Tutorial 2: Intro to C++

Review: C++ Intro

What are the differences between C++ and C?

C++ provides all features available in C (C programs will still compile with the GCC C++ compiler). Additional language features include (we'll explore these in more detail as the course progresses):

- classes
- templates
- operator/method overloading
- references
- exceptions
- namespaces
- runtime type identification

C++11/14 also provides extra features such as (again, we'll explore these in detail as the course progresses):

- auto type inference
- foreach loops
- lambda functions
- nullptr value
- different pointer types

As of C++11 it is also recommended to use #pragma once as an include guard, rather than the #ifndef,#define,#endif combination.

In addition to the language features, there are also library features, which includes all default headers available in C, as well as the C++ Standard Library, which includes:

- iostreams
- strings
- containers
- algorithms

C++11 also provides some additions to the standard library, such as:

- more containers
- tuples
- threads
- regex

What are the differences between C++ and Java?

In addition to some features in the above list, the main differences between C++ and Java are that C++ allows functions not attached to classes and it allows multiple inheritance (it also gives you the choice between static and dynamic binding when using inheritance). Unlike Java the Standard Library for C++ is relatively small and usually you would look for different C++ libraries to perform whatever task you need (the largest C++ libraries are boost and QT) as opposed to Java where you typically use whatever functionality is available in the default libraries. C++ also has no garbage collector thread and as of C++11 it relies on its different pointer types to perform garbage collection. C++ code is compiled into native machine code, as opposed to Java which compiles into bytecode.

Finally it's worth noting that due to C++ providing the performance of C, along with easier development that comes with its language features, both Java and GCC are written in C++.

How can object oriented programming be used in C++?

Object Oriented Programming means programming in terms of objects and their interactions rather than functions. C++ supports OOP through its use of classes, inheritance/polymorphism and dynamic binding. You will learn more about these features later.

How can generic programming be used in C++?

Generic programming means programming using data where we don't care what type that data or object is. C++ supports generic programming through its use of templates. In fact, templates in C++ are turing complete which means code can be written using templates and executed at compile time rather than runtime. This is called template metaprogramming. You will learn more about templates later.

Programming Concepts: Streams

Look at hello_world.cpp and input.cpp from the Week1 examples folder as a reference.

What is iostream?

iostream is a header file which provides streams for console io.

What is a stream?

A stream is a buffer of data that (in this context) sends its data once its buffer is full. A stream can either be an output stream (sending data, ie from your program to the console using the <<

operator) or an input stream (receiving data using the >> operator). C++ provides the same streams as C (stdout as std::cout, stdin as std::cin and stderr as std::cerr). File streams are also available for writing/reading data from files. The << operator is called the stream insertion operator and >> is called the stream extraction operator. Note that C++ still allows bit shifting using the same syntax, and the compiler will choose whether to use the bitwise or stream operator based on whether a stream is being used or not.

What is std?

std is the standard namespace for C++ which contains the standard library. When using a function or container from the standard library, you must either declare you are using std, declare you are using that part of the std, or use the std:: prefix.

What is boost?

Boost is a third party library that is often used with c++. Features that will eventually make their way into the c++ standard library are often tested in boost and so there can be a lot of overlap between boost and the standard library. You can find out more information here: http://www.boost.org/

What is cout/cerr and how can it be used to output different data types?

cout can be used to output any primitive data type as well as C-style strings (char arrays) and C++ strings using the stream insertion operator:

```
cout << “Hello World\n”;
```

You can also output complex data types (such as user defined classes) provided the stream operators of that class have been overloaded (you will learn more about this later).

What is cin and how can it be used to input data?

cin can be used to retrieve data from the console using the stream extraction operator:

```
int x, y;
cin >> x >> y;
```

cin will automatically attempt to convert any data read from the console to the correct type (in the above case, int) and it will stop as soon as any invalid data is read (ie, if x is entered as a string, y will not be read). cin will separate different data types to be entered using whitespace.

How can data be read from/written to files using streams?

Data can be read from/written to files using the same stream operations as cout/cin. To use file streams you must first include the correct header:

```
#include <fstream>
```

You can then open a file by simply creating the appropriate stream. To create a stream to read data from a file you would use ifstream:

```
std::ifstream fstr(filename);
```

And to create a stream to write data to a file:

```
std::ofstream fstr(filename);
```

It is also a good idea to check if the file was opened correctly using the method `is_open()`

You can then read and write data using stream operations, or using the `read()` and `write()` methods on the stream object. Finally, you can close the stream using the `close()` method.

Compilation: C++11

How do we compile a C++ program to use C++14?

To compile a program using C++11 features, the C++11 std flag must be included, eg:

```
g++ foo.cpp -Wall -pedantic -std=c++14 -o foo
```

On the uni servers (jupiter/saturn/titan) the default C++ compiler version does support C++14, so you need to use a newer version of g++. To do this you can run the command:

```
source /opt/rh/devtoolset-6/enable
```

Errors: Warnings

What is a compiler warning and how do we check for them?

A compiler warning is anything the compiler finds in your code that it thinks could potentially cause problems, but doesn't produce any errors that prevent the program from compiling (eg,

nothing that prevents construction of the syntax tree). Not all warnings are reported by default and if you want to ensure that they are, you should use the following flags:

-Wall (enables most, but not all, compile warnings)
-Wextra (enables more warnings not covered by -Wall)
-pedantic (enforces standards compliance by warning about use of platform specific functions)

If you want your program to stop compiling when it encounters a warning or error (to prevent large screens of warnings and make it easier to tackle them one at a time), use the flags:

```
-Wfatal_errors -Werror
```

What are some common examples of compiler warnings that will crash your program?

An easy compile warning to introduce is to declare a function that returns a particular data type, and then not return anything within that function, eg:

```
int warning() { /* Not returning anything here */ }
```

This will only be reported when enabling warnings and can crash your program or cause other problems if your program relies on the data being returned from that function, eg:

```
int array[10];
array[warning()] = 1;
```

Debugging: None this week

Exercises:

Write a function that reads ints from the terminal and stores them in a text file.

```
void writeData(std::string filename)
{
	std::ofstream fstr(filename);
	if (!fstr.is_open())
		return;

	int i;
	while (std::cin >> i)
```

```
                fstr << i << " ";

        fstr.close();
}
```

This will continue accepting input until any non-integer is encountered.

How would this function be changed if we were going to write the data as binary instead of ascii?

```
void writeData(std::string filename)
{
        std::ofstream fstr(filename, std::ios::out | std::ios::binary);
        if (!fstr)
                return;

        int i;
        while (std::cin >> i)
                fstr.write((char*)&i, sizeof(int));
        fstr.close();
}
```

Write a function that reads binary data from a file, assuming the file first stores the number of ints in the file, followed by that number of ints, and then prints them to the terminal.

```
void readData(std::string filename)
{
	std::ifstream fstr(filename, std::ios::binary);
	if (!fstr.is_open())
		return;
	int size;
	fstr.read((char*) &size, sizeof(int));
	if(!fstr)
	{
		/* error!!!! */
	}
	int array[size];
	fstr.read((char*) array, sizeof(int) * size);

	fstr.close();
	for (int i = 0; i < size; i++)
		std::cout << array[i] << “, “;
}
```