Mr. Giansante

C++ Programming

Math with C++

August 2018

Math with C++

Source:
Lesson 12 - Mathematical functions in C++ - The cmath library
by David Capka
https://www.ict.social/cplusplus/course/basics/
mathematical-functions-in-cplusplus-the-cmath-library

cmath Standard Library

The `cmath` standard library provides a variety of functions for solving common math problems.

```
#include <cmath>
```

Basic Mathematical Operators

You can use the following mathematical operators when writing equations:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder from Division (modulo)

Order of Operations

C++ follows the standard rules for the order in which it evaluates operations. This is sometimes refered to as "**BEDMAS**".

Brackets
Exponents

Division
Multiplication } in order they appear

Addition
Subtraction } in order they appear

Example - Converting Temperature

The formula to convert Fahrenheit (°F) to Celsius (°C) is:

$$°C = (°F - 32) \times \frac{5}{9}$$

In C++, we could declare 2 variables and use the following formula:

```
C = ( F - 32 ) * ( 5.0 / 9.0 );
```

Division

Programming languages often differ in how they perform the division of numbers. You need to be aware of these issues to avoid surprises

Example.

```
int a = 5 / 2;
double b = 5 / 2;
double c = 5.0 / 2;
double d = 5 / 2.0;
double e = 5.0 / 2.0;
int f = 5 / 2.0;

cout << "a=" << a << ", b=" << b << ", 
c=" << c << ", d=" << d << ", e=" << e 
<< ", f=" << f << endl;
```

We divide 5/2 several times in the code. Mathematically, it's 2.5. Nonetheless, the results will not be the same in all cases. Can you guess what we'll get in each case?

The program output will be as follows:

```
a=2, b=2, c=2.5, d=2.5, e=2.5, f=2
```

We see the result of this division is sometimes decimal and sometimes whole.

The data type of the variable we're assigning the result to is not all that matters. What matters most is the data type of the numbers we divide by. If one of the numbers is decimal, the outcome will always be a decimal number.

The division of 2 integers always returns an integer.

Keep in mind that if you compute the average and want a decimal result, at least one variable must be cast to double.

```
int sum = 10;
int count = 4;

double average = (double)sum / (double)count;
```

Math with C++

Remainder from Division

`%` (modulo) returns the remainder after dividing integers. This can be useful, for example if you want to determine if a number is even (ie. divisible by 2).

```
     if(myNum % 2 = 0)
          { cout << "Number is even"; }
```

abs()

`abs()` returns the absolute value of its parameter.

pow()

`pow()` takes two input parameters. The first is the base of the power and the second is the exponent.

Example. Calculate 2^3

```
     cout << pow(2, 3);
```

sqrt()

`Sqrt()` is an abbreviation of SQuare RooT, which returns the square root of the number given as a double.

General Root

C++ lacks any general root function. However, we can use the rules of exponents to solve this problem.

Example. Calculate the 3rd root of 8.

Recall that $\sqrt[3]{8} = 8^{1/3}$

Therefore, we can write:

```
    cout << pow(8, (1.0/3.0));
```

Note: It is very important to write at least one number with a decimal point when we are dividing, otherwise, C++ will assume that we want it to apply whole-number division, and the result would have been $8^0 = 1$ in this case.

round(), ceil(), floor(), and trunc()

All these functions are related to rounding and they all accept a parameter of the double type. Their return value is also of the double type.

`round()`	takes a decimal number as a parameter and returns the number, rounded as a double data type. It rounds in the same way we learned in school (anything over 0.5 is rounded upwards, otherwise the number is rounded downwards).
`ceil()`	rounds upwards no matter what
`floor()`	rounds downwards no matter what
`trunc()`	cuts the decimal part off and leaves the whole number part intact (does not round it whatsoever).

If you think that `floor()` and `trunc()` do the same thing, think again! They behave differently for negative numbers. `floor()` rounds negative numbers down to an even "more negative" number, `trunc()` always rounds to zero when the input is negative.

sin(), cos(), tan()

Classic trigonometric functions, all take an angle which has to be entered in radians (not degrees).

To convert degrees to radians we multiply them by `(M_PI / 180)`

acos(), asin(), atan()

Inverse trigonometric functions, which return the original angle according to the trigonometric value. They're the inverse functions for `sin()`, `cos()`, and `tan()`.

The parameter is a function value and the returned value is the original angle in radians (returned as a double).

If we wanted to get an angle in degrees, we'd have to divide the radians by `(180 / M_PI)`