Announcements

• Midterm #2 is on Tuesday
  – Code released after class
  – Same format as Midterm #1
    • No essays
    • Some cases print multiple lines, and some questions ask you for all of them.
    – Unintentionally (unavoidably) cumulative
  – Any Questions?

• PA7 not assigned until next week
  – Released on web site after the exam

Object Oriented Topics

• Encapsulation
• Inheritance
  – As union
  – As abstraction
  – Optional (not a single tree)
  – Multiple inheritance
  – Pure virtual classes
  – Shadowing
• Dispatch
  – Static
  – Dynamic (virtual)
  – Implementation
• Slicing
• Static methods & members

Static Methods (Example)

```cpp
class Math {
  public:
    static double sqrt(double input);
};
```

Static Methods

• Static declaration precedes return type
  – Like ‘virtual’ or ‘inline’
• Static methods have no hidden argument
  – Therefore, they cannot access object fields
  – They cannot be declared const
    • i.e. no const after parameter list
    • Because there is no hidden variable to be const
• Static methods are stand-alone functions
  – An alternative is to just make them functions
  – Therefore, less common in C++ than Java
  – Semantically, static methods are bound to their class
    • As in the Math::sqrt example
    • In essence, the Math class is used like a namespace

Static Data Fields (Example)

```cpp
class CountedObject {
  public:
    CountedObject() { ctr++; }
    virtual ~CountedObject() { ctr--; }
    static int Count() { return ctr; }
    protected:
      static int ctr;
  };

// CountedObject.cpp
// initialize static data field
int CountedObject::ctr = 0;
```
Static Data Fields
• Most data fields are stored inside objects
• Static data fields are stored once per class
  – They are access-controlled global variables
  – Never use global variables. Static fields are better.
• Initialized in .cpp files
  – Initialization will happen before main is entered
• Useful for “per class” data
  – E.g. reference counting
  – Previous example
  – Efficient way of storing class properties
  – Every Quagga has the same body temperature, so...

End of Material for Midterm #2
... but I do have some odds and ends to discuss....

Friend (example)
Class ListNode {
  private:
  int element;
  ListNode* next;
  // Note private constructor
  ListNode(int theElement,
  ListNode *n = NULL)
  : element(theElement), next(n) {}

  friend class IntQueue;
};

Friend
• Friends are exceptions to access schemes
• If class A declares class B to be a friend
  – B can access the private data in A
  – Note that ‘friend’ gives access
    • It doesn’t take it
• In the previous example
  – ListNode’s constructor is private
  – So only IntQueues and ListNodes can make
    ListNodes
• Friendship is not inherited
  – Private data of classes that inherit A are not visible to B

Methods as Friends
Class ListNode {
  private:
  int element;
  ListNode* next;
  // Note private constructor
  ListNode(int theElement,
  ListNode *n = NULL)
  : element(theElement), next(n) {}

  friend void IntQueue::Push(int v);
};

Unsolicited Opinion
• ‘Friend’ is usually poor style in C++
  – Used as a fix for a poor OO design
  – Many style manuals ban it
• Avoid it whenever possible
  – OK as a patch before refactoring
  – OK with private constructor design pattern
• If you must use it:
  – Comment why (extensively)
  – Friend methods less bad than Friend classes
Incomplete Class Definition
(Example #1)

```cpp
class OddListElement;
class EvenListElement {
    public:
        void Push();
    protected:
        OddListElement* next;
};
class OddListElement {
    public:
        void Push();
    protected:
        EvenListElement* next;
};
```

Incomplete Class Declarations

- Allow a class to be used before it is declared
  - Useful for recursive data structures
- Limited to situations where the size of the class doesn't matter
  - In previous example, we declared a pointer
  - We could also have declared a reference
  - The body of Push() can use real objects
    - Because both have been declared before Push() is implemented (in .cpp file)

Incomplete Class Declaration
(Example #2)

```cpp
class Quagga;
class Herd {
    public:
        void Join(Quagga* new_member);
    protected:
        vector<Quagga*> members;
};
```

More Incomplete Declarations

- If nothing in class A's .h file needs to know the size of B
  - But it uses pointers or references to Bs
- Then put an incomplete declaration of B in A.h
- And include B.h in A.cpp
- This reduces compilation dependencies
  - A.h no longer depends on B.h
  - Therefore other files that include A.h don't on B.h
  - Although A.cpp does
  - If B.h is changed, fewer files need to be recompiled
  - Requires correct dependencies in Makefile