APRENDERAPROGRAMAR.COM

FUNCIONES CON Y SIN PARÁMETROS EN C. INVOCAR. SIGNATURAS. ARGUMENTOS. TIPO DE RETORNO (CU00549F)

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

Fecha revisión: 2031

Resumen: Entrega nº49 del curso básico "Programación C desde cero".

Autor: Mario Rodríguez Rancel

PASO DE PARÁMETROS A FUNCIONES

Ya hemos visto que en C la signatura de una función (una declaración del tipo *int calculo ();*) siempre lleva unos paréntesis donde se pueden indicar parámetros de entrada. Los valores que se pasan a la función se denominan argumentos para la función y son indicados por el programador.

Veamos una situación de función sin parámetros y otra con parámetros.

a)

```
void calcular() { ... }
```

b)

```
double calcular (int a, int b, double c) { ... }
```

El primer código corresponde a una función con tipo de retorno nulo (*void*) y sin parámetros.

El segundo código corresponde a una función con tipo de retorno *double* y que trabaja con tres parámetros que se deberán pasar cuando sea invocada: el parámetro a de tipo entero, el parámetro b de tipo entero y el parámetro c de tipo *double*.

Una función puede requerir ningún, uno, dos, tres, cuatro o más argumentos, según se haya especificado en la signatura.

Dentro de los paréntesis se indican los parámetros de entrada requeridos, y el tipo de dato que han de ser. Si existen varios parámetros se escriben separados por comas, por ejemplo: *(int a, int b, double c)*. Si no se especifica tipo de dato para los parámetros el compilador considerará que se tratan de tipo *int*. No es aconsejable dejar sin indicar el tipo de un parámetro de una función, ya que genera inseguridad cuando menos en la intención del programador. Por ejemplo *(double numero, valor, nombre)* daría lugar a que los parámetros requeridos fueran *numero* como real de precisión doble, y *valor* y *nombre* serían considerados tipo *int*. Pero repetimos que no es aconsejable dejar sin indicar el tipo de los parámetros de una función.

La llamada a una función se hace con la sintaxis: *nombreFuncion (param1, param2, ..., paramN);*

En caso de no requerirse ningún parámetro se dejan los paréntesis vacíos: *nombreFuncion();*

El tipo de los argumentos debe ser compatible con el tipo indicado en la signatura de la función. Es decir, si la función espera por ejemplo un tipo *char* y le pasamos un *double* obtendremos un error.

Una declaración de función genérica que espera un parámetro de entrada podría ser esta:

```
double raiz (double numero) {

            .
            .
            .
}
```

Dado que una función con tipo de retorno distinto de *void* devuelve un resultado, se pueden usar para asignar contenido a una variable, para mostrar algo en pantalla, o para cualquier proceso donde pudiéramos insertar una variable del mismo tipo que la función. Por ejemplo:

resultado = raiz(dato)

printf("La raiz de 9 es \n", raiz(9.0));

Ejecuta este código para comprobar distintas formas de uso de una función:

```
//Programa Ejemplo04 aprenderaprogramar.com
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double dato=0.0; double resultado=0.0;
void raiz (double numero);
int main() {
   raiz(9);    raiz(-144);
   printf("Introduzca un dato para calcular raiz: ");
   scanf("%lf", &dato);
   raiz(dato);
   return 0;
}
void raiz(double numero) {
   numero = abs(numero);    resultado = sqrt(numero);
   printf("Raiz calculada: %.2lf \n\n", resultado);
}
```

Si tratáramos de realizar una llamada del tipo *raiz("Nueva York")* obtendríamos un error del tipo: "*expected 'double' but argument is of type 'char *'*"

El programa anterior utiliza una función con tipo de retorno *void* (que en otros lenguajes se denomina procedimiento). Pero hay un aspecto de diseño que podemos considerar incorrecto: una función como *raiz*, cuyo objetivo es calcular una raíz cuadrada, posiblemente debería limitarse a recibir el valor para el cual se quiere calcular la raíz, hacer las validaciones o comprobaciones oportunas y devolver el resultado. ¿Tiene sentido que se encargue de mostrar por pantalla el resultado? En principio no, una función debe tener una tarea concreta y bien delimitada, evitando hacer tareas complementarias o que puedan corresponder a otras funciones. El mismo "objetivo" cumplido con una función con tipo de

retorno *double* y con un cambio en el diseño lo exponemos a continuación. Ejecútalo y comprueba el resultado. Recuerda que una función con un tipo de retorno, además de ejecutar un código, devuelve un valor.

```
//Programa Ejemplo04 rediseño 1 aprenderaprogramar.com
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double raiz (double numero);

int main() {
   double dato=0.0; printf("Raiz calculada: %.2lf \n\n", raiz(9));
   printf("Raiz calculada: %.2lf \n\n", raiz(-144));
   printf("Introduzca un dato para calcular raiz: ");
   scanf("%lf", &dato);     printf("Raiz calculada: %.2lf \n\n", raiz(dato));
   return 0;
}

double raiz(double numero) {
   double resultado=0.0;  numero = abs(numero);  resultado = sqrt(numero);
   return resultado;
}
```

Aún hay algo que llama la atención en este diseño: dentro del método *main* se repite en diferentes ocasiones la invocación a *printf* repitiendo un mensaje similar. ¿No podríamos crear una función que haga el código más compacto y fácil de entender? Eso es lo que se ha intentado en el siguiente código. Ejecútalo y comprueba los resultados:

```
//Programa Ejemplo04 rediseño 2 aprenderaprogramar.com
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double raiz (double numero); void mostrar(double numero);

int main() {
   double dato=0.0;
   mostrar(raiz(9));     mostrar(raiz(-144));
   printf("Introduzca un dato para calcular raiz: ");
   scanf("%lf", &dato);     mostrar(raiz(dato));
   return 0;
}

double raiz(double numero) {
   double resultado=0.0;     numero = abs(numero);
   resultado = sqrt(numero);     return resultado;
}

void mostrar(double valor) {     printf("Raiz calculada: %.2lf \n\n", valor); }
```

Hemos hecho dos rediseños o refactorizaciones del código, y posiblemente podríamos hacer algunas más. Es frecuente reescribir programas para tratar de hacerlos más eficientes y más legibles, pero tampoco debemos obsesionarnos con lograr "el código perfecto". Esto, cuando hablamos de programas de una gran extensión, se convierte en una tarea que consume demasiados recursos como para ser un objetivo realista.

EJERCICIO Nº1

Escribe una función que use un bucle for para calcular la potencia de un número al que denominaremos base sobre otro número al que denominaremos exponente, ambos recibidos como parámetros.

Ejemplo: la función recibe como base el 4 y como exponente el 3. El resultado devuelto debe ser el resultado de multiplicar la base por sí misma 3 veces, en este ejemplo 4*4*4 = 64.

Escribe un programa donde se pida al usuario base y exponente y se muestre el resultado de calcular la potencia (base elevada al exponente). Tras esto, se debe pedir al usuario si quiere repetir con otros datos o no (s/n). En caso de que el usuario elija s se le volverá a pedir base y exponente, y en caso contrario el programa debe finalizar.

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

EJERCICIO Nº 2

Estudia el siguiente código y responde a las preguntas:

```
#include <stdio.h>
#include <stdlib.h>
// Ejercicios curso C aprenderaprogramar.com
void stars2 (int n) {
        int i;
        for (i=1; i<=n; ++i) {
                printf ("*");
        }
        printf ("\n");
}

int main (void) {
        int a;
        a=10;
        stars2 (20);
        stars2 (a);
        stars2 (a+2);
        return (0);
}
```

a) ¿Cuántas funciones se declaran en este código? ¿Es main una función?

b) Explica cuántos parámetros reciben y de qué tipo son los parámetros para cada una de las funciones que existan en este código.

c) ¿Cuántas veces se invoca la función stars2 en el código? ¿Qué ocurre con cada una de esas invocaciones? ¿Cuál es el valor devuelto por la función stars2?

d) ¿Qué ocurre si escribimos stars2(0)? ¿Por qué?

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

Próxima entrega: CU00550F

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