Polinomio &sumando){
 sumando = sumando + 1;  // viola la restriccion
 sumando.clear();          // viola la restriccion
}
```

Objetos protegidos. ¿Cómo?

```
 Polinomio Suma (const Polinomio &sumando){
 sumando = sumando + 1;  // viola la restriccion
 sumando.clear();          // viola la restriccion
}
```

```
Polinomio.cpp:  En la función 'Polinomio Suma (const
Polinomio)':
Polinomio.cpp:7:311:  error:  asignación de la ubicación de
sólo lectura 'sumando'
sumando = sumando + 1;
```

Objetos como parámetros

Objetos como entrada

Al ser solo de entrada, el objeto no se modifica. Luego, se pasa por referencia (por criterio de espacio) con protección **const**.

```
Polinomio Suma (const Polinomio &sumando);
```

Objetos como entrada/salida

El objeto se modifica luego, se pasa por referencia.

```
void funcionAlterar ( Polinomio & polinomio);
```

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

```
Circulo (int coorx, int coory, double r);
```

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).

```
bool mayorQue(const Circulo &);
```

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).

c) Calcular la distancia entre los centros de dos Círculos.

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).

c) Calcular la distancia entre los centros de dos Círculos.

```
double distanciaCirculo(const Circulo &);
```

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros
b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).
c) Calcular la distancia entre los centros de dos Círculos.
d) Comprobar si el Circulo de referencia se interseca con otro.

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros

b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).

c) Calcular la distancia entre los centros de dos Círculos.

d) Comprobar si el Circulo de referencia se interseca con otro.

```
bool intersecaSi(const Circulo &);
```

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros
b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).
c) Calcular la distancia entre los centros de dos Círculos.
d) Comprobar si el Circulo de referencia se interseca con otro.
e) Definir el constructor de copia

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros
b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).
c) Calcular la distancia entre los centros de dos Círculos.
d) Comprobar si el Circulo de referencia se interseca con otro.
e) Definir el constructor de copia

Supongamos definida la clase Complejos

f) calcular la suma de dos números complejos, miembro de la clase.

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros
b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).
c) Calcular la distancia entre los centros de dos Círculos.
d) Comprobar si el Circulo de referencia se interseca con otro.
e) Definir el constructor de copia

Supongamos definida la clase Complejos

f) calcular la suma de dos números complejos, miembro de la clase.

```
Complejo Suma (const Complejo &);
```

Ejercicio: Escribir el prototipo de:

Se supone definida la clase Circulo y se han de añadir nuevos métodos.

a) Definir el constructor con parámetros
b) Comprobar si el circulo en cuestión es mayor que otro (Circulo).
c) Calcular la distancia entre los centros de dos Círculos.
d) Comprobar si el Circulo de referencia se interseca con otro.
e) Definir el constructor de copia

Supongamos definida la clase Complejos

f) calcular la suma de dos números complejos, miembro de la clase.
g) calcular la suma de dos números complejos, función externa a la clase.

Revisando Cuadrado I

```
1  class Cuadrado {
2  private:
3    int x;
4    int y;
5    float l;
6  bool correcto( float _l){
7     return _l>0;
8   }
9  public:
10   Cuadrado()
11     :x(0), y(0), l(1){}
12   Cuadrado(int _x, int _y, float _l)
13     :x(_x), y(_y), l(1){
14     if (correcto(_l))
15         l = _l;
16   }
```

Revisando Cuadrado II

```
17   float area() const {
18    return l*l;
19   }
20   float perimetro() const {
21    return 4*l;
22   }
23   int getX() const {
24    return x;
25   }
26   int getY() const {
27    return y;
28   }
29   float getL() const {
30    return l;
31   }
```

Revisando Cuadrado III

```
32   bool mayorQ(const Cuadrado &c) const {
33   bool mas = false;
34   if (area() > c.area())
35       mas=true;
36   return mas;
37   }
38   bool contenidoEn(const Cuadrado &c) const {
39   bool cabe = false;
40        if ((x >= c.getX()) && ((x+l) <= (c.getX()+ c.getL())) &&
41            (y >= c.getY()) && ((y+l) <= (c.getY()+ c.getL())))
42            cabe = true;
43    return cabe;
44   }
45 };
```

Contenido del tema

Parámetros con valor por defecto

Una función o método puede tener parámetros con un valor por defecto

- Deben ser los últimos de la función.
- En la llamada a la función, si sólo se especifican un subconjunto de ellos, deben ser los primeros.

```
void funcion(char c, int i=7); // SOLO EN PROTOTIPO
void funcion(char c, int i) {
    ...
}
int main(){
    funcion('a',8);
    funcion('z');
}
```

Parámetros con valor por defecto: Ejemplo

```
#include <iostream>
using namespace std;
int volumenCaja(int largo=1, int ancho=1, int alto=1);
int main()
{
    cout << "Volumen por defecto: " << volumenCaja() << endl;
    cout << "El volumen de una caja (10,1,1) es: " <<
            volumenCaja(10) << endl;
    cout << "El volumen de una caja (10,5,1) es: " <<
            volumenCaja(10,5) << endl;
    cout << "El volumen de una caja (10,5,2) es: " <<
            volumenCaja(10,5,2) << endl;
    return 0;
}
int volumenCaja( int largo, int ancho, int alto )
{
    return largo * ancho * alto;
}
```

Contenido del tema

Variables locales static

Variable local static

Es una variable local de una función o método que no se destruye al acabar la función, y que mantiene su valor entre llamadas.

- Se inicializa la primera vez que se llama a la función.
- Conserva el valor anterior en sucesivas llamadas a la función.
- Es obligatorio asignarles un valor en su declaración.

```
#include <iostream>
double cuadrado(double numero){
    static int contadorLlamadas=1;
    std::cout<<"Llamadas a cuadrado: "
             <<contadorLlamadas<<std::endl;
    contadorLlamadas++;
    return numero*numero;
}
int main(){
    for(int i=0; i<10; ++i)
        std::cout<<i<<"^2 = "<<cuadrado(i)<<std::endl;
}
```

Contenido del tema

Sobrecarga de funciones

Sobrecarga de funciones

C++ permite definir varias funciones en el mismo ámbito con el mismo nombre. C++ selecciona la función adecuada en base al número, tipo y orden de los argumentos.

```
void funcion(int x){
   ...
}
void funcion(double x){
   ...
}
void funcion(char *c){
   ...
}
void funcion(int x, double y){
   ...
}
```

```
int main(){

    char *c;
    funcion(3);
    funcion(4.5);
    funcion(4,9.3);
    funcion(c);
}
```

Sobrecarga de funciones

Conversión implícita de tipos

C++ puede aplicar conversión implícita de tipos para buscar la función adecuada.

```
void funcion(double x){
  cout << "double" << x << endl;
}
void funcion(char *p){
  cout << "char *" << *p <<endl;
}
int main(){
    funcion(4.5);
    funcion(3); // conversion implicita
}
```

Sobrecarga de funciones

NO Distinción por tipo devuelto

C++ NO puede distinguir entre dos versiones de una función que se diferencian sólo en el tipo devuelto.

```
int funcion(int x){
  return x*2;
}
double funcion(int x){
  return x/3.0;
}
int main(){
    int x=funcion(3);
    double f=funcion(5);
}
```

Sobrecarga de funciones

Uso de const en referencias

C++ puede distinguir entre versiones en que un parámetro referencia es const en una versión y en la otra no.

```
#include <iostream>
using namespace std;
void funcion(double &x){
  cout << "funcion(double &x): " << x <<endl;
}
void funcion(const double &x){
  cout << "funcion(const double &x): " << x <<endl;
}
int main(){
    double x=2;
    const double A=4.5;
    funcion(A);
    funcion(x);
}
```

Sobrecarga de funciones

Uso de const en parámetros por valor

Sin embargo, C++ no puede distinguir entre versiones en que un parámetro por valor es const en una versión y en la otra no.

```
#include <iostream>
using namespace std;
void funcion(double x){
  cout << "funcion(double x): " << x <<endl;
}
void funcion(const double x){
  cout << "funcion(const double x): " << x <<endl;
}
int main(){
    double x=2;
    const double A=4.5;
    funcion(A);
    funcion(x);
}
```

Sobrecarga de funciones

Ambigüedad

A veces pueden darse errores de ambigüedad

```
void funcion(int a, int b){
   ...
}
void funcion(double a, double b){
   ...
}
int main(){
    funcion(2,4);
    funcion(3.5,4.2);
    funcion(2,4.2); //Ambiguo
    funcion(3.5,4); //Ambiguo
    funcion(3.5,static_cast<double>(4));
}
```

Sobrecarga de funciones

Otro ejemplo de ambigüedad

En este caso al usar funciones con parámetros por defecto

```
void funcion(char c, int i=7){
    ...
}
void funcion(char c){
    ...
}
int main(){
    funcion('a',8);
    funcion('z');
}
```

Cuestiones abiertas I

- La clase Cuadrado
  - Revisad el constructor Cuadrado (int, int, float)
  - sea, `c1 = Cuadrado(1,1,-1)` que contiene los miembros de c1?
  - Responded ¿es adecuado?
  - Si no fuera adecuado, como lo arreglaríais?
  - Mirad la función assert()
  - Definid el método `setCuadrado()`
  - Declarad el método `maximoCuadrado()`, tipo, argumentos, forma de pasar argumentos
  - Definid el método `maximoCuadrado()` (según el prototipo anterior) aquél cuya área sea mayor.
- En el ejemplo 3. ¿Cuántas variables hay declaradas?
- Existe diferencia entre devolver un valor y una referencia en el return?
- Existe problema en `double & Circulo::funcion(Circulo c1) .. .. return c1.radio;`

Cuestiones abiertas II

- Cuál es el atributo profiláctico que evita que un objeto pueda estar en la parte izquierda de una asignación. Escribe algunos ejemplos de uso. ¿cómo hace que se evite esta asignación?
- Cuáles son los puntos claves para la comunicación entre módulos ?
- Qué criterios has de tener en cuenta para establecer el prototipo de una función?
- Qué criterios has de tener en cuenta para establecer el prototipo de un método?

Cuestiones abiertas III

- Supongamos que tenemos las siguientes función sobrecargada:

```
#include <iostream>
using namespace std;

void funcionA(double x){
  cout << "double" << x << endl;
}

void funcionA(double x, int a=0){
  cout << "double" << x << "int" << a << endl;
}
int main(){
    funcionA(4.5);
    funcionA(1.2,3);
}
```

  Qué función es invocada con cada una de las llamadas del main?


creative
commons


BY
NC
SA