1.5 — A first look at operators

BY ALEX ON JUNE 6TH, 2007 | LAST MODIFIED BY ALEX ON MAY 14TH, 2017

Revisiting expressions

In lesson **1.1 -- Structure of a program**, we had defined an expression as “A mathematical entity that evaluates to a value”. However, the term *mathematical entity* is somewhat vague. More precisely, an **expression** is a combination of literals, variables, functions, and operators that evaluates to a value.

Literals

A **literal** is a fixed value that has been inserted (hardcoded) directly into the source code, such as 5, or `3.14159`. Literals always evaluate to themselves. Here’s an example that uses literals:

```
#include <iostream>

int main()
{
    int x = 2; // x is a variable, 2 is a literal
    std::cout << 3 + 4; // 3 + 4 is an expression, 3 and 4 are literals
    std::cout << "Hello, world!"; // "Hello, world" is a literal too

    return 0;
}
```

Literals, variables, and function calls that return values are all known as operands. **Operands** supply the data that the expression works with. We just introduced literals, which evaluate to themselves. Variables evaluate to the values they hold. Functions evaluate to produce a value of the function’s return type (unless the return type is void).

Operators

The last piece of the expressions puzzle is operators. **Operators** tell the expression how to combine one or more operands to produce a new result. For example, in the expression “3 + 4”, the + is the plus operator. The + operator tells how to combine the operands 3 and 4 to produce a new value (7).

You are likely already quite familiar with standard arithmetic operators from common usage in math, including addition (+), subtraction (-), multiplication (*), and division (/). Assignment (=) is an operator as well. Some operators use more than one symbol, such as the equality operator (==), which allows us to compare two values to see if they are equal.

Note: One of the most common mistakes that new programmers make is to confuse the assignment operator (=) with the equality operator (==). Assignment (=) is used to assign a value to a variable. Equality (==) is used to test whether two operands are equal in value. We’ll cover the equality operator in more detail later.

Operators come in three types:

Unary operators act on one operand. An example of a unary operator is the - operator. In the expression `-5`, the - operator is only being applied to one operand (5) to produce a new value (-5).

Binary operators act on two operands (known as left and right). An example of a binary operator is the + operator. In the expression `3 + 4`, the + operator is working with a left operand (3) and a right operand (4) to produce a new value (7).

Ternary operators act on three operands. There is only one of these in C++, which we’ll cover later.

Also note that some operators have more than one meaning. For example, the - operator has two contexts. It can be used in unary form to invert a number’s sign (eg. to convert 5 to -5, or vice versa), or it can be used in binary form to do arithmetic subtraction (eg. 4 - 3).

Conclusion

This is just the tip of the iceberg in terms of operators. We will take an in-depth look at operators in more detail in a future section.