1.8 — Introduction to literals and operators

BY ALEX ON FEBRUARY 1ST, 2019 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 31ST, 2020

Literals

Consider the following two statements:

```
std::cout << "Hello world!";
int x{ 5 };
```

What are *"Hello world!"* and *5*? They are literals. A **literal** (also known as a **literal constant**) is a fixed value that has been inserted directly into the source code.

Literals and variables both have a value (and a type). However, the value of a literal is fixed and can't be changed (hence it being called a constant), whereas the value of a variable can be changed through initialization and assignment.

Operators

In mathematics, an **operation** is a mathematical calculation involving zero or more input values (called **operands**) that produces a new value (called an output value). The specific operation to be performed is denoted by a construct (typically a symbol or pair of symbols) called an **operator**.

For example, as children we all learn that *2 + 3* equals *5*. In this case, the literals *2* and *3* are the operands, and the symbol + is the operator that tells us to apply mathematical addition on the operands to produce the new value *5*.

Author's note

For reasons that will become clear when we discuss operators in more detail, for operators that are symbols, it is common nomenclature to append the operator's symbol to the word *operator*.

For example, the plus operator would be called *operator*+, and the extraction operator would be called *operator*>>.

You are likely already quite familiar with standard arithmetic operators from common usage in mathematics, including addition (+), subtraction (-), multiplication (*), and division (/). In C++, assignment (=) is an operator as well, as are << (insertion) and >> (extraction). 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. There are also a number of operators that are words (e.g. new, delete, and throw).

The number of operands that an operator takes as input is called the operator's *arity* (almost nobody knows what this word means, so don't drop it in a conversation and expect anybody to have any idea what you're talking about). Operators in C++ come in three different *arities*:

Unary operators act on one operand. An example of a unary operator is the - *operator*. For example, given `-5`, *operator-* takes literal operand *5* and flips its sign to produce new output value *-5*.

Binary operators act on two operands (known as *left* and *right*). An example of a binary operator is the + *operator*. For example, given `3 + 4`, *operator+* takes the left operand (3) and the right operand (4) and applies mathematical addition to produce new output value *7*. The insertion (<<) and extraction (>>) operators are binary operators, taking std::cout or std::cin on the left side, and the item to output or variable to input to on the right side.

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

Note that some operators have more than one meaning depending on how they are used. For example, *operator-* has two contexts. It can be used in unary form to invert a number's sign (e.g. to convert 5 to -5, or vice versa), or it can be used in binary form to do subtraction (e.g. 4 - 3).

Chaining operators

Operators can be chained together such that the output of one operator can be used as the input for another operator. For example, given the following: *2 * 3 + 4*, the multiplication operator goes first, and converts left operand *2* and right operand *3* into new value *6* (which becomes the left operand for the plus operator). Next, the plus operator executes, and converts left operand *6* and right operand *4* into new value 10.

We'll talk more about the order in which operators execute when we do a deep dive into the topic of operators. For now, it's enough to know that the arithmetic operators execute in the same order as they do in standard mathematics: Parenthesis first, then Exponents, then Multiplication & Division, then Addition & Subtraction. This ordering is sometimes abbreviated *PEMDAS*, or the mnemonic "Please Excuse My Dear Aunt Sally".

Quiz time

Question #1

For each of the following, indicate what output they produce:

a)

```
std::cout << 3 + 4;
```

Show Solution

b)

```
std::cout << 3 + 4 - 5;
```

Show Solution

c)

```
std::cout << 2 + 3 * 4;
```

Show Solution

1.9 -- Introduction to expressions

Index

1.7 -- Keywords and naming identifiers

C++ TUTORIAL | PRINT THIS POST

21 comments to 1.8 — Introduction to literals and operators

Vitaliy Sh.
January 15, 2020 at 9:10 pm · Reply

in <p>'s "You are likely already..."; "Binary ..."
(+), (-), (*), (/), (=), (==), (<<), (>>).

** <code>?

Jim
December 12, 2019 at 9:25 am · Reply

I'm not sure if this is a dumb question or not.

But aren't the brackets and braces in C++ also considered operators? Including ( ), { }, [ ].

How are these signs treated in C++ $, %, ^,. ?
I'm sure that you can always output the words dollar and percent instead. The up-arrow (to the power of) could be extended out with multiplication> But I'm sure programmers have figured out how to do this in C++ by now.

nascardriver
December 13, 2019 at 1:43 am · Reply

(), [], % and ^ are operators. ^ is a bitwise xor, not a power operation.
{} isn't an operator. It's used for scoping and initialization.
$ has no use in C++.

Marie Bethell
July 26, 2019 at 8:42 am · Reply

Hi, I think there's a typo; "Consider the following two statement: ". There should be an s on the end of statement.

geekersquad

July 2, 2019 at 4:45 am · Reply

what is operand?

nascardriver
July 2, 2019 at 7:45 am · Reply

[CODE]
1 + 2
[CODE]
1 and 2 are operands
+ is an operator

Bella
June 26, 2019 at 5:24 am · Reply

How can you identify the difference between literals and variables?
I mean isn't a literal printed out the exact same way as a variable?
How do we know the difference. You wrote here...

"Literals and variables both have a value (and a type). However, the value of a literal is fixed and can't be changed (hence - called a constant), whereas the value of a variable can be changed through initialization and assignment."

std::cout << "Hello world!";
int x{ 5 }; // you said "Hello world!" and 5 are literals.

OKAY... you're saying "5" is a literal even though it has underwent the procedure of initialization by using { }.
The "hello world" text, I understand can be viewed as both a literal and variable(data)
but the - 'int x {5}' - I don't understand.
Is identifying the difference between literal and variable/ data important? As I'm struggling to understand this topic.

nascardriver
June 26, 2019 at 9:24 am · Reply

A literal is the value you write down in your code. 5 and "Hello world!" are literals, what you do with them doesn't matter. `x` is a variable.
If you don't understand it, you'll probably get the hang of it as you go along.

Nice
April 24, 2019 at 5:17 pm · Reply

Thank you so much for this, man. It's really helpful and easy to understand

km
March 26, 2019 at 1:43 pm · Reply

Does this not turn 5 into a variable?? int x { 5 }
I can change the value of x after this which means it's not a constant right?

nascardriver

March 27, 2019 at 8:15 am · Reply

It sets the value of @x to 5. You can change to value of @x, but you can never change 5.

Rutvik Kapade
February 16, 2019 at 8:16 am · Reply

dude i really love you ,this is like a online c++ bible for me and its free and really easy to understand ,god bless you. I am a student and i dont earn ,but this is helping me alot. =) thanks again!

Andu
February 10, 2019 at 9:19 am · Reply

Hi,

"For example, the plus operator would be called operator+, and the extraction operator would be called operator<<." Should be operator>> !?

Alex
February 10, 2019 at 1:17 pm · Reply

Fixed. Thanks!

the_G
February 7, 2019 at 9:43 am · Reply

This is sooooo useful Thanks SOOO much whoever made this <3

Alex
February 7, 2019 at 9:05 pm · Reply

You're welcome. Thanks for visiting.

Henrique dos Santos
February 5, 2019 at 10:27 am · Reply

It's PEMDAS, not PEDMAS according to https://en.wikipedia.org/wiki/Order_of_operations

Alex
February 7, 2019 at 6:18 am · Reply

Acronym fixed! Thanks for pointing out the omission.

Walter
April 15, 2019 at 5:50 am · Reply

The acronym works both ways, and I've heard PEDMAS myself. I think it might be a USA vs UK thing?

Alex
April 16, 2019 at 9:58 pm · Reply

Yeah, I learned PEDMAS, but I think PEMDAS is more common and better because "Please Excuse My Dear Aunt Sally" is an easy mnemonic.

Juglugs
October 18, 2019 at 12:01 pm · Reply

PEDMAS, PEMDAS and BODMAS all work. Multiplication & division are actually done in the same order, similar to addition and subtraction.