Lecture 22

Chapter 8
Topics

- Basic operator overloading
  - What it is
  - How to do it
  - Returning by const value
- Unary operator overload
- Overloading member functions
- Constructors for automatic type conversion
  - Pitfalls
- Friend function
- Friend class
  - Forward declaration
- Reference
- Overloading << and >>
Basic operator overloading

• What it is
  – This allows you to define how these work
    • +,-,%, ==, and so on

• Example
  – //Declaration
    – const Money operator +(const Money& amount1, const Money& amount2);
Basic operator overloading

• Example
  – //Declaration
    – const Money operator +(const Money& amount1, const Money& amount2);
  – //Definition
    const Money operator + (const Money& amount1, const Money& amount2)
    {
      ...
    }
Returning const value

• Example
  – const Money operator +(const Money& amount1, const Money& amount2);

• This means that the returned value cannot be changed.

Money m1(10.99), m2(23.57);
(m1 + m2).output();  //This will work fine.
(m1 + m2).input();   //This will not.
Unary operator overload

• Unary operator is an operator that takes one operand only.
  – Example
    • $X = -Y; // The – is the unary operator.$
Overloading member functions

• Same as overloading operation functions but instead it is part of a class.

• Example
  – To do m1 + m2
  – const Money operator + (const Money& amount 2) const;
  – To do -m1
  – const Money operator - () const;
Constructors for automatic type conversion

Money baseAmount(100, 60), fullAmount;
fullAmount = baseAmount + 25;
fullAmount.output();
//Output should be 125.60

• But 25 is of type int.

• Money = Money + int; // There is no function
Constructors for automatic type conversion

Money baseAmount(100, 60), fullAmount;
fullAmount = baseAmount + 25;
fullAmount.output();
//Output should be 125.60

• If there is a constructor for just one int then this will work.

• Then
Money = Money + static_cast<Money>(25); //Will work
Constructors for automatic type conversion

• Note: fullAmount = 25 + baseAmount; //would not work

• This is because in class int there is no operator for + that takes a Money and returns Money.
Friend function

• This function is not part of the class
• But it is given access to private variables of that class.
• It has to have a declaration in the function to work.
Friend function

//Example
class Money:
{
 public:
     friend const Money operator +(const Money& amount1, const Money& amount2);
     ...
};
int main() { ...}

const Money operator +(const Money& amount1, const Money& amount2);
Friend Class

• Same as with friend functions but with a class.
• This requires a forward declaration of the friend class.

```cpp
class A;
//forward declaration

class B
{
public:
    ...
    friend class A;  //friend assignment
    ...
};
class A
{
    ...
};
```
Reference

• This is similar to pass by reference
  int robert;
  int & bob = robert;

• bob is not a reference to robert.
• This is also know as an alias
• Note: these can be confusing and therefore never used.
Overloading «<< and »>>

Money yourAmount;
cin >> yourAmount;

• Allows you to do this
  – cout «<< yourAmount;

• Instead of something like this
  – cout «<< yourAmount.getAmount();
class Money
{
public:
    friend ostream& operator <<(ostream& outputStream, const Money& amount);
}

int main(){…}

ostream& operator <<(ostream& outputStream, const Money& amount)
{
    outputStream << '$';
    outputStream << amountDollars;
    return outputStream;
}

• outputStream in this case is the cout from the call on the slide before.