# Operators

## Lecture 12
## Section 14.5

Robb T. Koether

Hampden-Sydney College

Fri, Feb 9, 2018

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. Operators as Member Functions
5. Facilitators
6. Assignment

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. Operators as Member Functions
5. Facilitators
6. Assignment

# Operators as Functions

## Definition (Operator)

An operator is a function that can be represented by a symbol, such as $+$ or $*$.

- Different operators may have different numbers of arguments
  - Unary operators have 1 argument.
  - Binary operators have 2 arguments.
  - Ternary operators have 3 arguments.

# Operators as Functions

- Unary operators may be prefix or postfix.
- Binary operators are infix.
- Most unary operators are prefix.
- What is an example of a prefix unary operator?
- What is an example of a postfix unary operator?

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. Operators as Member Functions
5. Facilitators
6. Assignment

# Operator Overloading

- Most operators can be overloaded.
  - Unary: `+`, `-`, `*`, `&`
  - Arithmetic: `+`, `-`, `*`, `/`, `%`
  - Equality: `==`, `!=`
  - Order: `<`, `>`, `<=`, `>=`
  - Logical: `&&`, `||`, `!`
  - Bitwise: `&`, `|`, `~`, `^`
  - Shift: `<<`, `>>`
  - Assignment: `=`
  - Compound assignment: `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=`
  - Increment and decrement: `++`, `--`
  - Allocation: **`new`**, **`delete`**
  - Miscellaneous: `,`, `->*`, `->`, `()`, `[]`

# Operator Overloading

- A few operators cannot be overloaded.
  - Member access: `.`
  - Member access: `.*`
  - Scope: `::`
  - Selection: `?:`
  - Size of: **`sizeof`**

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. Operators as Member Functions
5. Facilitators
6. Assignment

# Operators as Non-member Functions

- The name of an operator function consists of the keyword operator followed by the symbol for the operator.
- The expression

  ```
  a + b
  ```

  is interpreted as

  ```
  operator+(a, b)
  ```

- As a non-member function, an operator does not have direct access to the private data members of the objects.
- However, it may use the inspector functions to get copies of the data members.
- Although this works, it can be very awkward.

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. **Operators as Member Functions**
5. Facilitators
6. Assignment

# Binary Operators as Member Functions

## Operator as Member Function

```
Type ClassName::operator+(Parameters);
```

- An operator may be defined as a member function of a class.
- That may or may not be a good idea.
- A binary operator is invoked by the left operand of the expression.
- Thus, the expression `a + b` is interpreted as `a.operator+(b)`.

# Binary Operators as Member Functions: Considerations

- Advantage
  - The operator has access to the private members of the left operand.
- Disadvantages
  - If `a` and `b` are objects of different classes, then `a + b` and `b + a` will invoke different functions.
  - The left operand may be a member of a class that we do not have access to.

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. Operators as Member Functions
5. Facilitators
6. Assignment

# Implementing Binary Operators with Facilitators

## Definition (Facilitator)

A facilitator is a member function that is invoked by a non-member operator.

- The facilitator performs the work of the operator.
- The operator simply
  - Invokes the facilitator.
  - Returns the appropriate object, typically the same one that is returned by the facilitator.

```
return a.facilitator(b);
```

# Binary Operators with Facilitators

- A binary operator has two parameters.
- The corresponding facilitator has one parameter, namely, the right operand.
- If we write the facilitator as a member function, then we write the operator as a non-member function.
- The operator is invoked by the operands as an *ordered pair*.
- We may then use *either* operand to invoke the facilitator.

# Binary Operators with Facilitators: Considerations

- Advantages
  - The left operand need not be an object of the same class as the facilitator.
  - The expressions `a + b` and `b + a` can be handled by the same facilitator, even if `a` and `b` are objects of different types.
- Disadvantage
  - Requires an additional function call.

# Using Operators with Mixed Types

### Example (Operators with Mixed Types)

```
Point operator+(double s, const Point & p)
{
    return p.scalarMultiply(s);
}
Point operator*(const Point& p, double s)
{
    return p.scalarMultiply(s);
}
```

# Outline

1. Operators as Functions
2. Operator Overloading
3. Operators as Non-member Functions
4. Operators as Member Functions
5. Facilitators
6. Assignment

# Assignment

## Assignment

- Read Section 14.5.