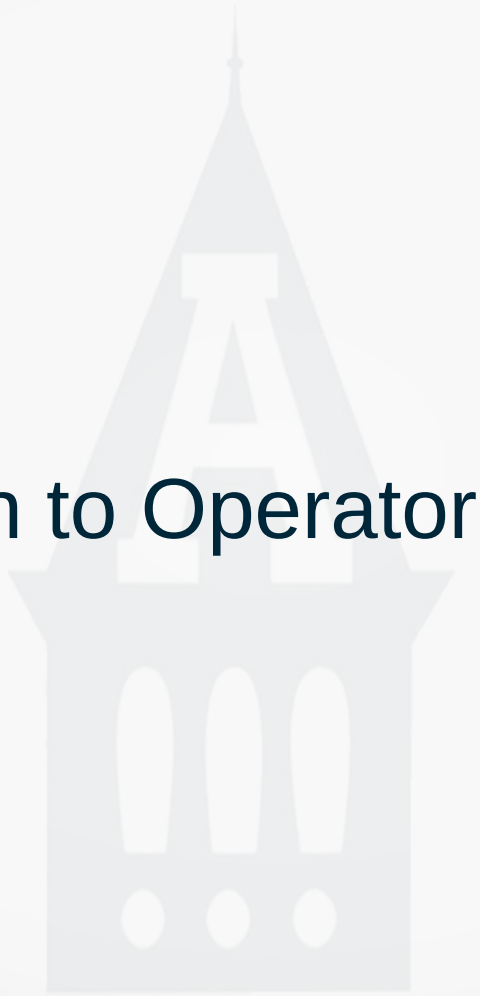
CS 3460



Introduction to Operator Overloading

Operator Overloading

- The ability to define behavior of operators on user defined types
- Some of these operators are: `+ - * / == !=`
- No similar concept in Java; has a different philosophy
- Generally, but not always, implemented as methods in a class
  - Overloaded operators ***are*** methods/functions
  - No single general form, depends on operator

Code Demo – Employee & Company Classes

Operator Overloading

- `Company` lacks a way to add a new `Employee`
  - Could write an `addEmployee(…)` method; good idea
  - But we want to learn operator overloading, so let's use the `+=` operator instead
- General form of the `+=` operator

```
[class]& operator+=(const [type]& rhs) { … body … }
```

- A few notes…
  - Use of the `const` reference parameter. `const` not required, but probably a good idea
  - Reference type also not necessary, but prevents copy

+= Operator Example

- The following is the `+=` implementation for `Company`

```
Company& operator+=(const std::shared_ptr<Employee>& employee);


Company& Company::operator+=(const std::shared_ptr<Employee>& employee)
{
    if (employee == nullptr)
        return *this;

    this->m_employees.push_back(employee);

    return *this;
}
```

Code Demo – Adding Employees

-= Operator Example

- Let's do another, the -= operator

```
Company& operator-=(const std::shared_ptr<Employee>& employee);

Company& Company::operator-=(const std::shared_ptr<Employee>& employee)
{
    if (employee == nullptr)
        return *this;

    auto iterator = std::remove_if(
        m_employees.begin(), m_employees.end(),
        [employee](std::shared_ptr<Employee> test) {
            return *test == *employee; // See next slide
        });

    m_employees.erase(iterator);

    return *this;
}
```

Comparing Employees

- Need to provide an equality operator ==

```
bool operator==(const Employee& rhs);

bool Employee::operator==(const Employee& rhs)
{
    return m_nameFirst == rhs.m_nameFirst &&
           m_nameLast == rhs.m_nameLast;
}
```

- Then can remove like this...

```
myCompany -= myCompany.findbyName("Larry", "Stackhouse");
```

-= Operator Implementation Notes

- *Erase-Remove* idiom
  - Mark items for removal (place them at end of the container...if possible)
  - In second step, use the `.erase()` member to remove in a single pass
- Why do this?
  - Remove multiple items in one `remove_if`
  - Improved performance through one or more passes to mark for removal and then one pass for removal

Assignment Operator

- Consider the following code

```
Employee e1("Luke", "Seamons", 0);
Employee e2("Lapriel", "Sanders", 0);

e1 = e2;

std::cout << e1.getFullName() << std::endl;
```

- This does what you likely want; and we didn't do anything!
  (demonstrate this code)
- C++ provides a default = operator implementation
  - Member by member copy; regardless of visibility
  - Not typical for other operators, this is an exception

Assignment Operator Implementation

- Let’s provide an implementation

```
Employee& operator=(const Employee& rhs);

Employee& Employee::operator=(const Employee& rhs)
{
    m_nameFirst = "Copy of " + rhs.m_nameFirst;
    m_nameLast = "Copy of " + rhs.m_nameLast;
    m_yearsOfService = rhs.m_yearsOfService;

    return *this;
}
```

(and demonstrate)

Operator Overloading Notes

- Overloading the + operator and = operator doesn't mean the += operator is overloaded
  - Each must be individually overloaded
- Similarly with the < and == operators
- Great Stack Overflow post about the "basic rules and idioms for operator overloading"; read it!

https://stackoverflow.com/questions/4421706/what-are-the-basic-rules-and-idioms-for-operator-overloading

Operator Overloading Notes

- Remember there are pre and post `++` and `--` operators.
  - post-increment: `class operator++(int);`
    - Semantics
      - make a copy
      - increment the value (not the copy)
      - return the copy
    - Return type is by-value
  - pre-increment: `class& operator++();`
    - Semantics
      - increment the value
      - return the incremented value
    - Return type is by-reference