1.2 — Comments

BY ALEX ON MAY 30TH, 2007 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 5TH, 2020

A **comment** is a programmer-readable note that is inserted directly into the source code of the program. Comments are ignored by the compiler and are for the programmer's use only.

In C++ there are two different styles of comments, both of which serve the same purpose: to help programmers document the code in some way.

Single-line comments

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

```
1 std::cout << "Hello world!"; // Everything from here to the end of the line is ignored
```

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

```
1 std::cout << "Hello world!\n"; // std::cout lives in the iostream library
2 std::cout << "It is very nice to meet you!\n"; // these comments make the code hard to read
3 std::cout << "Yeah!\n"; // 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. If the lines are fairly short, the comments can simply be aligned (usually to a tab stop), like so:

```
1 std::cout << "Hello world!\n";                // std::cout lives in the iostream library
2 std::cout << "It is very nice to meet you!\n"; // this is much easier to read
3 std::cout << "Yeah!\n";                       // don't you think so?
```

However, if the lines are long, placing comments to the right can make your lines really long. In that case, single-line comments are often placed above the line it is commenting:

```
1 // std::cout lives in the iostream library
2 std::cout << "Hello world!\n";
3
4 // this is much easier to read
5 std::cout << "It is very nice to meet you!\n";
6
7 // don't you think so?
8 std::cout << "Yeah!\n";
```

Author's note

The statements above represent one of our first encounters with snippets of code. Because snippets aren't full programs, they aren't able to be compiled by themselves. Rather, they exist to demonstrate specific concepts in a concise manner.

If you would like to compile a snippet, you'll need to turn it into a full program in order for it to compile. Typically, that program will look something like this:

```
1 #include <iostream>
2
3 int main()
4 {
5     // Replace this line with the snippet of code you'd like to compile
```

```
6
7       return 0;
8   }
```

Multi-line comments

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

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

Since everything between the symbols is ignored, you will sometimes see programmers "beautify" their multi-line comments:

```
1   /* This is a multi-line comment.
2    * the matching asterisks to the left
3    * can make this easier to read
4    */
```

Multi-line style comments can not be nested. Consequently, the following will have unexpected results:

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

When the compiler tries to compile this, it will ignore everything from the first /* to the first */. Since "*this is not inside the comment* */" is not considered part of the comment, the compiler will try to compile it. That will inevitably result in a compile error.

This is one place where using a syntax highlighter can be really useful, as the different coloring for comment should make clear what's considered part of the comment vs not.

Warning

Don't use multi-line comments inside other multi-line comments. Wrapping single-line comments inside a multi-line comment is okay.

Proper use of comments

Typically, comments should be used for three things. First, for a given library, program, or function, comments are best used to describe *what* the library, program, or function, does. These are typically placed at the top of the file or library, or immediately preceding the function. For example:

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

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

```
1   // 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 library, program, or function 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 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, which 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 its goal without having to understand what each individual line of code does.

Third, 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;
```

Reason: 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;
```

Reason: Now we know why the player's sight is being set to 0

Bad comment:

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

Reason: We can see that this is a cost calculation, but why is quantity multiplied by 2?

Good comment:

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

Reason: Now we know why this formula makes sense.

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.

Best practice

Comment your code liberally, and write your comments as if speaking to someone who has no idea what the code does. Don't assume you'll remember why you made specific choices.

Author's note

Throughout the rest of this tutorial series, we'll use comments inside code blocks to draw your attention to specific things, or help illustrate how things work (while ensuring the programs still compile). Astute readers will note that by the above standards, most of these comments are horrible. :) As you read through the rest of the tutorials, keep in mind that the comments are serving an intentional educational purpose, not trying to demonstrate what good comments look like.

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:

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

Commented out:

```
1 //    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:

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

Commented out:

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

or

```
1 /*
2     std::cout << 1;
3     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 compile the code 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 new 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 sometimes be useful 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.

For Visual Studio users

You can comment or uncomment a selection via Edit menu > Advanced > Comment Selection (or Uncomment Selection).

For Code::Blocks users

You can comment or uncomment a selection via Edit menu > Comment (or Uncomment, or Toggle comment, or any of the other comment tools).

Tip

If you always use single line comments for your normal comments, then you can always use multi-line comments to comment out your code without conflict. If you use multi-line comments to document your code, then commenting-out code using comments can become more challenging.

If you do need to comment out a code block that contains multi-line comments, you can also consider using the `#if 0` preprocessor directive, which we discuss in lesson **2.10 -- Introduction to the preprocessor**.

Summary

- At the library, program, or function level, use comments to describe *what*.
- Inside the library, program, or function, use comments to describe *how*.

- At the statement level, use comments to describe *why*.

1.3 -- Introduction to variables

Index

1.1 -- Statements and the structure of a program

C++ TUTORIAL | PRINT THIS POST

134 comments to 1.2 — Comments

« Older Comments 1 2 3

Vitaliy Sh.
January 3, 2020 at 9:47 am · Reply

List of possible typos (and some questions):

"the comments can simply be aligned (usually to a tab stop), like so:"
** aligned by spaces? If user changes tab width... tab stop will shift as well?

"The statements above represent our first encounters with snippets of code."
** Is code in 0.7 and 0.8 (cin's .clear(), ignore(), get()) isn't a snippet because it isn't an concept (abstract idea)?

"Since this is not inside the comment */ is not considered part of the comment, the compiler will try to compile it."
** ``Since "this is ... */" is not ...`` -- enhance readability by "" ?

"// This function uses newton's method to approximate the root of a given equation."
** ``Newton's`` (one below is capitalized)?

"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."
** ``... to what this person is trying ...``
** https://stallman.org/articles/genderless-pronouns.html

"Second, within a library, program, or function ..."
** Maybe your insert "First" and "Third":
** ``First, for a given library, program, or function, ...``;
** ``Third, at the statement level, ...``

"... assign a percentage. This percentage is ..."
** Maybe split this, and unite "This percentage is" and "used to calculate a letter grade." ? Looks uneven.

"The compiler won't let you run if there are compiler errors."
** ``run the program if`` ?

"Commenting out the broken code will ensure the broken code doesn't execute and cause problems until you can fix it."
** ``(and cause problems)``. Funny: when i fix it, it will execute and cause problems!

"consider using the #if 0 preprocessor directive,"
** Please, add the <code> tag (zero VS o).

PS: The avatar below is brilliant.

nascardriver
January 5, 2020 at 6:20 am · Reply

> tabs/spaces
I don't know, test it, probably varies between editors.

> snippet
Lesson updated to say "one the the first encounters".

> not inside the comment
I added quotes.

> Newton
Capitalized. Comments don't always follow proper grammar. If there are many "mistakes" in a lesson, but they're consistent, don't be bothered by them.

> he or she
Genders are a mess, there is no good solution. Won't change.

> First, Second, Third
Updated.

> Percentage
Updated.

> Compiler
Updated, sentence was wrong, the compiler doesn't run the program.

> #if 0
Added code tags.

Vitaliy Sh.
January 5, 2020 at 7:04 am · Reply

Hi sir!
"The statements above represent on of our first encounters"
** ``one of our``

nascardriver
January 5, 2020 at 7:05 am · Reply

Ayy, thanks!

Alica
January 1, 2020 at 2:07 am · Reply

Thank you so much for providing nice tutorial

Jonas
June 7, 2019 at 1:17 pm · Reply

Found a typo under 'Proper use of comments'. The first comment should say (asterisks mark the erroneous word):

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

Vitaliy Sh.
January 2, 2020 at 11:02 pm · Reply

```
1 // □
2 std::trollmode.emit("force joke");
```

"... based on THEIR test...", sir Jonas!

skibi
May 11, 2019 at 4:20 am · Reply

I don't really understand this concept at all

Georges Theodosiou
April 2, 2019 at 12:51 am · Reply

Dear Teacher,
Please let me say you that following program works in Visual Studio 2017, 2019, and online compilers
http://cpp.sh/
https://www.onlinegdb.com/online_c++_compiler

```
1 # include <iostream> Everything from here to the end of the line is ignored
2
3 int main()
4 {
5     std::cout << "Hello world! \n";
6     return 0;
7 }
```

Regards

nascardriver
April 2, 2019 at 3:01 am · Reply

Behavior is undefined. Make sure you followed lesson 0.10 and 0.11.

Georges Theodosiou
April 2, 2019 at 4:59 am · Reply

Mr. nascardriver,

Please accept my many thanks for you responded to my message and that immediately. Many more for your message is instructive.
I appreciate that "behavior is undefined".
However after I have configured V.S. 2017 and 2019 according to lessons 0.10 and 0.11 (setting "Disable Language Extensions" to Yes (/Za), and "Warning level" to Level4 (/W4)) warnings are many.
In V.S. 2017 of the kind
"'Project5.exe' (Win32): Loaded 'C:\Windows\SysWOW64\KernelBase.dll'. Cannot find or open the PDB file."
and in V.S. 2019 RC of the kind
"'Project-comment.exe' (Win32): Loaded 'C:\Windows\SysWOW64\KernelBase.dll'."
Only name in .dll file is different in warnings.

With regards and friendship
Georges Theodosiou

Badreddine Boukheit
August 24, 2019 at 11:57 pm · Reply

include <iostream> Everything from here to the end of the line is ignored

there is an error in this line sir , you should put // before the things that you do not want the compiler to compile , the line should be :

include <iostream> //Everything from here to the end of the line is ignored

Georges Theodosiou
August 31, 2019 at 12:27 am · Reply

Badreddine Boukheit,
Please accept my thanks for your response and many more for your instruction.
Problem is that this program works. Regards.

Kaladin
September 29, 2019 at 1:49 am · Reply

May be ,you did not rebuild your program before running it. That's why it might be working. Try rebuilding it.

Georges Theodosiou
October 1, 2019 at 5:28 am · Reply

Mr. Kaladin,
Please accept my thanks for your comment and many more for your suggestion.
In turn I suggest you read Mr. NASCAR driver's above comment: "Behavior is undefined. Make sure you followed lesson 0.10 and 0.11." Regards.

Todd Riemenschneider
October 2, 2019 at 5:45 pm · Reply

Thanks for the tip Georges!

Georges Theodosiou
October 7, 2019 at 6:25 am · Reply

Mr. Todd Riemenschneider
Please let me express my thanks for your comment and many more for your thanks. Also let me a greater tip: Undefined behavior is the greatest problem I face in learning programming. Regards.

Bastian
March 22, 2019 at 8:10 am · Reply

if you have a bit of code that you will frequently comment out and put back in, you can just put /**/ below the code you want commented out, then when you need it commented out, you can just place /* above the code, like

```
/*
exampleCode();
/**/

//to

exampleCode();
/**/
```

Matt
July 27, 2019 at 12:24 pm · Reply

Thanks this is a smart way to comment out thing

Mr.F
March 11, 2019 at 6:02 pm · Reply

Hey Alex.
3) To find the source of an error. If a program isn’t producing the desired results (or is crashing), it can sometimes be useful 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. But won't that result in a compile error if I exclude one line of the code?

Alex

March 11, 2019 at 10:26 pm · Reply

Only if you comment out a declaration and not the use of it later.

If you're strategic about what you comment out, you can avoid this. The easiest thing to comment out are function calls that don't use a returned value. You can try commenting out parts of functions, or even entire functions as long as you comment out all the calls to those functions too.

Louis Cloete
February 6, 2019 at 2:38 pm · Reply

Hey Alex!

I must say, I like this new styling of the pages a lot! Just a comment (no pun intended) about multiline comments to disable code: If you want to disable code and you are not sure if it contains multiline comments, you can do this:

```
#if 0
/*code*/
#endif
```

Then you don't have to worry about multiline comments inside the block you want to disable.

Alex
February 7, 2019 at 8:06 pm · Reply

Good point. I'll mention it. Thanks for sharing!

Smidge
February 3, 2019 at 11:05 pm · Reply

Thanks again for this wonderful tutorial, however you missed a single semi-colon on one of your examples.

```
// std::cout lives in the iostream library
std::cout << "Hello world!\n";

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

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

Alex
February 4, 2019 at 9:00 pm · Reply

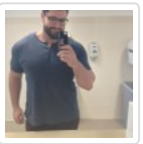
I sure did. Thanks for pointing this out! Fixed.

Ajay Shinde
January 24, 2019 at 10:29 am · Reply

did i use comments properly?

```
/*
    to know the cout funtion in c++
*/
```

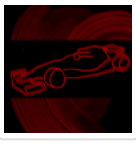
```
4
5  //to make cout function acvailable to us
6  #include<iostream>
7
8  //instruction for OS to start executing program from here
9  int main(){
10
11 // to print hello world to screen
12 std::cout<<"Hello World!";
13
14 }
```



Felipe
August 14, 2019 at 4:25 pm · Reply

I think that a comment if for you and other people is understandable what you want to comment, it is valid, for me this is valid and, it is a good way to comment. Good luck in this bro!

PD: Remember put
before '}', put 'return 0;'
And if you wan't to pause the program put on first line '#include <conio.h>' and before of return 0; , 'getche();', or if you have an error put '_getche();'



nascardriver
August 14, 2019 at 11:03 pm · Reply

> And if you wan't to pause the program put on first line '#include <conio.h>'
Don't do this. <conio.h> is a Windows-exclusive header, it won't work on other platforms.

Henric
January 2, 2019 at 3:03 pm · Reply

Hi Alex!
These are some amazing tutorials and I think it's great that you still are improving it after a long time!

There seems to be a typo in the "Proper use of comments" section. The first comment says "This program calculate the student's final grade..." rather than "...calculateS the student's final grade..."

Allu orton
August 22, 2018 at 11:42 pm · Reply

Big thanks Alex

boredprogrammer
May 19, 2018 at 2:07 am · Reply

For uncommenting code blocks using common comment syntax is not a good idea.

```
/*
 a = TEST;
 cout << a;
*/
```

In general, it's not a good idea. Why? we tend to do that for different reasons but mostly to test some code or keep some code for later testing. The problem is, it's easy to forget those pieces of code with many comments around. Or popullate your code with commented code that other programer could see and think "what is that? is commented because it's not used anymore? or a piece of code pending of testing? or the developer forgot to uncomment a vital piece of code?".

This is the main problem, and worst, it also happened to me with my own code, when time later I found commented code and I wasn't sure why it was commented (of course I didn't leave any message).

I think the best approach is using #if-#endif preprocessor blocks to "comment" code. It transmits the idea that code was anulated on purppose:

```
#if 0
  a = TEST;
  cout << a;
#endif
```

Even better, if you place a guard label instead the 0

```
#define CODE_TO_TEST   0

#if CODE_TO_TEST
#pragma warning Code requires testing
  a = TEST;
  cout << a;
#endif
```

We can know exactly where and what is happening. I think this approach is much better and avoid some confusions, also enables the compiler to performs some checks, impossible to do with regular comments.

Samira Ferdi
April 18, 2018 at 6:55 pm · Reply

Hi Alex and nascardriver! I back to this lesson because I have issues about commenting and I don't know what should I gonna do with commenting if I try make a program.

These are my issues:
1. Should I give comment every line of my code?
2. What's the difference 'bout "given/at" library, program, or function and "inside" library, program, or function? Could you give me examples?
3. Why you explain WHAT at the function/library/program and HOW inside the function/library/program? What about if I explain both WHAT and HOW at the function/library/program or I explain both WHAT and HOW inside the function/library/program?
4. What is the comments that put at the top - above any source code? I saw many source code that put comments at the top
5. Library in C++ like file.cpp? What about if I have a libs folder that have many file.cpp, how can I give a comment to it?

nascardriver
April 19, 2018 at 5:54 am · Reply

Hi Samira!

1. No. See the "Proper use of comments" section.
2. I don't understand.
3.
> Why you explain WHAT at the function/library/program and HOW inside the function/library/program?
When you use a library you usually don't care how it's doing what it's doing, you care about how you use

it and what the functions do. The 'what' comments are there for the user of the library, the 'how' comments are there for the developer of the library.
> What about if I explain both WHAT and HOW at the function/library/program
The user of your library doesn't care about how it's doing what it's doing, only tell them what they want to know.
> I explain both WHAT and HOW inside the function/library/program
'what' has to go in the header so you'd have duplicate comments.
4.
You either talk about file information (Created at date, Created by, Usage rights).
You can ignore those, I assume they're used in development teams without version control.
Or you're talking about documentation comments

```
/**
 * Does this and that
 * @param x Used for big calculation
 * @return Mass of the sun
 */
```

They are used to allow documentation generation so you have a nice overview of your code's functionality as a pdf or html.
5. I don't understand.

Alex is better at understanding questions than I am. If you give him a couple of days he might drop a comment. If he doesn't, you can try reformulating your questions and giving examples so I can understand them.

Samira Ferdi
April 19, 2018 at 4:24 pm · Reply

Thank you for replying, nascardriver! I really really appreciate that. This is my reformulated question for number 2 and 5.

2. In the proper use of comments, Alex said that at library, program, and function describe 'what'. Inside the library, program or function describe 'how'. What is the difference betwen 'at' and 'inside'?

5. The library is the collection of function, right? And the library usually a folder like libs folder. How can I comment 'what' at library (libs folder)?

nascardriver
April 20, 2018 at 7:29 am · Reply

"At the library, program, or function"
"At the library, program, or function level"
The "level" makes a difference.

"At the library, program, or function level"
Means in the header file

"Inside the library, program or function"
Means in the source file

> The library is the collection of function, right?
A library can contain everything you can write in C++.

> How can I comment 'what' at library (libs folder)?
You'll usually have a main class (Not named "main", I mean the primary class) which you can use the describe the library as a whole. But you won't normally need to describe the library in

your code, because if a user has your code he already knows what it's about, he wouldn't have downloaded it otherwise. The general library or program is described in a readme.md file. For example this one for Visual Studio Code
https://github.com/Microsoft/vscode/blob/master/README.md

Alistair Ian McCall
April 11, 2018 at 7:13 am · Reply

I am a 68-9 yrs. old absolute beginner, with no foreknowledge of the subject, but I am willing to learn. I would be very happy for any pointers you may be able to offer.

A. I. McCall

vinayak ruhela
April 7, 2018 at 9:48 pm · Reply

what is the work of comments?

Khaled Morgham
February 26, 2018 at 12:37 am · Reply

Really, you have made me like programming in C++. Thanx

anjali
February 14, 2018 at 7:18 am · Reply

thanks for the lesson but i have a doubt, i didn't understand the purpose of multi line comment.please explain..

Alex
February 15, 2018 at 5:42 pm · Reply

Multiline comments are used when comments are lengthy. For example, if you're describing what a function does, that could take many lines. You could do that by using many single line comments in sequence, or one multi-line comment containing as many lines as you need.

The following are really two ways of doing the same thing:

```
1 // line 1
2 // line 2
3 // line 3
4
5 /*
6   line 1
7   line 2
8   line 3
9 */
```

« Older Comments 1 2 3