C/C++ Programming
Session 1

Dr Jim Lupo
LSU/CCT HPC Enablement
jalupo@cct.lsu.edu
Resources

http://reu.cct.lsu.edu/documents/C_Course/index.php
Handouts, slide decks, sample programs for this course – just follow the links.

https://www.hpc.lsu.edu/training/index.php
Links to weekly training tutorial archives – slides & recordings

https://moodle.hpc.lsu.edu
HPC101 – Intro to HPC
HPC102 – Remote Access with SSH
HPC103 – Survival Skills with the TCSH Shell
HPC104 – Survival Skills with the BASH Shell
HPC106 – User Environment Training
HPC114 – Survival Skills for C Programmers
HPC115 – HPC Software Development
HPC116 – Survival Skills for Fortran Programmers
# Subject Overview

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun 12</td>
<td>Jumpstart</td>
<td>Jul 10</td>
<td>Functions</td>
</tr>
<tr>
<td>Jun 17</td>
<td>Basic CS</td>
<td>Jul 15</td>
<td>Arguments</td>
</tr>
<tr>
<td>Jun 19</td>
<td>Data Types</td>
<td>Jul 17</td>
<td>File I/O</td>
</tr>
<tr>
<td>Jun 24</td>
<td>Operators</td>
<td>Jul 22</td>
<td>Strings</td>
</tr>
<tr>
<td>Jul 1</td>
<td>Control</td>
<td>Jul --</td>
<td>Arrays</td>
</tr>
<tr>
<td>Jul 3</td>
<td>Nesting</td>
<td>Jul --</td>
<td>Objects</td>
</tr>
<tr>
<td>Jul 8</td>
<td>Structures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But, our milage may vary!

Meet in 1008B DMC from 10am – noon.
Introduction

There are many languages to use if you want to program on HPC (high performance computing) systems. Fortran and C/C++ are among the most popular for efficient code.

This series of talks will get you familiar with the C/C++ language. Once you understand the concepts, adapting to other languages is relatively straightforward.

Today is a jumpstart session – I'll use the proper terminology, but much you'll need to take on “faith” till we cover it in detail later.
Working with Editors

vim – it's the bomb

emacs – only real programmers use emacs

It's all about your editor religious beliefs!

notepad – it's for masochists
VIM Is Your Friend

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

int main ( void ) {
  float gallons;
  cout << "How many gallons of water? ";
  cin >> gallons;
  cout << "That would weight 
    << 8.0 * gallons
    << " pounds." << endl;
  return ( 0 );
}
```

"s1_b.C" 11L, 247C
Traditional First Program

1. #include <iostream>
2. using namespace std;
3. int main ( void ) {
4.   cout << "Hello World!\n";
5.   return ( 0 );
6. }

Note: "_" indicates a space.
Compile The Program

I'll be running the examples on Mike.

1. Create source file: $ vim hello.C
2. Compile: $ icpc hello.C
3. Execute result: $ ./a.out

Could do: $ icpc -o hello hello.C
           $ ./hello
Compilers

icc . . The Intel C compiler is the default C compiler.
icpc . . The Intel C++ compiler is the default C++ compiler.

gcc . . The GNU C compiler.
g++ . . The GNU C++ compiler

pgcc . . The Portland Group C compiler
pgCC . . The Portland Group C++ compiler

They all work the same way, unless you need to get fancy. Could do the exercises with most any Linux or Cygwin. Working with Microsoft Visual C++ is possible, but you push a button to make it compile.
Deconstructing: Source Syntax

Whitespace separates words and is ignored unless it is found between double quotes:

```c
int main ( void );
```

could just as well be:

```c
int main ( void );
```

A bit harder to read, but legal.
Whitespace in Strings

Whitespace? It consists of non-printing characters, such as a literal space, tab character, newline character, etc.

A string is a sequence of characters between double quotes.

```
"Hello World!\n"
```

It is *not* the same as (note extra whitespace):

```
"Hello  World!\n"
```
Deconstructing: Statements

Begins with a language keyword or object.

Ends with a semi-colon “;”

```c
cout << "Hello World!\n"; return (0);
```

Think of statements as the C language equivalent of a sentence in English.
Statement Blocks

{ } .. curly braces are used to delimit a group of statements.

In the example program, there is one statement block which is the body of the function `main`.

```c
int main(void){
    ... statements ...
}
```

Think of statement blocks as the C language equivalent of a paragraph in English.
Deconstructing: Header Files

1. `#include <iostream>`

Line 1: tells the compiler to reference a *header file*.

Header files contain definitions of variables and functions.

The hard part of C/C++ is learning how to find and use all the predefined facilities, libraries, and tools.

Investigate before you program!
System Header File

There are two ways to specify a header file, both basically tell the compiler where to look for the file:

```c
#include <iostream>
```

The `"< >"` construct tells the compiler to look in a known system location for the file `"iostream"`.

```c
#include "myheader.h"
```

Tell compiler looks in same directory as the source file for a file named `"myheader.h"`, or in locations specified by `-I (include)` options (i.e.

```c
icpc -I /my/custom/files foo.C
```
Header File Contents

There is nothing fancy about header file contents. The contents are read in and compiled just as if they had been typed in the source file.

Assume we had a file named `hello.h` which contained:

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

How could we rewrite our sample program?
Revised First Program

1. #include <iostream>
2. using namespace std;
3. int main (void) {
4. #include "hello.h"
5.    return (0);
6. }

C/C++ For REU Participants
Session 1
Deconstruction: Namespaces

Namespaces are a way of keeping multiple developers from stepping on each other by accidentally using the same names for different things. Note that:

```cpp
use namespace std;
cout << "Hello_World!\n";
```

is the same as explicitly using:

```cpp
std::cout << "Hello_World!\n";
```

where `::` is the scope resolution operator.
The “Main Line”

```c
int main ( void )

This statement marks the start of the program. It's a function and every C/C++ program must have exactly one main function. Since it's the first user code that begins executing, and often the last when the program exits, it is commonly called the main line. Other languages have similar constructs. **There must be only 1!**

```c
void .. means that some optional elements have been purposely, and legally, left out.

```c
int ... declares the functions produces an integer result – much more later.

```
cout

**cout** .. an *iostream* object responsible for sending output to stdout. On Linux/UNIX systems, every program has 3 I/O channels, or streams, associated with it. If you use pipes and redirection, you're aware of these already:

*stdin* ... standard input. Normally the keyboard.
*stdout* .. standard output. Normally the console.
*stderr* .. standard error. Normally the console.

Example:  $ cat first.C | less

This pipes *stdout* from *cat* into *stdin* of *less*. 
return

The process of running a program can be described as a sequence of *calls*:

- User calls the operating system (OS).
- OS calls the main function.
- Main function calls other functions.
- Functions return results to their callers.
- Main function returns a result to the OS.
- The OS delivers the result to the user.

The return statement is responsible for releasing execution control back to the caller and provides a value.
Exit Status

When a main line exits with a return value, that value becomes the command *exit status* result. If you do shell scripting, you may notice that the status value for success is usually “0”, but you will see some other non-zero number if there was an error of some sort.

```
$ ./hello
Hello World!
$ echo $? 
0
```

0 could mean no error, but technically it answers “was there an error” with “False”. We will see why later.
More on cout

Two more features should be pointed out regarding the cout line in the example:

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

```cpp
<< .. This is called the insertion operator (our first operator of many), and is used to insert data into the cout object for display.

It does exactly what you tell it to do, so you have to explicitly indicate when you want to end one line and start a new one on screen. The "\n" represents the newline character (escaped or meta-character).```
Exercises

Try removing the "\n" and rerun the program to see the difference.

Since multiple insertion operators can be used to build up a line of text, try using the "endl", or end line object, like so:

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

C programmers tend to favor the explicit use of control characters, but `endl` is more portable. Problem is operating systems have different end-of-line conventions:

```
"\n" – Linux/Unix, "\l" – MacOS, "\l\n" – Windows.
```
**What About User Input?**

Here's a jumpstart program that prompts the user for input:

```c++
1. #include <iostream>
2. using namespace std;
3. int main ( void ) {
4.     float gallons;
5.     cout << "How many gallons of water? ";
6.     cin >> gallons;
7.     cout << "That would weight "
8.         << 8.0 * gallons
9.         << " pounds."
10.    return ( 0 );
11. }
```
What's New?

>> ..... the extraction operator takes data from the 
cin object and places it in a variable.
float .. defines (nominally) a 32-bit variable to hold a 
floating point number.
* ..... the multiplication operator.

Note that it also shows the use of multiple insertion 
operators across multiple source lines.

Bonus question: why is the "\n" left off the question?
Summary

Basic source code syntax.
Using the compiler.
Executing results.
Header files.
Statements and blocks.
Rudimentary I/O
Rudimentary Strings
Hint at variables.

Now its just a matter of filling in the details.