PROGRAMACIÓN I
C++

UCM

Grado en Estadística Aplicada. EUE.

Tema 4.- Programación modular

Isabel Riomoros

Programación con subprogramas

TEMA 4

- Introducción.
- Diseño descendente.
- Subprogramas.
  - Definición
  - Transferencia/Retorno
- Estructura de una función.
  - Nombre de una función
  - Tipo de retorno (tipo del valor devuelto)
  - Valor retorno
  - Lista de parámetros
    - Paso por valor
    - Paso por referencia
- Declaración de funciones: Prototipos.
- Llamada a funciones.
- Ejercicios y ejemplos.

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:

  Técnica de diseño conocida como **TOP DOWN**

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

C++

Diseño descendente: abstracción y refinamientos

- Divide un problema en subproblemas
- Asocia a cada subproblema un **subprograma**
- Utilizamos **abstracción**:
  - NO importa como se resuelve un subproblema
  - Nos centramos en la funcionalidad del subprograma
  - Que subproblema resuelve, que datos necesita, que datos produce
  - Bajo que condiciones se debe ejecutar
- Aplicamos **refinamientos sucesivos**
  - Objetivo: división en **subprogramas independientes**
  - Funcionalidad clara y bien definida
  - Aislados: minimizar las dependencias con otros subprogramas
  - Transferencia de información simple y bien definida

C++

Diseño descendente: ventajas

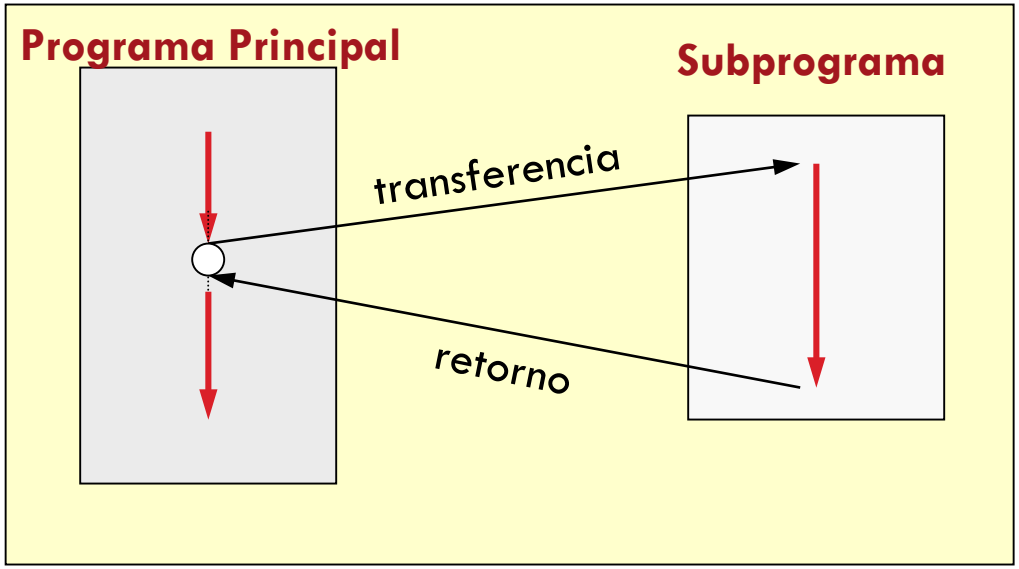
- Los subprogramas encapsulan y aíslan las diferentes tareas que componen un programa.
- Simplificación en el diseño y solución del programa.
- Si el método para solucionar una tarea debe cambiar, el aislamiento evita que dicho cambio influya en las otras tareas.
- Permite que el programador este concentrado en la solución individual del subproblema concreto.
- Simplifica la comprensión y legibilidad del programa.
- Facilita la detección y corrección de errores (depuración).
- Posibilidad de reutilización del subprograma en otro contexto.

C++

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.



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

Definición de subprogramas

- El cuerpo del subprograma especifica la secuencia de acciones que resuelven un subproblema.
- Define sus propias variables locales de trabajo.
- Donde sea necesaria la resolución de dicho subproblema, se realizara una llamada al subprograma.
- La transferencia de información se realiza a través de los parámetros:
  - **Parámetros Formales**: aparecen en la definición del subprograma.
  - **Parámetros Actuales**: aparecen en la llamada al subprograma.
- Abstracción de procesamiento y de operaciones
  - **Funciones**: calculan un único valor a partir de la información de entrada.
  - **Procedimientos:** procesamiento general de información.

C++

Definición de subprograma

- **Definición de Funciones**
  - El cuerpo de una función solo debe tener una sentencia return, se suele escribir la última sentencia del cuerpo de la función.
  - El valor resultado de la función es el resultado de evaluar la expresión de la sentencia return.

- **Definición de Procedimientos, a veces llamados funciones**
  - El cuerpo de un procedimiento no debe tener ninguna sentencia return.
  - Procesa información que se transfiere a través de los parámetros.

```
int menor (int x, int y)
{
    int z;
    if (x < y) {
        z = x;
    } else {
        z = y;
    }
    return z;
}
```

```
void ordenar (int& x, int& y)
{
    if (x > y) {
        int z = x;
        x = y;
        y = z;
    }
}
```

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 ...;
  }
```

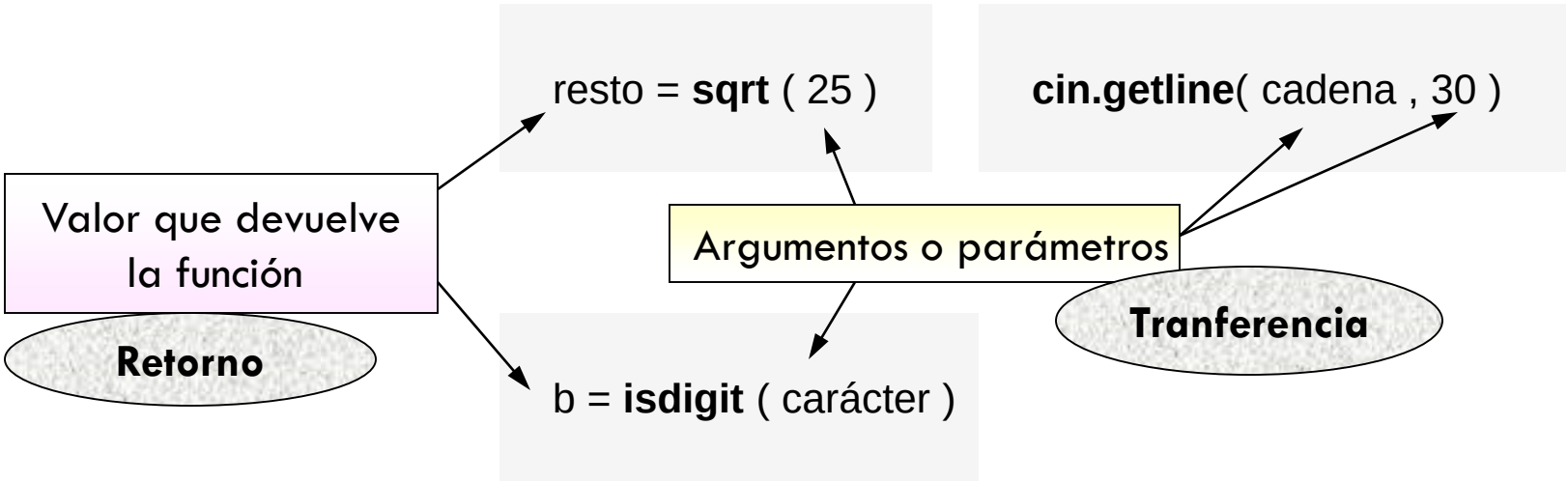
```
float cubo (int a)
  {
     ....
     return ...;
  }
```

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

C++

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**.

C++

Estructura de una función

Sintaxis

```
<tipo_resultado> <nombre_de_la_función> ( lista_de_parámetros )
{
        cuerpo_de_la_función ;
        return  <expresión> ;
 }
```

Palabra reservada

Tipo_resultado: Es el tipo de dato que devuelve la función.
Nombre de la función- Nombre significativo que resuma para que sirve la función.
Cuerpo de la función- instrucciones necesarias para resolver el problema.
Expresión: valor que devuelve la función.
Lista de parámetros: aparecen con su tipo. La función utiliza éstos valores en el cuerpo.

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

Variable local

C++

Estructura de 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 correspondientes.

Programa principal

```
int  main()
   {
        int x, y, mayor ;
        cin >> x >> 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++

Estructura de una función: 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 caracteres.
- No debe contener espacios en blanco.
- C++ distingue entre mayúsculas o minúsculas.

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

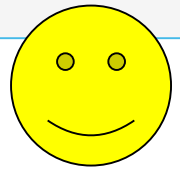
Es muy importante en la fase de diseño de un algoritmo, utilizar nombres que nos permitan intuir la tarea que realizan las funciones, sobre todo a la hora de mantener y modificar programas.

C++

Estructura de una función: Nombre de la función

Por ejemplo, qué hacen los siguientes programas:

```
...
int main()
 {
          ....
      llenar_matriz();
      calcular_media();
      mayores_que_la_media();
      imprimir_mayores();
 }
```



Parece más intuitivo, ¿no?

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

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

¿qué función hemos de modificar?

C++

Estructura de una función: Tipo de dato de retorno

Las funciones en C++ las podemos dividir en varios tipos:

- Funciones que realizan una tarea específica pero que **no devuelven valores** al programa principal o a la función que la llamó.

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

  Se llaman *Procedimientos*

- 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 exitosa 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++

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 struct).

2. Una vez que se ejecuta ésta 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.

C++

Estructura de una función: Valor de retorno


```
int maximo (int a, int b )
  {
     .......
    return max;
  }
```

```
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++

Estructura de una función: Valor de retorno


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

Para facilitar la depuración y el mantenimiento, codifica los subprogramas con un único punto de salida.

C++

Ejemplos sencillos

- Escribir un 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++

Estructura de una función: Valor de retorno

Cuando se llama a una función, debe haber una variable que guarde el valor que devolverá la función, es decir, llamaremos a la función mediante una sentencia de asignación, por ejemplo:

```
resultado = suma (6 , 8 );
```

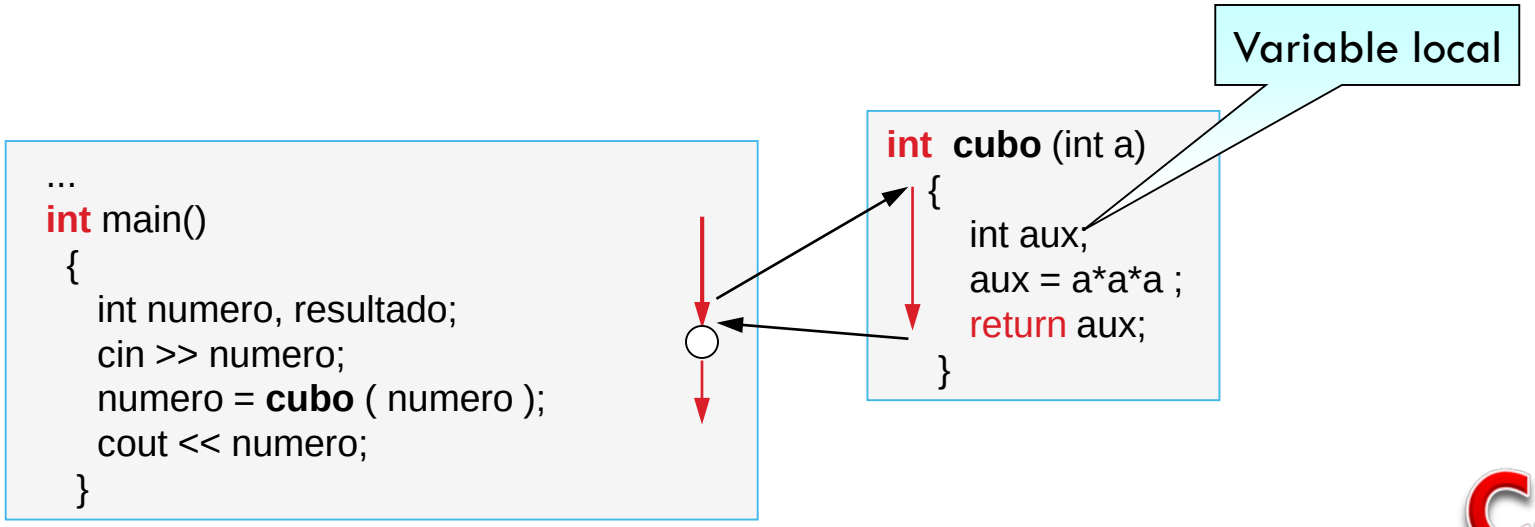
C++

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.



C++

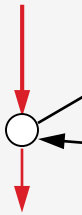
Estructura de una función: Lista de parámetros

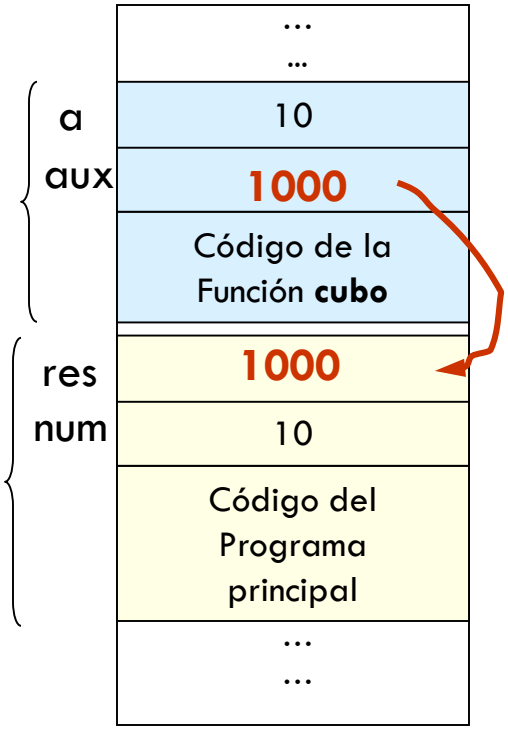
- C++ proporciona dos métodos para realizar ésta transferencia de datos a la función. Hablaremos a partir de ahora de *paso de parámetros a la función.*

1. Paso de parámetros por valor

```
...
int main()
 {
    int num = 10 , res;
    res = cubo ( num );
    cout << res;
 }
```

```
int  cubo (int a)
 {
    int aux;
    aux = a*a*a ;
    return aux;
 }
```





* El programa principal se interrumpe para comenzar la ejecución de la función

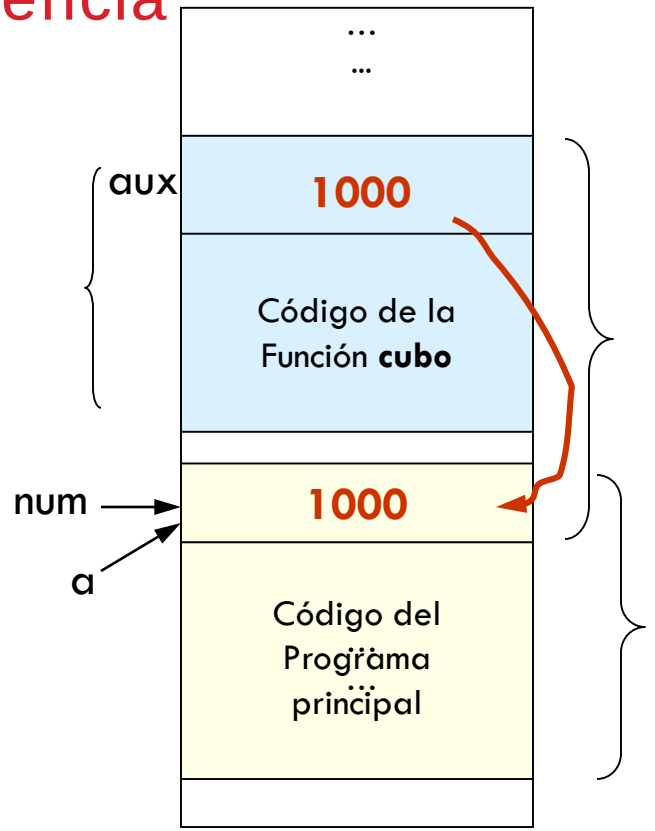
* Se reserva memoria para el código de la función, para las variables locales y para los parámetros.

Estructura de una función: Lista de parámetros

2. Paso de parámetros por referencia

```
...
int main()
 {
    int num = 10;
    cubo ( num );
    cout << num;
 }
```

```
void  cubo (int & a)
   {
      int aux;
      aux = a*a*a ;
      a = aux;
      return;
   }
```



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

C++

Estructura de una función: Lista de parámetros

Paso de parámetros por valor (int x)

- 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 éste método cuando no necesitemos que se modifiquen los parámetros con los que se llama.

C++

Estructura de una función: Lista de parámetros

Paso de parámetros por referencia (int& x)

- 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 éste 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)
  {      ....     }
```

C++

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

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++

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> <nombre_de_la_función> (*lista_de_parámetros*) ;

Sintaxis del prototipo

```
#include <iostream.h>

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++

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 ';'.

C++

Funciones: llamada

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

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

- La llamada **a un procedimiento** constituye por si sola una sentencia que puede ser utilizada como tal en el cuerpo de subprogramas y del programa principal.

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

C++

Funciones: llamada

- Cuando se produce una llamada a un subprograma:

1. Se establecen las vías de comunicación entre los algoritmos 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++

Funciones: 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++

Parámetros con referencia constante

- *Para evitar copias en el paso de parámetros por valor, los parámetros de entrada se pueden pasar por referencia constante.*

const *tipo_Parámetro & nombre_Parámetro*

Ejemplo:

void func(const string & nomb);

- *En el cuerpo de la función, el parámetro nomb, es una constante local.*
- *En las llamadas, el argumento tiene que ser una variable. Se pasa la referencia de la variable.*
- *Recomendado cuando el tipo del parámetro no es básico, y no se necesita una variable local.*

C++

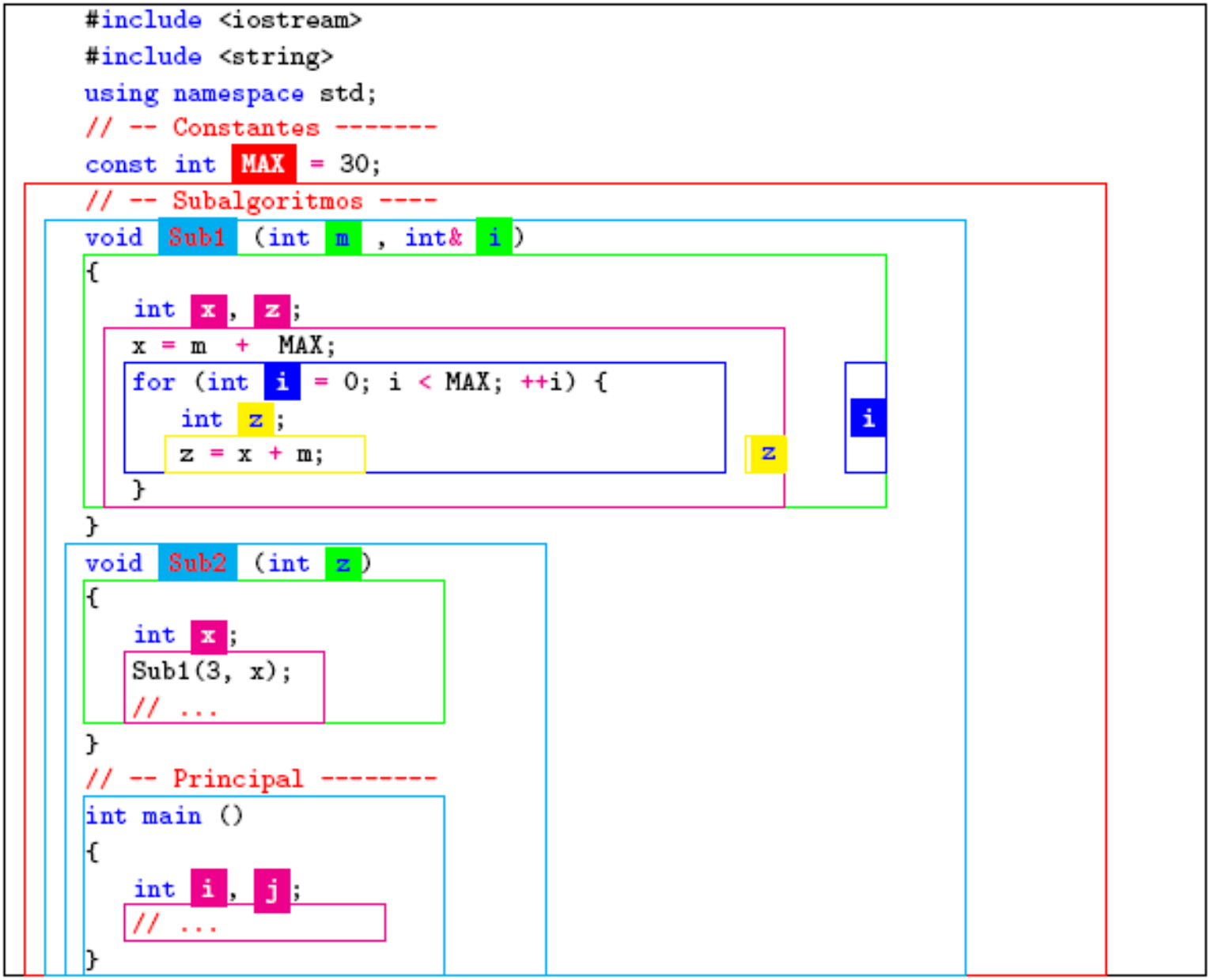
Reglas de ámbito

1. Un identificador es visible y accesible en la zona comprendida entre el lugar en que se declara (creación) y el final del bloque o cuerpo del subprograma donde ha sido declarado (destrucción). A esta zona se le denomina ***ámbito o zona de visibilidad del identificador***.

2. Si dentro del ámbito de un determinado identificador, este mismo aparece declarado en un nivel de anidamiento (bloque) mas interno, entonces dentro del ámbito de esta segunda declaración, la declaración mas externa quedará ocultada (y por lo tanto no accesible).

3. Si la variable de control del bucle for se declara en la zona de inicialización del mismo, entonces su ámbito de visibilidad se extiende hasta el final del cuerpo del bucle.

C++


#include <iostream>
#include <string>
using namespace std;
// -- Constantes -------
const int MAX = 30;
// -- Subalgoritmos ----
void Sub1 (int m , int& i)
{
int x, z;
x = m + MAX;
for (int i = 0; i < MAX; ++i) {
int z;
z = x + m;
}
z
i
}
void Sub2 (int z)
{
int x;
Sub1(3, x);
// ...
}
// -- Principal --------
int main ()
{
int i, j;
// ...
}

Subprogramas en línea

- La llamada a un subprograma conlleva un pequeño coste debido al control y gestión de la misma que ocasiona cierta perdida de tiempo de ejecución.

- Cuando el subprograma **es muy pequeño** y nos interesa eliminar el coste asociado a su invocación, podemos hacer que el subprograma se traduzca como código en línea en vez de como una llamada a un subprograma.

- Se escribirá la palabra reservada **inline** justo antes del tipo. Así tendremos los beneficios de la abstracción, pero se eliminan los costes asociados a la invocación.

```
inline int calcular_menor(int a, int b)
{
  return (a < b) ? a : b ;
}
```

C++

Sobrecarga de subprogramas y operadores

- Se denomina sobrecarga cuando distintos subprogramas se denominan con el mismo identificador u operador, pero se aplican a parámetros distintos.

- En C++ es posible sobrecargar tanto subprogramas como operadores siempre y cuando tengan parámetros diferentes, y el compilador pueda discriminar entre ellos por la especificación de la llamada.

```
void imprimir(int x)
{
   cout << "entero: " << x << endl;
}
void imprimir(double x)
{
   cout << "real: " << x << endl;
}
```

C++

Ejercicios


```
void escr(char ch, int longitud)
{
  while (longitud >0){
        putchar(ch);
        longitud--;
        }
}
```

putchar(c)- escribe el carácter c en el output

getchar()- función lee un caráter del input

a) Si `ch` tiene el valor `'X'` y `numero` el valor 5, ¿cuál sería el efecto de ejecutar cada una de las siguientes llamadas a la función:

```
escr(ch,4*numero-12)      escr(ch,6)      escr(5,numero)
escr('/',numero)          escr('.',6)     escr('p',-10)
```

b) Escribe llamadas a la función `escr` para que cuando se ejecuten produzcan las siguientes salidas: 1) 35 guiones sucesivos; 2) 6 veces tantos espacios en blanco como el valor de `numero`; 3) el valor actual de `ch` 14 veces.

C++

Ejemplo: Ejecutar y ver la salida


```
//Ejemplo de parámetros por valor y referencia
#include <iostream>
using namespace std;
void dos(int x, int y, int& z)
{
  z=x + y + z;
  cout<<"x= "<<x<<" y= "<<y<<" z= "<<z<<endl;
}
main ()
{
  int a=5,b=8,c=3;
  dos(a,b,c);
  cout<<" dos(a,b,c)   a= "<<a<<" b= "<<b<<" c= "<<c<<endl;
  dos(7,a+b+c,a);
  cout<<" dos(7,a+b+c,a)   a= "<<a<<" b= "<<b<<" c= "<<c<<endl;
  dos (a*b, a/b,c);
  cout<<" dos (a*b, a/b,c)   a= "<<a<<" b= "<<b<<" c= "<<c<<endl;
  system("pause");
  return 0;
}
```

C++

¿Cuál será la salida?

```
#include <iostream>
using namespace std;
int a,b,c,x,y;
void primero()
{
  a=3*a;
  c=c+4;
  }
void segundo()
{
  int b;
  b=8;
  c=a+b+c/3;
  }

}
```

```
void tercero(int x, int y)
{
  x=x+4;
  y=y+1;
  cout<<"a= "<<a<<" b= "<<b<<" c= "<<c<<" x= "<<x<<" y= "<<y<<endl;
}
main ()
{
  a=3;b=2;c=1;x=11;y=22;
  primero();
   cout<<"d primero a= "<<a<<" b= "<<b<<" c= "<<c<<" x= "<<x<<" y= "<<y<<endl;
  segundo();
   cout<<"d de 2º a= "<<a<<" b= "<<b<<" c= "<<c<<" x= "<<x<<" y= "<<y<<endl;
  tercero(a,b);
   cout<<"d de tercero a= "<<a<<" b= "<<b<<" c= "<<c<<" x= "<<x<<" y= "<<y<<endl;
  system("pause");
  return 0;
```

Ejemplo


```
unsigned int a,b,c;
int si;
int uno(unsigned int x,unsigned int y)
    ......
void dos (unsigned int& x, unsigned int y)
    ......
unsigned int tres(unsigned int x)
```

¿ Cuáles de las siguientes llamadas son válidas?

```
a) if (uno(a,b))  ...        b) dos(a,b+3);
c) si = uno(c,5);            d) si=dos(c,5);
e) dos(a,tres(a));           f) dos(tres(b),c);
g) if (tres(a)) ...          h) b=tres(dos(a,5));
i) dos(4,c);                 j) si=uno(dos(a,5));
```

C++

Ejercicios propuestos

- ***Dibujar una escalera***, solicitando el margen el número de espacios en horizontal, el numero de vertical y el número de peldaños.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

- ***"Números Amigos".***

  Dos números son amigos cuando la suma de los divisores del primero es igual al segundo y viceversa. Escribir un programa que permita encontrar al menos un par de amigos.

C++

Ejercicios propuestos

Escriba las siguientes funciones:

a) El volumen, v, de un cilindro está dado por la siguiente fórmula: $v = \pi r^2 l$ donde r es el radio del cilindro y l es la altura. Utilizando esta fórmula escriba una función llamada `volcil()` que acepte el radio y la altura de un cilindro y devuelva el volumen. Incluya esta función en un programa de prueba.

b) La superficie, s, de un cilindro está dada por la siguiente fórmula: $s = 2\pi r l$ donde r es el radio del cilindro y l es la altura. Utilizando esta fórmula escriba una función llamada `areasup()` que acepte el radio y la altura de un cilindro y que devuelva la superficie. Incluya esta función en un programa de prueba.

c) Realice una única función que acepte el radio y la altura de un cilindro y devuelva su volumen y su superficie con las fórmulas de los ejercicios anteriores.

C++

Ejercicio


```
#include <stdio.h>
void tiempo(int& min, int& hora);
int main() {
  int min,hora;
  printf("Introduzca dos números : ");
  scanf("%d%d", &min, &hora);
  tiempo(min, hora);

  return 0;
}
void tiempo(int& min, int& hora) {
  int seg;
  seg=(hora *60 + min) * 60;
  printf("El número total de segundos es %d", seg);
}
```

Contesta a las siguientes preguntas:

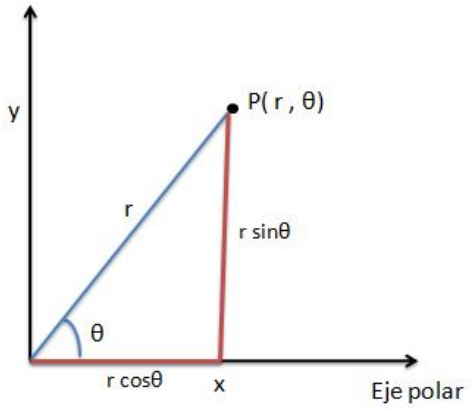
- ¿Qué hace? _____
- ¿Es adecuado el paso de argumentos? ¿Por qué? _____
- Si se cambiara el tipo de paso de argumentos, ¿habría alguna diferencia en la ejecución con respecto al programa anterior? _____
- ¿Hay algún problema al utilizar los mismos nombres en los argumentos y los parámetros tanto en la función que hace la llamada como en la llamada?

  _____

C++

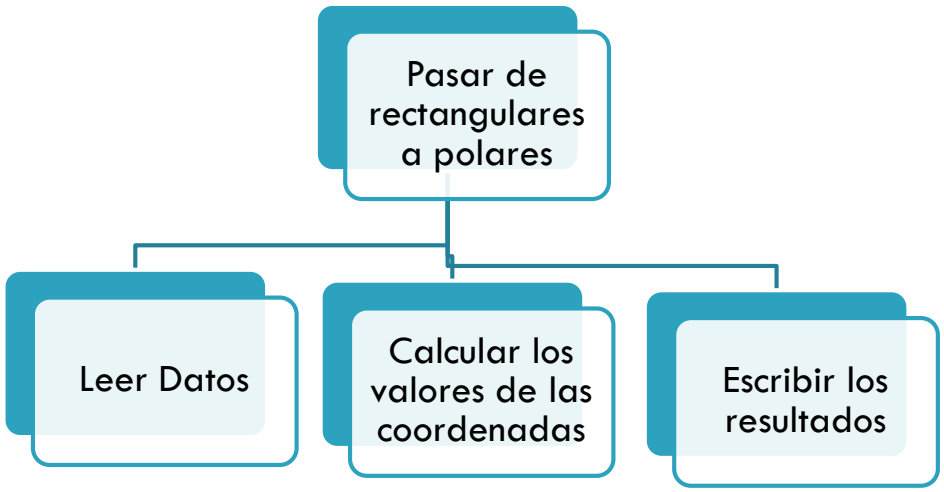
Pasar de coordenadas rectangulares a polares

Tenemos la coordenada rectangular de un punto (x,y) y queremos pasarlo a coordenadas polares



$$r = \sqrt{x^2 + y^2}$$

$$\theta = tan^{-1}\left(\frac{y}{x}\right) \quad x \neq 0$$



C++

Pasar de coordenadas rectangulares a polares


```
#include <iostream>
#include <math.h>
using namespace std;
void pasoPolares(float,float,float&,float&);
int main()
{
 float x,y,distancia,angulo;
 cout<<"Dame la x"<<endl;
 cin>>x;
 cout<<"Dame la y"<<endl;
 cin>>y;
 pasoPolares(x,y,distancia,angulo);
 cout<<"("<<x<<","<<y<<")"<<"--->"<< "r= "<<distancia<<" angulo= "<<angulo<<endl;
 system("pause");
 return 0;
}
void pasoPolares(float coorX, float coorY, float& r,float& a)
{
  const float AGRADOS=180.0/3.1416;
  r=sqrt(coorX*coorX+coorY*coorY);
  a=atan(coorY/coorX)*AGRADOS;
  return;
}
```

C++

Escribe un programa que lea un numero entero N por teclado y dibuje un triangulo de asteriscos con altura N.

Por ejemplo si N = 5 deberá dibujarse:

```
    *
   ***
  *****
 *******
*********
```

```
#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 );
//asteriscos
   cout << endl ;//saltar de línea
}
```

```
void escribirTriangulo ( int nf)
{
  for (int f = 0; f < nf; ++f)
      escribirFila (f, nf );
}
int main ()
{
  cout << " Introduzca numero de filas : ";
  int nFilas ;
  cin >> nFilas ;
  escribirTriangulo ( nFilas );
  system("pause");
  return 0;
}
```

C++

Ejercicios propuestos


```
    1
   121
  12321
 1234321
123454321
```

```
         1
        232
       34543
      4567654
     567898765
    67890109876
   7890123210987
  890123454321098
 9012345676543210
0123456789876543210
123456789010987654321
.....................
```

Escribe un programa que calcule el valor de S para un número real X $(0 \leq X \leq 1)$ dado por teclado, utilizando la serie de Taylor:

$$S = 1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \cdots$$

Diseña un programa que encuentre el primer número perfecto mayor que 28. Un número es perfecto si coincide con la suma de sus divisores (salvo él mismo).

Por ejemplo, 28 es perfecto ya que $28 = 1 + 2 + 4 + 7 + 14$

C++

Soluciones


```
#include <iostream>
using namespace std ;
const int MAX_FILAS = 10;
void escribirCaracter ( int n, char simb )
{
  for (int i = 0; i < n; ++i)
    cout << simb ;
}
void escribirAscendente ( int n)
{
  for (int i = 1; i <= n; ++i)
    cout << i;
}
void escribirDescendente (int n)
{
  for (int i = n; i >= 1; --i)
    cout << i;
}
```

```
1
121
12321
1234321
123454321
```

```
void escribirFila ( int f, int nf)
{
  escribirCaracter (nf-f, ' ');
  escribirAscendente (f);
  escribirDescendente (f- 1);
  cout << endl ;
}
void escribirTriangulo ( int nf)
{
  for (int f = 1; f <= nf; ++f)
    escribirFila (f, nf );
}
int main ()
{
  cout << " Introduzca numero de filas : ";
  int nFilas ;
  cin >> nFilas ;
  if ( nFilas < MAX_FILAS )
    escribirTriangulo ( nFilas );
  system("pause");
  return 0;
}
```

C++

Soluciones


```
#include <iostream>
using namespace std ;
void esc_caracter ( int n, char simb )
{
  for (int i = 0; i < n; ++i)
     cout << simb ;
}
void esc_ascendente ( int a, int b)
{
  for (int i = a; i <= b; ++i)
    cout << (i %10);
}
void esc_descendente (int a, int b)
{
  for (int i = a; i >= b; --i)
    cout << (i %10);
}
```

```
        1
       232
      34543
     4567654
    567898765
   67890109876
  7890123210987
 890123454321098
9012345676543210
0123456789876543210
1234567890109876543210
. . . . . .
```

```
void esc_fila ( int f, int nf)
{
  esc_caracter (nf-f, ' ');
  esc_ascendente (f, 2*f- 1);
  esc_descendente (2*f-2, f);
  cout << endl ;
}

void esc_triangulo ( int nf)
{
  for (int f = 1; f <= nf; ++f)
    esc_fila (f, nf );
}
int main ()
{
  cout << " Introduzca numero de filas : ";
  int n_filas ;
  cin >> n_filas ;
  esc_triangulo ( n_filas );
  system("pause");
  return 0;
}
```

C++

Escribe un programa que calcule el valor de S para un número real X $(0 \leq X \leq 1)$ dado por teclado, utilizando la serie de Taylor:

$$S = 1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \cdots$$

```
#include <iostream>
using namespace std ;
const int LIMITE_FACTORIAL_UNSIGNED = 14;
const double LIMITE = 1e-4;
double potencia ( double base , int exp )
{
  double res = 1;
  for (int i = 0; i < exp; ++i)
    res *= base ;
  return res ;
}
int factorial ( int n)
{
  int res = 1;
  for (int i = 2; i <= n; ++i)
    res *= i;
  return res ;
}
double termino ( double x, int i)
{
  return potencia (x, i) / double ( factorial (i ));
}
```

```
double serie ( double x)
{
  int i = 0;
  double res = 1;
  double term ;
  do {
     ++i;
     term = termino (x, i);
     res += term ;
   } while ( term >= LIMITE );
  return res ;
}
int main ()
{
  cout << " Introduzca el valor de X [0..1]: ";
  double x;
  cin >> x;
  if (! (x >= 0 && x <= 1))
    cout << " Error . Valor de X fuera de rango " << endl ;
  else cout << " Serie : " << serie (x) << endl ;
  system("pause");
  return 0;
}
```

Diseña un programa que encuentre el primer número perfecto mayor que 28. Un número es perfecto si coincide con la suma de sus divisores (salvo él mismo).

Por ejemplo, 28 es perfecto ya que $28 = 1 + 2 + 4 + 7 + 14$


```
#include <iostream>
using namespace std ;
int PRIMER_NUMERO = 29;
int suma_divisores (int n)
{
  int suma = 0;
  for (int i = 1; i <= n / 2; ++i)
  if (n % i == 0)
    suma += i;
return suma ;
}
inline bool es_perfecto (int n)
{
  return n == suma_divisores (n);
}
```

```
int primer_perfecto ()
{
  int i = PRIMER_NUMERO ;
  while (! es_perfecto (i))
    ++i;
return i;
}
int main ()
{
  cout << " Primer perfecto mayor que " <<
                          PRIMER_NUMERO << ": " ;
  cout<< primer_perfecto () << endl ;
  system("pause");
  return 0;
}
```

C++

Errores frecuentes

- Transmitir tipos incorrectos de datos
- Declarar localmente la misma variable dentro de la función que llama y la que hace la llamada
- Una variable local tiene el mismo nombre que una variable global
- Omitir el prototipo de la función llamada antes o dentro de la función que llama
- Terminar la línea de encabezado de una función con ;
- Olvidar incluir en la línea de encabezado el tipo de datos de los parámetros de la función

C++

Recursión

- ***Recursión,*** *repetición por autorreferencia, cuando un subprograma se llama a sí mismo; un proceso que se repite eternamente a menos que una estructura de control lo termine.*

Una recursión correcta debe satisfacer los siguientes requisitos:

- Todo subprograma recursivo debe contener una estructura de control que evite continuar con la recursión cuando se llegue al estado final.
- Se debe llegar necesariamente al estado final
- Cuando se llegue al estado final el subprograma debe haber completado el cálculo correcto

C++

Recursión

- Una función se dice definida recursivamente si su definición consta de dos partes:
  - Un caso base, en el cual se especifica el valor de la función para uno o mas valores
  - Un paso inductivo o recursivo, en el cual el valor de la función para el (los) valor(es) del parámetro se define en términos de los valores de la función definidos previamente y los valores de los parámetros.

- La condición para que el proceso recursivo termine, es decir no genere un nuevo cálculo, se denominará **caso base**.
- La recursión es una forma de descomponer un cálculo en otro más sencillo de la misma especie y continuar esta descomposición hasta llegar al caso trivial, caso base.
- Cuando **no utilizar** la **recursión:**
  - cuando dificulte la compresión del algoritmo
  - cuando produzca demandas excesivas de memoria o tarde mucho en ejecutarse

C++

Recursión: ¿Cuál será la salida?


```
#include<iostream>

using namespace std;

void p(int n)
{
    cout << n << endl;
    p(n+1);
}

int main()
{
    p(1);
    system("pause");
    return 0;
}
```

C++

Recursión: ¿Cuál será la salida?

La función p es una función recursiva mal definida:

no tiene caso base y el caso recursivo se aplica incondicionalmente.
Se trata de un caso de recursión infinita.

si la variable "local" esta definida, el desbordamiento de pila se produce con menos invocaciones de p. Esto se debe a que el registro de activación ocupa mucha mas memoria. double local[10000];

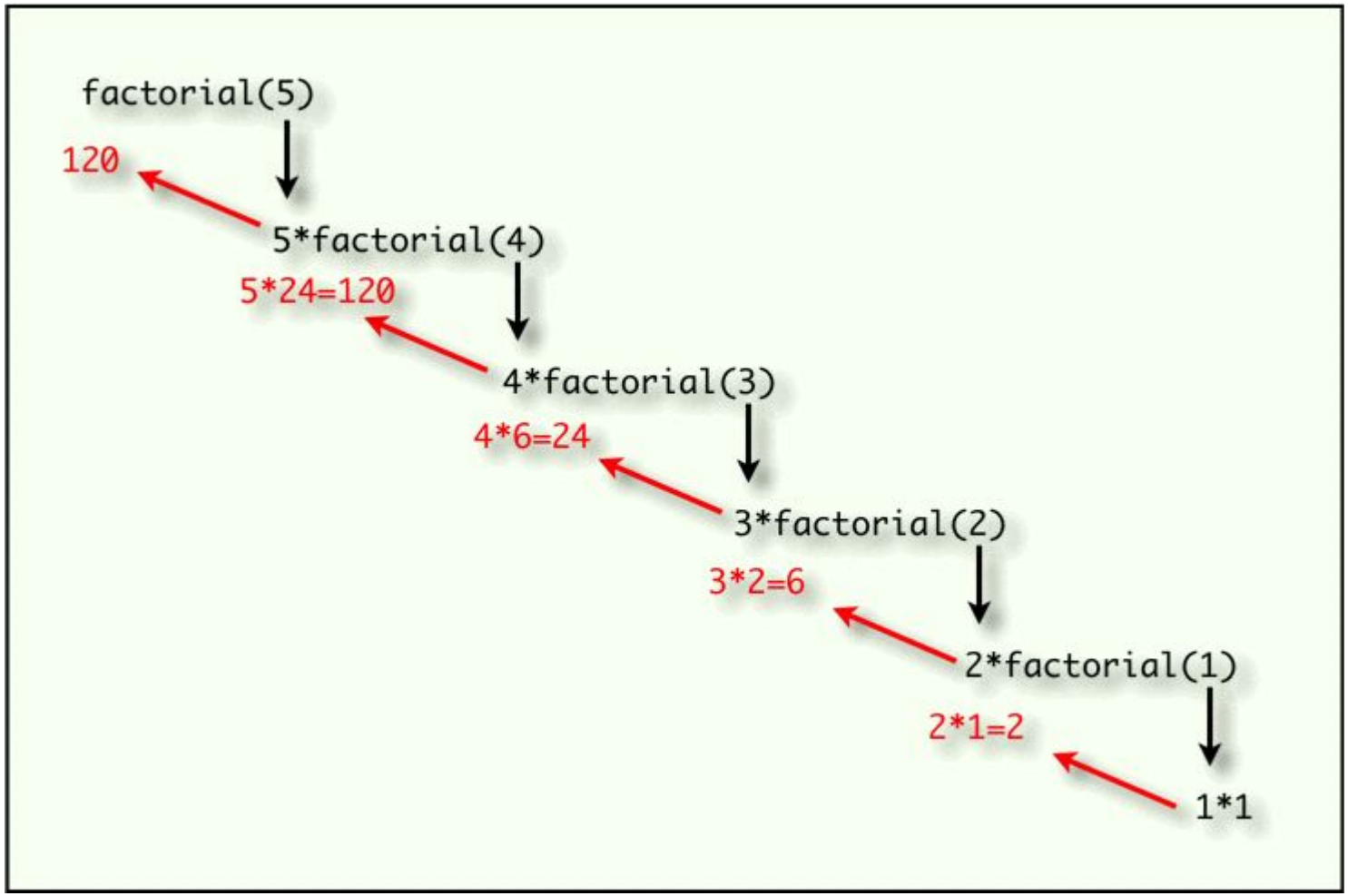
C++

Factorial recursivo


```
double factorial(double a)
{
   if (a<=1) return 1.0;
   else return (a *factorial(a-1.0));
}
int main ()
{ double n;
  cout << "Numero entero:";
  cin>> n;
  cout<<"El factorial de "<<n<<" es: "<< factorial(n);
  return 0;
}
```

C++

Traza para el factorial recursivo



C++

Traza para el factorial recursivo


```
#include <iostream>
using namespace std;
void EscribeBlancos(int n) {
    if (n>0)
    {
     cout << ' ';
     EscribeBlancos(n-1);
    }
}
int traza_factorial(int n, int margen) {
    const int sangrado= 4; // aumento de margen izquierdo
    int result;
    EscribeBlancos(margen);
    cout << "entrando en factorial(" << n << ")\n";
    if (n==0)
       result= 1;
    else
       result= n*traza_factorial(n-1, margen+sangrado);
    EscribeBlancos(margen);
    cout << "saliendo de factorial(" << n << ") con valor " <<  result << endl;
    return result;
}
```

```
int main() {
    int i;
    cout << "numero (negativo para acabar): ";
    cin >> i;
    while (i >= 0)
    {
     cout << traza_factorial(i,0) << endl;
     cout << "numero (negativo para acabar): ";
     cin >> i;
    }
    system("pause");
    return 0;
}
```

C++

Recursión: Escribir blancos


```
#include <iostream>
using namespace std;
// El caso base no hace nada. Esto permite escribir el procedimiento  solo con un if (sin else).
//Esto solo puede hacerse con  procedimientos, las funciones siempre tienen al menos un caso
// base explicito.
void EscribeBlancos(int n)
{
    if (n>0)   {
     cout << ' ';  // cout << '*' // para depurar
     EscribeBlancos(n-1);
    }
}

int main() {
    int i;
    cout << "numero de blancos (negativo para acabar): ";
    cin >> i;
    while(i>=0)    {
      EscribeBlancos(i);
      cout << '$' << endl;
      cout  << "numero de blancos (negativo para acabar): ";
      cin >> i;
    }
    system("pause");
    return 0;
}
```

C++

Recursión: Invertir Cadena Caracteres


```
#include <iostream>
using namespace std;
void invierte()
{
    char ch;

    ch= getchar();

    if (ch != '\n')
    {
        invierte();
    }

    cout << ch;
}
int main()
{
    cout << "teclea una cadena: ";
    invierte();
    cout << endl;
    system("pause");
    return 0;
}
```

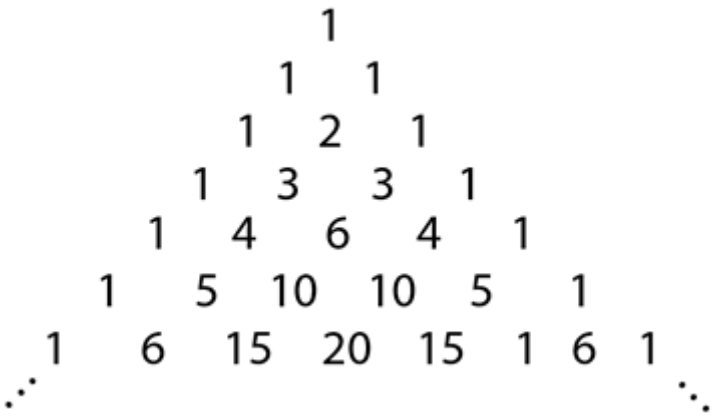
C++

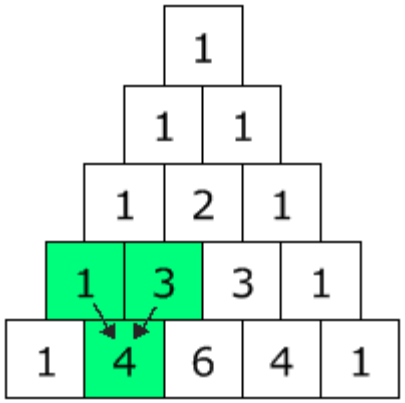
Potencia con exponente positivo o negativo


```
double potencia(double x, int n){
  if(n<0) return 1/potencia(x,-n);
  double pot;
  pot= 1;
  for (int i=1;i<=n;i++)
      pot=pot*x;
  return pot;
}
```

C++

Triángulo de Pascal





El triángulo de Pascal es un triángulo de números enteros, infinito y simétrico Se empieza con un 1 en la primera fila, y en las filas siguientes se van colocando números de forma que cada uno de ellos sea la suma de los dos números que tiene encima. Se supone que los lugares fuera del triángulo contienen ceros, de forma que los bordes del triángulo están formados por unos.

P(i,1)=1
P(i,i)=1
P(i,j)=P(i-1,j-1) + P(i-1,j) para 1<j<i

C++

Triángulo de Pascal


```
(1)     int v[51][51];
(2)     int P(int i, int j){
(3)         if(j==1 || i==j){
(4)             return 1;
(5)         }if(v[i][j]!=0){
(6)             return v[i][j];
(7)         }v[i][j]=P(i-1, j-1)+P(i-1, j);
(8)         return v[i][j];
(9)     }
```

C++