Friends and Unary Operators

Lecture 13
Sections 11.3, 11.6

Robb T. Koether

Hampden-Sydney College

Wed, Feb 15, 2017

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

Outline

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

Operators as Friends

Definition (Friend)

A friend of a class is a function or a class that is given access to the private members of that class through the keyword **`friend`**.

- The class must declare who its friends are.

Operators as Friends

Friends

```
class class-name
{
    friend function(parameters);
    friend other-class-name;
}
```

- To make a function (an operator) or another class a friend of *this* class, use the **`friend`** keyword in the definition of *this* class.

Writing Operators as Friends

- Declare the operator to be a friend of the class.
- Write the operator as a non-member function, except that...
- The operator may access the data members of the operands directly.

Operators as Friends: Considerations

- Advantages
  - Only one function call is needed (no facilitator or inspector needed).
  - The operator has direct access to the data members.
- Disadvantages
  - “Friendship” violates the data-hiding principle.
    - Now that the function has access to the private data members, it can do anything it wants.
    - The integrity of the class is no longer under the control of the class.

Choosing a Method

- The preferred method is to use facilitators.
- Exceptions
  - Operators that must be member functions (e.g., `=`, `[]`).
  - Unary operators (e.g., `++`, `--`)
  - Binary operators in which the left operand will always be an object of the class (e.g., matrix multiplication).
- In the exceptional cases, write the operator as a member function.
- Only in very rare cases will we use friends.

Outline

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

Outline

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

Unary Operators

- The following operators must be implemented as member functions.
  - The assignment operator `=`.
  - The subscript operator `[]`.

The Subscript Operator

Prototypes

```
type2 operator[](type1) const;     // Returns r-value
type2& operator[](type1);          // Returns l-value
```

- *type1* can be any type, but it is usually **int**.
- The operator will return a value of *type2*.
- *type1* and *type2* can be the same.

The Subscript Operator

Usage

```
int temp = list[i];          // r-value
list[i] = list[i + 1];       // l- and r-values
list[i + 1] = temp;          // l-value
```

Outline

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

Unary Operators

- Unary operators should be implemented as member functions.
- The operator is invoked by a single operand.
- The expression `*a` is interpreted as `a.operator*()`
- There is no issue of left operand vs. right operand.

Outline

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

The Pre-Increment Operator

The Pre-Increment Operator

```
type& type::operator++()
{
//  Increment the object
        ⋮
    return *this;
}
```

- The pre-increment operator should return the object *by reference*.
- The expression uses the returned value.
- What will `++(++a)` do?

The Post-Increment Operator

- The post-increment operator should return the object *by value*.
- Include one unused and unnamed **`int`** parameter to distinguish post-increment from pre-increment.
- The designers of C++ apologize for this completely artificial mechanism.

The Post-Increment Operator

The Post-Increment Operator

```
type type::operator++(int)
{
    type original = *this;
//  Increment the object
        ⋮
    return original;
}
```

- The expression uses the returned value.
- What will `(a++)++` do?
- What about `++(a++)` and `(++a)++`?

The Increment and Decrement Operators

Example (The Increment and Decrement Operators)

- `RationalIncrement.cpp`.
- `IncrementTest.cpp` will test `++(++a)`, `(++a)++`, `++(a++)`, `(a++)++`, etc.

Outline

1. Operators as Friends
2. Operators as Member Functions
   - Operators that Must be Member Functions
   - Unary Operators
3. The Increment and Decrement Operators
4. Assignment

Assignment

Assignment

- Read Sections 11.3, 11.6.