Overloading Comparison Operators

CS 1037a – Topic 9

Some standard operators

- some *operators* are defined for all basic types like *int*, *float*, *char*,… (some are defined for *pointers* too)
  - operator+, operator-, operator+=, operator*, operator/
  - operator= (*copy* or *assignment* operator)
  - operator==, operator!=
  - operator<, operator>, operator<=
- standard operator [ ] is defined for *pointers* only
  - if p is a pointer, p[i] returns *(p+i) (more in Topic 2)

C++ allows to define/overload operators for objects of any class!

Overloading Operators

- Examples of operator overloading we saw earlier:
  - assignment (copy) operator= overloaded for *Point* (topic 1)
  - arithmetic operator+ and operator* overloaded for *Point* (topic 1)
  - operator [ ] overloaded (defined) for class *Table2D* (topic 2)

In many cases **operator overloading simplifies code and makes it read like standard math**.

Comparison of Objects

Expressions like a == b or a < b for two items a and b of the same class can be used only if operators == and < are overloaded (defined) for this particular class

Later in this course we will often need to overload (define the meaning of) operators like == or <

- to compare objects for equality (e.g. when searching for certain Item in a list)
- to compare which object has larger value (e.g. when sorting Items)

First, one must know exactly what *equality* or *less than* mean for two objects of a given class

Equality of Objects

- We have a strong intuitive grasp of what it means for two values of most primitive types (int, char, bool, *etc*) to be considered equal using operator ==
- With double values, testing for strict equality using == could be unsafe: many doubles can only be stored approximately

Equality of Objects

- What does it mean for two ***objects*** to be equal?
  - Do they share the same memory address (***i.e.****:* are they the same object)
  - Can they be separate objects with identical contents?
  - Can the condition for equality be weaker?

Example: Operator Overloading

- Recall the Point class: its original .h file is on the next slide
- First, we'll consider two points to be equal if they have identical coordinates
- We'll implement == and != ; comparison operators such as < won't make much sense in this context

Example: Operator Overloading (cont'd)

```
class Point {                                   // Point.h
public:
    Point( );
    Point( int x_init, int y_init );
    void shift( int x_amount, int y_amount );
    void rotate90( );
    int x, y;
 };
```

Example: Operator Overloading (cont'd)

- We'll declare == and != as ***non-member functions*** of class Point:
  - Prototypes come after the " }; " that marks the end of the member portion of the class declarations in the .h file
  - Same as what we did in examples 1-4 on slides 5-8 (operators + and *)
  - Non-member functions are automatically *public*

Example: Operator Overloading (cont'd)

```
// … the end portion of file Point.h

    ....

    int x, y;

};

 bool operator == (const Point& p1, const Point& p2 );

 bool operator != (const Point& p1, const Point& p2 );
```

Note: Non-member functions are not called against a specific object; the two objects used by the function have been passed here as *reference parameters for efficiency*

Example: Operator Overloading (cont'd)

```
// … function definitions added at the end of  Point.cpp
bool operator == ( const Point& p1, const Point& p2 ) {
    return (( p1.x == p2.x ) &&
            ( p1.y == p2.y );
}
bool operator != ( const Point& p1, const Point& p2 ) {
    return  ! ( p1 == p2 );
}
```

The normal operator == for integers is used here;

== for the Point class is used here

Example: Operator Overloading (cont'd)

- Technically, the names of these functions are "*operator ==*" and "*operator !=*"
- Function call looks like a normal use of the operator: == or != appears between two objects from the Point class
  - See previous slide

Example: Operator Overloading (cont'd)

- ***Alternative approach***: == and != could have been *member functions* in the public portion of Point
- Prototypes would then be:

```
// … in the public portion of file Point.h
bool operator == (const Point& p );
bool operator != (const Point& p );
```

Example: Operator Overloading (cont'd)

```
// … member functions are added to Point.cpp

bool Point :: operator == ( Point& p ) {
    return ( (x == p.x )  &&
             (y == p.y)   );
}
bool Point :: operator != ( Point& p ) {
    return ! ( (*this) == p );
}
```

In a member function one can use the keyword

this (pointer value)

to get address of the object against which the function is called

Whenever possible, we will use *non-member functions* when overloading operators

Example: Operator Overloading (cont'd)

- When overloading arithmetic operators, we must use non-member functions
- ***E.g.****:* When adding two Points, the result is a third Point
- Prototype for the overloaded + operator:

```
Point operator + (const Point& p1, const Point& p2 );
```

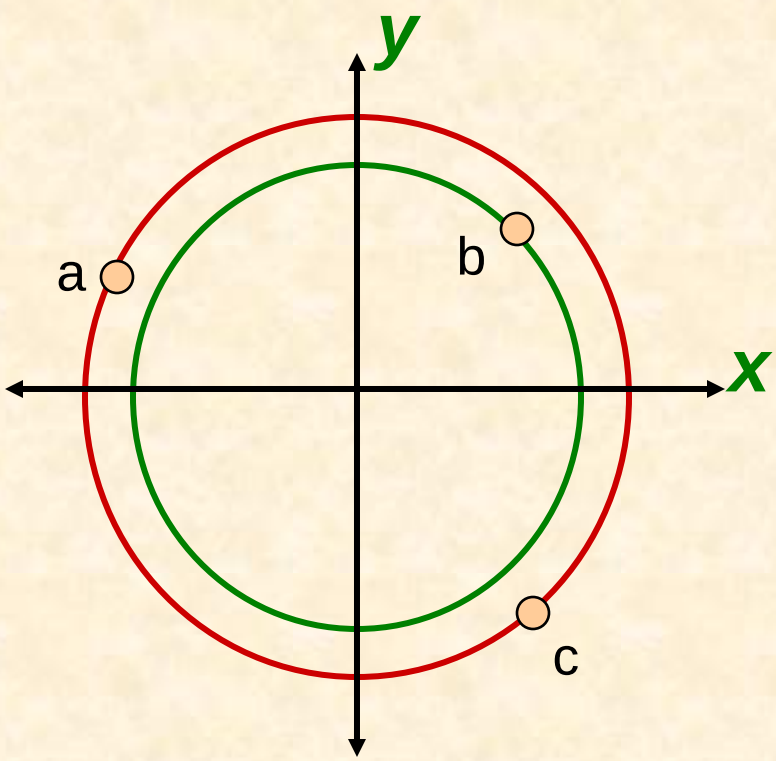
(see earlier slide 1-55 for code defining this operator)

Overloading Exercise

- We don’t have a clear concept yet of one Point being less than (or greater than) another
- Suppose we change the definition of equality so that all points that lie on a circle centered at the origin are considered equal

Overloading Exercise (cont'd)

- A point on a circle close to the origin is less than a point on a circle farther away
- Write functions to overload the operators ==, !=, <, <=, > and >= using this definition



a == c, and b < c