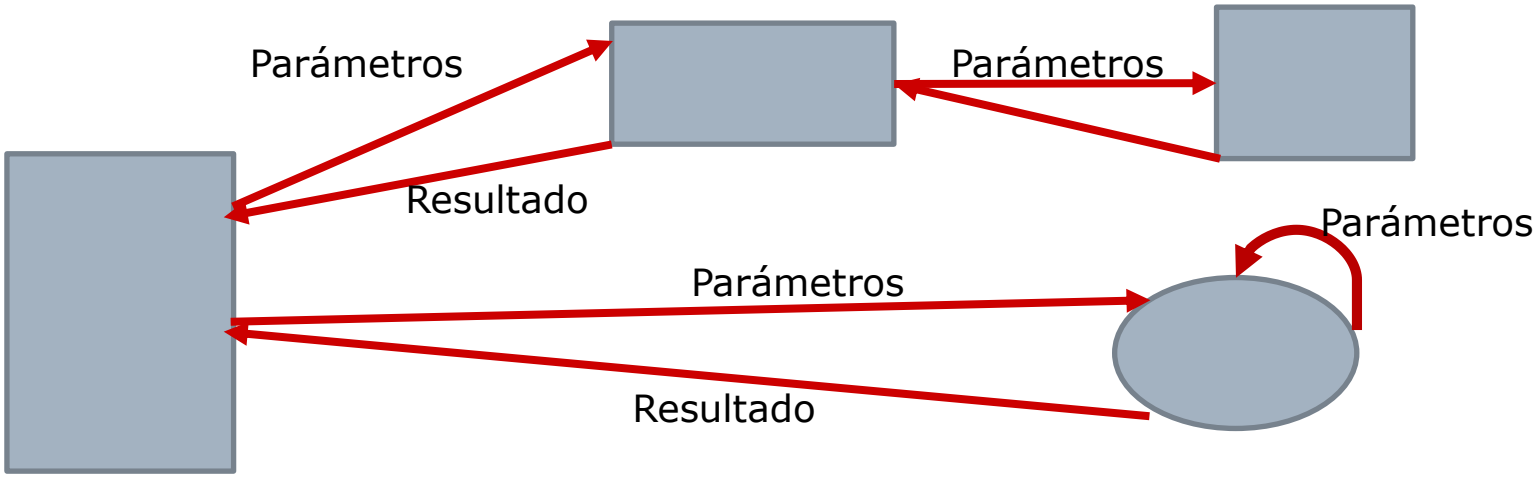
Introducción a la Programación

Funciones en C

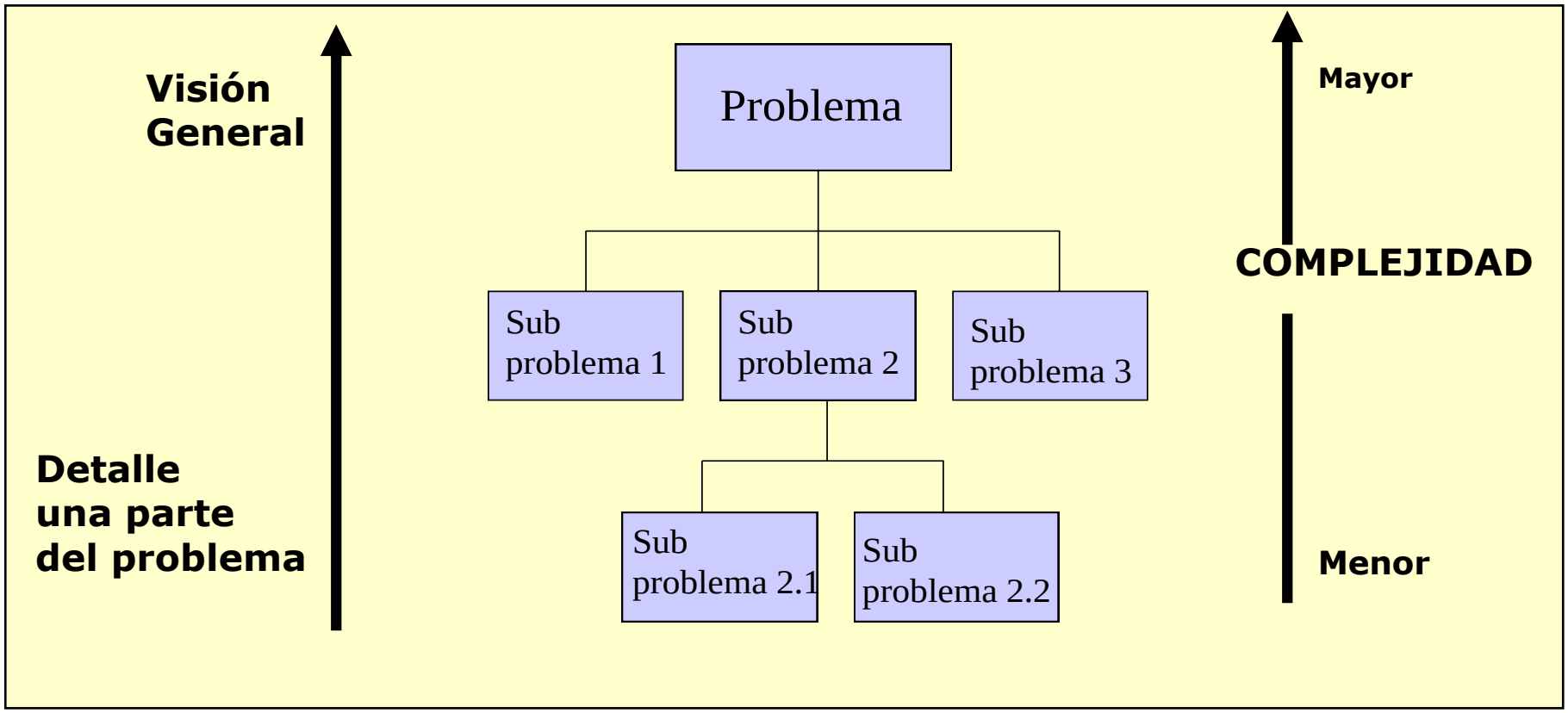


Uso de Funciones en C

- La mayoría de los programas descritos hasta ahora pueden ser resueltos directamente con un solo programa.
- Sin embargo existen problemas más complejos y para solucionarlos se debe usar la estrategia "*Dividir para Conquistar*" o sea subdividir el problema en subproblemas más pequeños y menos complejos.

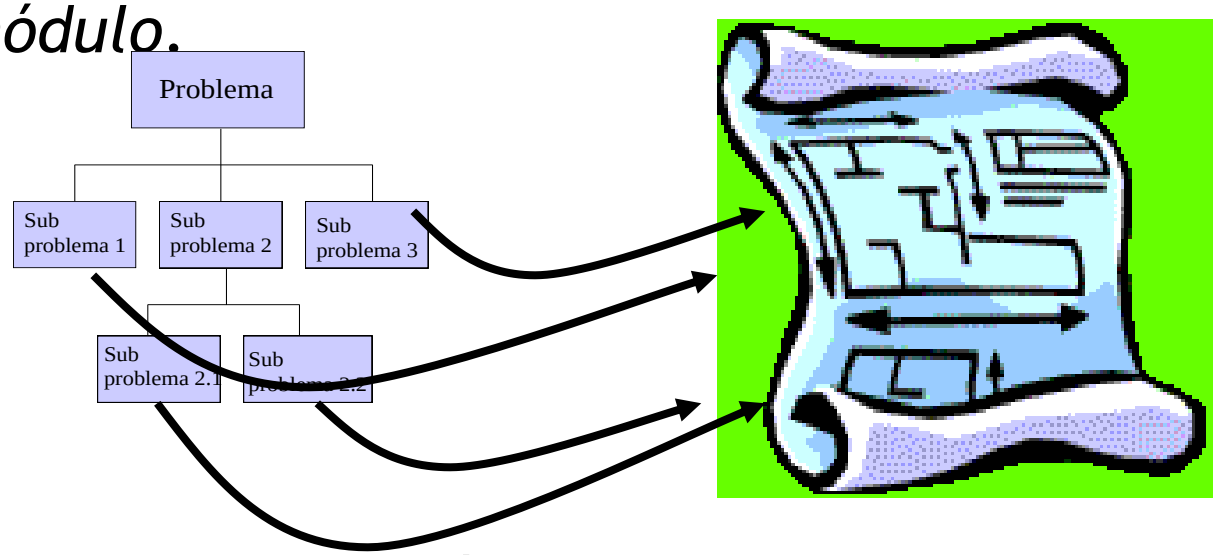
Dividir para Conquistar

- Aplicando esta estrategia se divide el problema en subproblemas, obteniéndose un **Diagrama de Descomposición del problema.**



Programación Modular

- La idea de que el problema haya sido dividido en subproblemas es para generar un subprograma para cada *módulo*.



- El programa de solución al subproblema, no debe de perder de vista el problema general. A cada parte del problema se le llama **Módulo**

Programación Modular

- Cada uno de los módulos son independientes entre si, pero en conjunto resuelven el problema general.
- Se puede entonces "*construir*" cada módulo independientemente y pueden ser además probados de la misma forma. Se facilita también la detección y corrección de errores.
- Cada módulo pasa a ser una ***función***.

¿ Qué es una Función?

- Matemáticamente es una operación que toma uno o más valores de *entrada* (llamados argumentos) y produce **un** resultado (valor de la función).

- Ejemplo:

$$f(x) = \frac{x}{(1+x^2)}$$

$$y = \textit{factorial}\ (\ n\ )$$

Funciones y Procedimientos

- Un módulo puede implementarse en el algoritmo (y luego en el programa) mediante la forma de una función o de un procedimiento:
  - **Función** : es un módulo que entrega **un** valor como resultado. Puede recibir varios (o ningún) datos para trabajar y obtener esos resultados.

Funciones en C

- Un programa C está formado por un conjunto de funciones. (Al menos contiene la función main) .
- Una función se define con el *nombre* de la función anteponiendo el *tipo de valor que retorna* y por una lista de *argumentos* encerrados entre paréntesis. El cuerpo de la función está formado por un conjunto de declaraciones y de sentencias delimitadas por las llaves.

Forma general:

```
tipo_resultado    nombre_función(tipo param1, tipo param2, ...)
{
        ....
        ....
        return (parámetro/expresión/estructura);  //Opcional
}
```

Funciones en C

- El valor que debe devolver una función se indica con la palabra return. El valor devuelto debe ser del mismo tipo de dato que el que se ha definido la función.

Ejemplo:

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

La función llamada *suma* recibe como parámetros los valores enteros **a** y **b**. Por otra parte la función retorna un valor de tipo entero.

Funciones en C

- Cuando el programador crea una función propia, para poder usarla esta debe ser primero declarada. A esto se le conoce como **prototipo de la función**.

- El *prototipo de la función* consta de su nombre y de información importante como el tipo del valor que devuelve y de los parámetros que esta recibirá. Los *prototipos de función* deben ir antes de la función main().

Ejemplo:

```
void mensaje (void)   /* función que no retorna valor ni recibe parámetros */
int factorial(int n)      /* retorna un valor entero y recibe un parámetro  */
```

Funciones que no devuelven valores
(procedimiento)

- En algunos casos, se necesita que una función no retorne un valor en forma directa. En este caso la función retorna un valor *void*.

- El tipo *void* indica que la función no retorna valor. También, *void* puede ser usado para indicar que la función tampoco recibe parámetros.

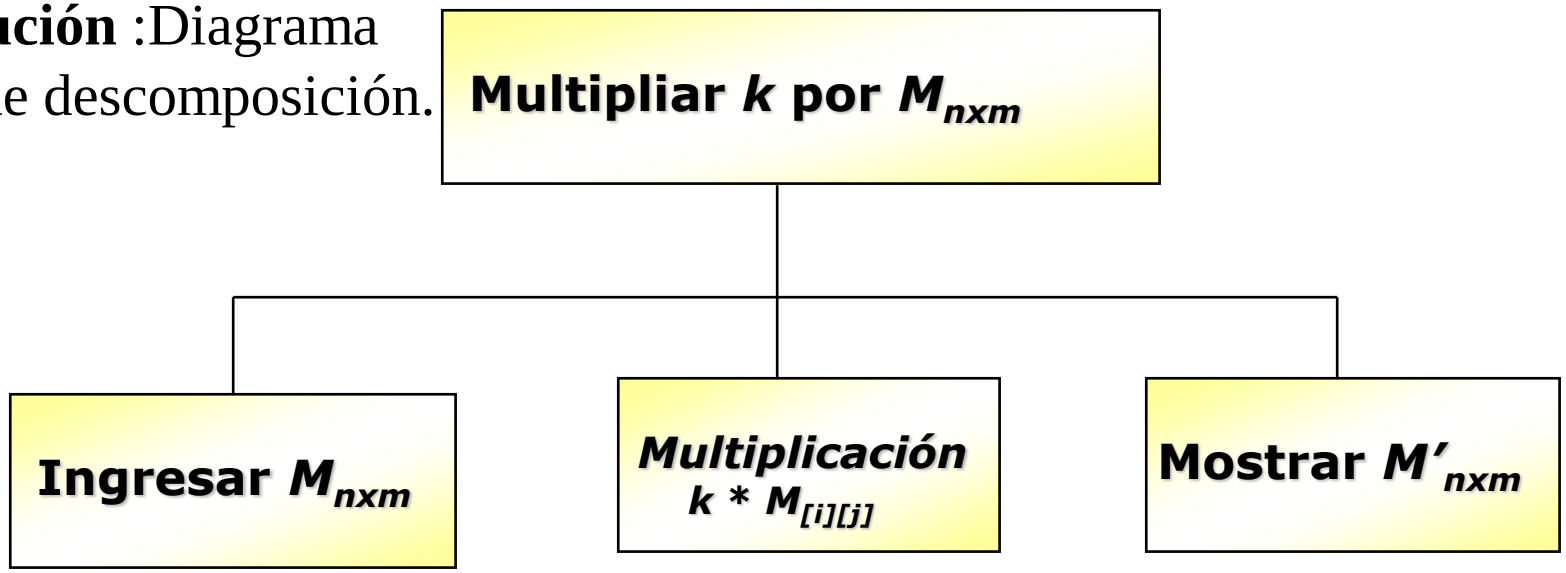
Ejemplo :

```
void mensaje( void )
{
    printf(“ Esta función solo imprime este mensaje “);
}
```

Ejemplo

Problema : Multiplicar un número *k* por una matriz de orden *n x m* . Tanto k, como el orden y los elementos de la matriz deben ser ingresados por el usuario.

Solución :Diagrama de descomposición.



Solución :

```
/* Programa que Multiplica Mmxn por k  */
#include <stdio.h>
int  k, i , j, m[10][10]
void ingresar_matriz(void)
void  multiplicación(void)
void mostrar_matriz(void )
main( )
{
printf(“ Ingrese escalar a multiplicar :”);
scanf(“%d”,&k);
ingresar_matriz( );
multiplicación( );
mostrar_matriz( );
}
```

```
void ingresar_matriz(void)
 {
  ...
 }
```

```
void  multiplicación(void)
  {
   float c;
    ...
   }
```

```
void mostrar_matriz(void )
 {
  ...
 }
```

Ámbito de las variables

- Las variables de un programa se clasifican en globales y locales, de acuerdo a su ámbito de existencia:
- Las **variables locales** son aquellas que se utilizan en la definición de un subprograma (función o procedimiento). Sólo tienen vigencia en el subprograma en que están definidas y son desconocidas fuera de él.
- Las **variables globales** tienen vigencia sobre todo el programa, tanto en el programa principal como en los subprogramas.

Comunicación entre funciones en C

- Cuando la función **main( )**, u otra función, llama a otra función esta comienza a ejecutarse hasta llegar a su última instrucción(return) y devuelve el control a la función que realizó la llamada.

- Es usual que los módulos llamadores tengan alguna información que comunicar al módulo llamado.

Ejemplo : u = unidades(n)

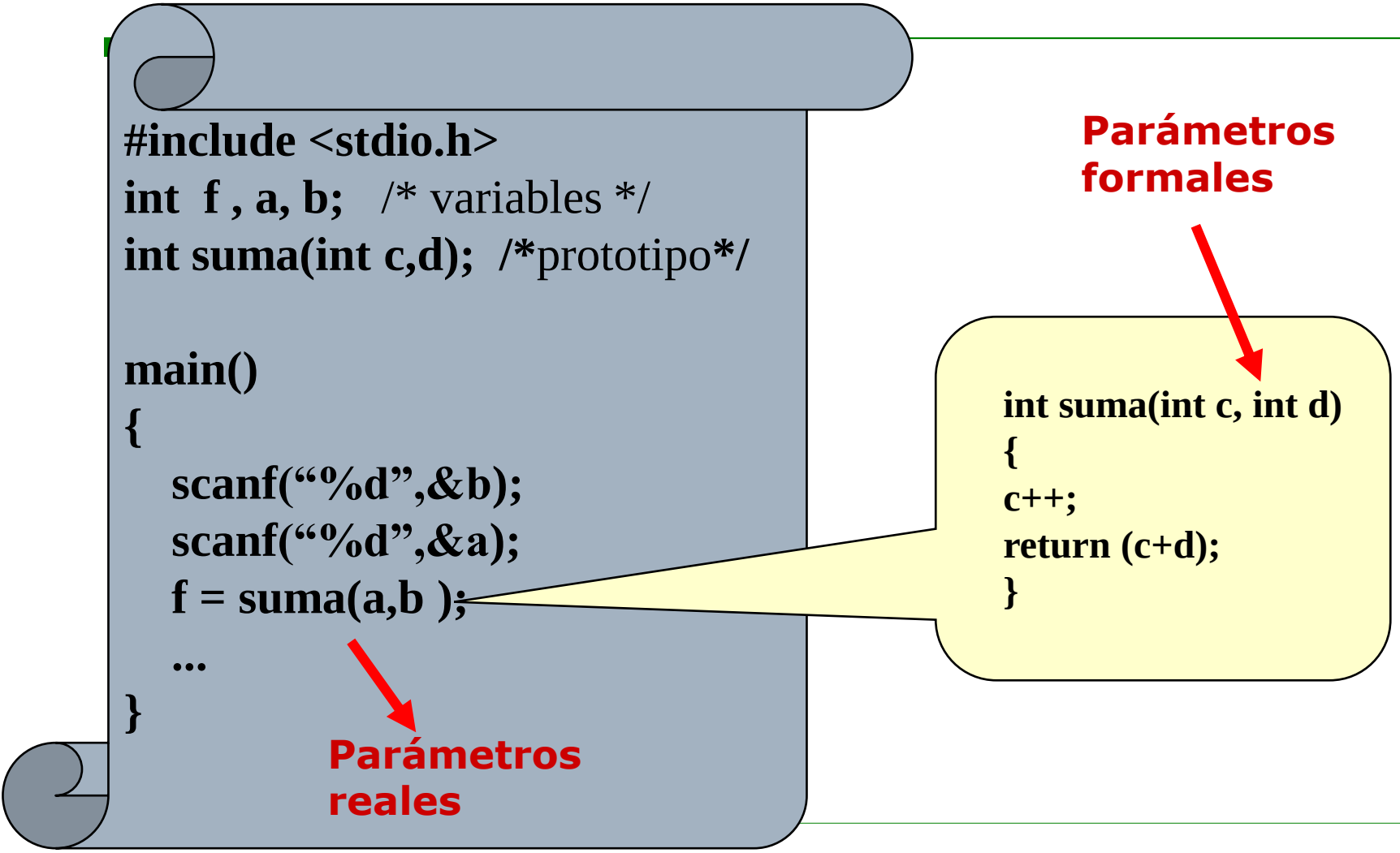
printf(" El resultado es %d ", sum);

Comunicación entre funciones en C

- Los datos que se comunican al módulo llamado se conocen como *parámetros*.

- Entonces la comunicación entre módulos se realiza a través de parámetros, al momento de realizar la llamada invocando el nombre del módulo más el o los parámetros reales (datos de llamada) si es que se requieren. Si se llamó a una función, esta puede devolver el resultado de su trabajo mediante RETURN( ).

Parámetros


#include <stdio.h>
int f , a, b; /* variables */
int suma(int c,d); /*prototipo*/
main()
{
scanf("%d",&b);
scanf("%d",&a);
f = suma(a,b);
...
}
Parámetros reales
Parámetros formales
int suma(int c, int d)
{
c++;
return (c+d);
}

Paso de parámetros

- Cuando un programa llama a un subprograma la información importante se comunica a través de la ***lista de parámetros*** y se establece una correspondencia automática entre los parámetros *reales* y *formales*. Los parámetros **reales** son "sustituidos" por los parámetros **formales** y estos son los utilizados.
- Existen métodos diferentes para el paso de parámetros a funciones. Es preciso conocer la disponibilidad y la forma en que los lenguajes de programación apoyan este concepto.
- Un mismo programa puede producir distintos resultados bajo diferentes formas de paso de parámetros.

Paso de parámetros por valor

- En este caso los parámetros se tratan como variables locales y los valores de los parámetros reales se *copian* en los correspondientes parámetros formales.

- Los cambios que se produzcan en los parámetros por efecto del subprograma (función) **no afectan** a los argumentos originales.

Paso de parámetros por referencia

- El módulo que llama envía la dirección de memoria del parámetro real. Entonces una variable pasada como parámetro por referencia **puede ser modificada** directamente por el subprograma.

- En lenguaje C, cuando se realiza un paso de parámetro por referencia el argumento se precede del símbolo &.

Ejemplo : Suma(a, b, &c)

Scanf("%d", &n)

Comparación de métodos de paso de parámetros

```
...
int a, b, c;
a=5; b=6; c=20;
sumar(a,b,c);
printf(" el valor de c es %d", c);
...  }


void sumar(int x, int y, int z)
{
      z = x  + y;
}
```

Paso de parám. **por valor**

```
...
int a, b, c;
a=5; b=6; c=20;
sumar(a,b,&c);
printf(" el valor de c es %d", c);
...  }


void sumar(int x, int y, int *z)
{
    *z = x + y;
}
```

Paso de parám. **por referencia**

Ejemplo 1 Funciones

- Construir un programa en C que permita evaluar la siguiente función:

$$f(x) = \begin{cases} (x^2 + 5*x) \ / \ (x-10)! & x > 10 \\ (x^2 + 2*x) \ / \ (x-5)! & 5 < x < 10 \\ (x^3) \ / \ (x)! & 0 <= x <= 5 \\ (-x)! & x < 0 \end{cases}$$

Ejemplo 1: Sin uso de funciones

```
#include <stdio.h>
int x,i;
float f,factx;

main()
{
     printf("\nEvaluación de f(x)“ );
     printf("\n\nIngrese valor a evaluar");
     scanf("%d",&x);
     factx=1;
     if (x  > 10)
      {
         for (i=1;i <= x-10;i++) factx=i*factx;
          f= ((x*x)+5*x)/factx;
      }
     else if ((x > 5)&& (x <= 10))
      {
         for (i=1;i <= x-5;i++) factx=i*factx;
          f= ((x*x)+2*x)/factx;
      }
else if (( x >= 0 ) && ( x <= 5 ))
     {
          if (x == 0 ) f=x*x*x;
          else
          {
             for (i=1;i<=x;i++) factx=i*factx;
             f= (x*x*x)/factx;
          }
     }
     else
     {
          for (i=1;i<= -x ;i++) factx=i*factx;
          f= factx;
     }
     printf ("\nEl resultado de f(%d) =%f",x,f);
}
```

```c
#include <stdio.h>
int x;
float f;
float factx (int p);
main()
{
    printf("\nEvaluación de f(x)");
    printf("\n\nIngrese valor a evaluar >>>>");
    scanf("%d",&x);
    if (x  > 10)
     {
        f= ((x*x)+5*x)/factx(x-10);
     }
    else if ((x > 5)&& (x <= 10))
     {
        f= ((x*x)+2*x)/factx(x-5);
     }
     else if (( x >= 0 ) && ( x <= 5 ))
     {
        if (x == 0 ) f=x*x*x;
        else
        {
           f= (x*x*x)/factx(x);
        }
     }
else
     {
        f= factx(-x);
     }
     printf ("\n El resultado de f(%d) =%f",x,f);
     getchar();
     getchar();
}

float factx(int p)

{
  int i,fx;
   fx=1;
   if (p==0) return(fx);
   else
   {
      for (i=1; i<=p;i++) fx=i*fx;
   }
   return(fx);
}
```

Ejemplo: Función recursiva

```
#include <stdio.h>

int x;
float f;
float factx (int p);

main()
{
    printf("\nEvaluación de f(x)");
    printf("\n\nIngrese valor a evaluar >>>>");
    scanf("%d",&x);
    if (x  > 10)
     {
        f= ((x*x)+5*x)/factx(x-10);
     }
    else if ((x > 5)&& (x <= 10))
     {
        f= ((x*x)+2*x)/factx(x-5);
     }
else if (( x >= 0 ) && ( x <= 5 ))
      {
         if (x == 0 ) f=x*x*x;
         else
         {
            f= (x*x*x)/factx(x);
         }
      }
      else
      {
         f= factx(-x);
      }
      printf ("\n El resultado de f(%d) =%f",x,f);
      getchar();
      getchar();
}

float factx(int p)
{
   if (p==0) return(1);
   else return(p*factx(p-1));
}
```

Ejercicios

- Construir un programa en C que para resolver la siguiente sumatoria:

$$\sum_{x=a}^{b} f(x)$$

Con a y b mayores que 0 y f(x)=

$\mathbf{(x)^{(b-x)!}}$	Si 0<x<5
$\mathbf{((x+b)!)^{a}}$	Si 5≤x ≤10
x!	Si x >10

- Construir un programa en C que permita leer dos cadenas de largo máximo 15 e indique si las cadenas ingresadas son idénticas. Su programa debe contener al menos una función que reciba como parámetro dos cadenas y retorne un 0 si son idénticas y un 1 en caso contrario.

- Construir un programa en C que permita contar cuantas veces se repite una palabra dentro de una frase. La palabra y la frase deben ser ingresadas por teclado.

Funciones en C

Funciones recursivas