Operator Overloading

CMSC 202

Let's Take a Closer Look…

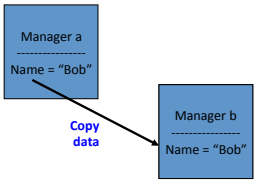
```
// In Employee.h
class Employee
{
   public:
    void SetManager(const Manager& boss);
   private:
     Manager m_boss;
};

// In Employee.cpp
void Employee::SetManager(const Manager& boss)
{
    m_boss = boss;
}

// In main…
Employee me;
Manager boss;
me.SetManager(boss);
```

Does this work?

If so, how???

Assignment Operator

- Compiler creates a default assignment operator
  - Copies data member values




```
Manager a("Bob");
Manager b;
b = a;
```

Not the same Manager!

Just have same data!

Other Operators?

- Does this work with other operators?

```
Money a(2, 50);     //   2.50
Money b(3, 20);     //   3.20
Money c;
c = a + b;
```

- Unfortunately, no…
  - But…we can define it ourselves!

Review: Function Overloading

```
void swap (int& a, int& b);
void swap (double& a, double& b);
void swap (Bob& a, Bob& b);
```

- Same (or similar) functionality for different types…
- Function signatures include
  - Function name
  - Parameter list (both number and types)
- Sidenote
  - C++ compiler has a built-in function called "swap"

Closer Look at Operators…

- We could do…

```
Money a(2, 50);     //   2.50
Money b(3, 20);     //   3.20
Money c;
c = Add(a, b);    // we write…
```

- Or…we can use
  - Operator Overloading and do this:

```
c = a + b;        // we write…
```

Operator Overloading

- Define a function that overloads an operator to work for a new type
- Example:

Function Name…essentially

```
const Money operator+ (const Money& a, const Money& b)
{
    return Money(a.GetDollars() + b.GetDollars(),
                 a.GetCents() + b.GetCents());
}
```

What's going on here?

How could this function be improved?

Operator Overloading

- Can also be overloaded as member functions
  - First object in statement becomes the "calling" object
    - `a + b`
      is equivalent to
      `a.operator+(b)`
- Example:

One parameter!

```
const Money Money::operator+ (const Money& b) const
{
    return Money(  m_dollars + b.m_dollars,
                   m_cents + b.m_cents);
}
```

Why const?

Notice: implicit object!

Return by const value?

```
const Money operator+ (const Money& a, const Money& b);
const Money operator+ (const Money& b) const;
```

- Why return by const value?
  - Imagine this
    - `Money a( 4, 50 );`
    - `Money b( 3, 25 );`
    - `Money c( 2, 10 );`
    - `(a + b) = c;`

Evaluates to an unnamed object if we don't return by const!

Why is this an issue?

Think about:

```
Money d;
d = (a + b) = c;
```

What is this supposed to mean? (`d` gets `c`'s value)

Return by const value prevents us from altering the returned value…

Why not return by const-ref?

```
const Money operator+ (const Money& a, const Money& b)
{
    return Money(a.GetDollars() + b.GetDollars(),
                 a.GetCents() + b.GetCents());
}
```

- Look closely…
  - We return a copy of a temporary Money object…
  - It goes out of scope when the function returns!

Other Operators?

- You can overload just about anything, but you should be VERY careful…
  - [ ]
  - * multiplication, pointer dereference
  - / division
  - + addition, unary positive
  - - substraction, unary negative
  - ++ increment, pre and post
  - -- decrement, pre and post
  - = assignment
  - <=, >=, <, >, ==, != comparisons
  - …
  - Many, many others…

Practice

- Let's overload the multiplication on money…
  - Ignore "roll-over"
  - Member function?
  - Non-member function?

```
// In Money.h
class Money
{
public:
    Money( int dollars, int cents );
    int GetDollars();
    int GetCents();
    void SetDollars( int dollars );
    void SetCents( int cents );

private:
    int m_dollars;
    int m_cents;
};

// In main…
Money m( 100, 00 );

m = m * 10;
```

Challenge

- Fix the multiplication operator so that it correctly accounts for rollover.

Challenge II

- Overload the + operator to add a Passenger to a Car:

```
class Car
{
   public:
      // some methods
   private:
      vector<Passenger> passengers;
};
```

Why is overloading the + operator this way not such a good idea?