Basic C++ layout and syntax

- just a very preliminary look at the general structure of C++ code
- just want to start giving folks the look/feel, don't worry too much about the gory details just yet
- assuming this is brand new to folks
- if you've already got C++ experience then treat the first half of the course as easy marks, but attend and keep up anyway (lots of people in the past have realized too late that I'd moved into things they *didn't* know yet, and struggled to catch up)

Overall structure

- I'll treat the structure of a typical program as 5 parts (not all programs will actually use all the parts)
- instructions to the compiler (preprocessor directives)
- definitions for any special data types/values used
- short prototypes for any subroutines we'll create/use
- the main routine (guides the program execution)
- full definitions of our created subroutines

In the beginning...

- for a few weeks we'll just be using a subset of those parts
- the instructions to the compiler will just be a list of the existing code libraries we want to include for use
- we won't define any special data types, though we may define some special constant values we want to use
- we won't be creating any subroutines

A quick example

```
// my first program

#include <cstdio>

int main()
{
   printf(“Hey, it works!\n”);
   return 0;
}
```

C++ syntax

- again, the syntax (grammar) rules for C++ are much simpler than the rules for most natural languages, but the compiler will enforce them very strictly
- if your program isn't exactly correct grammatically then the compiler will generate one or more error messages and will not produce a new executable
- we'll discuss the syntax rules for each new language feature we introduce

A few broad rules

- to use items from an existing C++ library we must #include the library (cstdio is the C standard input/output library)
- a program must include a main routine, where the program execution actually begins
- the statements in a routine (like main) are grouped together within the { }
- individual statements are generally separated by semicolons
- words in the language are case sensitive, e.g. Main and main are NOT the same thing

Edit/compile/execute reminder

- we would write the program using an editor, e.g. creating a file named firstprog.cpp
- we would use the g++ compiler to translate this, creating a new executable, e.g.

```
g++ firstprog.cpp -o firstprogx
```

- if it compiled cleanly, we could run it using

```
./firstprogx
```

- the results of the run should be something like

```
Hey, it works!
```