

Operator Overloading

- Assignment Operator

Header:

```
ClassName& operator=(const ClassName& inputList)
```

Definition:

Needs to do a deep copy of data from one object to the other

- Copy all ints, doubles, and primitive types

- Force initialization, and copying of arrays and objects

- Manage dynamic allocation as appropriate

OOP: Operator Overloading

- Example: Assignment Operator

```
Automobile& Automobile::operator=(const Automobile  
    &inputAuto)
```

```
{
```

```
    if (this != &inputAuto) {
```

```
        year = inputAuto.year;
```

```
        model = inputAuto.model;
```

```
        delete [] tirePressures;
```

```
        tirePressures = new int[4];
```

```
        for (int j = 0; j < 4; j++) tirePressures[j] =  
            inputAuto.tirePressures[j]; } 
```

```
        return *this;
```

```
}
```

OOP: Operator Overloading

Purpose of assignment operator:

Set variables of calling object equal to variables of input object

Why have a return type of `Automobile&` and return something at the end?

```
Automobile& Automobile::operator=(const Automobile  
    &inputAuto)  
{  
    // copy data from inputAuto  
    return *this;  
}
```

OOP: Operator Overloading

- Why a return value?
 - Need to support multiple assignment statements

Automobile A, B;

Automobile C("Toyota", 2002, Other parameters);

A = B = C;

B = C => Copies member variables from C into A

A = (B=C evaluation) => What is B=C evaluation?

Return a copy of the Automobile just made, so it can be assigned to A. If don't pass anything back, potential that A could get garbage.

OOP: Operator Overloading

- Why returning reference values?
 - Allows returned object to act as an l-value (LHS of another assignment statement)
 - $(A=B) = C$; // this is legal for primitives, so it should be for our types as well
- Also need self-check at beginning of assignment operator
 - In assignment, clean up dynamically allocated data, allocate new memory
 - Self assignment – clean up blows away data; now no data to copy over either! (so you just lost your data!)

OOP: Operator Overloading

- Note that copy constructor and assignment are essentially equivalent code
- Same purpose: Copy the variables from one object into another, where another is already created (assignment) or just being created (copy constructor)

Equivalent statements:

Automobile A(B); A = B;

Copy Constructor Destructor of A
Followed by Assignment

OOP: Operator Overloading

- Different Operators have different semantics
 - Some directly affect the object for which the function is being called
 - Others generate a new object using the data from the current object and function parameters
 - Compare assignment (=) to addition (+)

OOP: Operator Overloading

■ Assignment:

- Object objectOne, objectTwo;
objectOne = objectTwo;
- objectOne's variables are updated to be equal to objectTwo's variables

■ Addition:

- Object objectOne, objectTwo, objectThree;
objectThree = objectOne + objectTwo;
objectOne is not changed, even though + is called on it

OOP: Operator Overloading

- Not all operators are member functions
 - Don't make sense in the context of working on the current object
 - Classic example: << (insertion) operator
 - `cout << r` is equivalent to `cout.operator<<(r);`
 - << used to push representation of object onto stream
 - << should only be defined for streams

OOP: Operator Overloading

- Declare << overloading as a free non-member function
- In header:
`friend ostream& operator << (ostream & out, const Rectangle & r)`
- In cpp file:

```
ostream& operator << (ostream & out, const Rectangle & r)
{
    out << r.xPosition << " " << r.yPosition << " " << r.width << " " << r.height
    << endl;
    return out;
}
```

Have to return out variable as a reference parameter, so can plug together multiple insertion operations:

```
Rectangle a, b;
cout << a << b;
```

OOP: Operator Overloading

- When using free, non-member functions:
 - Not actually calling member operation on the object, but instead passing object in
 - Can overload other operators this way, such as addition:

```
friend Rectangle& operator + (Rectangle& r1, Rectangle& r2);
```

Friends

- A function is granted friend status from a class, allowing it to read and write private parts of the class.
- Friendship is a one-way function.
 - The friend can read and write variables and functions from the granting class.
 - The granting class does not have any privileged status with the friend just because it made it a friend.

OOP: First Class Classes

- Really useful classes (for usage and programming) should have at least these five methods implemented:
 - Multi argument constructor
 - No argument constructor
 - Copy constructor
 - Assignment
 - Insertion (<<) operator