1.2 — Comments

BY ALEX ON MAY 30TH, 2007 | LAST MODIFIED BY ALEX ON OCTOBER 28TH, 2016

Types of comments

A **comment** is a line (or multiple lines) of text that are inserted into the source code to explain what the code is doing. In C++ there are two kinds of comments.

The `//` symbol begins a C++ single-line comment, which tells the compiler to ignore everything to the end of the line. For example:

```
std::cout << "Hello world!" << std::endl; // Everything from here to the right is ignored.
```

Typically, the single line comment is used to make a quick comment about a single line of code.

```
std::cout << "Hello world!" << std::endl; // cout and endl live in the iostream library
std::cout << "It is very nice to meet you!" << std::endl; // these comments make the code hard to read
std::cout << "Yeah!" << std::endl; // especially when lines are different lengths
```

Having comments to the right of a line can make both the code and the comment hard to read, particularly if the line is long. Consequently, the // comment is often placed above the line it is commenting.

```
// cout and endl live in the iostream library
std::cout << "Hello world!" << std::endl;

// this is much easier to read
std::cout << "It is very nice to meet you!" << std::endl;

// don't you think so?
std::cout << "Yeah!" << std::endl;
```

The `/*` and `*/` pair of symbols denotes a C-style multi-line comment. Everything in between the symbols is ignored.

```
/* This is a multi-line comment.
   This line will be ignored.
   So will this one. */
```

Since everything between the symbols is ignored, you will sometimes see programmers “beautify” their multiline comments:

```
/* This is a multi-line comment.
 * the matching asterisks to the left
 * can make this easier to read
 */
```

Multi-line style comments do not nest. Consequently, the following will have unexpected results:

```
/* This is a multi-line /* comment */ this is not inside the comment */
// The above comment ends at the first */, not the second */
```

Rule: Never nest comments inside of other comments.

Proper use of comments

Typically, comments should be used for three things. For a given library, program, or function, comments are best used to describe *what* the library, program, or function, does. For example:

```
// This program calculate the student's final grade based on his test and homework scores.
```

```
// This function uses newton's method to approximate the root of a given equation.
```

```
// The following lines generate a random item based on rarity, level, and a weight factor.
```

All of these comments give the reader a good idea of what the program is trying to accomplish without having to look at the actual code. The user (possibly someone else, or you if you’re trying to reuse code you’ve already previously written) can tell at a glance whether the code is relevant to what he or she is trying to accomplish. This is particularly important when working as part of a team, where not everybody will be familiar with all of the code.

Second, within a library, program, or function described above, comments can be used to describe *how* the code is going to accomplish its goal.

```
/* To calculate the final grade, we sum all the weighted midterm and homework scores
    and then divide by the number of scores to assign a percentage.  This percentage is
    used to calculate a letter grade. */
```

```
// To generate a random item, we're going to do the following:
// 1) Put all of the items of the desired rarity on a list
// 2) Calculate a probability for each item based on level and weight factor
// 3) Choose a random number
// 4) Figure out which item that random number corresponds to
// 5) Return the appropriate item
```

These comments give the user an idea of how the code is going to accomplish it's goal without going into too much detail.

At the statement level, comments should be used to describe *why* the code is doing something. A bad statement comment explains *what* the code is doing. If you ever write code that is so complex that needs a comment to explain *what* a statement is doing, you probably need to rewrite your statement, not comment it.

Here are some examples of bad line comments and good statement comments.

Bad comment:

```
// Set sight range to 0
sight = 0;
```

(yes, we already can see that sight is being set to 0 by looking at the statement)

Good comment:

```
// The player just drank a potion of blindness and can not see anything
sight = 0;
```

(now we know WHY the player's sight is being set to 0)

Bad comment:

```
// Calculate the cost of the items
cost = items / 2 * storePrice;
```

(yes, we can see that this is a cost calculation, but why is items divided by 2?)

Good comment:

```
// We need to divide items by 2 here because they are bought in pairs
cost = items / 2 * storePrice;
```

(now we know!)

Programmers often have to make a tough decision between solving a problem one way, or solving it another way. Comments are a great way to remind yourself (or tell somebody else) the reason you made one decision instead of another.

Good comments:

```
// We decided to use a linked list instead of an array because
// arrays do insertion too slowly.
```

```
// We're going to use newton's method to find the root of a number because
// there is no deterministic way to solve these equations.
```

Finally, comments should be written in a way that makes sense to someone who has no idea what the code does. It is often the case that a programmer will say "It's obvious what this does! There's no way I'll forget about this". Guess what? It's not obvious, and you will be amazed how quickly you forget. ☺ You (or someone else) will thank you later for writing down the what, how, and why of your code in human language. Reading individual lines of code is easy. Understanding what goal they are meant to accomplish is not.

To summarize:

- At the library, program, or function level, describe *what*
- Inside the library, program, or function, describe *how*
- At the statement level, describe *why*.

Commenting out code

Converting one or more lines of code into a comment is called **commenting out** your code. This provides a convenient way to (temporarily) exclude parts of your code from being included in your compiled program.

To comment out a single line of code, simply use the // style comment to turn a line of code into a comment temporarily:

Uncommented out:

```
    std::cout << 1;
```

Commented out:

```
//    std::cout << 1;
```

To comment out a block of code, use // on multiple lines of code, or the /* */ style comment to turn the block of code into a comment temporarily.

Uncommented out:

```
    std::cout << 1;
    std::cout << 2;
    std::cout << 3;
```

Commented out:

```
//    std::cout << 1;
//    std::cout << 2;
//    std::cout << 3;
```

or

```
/*
    std::cout << 1;
    std::cout << 2;
    std::cout << 3;
*/
```

There are quite a few reasons you might want to do this:

1) You’re working on a new piece of code that won’t compile yet, and you need to run the program. The compiler won’t let you run if there are compiler errors. Commenting out the code that won’t compile will allow the program to compile so you can run it. When you’re ready, you can uncomment the code, and continue working on it.

2) You’ve written code that compiles but doesn’t work correctly, and you don’t have time to fix it until later. Commenting out the broken code will ensure the broken code doesn’t execute and cause problems until you can fix it.

3) To find the source of an error. If a program isn’t producing the desired results (or is crashing), it can be useful in some cases to disable parts of your code to see if you can isolate what’s causing it to not work correctly. If you comment out one or more lines of code, and your program starts working as expected (or stops crashing), odds are whatever you last commented out was part of the problem. You can then investigate why those lines of code are causing the problem.

4) You want to replace one piece of code with another piece of code. Instead of just deleting the original code, you can comment it out and leave it there for reference until you’re sure your new code works properly. Once you are sure your new code is working, you can remove the old commented out code. If you can’t get your new code to work, you can always delete the new code and uncomment the old code to revert back to what you had before.

Commenting out code is a common thing to do while developing, so many IDEs provide support for commenting out a highlighted section of code. How you access this functionality varies by IDE (in Visual Studio, you can find it in the Edit menu under Edit->Advanced->Comment Selection/Uncomment Selection).