Funciones

Una estrategia para la resolución de problemas complejos es la división del problema en otros problemas más pequeños también denominados subproblemas.

Estos subproblemas se implementan a través de módulos o bloques que también conocemos como subprogramas. De modo que en general un programa se compone de un conjunto de subprogramas y un programa principal desde donde son llamados dichos subprogramas.

Esta técnica de dividir el problema en subproblemas se basa en el lema de divide y vencerás. Cada subproblema es deseable que sea independiente de los restantes. Decimos entonces que el problema se resuelve con el programa principal, al que denominamos conductor y los subprogramas que se denominan *funciones*.

C fue diseñado para hacer funciones eficientes y fáciles de usar. Los programas en C consisten generalmente de varias funciones pequeñas en vez de pocas grandes.

Funciones en C

Una función es un fragmento de código que realiza una tarea bien definida. Por ejemplo, la función printf imprime en la salida estandar los argumentos que le pasamos.

Dicho de otra manera, una función es una rutina o conjunto de sentencias o instrucciones que realizan una determinada tarea, es decir son bloques de código C utilizados para dividir un programa en subprogramas con un nombre específico asociado. Cabe recordar que en C la función principal es main().

Sintaxis o forma general de una función

```
<Especificador_de_tipo> <Identificador_de_función>(Lista de parámetros)
{
    /*
    ...
    Cuerpo de la función
    ...
    */
}
```

El especificador de tipo especifica el tipo de variable que devuelve la función, puede ser cualquier tipo válido. Si no se especifica ningún tipo el compilador asume por defecto un valor de tipo entero.

El identificador de función es el nombre que le asignamos a la función.

La lista de parámetros es una lista de posibles argumentos que son pasados a la función. Se escriben separados por comas. Cabe mencionar que puede haber funciones sin parámetros de entrada en ese caso deberá especificarlo a través de la palabra reservada void (quedar más claro luego cuando lo apliquemos a un ejemplo).

El cuerpo de la función es un conjunto de sentencias a través de las cuales realizamos una tarea específica para la cual ha sido creada la función, y suele tener la siguiente estructura:

```
...
{
definición de variables locales;
sentencias;
return;
}
```

A través de la expresión o palabra reservada ***return***, se devuelve el valor o resultado esperado por la función, dicho valor puede ser una variable, una constante o una expresión.

En resumen, ***return*** tiene dos usos importantes. Primero, forzar la salida inmediata de la función en que se encuentra, es decir, provoca que la ejecución del programa vuelva al código que llamó a la función. Segundo, puede ser usado para devolver un valor.

Ámbito de definición y desarrollo de una función

De la misma forma que procedemos con las variables, cuando una función va a ser usada en un programa, debe ser declarada.

La declaración consiste en especificar el tipo de datos que va a retornar la función, el o los argumentos y su tipo, si es que son necesarios. Por último el desarrollo de la función que no es ni más ni menos lo que hace la función o lo que se va a ejecutar cuando luego se invoque.

Veamos esto mostrando de forma genérica el ámbito de la definición y desarrollo de una función. Existen dos formas o ámbitos para llevar a cabo esta tarea y se describen a continuación.

Definición y desarrollo

En este caso tanto la declaración como el desarrollo se encuentran en el mismo ámbito, Luego de las directivas de preprocesamiento, como se observa a continuación:

```
#include <...>
<Especificador_de_tipo> <Identificador_de_función>(argumentos)
{
      /*
       ...
       Cuerpo de la función...
      */
}
main()
{
      Sentencias;
     Llamada a la función;
}
```

Ejemplo: Este ejemplo crea una función que simplemente suma dos números.

```
#include <stdio.h>
#include <stdlib.h>
//Definición y desarrollo de la función "suma".
int suma(int a,int b)
{
    return a+b;
}
int main() {
    system("cls");
    int a,b;
    printf ("Ingrese un valor\n");
    scanf("%d",&a);
    printf ("Ingrese otro valor\n");
    scanf("%d",&b);
    //se invoca la función "suma" con los parámetros a y b
    printf("La suma de %d + %d = %d",a,b,suma(a,b));
    return 0;
}
```

Prototipo y desarrollo

En este caso la declaración (prototipo) se realiza luego de las directivas de preprocesamiento y el desarrollo al finalizar el programa principal, es decir luego de la llave de cierre.

El compilador utiliza los prototipos para verificar las llamadas de las funciones. Es conveniente incluir prototipos de función para aprovechar la capacidad de C de verificación de tipo.

No es necesario especificar nombres de parámetros en los prototipos de función, pero se pueden especificar para mejorar la comprensión del código fuente. Es necesario colocar punto y coma al final de un prototipo.

Una llamada de función que no coincida con el prototipo de la función causará un error de sintaxis.

```
#include <...>
//Prototipo de la función
<Especificador_de_tipo><Identificador_de_función>(argumentos);

int main()
{
    Sentencias;
    Llamada a la función;

    return 0;
}

//Desarrollo de la función
<Especificador_de_tipo><Identificador_de_función>(argumentos)
{
     /*
    ...
    Cuerpo de la función
    ...
    */
}
```

Ejemplo:

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

int suma(int,int); //Prototipo de la función "suma"

int main()
{
    system("cls");
    int x,y;
    printf ("Ingrese un valor\n");
    scanf("%d",&x);
    printf ("Ingrese otro valor\n");
    scanf("%d",&y);
    printf("La suma de %d + %d = %d",x,y,suma(x,y));
    return 0;
}

//Desarrollo de la función "suma"
int suma(a,b)
{
    return a+b;
}
```

Cabe mencionar que por cada función debe haber un prototipo asociado de manera que el compilador entienda que existe una nueva función con su correspondiente desarrollo.

Antes de continuar con el tema específico de funciones es conveniente definir o repasar algunos conceptos que toman valor o trascienden a la hora de usar dichas funciones.

Variable locales y globales

Si bien el concepto de variable y sus variantes ya han sido definidos en otro apartado, cabe retomar su concepto aquí, por su rol importante en el uso de funciones.

Variables globales

Las variables globales se declaran fuera del cuerpo de cualquier función y se pueden acceder desde cualquier punto del programa. El siguiente ejemplo, define una variable global "nuevavar" y una variable local "a" en el cuerpo de main:

```c
#include <stdio.h>
#include <stdlib.h>
int nuevavar; // Variable global.
void carga(int);//Prototipo de la función

int main(void)
{
 int a; // Variable local en main.

 system("cls");
 printf("ingresar un valor para a\n");
 scanf("%d",&a);
 system("cls");
 printf("Valor de a antes de llamar a la función es %d \n",a);
 carga(a);//Llama a la función carga con un argumento entero
 printf("Se cargo contador con %d\n",nuevavar);
 printf("la variable a luego de llamar a la función es %d \n",a);
 return 0;
}

 //La función "carga" ve las dos variables.
 // "a" es parámetro de entrada.
 //nuevavar es global.

void carga(int a)
 {
    a++;
    nuevavar = a;
 }
```

En el ejemplo a no se ve afectada, nuevavar si, por estar declarada globalmente. Este tipo de usos no es apropiado justamente por lo poco modular que es la función carga.

Variables locales

Las variables que declaramos justo al principio del cuerpo de una función son variables locales, pueden ser utilizadas únicamente en la función que fueron declaradas.

En el siguiente ejemplo main(), tiene dos variables locales que se pueden ver dentro de la función porque funcionan como parámetro de entrada. La función "sumar" tiene una variable local llamada "sum" que se utiliza simplemente para cargar la suma y puede utilizarse fuera de la misma.

```c
#include <stdio.h>
```

```c
#include <stdlib.h>

int sumar(int,int);//Prototipo de la función

int main(void)
{
    int a,b; // Variables locales a main().
    system("cls");
    printf("ingresar un valor para a\n");
    scanf("%d",&a);
    printf("ingresar un valor para b\n");
    scanf("%d",&b);
    printf("La suma entre %d + %d = %d",a,b,sumar(a,b));
    return 0;
}

//función que suma dos variables
int sumar(int a,int b)
{
    int sum; //Variable local de la función "sumar"
    sum=a+b; //Guarda la suma y la devuelve a main a través de return
    return sum;
}
```

Variables locales en un bloque

Este concepto también lo incorporamos anteriormente, cabe mencionarlo nuevamente porque su funcionalidad es similar a la que cumple una variable local a una función.

Las variables locales se pueden definir en cualquier bloque de un programa, por lo tanto nacen y mueren en ese mismo bloque.

El siguiente ejemplo refleja esta situación: La variable j se declara dentro del ciclo (for i) y solo existe para ese ciclo, es decir cuando sale del bloque la variable deja de existir.

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    system("cls");

    for (i = 0; i < 3; i++)
```

```
    {//Comienzo bloque
        int j; /*Variable local, sólo existe para el bloque
                en la que fue declarada*/

        for (j = 0; j < 3; j++)
        {
            printf("%d-%d  ", i, j);
        }
        printf("\n");
    }//Fin del bloque

    return 0;
}
```

Formas de una función

Las funciones en C tienen cuatro formas particulares, estas formas dependen de: por un lado si tienen o no tienen argumentos, por el otro lado depende de si devuelven o no un valor determinado.

Una función en C sólo puede devolver un valor. Para devolver dicho valor, se utiliza la palabra reservada **return** cuya sintaxis ya conocemos:

```
return expresión;
```

Donde "expresión", puede ser cualquier tipo de dato salvo un array. Además, el valor de la expresión debe coincidir con el tipo de dato declarado en el prototipo de la función. Por otro lado, existe la posibilidad de devolver múltiples valores mediante la utilización de estructuras como argumentos

Por lo tanto la sentencia ***return*** permite finalizar la ejecución de la función y devolver un valor. En el caso de que la función sea void, no existirá la sentencia ***return*** o se puede escribir como return sin expresión, la ejecución finaliza al ejecutar la última instrucción del bloque y el flujo de mi programa regresa a donde fue invocada dicha función.

A continuación explicamos las diferentes formas que puede tomar una función, las siguientes líneas de código están fuera de contexto si desea probarlas en un IDE en particular, deberá completar el algoritmo.

La función no devuelve ningún valor y no tiene parámetros de entrada.

```c
//El "void" que se antepone al nombre de la función indica que no
retorna ningún valor

void suma(void)//No existen argumentos de entrada
{
    int x,y; //Declaración de variables locales
    printf ("Ingrese un valor\n");
    scanf("%d",&x);
    printf ("Ingrese otro valor\n");
    scanf("%d",&y);
    printf("La suma de %d + %d = %d",x,y,x+y);
}
```

En este caso, la función se invoca de la siguiente forma:

```c
...
int main()
{
    ...
    suma();//Llamada a la función
    return 0;
}
```

La función no devuelve ningún valor y tiene parámetros de entrada.

```c
//El "void" que se antepone al nombre de la función indica que no
retorna ningún valor

void suma(int a, int b) //Ingresan dos argumentos enteros

{
    //Se muestra en pantalla la suma de los dos argumentos ingresados
    printf("La suma de %d + %d = %d",a,b,a+b);
}
```

En este caso, la función se invoca de la siguiente forma, donde x e y son las variables que entran como argumento desde el programa principal:

```c
...
```

```
int main()
{
    int x,y;
    ...
    suma(x,y);//Llamada a la función
    ...
    return 0;
}
```

La función devuelve un valor y tiene parámetros de entrada.

```
//El "int" que se antepone al nombre de la función indica que
retorna un valor entero
...
int suma(int a, int b)//Ingresan dos argumentos enteros

{
    return a+b;//Retorna la suma de los dos argumentos
}
```

En este caso, podemos invocar a la función de dos formas posibles, una usándola directamente en otra función por ejemplo con printf() y otra asignando a una variable, tomando la precaución de que dicha variable esté debidamente declarada o siguiendo las reglas de promoción de C.

```
...
int main()
{
    int x,y,res;
    ...
    //Una forma de llamar a la función
    printf("El resultado es: %d",suma(x,y));
    //Otra forma de invocar a la función
    res=suma(x,y);
    return 0;
}
```

La función devuelve un valor y no tiene parámetros de entrada.

```
//El "int" que se antepone al nombre de la función indica que
retorna un valor entero

int suma(void) //No existen argumentos de entrada
{
    int a,b;
    printf ("Ingrese un valor\n");
    scanf("%d",&a);
    printf ("Ingrese otro valor\n");
    scanf("%d",&b);
    return a+b;//Retorna el resultado de la suma
}
```

La función se invoca sin argumentos, también teniendo la precaución de utilizar la variable del mismo tipo si es que se va a asignar a un valor específico.

```
...
int main()
{
int res;
    ...
    res=suma();//llamada a la función
    ...
    return 0;
}
```

En los cuatro casos solo se muestra el desarrollo de la función; para probarlo en un contexto de programa, corresponde recordar que hay que declarar la función (prototipo) y estudiar el lugar donde debe hacerse la invocación correspondiente.

A continuación mostramos un ejemplo completo de un programa en el que desarrollamos una función "suma" que toma como argumentos dos números enteros y retorna un valor correspondiente a la suma de ambos números.

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

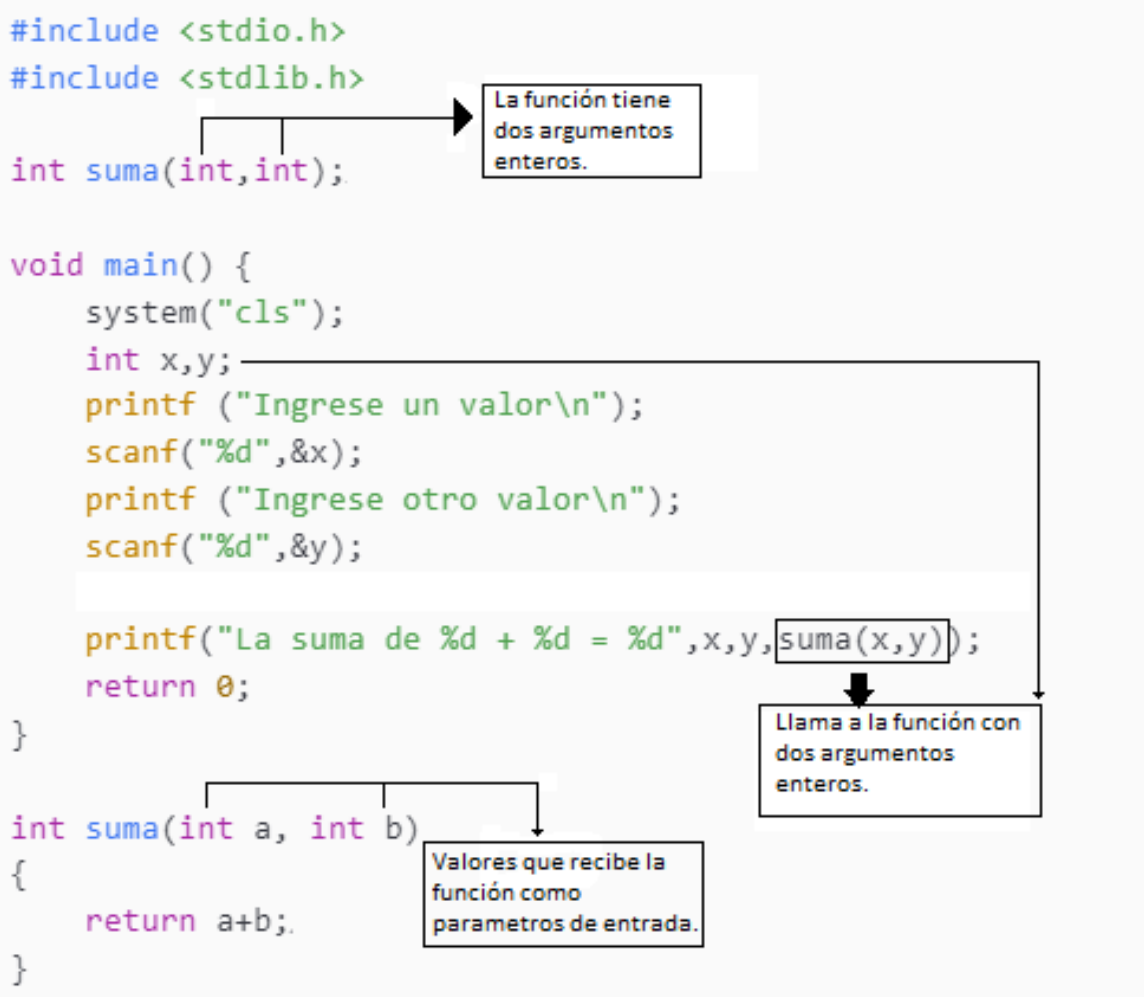
```c
#include <stdlib.h>

int suma(int,int);//Prototipo de la función

void main() {
    system("cls");
    int x,y;
    printf ("Ingrese un valor\n");
    scanf("%d",&x);
    printf ("Ingrese otro valor\n");
    scanf("%d",&y);
    //Se invoca a la función directamente en la salida
    printf("La suma de %d + %d = %d",x,y,suma(x,y));
    return 0;
}
//Desarrollo de la función
int suma(int a, int b)
{
    return a+b;//Retorna el resultado de la suma
}
```

Todos los ejemplos mostrados hasta aquí trabajan con el tipo int a los simples efectos de ejemplificar. Las funciones pueden retornar y tomar como argumentos cualquier tipo primitivo válido. En la medida que se avance en los ejercicios iremos viendo algunas de las diferentes posibilidades.

Paso de parámetros

Pasar parámetros o argumentos a una función, es básicamente entregarle un valor o valores a la función al momento de ser llamada. Si queremos que una función acepte valores externos a ella, es necesario que en su declaración coloquemos dichos parámetros, que son los "contenedores" donde se alojan las variables con las que nuestra función realizará su trabajo.

Volviendo al ejemplo anterior veamos en detalle cuales son los argumentos o parámetros de nuestra función y de qué forma dicha función los parametriza.



En este ejemplo cabe destacar que las variables que ingresan o pasan a la función, son variables de trabajo que solo usa la función para realizar una tarea específica sin modificar el valor original de ninguno de sus argumentos.

En la mayoría de los lenguajes hay dos formas de pasar las variables a una función, **por valor** o **por referencia**.

Pasaje por valor

En C el paso de parámetros está construido de tal forma que el paso de información es unidireccional: la información pasa de la función que hace la llamada a la función que recibe la llamada, pero no en el sentido inverso . Esto se denomina paso de parámetros **por valor**.

La función (o subrutina) recibe sólo una copia del valor que tiene la variable, y no puede modificarla.

Veamos el siguiente ejemplo:

```c
#include <stdio.h>
#include <stdlib.h>
void mostrar(int);
void main()
{   int valor=2;
    printf ("Antes de llamar a la funcion: %d\n",valor);
    mostrar(valor);
    printf ("Luego de llamar a la funcion: %d\n",valor);
    return 0;
}
void mostrar(int valor)
{   valor=5;
    printf("El valor dentro de la funcion: %d\n",valor);
}
```

La salida que provoca el ejemplo anterior es la siguiente:

```
El valor antes de llamar a la funcion: 2
El valor dentro de la funcion: 5
El valor luego de llamar a la funcion: 2
```

Nótese que la llamada a la función no afecta de ninguna manera el argumento o parámetro que ingresa desde el programa principal.

Pasaje por referencia

Esta forma en cambio, nos lleva a entregar prácticamente la variable original, es decir, si realizamos algún cambio en el parámetro de nuestra función, esto equivaldría a estar actuando directamente sobre la variable original.

En C normalmente los parámetros a funciones y subrutinas internas únicamente se pasan por valor o copia, no existe el paso por referencia conceptualmente hablando.

Existe una implementación muy particular de referencia denominada "puntero". Por definición, un puntero es un tipo especial de variable que almacena una dirección de memoria. De esta forma la función recibe dicha dirección y el impacto de modificar el valor contenido en dicho espacio de memoria, sí modifica el valor del parámetro que fue pasado. Profundizaremos en este tema en otro apartado, cuando abordemos el tema de punteros.