Review: Syntax of Overloading

• Like any other method or function, except
  – Name is `operator op`
  – Where “operator” is a keyword
  – Where `op` is the operator symbol being overloaded

Example of Overloading

```cpp
class Complex {
public:
    Complex(double real_part = 0.0, double imaginary_part = 0.0)
        : real(real_part), imaginary(imaginary_part) {}  
inline double Real() const {return real;}
inline double& Real() {return real;}
inline double Imaginary() const {return imaginary;}
inline double& Imaginary() {return imaginary;}
void Add(const Complex& add_in);
void Subtract(const Complex& sub_in);
void Multiply(const Complex& mult_in);
bool Divide(const Complex& div_in);
Complex operator + (const Complex& in) const;
Complex operator - (const Complex& in) const;
Complex operator * (const Complex& in) const;
Complex operator / (const Complex& in) const;
};
```

Review: What can be overloaded?

• Math:
  + - * / % == *= -= /= += %=
• I/O:
  << >>
• Logic:
  & | ! ~ <<= >>= ^ ^= & &=
• Indexing:
  () [] ++ -- , -> .* new delete new[] delete[]
• You cannot overload
  . * ? sizeof

When should you overload?

• When a method exactly matches the expectations we have about an operator
  – I/O
    – << and >>
  – Functors
    – ()
  – Numerics
    – + - / % += -= *= /= %=
  – Logic operators
    – | e.g. streams
  – Containers
    – []
  – Iterators
    – + ++
  – Sortables
    – <= >= 

Announcements

• Competing talks today @ 11AM
  – Cloud computing (Morgan Library): Dilma Da Silva
  – Software engineering (CSB 1130): Adrian Nistor
• Recitations:
  – Google Test (unit testing harness)
    – Mandatory
• P A5 is due Wednesday
  – Start reading as soon as you have time
  – Quiz will appear Wednesday
  – Quiz isn’t due until the Monday you get back
• No programming over spring break
• Reading: Chapter 6
  – Start reading as soon as you have time
  – Quiz will appear Wednesday
  – Quiz isn’t due until the Monday you get back
• Start of class.
Object Oriented Programming

• Why do we use objects?
  – Encapsulation
    • Collect related data & code together
      – Think about PA5...
  – Polymorphism
    • Goal: abstraction
      – Quaggas have properties
      – Some properties shared by all Equines
      – Some shared by all Mammals
      – Or all Animals
    • Write methods at most abstract level possible
      – Avoids repeating code

PA5 examples

• Reading a file has
  – Data
  – Stream
  – Line numbers
  – Operations
    • Read next Pose
    • Error checking
  • A file reader is a reasonable class

PA5 examples

• Points (Point3D) have
  – Data
    • X, Y & Z
  – Operations
    • Data access
    • Distance between two points
• Point3D is a reasonable class
  – That’s why I gave it to you

PA5 examples

• Poses contain
  – Data
    • 25 Point3Ds
  – Operations
    • Compare one pose to another
    • Normalize
      – Given translation and scale
• Pose is a reasonable class

PA5 examples

• Sequences of Poses contain
  – Data
    • Arbitrary length sequence of poses
  – Operations
    • Average position of base of spine
      – Use as translation
    • Scale
    • Normalization
    • Compare to another sequence of poses
• PoseSequence (or Video or...) is a reasonable class

PA5 examples

• Tables of values
  – Examples: D & W
  – Data
    • N x M array
  – Operations
    • Compute D from two Pose Sequences
    • Compute W from D
    • Output Table (PA4)
• Table is a reasonable class
OO: C++ vs Java

- In C++, objects are optional
  - In Java, all functions are methods of a class
  - In C++, they don't have to be

- When should you create an object?
  - Data supports unique operations
    - Encapsulation
  - When it shares properties/code with a more abstract form
    - Polymorphism
  - Efficiency should not be a consideration

When do you write stand-alone functions?

- Not manipulating state
  - Parameters are the only data touched
- No side-effects
  - Output value is the only result
  - No other state is changed (including parameters)
  - Example: sqrt
- Functions support parallelism
  - Move function to parameters (no other state)
  - Apply two functions to same parameters (no side effects)

OO: C++ vs Java (cont.)

- Java has an inheritance tree
  - Every class has exactly one parent
  - Object is the root of all classes
- C++ has an inheritance forest
  - No unique root
  - Many classes have no parent
  - A few classes have multiple parents
  - Inheritance relations form a directed acyclic graph

What is inheritance?

- If class B inherits class A, what does that mean?
  - Inheritance is union
    - Every field in A is included in B
    - Every method in A is included in B
    - B then adds its own fields and methods
    - Note: you cannot inherit part of a parent
  - Inheritance is mandatory in Java
    - Every class has a parent
    - Every object has extra data
  - Inheritance is optional in C++
    - Most classes have no parent
    - Makes inheritance “lighter”
    - Makes run-time typing harder

Inheritance vs Inclusion

- Many of you have defined a Pose class
  - Should it inherit vector?
  - Or just include a field of type vector?
  - Why?

- Remember: inheritance is union
  - Can you push_back onto a Pose?
  - That would create a 26th Point3D...
  - In this case, inclusion is what you want