CISC181 Introduction to Computer Science

Dr. McCoy

Lecture 25
December 1, 2009

8.1 Introduction

- Use operators with objects (operator overloading)
  - Clearer than function calls for certain classes
  - Operator sensitive to context
- Examples
  - <<
    - Stream insertion, bitwise left-shift
  - +
    - Performs arithmetic on multiple types (integers, floats, etc.)
- Will discuss when to use operator overloading

8.2 Fundamentals of Operator Overloading

- Types
  - Built in (`int`, `char`) or user-defined
  - Can use existing operators with user-defined types
    - Cannot create new operators
- Overloading operators
  - Create a function for the class
  - Name function `operator` followed by symbol
    - `operator+` for the addition operator `+`

8.2 Fundamentals of Operator Overloading

- Using operators on a class object
  - It must be overloaded for that class
    - Exceptions:
    - Assignment operator, `=`
      - Memberwise assignment between objects
    - Address operator, `&`
      - Returns address of object
    - Both can be overloaded
- Overloading provides concise notation
  - `object2 = object1.add(object2);`
  - `object2 = object2 + object1;`

8.3 Restrictions on Operator Overloading

- Cannot change
  - How operators act on built-in data types
    - I.e., cannot change integer addition
  - Precedence of operator (order of evaluation)
    - Use parentheses to force order-of-operations
  - Associativity (left-to-right or right-to-left)
  - Number of operands
    - `&` is unitary, only acts on one operand
- Cannot create new operators
- Operators must be overloaded explicitly
  - Overloading `+` does not overload `+=`

8.3 Restrictions on Operator Overloading

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

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

8.4 Operator Functions As Class Members Vs. As Friend Functions

- Operator functions
  - Member functions
    - Use `this` keyword to implicitly get argument
    - Gets left operand for binary operators (like `+`)
    - Leftmost object must be of same class as operator
  - Non member functions
    - Need parameters for both operands
    - Can have object of different class than operator
    - Must be a `friend` to access `private` or `protected` data
  - Called when
    - Left operand of binary operator of same class
    - Single operand of unitary operator of same class

8.4 Operator Functions As Class Members Vs. As Friend Functions

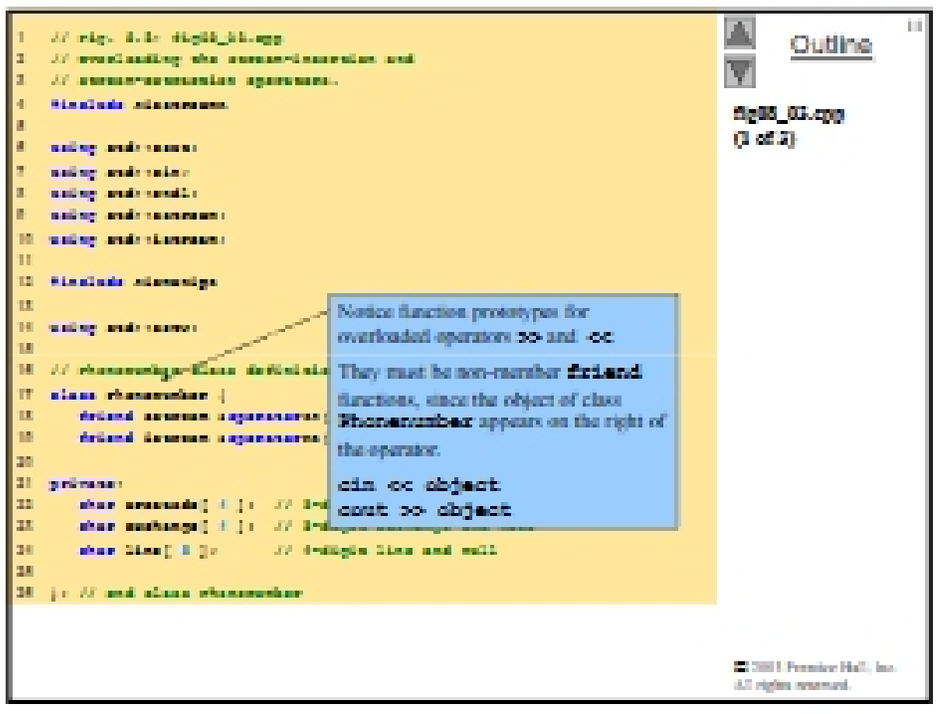
- Overloaded << operator
  - Left operand of type `ostream &`
    - Such as `cout` object in `cout << classObject`
  - Similarly, overloaded `>>` needs `istream &`
  - Thus, both must be non-member functions

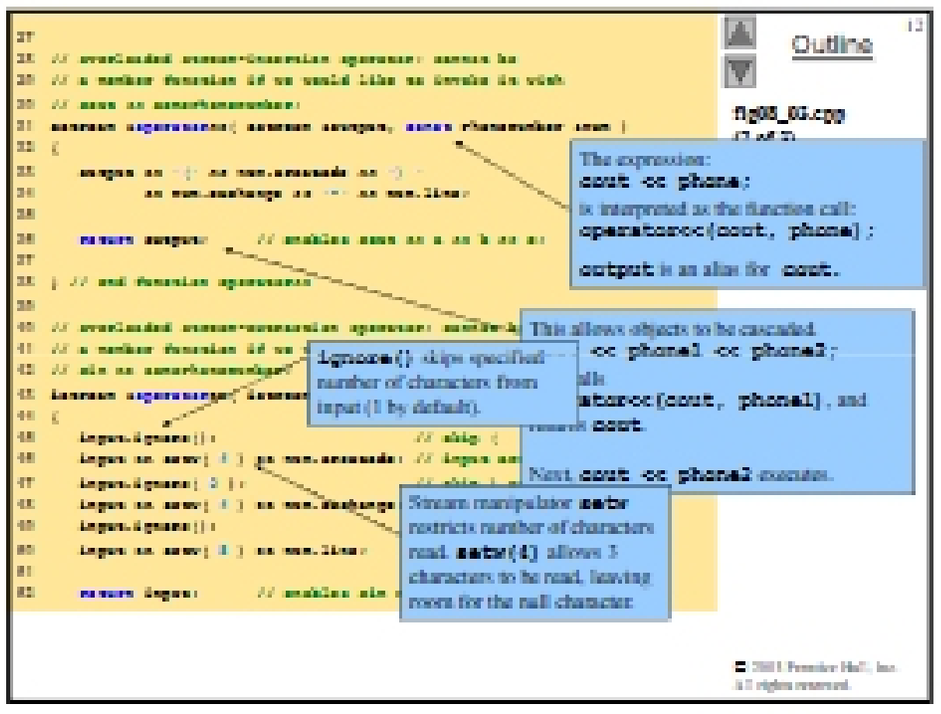
8.4 Operator Functions As Class Members Vs. As Friend Functions

- Commutative operators
  - May want `+` to be commutative
    - So both "`a + b`" and "`b + a`" work
  - Suppose we have two different classes
  - Overloaded operator can only be member function when its class is on left
    - `HugeIntClass + Long int`
    - Can be member function
  - When other way, need a non-member overload function
    - `Long int + HugeIntClass`

8.5 Overloading Stream-Insertion and Stream-Extraction Operators

- << and >>
  - Already overloaded to process each built-in type
  - Can also process a user-defined class
- Example program
  - Class `PhoneNumber`
    - Holds a telephone number
  - Print out formatted number automatically
    - `(123) 456-7890`





```
} // end function operator!

int main()
{
   PhoneNumber phone; // create object phone

   cout << "Enter phone number in the form (123) 456-7890:\n";

   // cin >> phone invokes operator>> by implicitly issuing
   // the non-member function call operator>>( cin, phone )
   cin >> phone;

   cout << "The phone number entered was: " << phone;

   // cout << phone invokes operator<< by implicitly issuing
   // the non-member function call operator<<( cout, phone )
   cout << phone << endl;

   return 0;

} // end main
```

```
Enter phone number in the form (123) 456-7890:
(800) 555-1212
The phone number entered was: (800) 555-1212
```

Outline

fig08_03.cpp
(2 of 2)

fig08_03.cpp
output (2 of 2)

8.6 Overloading Unary Operators

- Overloading unary operators
  - Non-`static` member function, no arguments
  - Non-member function, one argument
    - Argument must be class object or reference to class object
  - Remember, `static` functions only access `static` data

8.6 Overloading Unary Operators

- Upcoming example (8.10)
  - Overload ! to test for empty string
  - If non-`static` member function, needs no arguments
    - `!a` becomes `a.operator!()`

```
class String {
public:
   bool operator!() const;
   ...
};
```

  - If non-member function, needs one argument
    - `a!` becomes `operator!(a)`

```
class String {
   friend bool operator!( const String & )
   ...
}
```


8.7 Overloading Binary Operators

- Overloading binary operators
  - Non-`static` member function, one argument
  - Non-member function, two arguments
    - One argument must be class object or reference
- Upcoming example
  - If non-`static` member function, needs one argument

```
class String {
public:
   const String &operator+=( const String & );
   ...
};
```

  - `y += z` equivalent to `y.operator+=( z )`

8.7 Overloading Binary Operators

- Upcoming example
  - If non-member function, needs two arguments
  - Example:

```
class String {
   friend const String &operator+=(
                String &, const String & );
   ...
};
```

  - `y += z` equivalent to `operator+=( y, z )`