CME 211: Lecture 14

Topics:

- Introduction to C++
- Hello world
- Variables
- Strings

C++

- Programming language developed at Bell Labs starting in the late 70s
- B and C were languages also developed at Bell Labs
- `++` is the increment operator, so C++ is an incremental change or improvement to C
- There is also an A+ language but that wasn't developed until the 1980s
- In 1995 Sun Microsystems releases first implementation of Java programming language. The new language was supposed to be similar to C++, but with cleaner, more consistent syntax.
- Microsoft releases implementation of C# (C-sharp) language in 2000 in order to provide another alternative to C++.
- D is a programming language that came out in 2001 and is yet another atempt at "re-engineering" of C++
- C++ is now a standard or specification, and multiple companies or organizations implement the standard.
- Latest version of the standard was ratified in 2014 and is referred to as C++14. Next update is expected to come in 2017. The document defining the standard is 1,366 pages long.
- We will use C++11, which was a substanial update and improvement over C++03. (C++14 was a smaller update, C++17 is a big update – but I don't really know what's in there.)

C versus C++

- C is almost a subset of C++.
- C is a lower level language that has fewer abstractions over the hardware
- Code optimization is more challenging for C++ as it is more complex language than C.
- C is still used for many applications: Linux kernel, CPython interpreter, low power or embedded systems, etc.

Hello world

In this lecture, we are going to start with a C++ source file and modify it to show various things about C++. If you desire to compile and execute subsequent code block on your own, please modify the provided source (or start a new source file).

Let's start with the file: `src/hello.cpp`

```cpp
#include <iostream>

int main()
{
  /* Hello world program (this is a comment)
     this form of comment can span multiple lines
  */

  // this is also a comment, but only goes to the end of the line

  std::cout << "Hello world" << std::endl;

  return 0; // Return value of the function
}
```

- `#include` statments are analogous to Python `import` statements. This is how functionality from other source files and libraries is made available to this program. `<iostream>` has standard C++ input output functionality.
- The `main` function is the entry point into the program. Code from the `main` function body will be executed upon starting the program.
- The `return` keyword returns a value from the function. The type of the return value must match the specified output type of the function (`int` in this case).
- There are two forms of comments in C++
- Text between an `/*` and an `*/` is a comment, this can span multiple lines
- Text after a `//` is also a comment, these comments go to the end of the line

Compilation

- C++ programs have to be compiled
- Compilation is the process of translating the human readable source code into an executable containing the machine instructions that the computer will use while the program is running
- Although C++ source code can be made portable and compiled on multiple machines (Linux, Windows, Mac) the executables are specific to an operating system and underlying processor

Compiling on corn

- We will use the GNU (GNU's Not Unix) compilers that should be available with any Linux distribution

```
$ ls
hello.cpp
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp
$ ls
a.out  hello.cpp
```

Details:

- `g++`: GNU C++ compiler program
- `-std=c++11`: tells the compiler to use the C++11 standard
- `-Wall`, `-Wconversion`, `-Wextra` are flags to enable all warnings
- `hello.cpp` is the name of the C++ source file to compile

Running Hello world

```
$ ls
a.out  hello.cpp
$ ./a.out
Hello world
```

An explanation for the dot slash: http://www.linfo.org/dot_slash.html

Why `a.out`?: https://en.wikipedia.org/wiki/A.out

Naming your executable

Specify the output executable name:

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o hello
$ ls
a.out  hello  hello.cpp
$ ./hello
Hello world
```

Streams

Standard C++ uses "streams" for output. Think of a stream as something you can keep putting information into. In this context, we keep passing strings (and other identifiers) to the output stream, which is then sent to the terminal. There are other forms of streams as well.

```
std::cout << "Hello world" << std::endl;
```

- `cout` is in the `std` namespace and refers to the standard output (stdout) stream
- `endl` is in the `std` namespace and inserts a newline character (`\n`) and flushes the buffer.
- `<<` is the stream insertion operator
- You can also put in a newline yourself, and let the buffer flush automatically as necessary

```
#include <iostream>

int main() {
  std::cout << "Hello world\n";
  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o newline
$ ./newline
Hello world
```

Include (header) files

- When we do `#include` that is somewhat analogous to an import in Python, giving us access to functionality defined in another file
- In C++ the access to even fundamental functionality like outputting to the screen requires specifying the proper include file(s)
- Include files in C++ work a bit differently when it comes to namespaces

- However, namespaces in C++ still generally serve the same purpose as namespaces in Python

Namespaces

- In Python the name of the namespace comes from the file name, and everything in the file is automatically in that one namespace
- A C++ include file might contain functions, classes, etc. that are not in a namespace at all
- An include file could also contain functions, classes, etc. from multiple namespaces
- Namespaces can also span multiple include files, like for the C++ standard library

C++ Standard Library

- The C++ Standard Library is all the built in functionality that is part of the C++ language
- Namespace for this library is `std`
- `iostream` contains `cout` in the `std` namespace
- By default, when using `cout`, we need to specify the namespace and fully qualify the symbol as `std::cout`

Scope resolution operator

- The `::` is called the scope resolution operator
- Used to indicate what namespace something comes from
- If a namespace is required that will typically be listed in the documentation, or by inspecting the include file
- Will talk about namespaces more when we start writing our own include files

Common mistakes

Forgetting to `#include <iostream>`:

```
int main() {
  std::cout << "Hello world" << std::endl;
  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o hello
hello.cpp: In function ‘int main()’:
hello.cpp:11:3: error: ‘cout’ is not a member of ‘std’
   std::cout << "Hello world" << std::endl;
   ^
hello.cpp:11:33: error: ‘endl’ is not a member of ‘std’
   std::cout << "Hello world" << std::endl;
                                ^
```

Common mistakes

Forgetting the `std` namespace:

```
#include <iostream>

int main() {
  cout << "Hello world" << endl;
  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o hello
hello.cpp: In function ‘int main()’:
hello.cpp:11:3: error: ‘cout’ was not declared in this scope
   cout << "Hello world" << endl;
   ^
hello.cpp:11:3: note: suggested alternative:
In file included from hello.cpp:1:0:
/usr/include/c++/4.9.2/iostream:61:18: note:   ‘std::cout’
   extern ostream cout;  /// Linked to standard output
                  ^
hello.cpp:11:28: error: ‘endl’ was not declared in this scope
   cout << "Hello world" << endl;
                           ^
hello.cpp:11:28: note: suggested alternative:
In file included from /usr/include/c++/4.9.2/iostream:39:0,
                 from hello.cpp:1:
/usr/include/c++/4.9.2/ostream:564:5: note:   ‘std::endl’
     endl(basic_ostream<_CharT, _Traits>& __os)
     ^
```

Another namespace option

```
#include <iostream>

// this is not considered good practice
using namespace std;

int main() {
  cout << "Hello world" << endl;
  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o hello
$ ./hello
Hello world
```

Another namespace option

```
#include <iostream>

// good practice, (but not when you write a header file!)
using std::cout;
using std::endl;
```

```cpp
int main() {
  cout << "Hello world" << endl;
  return 0;
}
```

Blocks of code

- Blocks of code, such as the code comprising a function, conditional, loop, etc. are indicated by enclosing them in curly brackets
- There are very few places where whitespace matters to the compiler

```cpp
#include <iostream>
int main(){std::cout<<"Hello world"<<std::endl;return 0;}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o hello
$ ./hello5
Hello world
```

Bracket style

```cpp
#include <iostream>

int main()
{
  /* Hello world program */
  std::cout << "Hello world" << std::endl;
  return 0;
}
```

```cpp
#include <iostream>

int main() {
  /* Hello world program */
  std::cout << "Hello world" << std::endl;
  return 0;
}
```

Return value from `main()`

```cpp
#include <iostream>

int main() {
  /* Hello world program */
  std::cout << "Hello world" << std::endl;
  return 7;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra hello.cpp -o hello
$ ./hello
Hello world
$ echo $?
7
```

```
$ ls
a.out  hello  hello.cpp
$ echo $?
0
```

- Unix standard: programs return `0` under normal conditions and other numbers on error conditions

Variables

File: `src/variables.cpp`

```
#include <iostream>

int main() {
  a = 2;
  b = 3;
  c = a + b;

  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
variables.cpp: In function ‘int main()’:
variables.cpp:4:3: error: ‘a’ was not declared in this scope
   a = 2;
   ^
variables.cpp:5:3: error: ‘b’ was not declared in this scope
   b = 3;
   ^
variables.cpp:6:3: error: ‘c’ was not declared in this scope
   c = a + b;
   ^
```

Declaring variables

```
#include <iostream>

int main() {
  int a;
  int b, c;

  c = a + b;

  std::cout << "c = " << c << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 variables.cpp -o variables
$ ./variables
c = 32767
```

Compiler warnings

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
variables.cpp: In function ‘int main()’:
variables.cpp:9:12: warning: ‘a’ is used uninitialized in this function [-Wuninitialized]
   c = a + b;
           ^
variables.cpp:9:12: warning: ‘b’ is used uninitialized in this function [-Wuninitialized]
```

- Enable them, read them, and fix them
- We will not have any sympathy if you have bugs that would have been caught by enabling warnings
- You will lose points if compilation of your program generates warnings

Initializing variables

```
#include <iostream>

int main() {
  int a = 2;
  int b;

  b = 3;
  int c = a + b;

  std::cout << "c = " << c << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
$ ./variables
c = 5
```

Type errors

```
#include <iostream>

int main() {
  int a;
  a = "hello";

  std::cout << "a = " << a << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
variables.cpp: In function ‘int main()’:
variables.cpp:10:5: error: invalid conversion from ‘const char*’ to ‘int’ [-fpermissive]
   a = "hello";
     ^
```

Mixed number types

```
#include <iostream>

int main() {
  int a, b;

  a = 2.7;
  b = 3;
  int c = a + b;

  std::cout << "c = " << c << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
variables.cpp: In function ‘int main()’:
variables.cpp:6:5: warning: conversion to ‘int’ alters ‘double’ constant value [-Wfloat-conversion]
   a = 2.7;
     ^
```

Mixed number types

```
#include <iostream>

int main() {
  int a, b;

  a = (int)2.7; // int(2.7) would also work

  b = 3;
  int c = a + b;

  std::cout << "c = " << c << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
$ ./variables
c = 5
```

Double precision floating point

```
#include <iostream>

int main() {
  int a;
  double b = 3.14;

  a = 2;
  double c = a*b;
```

```
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
$ ./variables6
c = 6.28
```

Rounding

```
#include <iostream>
#include <cmath>

int main() {
  int a;
  double c = 2.7;

  a = (int)round(c);
  // note that the round() function
  // is not in the std namespace

  std::cout << "a = " << a << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra variables.cpp -o variables
$ ./variables
a = 3
```

Key data types

- C++ has all of the data types that we talked about when we looked at computer representation of data in conjunction with NumPy
- Various signed and unsigned integers
- Floating point (float, double, long double)
- A Boolean data type
- A string object

Boolean

```
#include <iostream>

int main() {
  bool equal;

  equal = 2 == 3;
  std::cout << equal << std::endl;
```

```
  equal = true; // All lowercase

  std::cout << equal << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra boolean.cpp -o boolean
$ ./boolean
0
1
```

Strings

- There are two options for strings in C++
- An array of characters (same as C)
- A string object
- The latter is much safer and easier to use

String example

```
#include <iostream>
#include <string>

int main() {
  std::string hello = "Hello world";

  std::cout << hello << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra string.cpp -o string
$ ./string
Hello world
```

String concatenation

```
#include <iostream>
#include <string>

int main() {
  std::string hello = "Hello";
  std::string world = " world";

  std::string helloworld = hello + world;

  std::cout << helloworld << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra string.cpp -o string
$ ./string
Hello world
```

String finding

```
#include <iostream>
#include <string>

int main() {
  std::string hello = "Hello";
  std::string lo = "lo";

  std::cout << hello.find(lo) << std::endl;

  return 0;
}
```

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra string.cpp -o string
$ ./string
3
```

Recommended reading

- **C++ Primer, Fifth Edition** by Lippman et al.
- Available on Safari ProQuest: http://proquest.safaribooksonline.com/book/programming/cplusplus/9780133053043
- Chapter 1: Getting Started, Sections 1.1 - 1.3
- Chapter 2: Variables and Basic Types, Sections 2.1 - 2.2
- Chapter 3: Strings, Vectors, and Arrays: Sections 3.1 - 3.2

Resources

- Online C++ compiler for small tests: http://coliru.stacked-crooked.com/
- http://www.cppreference.com
- http://www.cplusplus.com
- http://www.linfo.org/index.html