



Operadores de C++

Autor: Juan Camilo Correa Chica
Profesor Facultad de Ingeniería
Universidad de Antioquia

Los operadores de C++ son un conjunto de símbolos, caracteres especiales del código ASCII, o palabras que representan alguna operación, ya sea matemática, lógica o relacional que puede realizar C++ entre uno o más operandos, dependiendo de la naturaleza de la misma.

Los operadores sirven para construir expresiones que hacen parte de un algoritmo, en general, lo que hacen es manipular variables y valores para luego arrojar un resultado de acuerdo con el tipo de operación ejecutada.

En general, se puede afirmar que con ayuda del conjunto de operadores de C++ se pueden representar fórmulas matemáticas, calcular y comparar cantidades, y ejecutar operaciones lógicas que corresponden, respectivamente, a ecuaciones, comparaciones y relaciones entre operandos de un algoritmo en particular.

Los operadores de C++ se clasifican en tres tipos de acuerdo con la cantidad de operandos que involucra la operación, y son:

1. Unarios cuando solo hay un operando (incremento, negación, etc.)
2. Binarios cuando hay dos operandos (suma, resta, mayor o igual, etc.)
3. Ternarios cuando se involucran tres operandos
(solo hay un operador ternario en C++)

También se clasifican de acuerdo con la naturaleza de la operación que permiten realizar: aritméticos si ejecutan operaciones matemáticas; relacionales o de comparación cuya utilidad es comparar dos cantidades; lógicos si ejecutan operaciones cuyo resultado es un valor booleano; de asignación cuando asignan un valor a una variable; y de operaciones a nivel de *bit* si la operación en cuestión manipula uno o varios operandos mediante alguna operación a nivel binario.



A continuación, en las **tablas 1 a la 6**, se presenta la clasificación de los operadores de C++:

Tabla 1. Operadores aritméticos de C++

| Operador | Operación | Clase |
|----------|---------------------------------|---------|
| + | Suma, adición | Binario |
| - | Resta, sustracción | Binario |
| * | Multiplicación, producto | Binario |
| / | División, cociente | Binario |
| % | Modulo o residuo de la división | Binario |
| ++ | Incremento en 1 | Unario |
| -- | Decremento en 1 | Unario |

Tabla 2. Operadores relacionales o de comparación de C++

| Operador | Operación | Clase |
|----------|---|---------|
| == | Igualdad | Binario |
| != | Desigualdad | Binario |
| < | Menor (oper. izq. menor que oper. der.) | Binario |
| <= | Menor o igual | Binario |
| > | Mayor | Binario |
| >= | Mayor o igual | Binario |



Tabla 3. Operadores lógicos de C++

| Operador | Operación | Clase |
|----------|-----------------------|---------|
| && | AND lógica (booleana) | Binario |
| | OR lógica (booleana) | Binario |
| ^ | XOR lógica (booleana) | Binario |
| ! | NOT lógica (booleana) | Unario |

Tabla 4. Operaciones binarias a nivel de bit de C++

| Operador | Operación | Clase |
|----------|------------------------------------|---------|
| & | AND binaria | Binario |
| | OR binaria | Binario |
| ^ | XOR binaria | Binario |
| ~ | NOT (complemento) binaria | Unario |
| << | Corrimiento de bits a la izquierda | Binario |
| >> | Corrimiento de bits a la derecha | Binario |

Tabla 5. Operadores de asignación de

| Operador | Operación | Clase |
|----------|---|---------|
| = | Asignación (valor oper. der a variable oper.izq.) | Binario |
| += | Suma y asignación | Binario |
| -= | Resta y asignación | Binario |
| *= | Producto y asignación | Binario |
| /= | División y asignación | Binario |
| %= | Modulo y asignación | Binario |
| <<= | Corrimiento a izquierda y asignación | Binario |
| >>= | Corrimiento a derecha y asignación | Binario |



| | | |
|----|--------------------------|---------|
| &= | AND binaria y asignación | Binario |
| ^= | XOR binaria y asignación | Binario |
| = | OR binaria y asignación | Binario |

Tabla 6. Otros operadores de C++

| Operador | Operación | Clase |
|----------|---|----------|
| sizeof() | Tamaño en bytes del operando | Unario |
| * | Valor en la dirección de un puntero | Unario |
| + | Signo positivo | Binario |
| - | Signo negativo | Unario |
| . | Acceso a un miembro de estructura | Unario |
| & | Dirección en memoria, referencia | Unario |
| -> | Acceso a un miembro de puntero a estructura | Unario |
| () | Paréntesis en fórmula matemática. Conversión. | Unario |
| [] | Corchetes en fórm. mat. Acceso en arreglos. | Unario |
| :? | Operador ternario de comparación | Ternario |
| :: | Resolución de contexto (ámbito) | Binario |
| new | Reserva dinámica de memoria en el heap | Unario |
| delete | Liberación de memoria reservada con op. new | Unario |

Jerarquía (priorización) de operadores de C++

C++ define un sistema de prioridades o jerarquías para la correcta ejecución de operaciones que involucran varios operadores en una misma expresión. En primer lugar, todos los operadores unarios tienen mayor prioridad que los operadores binarios y el operador de menor prioridad es el único ternario.

En el caso de operaciones matemáticas (que solo involucran operadores de índole matemática) los operadores con mayor prioridad son los corchetes y paréntesis, seguidos por los operadores de producto, división y módulo, y dejando con la menor prioridad a los operadores de suma y resta.



Cuando dos o más operadores aritméticos de igual prioridad están en una misma operación, entonces se evalúan de izquierda a derecha. Es decir, se da mayor prioridad al operador que se encuentre más hacia la izquierda en la expresión, y así en lo sucesivo.

Los operadores relacionales o de comparación tienen, todos en conjunto, una prioridad inferior a los operadores aritméticos, es decir, primero se ejecuta cualquier (o todos) operador aritmético antes que cualquier operador relacional. Dentro del conjunto de los operadores relacionales primero se ejecutan los operadores de comparación menor, mayor, menor o igual y mayor o igual, y tienen menor prioridad los de igualdad y desigualdad. Cuando hay varios operadores relacionales de igual prioridad en una misma expresión, entonces se resuelve la prioridad de izquierda a derecha (en la expresión).

Los operadores de operaciones binarias a nivel de *bit* tienen menor prioridad que los operadores relacionales, a excepción del operador *not* binario que, por ser unario, tiene mayor prioridad que cualquier operador binario, y los operadores de corrimiento de *bit* que tienen una prioridad mayor a la de los operadores relacionales. De resto, el operador *and* tiene mayor prioridad que el operador *xor* y este a su vez mayor prioridad que el operador *or*. Cuando hay varios operadores binarios de igual prioridad en una misma expresión entonces se resuelve la prioridad de izquierda a derecha (en la expresión).

Los operadores lógicos son los siguientes, en el orden de prioridades, a excepción del operador *not* lógico que por ser unario tiene mayor prioridad que cualquier operador binario. El operador *and* lógico tiene mayor prioridad que el operador *or*. Cuando hay varios operadores lógicos de igual prioridad en una misma expresión entonces se resuelve la prioridad de izquierda a derecha (en la expresión).

La menor prioridad corresponde a los operadores de asignación, y es apenas lógico, ya que estos deben esperar que se ejecuten todas las operaciones de una expresión antes de asignar un resultado. Cuando hay varios operadores de asignación en una misma expresión entonces se resuelve la prioridad de derecha a izquierda (en la expresión).

La **tabla 7** resume los niveles de jerarquía de los operadores de C++:



Tabla 7. Jerarquía de operadores de C++ [1]

| Operador | Operación | Clase |
|----------|-------------------------------------|-------------|
| 1 | :: | Izq. a der. |
| 2 | ++ -- (post-inc/dec) | Izq. a der. |
| 2 | () | Izq. a der. |
| 2 | [] | Izq. a der. |
| 2 | . -> | Izq. a der. |
| 3 | ++ -- (pre-inc/dec) | Der. a izq. |
| 3 | ~ ! | Der. a izq. |
| 3 | + - (signos) | Der. a izq. |
| 3 | * & (punteros) | Der. a izq. |
| 3 | new delete | Der. a izq. |
| 3 | sizeof | Der. a izq. |
| 3 | (tipo) Conversión | Der. a izq. |
| 4 | . * -> * (acceso a puntero) | Izq. a der. |
| 5 | * / % | Izq. a der. |
| 6 | + - | Izq. a der. |
| 7 | << >> | Izq. a der. |
| 8 | < > <= >= | Izq. a der. |
| 9 | == != | Izq. a der. |
| 10 | & | Izq. a der. |
| 11 | ^ | Izq. a der. |
| 12 | | Izq. a der. |
| 13 | && | Izq. a der. |
| 14 | | Izq. a der. |
| 15 | = *= /= %= += -= >>= <<= &= = = ^= | Der. a izq. |
| 15 | ?: | Der. a izq. |

Nota importante

El operador módulo (%), los operadores binarios a nivel de bit (~, <<, >>, &, |, ^) y los operadores para las asignaciones con operadores binarios (=%, >>=, <<=, &=, |=, ^=), no están implementados para los tipos de dato de punto flotante float y double, luego estos operadores no tienen sentido al usarlos en operandos de esos tipos.

**Ejemplos:**

1. ¿Cuál es la salida en pantalla del siguiente programa en C++?

```
int main(){
    bool ints = sizeof(long) >= sizeof(int) && sizeof(int) >= sizeof(short);
    bool floats = sizeof(double) >= sizeof(float);
    bool chars = sizeof(char) == 1;
    int r = ints && floats && chars;
    std::cout << r << std::endl;
}
```

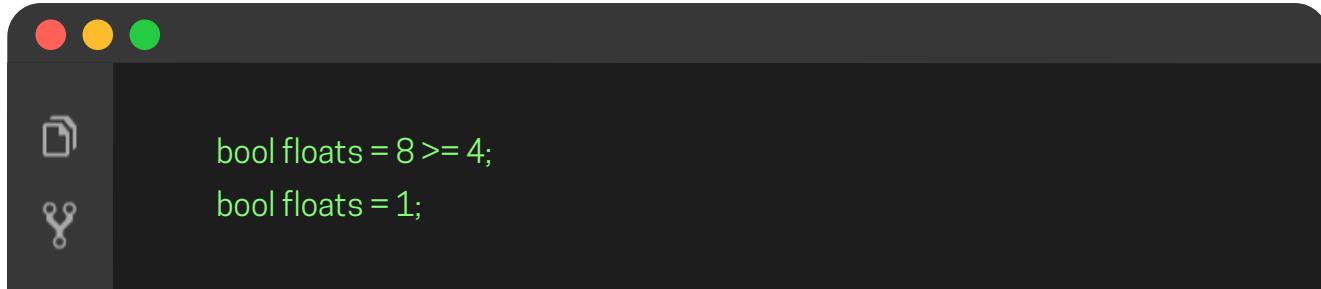
Este programa imprime un número uno (1) en su salida en pantalla. Esto pasa porque en las declaraciones de las variables ints, floats y chars el operador sizeof() es el de mayor jerarquía y por tanto son las operaciones que lo involucran lo primero en resolverse, luego sizeof(long) = 8, sizeof(int) = 4, sizeof(short) = 2, sizeof(double) = 8, sizeof(float) = 4 y sizeof(char) = 1.

Luego, para la variable ints el operador que sigue en la jerarquía es el operador de comparación `>=`, entonces se resuelven las comparaciones con los valores obtenidos de los operadores sizeof(). Y el operador `&&` es el último en la jerarquía y se resuelve para los operandos obtenidos en las operaciones anteriores. Al final esto es lo que sucede en orden de jerarquía de los operadores:

```
bool ints = 8 >= 4 && 4 >= 2;
bool ints = 1 && 1;
bool ints = 1;
```



Para la variable floats se resuelve el operador `>=`.



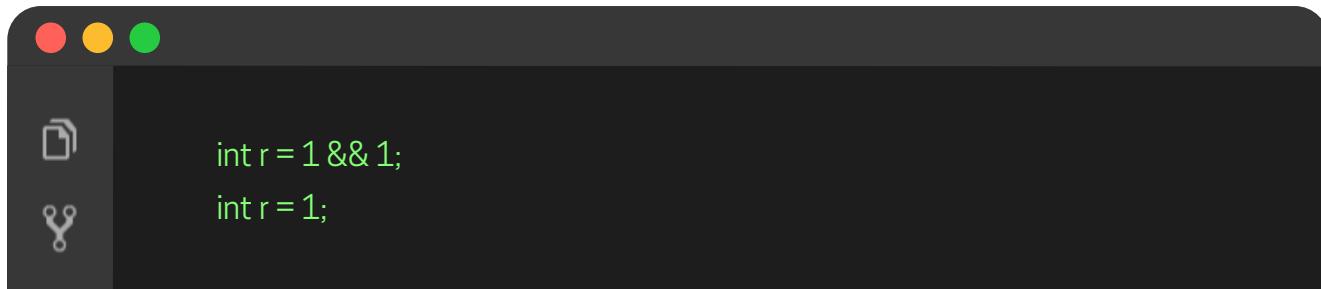
```
bool floats = 8 >= 4;
bool floats = 1;
```

Para la variable chars se resuelve el operador `==`.



```
bool chars = 1 == 1;
bool chars = 1;
```

En la operación para calcular el valor que se asigna a la variable r se involucran operadores con igual jerarquía, se observa en la tabla 7 que para el operador `&&` se resuelve a igual jerarquía de izquierda a derecha, quedando la operación en el siguiente orden:



```
int r = 1 && 1;
int r = 1;
```



2. ¿Cuál será el valor asignado a la variable z en el siguiente programa?

```
int main(){
    int y=10;
    int z=1;
    z *= ++y + 5;
    std::cout << z << std::endl;
}
```

De acuerdo con la tabla 7, los operadores de asignación son los operadores junto con el operador ternario que tienen el nivel más bajo de jerarquía y por tanto son siempre los últimos en operar en cualquier operación en la que estén involucrados, en este caso el operador producto y asignación *=. Luego el operador de más alta jerarquía es el operador pre-incremento, ++, dejando en segundo lugar al operador adición o suma. Al final la operación se resuelve en el siguiente orden:

```
z *= 11 + 5;
z *= 16;
z = 16;
```



3. ¿Cuál será el valor asignado a la variable x en el siguiente programa?

```
int main(){
    int a = 15;
    int b = 3;
    int c = 19;
    int d = 10;
    int x = a*b-c%d;
    std::cout << x << std::endl;
}
```

En esta oportunidad los operadores * y % tienen ambos la misma prioridad que es superior a la del operador -. Se resuelven las operaciones en ambos lados del operador -, resolviendo primero la del operador * ya que entre este operador y el operador % la prioridad se define de izquierda a derecha. En segundo lugar, se resuelve la operación del operador % y finalmente la resta. A continuación, el detalle del orden de las operaciones.

```
int x = 15*3-19%10;
int x = 45-19%10;
int x = 45-9;
int x = 36;
```



Referencias

1. C Plus Plus. (2020). Operators. [En línea]. Disponible en <http://www.cplusplus.com/doc/tutorial/operators/>