COP 3331
OBJECT ORIENTED DESIGN
SPRING 2017

WEEK 5: EVEN MORE ABOUT CLASSES AND OPERATOR OVERLOADING

SCHINNEL SMALL

USF
UNIVERSITY OF SOUTH FLORIDA

MEMBERWISE ASSIGNMENT AND COPY CONSTRUCTORS

MEMBERWISE ASSIGNMENT

- The = operator can be used to assign one object to another, or to initialize an object with other's data

- Given two objects, `obj2 = obj1;` copies all member values from `obj1` and assigns to the corresponding members variables of `obj2`

MEMBERWISE ASSIGNMENT

- Example: consider the following class definition:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle
{
   private:
      double width;
      double length;
   public:
      Rectangle(double, double);   // Constructor
      void setWidth(double);
      void setLength(double);
      double getWidth() const  { return width; }
      double getLength() const { return length; }
      double getArea() const { return width * length; }
};
#endif
```

MEMBERWISE ASSIGNMENT

- Declaring two objects of type Rectangle:

```
Rectangle box1(10.0, 10.0);
Rectangle box2(20.0, 20.0);
```

- Performing member assignment

```
box2 = box1;
```

- Can also perform memberwise assignment during initialization:

```
Rectangle box2 = box1;
```

MEMBERWISE ASSIGNMENT

- Memberwise assignment works well in most cases, except for one:

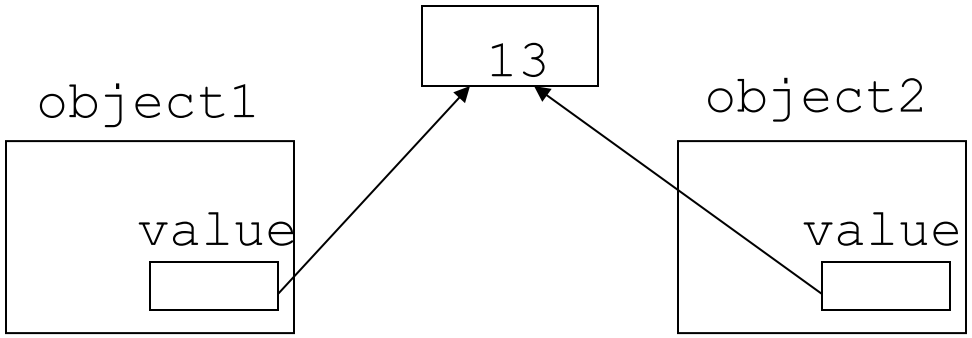
- Consider this class definition consisting of a pointer:

```
class SomeClass
{
   private:
      int *value;
   public:
      SomeClass(int val = 0)
      {value=new int; *value = val;}
      int getVal();
      void setVal(int);
}
```

MEMBERWISE ASSIGNMENT

- When we perform memberwise copy with objects containing dynamic memory

```
SomeClass object1(5);
SomeClass object2 = object1;
object2.setVal(13);
cout << object1.getVal(); // also 13
```



COPY CONSTRUCTORS

- The solution to this problem is to create a *copy constructor*

- A copy constructor is a special constructor that is called when an object is called with another object's data

- It has the same form as other constructors, except it has a reference parameter of the same type as the object itself
  - reference parameters MUST be used by copy constructors

COPY CONSTRUCTORS

- Syntax:

```
className(const className& otherObject);
```

  - Since copy constructors are required to use reference parameters, the const prevents the constructor from modifying the arguments data

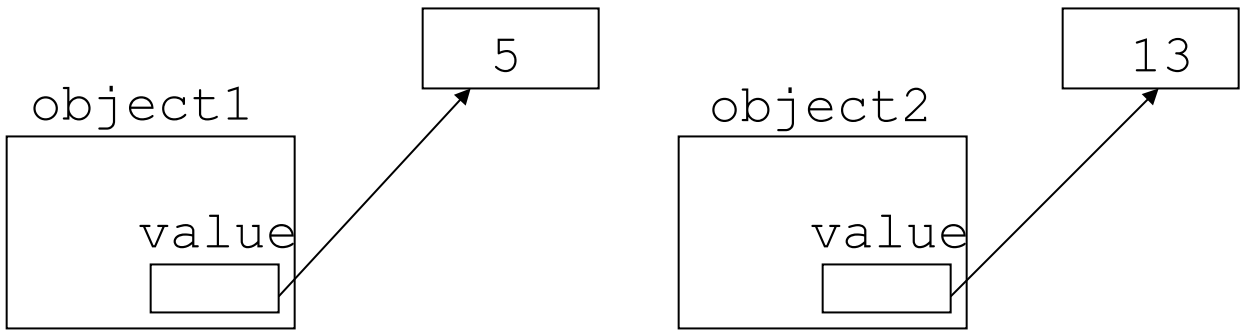
- Example:

```
SomeClass::SomeClass(const SomeClass &obj)
{
   value = new int;
   *value = obj.value;
}
```

COPY CONSTRUCTORS

- Each object now points to separate dynamic memory:

```
SomeClass object1(5);
SomeClass object2 = object1;
object2.setVal(13);
cout << object1.getVal(); // still 5
```



OPERATOR OVERLOADING

OPERATOR OVERLOADING

- C++ allows you to redefine how standard operators work when used with class objects

- Why is this necessary?
  - Assignment and member selections are the only built-in operations on classes
  - Therefore, other operators can't be applied directly to class objects

- Operator overloading provides a way to create more intuitive code

OPERATOR OVERLOADING…

- Consider:
  - Which would be preferable? (Suppose today is an object)

```
today.add(5);    OR    today += 5;
```

- Most existing C++ operators can be overloaded to manipulate class objects

- The `operator` function is used to overload the operator

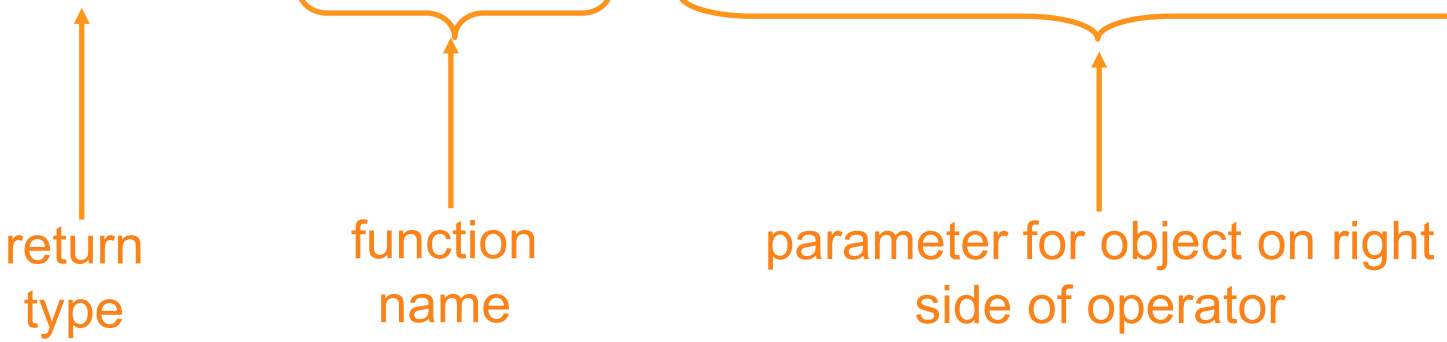
OPERATOR OVERLOADING…

- Syntax:

```
returnType operator operatorSymbol(formal parameter list)
```

- Example:

```
SomeClass operator=(const SomeClass &rval)
```



- Operator is called via object on left side

OPERATOR OVERLOADING…

- To overload an operator for a class:
  - Include operator function in the class definition
  - Write the definition of the operator function

- To call the overloaded operator function you could write:

```
object1.operator=(object2);
```

- However, you can call the overloaded operator in a more conventional form

```
object1 = object2;
```

OPERATOR OVERLOADING EXAMPLE…

- See Student Test Score Example on Canvas
  - In Week 5 examples

THE THIS POINTER

- Every object of a class maintains a (hidden) predefined *pointer to itself* called `this`
- When an object calls a member function, the `this` pointer is referenced by the member function
- The this pointer always points to the object of the class whose function is being called

OPERATOR OVERLOADING NOTES

- There are several rules/restrictions to consider when using operator overloading:

- C++ does NOT allow new operators to be created
  - This is why operator overloads are an option!

- Operator overloading is NOT automatic
  - Functions must be written to overload an operator

OPERATOR OVERLOADING NOTES

- Operator overloaded functions **must be non-static**, because they must be called on an object of the class and must operate on that object

- You do not have to perform overloaded operations on:
  - = (if you are performing memberwise assignment)
    - If you have a class with a pointer member, you should overload the operator (as shown in the example)
  - & (can return a pointer to the object)
  - , (evaluates left and right expression, returns right)

OPERATOR OVERLOADING NOTES

- Most of C++'s operators can be overloaded:

+	-	*	/	%	^	&	|	~	!	=	<
>	+=	-=	*=	/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++	--	->*	,	->
[]	()	new	delete								

- The following operators cannot be overloaded:

```
.   .*   ::   ?:   sizeof
```

- You cannot change an operators precedence, associativity or "arity" (e.g. binary, unary)

OPERATOR OVERLOADING NOTES

- Overloading Math Operators are very useful in classes
- ++, -- operators overloaded differently for prefix vs. postfix notation
- Overloaded relational operators should return a `bool` value
- Overloaded stream operators must return reference to `istream`, `ostream` and take `istream`, `ostream` objects as parameters

MATH OPERATOR OVERLOADING EXAMPLE

- See FeetInches code on canvas

OVERLOADING ++ AND -- OPERATORS

- There are different procedures for overloading pre and postfix operators

- Prefix syntax:
  - Prototype:
  - Definition:

```
className operator++();
```

```
className className::operator++()
{
      //increment the value of the object by 1
     return *this;
}
```

OVERLOADING ++ AND -- OPERATORS

- Example:

```
FeetInches FeetInches::operator++()
{
   ++inches;
   simplify();
   return *this;
}
```

- The function works as follows:
  - First, the function increments the object's inches member
  - The, it calls the simplify function and
  - The dereferenced this pointer is returned

OVERLOADING ++ AND -- OPERATORS

- The operator function allows the ++ to perform properly in statements like this:

```
distance2 = ++distance1;
```

- Remember, the above statement is equivalent to

```
distance2 = distance1.operator++();
```

OVERLOADING ++ AND -- OPERATORS

- To overload a post fix operator, you the following syntax:
  - Prototype:

```
className operator++(int);
```

  - Function Definition

```
className className::operator++(int u)
{
    className temp = *this;     //use this pointer to copy
                                //the value of the object
      //increment the object

    return temp;  //return the old value of the object
}
```

OVERLOADING ++ AND -- OPERATORS

- Example:

```
FeetInches FeetInches::operator++(int)
{
    FeetInches temp(feet, inches);
    inches++;
    simplify();
    return temp;
}
```

- This function works as follows:
  - The dummy parameter (int) tells the compiler that this function is designed to be used in postfix mode
  - The temporary local variable is a copy of the object being incremented before the increment takes place
  - The contents of temp is returned after inches is incremented and simplify called

OVERLOADING ++ AND -- OPERATORS

- See FeetInches version 2 on canvas

ANNOUNCEMENTS

ANNOUNCEMENTS

- No class next week due to Engineering event/ conference
- Assignment will be posted on Canvas
  - Deadline will be extended appropriately
- Slides will be posted on Canvas for next week
  - Assignment will be posted based on those slides too
- Next class meeting will involve review for midterm