Why C++?

- a better C
  - type safe, e.g., I/O streams
  - better support for ADTs, encapsulation

- object-oriented programming
  - add inheritance to encapsulation
  - OO isn’t a silver bullet, but it helps in dealing with the complexity of software development

- non OO programming
  - sometimes non-class approach has merits

- generic programming
  - STL, the standard template library
Why inheritance?

● **standard shape example**
  ➤ difficult to extend
  ➤ access to source
  ➤ change *all* functions
  ➤ what is state?

● **inheritance**
  ➤ models *is-a*
  
  *Liskov substitution principle*
  
  ➤ cost? (who pays?)

```cpp
class Shape
{
    public:
        enum Kind {circle, square, ...};
        double area() const;
        void rotate();
    private:
        Kind myKind;
};

double Shape::area() const
{
    switch (myKind)
    {
        case Shape::circle:
            return PI * myDim * myDim;
        case Shape::square:
            //...
    }
}
```
Inheritance

- **virtual functions provide runtime polymorphism**
  - must use pointers or references
  - can extend base class without access to source
- **in C++ there are several kinds of inheritance**
  - public, private, virtual, ...
  - multiple inheritance
- **we’ll use single, public inheritance, trying to model “is-a” as much as possible**
- **General goal: base classes are abstract, have one pure virtual function**
  - model an interface, not an implementation
  - see roulette program for example, also map hierarchy
shapes: C++ features, what is “is-a”? 

```cpp
class Shape
{
    public:
        virtual double area() const = 0;
};

class Square : public Shape
{
    public:
        virtual double area() const {return mySide * mySide;}
    private:
        double mySide;
};

class Rectangle : public Shape
{
    public:
        virtual double area() const {return myWidth * myHeight;}
};
```
Designing for change (extension)

- How does inheritance help?

- What about different table formats in hyperwag?
  - is more than one layout possible? problems?

  ```java
  if (headerStyle == normal) ...
  else if (headerStyle == gaudy) ...
  else if ...
  ```

- factory pattern helps here
  - abstracts object creation, multiple look-and-feel
  - can use factory for reading format, writing format, ...
Factory Pattern

- Consider classes for different look-and-feel styles
  
  NormalLayout wagNL;  
  GaudyLayout wagGL;  
  wagNL.makeHeader();  
  wagGL.makeHeader();

  ➤ problems, how to choose styles, when to choose

- Does inheritance help? (assume ABC, Layout)
  
  Layout * wag = new NormalLayout;  
  Layout * wag = new GaudyLayout;  
  wag->makeHeader();

  ➤ is this better? are there problems?

- Use a Layout Factory
  
  Layout * wag = factory->makeLayout();  
  wag->makeHeader();

  ➤ dependencies exist, but in factory
  ➤ parameters to factory can set the look-and-feel returned
Design patterns

- Add to vocabulary we use to communicate and think about design
- Form part of a tool-kit we use when thinking about design
- Help find solutions to design problems

➤ factory
➤ iterator  see usewords.cc
➤ proxy
➤ also: adapter, singleton, observer, command
➤ also: model-view-controller, client-server, …
➤ GOF, gang-of-four, Gamma, Helms, Johnson, Vlissides
➤ we’re using Buschmann et al
usewords.cc

- **iterators**
  - internal, external
  - creation/deletion
  - use of inheritance and naming conventions

- **STL, Duke-classes**
  - what is a templated class/function?
  - how are templates instantiated?
  - drawbacks?
Data Structures, STL, Generic Programming

- **common container classes/data structures**
  - vector: growable array
  - map: dictionary, (search tree, hashtable)
  - set: union, intersection, membership

- **STL, Standard Template Library**
  - not just implementations, but way of thinking
  - little/no inheritance, lots of templates
  - algorithms and functions generalized too

- **default often not-safe, implementations make heavy use of “new C++ features”**
Thinking about thinking about hyperwag

- **issues in specification**
  - ambiguities
  - missing pieces

- **classes: nouns; member functions: verbs**
  - what, not how
  - responsibilities and collaboration among classes

- **other design/problem questions?**
Designing Classes and Programs

● Where do classes come from?
  ➤ nouns: brainstorm, cull, combine, divide
  ➤ attributes vs. classes, keep cohesion high

● Things to watch out for
  ➤ kitchen sink classes: keep classes highly cohesive
  ➤ God classes: know about everything in the app
  ➤ classes with little behavior (only get/set), and few collaborators

● CRC cards: classes, responsibilities, collaborators
  ➤ 3x5 card, class name at top, list responsibilities and collaborators
## CRC card template

<table>
<thead>
<tr>
<th>Class name</th>
<th>subclasses/superclasses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>responsibilities</td>
</tr>
</tbody>
</table>

**WagReader (abstract)**

<table>
<thead>
<tr>
<th>parseStream</th>
<th>Appointment</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAppointment?</td>
<td>Stream</td>
</tr>
<tr>
<td>iterators?</td>
<td>WagMaker</td>
</tr>
</tbody>
</table>

**Other classes in hyperwag?**
working in groups/teams

- work together/debate the design, don’t code until the design is understood by everyone
  - what about initial hyperwag prototype?
  - Doom: Design and implementation are iterative processes

- what about controlling the source code, building the program?
  - use RCS or CVS, start now (hopefully CVS soon, see web)
  - build/make everyday, versions ok, a working program is a wonderful thing
  - who’s in charge?
Brooks’ team (see chapter 3, *Mythical Man Month*)

- surgeon, chief programmer
- copilot
- administrator
- editor
- program clerk
- toolsmith
- tester
- language lawyer
- secretaries (2)

➤ necessary? what about three-person teams
Other team formats

● Everyone participates in design
● Everyone codes
● Someone has to be in charge
● What about different views, levels of experience?

● What can be done when someone doesn’t deliver?
● How to live with the team (or not)
● What’s the best way to get everyone to help?

● Don’t leave anyone behind, don’t get left behind