OntarioTech Science

Kourosh Davoudi
kourosh@uoit.ca

Week 6: Operator Overloading

CSCI 1061: Programming Workshop II

Learning Outcomes

In this week, we learn:

- Foundation of operator overloading
- Define member operator functions:
  - Binary (e.g., =, +)
  - Unary (e.g., ++)
- Define helper function for overloading

What is Operator Overloading?

- The core language defines the logic for the operands of fundamental types:

```
        int x =1, y=2;    // Initialization
        x = x + y;    // Assignment
```

- Is it possible to define (overload) operators so that they can be used with operands of class type?

```
        Student x("Sarah", 115, 75.6) ;

        y = x + 3 ;        // for example, we can define + for
                           // Student and int
```

How to Overload Operators?

```
Student x("Sarah", 115, 75.6) ;
x = y + 3;
```

- You need to define following function:
  - Helper function: `Student operator+(Student, int)`

For `y + 3` we call `operator+(y, 3)`

?

OR

  - Member function: `Student operator+(int)`

For `y + 3` we call `y.operator+(3)`

Important Background

- Think about the operator as a function (always returns a value)

```
x + 3;   // x: 1'st argument, y: 2'nd argument
```

- Some operators have a side effect

```
x  = 2 + (y = 3);
z += 2;
k++;
```

They change the operand

- Some operators should support cascading:

```
x = y = z ;    // x =(y =z) , so (y=z) should be z in order to work
cout << x << y;    // (cout << x) << y, so (cout << x) should be cout
```

Which Operator Can be Overload?

- C++ lets us overload the following operators:
  - Binary arithmetic (+ - * / %)
  - Assignment - simple and compound (= += -= *= /= %=)
  - Unary - pre-fix post-fix plus minus (++ -- + -)
  - Relational (== != > >= < <=)
  - Logical (&& || !)
  - insertion, extraction (<< >>)

Helper or Member Function?

- We overload operators in either of two ways, as:
  - Member operators - part of the class definition
  - Helper operators - supporting, but outside the class definition (usually friend)

Some Limitations:

- For (assignment) operator= overloading function must be declared as a class member.
- When an operator function is implemented as a member function, the leftmost (or only) operand must be an object (or a reference to an object) of the operator’s class.

Overloading as a Member Function

- The signature of an overloaded member operator consists of:
  - the operator keyword
  - the operation symbol
  - the type of its right operand, if any
  - the const status of the operation

Shows + cannot change any operand

Example:

```
Type2 operator+(Type1)  const ;  // A + B
Note: A+B   calls   A.operator+(B)
```

Overloading as a Member Function

Example: operator= // Assignment

```
Student & operator=(const Student &);
```

```
Student & Student::operator=(const Student &d)
{
    id = d.id;
    grade = d.grade;
    delete [] name;

    name = new char[strlen(d.name)+1];
    strcpy(name,d.name);

    return *this;     // this the assigned object
}
```

Note:

- We do not change c
- = has side effect

```
a = c;
```

We call

```
a.operator=(c)
```

- Assign c to a (side effect)
- Returns c

Overloading as a Member Function

Example: operator+=   // Assignment

```
Student & operator+=(const int &);
```

```
Student & Student::operator+=(const int &g)
{
    grade += g;
    return *this;
}
```

Note:
- We do not change c
- = has side effect

```
a += 4;
```

We call

```
a.operator+=(4)
```

- Assign new value to a (side effect)
- Returns the new value

Overloading as a Member Function

Example: operator++ // Increment (postfix and prefix)

```
Student & operator++(); // prefix
Student operator++(int); // postfix
```

```
Student & Student::operator++()
{
    grade++;
    return *this;
}
```

```
Student Student::operator++(int)
{
    Student temp = *this;
    grade++;
    return temp;
}
```

```
++a
a.operator+=()
```

```
a++
a.operator+=(int)
```

- Assign new value to a (side effect)
- Returns the new value

Overloading as a Helper Function

- Good candidates: Those who do not change the operands
  - Example

== , + , -

- You have to define >> and << as helper functions and NOT member

>> , <<

(The reason is that the leftmost operand is cin
or cout and not our class type)

Overloading as a Helper Function

Example: operator== //

friend bool operator==(const Student &, const Student &);

```
bool operator==( const Student &s, const Student &t)
{
    return s.id == t.id;
}
```

```
(a == b);
```

We call

No side effect!

```
operator==(a, b)
```

Overloading as a Helper Function

Example: operator<<   //

friend ostream & operator<<(ostream &, const Student &);

```
ostream & operator<<(ostream & os, const Student &s)
{
    os << "\tname: " << s.name << endl;
    os << "\tID: " << s.id << endl;
    os << "\tGrade " << s.grade << endl;
    return os;
}
```

```
cout << a;
```

We call

```
operator<<(cout, a)
```

Overloading as a Helper Function

Example: operator>>

friend istream & operator<<(istream &, Student &);

```
cin >> a ;
```

We call

```
Operator>>(cin, a)
```

```
istream & operator>>(istream & is, Student &s)
{
    char tmp_name[200];
    int tmp_id;
    double tmp_grade;


    cout << "\tname: ";
    is >> tmp_name;

    cout << "\tID: ";
    is >> tmp_id;

    cout << "\tGrade ";
    is >> tmp_grade;

    s = Student(tmp_name,tmp_id, tmp_grade);

    return is;
}
```