Envio de parámetros en las funciones en C++:

Implementar una función que calcule el factorial de forma recursiva.

Parámetros por valor (se manda copia de valores a los parámetros):

// Función que intercambia dos valores pero no definitivamente:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

void interc(int i, int j);
int main()
{       int m, n;
        m=2; n=5;
        printf("MAIN1:   m=%d,  n= %d\n",  m, n);
        interc(m,n);
        printf("MAIN2:   m=%d,  n= %d\n",  m, n);
        return 0;

}

void interc( int i, int j)
{
        int t;
        printf("FUNC1:   i=%d,  j= %d\n",  i, j);
        t=i;
        i=j;
        j=t;
        printf("FUNC2:   i=%d,  j= %d\n",  i, j);
}
```

Paso de Argumentos POR REFERENCIA:

- Esto es otra forma de conseguir que una función devuelva valores (aparte de usando `return`).
- Además, de esta forma una función puede devolver tantos valores como se deseen (cada uno en un argumento distinto).
- **En lenguaje C, TODOS los pasos de argumentos son por VALOR.**
- Se llama **Paso de Argumentos POR REFERENCIA** a una técnica que permite a una función modificar variables utilizadas como argumentos actuales.

- **En C++, la TÉCNICA de paso de argumentos por REFERENCIA se simplifica:**
  - **1. En la Definición de la Función:**
    - Sencillamente hay que marcar qué argumentos tendrán un paso de argumentos por referencia.
    - Esto se hace **añadiendo al tipo del argumento el signo &.**
      - Ej.:
```
void Func1(int& x){
    x = x + 3;
}
```
(el arg. **x** pasa por referencia)
  - **2. En la Llamada a la Función:**
    - El argumento actual en un paso de arg. por referencia se usa normalmente (como en un paso por valor).
      - Ej.:
```
a=1;
Func1(a);
```
(la variable a pasa por referencia, pero no se indica nada)
      - Tras esa llamada, la variable **a** valdrá **4**.
    - En un paso de arg. por referencia el argumento actual debe ser obligatoriamente una variable.

Paso de args. POR REFERENCIA: Otro Ejemplo en C++

En C++

- **Ejemplo:** ¿Qué **salida** produce el siguiente programa?

```
#include<stdio.h>

/* Intercambio de valores entre dos variables */
void swap (int& x, int &y){
  int aux=x;
  x = y;
  y = aux;
}

int main(){
  int x=1, y=2, z=3;
  swap(x,y);
  swap(y,z);
  printf("\n- Valores: %i, %i y %i.", x, y, z);
  getchar();
  return 0;
}
```

  - **Resultado:**

```
- Valores: 2, 3 y 1.
```

Ejercicio: cambiar el código anterior para que la función intercambia de valores definitivamente.

// Funcion que intercambia dos valores y definitivamente!:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void interc1(int &i, int &j);
int main()
{   int m, n;
          m=2; n=5;
printf("MAIN1:   m=%d,  n= %d\n",  m, n);
interc1(m,n);
printf("MAIN2:   m=%d,  n= %d\n",  m, n);
getch();
return 0;

}

void interc1( int &i, int &j)
{ int t;
printf("FUNC1:   i=%d,  j= %d\n",  i, j);
t=i;
i=j;
j=t;
printf("FUNC2:   i=%d,  j= %d\n",  i, j);
}
```

&k // variable k llevará la dirección del paramento enviado

Parametos por defecto: (directamente se asigna valor a argumento en llamado a una función o en el anuncio de una función

```
double bisec (double a, double b, int &raiz, double epsX=1.0E-6, double epsF=1.0E-8);
int main();
{ ….}
int bisec(double a, double b, double &raiz, double epsX=1.0E-6, double epsF=1.0E-8)
{ …..}
```

Llamada de la function:

```
bisec(a, b, ind, 0.0005, 1.0e-5);
bisec(a, b, ind, 0.0005);
```

bisec(a, b, ind);

Variables locales y globales:

Variables LOCALES: Son aquellas que se declaran **dentro** de una función. Las variables de sus **argumentos formales** son **locales**.
– Sólo tienen sentido y sólo pueden usarse **dentro** de esa función.
– Su **nombre** es totalmente **independiente** de las variables locales de otras funciones, incluyendo la función **main()**.

Variables GLOBALES: Mejor NO usarlas

- Son variables que se declaran **fuera** de todas las funciones: Pueden ser utilizadas por todas las funciones que haya después de su declaración.
  - Normalmente, estas variables se declaran antes que las funciones, por lo que su **ámbito o visibilidad** es global: Pueden usarse en cualquier función.
  - **Su uso está desaconsejado, especialmente en programadores noveles**, porque complica la comprensión de los programas y pueden dar lugar a *efectos laterales* erróneos que suelen ser muy difícil de localizar.
  - Si hay **ambigüedad** entre **variables locales y globales** se usan las **locales**.

Ejercicio:

- **Ejemplo:** ¿Qué **salida** produce el siguiente programa para distintas **entradas**?

```
#include<stdio.h>
float func (float x, float y){
  x = x + 1;
  y = y + 2;
  return x - y;
}
int main(){
  float x,y,z;
  printf("\n- Introduzca un número: ");
  scanf("%f",&x);
  y = x + x;
  z = func(y,x);
  x = func(y,z);
  func(x,y);
  printf("\n- Valores: %.1f, %.1f y %.1f.", x, y, z);
  return 0;
}
```

Ejemplo de Ejecución:

```
- Introduzca un número: 3

- Valores: 3.0, 6.0 y 2.0.
```

Ejercicios: en el siguiente programa cual salida esperamos:

```c
#include <stdio.h>
int a; /* Esta es u

int func1 (int a){
  return a*a;
}

int func2 (int x){
  int a=1;
  return a+x;
}

int func3 (int x){
  return a+x;
}

 int main(){
   int x=3;
   a = x - 1;
   x=func1(a);
   a=func2(x);
   x=func3(a);
   printf("\n-Valores: %i y %i.",x,a);
   return 0;
 }
```

Salida:

```
- Valores: 10 y 5.
```

Ejercicio:
Considere el siguiente programa:

```c
#include <stdio.h>
int i; /* Precaución: variable global */
/************************/
void F1() {
  i=i+2;
}
/************************/
void F2() {
  int i;
  i=0;
  F1();
  printf("%d\n",i);
}
/************************/
void F3() {
  int i;
  i=4;
}
/************************/
void F4() {
  i=i+3;
  printf("%d\n",i);
  F3();
  printf("%d\n",i);
}

/* FUNCIÓN PRINCIPAL */
int main() {
  i=12;
  F2();
  printf("%d ",i);
  F4();
  printf("%d",i);
  return 0;
}
```

a) Determina el **ámbito** de cada una de las tres variables `i` declaradas en el programa.

b) Muestra lo que ocurre cuando se ejecuta el programa.

c) ¿Qué ocurre si se declara la variable `i` del módulo principal justo antes de la función `main()`?