# 1.10 — Introduction to expressions

ALEX SEPTEMBER 16, 2022

## Expressions

Consider the following series of statements:

```
// five() is a function that returns the value 5
int five()
{
    return 5;
}

int main()
{
    int a{ 2 };             // initialize variable a with literal value 2
    int b{ 2 + 3 };         // initialize variable b with computed value 5
    int c{ (2 * 3) + 4 };   // initialize variable c with computed value 10
    int d{ b };             // initialize variable d with variable value 5
    int e{ five() };        // initialize variable e with function return value 5

    return 0;
}
```

Each of these statements defines a new variable and initializes it with a value. Note that the initializers shown above make use of a variety of different constructs: literals, variables, operators, and function calls. Somehow, C++ is converting all of these different things into a single value that can then be used as the initialization value for the variable.

What do all of these have in common? They make use of an expression.

An **expression** is a combination of literals, variables, operators, and function calls that calculates a single value. The process of executing an expression is called **evaluation**, and the single value produced is called the **result** of the expression.

### Related content

While most expressions are used to calculate a value, expressions can also identify an object (which can be evaluated to get the value held by the object) or a function (which can be called to get the value returned by the function). We talk more about this in lesson 9.2 -- Value categories (lvalues and rvalues) (https://www.learncpp.com/cpp-tutorial/value-categories-lvalues-and-rvalues/).

For now, we'll assume all expressions calculate values.

When an expression is evaluated, each of the terms inside the expression are evaluated, until a single value remains. Here are some examples of different kinds of expressions, with comments indicating how they evaluate:

```
2               // 2 is a literal that evaluates to value 2
"Hello world!"  // "Hello world!" is a literal that evaluates to text "Hello world!"
x               // x is a variable that evaluates to the value of x
2 + 3           // operator+ combines values 2 and 3 to produce value 5
x = 2 + 3       // 2 + 3 evaluates to value 5, which is then assigned to variable x
std::cout << x  // x evaluates to the value of x, which is then printed to the console
five()          // evaluates to the return value of function five()
```

As you can see, literals evaluate to their own values. Variables evaluate to the value of the variable. We haven't covered function calls yet, but in the context of an expression, function calls evaluate to whatever value the function returns. And operators (such as operator+) let us combine multiple values together to produce a new value.

Note that expressions do not end in a semicolon, and cannot be compiled by themselves. For example, if you were to try compiling the expression `x = 5`, your compiler would complain (probably about a missing semicolon). Rather, expressions are always evaluated as part of statements.

For example, take this statement:

```
int x{ 2 + 3 }; // 2 + 3 is an expression that has no semicolon -- the semicolon is at the end of the statement containing the expression
```

If you were to break this statement down into its syntax, it would look like this:

```
type identifier { expression };
```

Type could be any valid type (we chose int). Identifier could be any valid name (we chose x). And expression could be any valid expression (we chose `2 + 3`, which uses two literals and an operator).

### Key insight

Wherever you can use a single value in C++, you can use a value-producing expression instead, and the expression will be evaluated to produce a single value.

## Expression statements

Certain expressions (like `x = 5`) are useful by themselves (in this case, to assign the value `5` to the variable `x`). However, we mentioned above that expressions cannot be executed by themselves -- they must exist as part of a statement. So how can we use such expressions?

Fortunately, it's easy to convert any expression into an equivalent statement (called an expression statement). An **expression statement** is a statement that consists of an expression followed by a semicolon. When the statement is executed, the expression will be evaluated.

Thus, we can take any expression (such as `x = 5`), and turn it into an expression statement (`x = 5;`) that will compile.

The result of an expression statement is discarded before the next statement is executed.

### For advanced readers

An expression whose result is discarded is called a **discarded-value expression**). Expression statements are by far the most common type of discarded-value expressions.

Other discarded-value expressions include the left operand of the comma operator, and any expression that is cast to type `void`.

## Useless expression statements

We can also make expression statements that compile but have no effect. For example, the expression statement (`2 * 3;`) is a expression statement whose expression evaluates to the result value of 6, which is then discarded. While syntactically valid, such expression statements are useless. Some compilers (such as gcc and Clang) will produce warnings if they can detect that an expression statement is useless.

## Quiz time

**Question #1**

What is the difference between a statement and an expression?

Show Solution (javascript:void(0))

**Question #2**

Indicate whether each of the following lines are statements that do not contain expressions, statements that contain expressions, or are expression statements.

a)

```
int x;
```

Show Solution (javascript:void(0))

b)

```
int x = 5;
```

Show Solution (javascript:void(0))

c)

```
x = 5;
```

Show Solution (javascript:void(0))

d)

```
foo(); // foo is a function
```

Show Solution (javascript:void(0))

e) Extra credit:

```
std::cout << x; // Hint: operator<< is a binary operator.
```

Show Solution (javascript:void(0))

---

**Question #3**

Determine what values the following program outputs. Do not compile this program. Just work through it line by line in your head.

```
#include <iostream>

int main()
{
    std::cout << 2 + 3 << '\n';

    int x{ 6 };
    int y{ x - 2 };
    std::cout << y << '\n';

    int z{ };
    z = x;
    std::cout << z - x << '\n';

    return 0;
}
```

Show Solution (javascript:void(0))

---


**Next lesson**
1.11 Developing your first program

**Back to table of contents**

**Previous lesson**
1.9 Introduction to literals and operators



Leave a comment...

Name*

Email*

Find a mistake? Leave a comment!

Notify me about replies:

POST COMMENT

Avatars from https://gravatar.com/ are connected to your

provided email address.

205 Comments

Newest