The Terrifying Tiger

In week 1, as a fierce tiger called Fangs, your sabres bared as you bring the jungle to a halt with your fearsome roar, you will complete your first quest. You will write a program that must print the words "`Hello world`" on the screen.

In week 42, as a humble mouse called Peony, whose soft steps will tread the well worn trails of humble mice before her, you will quietly toil to find the maximum flow of grains from anywhere to anywhere in a vast universe of hundreds of thousands of tunnels.

You are embarking on the mega quest of learning C++ through my three-quarter sequence. This is Level BLUE, CS2A. It has nine quests you must complete in 12 weeks, in addition to some other tests you have to pass.

The secret password to unlock each quest will appear after you have completed a certain number (a secret, known only to a select few) of *miniquests*. For instance, you will unlock the password for quest 2 by completing this one. You may not share these quest passwords with other students. Completion of a quest only counts if all previous quests have also been completed. Thus it doesn't help to get the secret passwords to locked quests from your friends anyway.

In most quests, the total number of rewards won will usually be indicated by some number of objects you would have collected - fruits, nuts, birds, flowers, etc. You can win these by completing *miniquests*.

Some miniquests have hidden rewards.. You won't even see these until after you've won them. And the way to win the unseen rewards is focus on micro detail in the spec and make sure you've got it right. Your classmates and I can give you hints. But you have to find these treasures yourself. That's where the fun is. And in giving out hints too. That's also kinda fun.

Please don't publish cheat sheets! Be a good sport. Enjoy and leave it for others to enjoy just as you found it.

I have set up the rewards such that they can be won even after you have unlocked the next quest. Then you will have the option of advancing to the next quest as well as trying to eke more points out of the current one. You can even jump back and forth. When there are no more points to be won in a quest, I will usually tell you.

These quests start out small and easy, and progressively get harder and more detailed. In later quests you can expect to submit multiple files including header files that you will create. You will also have the opportunity to earn more rewards in the later quests.[1]

One other thing about these quests is that I will also be using them as instructional vehicles. Some information you will not find in the modules or your recommended reading may be present in your quest specs. Thus it is going to be important for you to carefully read the specs every time.

Your first quest - Hello world

It is **a trivial quest**. This means that you may as well copy your template code from most IDEs directly into your submission file and make minor adjustments to pass this quest. You will succeed.

Even though this is a trivial quest, it is important for you to complete it because:

1. It is meant to give you some practice in how you will submit future code.
2. You will only get the secret code to open up the next quest after you ace this one.

Once each week on average, you must implement your source code for a new quest in one or more files. Then when you have them ready, you will submit them into the questing site.

I will run tests on your code and report back with the rewards you scored in the quest. You can submit many times each week. The number of allowed submissions is high enough for you to be able to try several times, but it is limited. So try and avoid trivial submissions with frivolous modifications. These would simply eat up a resubmission opportunity. If you run out of chances to submit and can't reach me in time to request a couple more... yeah, terribly sorry 'bout that. Hope that doesn't happen again.

Also, be aware of when the scores for each quest get frozen. That is, submissions after this "freeze date" don't count towards your grade. These *freeze dates* will be given in the course syllabus.

[1] I think it's best to move on to the next quest as soon as you get the password for it and then keep coming back to improve earlier quests as and when you get time. In the past, many students *got stuck* at early quests when they could have easily advanced to later, funner ones.

Let's get started

Here's what your c++ program ought to look like:

```
0  //  Student ID: 12345678
1  //  TODO - Replace the number above with your actual Student ID
2
3  #include <iostream>
4
5  using namespace std;
6
7  // Feel free to use the alternate signature of main() below
8  int main(int argc, const char * argv[])
9  {
10    // TODO - Your code goes here
11
12    return 0;
13 }
```

Most weeks, I'll give you chunks of code like this in the spec that you can copy into your IDE or editor by typing in (not copy/paste). Then you will flesh them out by replacing the red TODO portions with your own code and testing them locally on your machine before submitting them.

The program in the box above is in the form of "source code". That is, as human-readable c++. The numbers in the left margin are for our reference below. They are not part of the actual source file.

Once you type in, save, and *compile* your source code, the compiler will turn it into binary machine instructions that will no longer be easily readable by you. But it will be easily readable by the cpu which will run it to produce the programmed effect. Normally, we observe this effect (the output) in many different situations and make sure they all make sense before letting the program go wild on its own.

Your source code should be saved in a text file named `hello_world.cpp`. When compiled and run, it should print the words "`Hello world`" followed by a single newline character.

Note that everything should match exactly: character casing, spacing, indentation, etc. It's very easy to get it right, so you have to get it exactly correct in order to ace this quest. Read the modules and the recommended reading, look up information on the Net, or discuss how to do this with your friends if necessary.

When your file is ready, you can submit it into my code-testing website (see below). If you pass my tests correctly, you will score rewards and also be given the secret password for your next quest.

Editor

Some students find it easy to use a simple editor like TextEdit (in plain ASCII mode) or Notepad for their initial labs. But it gets unwieldy pretty soon. I recommend that you graduate to using a

proper IDE like Xcode or VSCode at your earliest (ideally week 2 or soon thereafter). It will greatly simplify your programming life.

Style

I have stopped grading students on programming style because it is a subjective thing and I don't want to force my style upon you. But keep in mind that if you follow a messy coding style you will very quickly find that programming becomes a chore rather than a pastime as it should be. I recommend that you review and adopt the popular K&R style for C++ coding.

Overview of the provided template code

In C++ you can assume that your code is always presented in the form of functions. Functions are blocks of code that perform a sequence of actions. Functions can invoke each other and when they are done executing their code they will return control back to their *callers*, to the block of code that invoked the just-concluded function.

A function's behavior can be controlled and modified via special variables you pass into them called parameters. When a function is invoked, the run-time system sets things up such that the called function has access to variables passed to it by its invoker. For example, a function called `my_function` might have the following *signature:*

```
int my_function(int param1, string param2, string param3);
```

This means that when this function is invoked by a caller, they will need to supply three variables to provide values for the parameters. The types of these variables can be found in the function's signature. Also the **int** keyword before `my_function` indicates that when `my_function` is done executing, it will return a value back to the function that invoked it. The data type of this return value will be **int**.

A typical invocation of `my_function` might be like:

```
int return_value = my_function(n, name, address);
```

where n is an integer, and name and address are string variables.

Also, you may call the function by passing *literal values* instead of variables. E.g.

```
int return_value = my_function(2, "John", "123 First Ave");
```

We'll study all this in more detail in the coming weeks. For now, it suffices to understand what functions do at the level described.

Ever wonder how your program actually starts? If all you have are a bunch of functions, which function gets called first?

It is `main()`. `main()` is a special function. It is called by the run-time system when your program is invoked. Once your `main()` has started, your code starts running and everything is up to you from then on. you can call other functions you have defined, or simply exit.

You will notice that `main()` returns a value. What is it?

It is the value used to report the status of completion of your program back to the run-time system. Again, conventionally, we return 0 to say that there were no errors. (However, consider that in C and C++, zero is synonymous with `false` and non-zero with `true`. Why does a program return `false` upon successful completion? Discuss this in the forums).

Let's now walk through the individual lines of the template code.

```
1  // Student ID: 12345678
```

This is an inline comment. These comments start with two forward slashes and continue to the end of the line. Every program you submit should have this line at the very top. This is how I associate your lab score with your student record in Canvas. It's important that you don't mess up this line. Only replace the numeric portion with your own ID. The rest of it should stay.

Also, the IDs in the multiple files all better be the same. I'll select a random one of them to extract your ID.

```
3  #include <iostream>
```

This is called an include-file directive. `iostream` is called a *header file*. It tells the compiler to import certain variables, types and data that have been pre-created for you and placed in that file. For example, the `cout` variable that you will use in your code is in the `iostream` header file.

```
5  using namespace std;
```

This is a using-namespace directive. Remember how the iostream library provides a bunch of variables for you? They are all provided within a *namespace* called `std`. You can think of it as if all those variables had the letters `std::` prefixed before their names. Thus we can avoid the problem of system variables conflicting with our variables of the same name (if we have any).

However, this makes for cumbersome typing. So the "using namespace" directive was created. It tells the compiler to look up variables by prefixing their names with the namespace qualifier if local matches to their names can't be found.

For example, suppose you have "`using namespace std`" in your code, and you use a certain variable, say `endl,` in your program without actually declaring such a variable.

Normally the compiler would complain about undefined variables and throw in the towel at this point. But because of the using-namespace directive, it will look for a variable called `std::endl` (that is `endl` defined within the `std` namespace) before doing so.

```
8  int main(int argc, const char *argv[])
```

This is the beginning of the definition of the `main()` function. The signature suggests that it accepts two parameters. The first is of type integer and the second is an array of pointers to

characters. Now since `main()` is invoked by the operating system, you don't really have to worry too much about these parameters at this point. They are called command-line arguments and will be supplied by the OS. You don't need to use these parameters in any of our quests in 2a. Feel free to look up information about them, or ignore them for now. FWIW, you can also declare main as:

```
8  int main()
```

In this alternate form, it doesn't take any parameters. The run-time system can handle either case correctly. We didn't use this form in our function.

Following main is a block of code enclosed in curly braces. This will contain the entire logic of your program. If you have other functions helping out your main(), they are only useful insofar as they are either invoked directly or indirectly (via another function that is likewise invoked by main) from main.

```
10      // TODO - Your code goes here
11
12      return 0;
```

Your code to print "Hello world" followed by a newline ought to go where the red TODO marker is.

The "`return 0;`" statement sends the exit status 0 back to the run-time system.

Submission

When you think you're happy with your code and it passes all your own tests, it is time to see if it will also pass mine.

1. Head over to https://quests.nonlinearmedia.org
2. Enter the secret code for this quest in the box.
3. Drag and drop your `hello_world.cpp` file into the button and press it. Make sure your filename is exactly as above and that it has your Student ID.
4. Wait for me to complete my tests and report back (usually a minute or less).

Points and Extra Credit Opportunities

I monitor the discussion forums closely and award extra credit points for well-thought out and helpful discussions.

Happy hacking,

&