Reading:
- Mitchell, Ch. 12

Objectives
1. be able to recognize and explain the difference between C++ objects created automatically on the stack and those created explicitly (only heap)
2. be able to describe the difference between virtual and non-virtual functions in C++
3. be able to explain how C++ object layouts and vtables work
4. be able to explain why C++'s virtual function dispatch is more efficient than SmallTalk's message dispatch
5. be able to diagram the memory layout associated with C++ classes and objects (single inheritance case).
6. be able to describe issues with multiple inheritance
7. be able to describe the difference between virtual and non-virtual base classes

Outline
1. 1980's “C with classes”
2. goals
   (a) data abstraction and object-oriented features
   (b) better static type checking
   (c) backwards compatibility with C (i.e., most C code should compile in C++ and continue to work)
   (d) efficiency: “If you do not use a feature you should not pay for it”
3. non-OO enhancements
   (a) bool type
   (b) reference types and pass-by-reference
   (c) user-defined overloading (including operators; too much?)
   (d) templates (generics for functions and classes)
   (e) exceptions
   (f) namespaces (at least since 1998 standard)
4. OO concepts
   (a) classes (enhancement to C structures, including methods)
   (b) objects
(c) dynamic lookup (only for virtual member functions)
(d) encapsulation
   i. public, private, protected, friend
(e) inheritance (without subtyping through private inheritance)
(f) subtype (through public inheritance)
(g) abstract class

5. differences from other OO languages

(a) dynamic lookup is not default
(b) multiple ways to allocate objects
   i. local object variables (Point p1(2,3))
   ii. reference to local object variable (Point& p2 = p1;)
   iii. pointer to local object variable (Point* p3 = &p1;)
   iv. reference to dynamically allocated object (Point& p5 = *(new Point(2,3);)
   v. pointer to dynamically allocated object (Point* p4 = new Point(2,3);)
(c) no standard garbage collection
   i. local object variables have function-local scope and are allocated in that function’s activation record
   ii. explicit deallocation of dynamically allocated objects
   iii. correct deallocation is not enforced (memory leaks, dangling pointers)
   iv. But check out the Boehm garbage collector for C/C++: 

6. vtable and dynamic lookup for virtual functions

(a) constant time lookup as offset in vtable (virtual table)
(b) single inheritance: base class prefixes derived classes
(c) multiple inheritance: pointers to different offsets within the vtable

7. language issues with multiple inheritance

(a) :: for scope resolution
(b) diamond inheritance and virtual base class

Vocabulary

member function, data member, virtual function (function with dynamic lookup), base class (superclass), derived class (subclass), abstract function, abstract class, multiple inheritance, diamond inheritance virtual base class virtual base class