MTH 4300, Lecture 1

Introduction + What's Different About C++?; Hello World; Declaring Variables, `cout`, `cin`

E Fink

January 29, 2025

1. Introduction + What's Different About C++?

What will be studying in this class?

- Object-oriented programming
- Implementation of basic data structures
- Problem solving using recursion
- The C++ language

Introduction + What's Different About C++?

Why C++? Why not more Python?

- It's good to be versed in more than one language.
- Naive C++ code is *far* faster than equivalent naive Python code. (See **L1x1_billion.cpp** and also **python_billion.py**.)
- Because of the previous bullet, C++ is extensively used in, e.g., computational finance and games, where performance is critical.
- C++ is also *statically typed*, which means that the compiler which builds your program does an extensive search for type errors before your code starts running. This makes finding errors early on far easier. (See **python_sneaky_error.py** for an example of nasty behavior that would never happen in C++.)

Introduction + What's Different About C++?

You'll surely notice that C++ is far clunkier than Python. C++ was originally designed in the early 80's. It was designed to be backwards compatible with the C language, which was designed in the early 70's. There were some design choices at those points that did not have the benefit of the explosion of software engineers that were to come. The aforementioned static typing also adds to that clunkiness, although this is probably a benefit on balance.

What's Different about C++?

We've mentioned the *compiler* in passing. Prior to running C++ code, you feed it to a program called a compiler, which translates it to an executable file. (Actually, there are several programs working in sequence, in addition to the compiler.) The compiler performs optimizations to produce machine code instructions that are efficient as possible; this is one of the primary sources of the speed of a well-written C++ program.

A side benefit to this is that once you have created a C++ executable for a particular operating system, you can pass that executable to any other computer running that operating system, and it will run. The other computer doesn't have to have a working C++ compiler.

2. Hello World!

Hopefully you've already opened up **L1x2_hello.cpp**

Every C++ program contains a function named `main()`. This is the "entry point" to the program. When you run a program, you are really calling its `main()` function, which of course could call upon other functions. The body of this function, and every function in C++, is enclosed in curly braces.

The two statements are the instructions executed when `main()` is called. Note that each statement should end with a semicolon.

The line that starts with `std::cout` is responsible for printing. The function `returns` the value 0 to the operating system, which commonly signifies successful completion of the program.

And // means that the rest of the line is a comment.

Hello World!

Now, compile! Follow the instructions on the provided sheet:

- Look at the top for the “Terminal” menu. Select “Open a New Terminal.” At the bottom of your screen, a prompt should appear.
- Click next to the prompt, and then type

  ```
  g++ L1x2_hello.cpp -o hello
  ```

  or if you are using a Mac, try

  ```
  clang++ -std=c++17 L1x2_hello.cpp -o hello
  ```

- Finally, run your program, by typing `./hello` on the next line of your command prompt.

3. Declaring Variables, `cout`, `cin`

In C++, we introduce new variables by *declaring* them. A variable declaration starts with the name of a data type, and is followed by the names of one or more variables, separated by commas; usually, each variable is also assigned a value using the assignment operator = (like in Python).

L1x3_variables.cpp

In this example, we introduce three data types: `int` representing integers, `double` representing non-integer real numbers (what we call `floats` in Python), and `std::string` representing strings. We also introduced four variables, `x`, `y`, `z` and `message`.

To use the `std::string` data type, we #`include <string>`.

Declaring Variables, `cout`, `cin`

`std::cout` Syntax:

`std::cout` << *fill1* << *fill2* << *fill3* << . . . ;

Each *fill* can be (for starters) a single *literal*, a single *expression*, or `std::endl` (for newlines).

`std::cout` is technically a strange variable, called the *character output stream*. You add characters to this stream using the *stream insertion operator*, which is <<. Generally, when a `cout` statement is finished evaluating (or when a `std::endl` is reached), everything in the stream gets *flushed* out onto the standard output – which causes the characters to appear in the console box on your screen!

Declaring Variables, cout, cin

For input, there is also the std::cin variable, which is quite different from Python. **L1x4_cin.cpp**

Syntax:

std::cin >> *var*;

where *var* MUST be a variable (which could be of any number of types).

When a line like this is encountered, the program will pause and wait for the user to type in some input and press Enter. Then, if the user happens to enter a value that is compatible with the type of *var* – for instance, if *var* is a double, and the user enters 123.45 – that value will be assigned to *var*. (No string-to-float or string-to-int conversions needed!)

std::cin is also technically a variable, the *character input stream*, and the operator >> is called the *stream extraction operator*. More about cin later.

Declaring Variables, `cout`, `cin`

L1x5_average.cpp

Create a program which does the following:

- Declares `int` variables `x` and `y`.
- Asks the user to enter values for these variables, and write code which accepts those values.
- Prints out `The average is:`, with the properly-calculated average of `x` and `y` on the next line, using a SINGLE `cout` statement.