Chapter 10 -- Object orientation

- Languages like "standard" Pascal and C -- had no classes
  - how to implement an AVL tree (or other data structure)?
    - record and procedures/functions to manipulate it
    - procedure/functions -- required tree as a parameter
  - Issue -- any code could manipulate the AVL nodes ...
- Someone asked: Why can't we declare code associated with the AVL node, restrict other code from messing with nodes
- Later, can't we make it more generic?
- Object idea
  - Declare data types and operations on those types
    - scoping, other features to control visibility of names
    - (book) Module types -> multiple instances of an abstraction, Classes -> family of abstractions
      - Modules typically don't have inheritance, Classes typically do have inheritance
  - encapsulation, inheritance, dynamic method binding started in Simula (from Norway)
  - Other early languages: Clu, Modula, Euclid
  - Smalltalk was first totally object oriented language --- but used "messages"
    - dynamic typing -> slow (on slow computers), needed its own graphical IDE ...
    - not widely adopted
  - GUI programming popularized the idea of a generic "widget" implemented by objects
  - Idea of a "basic functionality" and then a "specialized functionality"
    - Dynamic method binding allowed custom behavior when needed

OO Basics

- Classes -- currently the core of object orientation
  - Data fields
  - Subprogram "fields" (aka methods)
    - Can’t call a method without an "object", an instance of the class
    - objvar.method(arg1,arg2) => implementation has objvar as a "hidden" parameter
  - Visibility in various forms
    - KPL -- all are visible (Public)
    - Public -- available to any with "access"
    - Private -- available only to class code (methods or static proc/funcs)
    - Protected (C++) -- available only to class and derived classes
      - visibility applies to both data fields and methods/functions/procedure
  - Many different syntaxes for classes -- C++ class X { ... } (often in a header file), X::method elsewhere
  - Constructors / constructor methods, possible overloading: run at allocation / coming into scope
  - Destructors: run at deallocation / leaving scope
  - Inheritance: "Base class" or "Super class": "core" functionality
  - "derived class" or "sub class": using base/super and adding both data and methods
    - Can override a method from base/super, possible data hiding
    - Methods to access base/super methods (data?) even if overridden
  - C++: concept of an abstract base class. declare methods but no implementation
    - Can’t instantiate a base class, have a derived class that implements all methods.

OO Basics (page 2)

- Control of inheritance: e.g. Queue extends List, but might not want List operations visible
  - private inheritance (C++, somewhat unique to C++): Queue may not want to expose list implementation.
    - most languages don't allow private inheritance
    - has-a vs is-a relationship:
      - chess board has chess pieces, chess pieces can move, specialized movements
      - pawn is-a chess piece, chess board has-a chess piece on it.
      - pawn and chess board wouldn't have the same base class (possibly)
- Objects and generics
  - Java Object type
  - C++ template classes
- "this" parameter
  - C/Pascal: Queue_add(Q,E);
  - C++ and others: queue Q; .... Q.add(E)
  - Both have 2 parameters: Q and E
  - Inside add (Queue_add) access is different. Q.add(E) has a hidden parameter
  - C++ this, others self, may be other names
- static methods and data fields
  - method does not have hidden parameter
  - data field is global, one copy for all instances of the class

Constructors/Destructors

- May have more than one constructor: C++, Smalltalk, Eiffel, Java and C#
  - overloading, different parameters
  - different names (e.g Complex type, new_cartesian, new_polar...)
  - order for base/derived classes ... base first
- Some languages don't have constructors/destructors (Modula-3 and Oberon)
  - Ada 95 has automatic constructor/destructor only for types derived from "Controlled"
- C++ copy constructor
  - Reference vs Value situation
  - Constructor: same name as class, can have several versions
  - class point { double x, y; point(double x1, y1); point (); point(point &p); }
  - point p1; // calls point();
  - point p2(1,1); // calls point(double x1,y1)
  - point p3 = p2; // calls point::point(point &p), copy constructor!
    - Difference: p3 = p2; // calls point::operator=(point&)
  - point p4(p2); // calls point(point &p); -- also a copy constructor
  - Copy constructor typically wants to do deep copy.
- Lot more to study in constructors in several languages .... but due to lack of time...

Dynamic Method Binding

- Part of inheritance/type extension: Derived class D has all the members (data,methods) of base class C
  - Memory layout?
- Example (used by book) class person {..., class student : public person... class prof : public person ...
  - student s; prof p; ... person *x = &s; or person *y = &p;
  - s.print_mailing_label(); p.print_mailing_label(); // works....
  - x -> print_mailing_label(); // What method gets called?
  - Possible methods
    - Base class only has print_mailing_label ... doesn't use anything specific to either
    - Each derived class has a print_mailing_label?
    - Typically in this way, base class has a print_mailing_label, derived classes override
    - Static -- use the base class version
    - Dynamic -- use the derived class version ... even though we have a pointer to base class
      - How?
    - Abstract Class / virtual function: class person { public: virtual void print_mailing_label(); ...
      - No code given in base class for print_mailing_label
      - Method table (vtable) in the object - table of pointers to virtual functions that are overridden
  - A lot more in 10.4 ... we need to move on
  - Read 10.5 Interface Inheritance
  - Multiple inheritance
    - Deriving from two base classes
      - class student : public person, public computer_user { ...
  - Book puts this material in the supplement ... but it is good stuff

Object-Oriented Programming recap

- Three aspects of object oriented programming
  - encapsulation & visibility control -- allows an abstraction to be hidden
  - inheritance -- allows reuse of an abstraction in a larger abstraction
  - dynamic method binding -- allows behavior to match current abstraction
- Languages support these in various ways
  - Pure object-oriented language?
    - Smalltalk and Ruby -- close
  - C++, Ada 95, Fortran 2003
    - van Neumann language that permits the programmer to write object=oriented