Operator Overloading

Module 4

OOP (IT 2005)

4th Semester ECSc

Contents

What is an Operator Overloading?

- An operator overloading is an approach that allows programmers to define an operator to a class by overloading the built-in operator.
- This enables the programmer to perform specific computation when the operator is applied on class objects.
- For example, assume we have three objects c1, c2 and c3 from a class Complex.
- We can perform c3 = c1 + c2 if we overload + operator.

Syntax of Operator Overloading

```
class class_name
{
   private:
     // data members
   public:
     // constructors
     return_type operator symbol (arguments);
     // member functions
     // destructor
};
```

Operator overloading declaration

```
return_type class_name :: operator symbol (arguments)
{
    // function body
}
```

Definition outside class body

Overloading of Unary Operators

- Unary operators:
- Prefix ++ operator
- Postfix ++ operator
- Prefix -- operator
- Postfix -- operator

Recall:
y = x++; // postfix ++
is equivalent to
y = x
x = x+ 1

Recall:
y = ++x; // prefix ++
is equivalent to
x = x + 1
y = x

Program Example 4.1

- Write a C++ program that demonstrates the overloading of prefix and postfix increment operators.

```
#include<iostream>
using namespace std;

class Number
{
    private:
        int x;
    public:
        Number();
        Number(int n);
        Number operator++(); // overloading prefix increment operator
        Number operator++(int y); // overloading postfix increment operator
        void display();
};
```

```
#include<iostream>
using namespace std;

class Number
{
    private:
        int x;
    public:
        Number();
        Number(int n);
        Number operator++(); // overloading prefix increment operator
        Number operator++(int y); // overloading postfix increment operator
        void display();
};
```

Notice the int inside the parenthesis; it is a syntax not a function parameter.

```
Number::Number()
{
    x = 0;
}
Number::Number(int n)
{
    x = n;
}
Number Number::operator++()
{
    ++x;
    return Number(x);
}
Number Number::operator++(int y)
{
    y = x++;
    return Number(y);
}
```

```
void Number::display()
{
    cout << x << endl;
}
```

Alternative codes

```
Number Number::operator++()
{
    ++x;
     Number N(x);
    return N;
}
```

```
Number Number::operator++(int y)
{
    y = x++;
     Number N(y);
    return N;
}
```

```
Number::Number()
{
    x = 0;
}
Number::Number(int n)
{
    x = n;
}
Number Number::operator++()
{
    ++x;
    return Number(x);
}
Number Number::operator++(int y)
{
   y = x++;
    return Number(y);
}
```

```
void Number::display()
{
    cout << x << endl;
}

  int main()
  {
      Number N1(10);
      Number N2;
      N2 = ++N1;
      cout << "Prefix increment: ";
      N2.display();
      Number N3(20);
      Number N4 = N3++;
      cout << "Postfix increment: ";
      N4.display();
      cout << "N3: ";
      N3.display();
      return 0;
  }
```

```
Number::Number()
{
    x = 0;
}
Number::Number(int n)
{
    x = n;
}
Number Number::operator++()
{
    ++x;
    return Number(x);
}
Number Number::operator++(int y)
{
    y = x++;
    return Number(y);
}
```

```
Microsoft Visual Studio Debug C
Prefix increment: 11
Postfix increment: 20
N3: 21
```

```
    Number N1(10);
    Number N2;
    N2 = ++N1;
    cout << "Prefix increment: ";
    N2.display();
    Number N3(20);
    Number N4 = N3++;
    cout << "Postfix increment: ";
    N4.display();
    cout << "N3: ";
    N3.display();
    return 0;
}
```

Overloading of Binary Operators

- Binary operators:
- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulo division (%)

Program Example 4.2

- Write a C++ program to add two objects of a class Number using operator overloading.

```cpp
#include<iostream>
using namespace std;

class Number
{
    private:
        int x;
    public:
        Number();
        Number(int n);
        Number operator+(Number&);
        void display();
        ~Number();
};

// default constructor definition
Number::Number()
{
    x = 0;
}
```

```cpp
// parameterized constructor definition
Number::Number(int n)
{
    x = n;
}

// operator overloading definition
Number Number::operator+(Number& N)
{
    Number temp;
    temp.x = x + N.x;
    return temp;
}
```

```
// member function definition
void Number::display()
{
    cout << x << endl;
}
// destructor definition
Number::~Number()
{
    cout << "Objects successfully deleted:)" << endl;
}

// driver function
int main()
{
    Number N1(10), N2(20), N3;
    N3 = N1 + N2;
    cout << "Sum: ";
    N3.display();
    return 0;
}
```

```
// member function definition
void Number::display()
{
    cout << x << endl;
}
// destructor definition
Number::~Number()
{
    cout << "Objects successfully deleted:)" << endl;
}

// driver function
int main()
{
    Number N1(10), N2(20), N3;
    N3 = N1 + N2;
    cout << "Sum: ";
    N3.display();
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
Objects successfully deleted:)
Sum: 30
Objects successfully deleted:)
Objects successfully deleted:)
Objects successfully deleted:)
```

Using cin and cout for Objects

- Suppose we change the driver program shown in the previous slide

```
// driver function
int main()
{
    Number N1, N2, N3;
    cout << "Enter two numbers: ";
    cin >> N1 >> N2;
    N3 = N1 + N2;
    cout << "Sum: " << N3 << endl;
    return 0;
}
```

Error since we cannot use >> and << directly on objects

Overloading Input and Output Operators

- We can extend the operations of stream extraction operator >> and stream insertion operator << for user-defined data types.
- See Program Example 4.3

```
#include<iostream>
using namespace std;

class Number
{
    private:
        int x;
    public:
        Number();
        Number(int n);
        Number operator+(Number&);
        friend istream& operator>>(istream&, Number&);
        friend ostream& operator<<(ostream&, Number&);
        ~Number();
};
```

cin is an object of istream class
cout is an object of ostream class

```
// default constructor definition
Number::Number()
{
    x = 0;
}
// parameterized constructor definition
Number::Number(int n)
{
    x = n;
}
// operator overloading definition
Number Number::operator+(Number& N)
{
    Number temp;
    temp.x = x + N.x;
    return temp;
}
```

```
istream& operator>>(istream& input, Number& N)
{
    input >> N.x;
    return input;
}
ostream& operator<<(ostream& output, Number& N)
{
    output << N.x;
    return output;
}
// destructor definition
Number::~Number()
{
    cout << "Objects successfully deleted:)" << endl;
}
```

```
// driver function
int main()
{
    Number N1, N2, N3;
    cout << "Enter two numbers: ";
    cin >> N1 >> N2;
    N3 = N1 + N2;
    cout << "Sum: " << N3 << endl;
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
Enter two numbers: 11 21
Objects successfully deleted:)
Sum: 32
Objects successfully deleted:)
Objects successfully deleted:)
Objects successfully deleted:)
```

Important Points

- The operators that cannot be overloaded are:
  - Scope resolution operator (::)
  - Member selection operator or dot operator
  - Ternary operator (?:)
  - sizeof operator
- Operator overloading should not change the operation performed by an operator.
- Avoid overloading the assignment operator (=) and address operator (&)
- Number of operands cannot be changed for an operator.

Program Exercise 4.1

- Write a C++ program to perform subtraction, multiplication, division and modulo division between two objects of a class Number using operator overloading. Enter the object values from the keyboard and display the result of the object using overloading of >> and << operators.

Program Exercise 4.2

- Write a C++ program to add, subtract and multiply two matrices using the concepts of class and operator overloading.

Program Exercise 4.3

- Write a C++ program to increment and decrement time in seconds using the concepts of class and operator overloading. Enter the user-defined input using the overloaded stream insertion operator. Display the class object using overloaded stream extraction operator.

Operator Overloading

Module 4

OOP (IT 2005)

4th Semester ECSc