PROGRAMACIÓN I
C++

UCM

Grado en Estadística Aplicada. EUE.

2

Tema 4.- Programación modular

Isabel Riomoros

Programación con subprogramas

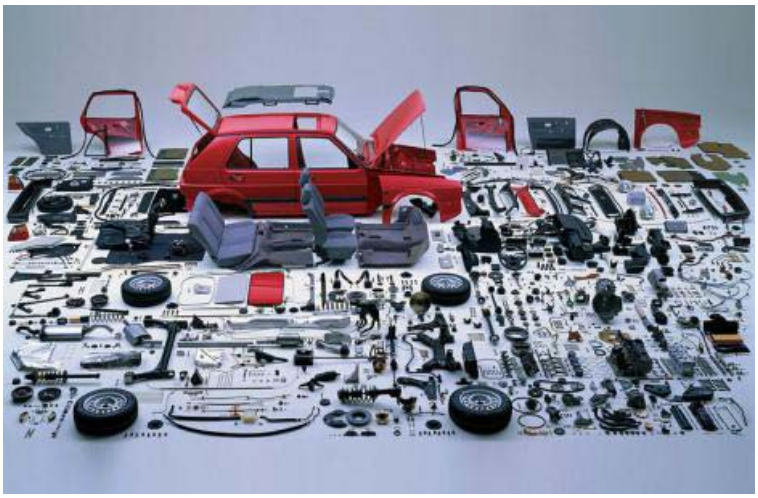
TEMA 4

- Introducción.
- Diseño descendente.
- Funciones.
  - Definición. Transferencia (parámetros)/Retorno.
  - Llamada a funciones. Parámetros reales y actuales.
  - Declaración de funciones: Prototipos
- Parámetros de entrada, salida y entrada/salida
  - Por valor
  - Por referencia
- Ámbito y visibilidad.
- Ejemplos de funciones.
- Ejercicios.

C++

Introducción

- La **abstracción** es la herramienta mental que nos permite analizar, comprender y construir sistemas complejos.
- Identificamos y denominamos conceptos abstractos con significado. Aplicamos Refinamientos Sucesivos.



C++

Introducción

- A veces, un determinado problema complejo lo podemos **(y debemos**) dividir en problemas más sencillos. Estos subproblemas se conocen como **subprogramas**.

- A la hora de diseñar el programa:

  - Descomponemos el problema original en **subproblemas**
    - Qué subproblema resuelve, qué datos necesita, qué datos produce
    - Bajo que condiciones se debe ejecutar

  - Se pueden codificar de forma independiente e incluso por diferentes personas.

  - El problema final queda resuelto y estructurado con subprogramas, así es más sencilla su lectura y mantenimiento.

C++

Diseño descendente: abstracción y refinamientos

- Una aplicación informática suele ser compleja y estar compuesta por miles de líneas de código
- Necesitamos descomponer el programa en una serie de subprogramas más pequeños y simples

Diseño descendente

Ventajas

- Descomponer la complejidad del problema
- Reutilizar del código en otro contexto
- Aumentar la legibilidad
- Disminuir el número de errores y si los hay que estén localizados
- Facilitar la reutilización del subprograma en otro contexto

C++

Subprogramas- Funciones

¿Qué subproblema resuelve?

¿Qué datos necesita?

¿Qué datos produce?

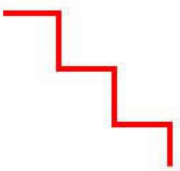
¿Bajo qué condiciones se debe ejecutar?

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***

C++

Dibujar una escalera



Leer datos (numEscalones, numH, numV)
Dibujar escalera
    Desde 1 hasta el numEscalones hacer
        DibujarHorizontal
        Dibujar vertical

C++

Definición de una función

Sintaxis

```
<tipoResultado> <nombreFuncion> ( parametrosFormales ) {
        cuerpo_de_la_función ;
        return  <expresión> ;
}
```

Palabra reservada

Tipo_resultado: Es el tipo de dato que devuelve la función. void, int, float, double, char, bool, struct

Nombre de la función- Nombre significativo que resuma para que sirve la función.

Cuerpo de la función- instrucciones necesarias para resolver el problema.

Lista de parámetros: aparecen con su tipo. La función utiliza éstos valores en el cuerpo.

return,

- ***Expresión:*** valor que devuelve la función, puede ser una constante, expresión, una variable,… pero siempre compatible con el **tipoResultado**.
- Puede haber más de un return, **NO** es aconsejable
- En cuanto se ejecute el return se termina la función
- **NO** hay return, si el **tipoResultado** es void.

C++

Tipo de datos de retorno

Las funciones en C++ las podemos dividir en:

- Funciones que realizan una tarea específica pero que **devuelven cero o varios valores** al programa principal o a la función que la llamó.

  El tipo de dato de retorno ha de ser → **void**

- Funciones que realizan operaciones con los argumentos o manipulan datos y **devuelven un valor**. Dicho valor, puede ser el resultado de esas operaciones ó un indicador de si la manipulación de los datos ha sido un éxito o no.

Si la función devuelve un valor, ha de ser uno de los siguientes:

- Tipos simples: **int**, **char**, **float**, …
- Tipos definidos: Un tipo **struct**
- Un puntero a cualquier tipo C++

Lo veremos más adelante

C++

Ejemplo


```
int  maximo (int a, int b )  {
        int m;
        if (a<b)
            m=b;
        else
            m=a;
      return m;
    }
```

Variable local

Dibujar num asteriscos

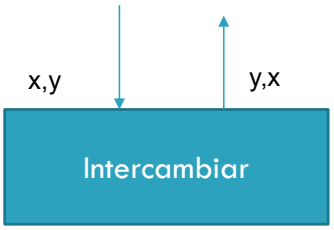
```
void  dibujarAsteriscos(int  num)  {
        For(int i=1;i<=num;i++){
            cout<<"*";
        }
}
```

No hay return

No devuelve ningún valor

C++

Intercambiar el valor de dos variables



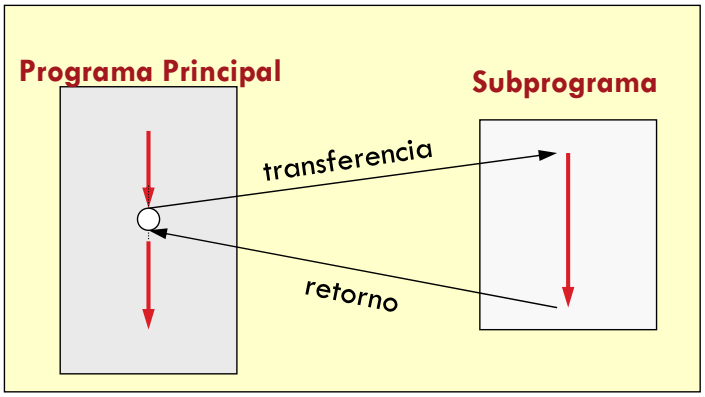

```
void intercambiar(int & x, int & y){
    int aux;
    aux=x;
    x=y;
    y=aux;
 }
```

devuelve dos valores

C++

Transferencia-retorno

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



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

C++

Transferencia / Retorno:


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

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

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

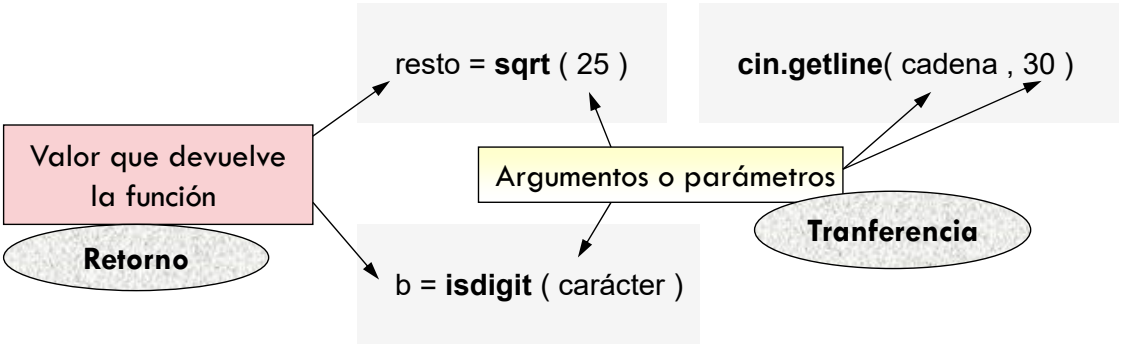
```
float cubo (int a)
  {
     ....
     return ...;
  }
```

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

C++

Transferencia / Retorno:

- 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**.

C++

Llamada a una función

- Una vez que se ha diseñado y codificado una función, se puede usar. Para usar una función, debemos ***llamarla*** ***o*** ***invocarla***. Una llamada, produce la ejecución de las instrucciones que se encuentran en el cuerpo.
- Se llama a la función con el nombre y los parámetros actuales correspondientes.

Programa principal

```
int main()   {
        int x, y, mayor ;
        cin >> x ;
        cin>> y ;
        mayor = maximo( x, y);
        cout << mayor;
   }
```

```
int maximo ( int a, int b ) {
        int m;
        if (a<b)
            m=b;
        else
            m=a;
    return m;
   }
```

maximo : int × int → int

C++

Llamada a una función

La llamada se realiza mediante el nombre seguido por los **parámetros actuales**.

```
nombreFuncion(listaParametrosActuales)
```

- La llamada **a una función** cuyo **tipoResultado es distinto de void** no puede constituir por si sola una instrucción, sino que debe aparecer dentro de alguna estructura que utilice el valor resultado de la función.

```
mayor = maximo( x, y);
```

- La llamada **a una función** cuyo **tipoResultado es void** constituye por si sola una instrucción que puede ser utilizada como tal en el cuerpo de subprogramas y del programa principal.

```
mostrarH();
```

- Un subprograma puede invocar a otros subprogramas (definidos previamente)

C++

Escribir Hola


```
void mostrarH() {
   cout << "*   *" << endl;
   cout << "*   *" << endl;
   cout << "*****" << endl;
   cout << "*   *" << endl;
   cout << "*   *" << endl <<endl;
}


 void mostrarO() {
    cout << "*****" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*****" << endl << endl;
 }
```

```
int main() {
   mostrarH();
   mostrarO();
   mostrarL();
   mostrarA();
   return 0;
}
```

C++

Menú


```
void menu(){
  int op;
  cout << "1 – Escribir HOLA" << endl;
  cout << "2 – Escribir ADIOS" << endl;
  cout << "0 – Cancelar" << endl;
  cout << "Opción: ";
  cin >> op;
  if (op == 1) {
    escribirHola();
  }
  else if (op == 2) {
    escribirAdios();
  }
}
```

C++

Llamada a una función

- Cuando se produce una llamada a un subprograma:

1. Se establecen las vías de comunicación entre los programas o subprogramas llamante y llamado (**parámetros**).

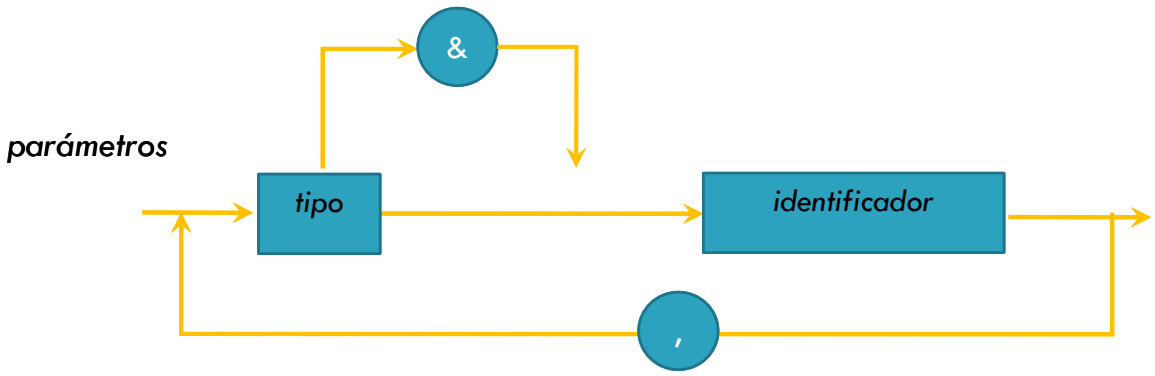
2. Se crean las variables locales.

3. El flujo de control pasa a ejecutar la primera instrucción del cuerpo del subprograma llamado, ejecutándose este.

4. Cuando finaliza la ejecución del subprograma, las variables locales previamente creadas se destruyen y el flujo de control continua por la siguiente instrucción a la llamada realizada.

C++

Parámetros



C++

Nombre de la función

El nombre que se les da a las funciones, debe ser un identificador válido, es decir,

- Debe comenzar con una **letra** o subrayado (_).
- Después de la primera letra pueden aparecer otras letras, dígitos y subrayado (_)
- No debe contener espacios en blanco.
- C++ distingue entre mayúsculas o minúsculas.

Nombres de funciones: **leer** , **visualizarTabla1** , **leerMatriz** , etc ...

Es muy importante en la fase de diseño de un algoritmo, utilizar nombres que nos permitan intuir la tarea que realizan las funciones, sobre todo a la hora de mantener y modificar programas.

C++

Nombre de la función

Por ejemplo, qué hacen los siguientes programas:

```
...
int main()   {
            ....
        llenarMatriz();
        calcularMedia();
        mayoresDeLaMedia();
        imprimirMayores();
   }
```

```
...
int main()  {
            ....
            primera_funcion();
            segunda_funcion();
            funcion_3();
            mi_funcion();
   }
```

Parece más intuitivo, ¿no?

Si se nos pide un cambio en algún punto del programa, por ejemplo al actualizar la matriz,

¿qué función hemos de modificar?

C++

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.

C++

Ejemplos


```
char siguiente_car (char c ) {
    ....
   return car;
   }
```

```
bool encontrado ( ) {
    ....
   return enc;
  }
```

```
bool funcion( int a ) {
   bool negativo;
    if (a <0) {
        negativo = true;
        return negativo;
      }
    while (a < 100) {
       cout << a;
       a++;
     }
    return false;
  }
```

```
float media (float x, float y ) {
    ....
  return med;
  }
```

```
int main(){
  bool resultado;
  resultado = funcion (-5);
  resultado = funcion (5);
}
```

C++

```
Introduzca numero de filas : 9
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
```

```
//#include <iostream>
using namespace std ;
const char SIMBOLO = '*';
void escribirCaracter ( int n, char simb ){
  for (int i = 0; i < n; i++)
     cout << simb ;
}
void escribirFila ( int f, int nf){
   escribirCaracter (nf-f-1, ' ');//escribir blancos
   escribirCaracter (2*f+1, SIMBOLO ); // escribir asteriscos
   cout << endl ;//saltar de línea
}
void escribirTriangulo ( int nf){
  for (int f = 0; f < nf; ++f)
     escribirFila (f, nf );
}
//..
```

Escribir el primer perfecto a partir de un número

```
#include <iostream>
using namespace std ;
int PRIMER_NUMERO = 29;
int sumaDivisores (int n){
  int suma = 0;
  for (int i = 1; i <= n / 2; i++)
     if (n % i == 0)
          suma += i;
  return suma ;
}
bool esPerfecto (int n){
  return n == sumaDivisores (n);
}
int primerPerfecto (){
  int i = PRIMER_NUMERO ;
  while (! esPerfecto (i))
    i++;
return i;
}
```

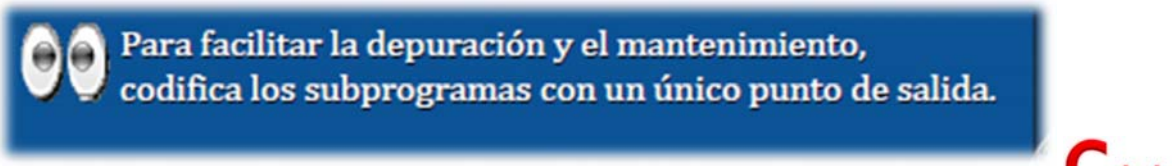
```
int main (){
  cout << " Primer perfecto mayor que " << PRIMER_NUMERO << ": " ;
  cout<< primerPerfecto () << endl ;
  system("pause");
  return 0;
}
```


Ejemplos


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

```
bool  dividir ( int a , int b, float & cociente )  {
     if   ( b = 0 )
        return false;
     else
          cociente = a/b;
     return true;
}
```

```
int main(){
   int resultado;
   bool ok;
   resultado = sumaTres (2,  x, y );
   ok = dividir (0, 3, resultado);
   if (ok ==true)
        cout << resultado;
   else
       cout << "error-división   por cero";
   cout << resultado;
}
```



C++

Declaración de las funciones: Prototipos

- A excepción de la función main(), en el módulo del programa debe aparecer la declaración de las funciones que se utilicen en dicho módulo. Esta declaración recibe el nombre de PROTOTIPO.

- **<tipo_resultado> <nombreFunción> (*listaParámetrosFormales*) ;**

Sintaxis del prototipo

```
#include <iostream>
....

 void  potencia (int x, int y, int& z );

int main()   {

 ...
}
 void potencia( int x, int y, int& z)  {
  ....
 }
```

Prototipo

Codificación

El prototipo, informa de la existencia de la función, el tipo de datos que devuelve y los parámetros que tiene definidos.

C++

Declaración de las funciones: Prototipos

También podemos definir la función de la siguiente manera:

```
#include <iostream>

void potencia( int x, int y, int& z)  {


  ....
 }


int main()    {


 ...
 }
```

C++

Ejemplos sencillos

- Escribir una función que:
  1. Dados dos valores enteros nos devuelva el mayor
  2. Dado un carácter nos devuelva el siguiente
  3. Dados dos números reales nos devuelve la media
  4. Dado un número y una cifra nos devuelve true si la cifra forma parte del número y false en caso contrario
  5. Factorial de un número. Dado un número nos devuelve su factorial
  6. Potencia. Dada la base (x) y el exponente(n) nos devuelve x^n
  7. Algoritmo de Euclides. Dados dos números enteros nos devuelve su máximo común divisor

- Escribir un programa que llame a cada una de las funciones

C++

Cálculo de la integral

Dividir [a,b] en n partes iguales.$[a,x_1],[x_1,x_2],...[x_{n-1},b]$.

Hacer h=(b-a)/n.

$$\int_a^b f(x)dx = h/2(f(a)+f(x_1))+h/2(f(x_1)+f(x_2))+...+h/2(f(x_{n-1})+f(b))$$

Hacer un programa que calcule integrales por el método anterior. Probar con distintos valores de n. La función es: **$f(x)=x^4 + x^3 + x$.**

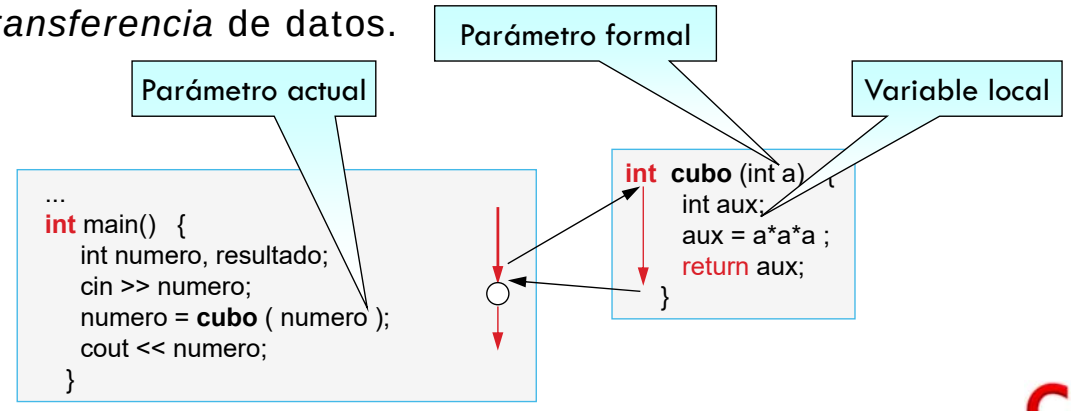
C++

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.

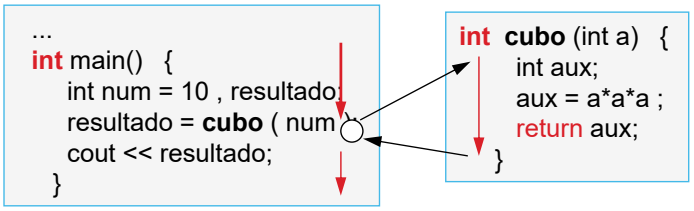


C++

Lista de parámetros

- C++ proporciona dos métodos para realizar ésta transferencia de datos a la función. Hablaremos a partir de ahora de *paso de parámetros a la función.*

1. Paso de parámetros por valor **(int x)**

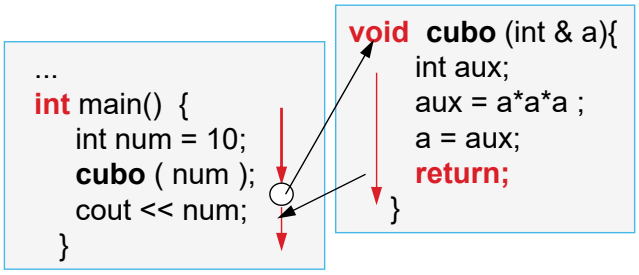


* El programa principal se interrumpe para comenzar la ejecución de la función
* Se reserva memoria para el código de la función, para las variables locales y para los parámetros.
* El compilador hace una copia *de los parámetros.* Cualquier modificación en el valor de los parámetros no se mantiene cuando termina la función.
*Utilizaremos éste método cuando no necesitemos que se modifiquen los parámetros con los que se llama.

C++

Lista de parámetros

2. Paso de parámetros por referencia (int& x)



* El compilador no reserva memoria para los parámetros, sino que utiliza la misma porción de memoria.

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

* Usaremos éste método cuando necesitamos que la función modifique el valor de los parámetros y que devuelva el valor modificado

C++

Parámetros

- El número de parámetros actuales debe coincidir con el de parámetros formales.
- Correspondencia posicional entre parámetros actuales y formales.
- El tipo del parámetro actual debe coincidir con el tipo del correspondiente parámetro formal.
- Un parámetro formal de salida o entrada/salida requiere que el parámetro actual sea una variable.
- Un parámetro formal de entrada requiere que el parámetro actual sea una variable, constante o expresión.

C++

Ejemplo de uso de paso de parámetros


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

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

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

   }
```

```
int areaRectangulo (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

C++

Ordenar tres valores


```
void ordenarTres(float & a, float &b, float & c){
   if (a>b)
      intercambiar(a,b);
   if (b>c)
      intercambiar(b,c);
   if (a>b)
      intercambiar(a,b);
}
```

C++

¿Qué llamadas son correctas?

Dadas las siguientes declaraciones:

```
int i;
double d;
void proc(int x, double & a);
```

¿Qué llamadas son correctas?

1. `proc(3, i, d);`
2. `proc(i, d);`
3. `proc(3*i + 12, d);`
4. `proc(i, 23);`
5. `proc(d, i);`
6. `proc(3.5, d);`
7. `proc(i);`

C++

```
int main() {
  int a = 1, b = 2, c = 3;
  otro(a, b, c);
  cout << "main dice: ";
  cout << "a = " << a << " b = " << b << " c = " << c << endl;
  otro(b, a ,c);
  cout << "main dice: ";
  cout << "a = " << a << " b = " << b << " c = " << c << endl;
  return 0;
}

void otro(int a, int &b, int c) {
  c= a + b;
  b = a;
  a = c;
  cout << "otro dice: ";
  cout << "a = " << a << " b = " << b << " c = " << c << endl;
}
```

Indica, sin usar un compilador, cuál será la salida del programa.

otro dice: A = 3 B = 1 C = 3
main dice: A = 1 B = 1 C = 3
otro dice: A = 2 B = 1 C = 2
main dice: A = 1 B = 1 C = 3

Dado el siguiente programa en C++:

```
int main() {
   char a = 'B', b = 'C', c = 'A', d = 'D';
   vueltas(a, b, c, d);   // 1ª llamada
   vueltas('A', b, c, a); // 2ª llamada
   vueltas('D', c, c, a); // 3ª llamada
   return 0;
}

void vueltas (char a, char &b, char &d, char &c) {
   b = a;
   c = d;
   d = 'B';
}
```

Rellena la siguiente tabla con el valor de las variables del programa principal después de cada llamada al procedimiento vueltas().

	a	b	c	d
Valor inicial	'B'	'C'	'A'	'D'
Después de la 1ª llamada				
Después de la 2ª llamada				
Después de la 3ª llamada				

Cálculo del n-ésimo número primo


```
#include <iostream>
using namespace std;
int leeNumero();
bool esPrimo(int n);
void escribeResultado(int numPrimo,int valor);
int main() {
   int n, numPrimos = 0, i = 0;
   n = leeNumero();
   while (numPrimos < n) {
      i++;
      if (esPrimo(i))
         numPrimos++;
}
    escribeResultado(n,i);
    system("pause");
    return 0;
}
```

C++