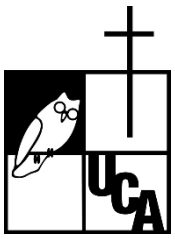
Universidad Centroamericana “José Simeón Cañas”
Facultad de Ingeniería y Arquitectura

Tutorías - Fundamentos de Programación Ciclo 01/2021
Tutor: Ernesto José Canales Guillén



Funciones definidas por el usuario en C++

¿Qué es una función?

Las funciones son como bloques de construcción. Permiten dividir programas complicados en piezas manejables. También tienen otras ventajas:

- Mientras trabaja en una función, puede concentrarse solo en esa parte del programa y construirlo, depurarlo y perfeccionarlo.
- Diferentes personas pueden trabajar en diferentes funciones simultáneamente.
- Si se necesita una función en más de un lugar en un programa o en diferentes programas, puede escribirlo una vez y usarlo muchas veces.
- El uso de funciones mejora en gran medida la legibilidad del programa porque reduce la complejidad de la función **main**.

Las funciones se denominan a menudo **módulos**. Son como programas en miniatura; puedes ponerlos juntos para formar un programa más complejo. Cuando se analizan las funciones definidas por el usuario, verás que este es el caso. Esta capacidad es menos evidente con ***funciones predefinidas***. Porque su código de programación no está disponible para nosotros. Sin embargo, debido a que las funciones ya están escritas para nosotros, las aprenderá primero para que pueda usarlas cuando sea necesario.

Predefined Functions

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cmath>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int` (`double`)	`int` (`double`)
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle: `x`: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than `x`:`floor(45.67) = 45.00`	`double`	`double`
`islower(x)`	`<cctype>`	Returns 1 (`true`) if `x` is a lowercase letter; otherwise, it returns 0 (`false`); `islower('h')` is 1 (`true`)	`int`	`int`
`isupper(x)`	`<cctype>`	Returns 1 (`true`) if `x` is an uppercase letter; otherwise, it returns 0 (`false`); `isupper('K')` is 1 (`true`)	`int`	`int`
`pow(x, y)`	`<cmath>`	Returns x^y; if `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`sqrt(x)`	`<cmath>`	Returns the nonnegative square root of `x`; `x` must be nonnegative: `sqrt(4.0) = 2.0`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, it returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of `x` if `x` is lowercase; otherwise, it returns `x`	`int`	`int`

User-Defined Functions

Las funciones definidas por el usuario en C ++ se clasifican en dos categorías:

- ***Value-returning functions***: funciones que tienen un tipo de devolución. Estas funciones devuelven un valor de un tipo de datos específico usando la declaración return.
- ***Void functions***: funciones que no tienen un tipo de retorno. Estas funciones no usan una declaración return para devolver un valor.

Sintaxis básica de una función de retorno (*Value-returning functions*)

```
functionType functionName(formal parameter list)
{
    statements
}
```

Sintaxis básica de una función sin retorno (*Void functions*)

```
void functionName(formal parameter list)
{
    statements
}
```

Sintaxis básica del *formal parameter list*

```
dataType identifier, dataType identifier, ...
```

Sintaxis básica para llamar una función

```
functionName(actual parameter list)
```

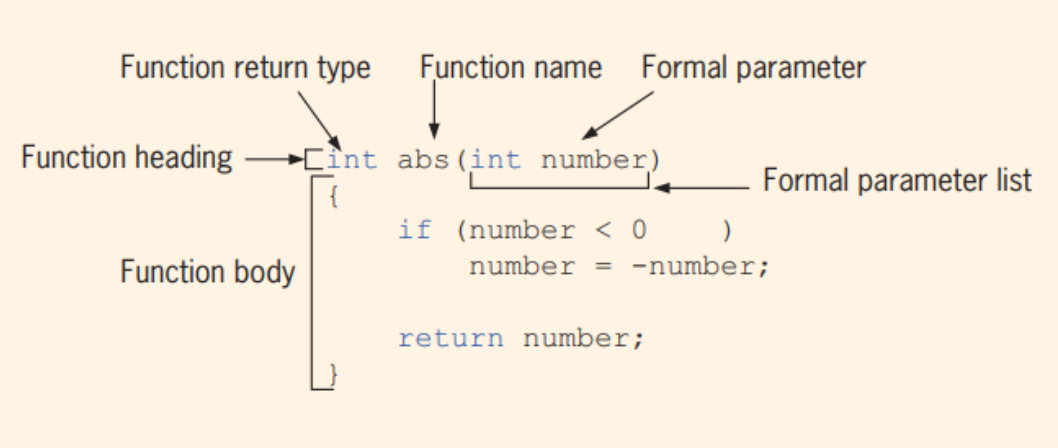
Sintaxis del *return*

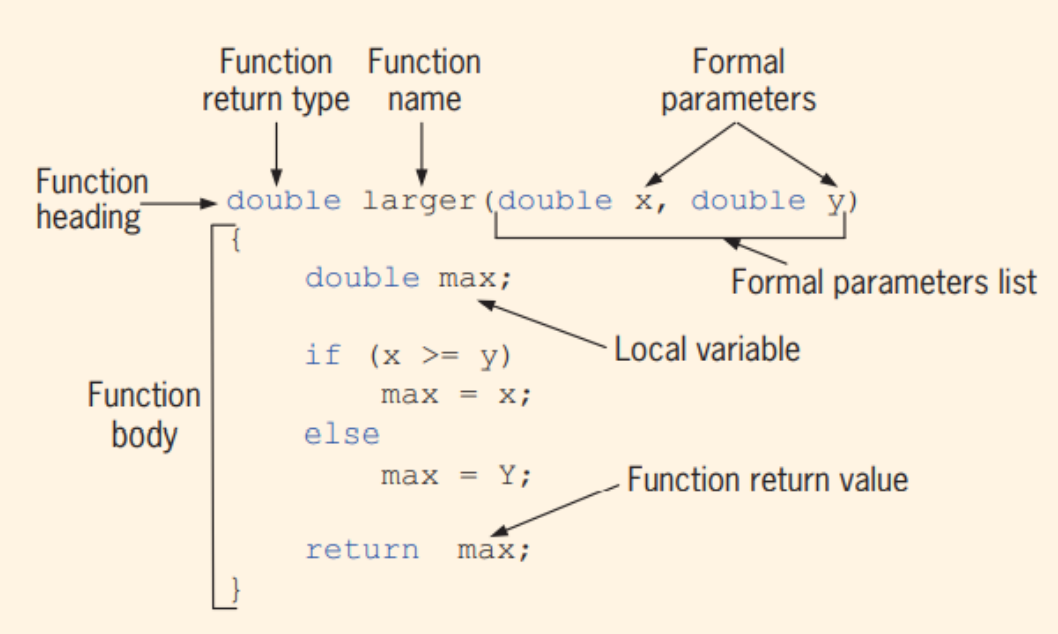
Cuando una declaración return se ejecuta en una función, la función termina inmediatamente, y el control vuelve al origen que llama. Además, la declaración de llamada a la función se reemplaza por el valor devuelto por la declaración return. Cuando una declaración return se ejecuta en la función principal (main), el programa termina.

```
return expr;
```

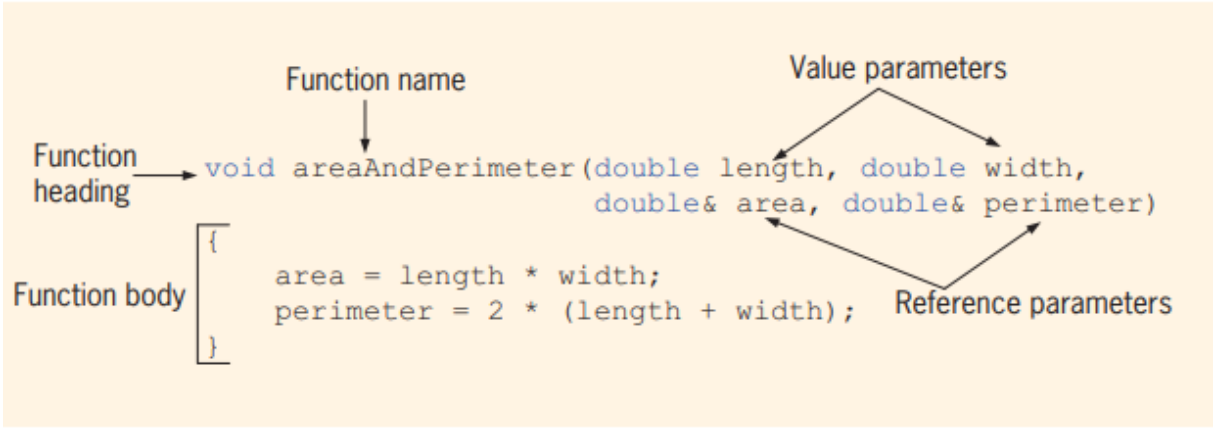
Ejemplos:

Value-returning function





Void functions



Prototipo de función (*Function Prototype*)

Ahora que tiene una idea de cómo escribir y usar funciones en un programa, La siguiente pregunta se relaciona con el orden en que deben aparecer las funciones definidas por el usuario en un programa. Por ejemplo, ¿coloca la función larger antes o después de la función main? ¿Debería colocarse larger antes de compareThree o después?

Siguiendo la regla que primero debes declarar un identificador antes de poder usarlo y saber que la función main usa el identificador larger, lógicamente debes colocar larger antes de main.

En realidad, los programadores de C ++ suelen colocar la función principal antes que todas las demás funciones definidas por el usuario. Sin embargo, esta organización podría producir un error de compilación porque las funciones se compilan en el orden en que aparecen en el programa.

Para solucionar este problema de identificadores no declarados, colocamos prototipos de funciones antes de cualquier definición de función (incluida la definición de principal).

Prototipo de función: el encabezado de la función sin el cuerpo de la función.

Sintaxis de un prototipo de función (*Function Prototype*)

Tenga en cuenta que el prototipo de la función termina con un punto y coma.

```
functionType functionName(parameter list);
```

Ejemplo:

```
1  void myFunction(); //Function Prototype
2
3  int main() {
4    myFunction();
5
6    return 0;
7  }
8
9  void myFunction() {
10   // code to be executed
11 }
```