

# W3101.002 Intro to C++

Joshua Reich

`reich@cs.columbia.edu`

Department of Computer Science  
Columbia University

Session 5



# Review

- ▶ Template functions and classes
- ▶ Separate Compilation & Make
- ▶ Debugging, Preprocessor, & GDB

# Deriving New Classes

*Inheritance* allows us to make new types of objects by adding to objects we've already made!

- ▶ how to declare a derived class
- ▶ specializing our derived class
- ▶ constructors
- ▶ using our new class
- ▶ overriding methods
- ▶ polymorphism and containers

CODE: `inheritance_example/`

# Pointers

A *pointer* is a low-level analogue of an *iterator*. A *pointer* points to some object, particularly it points to that object's memory address.

- ▶ declaration `int *ip`
- ▶ address operator `&`
- ▶ using pointers to change values
- ▶ pointers can point to functions
- ▶ you can do arithmetic with pointers - but usually its dangerous!!!

CODE: `pointers.cc`, `references.cc`

# Arrays

You can think of an *array* as a crappy `vector` you need to specify exactly how big it is beforehand and you can't query with `size()` etc.

- ▶ arrays are blocks contiguous memory
- ▶ declaration
- ▶ indexing
- ▶ use pointers like iterators!
- ▶ be careful not to overrun the array!

CODE: `arrays.cc`

WEB: <http://www.cplusplus.com/doc/tutorial/pointers.html>



## cstrings

C is not an Object Oriented language, thus it didn't have `string` so how were strings represented beforehand? Using *cstrings* which are simply an array of characters, with a special “stop-here” symbol `\0`.

- ▶ creating a `cstring`
- ▶ examining contents of a `cstring`
- ▶ quick initialization of arrays
- ▶ copying `cstrings`
- ▶ converting between `cstrings` and `strings`
- ▶ converting between `cstrings` and other types
- ▶ useful libraries: `<cstring>`, `<cctype>`, `<cstdlib>`

CODE: `cstrings.cc`, `strings.cc`, `atoi.cc`

TEXT: *Accelerated C++* Section 10.2



## Arguments to main

As you've seen many programs take arguments for the command-line (e.g. `ls -l`, `gdb a.out`, `emacs main.cc`, etc.).

C++ programs are no exception.

CODE: `arguments_to_main.cc`

TEXT: *Accelerated C++* Section 10.4

# File IO

Input and output from files allow our programs to store information in between runs! No more having to re-enter input every time.

- ▶ `<fstream>`
- ▶ `ifstream instream("filename");`
- ▶ `ofstream ostream("filename");`
- ▶ use just like `cin` and `cout`
- ▶ don't forget that filenames are *cstrings* not `string`

CODE: `file_io.cc`

CODE: `file_io2.cc`

TEXT: *Accelerated C++* Section 10.5



# Memory Management

*Memory Management* lets you allocate and destroy memory store directly.

Why - lets say you want to build your own data-structures - you don't like the STL ones. Or perhaps your program creates lots of data that you don't need and you want to make sure it is deleted so you don't run out of memory.

Memory management is dangerous and confusing, but you will see code using it, and may need to yourself as its the only way to get certain things done.

▶ `new, delete`

CODE: `memory.cc`

TEXT: *Accelerated C++* Section 10.6



# Lists

A `list` can't do everything a `vector` can but it can do some things a heck of a lot faster! The difference is `vector` provides *random access* (i.e. it uses indexing) whereas `list` only allows *sequential access* (i.e. through iterators). Remember there are lots of generalized algorithms in the STL and they work on practically all containers!

CODE: `list.cc`

# Maps

A `map` has a different purpose than either `list` or `vector`. It allows for the storage and manipulation of paired data. Let's say you want to keep track of the number of times each word appears in a sentence...

CODE: `map.cc`

## Major Topics Overview

- ▶ Variables
- ▶ Datatypes
- ▶ Operators
- ▶ I/O
- ▶ Flow Control
- ▶ Containers & Generic Algorithms
- ▶ Functions
- ▶ Error Handling
- ▶ Object Oriented Programming & Inheritance
- ▶ Templates and Generic Programming
- ▶ Separate Compilation and makefiles
- ▶ Preprocessor Facilities
- ▶ Debugging w/ GDB
- ▶ C-level facilities & File I/O



# Final

- ▶ open books, notes, materials
- ▶ no communications equipment allowed (all cell phones, pagers, etc. must be off)
- ▶ You will be graded on how well you demonstrate to me your ability to code in C++
- ▶ study tips
  - ▶ familiarize yourself with code samples from class
  - ▶ review homeworks and comments
  - ▶ read over lecture notes and assigned readings for text

