Getting Started - Set your Xcode preferences

- Generally when coding it helps to have line numbers so we can talk about what code we're looking at and where. Go into Xcode preferences, select "Text Editing" and select the 1st check box "Line numbers".
- Note the "Default line endings" option. Windows uses different line endings from Unix, which can really mess things up. The old Mac OS also used different line endings. In general it's best to keep with Unix line endings.
- Under the "Indentation" tab you can set it to use either tabs or spaces. Some people prefer to indent with tabs rather than spaces. Whole wars have almost started over if one should use tabs or spaces. Some systems will default tabs to be 8 wide while others 4 wide.

Alternative to Xcode

A free alternative to Xcode on Mac (and elsewhere) is the recently released Microsoft Visual Studio Code, which more closely resembles Microsoft Visual C++ as it's part of the Visual Studio suite.

Currently Visual Studio Code may be downloaded from https://code.visualstudio.com

Examining Hello World

Here is your first complete C++ program.

```
//
//  hello.cpp
//   my 1st C++ program
//

#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World" << endl;

    return 0;
}
```

Comments

- The opening portion of the program, the part with the `//` lines, these are just *comments* (also known as *remarks*).
- Comments have no influence on the way the program runs.

```
//  hello.cpp
//   my 1st C++ program
```

- Whenever there are two slashes together (`//`) it tells the compiler to ignore the rest of the line. The purpose of comments is to help make the code more understandable.
- Comments may be written in one of two ways.

```
// comment
```
or
```
/* comment */
```

- With a `//` comment, everything after the `//` is ignored, e.g.
```
// foobar is a placeholder name often used in programming
```
- This type of comment is a “single line” comment.
- Note though, that anything before the `//` is still counted
```
cout << "foobar";  // foobar is a placeholder name
```
- The other form of comment, `/* */` allows you to spread a comment over many lines.
```
/* foobar is a placeholder name often used in programming.
The etymology of foo is obscure, its use in connection with bar
is generally traced to the World War II military slang FUBAR,
later bowdlerised to foobar. */
```
- Anything between the `/*` and `*/` is ignored by the compiler.
- The `/* */` comment is a good way of omitting (“commenting out”) chunks of code.
- This type of comment may also be used to embed a comment anywhere in the code, even within a single line e.g.
```
cout << /* prints Hello World */ "Hello World" << endl;
```
- Generally, embedding a comment in the middle of a line is confusing, though sometimes can be useful.
- Comments are purely for making your code more understandable.
- Good programming is all about good commenting.
- Often comments are really written for yourself. When returning to code after many months, or even years, it is easy to forget exactly why you were programming a certain way.
- `/* */` style comments are the “original” C style comment, and comes from C.
- `//` style comments were introduced with C++, and wont work with some older C compilers.
- People often use a mixture of both styles of comments, using // comments most of the time, but `/* */` style comments for omitting blocks of code, or when necessary to embed a comment in a long line of code.

Main

- The `main()` function is called at startup, and is the designated entry point for C++ programs.
- All C++ programs, or at least all those we will be looking at, must have a `main()`. It’s very rare for a C++ program not to have a `main()`.
- The area between the `{ }` is, quite literally, the main launch area of the computer program.
- `main()` is a *function*, and functions will be talked about in greater depth in the chapter on Functions.
- Many C++ compilers require `main()` to have a `return 0;` at the end before the closing `}`. Reasons for this will become more apparent when we look at *functions*.
- For now, be content to write your program in the area between the `{ }`.

Introduction to Strings

- Text, like `"Hello World"` is generally stored by C++ in what is known as a `string`.
- A `string` is a sequence of characters. In general, a character is a symbol that you can type from the keyboard.
- The text of a string is always enclosed within double quotation marks ("").

- Be careful, some text editors will try to convert your double quotation marks to “smart quotes.” Smart quotation marks will not work when programming, and will cause an error when you compile your program. When programming, always use regular quotation marks.
- `string`, although “standard” in C++, is not an innate part of the C++ language. The `string` we use today evolved over time, and there are other older versions and implementations of `string`. C has it’s own “C style” string, that we will look at later, and is often still used in C++ programs. Microsoft, at one time, had its own version of `string`.
- The capability to use `string` in C++ is provided through the C++ Standard Library.
- This Standard Library changes and grows with the C++ language.
- To use the C++ Standard Library we use the keywords `using namespace std;` at the beginning of the C++ program, before `main()`.
- Strings will be looked into in more depth later, but for now, suffice to know that your text is stored in a `string`, and to use strings you must include `using namespace std;` at the beginning of your program.

The ;

- The `;` is a terminal symbol is C++.
- It’s a token that finishes something. What exactly it finishes depends on the context.
- Every expression of code in your program, needs to terminate with a `;` otherwise the compiler won’t know where the expression finishes, and will give an error.

Escape Characters

- You cannot express characters such as quotation marks and line breaks literally in a string.
- If you include quotation marks as part of a string, the compiler will think that the quotation mark indicates the end of the string.
- For example, to create a string quoting someone, you might write,

```
"Churchill said, "If you're going through hell, keep going"."
```

- But the compiler will interpret this as two separate strings.
- The solution is to use escape characters.
- Escape characters are used to represent characters that cannot be represented literally.
- To create an escape character add a backslash (`\`) before the particular character you wish to display.

```
"Churchill said, \"If you're going through hell, keep going\"."
```

- This will produce the desired string.
- Important escape characters you need to know are,

new line	`\n`
tab	`\t`
backslash	`\\`
single quote	`\'`
double quote	`\"`

Displaying Strings with cout

- To display a string in C++ we use a command called `cout`.
- `cout` is used to print strings and other data to the screen.
- Like `string`, `cout` is not an innate part of C++, and is actually part of a (very useful) library called the Standard Input/Output Streams Library.

- To use `cout` and the Standard Input/Output Streams Library in your program you must use the keywords `#include <iostream>` at the beginning of the C++ program, before `main()`.
- The Streams Library is also used to read and write files external to your program, and is discussed in greater depth in the *Streams and Files* portion of this course.
- To display a text string using `cout` use the `<<` characters, e.g,

```
cout << "Hello";
```

- This will print `Hello` to your terminal.

```
cout << "Hello";
cout << " World";
```

- This will print `Hello World` to your terminal.
- Notice though, there is no new line between `Hello` and `World` in this case. The stream does not see the break between the `cout` calls, even though they are on different lines.
- With `cout` you may use `endl` to denote a new line.
- The `endl` tells the stream there is an end to the line to print.

```
cout << "Hello" << endl;
cout << "World" << endl;
```

- Would print

```
Hello
World
```

With strings you may also use escape characters to denote a new line.

```
cout << "Hello\n";
cout << "World\n";
```

- Would also print

```
Hello
World
```

Storing Strings

- You use a what is called a variable (a “bucket” of information) to store a string.
- You store a string by putting it into memory as a variable.
- When you store a string you must provide a name for the variable.

```
string str = "A dragon is coming.";
```

- Use the name of the variable to display the stored string when you want to print it, e.g,

```
cout << str << endl;
```

- You should experiment displaying strings using variables in `main()` using `cout`.

ALL of your programs in this programming class should have,

```
#include <iostream>
using namespace std;

int main()
{
      // your program here!

      return 0;
}
```