

Funciones en C

Sintaxis

Una función es un mini programa dentro de un programa.

Las funciones contienen varias sentencias bajo un solo nombre.

Proporcionan un medio de dividir un proyecto grande en módulos pequeños.

Las funciones existen de modo autónomo, cada una tiene su ámbito.

La división del código en funciones hace que las mismas se puedan reutilizar en su programa y en otros programas.

Para reutilizar una función dentro de un programa, solo se necesita llamar a la función.

Si se agrupan funciones en bibliotecas otros programas pueden utilizar estas funciones

Las funciones en C no pueden anidarse es decir una función no se puede declarar dentro de otra función.

Dicho de otra forma:

Una función proporciona una forma conveniente de encapsular algunos cálculos, que se pueden emplear después sin preocuparse de su implementación.

Una función es un subprograma que se llama dentro de un programa principal. Las funciones son independientes del programa que la llama, por eso una función puede ser utilizada varias veces dentro de un programa o ser utilizada dentro de otros programas.

Una definición de función tiene la siguiente forma:

TIPO DE RETORNO NOMBRE (TIPO 1 ARG1, TIPO 2 ARG2.....)

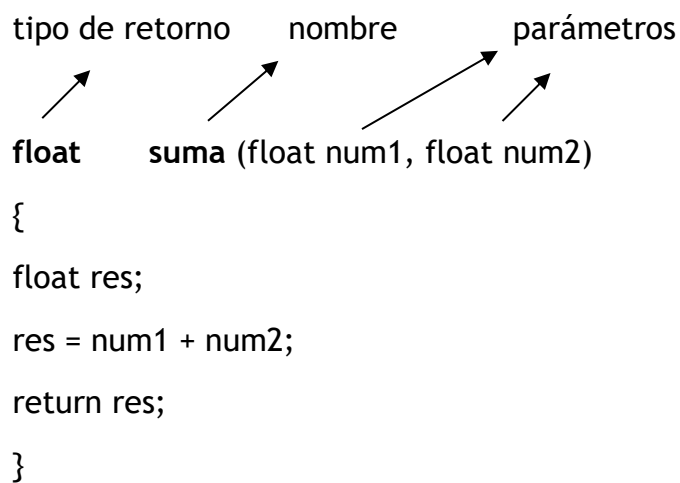


Argumentos formales

ya que representan los nombres de los elementos que se transfieren a la función desde la parte del programa que llama. También se los conoce como parámetros o parámetros formales.

Una estructura de función es:

```
Tipo de retorno    nombre (lista de parámetros)  
{  
  cuerpo de la función  
  return expresión;  
}
```



```
tipo de retorno  nombre  parámetros  
    ↗           ↗       ↗  
float    suma (float num1, float num2)  
{  
  float res;  
  res = num1 + num2;  
  return res;  
}
```

Tipo de retorno o resultado: Es el tipo de dato que vuelve la función y aparece antes del nombre de la función

Lista de parámetros: lista de parámetros tipificados (con tipo) tipo1 parametro1, tipo2 parametro2.

Cuerpo de la función: se encierra entre llaves {...} no hay punto y coma después de la llave de cierre

Declaración local: las constantes, tipo de datos y variables declaradas dentro de las funciones son locales a la misma y se perderán fuera de ella

Valor devuelto por la función: mediante la palabra return se regresa el valor de la función.

Una llamada a una función produce la ejecución de las sentencias del cuerpo de la función y un retorno a la unidad de programa llamadora después que la ejecución de la función se ha terminado, normalmente cuando se encuentra una sentencia return.

Nombre de una función: El nombre de una función comienza con una letra y puede contener hasta 32 letras.

Tipo de dato de retorno: Si la función no devuelve un valor “int”, se debe especificar el tipo de dato devuelto por la función.

Cuando devuelve un valor “int”, se puede omitir ya que por defecto “C” asume que todas las funciones son enteras.

Si la función no devuelve un resultado, se puede utilizar el tipo “void”

Resultados de una función

Una función puede devolver un único valor. El resultado se muestra con una sentencia “return”.

Llamada a una función

Las funciones para poder ser ejecutadas, han de ser llamadas.

La función llamada que recibe el control del programa se ejecuta desde el principio y cuando termina (se llega a la sentencia “return” o a la “}” final).

Es decir:

En el momento del **desarrollo de la función** luego de la definición se colocan “{” para indicar el inicio y “}” para indicar el fin de la función.

El cuerpo de la función es el conjunto de instrucciones entre esas llaves y debe incluir una o más instrucciones .

La última instrucción del cuerpo es return; o

```
return expresión;
```

return para volver un valor al punto de llamada, el tipo de la expresión debe ser el mismo que se define como tipo de salida o retorno en la definición de la función.

Una función no necesita regresar un valor, un return sin expresión hace que el control regrese al programa sin pasar ningún valor al programa principal. En este caso, que no regresa nada, se coloca como tipo de retorno void.

Ejemplo 1:

Dados dos números, mostrar el mayor (los números son distintos)

```
void maximo ( int x, int y)
{
    if(x>y)
        printf ("El valor máximo es %d", x);
    else
        printf ("El valor máximo es %d", y);
    return;
}
```

Ejemplo 2:

Dado un número natural mostrar su factorial.

```

long fac ( int n)
{
    int i ;
    long prod = 1;
    if ( n > 1)
        for ( i = 2; i <= n; i++ )
            prod *= i;
    return prod ;
}

```

Ejemplo 3:

Calcular la suma de n números reales.

```

float suma (int n)
{
    int i;
    float sum = 0.0, x;
    printf("Introduce %d números reales", n);
    for ( i=0; i<n; i++)
    {
        scanf("%f", &x);
        sum += x;
    }
    return sum;
}

```

¿Cómo se utilizan las funciones en un programa?

Si se utilizan funciones dentro de un programa, éstas deben estar en 3 formas distintas:

la primera es el prototipo,

la segunda es la llamada de la función dentro del main(), y

la tercera es la definición y desarrollo.

La primera es el Prototipo de una función. Es la declaración de una función.

Los prototipos contienen la cabecera de una función pero terminan con punto y coma.

Se ubican normalmente al principio de un programa, antes de la definición del "main()".

Cuando una función se declara se da su nombre y la lista de sus parámetros.

Cuando se define significa que existe un lugar en un programa donde existe la función desarrollada.

Los nombres de los argumentos en los prototipos no tienen significado, son independientes.

TIPO DE RETORNO NOMBRE (TIPO 1 ARG1, TIPO 2 ARG2 ...);

Los nombres de los argumentos formales o locales no tienen porque coincidir con los de los reales.

El tipo de dato del argumento local debe ser igual que al del tipo de dato del argumento real que recibe desde el punto de llamada.

Ejemplo de prototipos:

```
#include <stdio.h>

long fac ( int n);

void maximo ( int x, int y);

int main()

{ ...

}
```

La segunda es la llamada de la función dentro del main(),

Cuando se llama a la función dentro del programa principal los argumentos se llaman argumentos reales, ya que definen la información real que se transfiere.

En la llamada los argumentos van sin su tipo, ya que ya fueron definidos al inicio del programa.

Recordemos que los argumentos reales son variables que se utilizan en el programa o valores constantes.

La llamada a una función puede ser de 2 formas distintas, según retornen o no un valor.

Si la función no regresa ningún valor en la llamada solo se nombra la función.

Si la función regresa un valor, en la llamada debe haber una asignación del valor que regresa a otra variable del mismo tipo..

Ejemplos de prototipos y sus llamadas:

```
#include <stdio.h>

void mostrar(int n, int res);

int main()
{
    int n,res;
    ...
    mostrar(n,res);
    ...
}
```

```

#include <stdio.h>

int suma(int a, int b);

int main()
{
    int n1,n2,c;
    ...
    c=suma (n1,n2);
}

```

En el prototipo puedo no colocar el nombre de las variables, pero si es obligatorio el tipo y respetar el orden:

```
int power (int, int);
```

Si no se nombran parámetros, se toman como int

```
int power ( );
```

La tercera es la definición y desarrollo.

Al terminar el int main() se hace el desarrollo de la función con su definición, su cuerpo y su return.

La declaración prototipo y la definición de la función deben coincidir en cantidad de parámetros y formatos y el orden en que se enumeran.

Los nombres de los parámetros no siempre coinciden, son optativos.

La función que se invoca no puede alterar directamente una variable de la función que hace la llamada; sólo puede modificar su copia temporal.

Es decir, si entra una variable del programa principal y la modifico en la función, al volver no llega modificada al programa principal si la función es void.

Ejemplo:

Escribir a 2 columnas las potencias de 2 y de 3 con exponentes 0 al 9.

```
#include <stdio.h>

int power (int m,int n); /* función prototipo, luego se
desarrolla.*/

int main( )
{
    int i,a,b;
    for (i = 0; i < 10; ++ i)
    {if (i==0)
        {
            a=1;
            b=1;
        }
        else
        {
            a= power (2,i); /* llamadas a la función.*/
            b=power (3,i);
        }
        printf ("%3d  %6d  %6d \n", i, a , b);
    }/*fin del programa principal*/

    int power (int base, int n) /*definición (no va ;)*/*
    {
        int i, p;
        p = 1;
        for (i = 1; i <= n, i++)
            p *= base;
        return p; /*variable local */
    }
```

Cada llamada pasa 2 argumentos a `power()`, y cada vez regresa un entero `p`.

En el programa principal, los argumentos que se pasan a `power()` son uno por valor ya que pasa una constante y uno por referencia ya que pasa una variable.

Paso por valor (o paso por copia) significa que cuando “C” compila las funciones y el código que llama a la función, la función recibe una copia de los valores de los parámetros.

En la definición o desarrollo de `power()` los nombres `base` y `n` son argumentos locales. Por el orden en que están, la constante se corresponde con `base` y la variable `i` con `n`

```
a = power (2, i);
```



```
int power (int base, int n)
```

Las variables `i` y `p` que se declaran en la función son variables locales, es decir variables que no se ven en el principal y que si no regresan pierden su valor al salir de la función.

En nuestro ejemplo `p` regresa su valor por estar declarada en el `return` y en el principal su valor se asigna a una variable del mismo tipo, y la `i` como no regresa no modifica el valor de la variable `i` del principal.

Sus ámbitos son distintos, una es global o general (la del programa principal) y la otra es local (de la función).