CSE 303:
Concepts and Tools for Software Development

Dan Grossman

Spring 2005

Lecture 28— A Taste of C++, Wrap-Up

About the Final

- Next Tuesday, this room, 2:30–4:20.
- I think it's hard because it more explanation (less fill-in-the-blank) than the midterm, and a bit long.
- Intended to test second-half of the course, but that includes plenty of C issues.
- Does not cover C++ or html/cgi.
- Does cover debuggers, profilers, linking issues, testing and assertions, make, version-control, manual memory-management, and concurrency.
- Warning: Though the homework has not used Java, class has.

C++

An enormous language. Used in some department classes, many companies, ...

Hopefully knowing C and Java makes C++ easier to pick up.

- All of C (almost backwards-compatible)
  - Some new keywords
  - Compiled/linked differently
- Like Java: classes, inheritance, methods, fields, public vs. private...
  - Still distinguishes objects and pointers-to-objects
- More OO features/control
  - multiple inheritance, non-virtual functions, ...
- Other: templates, operator overloading, reference parameters, namespaces, exceptions, ...

The C part

- `struct` (and `class`) names can be used as type names.
- Functions taking different arguments can have the same name.
  - Compiler name-mangles so that linking still works
  - So you cannot just link C and C++ files together.
    - Either recompile the C as C++, or
    - Call from C++ to C using `extern "C"` declaration.
- Lots more libraries for convenient things like output and input (really cleverly using C++ features).
- namespaces: either add `using namespace std;` near the top of your file, or *prefix* things, e.g., `std::cout`.
  - That way you can avoid name conflicts so long as namespaces do not reuse names; very useful for large programs.
- All the same ways to crash; casts still not checked.

The "Java" part

Not everything is in a class, but you can define classes, with fields, methods, constructors, etc.

And you can call `new` to get a pointer to a heap-allocated object initialized by calling the appropriate constructor.

To get Java-style overriding, you must declare the method `virtual` in the superclass. (Java has no notion of non-virtual.)

Other minor notes:

- Can declare methods in the class definition (e.g., in a header file) and define them elsewhere (e.g., in a `.cc` file).
- Actually structs and classes are the same thing, except different defaults about public vs. private.
- No real need for static fields/members because not everything needs to be in a class.

More "OO"

- No interfaces. Instead the more-powerful multiple-inheritance.
  - A superclass with all "abstract" (pure virtual) methods is like an interface.
  - Multiple-inheritance has thorny issues in definition and implementation; see CSE341.
- There are different notions of casts
  - C has "numeric conversions" and "reinterpret the bits"
  - Java has "numeric conversions" compile-time "upcasts" and run-time checked "downcasts"
  - C++ essentially has all of the above.
- Destructors and `delete` (opposite of constructors — called right before the actually "free" of memory)
  - Java has finalizers; much less control over when they are called

Operator overloading

In C and Java, the language sets what types built-in operators require:

- `+` takes ints or floats (or in Java strings)
- `(...)` takes a function
- `*` takes a pointer

In C++, you can make any operator work for any type(s) by defining a "special" function that gets called when the operator is invoked.

Possible uses (in rough decreasing order of "accepted style"):

- Make arithmetic ops work on a new thing (e.g., bignums).
- Use function-call syntax for objects with one method.
- Automatic reference-counting
- Things like `lst++` for iteration.

Related: *copy constructors* let you change the behavior of assignment.

Templates

Classes, functions, etc. can be parameterized by types (and actually other things too).

- More convenient than `void*` and casts.
- On the surface like Java 1.5 generics and ML polymorphism.
- But fundamentally *just automatic code duplication* (more name-mangling):
  - Necessary because different *instantiations* of `T` may have different sizes (see CSE401).
  - Requires *leaking implementations* (must put templatized method and function bodies in header files!)
- Instantiation can be any type (not just class types).
- The Standard Template Library provides lots of data structures, but can be hard to use correctly and efficiently.

C++ "Summary"

- All the danger of C, but convenience and flexibility of OO programming.
- "Virtual methods" are what you are used to in Java.
- Operator overloading can let almost any syntax do almost anything.
- Templates allow automatic code duplication.

Putting it together:

- `std::cout` is just a global variable of type `ostream`.
- Operator overloading defines how << works for an `ostream` and a `char*`, an `ostream` and an `int`, etc.
- The operators do the printing and return the same `ostream`; implicit parentheses takes care of the rest.

Course Summary

A lot of comfort and de-magicalizing

- 90% of you thought 90% of your classmates knew more of "that practical stuff"

A lot of pragmatic application of computer science

1. There are concepts behind `gdb`, but there's also value in knowing a couple handfuls of commands
2. The best computer scientists are not the folks who have the largest number of utilities, options, and features memorized
   - But the best computer scientists also don't know the least
   - And they're definitely not afraid to learn new ones.

Some concepts too

We did *not* just pick up one strange set of rules after another!

- Shell-scripting is about automating program-invocation.
- C programming is about a lower level of computing where:
  - all data is just bits and pointers are explicit
  - memory is managed manually
- Debuggers and profilers are about how to characterize and measure running programs.
- Linking, make, etc. are about how code is created and combined to make running programs.
- Assertions and specifications are about assumptions and the robustness of larger programs.
- HTML and CGI is about making programs public.

Societal implications

And we talked about ethics too.

- We talked about it like computer scientists
- (Overly) logical, rational, understanding limitations of technology and humans
- "We" should not be making all the decisions ourselves, but we should be actively engaged in the process.

The Ultimate Goal

5 years from now, look at the course webpage and think this whole course was a waste of time.

After all, you won't remember not knowing this stuff.

And you won't understand why you need a teacher to help you pick up a new tool or automate a repetitive task.

From lecture 1: *There is an amorphous set of things computer scientists know about and novice programmers don't. Knowing them empowers you in computing, lessens the "friction" of learning in other classes, and makes you a mature programmer.*
"There is more to programming than Java methods"
"There is more to software development than programming"
"There is more to computer science than software development"
"There is more to computing's effects than computer science"