

W3101.002 Intro to C++

Joshua Reich

`reich@cs.columbia.edu`

Department of Computer Science
Columbia University

Session 1



Outline

Administrative Details

- General Information

- Approach & Topics

- Expectations, Evaluation, and Policies

Using CUNIX

- What Is It?

- What Do I Do With It?

Commandline Software Development

- Unix & Emacs

- Understanding The Software Development Process

Jumping Into C++

- First Program

- More Programs



Class Information

Your primary source for information should always be the class website:

`www.cs.columbia.edu/~reich/cs3101`

However what is written on the course website may be superseded by

- ▶ Class announcements
- ▶ Email sent out to the course mailing list
(make sure to check your Columbia email!)

Prerequisites

Fluency in at least one programming language.

If this is not the case for you, speak with me after class.

What You Will Learn

Students should leave this class with knowledge of fundamental C++ programming constructs and the tools necessary to quickly master more advanced techniques. Moreover, students will be expected to demonstrate ability to program effectively in C++, producing code that:

1. compiles (on CUNIX!)
2. produces correct output
3. utilizes proper C++ constructs
4. is coherent and clearly commented.

What You Won't Learn

This class is an *introductory* class and it's also only *10 hours* of course time.

Consequently the scope of what we will be able to cover will be limited, and the pace will appropriate to an accelerated but still introductory exposure to C++.

Some things we are extremely unlikely to learn include:

- ▶ The C programming language
- ▶ More intricate details of Object Oriented Programming
- ▶ Machine specific coding
- ▶ GUI programming
- ▶ Large-scale coding techniques and tools

Approach

We won't focus on details of C++ grammar and rarely used keywords (although we will try to cover much of this in passing). Instead we will focus on *doing things* with C++.

Course Goals:

- ▶ give you a broad understanding of what C++ is
- ▶ what it is capable of
- ▶ how to figure out where you need to look when you don't know how to do something

Basic Topics Covered

- ▶ Console I/O with stream classes
- ▶ Basic string handling
- ▶ Loop and flow-control statements
- ▶ Arrays
- ▶ Using functions and methods
- ▶ Using Standard Template Library (STL) containers (vectors, linked lists, and maps)
- ▶ Iterators
- ▶ Sorting and generic functions
- ▶ Basic class design
- ▶ Pointers and arrays
- ▶ File I/O
- ▶ Memory-management techniques



Advanced Topics

- ▶ Adding stream support to custom classes
- ▶ Conversion operators
- ▶ Operator overloading
- ▶ Friend functions
- ▶ Inheritance
- ▶ Polymorphism and virtual functions
- ▶ Handle idioms for classes, including reference counting

Expectations

You are responsible for:

- ▶ Assigned readings
- ▶ Material covered in lectures
- ▶ Homework
- ▶ Final

Method Of Evaluation

Performance will be evaluated on the basis of the following criterion:

1. Homework (2/3^{rds})
2. Class participation (instructor discretion)
3. Final Examination (1/3rd)



A Note About Cheating

Don't.

Common sense should inform you as to whether or not something is cheating, if you are still in doubt please refer to the SEAS academic conduct policy (linked from course website).

Of course you are welcome to simply ask me as to whether something is acceptable or not.



Grading Policy

There will be 5 assignments. Each will be assigned at the end of class and due at the beginning of the following class (CVN students have until the following morning). Homework 5 will be shorter and due the Friday before the final so as to give you some time to study.



Lateness

- ▶ It is in your best interest to finish homework on time.
- ▶ However, given that you will inevitably have conflicts, each student has 5 free late days.
- ▶ Any additional lateness past this will be penalized by 20% penalty on that assignment per day late.
- ▶ Note: late days are *discrete* (i.e., whether your homework comes in 1 hour or 23 hours after the deadline, you've still used 1 day).



Open Door Policy

My main interest is seeing you succeed in this class. I know what it's like to be a student and am not out to make your lives unnecessarily difficult. Therefore, if you have a problem, need an extension due to extenuating circumstances, want to discuss the your performance, or are just feeling down - please come speak with me.

I also value your feedback and suggestions - I want to make this a good course for you. Therefore feel free to come speak with me, or send me an email, even if its critical/negative.

Course Page & Courseworks

- ▶ tour of coursepage
- ▶ spotlight on syllabus / homework instructions
- ▶ brief look at courseworks

What Is A CUNIX Account?

A CUNIX account is an account you have on Columbia's servers. You connect to it and do pretty much anything one does on a computer, well at least a UNIX computer where you can't really change any of the settings and don't have a GUI interface...

While you may develop your software assignments on any system and compiler, *it must successfully compile on CUNIX*. If your software fails to compile on CUNIX you will receive *no credit*.

For more info on setting up a CUNIX account:

<http://www.columbia.edu/acis/publications/handouts.html>

<http://www.columbia.edu/acis/tools/index.html>



But What If I Don't Know How To Use Unix

You learn.

Don't worry, its easier than it looks and we will go through this step by step.



Connecting To CUNIX

First you need to connect to your account.

If you already have an secure shell (ssh) client installed you are golden.

If not go to <http://www.columbia.edu/acis/software/terminal.html> and download an ssh client. For good measure download a secure file transfer client while you are at it:

<http://www.columbia.edu/acis/software/filetransfer.html>



Starting Up Your Connection

This works differently on different systems, but three things will always be invariant.

1. the address you are connecting to -
`cunix.cc.columbia.edu`
2. your user name - in my case `jr535`
3. your secure password

Once You've Logged In

You will see a couple of messages and a command prompt - might look like a dollar sign, mine is a smiley face. The main thing is you can type things at it.

We will spend most of our time doing things like making and moving around in directories, writing, compiling, and testing code. But there are lots of other things one can do (web-browsing, reading email, playing chess, etc.)

Quick Intro To UNIX

CUNIX is a UNIX system. Therefore you need to know how to use UNIX.

While Graphic User Interfaces (GUIs) have been built for some variants of UNIX, the command-line remains the most powerful and foolproof method of getting UNIX to do what you need.

How does it work? Simple, you type the command you want to execute after the command prompt and then press the *return* key on your keyboard. Then (hopefully) the operation you requested is executed the way you had in mind. Of course you need to be careful about what you write as the computer will only do what you tell it, not what you wanted it to do.



A Cautionary Example

A cautionary example:

`rm mary*` will remove all files beginning with your evil ex-girlfriend Mary's name, while accidentally inserting a space and typing `rm mary *` will remove a file named `mary` and everything else in your current directory!

UNIX Tutorial

- ▶ directory structure
- ▶ man
- ▶ ls -l
- ▶ more, less
- ▶ pwd
- ▶ cd
- ▶ mkdir, rmdir
- ▶ cp, mv, rm
- ▶ exit
- ▶ fancy stuff (prompts, colors, .profile, etc.)



Learning More About UNIX

ACIS has a helpful web-page you may wish to examine:
<http://www.columbia.edu/acis/webdev/unix/index.html>

One of the best books I've read on learning new command-line tools (and using them) is:

Unix Power Tools, Third Edition (Paperback)

by Shelley Powers, Jerry Peek (Editor), Tim O'Reilly, Mike Loukides

O'Reilly Media, Inc.; 3 edition (October 1, 2002) ISBN:
0596003307

and of course web-tutorials are always good:
<http://www.ee.surrey.ac.uk/Teaching/Unix/>

Emacs Is The Best

I use `emacs` for pretty much all of my own software development. Its flexible, powerful, and runs on pretty much any operating system. Some like to use other editors like `vi`, `pico`, `nano`, or `ed` (if you are really crazy). However, `emacs` is the best.

Getting Started With Emacs

To start emacs for the command-line simply type `emacs`.

To save your file hold down the `control` key and `x` release them and then hold down `control` and `s` (`C-x C-s`).

`C-x C-c` will exit and return you to the command-line.

For the Emacs tutorial: `M-x` (depending on your system and keyboard pressing on the `escape` key and then afterward pressing `x` and then `enter`, or holding down `option` and `x`) then typing `help-with-tutorial` (`M-x help-with-tutorial`).

Another good resource is:

<http://www.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html>

Emacs Tutorial

- ▶ C-x C-s Save
- ▶ C-x C-x Quit
- ▶ C-s Search
- ▶ C-g Stop Command
- ▶ M-x goto-line Jump to Line
- ▶ M-x replace-string Replace one string with another
- ▶ C-x C-f Open new file
- ▶ C-x 4 C-f Open file in bottom
- ▶ C-x o Jump to other window
- ▶ C-x 1 Expand current window
- ▶ C-x b Switch buffer
- ▶ fancy stuff (colors, modes, .emacs)

Why Is This Important?

- ▶ as programmers you need to make the computer do things
- ▶ but if you want to do it right, you need to understand what is really going on
- ▶ otherwise, when things go wrong you will be lost
- ▶ consequently, we briefly discuss what is going on under the hood
- ▶ AND we get you close to the machine by using commandline and emacs

Once you've gotten a strong grasp of what is going on, GUI tools can be very helpful.

One of the best is Eclipse - <http://www.eclipse.org/>



How A Computer Works: The Quick Version

- ▶ computers have *memory*, *I/O*, and *processor(s)*
- ▶ *memory* stores information - usually in binary representation
- ▶ *processors* move information around and manipulate it
- ▶ *I/O* allows the computer to interact with the environment (i.e., display output and accept input)



Machine Language & Assembly

Let's say I want to tell a computer to place the value 97 in its a1 register.

- ▶ Computers understand 0's and 1's
A machine readable instruction for an x86 processor:

```
10110000 01100001
```

- ▶ People understand words, etc.
The same instruction in human readable notation:

```
mov  a1, 0x61
```

http://en.wikipedia.org/wiki/Assembly_Language

Assembly Is A Pain

- ▶ hard to read
- ▶ have to write *everything* out step by step
- ▶ need to learn a new language for each machine
- ▶ really easy to make mistakes



Hence Higher Level Programming Languages

Instead, take a bunch of more complicated constructs and make names for them, e.g.:

- ▶ print (whatever message is in here)
- ▶ do whatever is in these brackets a bunch of times times
- ▶ calculate the square root of (whatever is in here)
- ▶ etc.

But How Does The Computer Understand What We Want?

Turn our high-level instructions into assembly instructions using a special program called a *compiler*. Then turn the assembly into machine code using an *assembler*. Sometimes we throw both these tasks into one program.

Compiling & Disassembling

```
:-) ls
HelloWorld.cc
:-) g++ HelloWorld.cc -o HelloWorld.out
:-) ls
HelloWorld.cc HelloWorld.out*
:-) ./HelloWorld.out
Hello, World!
:-) objdump -d HelloWorld.out > HelloWorld.assembly
:-) ls
HelloWorld.cc HelloWorld.assembly HelloWorld.out*
```

Hello World!

```
#include <iostream>

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

CODE: HelloWorld.cc

Features Of This Program

Consider the analogy between a business letter and a computer program.

- ▶ `#include` statements / directives
 - ▶ core language vs. standard library
 - ▶ standard header
- ▶ `main` method
- ▶ return statements
- ▶ block statements { }
- ▶ basic output using `iostream` class
- ▶ namespace conventions

Strings, Simple I/O, And Basic Commenting

- ▶ `string` - don't forget the `include` statement!
- ▶ the standard input stream, `std::cin`
- ▶ commenting
 - ▶ syntax
 - ▶ style
 - ▶ commenting with clearly written code

CODE: `input.cc`, `commenting.cc`

Learning How To Do Arithmetic

How to add $2 + 2$

- ▶ basic expressions
 - ▶ result
 - ▶ side-effects
 - ▶ operators
 - ▶ associativity
 - ▶ add a ; and its a statement!

CODE: `2plus2.cc`



Pitfalls

- ▶ pitfall: leaving out `include` directives
- ▶ pitfall: forgetting to recompile
- ▶ pitfall: forgetting `-o`
- ▶ pitfall: C++ doesn't always warn you about misuse of types