Programación modular – Funciones en C++

Mag. Ing. Nancy López

Introducción

En la mayoría de los casos, un determinado problema complejo lo podemos (y debemos) dividir en problemas más sencillos. Estos subproblemas se conocen e el contexto de la programación como “Módulos” o **subprogramas**.

Método TOP DOWN

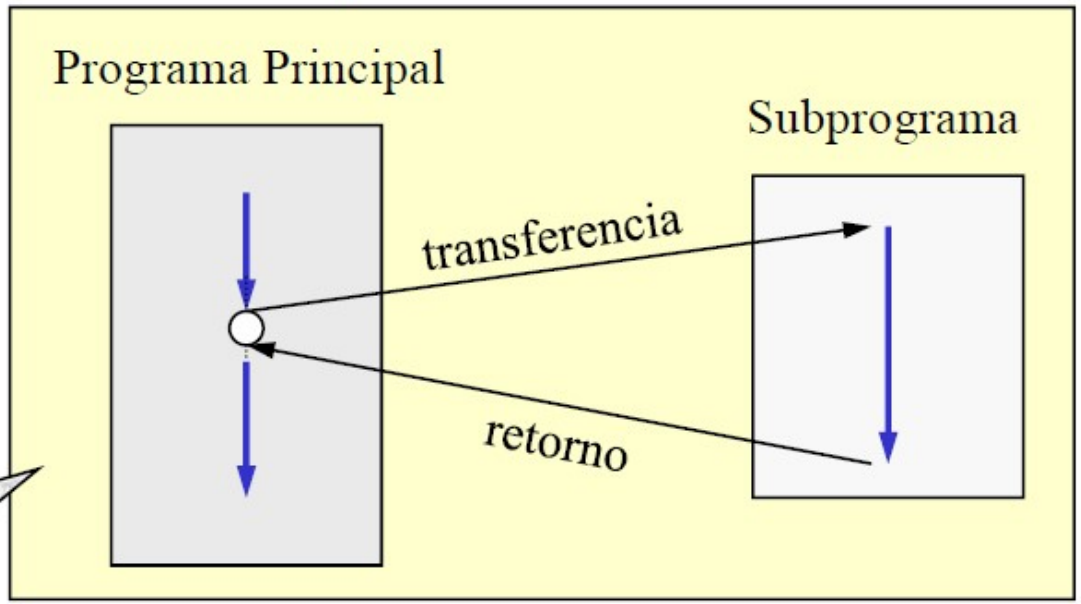
Desde el punto de vista del diseño:

- Se tratará de descomponer el problema original en partes.
- Se puede codificar de forma independiente e incluso por diferentes personas.
- El problema final queda resuelto y estructurado en forma de módulos, lo que hace más sencilla su lectura y mantenimiento.

Subprogramas

Un **subprograma** es una serie de instrucciones escritas independientemente del programa principal. Este subprograma está ligado al programa principal mediante un proceso de *transferencia/retorno*.

Transferencia
El control de ejecución se pasa al subprograma en el momento en que se requieren sus servicios.



Transferencia/retorno de **control** y **datos**

Retorno — El control de ejecución se devuelve al programa principal cuando el subprograma termina

Definición de FUNCIÓN

- C++ es un lenguaje modular, y por esta razón, se puede dividir en varios módulos, cada uno de los cuales realiza una tarea determinada. Cada módulo es un subprograma llamado función.
- Una función es un miniprograma dentro de un programa. Es un conjunto de sentencias que se pueden llamar desde cualquier parte del programa.
- Las funciones sirven para:
  - ⇨ realizar tareas concretas y simplificar el programa
  - ⇨ sirven para evitar escribir el mismo código varias veces.

Ventajas de utilizar funciones:

1.- Aislar mejor los problemas
2.- Escribir programas más rápido
3.- Programas más fáciles de mantener (más legibles y más cortos)

Diferencia entre procedimiento y función

- Un procedimiento NO devuelve datos.
- Una función SÍ devuelve datos.
- En un procedimiento podemos pedir y leer datos (cin y cout).
- En una función NO pedimos ni leemos datos, los recibimos como parámetros.

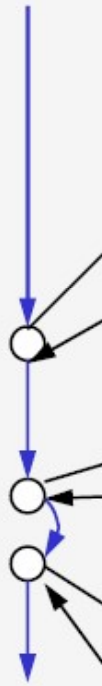
Transferencia / Retorno:

```
int main()
{
   int numero, absoluto;
   float  resultado;
   cin >> numero;
   if (numero > 0 )
        resultado = sqrt ( numero );
   else
      {
         absoluto = abs (numero);
         resultado = cubo( absoluto );
      }
    cout << resultado;
}
```

```
float  sqrt (int a)
   {
      float m;
      ....
      return m;
   }
```

```
int  abs (int a)
   {
      ....
      return ...;
   }
```

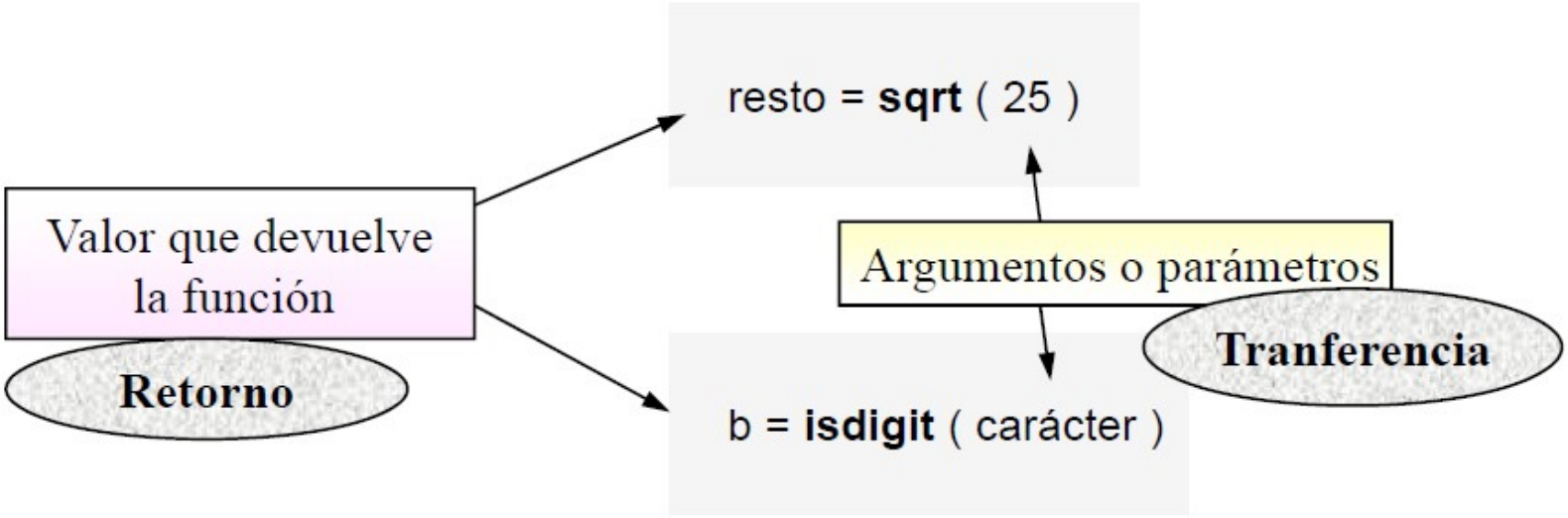
```
float  cubo (int a)
   {
      ....
      return ...;
   }
```



Transferencia/retorno de **control** y **datos**

Estructura de una función

Hasta ahora, hemos visto y utilizado funciones estándar, es decir definidas en una biblioteca.



C++ nos permite definir nuestras propias funciones. Pocas veces veremos un programa que no use funciones. Una de ellas, que usamos siempre, es la función **main**.

Estructura de una función

La estructura del prototipo de una función es la siguiente:

Prototipo

<tipoQueDevuelve> <nombreFunción>(tiposParámetros);

tipoQueDevuelve: Es el tipo de dato que devuelve la función.

tiposParámetros: Tipo/s de/los parámetro/s que recibe, separados por coma.

Ej. int suma(int, int);

Los prototipos se escriben antes de la función main().

Estructura de una función

La estructura de una función es la siguiente:

```
<tipoQueDevuelve> <nombreFunción>(tipos y nombre parámetros)
 {
   cuerpo de la función;
   return <expresión>;
 }
```

tipoQueDevuelve: Es el tipo de dato que devuelve la función.
tiposParámetros: Tipo/s y nombre de/los parámetro/s que recibe, separados por coma.

```
int suma (int a, int b)
{
   return a+b;
}
```

Estructura de una función

Una vez que se ha codificado la función, se puede usar, para ello se debe **invocar** la función.

Invocación

<nombre de la función> (parámetros)

Estructura de una función: Invocación

La invocación se puede hacer “on the fly”:

```
cout<<“La suma es: “<<suma(a,b);
```

O si se necesita operar con el resultado, asignamos el resultado a una variable:

```
res=suma(a,b);
```

También se la puede evaluar en un if:

```
if(suma(a,b)<0)
```

```
#include<iostream.h>
#include<conio.h>

int suma(int, int);

int main()
{
  int a, b;
  cout<<"Ingrese primer valor";
  cin>>a;
  cout<<"Ingrese segundo valor";
  cin>>b;
  cout<<"La suma es: "<<suma(a,b);
  return 0;
}

int suma(int a, int b)
{
  return a+b;
}
```

```
#include<iostream.h>
#include<conio.h>

int suma(int, int);

int main()
{
  int a, b;
  cout<<"Ingrese primer valor";
  cin>>a;
  cout<<"Ingrese segundo valor";
  cin>>b;
  cout<<"La suma es: "<<suma(a,b);
  return 0;
}

int suma(int x, int z)
{
  return x+z;
}
```

Estructura de una función: Valor de retorno

Una función solo puede devolver un valor. El valor se devuelve mediante la sentencia **return**

```
return <expresión> ;
```

1. C++ comprueba la compatibilidad de tipos, (no se puede devolver un valor de tipo int, si el tipo de retorno es por ejemplo de tipo char).

2. Una vez que se ejecuta esta sentencia, termina la ejecución de la función.

3. Una función puede tener cualquier número de sentencias return, pero **al menos debe haber una**.

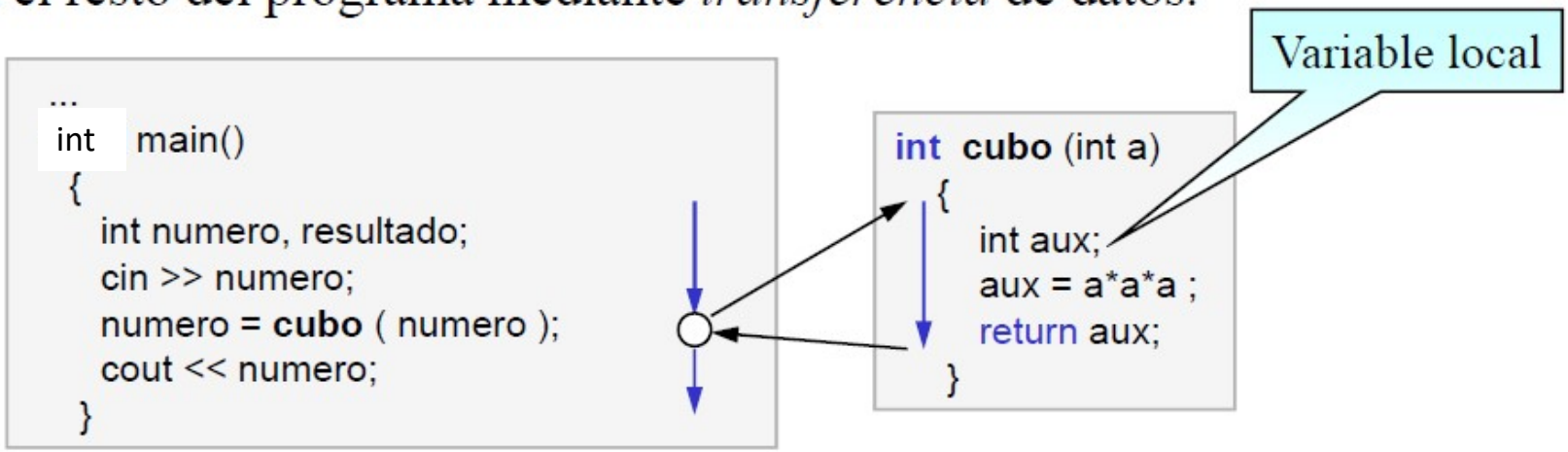
4. El valor devuelto puede ser: una constante, variable ó una expresión.

Estructura de una función: Lista de parámetros

Las funciones trabajan con dos tipos de datos:

1. Variables locales: declaradas en el cuerpo de la función. Estas variables solo son conocidas dentro de la función y se crean y se destruyen con la función.

2. Parámetros: Los parámetros permiten la comunicación de la función con el resto del programa mediante *transferencia* de datos.



Características importantes relativas a funciones

1. La instrucción **return**

   a) fuerza la salida inmediata de la función.

   b) sirve para devolver un valor. Dicho valor puede ser constante, variable ó una expresión.

return (4+i);	**return 7;**	**return x;**

2. No se pueden declarar unas funciones dentro de otras.

   (No se pueden declarar funciones anidadas)

3. Las constantes, variables y tipos de datos declarados en el cuerpo de la función son locales a la misma y no se pueden utilizar fuera de ella.

4. El cuerpo de la función encerrado entre llaves, no acaba en ';'.

Estructura de una función: Lista de parámetros

Paso de parámetros por valor

- Cuando se llama a la función, se pasa solo el valor de la variable.
- Este método también se llama *paso por copia*.
- El compilador hace una copia *de los parámetros*. Esto implica que cualquier modificación en el valor de los parámetros no se mantiene cuando termina la función.
- Utilizaremos este método cuando no necesitemos que se modifiquen los parámetros con los que se llama.

Todos los ejemplos que hemos visto hasta ahora, utilizan parámetros por valor.

Estructura de una función: Lista de parámetros

Paso de parámetros por referencia

- También se llama *paso por dirección.*

- Cuando se llama a la función, se pasa la dirección de memoria donde se encuentra almacenada la variable parámetro.

- El compilador no hace copia, no reserva memoria para los parámetros.

- Usaremos este método cuando necesitamos que la función modifique el valor de los parámetros y que devuelva el valor modificado.

Para pasar un parámetro por referencia, hay que poner el operador de dirección & detrás del tipo del parámetro.

```
void  cubo (int & a)
  {       ....      }
```

Ejemplo de uso de paso de parámetros

```
int main()
{
   int m;
   m = area_rectangulo( 2 , 3 );
   cout << m ;

   int lado1 = 2, lado2 = 6 ;
   m = area_rectangulo( lado1 , lado2 );
   cout << m;

   int b = 10, e = 4, r= 0;
   potencia (b, e, r);
   cout << r;

   }
```

```
int area_rectangulo (int a, int b)
{
   int aux;
   aux = a*b;
   a=0;
   b=0;
   return aux;
}
```

```
void potencia( int x, int y, int& z)
{
    z = 1;
    for ( int i=1;  i<= y ; i++ )
        z = z * x ;
}
```

Parámetros por valor: a, b, x, y

Parámetros por referencia: z