APRENDERAPROGRAMAR.COM

FUNCIONES EN C ¿QUÉ SIGNIFICA VOID? ¿QUÉ ES EL TIPO DE RETORNO? ¿PARA QUÉ SIRVE RETURN? MÓDULOS (CU00547F)

Sección: Cursos
Categoría: Curso básico de programación en lenguaje C desde cero

Fecha revisión: 2031

Resumen: Entrega nº47 del curso básico "Programación C desde cero".

Autor: Mario Rodríguez Rancel

FUNCIONES EN C. INTRODUCCIÓN

C está orientado a la programación por módulos en el sentido en que se describe el concepto de módulo en el curso "Bases de la programación nivel I" de aprenderaprogramar.com. Puede haber cierta confusión terminológica: los términos módulo, procedimiento, función y otros pueden tener distinto significado según el lenguaje de programación al que nos refiramos, de hecho en C no se usa el término procedimiento (*procedure*).

Vamos a hacer una clasificación libre tratando de conectar el lenguaje C con la terminología propia del curso "Bases de la programación nivel I" de aprenderaprogramar.com basado en pseudocódigo.

DESCRIPCIÓN	TERMINOLOGÍA DE C
Algoritmo principal	Función main
Módulo sin parámetros de entrada	Función sin parámetros de entrada
Módulo genérico con parámetros de entrada	Función con parámetros de entrada
Módulo de código que se ejecuta cuando es llamado desde algún punto del programa y no devuelve un valor	Función con tipo de retorno nulo (void). También se dice que es una función sin tipo de retorno.
Módulo de código que se ejecuta cuando es llamado desde algún punto del programa y devuelve un valor	Función con un tipo de retorno

FUNCIONES EN C. ASPECTOS PRINCIPALES

Una función en C es un fragmento de código que se puede llamar desde cualquier punto de un programa. En C podemos diferenciar entre dos tipos de funciones:

a) Aquellas cuyo tipo de retorno es **void (nulo)**, equiparables a lo que denominamos módulo genérico tipo procedimiento.

b) Aquellas cuyo tipo de retorno es **un tipo de dato** (como *int*, *double* o cualquier otro), equiparables a lo que denominamos módulo genérico tipo función.

Buscando analogías con el desarrollo que hicimos cuando hablamos de pseudocódigo, usaremos la función *main* de C para disponer en ella el código del "algoritmo principal" o guía del programa y el resto del código irá ordenado en diferentes funciones.

Habíamos dicho que un módulo no se ejecuta hasta que es llamado a ejecutarse desde el algoritmo principal de acuerdo con la sintaxis de pseudocódigo:

Llamar [Nombre del Módulo]

La forma habitual de una función en C es la siguiente:

```
tipoDeRetorno nombreFuncion ( tipoPar1 par1, tipoPar2 par2, ..., tipoParN parN) {
Instrucción 1;
Instrucción 2;
...
Instrucción n;
return valorDevueltoPorLaFuncion;
}
```

Como hemos dicho, una función devuelve un valor, de ahí que especifiquemos un tipo de dato para ella, que hemos indicado como tipoDeRetorno. En caso de que el tipo indicado en lugar de ser un tipo de dato válido en C (como *int*, *double* o cualquier otro) sea *void*, la función no devolverá nada y en lugar de terminar con *return valorDevueltoPorLaFuncion;* la terminaremos simplemente con *return;*. En realidad *return;* no es necesario si el tipo de la función es *void*, ya que C lo introducirá de forma automática si olvidamos incluirlo.

El flujo para una función sigue las reglas ya conocidas: al llegar el control a la sentencia *return* el flujo del programa vuelve a la sentencia inmediatamente posterior a la llamada efectuada. Si existe código posterior a la sentencia *return* final, éste será ignorado.

Las funciones pueden insertarse en el programa en cualquier orden, aunque siempre será recomendable tratar de disponerlas en el mismo orden en que esté previsto que se ejecuten.

La llamada a una función se realiza, cuando no hay parámetros que pasar, simplemente escribiendo su nombre seguido de unos paréntesis vacíos. La llamada a una función se hará normalmente para obtener un valor o asignar un valor a una variable, en expresiones del tipo:

printf (nombreDeLaFunción());
variable = nombreDeLaFunción();
If (nombreDeLaFunción() > variable) ...

Hay que recordar siempre que una "función" con tipo de retorno especificado ejecuta un código y devuelve un valor: podríamos decir que tiene una similitud importante con las variables: tener un valor.

La llamada a una función desde sí misma es posible, dando lugar a un anidamiento o recursión. Habrá de existir una condición que evolucione para dar lugar a la salida de la recursión, regresando el control del flujo a la instrucción posterior desde la que se autollamó el módulo. No vamos a desarrollar contenidos relativos a la recursión, ya que consideramos que es una materia de estudio avanzada.

Ejecuta estos dos pequeños programas para comprobar cómo trabajan las funciones con tipo de retorno *void* y las funciones con tipo de retorno especificado.

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

int sumaDosEnteros (int entero1, int entero2) {
   int resultado = 0;
   resultado = entero1 + entero2;
   return resultado;
}

int main() {
   printf("Bienvenidos al programa\n");
   printf("Si sumamos tres y cinco obtenemos %d\n",
sumaDosEnteros(3,5));
   return 0; // Ejemplos aprenderaprogramar.com
}
```

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

void sumaDosEnteros (int entero1, int entero2) {
   int resultado = 0;
   resultado = entero1 + entero2;
   printf("Si sumamos %d y %d obtenemos %d\n",
entero1, entero2, resultado);
   return; // Ejemplos aprenderaprogramar.com
}

int main() {
   printf("Bienvenidos al programa\n");
   sumaDosEnteros(3,5);
   return 0;
}
```

Como vemos, las funciones las hemos situado antes de la función *main*. Alternativamente, podemos optar por realizar sólo una declaración de la función antes de la función *main* y situar el cuerpo de las funciones después de la función *main*. Compruébalo ejecutando este código:

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

int sumaDosEnteros (int entero1, int entero2);

int main() {
   printf("Bienvenidos al programa\n");
   printf("Si sumamos tres y cinco obtenemos %d\n",
sumaDosEnteros(3,5));
   return 0; // Ejemplos aprenderaprogramar.com
}

int sumaDosEnteros (int entero1, int entero2) {
   int resultado = 0;
   resultado = entero1 + entero2;
   return resultado;
}
```

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

void sumaDosEnteros (int entero1, int entero2);

int main() {
   printf("Bienvenidos al programa\n");
   sumaDosEnteros(3,5);
   return 0; // Ejemplos aprenderaprogramar.com
}

void sumaDosEnteros (int entero1, int entero2) {
   int resultado = 0;
   resultado = entero1 + entero2;
   printf("Si sumamos %d y %d obtenemos %d\n",
entero1, entero2, resultado);
   return;
}
```

Podemos comprobar que a pesar de en un caso usar una función con tipo de retorno *int* y en otro caso usar una función sin retorno (o tipo de retorno *void*) el resultado del programa es el mismo: <<Bienvenidos al programa. Si sumamos 3 y 5 obtenemos 8>>.

Esto ya nos indica que normalmente existirán distintas vías o diseños para obtener un resultado. Normalmente en programación se habla de buenos diseños si son eficientes, claros y coherentes, y malos diseños si son ineficientes, poco claros o poco coherentes. Crear buenos diseños es algo que se aprende estudiando, practicando y observando la forma de diseñar que tienen otros programadores.

EJERCICIO RESUELTO Nº1: ENUNCIADO

El siguiente pseudocódigo es un ejemplo muy básico referente a la idea de introducir módulos a la hora de programar. Transformar este pseudocódigo en un programa en lenguaje C.

1. **Inicio [PROGRAMA COMUNICADO]**
   2. Mostrar "Comunicado de la empresa"
   3. Llamar Saludo
   4. Llamar Comunicado
   5. Llamar Despedida
6. **Fin**

Módulo Saludo
1. Mostrar "Con motivo de la celebración el próximo día 5 del Día Mundial del Medioambiente la empresa saluda a todos los empleados y les agradece el compromiso con el cuidado de la naturaleza"

FinMódulo

Módulo Comunicado
1. Mostrar "Con motivo de dicha conmemoración está previsto realizar un acto de plantación de árboles en los jardines del edificio central el próximo día 5 a las 12 del mediodía al que están todos invitados"

FinMódulo

Módulo Despedida
1. Mostrar "La empresa agradece su participación y les invita a sumarse al programa <<Empleados por una ciudad sostenible>>. Atentamente, El Director General"

FinMódulo

EJERCICIO RESUELTO Nº1: SOLUCIÓN

```
#include <stdio.h>
#include <stdlib.h>
void saludo(); void comunicado(); void despedida();
int main() {
   printf("Comunicado de la empresa\n\n");
   saludo();    comunicado();    despedida();   return 0;
}

void saludo() {
   printf("Con motivo de la celebracion el proximo dia 5 del Dia Mundial del Medioambiente la empresa saluda a ");
   printf("todos los empleados y les agradece el compromiso con el cuidado de la naturaleza\n\n");  return;
}

void comunicado() {
   printf("Con motivo de dicha conmemoracion esta previsto realizar un acto de plantacion de arboles en los ");
   printf("jardines del edificio central el proximo dia 5 a las 12 del mediodia al que estan todos invitados\n\n");
   return; // Ejercicios resueltos aprenderaprogramar.com
}

void despedida() {
   printf("La empresa agradece su participacion y les invita a sumarse al programa <<Empleados por una ");
   printf("ciudad sostenible>>.\n\n Atentamente, El Director General\n\n");    return; }
```

El único interés de este ejercicio es reforzar la idea de estructura basada en algoritmo principal e invocación de módulos a la hora de construir nuestros programas. En cuanto a los aspectos específicos del lenguaje C, vemos cómo hemos declarado los módulos o funciones en la cabecera del programa

pero los hemos desarrollado después de la función *main*. También es válido declarar y desarrollar los módulos directamente en la cabecera del programa.

Por otro lado, vemos cómo en las funciones con tipo de retorno nulo (*void*) hemos incluido la sentencia *return*, aunque en este tipo de funciones dicha sentencia no es obligatoria y la podemos omitir.

EJERCICIO

Estudia el siguiente código:

```
#include <stdio.h>
#include <stdlib.h>
int max(int num1, int num2);

int main () {
  int a = 100;
  int b = 200;
  int ret;
  ret = max(a, b);
  printf( "Max value is : %d\n", ret );
  return 0;
}

int max(int num1, int num2) {
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;
  return result; //Ejercicios aprenderaprogramar.com
}
```

a) Sin ejecutar el código (sólo pensando) responde: ¿para qué sirve la función max? ¿cuál será el resultado del programa?

b) ¿En qué línea se produce la llamada a la función max? ¿Qué valor se almacenará en la variable ret y por qué?

c) Reescribe el código de modo que todos los if lleven corchetes delimitadores { ... } tanto para los if como para los else.

d) Crea un programa análogo a este con una función que permita determinar, dados tres números, cuál es el mayor. Invoca la función pasándole como datos 23, 87, 45 y muestra el resultado devuelto por la función por pantalla.

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00548F

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=82&Itemid=210