Fundamentos de Programación 101

By Ernie

Ernesto José Canales Guillén

Círculos de estudio UCA

Ciclo Virtual 01/2021

Introducción a la programación en C++

Declarar una variable

Cuando declaramos una variable, no solo especificamos el nombre de la variable, también especificamos qué tipo de datos puede almacenar una variable. A La regla de sintaxis para declarar una variable es:

tipoDeDato Identificador;

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
double myDoubleNum = 9.98;   // Floating point number
char myLetter = 'D';         // Character
bool myBoolean = true;       // Boolean
string myText = "Hello";     // String
```

Operadores en C++

Los operadores se utilizan para realizar operaciones en variables y valores. C++ divide los operadores en los siguientes grupos:

Operadores de Asignación

Se utilizan para asignar valores a las variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Operadores de comparación

Se utilizan para comparar dos valores. El valor de retorno de una comparación es verdadero (1) o falso (0).

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Operadores lógicos

Se utilizan para determinar la lógica entre variables o valores.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Operadores bit a bit

Realizan operaciones sobre datos enteros a nivel de bit individual. Estas operaciones incluyen probar, configurar o cambiar los bits reales.

Operator	Description
&	Bitwise AND Operator
|	Bitwise OR Operator
^	Bitwise XOR Operator
~	Bitwise Complement Operator
<<	Bitwise Shift Left Operator
>>	Bitwise Shift Right Operator

Operadores aritméticos

Se utilizan para realizar operaciones matemáticas comunes.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Aritmética de caracteres

Debido a que el tipo de datos char también es un tipo de datos integral, C++ le permite realizar operaciones aritméticas en datos char. Sin embargo, debes usar esta habilidad con cuidado.

Hay una diferencia entre el carácter '8' y el entero 8.

El valor entero de 8 es 8.

El valor entero de '8' es 56, que es la clasificación ASCII secuencia del carácter '8'.

Expresiones

Hay tres tipos de expresiones aritméticas en C++.

Expresiones aritméticas

- **Expresiones integrales**: todos los operandos de la expresión son números enteros. Una la expresión integral produce un resultado integral.

- **Expresiones de coma flotante (decimal)**: todos los operandos de la expresión son puntos flotantes (números decimales). Una expresión de punto flotante produce un resultado de punto flotante.

- **Expresiones mixtas**: la expresión contiene números enteros y decimales números.

Reglas al evaluar una expresión mixta

1. Al evaluar un operador en una expresión mixta:
   - Los operandos enteros, por lo tanto, producen un resultado entero; los números de punto flotante producen un número de punto flotante.
   - Si el operador tiene ambos tipos de operandos, luego durante el cálculo, el entero se cambia a un número de punto flotante con la parte decimal de cero y se evalúa el operador. El resultado es un número de punto flotante.

2. Toda la expresión se evalúa de acuerdo con las reglas de precedencia; operadores que tienen el mismo nivel de precedencia se evalúa de izquierda a derecha. El agrupamiento por paréntesis permite mayor claridad.

Mixed Expression	Evaluation	Rule Applied

`3/2+5.5`	`=1+5.5`
	`=6.5`
		`3/2=1` (integer division; Rule 1(a))
		`(1+5.5`
		`=1.0+5.5` (Rule 1(b))
		`=6.5)`

`15.6/2+5`	`=7.8+5`
	
	`=12.8`
		`15.6/2`
		`=15.6/2.0` (Rule 1(b))
		`=7.8`
		`7.8+5`
		`=7.8+5.0` (Rule1(b))
		`=12.8`

`4+5/2.0`	`=4+2.5`
	
	`=6.5`
		`5/2.0=5.0/2.0` (Rule1(b))
		`=2.5`
		`4+2.5=4.0+2.5` (Rule1(b))
		`=6.5`

`4*3+7/5-25.5`	`=12+7/5-25.5`
	`=12+1-25.5`
	`=13-25.5`
	`=-12.5`
		`4*3=12` (Rule 1(a))
		`7/5=1` (integer division; Rule 1(a))
		`12+1=13` (Rule 1(a))
		`13-25.5=13.0-25.5` (Rule 1(b))
		`=-12.5`

Conversión de tipo (casting)

Cuando un valor de un tipo de datos se cambia automáticamente a otro tipo de datos, se dice que ha ocurrido una coerción de tipo implícita. Como los ejemplos en el la sección anterior ilustra, si no tiene cuidado con los tipos de datos, la coerción de tipo implícita puede generar resultados inesperados.

Para evitar la coerción de tipo implícita, C++ proporciona una conversión de tipo explícita a través del uso de un cast operador:

static_cast<tipoDeDato> (expresión)

Expression	Evaluates to
`static_cast<int>(7.9)`	`7`
`static_cast<int>(3.3)`	`3`
`static_cast<double>(25)`	`25.0`
`static_cast<double>(5+3)`	`=static_cast<double>(8)=8.0`
`static_cast<double>(15)/2`	`=15.0/2` (because `static_cast<double>(15)=15.0`) `=15.0/2.0=7.5`
`static_cast<double>(15/2)`	`=static_cast<double>(7)` (because `15/2=7`) `= 7.0`
`static_cast<int>(7.8 + static_cast<double>(15)/2)`	`=static_cast<int>(7.8+7.5)` `=static_cast<int>(15.3)` `= 15`
`static_cast<int>(7.8 + static_cast<double>(15/2))`	`=static_cast<int>(7.8 + 7.0)` `=static_cast<int>(14.8)` `= 14`

Self-learning

Self-learning

W3Schools

freeCodeCamp.org

ProgramacionATS

LACONIC

Bibliografía

- 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.