FAQs for coding in C++

Include library

Q: What does `#include <iostream>` do?
Q: Do we need to put semicolon (;) right after `#include <iostream>`?
Q: Can we include more than one library?

namespace

Q: What is `using namespace std;` ?
Q: Can the semicolon (;) in `using namespace std;` be omitted?

comments

Q: What do comments do?
Q: Since comments are not actual code and ignored by compiler, can we skip writing comments?
Q: What should be included in comments?
Q: What is comment requirement for CSCI 13500?

main method

Q: What does line int main() mean?
Q: Why is main method special?
Q: Can we replace int main() with void main() or just main()?

return statement

Q: What does return statement do?
Q: Do we always return 0 in main method?

A C++ code to print “Hello, World” to screen is as follows.

```
/*
   author: your name
   purpose: print "Hello, world!" to screen.
*/
#include <iostream>
using namespace std;
int main()
{
  //operator << pushes “Hello, world!” to cout.
  cout << "Hello, World!" << endl;
  return 0;
}
```

Q: What does `#include <iostream>` do?
A: It includes library iostream, which handles input and output stream, for example, input from keyboard, output to screen. Without this line, cout (character output) object will not work.

Q: Do we need to put semicolon (;) right after `#include <iostream>`?
A: No. closing angle bracket > is enough to separate include statement from the next statement.
To make things worse, if you accidently add ; after include statement as follows,

```
#include <iostream>;
```

A compilation warning is generated.

```
HelloWorld.cpp:1:20: warning: extra tokens at end of #include directive
      [-Wextra-tokens]
#include <iostream>;
                   ^
                   //
1 warning generated.
```

In short, do **not** add ; after include statement.

Q: Can we include more than one library?
A: Yes. If we want to use other functionalities provided in a specific library, then we introduce that library. For example, using #include <cstdlib> let us use rand method to generate a random integer.

Q: What is `using namespace std;` ?
A: Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

For example, cout and endl might be used in other scenario and thus have different meaning, but under std (means standard) namespace, cout and endl have their standard meaning, that is, cout means character output, and endl means endline.

Without statement
`using namespace std;`
You would have to replace each occurrence of cout with std::cout and every occurrence of endl with std::endl.

Q: Can the semicolon (;) in `using namespace std;` be omitted?
A: Semicolon (;) after using statement **cannot** be omitted, otherwise, compiler does not know that statement is finished, and your code cannot be compiled.

Q: What do comments do?
A: Contents (*not* code) enclosed in `/*` and `*/` or after `//` and until the end of line are called comments. Comments are not actual code and are completely ignored by compiler.

There are two types of comments.

- Multiple line comments are included in `/*` and `*/`.

```
/*
author: your name
purpose: print "Hello, world!" to screen.
*/
```

- Single line comment starts from `//` and ends at the end of line.
  `//operator << pushes “Hello, world!” to cout.`

- Since comments are not code, no need to end it with semicolon (;)

Q: Since comments are not actual code and ignored by compiler, can we skip writing comments?
A: Comments are necessary. In fact, professional programmers write meaningful yet concise comments. Without comments, your code might not be easily maintained by your fellow programmers, and vice versa.

Q: What should be included in comments?
A: The following information should be included in your comments: author, purpose of each key component methods (main method is one of them), sample input and/or sample output, and any key ideas or tricky parts not obvious to people.

If comments need to spread in multiple lines, use /* and */, otherwise, using //.

Q: What is comment requirement for CSCI 13500?
A: (courtesy Emira Hajj) For 135, I believe that automatic grading scripts involve checking the first 6 or so lines in the file. Every student should have this on top of every file they submit in order to get the full marks for comments:

```
/*
Author: your name
Course: CSCI-135
Instructor: their name
Assignment: title, e.g., Lab1A

Here, briefly, at least in one or a few sentences
describe what the program does.
*/
```

Students should also be adding comments to their code as they are going along. As the assignments get more complex they should just offer a brief description of the program up top and also use inline comments to clarify their code.

Q: What does line int main() mean?
A: It is the header of main method (aka declaration of main method), where **int** means return type, main is the name of a method, and () means no input parameter is needed.

Normally **int** main() is immediately followed by a pair of matched curly brackets { and }. The body of the method is enclosed in { and }. Inside that pair of { and }, codes are included. You may think { begins method body while } ends method body.

Warning: do not put a semicolon (;) after **int** main() if { is followed immediately. That is, do not put a semicolon (;) between int main() and {. Otherwise, semicolon (;) would have separated the header and body of main method and result in compilation error.

Q: Why is main method special?
A: Method main is special since it is the entry point of a C++ project, that is, the first statement of main method is the first code to run.
You may think a project as a company and methods are employees. Method main is like a CEO. In simplest situation, we have only one method in a C++ project and that method must be main.

Q: Can we replace int main() with void main() or just main()?
A: No. The return type of main method must be **int**. Otherwise, it cannot be the entry point method that we refer to in **Q: Why is main method special?**
Reserved word **void** means return type is empty or do not need to return anything.

For now, just use **int** main() for main method header. For more information, please refer to https://stackoverflow.com/questions/4207134/what-is-the-proper-declaration-of-main.

Q: What does return statement do?
A: Once a return statement is called, the current method finishes it running and return to the caller of current method.

Q: Do we always return 0 in main method?
A: Not necessarily. If we finish the code successfully, we would return 0, which implies everything is OK and the code finishes as expected. However, in case there is some error that we need to leave early, we would use a different integer other than 0, so that from that code the caller of main method can have some idea what problem it might have.

Which integer value to return depends on the user requirements. For example, suppose you write program for testing HP printer cartridge, if everything is OK, we would return 0, otherwise, we would return a corresponding error code.

Code	Issue and how to resolve it...
02	An issue most commonly associated with driver or printer cable problems, turning the printer off and on should see this error message disappear. A hardware problem may be the reason the error message persists.
10	Also known as "Supplies Memory Error", this code indicates that the machine is having difficulty with a toner cartridge chip in that it's unable to read it. Try resetting the printer initially and reinstall the toner cartridge if the problem persists. If these fail, it could be a hardware issue.

Here is a code to calculate the quotient of two integers. If denominator is zero, then print out "denominator cannot be zero" and return 1*, otherwise, calculate and print quotient, then return zero.

Based on the above setting, when the caller of this method sees 1, the caller knows denominator is zero.

* Suppose programmers decide to use digit 1 as an error code to indicate that denominator is zero and thus no quotient can be calculated.

```
#include <iostream>

/*
author: Tong Yi
purpose: enter two integers, and find out their quotient if
the denominator is not zero,
otherwise (if denominator is zero),
report error and stop the program early.

Sample input/output:
```

```
Enter a numerator: 5
Enter a denominator: 2
The quotient is 2

Enter a numerator: 5
Enter a denominator: 0
denominator cannot be zero.
*/
//multiple line comments start with /* and ends with */
//And there can be several lines between /* and */
//But // works from the first letter after // to the end of that line.
using namespace std; //; after this statement is needed.
   //anything after single-line commenter to the end of line
   //is treated as comments.
int main()
{
    cout << "Enter a numerator: "; //prompt user to enter a numeratorerator
       //without the prompt, the code also works,
       //but there will be some problems,
       //the cursor is waiting there,
       //the code in
       //int numerator;
       //cin >> numerator;
       //are waiting for input from keyboard,
       //but user does not know what to do without the prompt.
       //User-friendly.

    int numerator; //this line is like to register numerator as an int.
       //After this line, numerator has enough space to hold an int,
       //but what is the value of that integer is unknown/unset yet.
       //It is like you are registering into a college,
       //once you get registered, you get a room.
       //But what is inside that room is not known yet.
    cin >> numerator; //>> is called pull operator.
       //It is like the system pulls an int from cin (character input
       //or keyboard buffer) to variable numerator.

    cout << "Enter a denominator: ";
    int denominator; //declare an int variable denominator
    cin >> denominator;

    if (denominator == 0)
    {
       cout << "denominator cannot be zero.";
       return 1; //when denominator is 0, there is no need to continue.
           //we will stop running the rest code of main method
           //and return control to the caller of main method,
           //say, operating system.
    }

    //If we do not fall into the trap of
    //(denominator == 0) and thus need to leave early,
    //denominator != 0, where != means not equal
    cout << "The quotient is " << (numerator / denominator);
       //Integer division numerator / denominator,
       //suppose numerator is 5 and denominator is 2,
       //then quotient is like to divide 5 pens
       //among two students, find out
```

```
        //how many pens each student gets.
        //a pen cannot be divided.

    return 0;
}
```