



# Fundamentos de Programación 101

By Ernie

Ernesto José Canales Guillén

Círculos de estudio UCA

Ciclo Virtual 01/2021



# Funciones definidas por el usuario



# ¿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.



# ¿Qué es una función?

- 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
<code>abs (x)</code>	<code>&lt;cmath&gt;</code>	Returns the absolute value of its argument: <code>abs (-7) = 7</code>	<code>int</code> ( <code>double</code> )	<code>int</code> ( <code>double</code> )
<code>ceil (x)</code>	<code>&lt;cmath&gt;</code>	Returns the smallest whole number that is not less than <code>x</code> : <code>ceil (56.34) = 57.0</code>	<code>double</code>	<code>double</code>
<code>cos (x)</code>	<code>&lt;cmath&gt;</code>	Returns the cosine of angle: <code>x</code> : <code>cos (0.0) = 1.0</code>	<code>double</code> (radians)	<code>double</code>
<code>exp (x)</code>	<code>&lt;cmath&gt;</code>	Returns $e^x$ , where $e = 2.718$ : <code>exp (1.0) = 2.71828</code>	<code>double</code>	<code>double</code>
<code>fabs (x)</code>	<code>&lt;cmath&gt;</code>	Returns the absolute value of its argument: <code>fabs (-5.67) = 5.67</code>	<code>double</code>	<code>double</code>



Function	Header File	Purpose	Parameter(s) Type	Result
<code>floor(x)</code>	<code>&lt;cmath&gt;</code>	Returns the largest whole number that is not greater than <code>x</code> : <code>floor(45.67) = 45.00</code>	<code>double</code>	<code>double</code>
<code>islower(x)</code>	<code>&lt;cctype&gt;</code>	Returns 1 ( <code>true</code> ) if <code>x</code> is a lowercase letter; otherwise, it returns 0 ( <code>false</code> ); <code>islower('h')</code> is 1 ( <code>true</code> )	<code>int</code>	<code>int</code>
<code>isupper(x)</code>	<code>&lt;cctype&gt;</code>	Returns 1 ( <code>true</code> ) if <code>x</code> is an uppercase letter; otherwise, it returns 0 ( <code>false</code> ); <code>isupper('K')</code> is 1 ( <code>true</code> )	<code>int</code>	<code>int</code>
<code>pow(x, y)</code>	<code>&lt;cmath&gt;</code>	Returns <code>x<sup>y</sup></code> ; if <code>x</code> is negative, <code>y</code> must be a whole number: <code>pow(0.16, 0.5) = 0.4</code>	<code>double</code>	<code>double</code>
<code>sqrt(x)</code>	<code>&lt;cmath&gt;</code>	Returns the nonnegative square root of <code>x</code> ; <code>x</code> must be nonnegative: <code>sqrt(4.0) = 2.0</code>	<code>double</code>	<code>double</code>
<code>tolower(x)</code>	<code>&lt;cctype&gt;</code>	Returns the lowercase value of <code>x</code> if <code>x</code> is uppercase; otherwise, it returns <code>x</code>	<code>int</code>	<code>int</code>
<code>toupper(x)</code>	<code>&lt;cctype&gt;</code>	Returns the uppercase value of <code>x</code> if <code>x</code> is lowercase; otherwise, it returns <code>x</code>	<code>int</code>	<code>int</code>



# User-Defined Functions





# 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

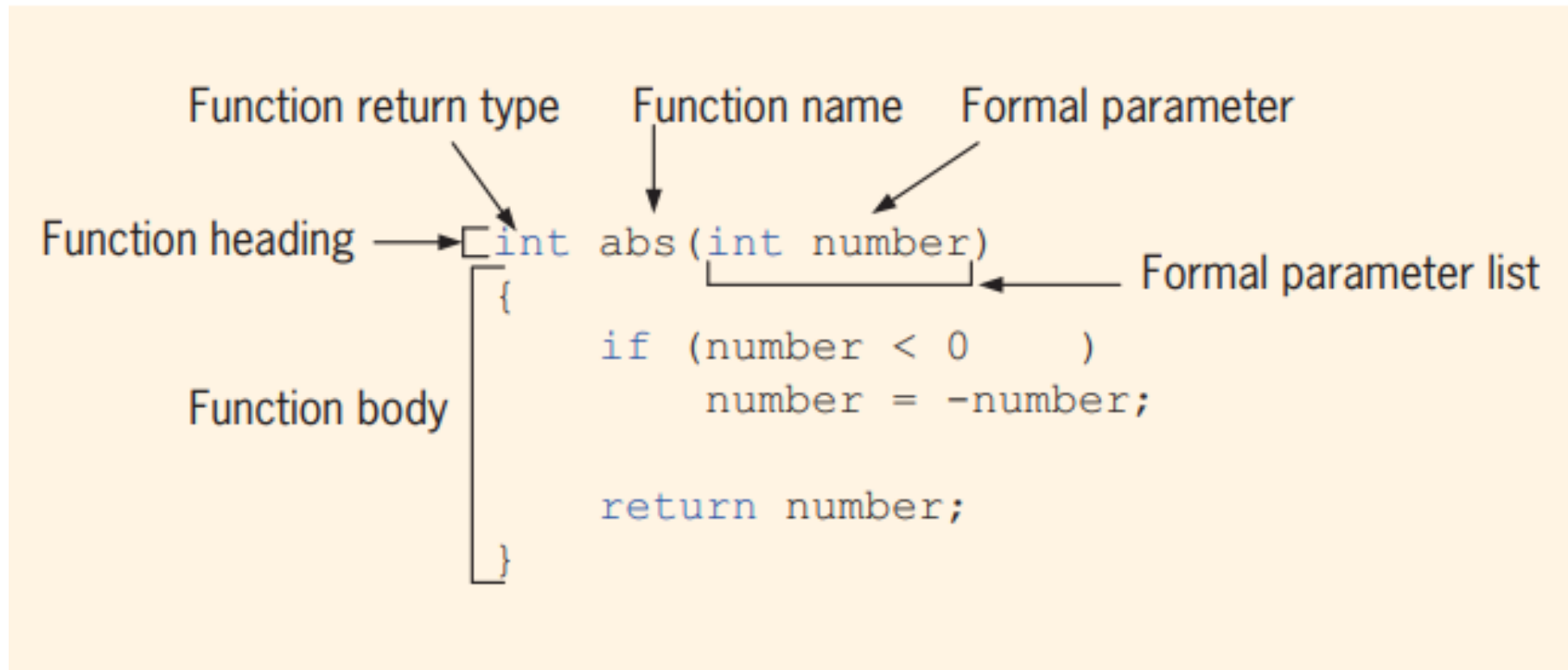
Cuando una declaración `return` se ejecuta en una función, la función termina inmediatamente, y el control vuelve al origen que llama. Además, la declaración de llamada a la función se reemplaza por el valor devuelto por la declaración `return`. Cuando una declaración `return` se ejecuta en la función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

```
return expr;
```

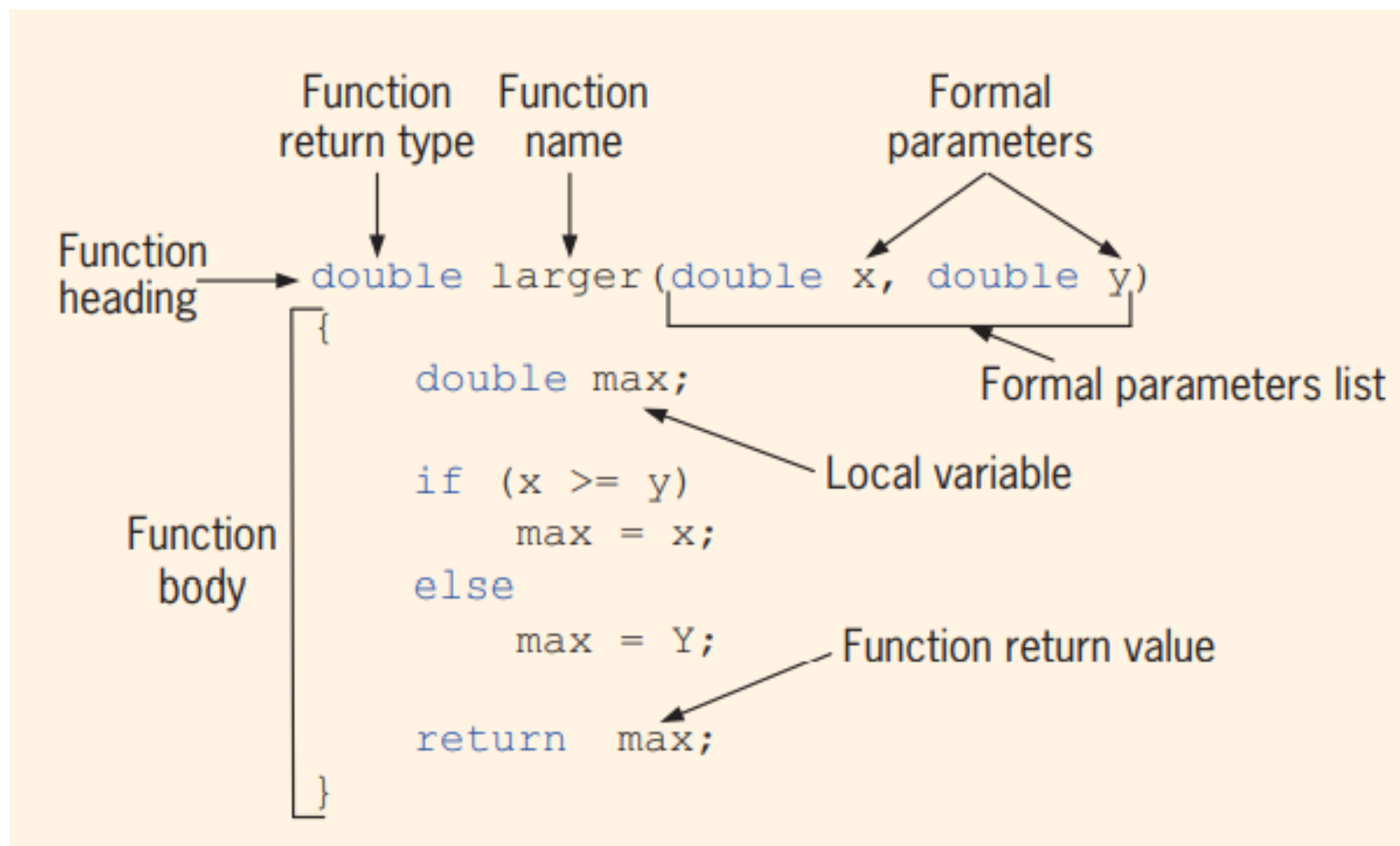


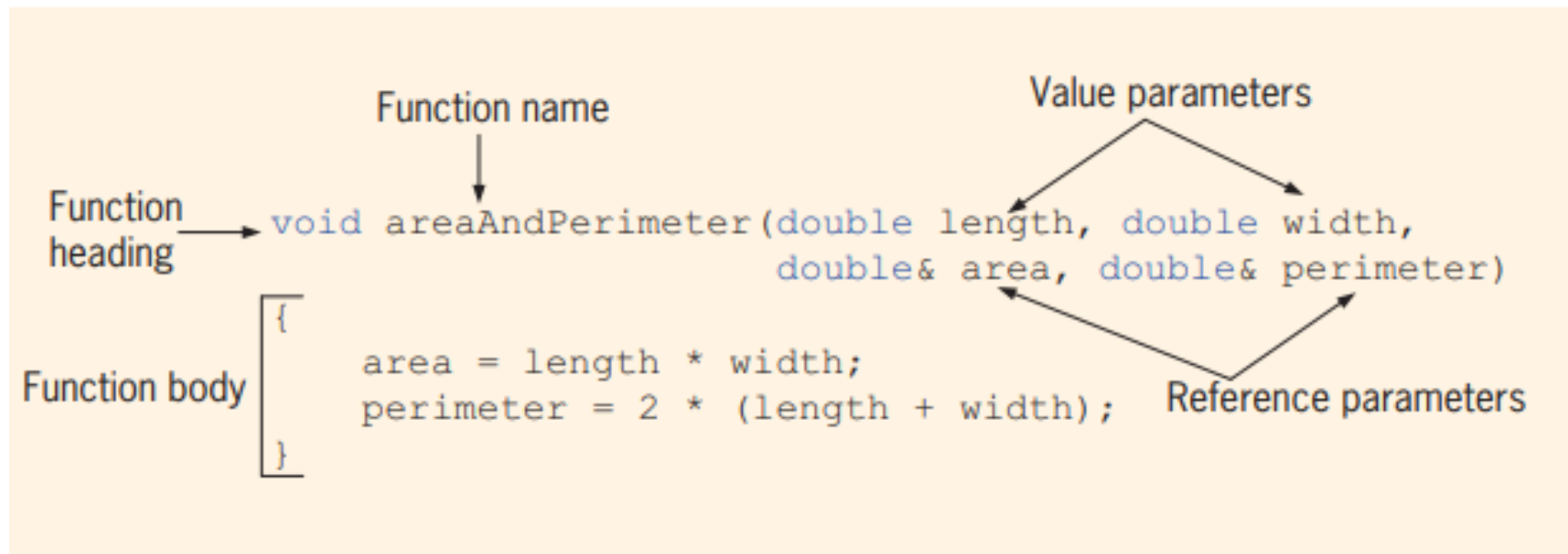
# Ejemplos

Value-returning y Void











# Function Prototype

El encabezado de la función sin el cuerpo de la función.



# 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.
- 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).



# Sintaxis de un prototipo de función (Function Prototype)

```
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 }
```



- L. J. Aguilar, Programación en C++. Algoritmos, estructuras de datos y objetos, Aravaca (Madrid): McGRAW-HILL, 2006.
- D. Malik, C++ Programming: From Problem Analysis to Program Design, Boston, MA: Cengage Learning, 2003.