LECTURE 29: C++

Today

- Reading
  - (Weiss Chapter 0 – interesting history)
  - Weiss Chapter 1, 2
- Objectives
  - History of C, C++, Java
  - Similarities/Differences
  - First C++ program

Extremely simplified history!

- C was developed in early 70's
  - Designed for systems programming
  - Provides low-level access to memory
  - Extremely popular still today
  - Fast
- C++ developed in late 70's by Bjarne Stroustrup
  - C with object oriented support
  - Backwards compatible with C
  - Still fast and still widely used today
- Java developed by Sun Microsystems in 90s
  - Uses C/C++ syntax
  - Explicitly disallows "bad programming"
  - Java bytecode runs on virtual machine

Different Goals

- C++
  - Correct programs run as fast as possible
- Java
  - Incorrect programs not allowed to run

Which language is better depends on the application!

Similarities between C++ and Java

- Primitive types: int, float, double, bool, char, void
  - C++ can also be signed (possibly negative) or unsigned (always positive)
  - C++ can also be long (more bytes) or short (fewer bytes)
- Syntax
  - Curly brackets
  - Function syntax: return type, name, input parameters
  - for loops, while loops, if statements, if-else, switch statements
  - The "." operator to call a function on an object
- C++ has a standard template library with many of same data structures available as Java

High-Level Differences

Java	C++
• Compiles to bytecode which is interpreted by JVM	• Compiles to native code specific to the architecture of the machine
• Enforces safety	• Safety left to programmer
• Garbage collector	• Possible to have a pointer to an object already returned to the system!
= Portable but slower	= Not portable but faster

Other differences

- C++ doesn't require classes
  - Can be used as a procedural language
  - All execution begins with `main` method

In Java, can get same behavior with `static` keyword

- The preprocessor: #include statements
  - Equivalent to copying and pasting file
  - The # symbol is a preprocessor directive, i.e. resolved before compile time
  - `#include <file>` for built-in system files
  - `#include "file.h"` for user defined files

Other differences

- Must declare all variables and functions before you use them.
  - Historically C++ compiler process source code from top to bottom
  - When function is called, compiler looks for functions it's already seen
- Possible options
  - Define all functions before you invoke them
  - Place function prototype at top of file
  - Create a .h (header) file to contain function prototypes and use `#include` to include header file

Other differences

- Namespaces are a generalization of packages
  - Named region of code contained in curly brackets
  - Helps disambiguate between variables and functions with same name
- The `std` namespace
  - Always have to specify
  - In C++, the `vector` type is in the `std` namespace
- To use `std`
  - Write `using std namespace;` at top of file
  - Use `::` operator, e.g. `std::vector` or `std::cout`

the "using" keyword is similar to import statement in Java

Operator Overloading

- Define a meaning for existing operations (e.g. + or []) for new class types

```
nums[i] // nums is a vector (i.e. an ArrayList in Java)
```

- Overloaded the [] operator so it acts like the "at" method
  - The "at" method is bounds checked and throws an exception
  - The operator[] method is not bounds checked
- See C++ documentation for vector
- Makes classes look like primitives

The biggest difference: memory management

- In Java, most types are objects
  - Except for local primitives such as int, double, boolean, etc
- In C++, everything is a primitive
  - Allocated on the stack not the heap
  - To allocate from heap, explicitly use `new` keyword
- *Changes the semantics of assignment*
  - Assignment now means copying!
  - Assignments happen all the time (more than you're aware of)