2.x — Chapter 2 summary and quiz

BY ALEX ON FEBRUARY 1ST, 2019 | LAST MODIFIED BY NASCARDRIVER ON FEBRUARY 7TH, 2020

Quick Summary

A **function** is a reusable sequence of statements designed to do a particular job. Functions you write yourself are called **user-defined** functions.

A **function call** is an expression that tells the CPU to execute a function. The function initiating the function call is the **caller**, and the function being called is the **callee** or **called** function. Do not forget to include parenthesis when making a function call.

The curly braces and statements in a function definition are called the **function body**.

The **return type** of a function indicates the type of value that the function will return. The **return statement** determines the specific **return value** that is returned to the caller. This process is called **return by value**. Functions can have a return type of **void** if they do not return a value to the caller. Failure to return a value from a non-void function will result in undefined behavior.

The return value from function *main* is called a **status code**, and it tells the operating system (and any other programs that called yours) whether your program executed successfully or not. By consensus a return value of 0 means success, and a positive return value means failure.

A **function parameter** is a variable used in a function where the value is provided by the caller of the function. An **argument** is the specific value passed from the caller to the function. When an argument is copied into the parameter, this is called **pass by value**.

C++ does not define whether function calls evaluate arguments left to right or vice-versa.

Function parameters and variables defined inside the function body are called **local variables**. The time in which a variable exists is called its **lifetime**. Variables are created and destroyed at **runtime**, which is when the program is running. A variable's **scope** determines where it can be accessed. When a variable can be accessed, we say it is **in scope**. When it can not be accessed, we say it is **out of scope**. Scope is a **compile-time** property, meaning it is enforced at compile time.

Refactoring is the process of breaking down a larger function into many smaller, simpler functions.

Whitespace refers to characters used for formatting purposes. In C++, this includes spaces, tabs, and newlines.

A **forward declaration** allows us to tell the compiler about the existence of an identifier before actually defining the identifier. To write a forward declaration for a function, we use a **function prototype**, which includes the function's return type, name, and parameters, but no function body.

A **definition** actually implements (for functions and types) or instantiates (for variables) an identifier. A **declaration** is a statement that tells the compiler about the existence of the identifier. In C++, all definitions serve as declarations. **Pure declarations** are declarations that are not also definitions (such as function prototypes).

Most non-trivial programs contain multiple files.

When two identifiers are introduced into the same program in a way that the compiler can't tell them apart, the compiler or linker will produce a **naming collision**. A **namespace** guarantees that all identifiers within the namespace are unique. The std namespace is one such namespace.

The **preprocessor** is a process that runs on the code before it is compiled. **Directives** are special instructions to the preprocessor. Directives start with a # symbol and end with a newline. A **macro** is a rule that defines how input

text is converted to a replacement output text.

Header files are files designed to propagate declarations to code files. When using the *#include* directive, the *#include* directive is replaced by the contents of the included file. When including headers, use angled brackets when including system headers (e.g. those in the C++ standard library), and use double quotes when including user-defined headers (the ones you write). When including system headers, include the versions with no .h extension if they exist.

Header guards prevent the contents of a header from being included more than once into a given code file. They do not prevent the contents of a header from being included into multiple different code files.

Quiz time

Question #1

Write a single-file program (named main.cpp) that reads two separate integers from the user, adds them together, and then outputs the answer. The program should use three functions:

- A function named “readNumber” should be used to get (and return) a single integer from the user.
- A function named “writeAnswer” should be used to output the answer. This function should take a single parameter and have no return value.
- A main() function should be used to glue the above functions together.

Show Hint

Show Hint

Show Solution

Question #2

Modify the program you wrote in exercise #1 so that readNumber() and writeAnswer() live in a separate file called “io.cpp”. Use a forward declaration to access them from main().

If you’re having problems, make sure “io.cpp” is properly added to your project so it gets compiled.

Show Solution

Question #3

Modify the program you wrote in #2 so that it uses a header file (named io.h) to access the functions instead of using forward declarations directly in your code (.cpp) files. Make sure your header file uses header guards.

Show Solution

3.1 -- Syntax and semantic errors

Index

2.13 -- How to design your first programs

C++ TUTORIAL | PRINT THIS POST

83 comments to 2.x — Chapter 2 summary and quiz

Ryan
February 5, 2020 at 8:37 am · Reply

Quick typos! Missing an apostrophe in "A variables scope determines". I think there's an unnecessary 'the' in "A definition actually implements the (for functions and types) or instantiates (for variables) an identifier."?

nascardriver
February 7, 2020 at 9:04 am · Reply

Yep, thanks!

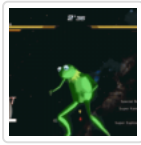
Kermit
January 11, 2020 at 1:24 am · Reply

Hey i did this before checking your solution. it works but im not sure if i followed ur question correctly.

```
#include <iostream>

using namespace std;

int readNumber()
{
    cout << "Enter Integer: ";
    int x{};
    cin >> x;
    return x;
}
void writeAnswer(int x)
{

    cout << "Addition is: " << readNumber() + x << '\n';
}


int main()
{
    writeAnswer(readNumber());
    return 0;
}
```

nascardriver
January 12, 2020 at 3:42 am · Reply

`writeAnswer` isn't supposed to call `readNumber`. Calling `readNumber` from `writeAnswer` decreases reusability of `writeAnswer`.



Kermit
January 14, 2020 at 2:38 am · Reply

yeah i get it :)

Arun
January 3, 2020 at 11:02 am · Reply

Would it be incorrect to include <iostream> in the io.h header file? It still compiles and everything works but is this something we should avoid?

nascardriver
January 5, 2020 at 6:22 am · Reply

Headers should include as few files as possible. "io.h" doesn't use anything from , so it should include it. Unnecessary includes slow down compilation and cause conflicts.

Mesko
December 24, 2019 at 8:46 am · Reply

io.h

```
#ifndef IO_H
#define IO_H

void writeAnswer();

#endif
```

io.cpp

```
#include <iostream>

int readNumber() {

    std::cout << "Type in an integer: ";
    int a;
    std::cin >> a;

    return a;
}

void writeAnswer() {

    int a = readNumber();
    int b = readNumber();

    std::cout << "The sum of two integers is : " << a + b;


}
```

main.cpp

```
#include <iostream>
#include "io.h"

int main() {

    writeAnswer();

    return 0;
}
```

I have done it like this, it works but I am now not sure if this is also a proper way to do it?

nascardriver
December 27, 2019 at 6:05 am · Reply

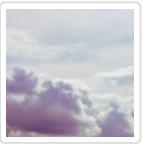
`writeAnswer` is no longer reusable, because it performs the calculation. The calculation was supposed to be performed by `main`. That way `writeAnswer` can be used for other calculations too.
The separation into different files is correct though.

Chayim
December 18, 2019 at 12:36 am · Reply

I don't understand anything in solution 3.
• Why does io.cpp need to include io.h if those 2 functions are defined in io.cpp?
• What does io.h add? Why is io.h needed if those 2 functions are defined in io.cpp and the linker will apply it in main( )?
• And why does main( ) have to include io.h if those functions are defined in io.cpp and the linker will apply it to main( )?

nascardriver
December 18, 2019 at 5:23 am · Reply

Including the header in the source file can help catching errors early (https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comment-398571). "main.cpp" can't use the functions without forward declarations. The forward declarations are in "io.h".



Chayim
December 17, 2019 at 11:09 pm · Reply

In solution question 3
Why is io.h not in the beginning before io.cpp?

Alex
December 19, 2019 at 1:21 pm · Reply

I... don't know. Fixed. :)

Gustav
October 31, 2019 at 3:57 am · Reply

Hi Alex, great tutorial, learning a lot!
I just want you to comment on my code

```
#include <iostream>


int readNumber(){
    int x;
    std::cout << "Enter an Integer: ";
    std::cin >> x;
    return x;
}

void writeAnswer(int y){
    std::cout << "The sum of two integers is : " << y << '\n';

}

int main()
{
    writeAnswer(readNumber() + readNumber());
    return 0;
}
```

Is there a better way of writing this?

nascardriver
October 31, 2019 at 4:34 am · Reply

Hey Gustav!

In your current code, this doesn't matter, but you might want to move the calls to `readNumber` out of the call to `writeAnswer`. It's unspecified if the left or right `readNumber` is called first. As I said, irrelevant now, but if one `readNumber` affects the other, this will cause trouble that you might not spot. Otherwise your code looks good, you could look into an auto-formatter to format your code consistently.

Gustav
October 31, 2019 at 6:58 pm · Reply

Thanks for your feedback, I just realized that I gave you the first draft of my code Anyways, here's the final iteration.

```
#include <iostream>


int readNumber(){
    int x;
    std::cout << "Enter an Integer: ";
    std::cin >> x;
    return x;
}

void writeAnswer(int y){
    std::cout << "The sum of two integers is : " << y << '\n';

}

int main()
{
    int x{readNumer()};
    int y{readNumer()};
    writeAnswer(x + y);
    return 0;
```

```
22 }
```

Onto the next chapters.

~Cheers!

Thevnort
October 13, 2019 at 6:56 pm · Reply

For question 3, as part of the quiz, why did you add '#include "io.h"' to io.cpp when the IDE (Visual Studio 2019) knows to reference the .cpp files?

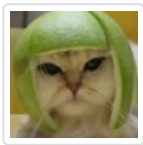
nascardriver
October 14, 2019 at 4:53 am · Reply

See here
https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comment-398571
or here
https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comment-413858

This question has been asked several times, you might get more information from reading the comments of lesson 2.11 https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comments

Ayberk
September 6, 2019 at 1:38 pm · Reply

If main function returns negative number then what happens ?



Alex
September 6, 2019 at 4:30 pm · Reply

Nothing different than if it returns a positive number or zero. Exit status codes can be negative, zero, or positive.

Ayberk
September 7, 2019 at 12:13 pm · Reply

Thanks!

Marie Bethell
August 23, 2019 at 12:33 pm · Reply

Does the order of #includes matter? As in, should #include <iostream> come first or #include "io.h". Thanks!

nascardriver
August 24, 2019 at 2:00 am · Reply

If the headers are poorly written, the order matter. Most of the time it doesn't matter. A common convention is local includes before global includes and each of the blocks sorted alphabetically.

```
#include "a.hpp"
#include "io.hpp"

#include <iostream>
#include <vector>
```

Marie Bethell
August 24, 2019 at 12:29 pm · Reply

Okay, thanks!

Nikolas
July 23, 2019 at 11:04 am · Reply

Is it bad practice to have your print/cout statements inside functions? I started learning to program with Python and was told it was bad practice to have my user interaction code inside functions

nascardriver
July 24, 2019 at 4:05 am · Reply

You can't have code outside of functions in cpp, so no, it's not bad practice.
You should separate logic and io.

John
October 18, 2019 at 9:44 pm · Reply

nascardriver, I want to start by saying thank you so much for your work here. The way things are worded and broken down has made it really easy to learn the concepts so far. I have a similar question as Nikolas:

```
int readNumber()
{
    std::cout << "Enter a number: ";
    int x {};
    std::cin >> x;
    return x;
}
```

2.5 — Why functions are useful, and how to use them effectively

New programmers often combine calculating a value and printing the calculated value into a single function. However, this violates the “one task” rule of thumb for functions. A function that calculates a value should return the value to the caller and let the caller decide what to do with the calculated value (such as call another function to print the value).

Just going off of the above, wouldn't you say that it is more appropriate to have the the cout statement in the main function? Although you are not really doing anything with the input from the user in readNumber() except for returning it, one can argue that communicating with the user for an input and getting(and returning) the input with the same function are two separate tasks.

nascardriver
October 19, 2019 at 3:11 am · Reply

I'd move the `std::cout` out of `readNumber`, but not into `main`. I'd add a new function `promptForNumber` that prints "Enter a number" and calls `readNumber`.
Then, in `main`, the coder can decide if they want a prompt to be printed or not, or print their own prompt.

John
October 19, 2019 at 5:59 pm · Reply

Thanks.

Jake
July 18, 2019 at 9:09 am · Reply

How come for the second quiz, in the main.cpp file you did not use #include <iostream>.

nascardriver
July 18, 2019 at 9:28 am · Reply

"main.cpp" doesn't use anything from <iostream>, so it shouldn't include it.
"io.cpp" uses `std::cin` and `std::cout`, so it needs to include <iostream>.

Jake
July 18, 2019 at 9:34 am · Reply

ok but will it do anything if I do have it included in main?

nascardriver
July 18, 2019 at 9:36 am · Reply

Slow down compilation and provide more possibilities for name conflicts.

Bharat
July 13, 2019 at 6:11 am · Reply

Hi,
just wondering why you did not use int x{0}. Instead you just did int x. Are they both the same?
Edit: Also, I copied and pasted excersise 1 into visual studio and there seems to be a build error. Any ideas?
Error: LNK2019 unresolved external symbol _main referenced in function "int __cdecl invoke_main(void)" (?invoke_main@@YAHXZ)Line 1

nascardriver
July 13, 2019 at 6:54 am · Reply

They're not the same, use

```
1 int x{};
```

Without the curly brackets, the value of `x` is undefined and reading from `x` produces undefined behavior.

There's nothing wrong with the code. Make sure you saved the file before compiling it.

Bharat
July 15, 2019 at 5:39 am · Reply

I still have the error in line 1, these are the errors. I saved it too.
1)
https://docs.microsoft.com/en-us/cpp/error-messages/tool-errors/linker-tools-error-lnk2019?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev15.query%3FappId%3DDev15IDEF1%26l%3DEN-US%26k%3Dk(LNK2019)%26rd%3Dtrue&view=vs-2019

2)
https://docs.microsoft.com/en-us/cpp/error-messages/tool-errors/linker-tools-error-lnk1120?f1url=https%3A%2F%2Fmsdn.microsoft.com%2Fquery%2Fdev15.query%3FappId%3DDev15IDEF1%26l%3DEN-US%26k%3Dk(LNK1120)%26rd%3Dtrue&view=vs-2019

nascardriver
July 15, 2019 at 5:48 am · Reply

Seems like a problem in your project settings then. Try googling for the error with respect to "invoke_main".

Bharat
July 15, 2019 at 9:54 am · Reply

ok thank you

Bharat
July 16, 2019 at 5:59 am · Reply

oh ok I found the issue, I was doing the code in a header file lol.

Alex
July 13, 2019 at 8:23 pm · Reply

I updated the answers to use x {} (as per the best practice to initialize all variables). Thanks for pointing out the omission.

Bharat
July 15, 2019 at 5:30 am · Reply

Ok, thanks to both of you.

Nirbhay
July 12, 2019 at 1:02 am · Reply

Hello!
Is #include "blabla.h" in the first line of blabla.cpp redundant?
As blabla.cpp already defines(so automatically declares) some functions in it. So it doesn't need to include the

declarations and therefore the header file.

Am I correct?

Thanks :)

nascardriver
July 12, 2019 at 1:06 am · Reply

Hello!

It can help you detecting errors early. See my comments here
If you like foxes
https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comment-413858
If you like cats
https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comment-398571

Nirbhay
July 12, 2019 at 1:28 am · Reply

Thanks :) I think it boils down to good practice in case of usual code (not libraries).

Samovar
July 3, 2019 at 12:24 am · Reply

I run the following:

[code]
#include<iostream>

int readNumber()
{
   std::cout << "Enter a number: ";
   int x;
   std::cin >> x;
   return x;
}

void writeAnswer(int x)
{
   std::cout << "The answer is " << x << '\n';
}

int main()
{
   int x { readNumber() };
   int y { readNumber() };
   writeAnswer(x + y); // using operator+ to pass the sum of x and y to writeAnswer()
   return 0;
}
[\code]

I run the code on Code::Blocks 13.12 on a MacBook Pro. The code did not compile. I receive
"line 18 error:expected ' ; ' at end of declaration."

as well as

"line 18 error:expected ' ; ' at end of declaration."

Please advise.

nascardriver
July 3, 2019 at 2:46 am · Reply

It appears you didn't enable C++11 or later. Make sure you followed lesson 0.10 and 0.11.
Closing code tags use a forward slash (/).

Samovar
July 3, 2019 at 3:59 am · Reply

I followed lesson 0.10 and 0.11.
4 warnings are enabled in the compiler flags.
1 phrase typed "-Wsign-conversion -Werror" in the other options.

Still, the same 2 errors persist:

"line 18 error:expected ' ; ' at end of declaration."
"line 19 error:expected ' ; ' at end of declaration."

The entire code is reproduced below:

Please advise.

```
#include<iostream>

int readNumber()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;
    return x;
}

void writeAnswer(int x)
{
    std::cout << "The answer is " << x << '\n';
}

int main()
{
    int x { readNumber() };
    int y { readNumber() };
    writeAnswer(x + y); // using operator+ to pass the sum of x and y to writeAn
swer()
    return 0;
}
```

nascardriver
July 3, 2019 at 4:04 am · Reply

My bad, changing the C++ version is part of lesson 0.6. The section is titled "Code::Blocks (for Linux or Windows)", it should be similar on Mac.

PRANAV AGARWAL
June 7, 2019 at 2:15 am · Reply

Thanks a lot, Sir.

Lawrence
May 25, 2019 at 4:31 pm · Reply

Thanks alot for all your hard work! Words can't explain how thankful we are!

alfonso
May 12, 2019 at 12:35 am · Reply

"Scope is a compile-time property, meaning it is enforced at compile time."
I need more help with this. I "know" the scope is created (and destroyed) at runtime. No function call then no scope at all. So what does it mean "enforced"?

Like that?
Lifetime of the scope -> runtime.
Scope (as access meaning) -> (is determined at) compile-time.

Alex
May 13, 2019 at 3:35 pm · Reply

Scope isn't created and destroyed at runtime. Scope is a compile-time property, meaning the compiler enforces the scope of an object. Enforces in this context means the compiler will give you an error if you try to access an object that is not in scope.

There's no such thing as "lifetime of the scope". Lifetime is a property of objects. Since objects are only created/destroyed at runtime, this is a runtime property.

For local objects, scope and lifetime are intertwined, in that the lifetime of local objects is defined by their scope -- that is, they are created at the point where they come into scope, and destroyed at the point where they go out of scope.

Paveun
May 10, 2019 at 12:19 pm · Reply

Is it required to include the header in the io.cpp file being referenced by said header?

```
#include "io.h"
```

Also in main(), is there a reason not to simply put

```
int main()
{
    writeAnswer(readNumber() + readNumber());
    return 0;
}
```

Thanks!

nascardriver
May 11, 2019 at 4:02 am · Reply

> Is it required to include the header in the io.cpp file
No, but it can be helpful.
See my comment in lesson 2.11 (mew.hpp and mew.cpp should be cat.hpp and cat.cpp)
https://www.learncpp.com/cpp-tutorial/header-files/comment-page-8/#comment-398571

> is there a reason not to simply put
It's unspecified whether the left or right @readNumber will be called first. Since they're the same, it should be a problem. If you require a specific order, you should use temporary variables.

Paveun
May 13, 2019 at 10:51 am · Reply

Thanks! Very helpful

Fisticuffs
March 15, 2019 at 9:01 am · Reply

The solution for question 3 includes io.cpp which is copied below. Although there is an io.h file as part of the solution, it isn't #included in io.cpp. However, there is a best practice note from section 2.11 that states that, "A code file should #include its paired header file (if it exists). If you're curious as to why, see nascardriver's comment here."

So, although not necessary to make this work correctly, should there be a '#include "io.h"' line in the io.cpp file?

```
#include <iostream>

int readNumber()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;
    return x;
}

void writeAnswer(int x)
{
    std::cout << "The answer is " << x << '\n';
}
```

Alex
March 19, 2019 at 4:41 pm · Reply

Yes, good catch. Answer amended.

Dave Sosa
February 26, 2019 at 3:54 pm · Reply

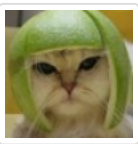
Hello just wondering why you need to put int "int x" in "voide writeAnswer(int x). Why is the "int x" necessary.

Alex
February 27, 2019 at 8:25 pm · Reply

The writeAnswer is expecting the caller to pass in a value to print -- therefore, it needs to take a parameter, so that the caller can pass in a numeric argument to print.

nascardriver
February 26, 2019 at 12:33 am · Reply

Hi Alex!
There's a trend to omit the trailing printed line feed lately. You talked about this in lesson 1.5 and I don't think it could be explained a lot better than you did. Maybe a spot-the-problem quiz helps.



Alex
February 27, 2019 at 10:17 am · Reply

Just to be clear, are you saying the trend is to move from this:

```
1 std::cout << "Line 1\n";
2 std::cout << "Line 2\n";
```

To this?

```
1 std::cout << "Line 1";
2 std::cout << "\nLine 2";
```

Or are you saying it's to move from this:

```
1 std::cout << "Line 1\n";
2 std::cout << "Line 2\n";
```

To this?

```
1 std::cout << "Line 1\n";
2 std::cout << "Line 2";
```

Please clarify and then I can try to address. :)

nascardriver
February 28, 2019 at 12:38 am · Reply

```
1 std::cout << "Line 1\n";
2 std::cout << "Line 2";
```

This, where "Line 2" is the last thing printed by the program. Your first example has the same problem.
Not printing a line feed at the end of the program will cause the prompt printed by the terminal to be printed on the same line as "Line 2".

phoenix blue
February 22, 2019 at 10:08 pm · Reply

excuse me, First thanks for the tutorial. In earlier chapter you said that it's better to use uniform intialization. But now why are you using direct intialization? (Quiz #1)

so uh.. which one should be prefered?
For now I will proceed to use uniform intialization following earlier chapter, I would like to know if there is reasoning behind the use of copy intialization or if I got it wrong.

nascardriver
February 23, 2019 at 9:28 am · Reply

Hi!

Use uniform initialization.
Alex uses copy/direct initialization in all lessons that were written before 2019.

Alex
February 24, 2019 at 7:57 pm · Reply

Nascardriver answered correctly. I've updated the quiz answers to use uniform initialization. Thanks for pointing out the issue.

Kushahl
February 22, 2019 at 8:29 am · Reply

In question #2

My source files are:

First program:

```
#include <iostream>              //This includes IOStream header file bot basic input and output
   int main()
   {
      int readnumber();
      void writeanswer(int x);
      int x,y;
      std::cout<<"Enter 2 numbers to add"<<'\n';
      y=readnumber();
      x=readnumber();
      writeanswer(x+y);
   return 0;
}
```

io.cpp

```
int readnumber()          //This function reads a function
   {
      int x;
      std::cin>>x;
      return x;
   }
   void writeanswer(int x)
   {
      std::cout<<" Your answer is : "<<x;
   }
```

stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// Firstprogram.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file

I get below error

:1>------ Build started: Project: Firstprogram, Configuration: Debug Win32 ------
1> io.cpp
1>c:\users\kusingh\documents\visual studio 2012\projects\firstprogram\firstprogram\io.cpp(4): error C2039: 'cin' : is not a member of 'std'
1>c:\users\kusingh\documents\visual studio 2012\projects\firstprogram\firstprogram\io.cpp(4): error C2065: 'cin' : undeclared identifier
1>c:\users\kusingh\documents\visual studio 2012\projects\firstprogram\firstprogram\io.cpp(9): error C2039: 'cout' : is not a member of 'std'
1>c:\users\kusingh\documents\visual studio 2012\projects\firstprogram\firstprogram\io.cpp(9): error C2065: 'cout' : undeclared identifier
1> Firstprogram.cpp
1> Generating Code...
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========

I do not get any compile error when I add include<iostream> in the file io.cpp, but I do get below warning after program is successfully executed.

'Firstprogram.exe' (Win32): Loaded 'C:\Users\kusingh\Documents\Visual Studio 2012\Projects\Firstprogram\Debug\Firstprogram.exe'. Symbols loaded.
'Firstprogram.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ntdll.dll'. Cannot find or open the PDB file.
'Firstprogram.exe' (Win32): Loaded 'C:\Windows\SysWOW64\kernel32.dll'. Cannot find or open the PDB file.
'Firstprogram.exe' (Win32): Loaded 'C:\Windows\SysWOW64\KernelBase.dll'. Cannot find or open the PDB file.
'Firstprogram.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcp110d.dll'. Symbols loaded.
'Firstprogram.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcr110d.dll'. Symbols loaded.
The program '[8976] Firstprogram.exe' has exited with code 0 (0x0).

My questions:

Should we not include same header files in multiple files ?
What does these warning signify even though program executed successfully ? I also noted that execcution window did not wait for me to press enter to exit unlike everytime. It did ask me in my successful program in previous question number 1.

Thankyou so much for these tutorials.

Regards
K

Arthur
February 21, 2019 at 10:59 am · Reply

hi, I have no experience with code and just started this tut. yesterday , I do not have an IDE yet (space problems on solid-state c drive) so am using notepad++ to write out answers for now. had to cheat a bit on answer 1 but was pretty close. as this tutorial is being updated I am unsure if I have used recommended brackets appropriately, and am wondering if I am understanding what the code is doing. I am assuming this program will work , will print to the console "basic addition:", "enter number :", "enter number :", "answer equals :" and the numbers will all be in a vertical line in the console. criticism is welcome. (hope I get the code tags right)

```
1  //  declares c++ library objects to allow input output
2  Include<iostream>
3
4
5  //  declares readNumber function with integer return value
```

```
int readNumber ()
{

    std::cout << "enter number  :" << '\n';//outputs input request to user

    int a {0};                              //initializes 'a'

    std::cin >> a;                          //accepts console input as 'a'

    return a;                               //returns value to main
}


//  declares writeAnswer function with null return value
//  initializes 'a' with argument 'a+b' @main
void writeAnswer (int a)
{
    std::cout << "answer equals :" << a << '\n';// outputs 'a' to console

}

//declares main function with integer return value
int main ()
{
    std::cout << "basic addition:" <<'\n';//outputs program purpose to console

    int a {readNumber()};                   //initializes 'a' with
                                            //argument calling readNumber

    int b {readNumber()};                   //initializes 'b' with
                                            //argument calling readNumber

    writeAnswer (a + B);                    //argument calling writeAnswer

    return 0;                              //returns 0 value
                                            //to system = program complete
}
```

thanks for any feedback

Arthur
February 21, 2019 at 2:17 pm · Reply

or it might work if I fixed line 2

2 #include<iostream>

:-\ lol

nascardriver
February 22, 2019 at 3:57 am · Reply

Hi Arthur!

If you don't have access to a normal compiler, you can use an online compiler (eg. https://www.onlinegdb.com/online_c++_compiler).

Apart from the include,
* Line 9, 13: You don't need to use << to concatenate strings. Use

```
std::cout << "basic addition:\n";
```

```
// or
// Note: This form of concatenation doesn't work with chars.
std::cout << "basic addition:" "\n";
```

* Line 38: 'B' should be 'b'.

Your initializations are correct.

Arthur
February 26, 2019 at 12:10 am · Reply

thank you for the feedback , I am thinking on Code::Blocks but would rather decide if I have aptitude before installing software. I went and looked at some other basic beginners material to see if I could apply things from this tutorial in to my own idea. I needed random numbers for the idea, not sure on that part, perhaps you could have a look and tell me if this is about where a noob should be this far in the tutorial? I didn't comment a lot but it is likely obvious to you...

```
                    // simple math game for kids
                    // ten correct answers to win
#include <iostream>
#include <cstdlib>
#include <ctime>

int randomNum(x)
{
    std::srand(std::time(nullptr));  // uses clock to randomize
    int x = std::rand ();
    x = 1 + std::rand ()/(RAND_MAX =1u)/9);  // gets random number between 1-9
    return x;
}

bool mathTest (a,b)
{
    int answer {0};

    if (a>b){  //use number size to give addition or subtraction questions
        std::cout << a << "-" << b << "=";
        answer = a-b;
            }
    else{
        std::cout << a << "+" << b << "=";
        answer = a+b;
        }
    int guess {0};
    std::cin >> guess;
    if (answer == guess){
    return true ;
                        }
        return false;
}

int main()
{
    int rightAnswers {1};
    std::cout << " get ten right answers to win " << std::endl;
    while (rightAnswers <= 10){ //cycles game till ten right answers given
        a=randomNum(x);
        b=randomNum(x);
        bool test = mathTest (a,b)
        if (test){
```

```
            std::cout << "  correct!" << std::endl;
            rightAnswers++;
                 }
        else {
            std::cout << "  incorrect!  try agian" << std::endl;
             }

                                 }
    std::cout >> " you win !";
     return 0;
}
```

nascardriver
February 26, 2019 at 12:28 am · Reply

There are multiple errors in your code. Try to compile it (You don't need to download anything, you can do it online), fix the errors and auto-format your code.
For being written without tests, the code looks good for a beginner.

Arthur
February 26, 2019 at 2:30 am · Reply

thanks again, I tried that online compiler you pointed me to, it helped , I got it to run but will need to review material and read some more on formatting the auto function on that site just moved all of the curly brackets. I would probably get more random questions having the child enter one of the numbers.

```
                        // simple math game for kids
                        // ten correct answers to win
#include <iostream>
#include <cstdlib>
#include <ctime>

int randomNum()
{
    int x = std::rand ();
    x = 1 + std::rand()/((RAND_MAX + 1u)/6);  // gets random number bet
ween 1-9
    return x;
}



int main()
{
    int rightAnswers {1};
    std::cout << " get ten right answers to win " << std::endl;
    while (rightAnswers <= 10){ //cycles game till ten right answers gi
ven
        int a{0};
        a=randomNum();
        int b{0};
        b=randomNum();

        int answer {0};


```

```
    if (a>b){  //use number size to give addition or subtraction questi
ons
        std::cout << a << "-" << b << "=";
        answer = a-b;
            }
    else{
        std::cout << a << "+" << b << "=";
        answer = a+b;
        }
    int guess {0};
    std::cin >> guess;
    int test{0};
    if (answer == guess){

    test=1 ;
                        }
    else {
        test=0;
         }
        if (test==1){
            std::cout << "  correct!" << std::endl;
            rightAnswers++;
                    }
        else {
            std::cout << "  incorrect!  try agian" << std::endl;
             }
             rightAnswers = rightAnswers + 0;
                                }
    std::cout << " you win !";
     return 0;
}
```

nascardriver
February 26, 2019 at 2:39 am · Reply

Formatted code for line references:

```
// simple math game for kids
// ten correct answers to win
#include <cstdlib>
#include <ctime>
#include <iostream>

int randomNum()
{
  int x = std::rand();
  x = 1 + std::rand() / ((RAND_MAX + 1u) / 6); // gets random numbe
r between 1-9
  return x;
}

int main()
{
  int rightAnswers{ 1 };
  std::cout << " get ten right answers to win " << std::endl;
  while (rightAnswers <= 10)
  { //cycles game till ten right answers given
    int a{ 0 };
    a = randomNum();
    int b{ 0 };
    b = randomNum();
```

```
25
26      int answer{ 0 };
27
28      if (a > b)
29      { //use number size to give addition or subtraction questions
30        std::cout << a << "-" << b << "=";
31        answer = a - b;
32      }
33      else
34      {
35        std::cout << a << "+" << b << "=";
36        answer = a + b;
37      }
38      int guess{ 0 };
39      std::cin >> guess;
40      int test{ 0 };
41      if (answer == guess)
42      {
43
44        test = 1;
45      }
46      else
47      {
48        test = 0;
49      }
50      if (test == 1)
51      {
52        std::cout << "  correct!" << std::endl;
53        rightAnswers++;
54      }
55      else
56      {
57        std::cout << "  incorrect!  try agian" << std::endl;
58      }
59      rightAnswers = rightAnswers + 0;
60    }
61    std::cout << " you win !";
62    return 0;
  }
```

* Line 9: Use braced initializer lists.
* Line 9: Unnecessary call to @std::rand.
* Missing call to @std::srand.
* Line 10: Signed/unsigned conversion
* Line 18: Should be a for-loop.
* Use '\n' instead of @std::endl, unless you have a reason to use @std::endl.
* Line 20-23: Favor initialization over assignment.
* @test should be a bool.
* Use ++prefix unless you need postfix++.
* Line 58 doesn't do anything.
* Missing printed trailing line feed.

Arthur
February 26, 2019 at 3:14 pm · Reply

ok I think I covered all of the issues , you suggested. I simplified 'randomNum' , tried to get rid of unnecessary things and gave more specific variable names. to learn the idea of this chapter test would at the

end of line 33 I be looking to pass 'answer' to a third function and return a bool true/false to line 42?

```
// simple math game for kids
// ten correct answers to win
#include <cstdlib>
#include <ctime>
#include <iostream>

int randomNum()      // gets random number between 1-10
{
  return 1 + std::rand() / ((RAND_MAX + 1u) / 10);
}

int main()
{
  std::srand(time(NULL));

  std::cout << " get ten right answers to win " << std::endl;
  for ( int rightAnswers = 0 ; rightAnswers < 10; )
  {                    //cycles game till ten right answers giv
en

    int numOne {randomNum()};
    int numTwo {randomNum()};

    int answer{ 0 };
    if (numOne > numTwo) //determines addition or subtraction q
uestions
    {
      std::cout << numOne << "-" << numTwo << "=";
      answer = numOne - numTwo;
    }
    else
    {
      std::cout << numOne << "+" << numTwo << "=";
      answer = numOne + numTwo;
    }
    int guess{ 0 };
    std::cin >> guess;  //get answer from user

    bool test{ false };
    if (answer == guess)  // checks user answer
    {
      test = true;
    }
        if (test)
    {
      std::cout << "  correct! \n";
      ++rightAnswers;
    }
    else
    {
      std::cout << "  incorrect!  try again \n";
    }
  }
  std::cout << " you win ! \n";
  return 0;
}
```

Arthur
February 26, 2019 at 5:28 pm · Reply

here is what I got for putting this into the files in question 3.

io.cpp

```cpp
#include <cstdlib>
#include <ctime>
#include <iostream>

int randomNum()      // gets random number between 1-10
{
  return 1 + std::rand() / ((RAND_MAX + 1u) / 10);
}


bool mathTest (int numOne,int numTwo)
{
    int answer{ 0 };
    if (numOne > numTwo) //determines addition or subtracti
on questions
    {
      std::cout << numOne << "-" << numTwo << "=";
      answer = numOne - numTwo;
    }
    else
    {
      std::cout << numOne << "+" << numTwo << "=";
      answer = numOne + numTwo;
    }
    int guess {0};
    std::cin >> guess;  //get answer from user

    if (answer == guess)  // checks user answer
    {
        std::cout << "  correct! \n";
        return true;
    }
    std::cout << "  incorrect!  try again \n";
    return false;
}
```

io.h

```cpp
#ifndef IO_H
#define IO_H

int randomNum();
bool mathTest (int numOne,int numTwo);

#endif
```

main.cpp

```cpp
 #include "io.h"
#include <cstdlib> // would not work
#include <ctime>   // unless I included
#include <iostream>// these in main cpp ?

int main()
{
  std::srand(time(NULL));
```

```
  std::cout << " get ten right answers to win \n";

  for ( int rightAnswers = 0 ; rightAnswers < 10; )
  {                    //cycles game till ten right answers
given

    int numOne {randomNum()};
    int numTwo {randomNum()};

    bool test {mathTest(numOne,numTwo)};
    if (test)
    {
      ++rightAnswers;
    }

  }
  std::cout << " you win ! \n";
  return 0;
}
```

Arthur
February 26, 2019 at 8:54 pm · Reply

thanks @nascardriver the online compiler worked good, and I appreciate the feedback. I think this one is what the tutorial is teaching. thanks @ Alex for the tutorial.
io.cpp

```
#include <cstdlib>
#include <ctime>
#include <iostream>

void gameStart(int finish)
{
      std::srand(time(NULL));
      std::cout << " 0 answers \n";
      std::cout << " get " << finish <<" correct to win
\n";
}
int randomNum()      // gets random number between 1-10
{
  return 1 + std::rand() / ((RAND_MAX + 1u) / 10);
}

bool mathTest (int numOne,int numTwo)
{
    int answer{0};
    if (numOne > numTwo) //determines addition or subtr
action questions
    {
      std::cout << numOne << "-" << numTwo << "=";
      answer = numOne - numTwo;
    }
    else
    {
      std::cout << numOne << "+" << numTwo << "=";
      answer = numOne + numTwo;
    }
    int guess {0};
    std::cin >> guess;  //get answer from user
```

```
33
34      if (answer == guess)  // checks user answer
35      {
36          std::cout << "  correct! \n";
37          return true;
38      }
39      std::cout << "  incorrect!  try again \n";
40      return false;
41  }
42  void gameOver(int rightAnswers)
43  {
44      std::cout << rightAnswers << " correct answers \n";
        std::cout << "      you win ! \n";
    }
```

io.h

```
1  #ifndef IO_H
2  #define IO_H
3
4  void gameStart(int turn);
5  int randomNum();
6  bool mathTest (int numOne,int numTwo);
7  void gameOver(int rightAnswers);
8
9  #endif
```

main.cpp

```
1   #include "io.h"
2
3  int main()
4  {
5  int finish {10};
6
7      gameStart(finish);
8
9    for ( int rightAnswers = 0 ; rightAnswers < finish; )
10   {
11     int numOne {randomNum()};
12     int numTwo {randomNum()};
13
14     bool test {mathTest(numOne,numTwo)};
15     if (test)
16     {
17       ++rightAnswers;
18     }
19
20   }
21     gameOver(finish);
22   return 0;
23 }
```

nascardriver
February 27, 2019 at 3:51 am · Reply

* Line 17: Initialize your variables with brace initializers. You used copy initialization.
* Line 14: Use std::rand instead of it's global-namespaced counterpart.
* Line 14: Use std::time instead of it's global-namespaced counterpart.
* Line 9: Signed/unsigned conversion. Let's ignore this until casts have been

covered.
* Line 14: Use @nullptr instead of @NULL.
* The while-loop was better. Sorry, I didn't go through your logic before.
* Still poor formatting. I hope this gets better once you use a code editor if you decide to keep going.
* @test is unnecessary. You can replace line 42 with line 38.
* Use '\n' unless you have a reason to use @std::endl.
* Unnecessary spaces before printed line breaks.

File separation for quiz 3 looks good. "io" doesn't fit the file's contents.

WSLaFleur
February 16, 2019 at 11:17 pm · Reply

So, here's my code for Quiz time, Question #1:

```
#include <iostream>
#include "Header.h"

// This tiny program prompts the user for two integers and adds them together.

void writeAnswer(int num1_and_num2)
{
    std::cout << "\n" << "Adding those two integers equals " << num1_and_num2 << "!" <<
"\n\n\n";
}

int readNumber()
{
    int num{};
    std::cout << "We're going to add integers! Please enter an integer: ";
    std::cin >> num;
    return num;
}

int main()
{
    bool programRunning{ true };
        while (programRunning == true)
        {
            system("cls");
            int num1{ readNumber() }, num2{ readNumber() };
            writeAnswer(num1 + num2);
            std::cin.ignore(2);
        }
        return 0;
}
```

When I spoiled the answer, I was surprised that you weren't using the initialization method you described for the variables (i.e. {}). Is this just something that wasn't updated to reflect changes to the core material?

Also, I don't really understand what I'm doing with std::cin.ignore(2) - the program is running as intended, but unless I feed the cin.ignore function a value greater than 1, it fails to pause. If you're not too busy to explain, then do you know why this is happening?

nascardriver
February 17, 2019 at 4:05 am · Reply

Hi!

> I was surprised that you weren't using the initialization method you described for the variables
Alex didn't update most of the lessons yet. Chapter D and onward use copy- and direct initialization.

* Line 8: Limit your lines to 80 characters in length for better readability on small displays.
* Line 22: Don't compare booleans to false/true.
* Use the auto-formatting feature of your editor.
* Line 25: Declare one variable per line.
* Don't use @system.
* Line 13: Initialize to a specific value (0).

@std::cin.ignore(count, delim) ignores all characters until delim is found or count characters have been ignored.
By passing 1 you're ignoring at most 1 character. Since the only character left by @readNumber is a line feed (Because you press enter and @std::cin::operator>> doesn't remove the line feed), this line feed is removed and execution continues.
If you pass 2 and the only character is a line feed, @std::cin.ignore will halt until a second character has been entered.
Pass @std::numeric_limits<std::streamsize>::max() and '\n':

```
1 // This will halt until enter is pressed
2 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

If you don't understand a standard function, you can look them up in a documentation:
https://en.cppreference.com/w/cpp/io/basic_istream/ignore
These might be hard to understand just yet, but once you know C++, a good documentation is priceless.

WSLaFleur
February 17, 2019 at 7:44 pm · Reply

Thanks for the reply, I've been using cppreference to dig into any syntax I'm unclear or just curious about, but I'm unfamiliar with concepts like 'delim', which makes understanding the full context difficult sometimes.

So I tend to squirrel down a rabbit hole of syntax links for a while and then return to writing my code after a bit. I realize we haven't hit booleans yet, I'll definitely adopt best practices for them whenever we arrive.

Thanks again!

Alex
February 18, 2019 at 8:12 pm · Reply

delim is just a parameter name for the delimiter character that you can pass in if you don't want to use the default value of EOF.

Suyash
February 8, 2019 at 10:35 pm · Reply

#include <iostream>
using namespace std;

int readNumber();
int writeAnswer();

int main()
{

```
    cout<<"Welcome "<<endl;
    cout<<"Enter your number "<<endl;
    int a=readNumber();//Unused variable
    cout<<"Enter the other number "<<endl;
    int b=readNumber();//Unused variable
    cout<<"The total is "<<writeAnswer();
    return 0;
}
int readNumber()
{
    int x;
    cin>>x;
    return x;
}
int writeAnswer()
{
    int a,b;//uninitialized variable
    return a+b;
}

/*
Hello Alex,
I wrote this after reading all three of the quiz questions, therefore I used declarations so as to make it easy for me for the next two. It compiles, but shows four  warnings, stating 1. Unused variables, 2.Uninitialized variable for each a&b. I modified writeAnswer (And its declaration) to:
void writeAnswer();
void writeAnswer()
{
    int a;//uninitialized variable
    cout<<a;
}
but then it shows an error saying:
too many arguments to function 'void writeAnswer()'
but then I initialised it in my declaration. so the other two errors are left
no match for 'operator<<'
and whole lot of ostream errors.

Thank you! for the tutorials
*/
```

nascardriver
February 9, 2019 at 7:37 am · Reply

Hi!

Functions cannot see variables declared inside other functions. @writeAnswer doesn't know about @a and @b in @main. You have to pass them as arguments.
@a and @b in @main get assigned a value, but after that, they're never used.
If @writeAnswer is declared

```
1 void writeAnswer(/*...*/)
```

the "void" means that it doesn't return a value. But in @main you're trying to print the value returned by @writeAnswer. This value doesn't exist, so you get an error.
Please use code tags (Yellow box below the comment text field) and exact error messages.