Object Oriented Programming
C++ Operator Overloading

Abeeda Akram

Operator Overloading

- Perform operations on class objects (variables of user defined ADTs) as performed on system defined datatypes.
- For Example:

```
cout << myobj;
myobj == otherobj;
myobj++;
myobj = otherobj + 3;
```

Operator Overloading Rules

- **You cannot**
  1. Change precedence of operators.

```
a=b+c*d; // order of execution *, +, =
a=b+c+d; // left hand rule b+c,  +d, =
```

  2. Change associativity of an operation.

```
a=b=c; //right to left
a+b-d; //left to right
```

  3. Use default parameters in operator functions.
  4. Change operands or parameters of an operation.

```
a+b; //binary operation take two operands
a++; //unary operation take one operand
```

  5. Create new operators.

Operator Overloading Rules Con…

1. The meanings of operators with built in types should remain same.

```
Point p1, p2(2, 3);
p1+p2; //means addition not subtraction
```

2. Can overload either for class objects of user defined class or for combination of objects user defined and built in datatypes.

```
Point p1, p2(2, 3);
p1+p2; //both are class objects of Point class
p1+2;  //class object and int
2+p1; //order matters for calling operator functions
cout << p1; //ostream and point class objects
```

Operator Overloading Rules Con…

- Operators that can be overloaded:

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

- Operators that cannot be overloaded:

.	.*	::	?:	sizeof

Operators classification

- **Unary Operators:**

  - (minus), !, ++ (pre and post), -- (pre and post), ~ (bitwise not) , & (address of)

- **Binary Operators:**

  1. Arithmetic **-, +, *, /, % , +=, -=, *=, /=, %=**
  2. Relational **==, !=, >=,<=, <, >**
  3. Assignment **=**
  4. Logical **&&, ||**
  5. Subscript **[]**
  6. Member access **->**
  7. Stream operators can be overload for file stream or command line stream

  << (stream insertion), >> (stream extraction)

  8. Bitwise: **&, |, >> (shift right), <<(shift left), ^ (XOR)**
  9. Memory management: **new, delete**

- **Operators must be overloaded explicitly**
  - Overloading **+** does not overload **+=**

Operator Function

- Operator function can be defined as
  1. Non-static member function of a class.
  2. Non-member function.
- Operator function header contains
  1. **return type**
  2. **operator reserve word**
  3. **operator symbol**
  4. **parameters list**

```
void operator ++ ();
//unary increment operator as member function
Point operator * (const Point & p);
// binary operator as member function
```

Member Functions

- Can be defined inside class as member or just add prototype and define outside as normal member functions.
- Operators that must be overloaded through member functions are:
  =, [], (), ->, &(address of operator)
- Unary operators: ***its good practice to define member function for unary operators.***
  - Member function, needs no argument.
- Binary operators:
  - Member function, needs one argument right operand can be class object or other datatype.
  - Left operand must be class object
- All operators can be overloaded through member functions in which left operand is class object for example:

```
Point p1, p2(2, 3);
p1 + p2; //both are class objects of Point class
p1++;
p1 = p2;
//left operand is class object member function will work
p1 + 3;
```

Unary Operator Minus (-)

- Member function **takes no argument** work on single operand must be the class object.
- Can be called in two ways.

```
Point p1(3, 4);
p1.operator-();
 Or
 -p1;
 Point p2 = -p1;
// cascaded call
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b;}
    Point operator-(); // prototype
};
//implementation
Point Point:: operator-() {
        Point p(*this);
        p.x = -p.x;
        p.y = -p.y;
        return p;
}
```

Unary Operators (++) pre increment

- Member function **takes no argument** work on single operand must be the class object.
- Can be called in two ways.

```
Point p1(3, 4);
p1.operator++();
Or
++p1;
Point p2 = ++p1;
// cascaded call
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b; }
    Point& operator++(); // prototype
};
//implementation
Point& Point:: operator++() {
        x++;
        y++;
        return *this;
}
```

Unary Operators post increment (++)

- Member function **takes no argument** work on single operand must be the class object.
- Can be called in two ways.

```
Point p1(3, 4);
p1.operator++(0);
// dummy zero to tell system post increment
Or
p1++;
Point p2 = p1++;
// cascaded call
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b;}
    Point& operator++(); // pre
    Point operator++(int);
    //post with dummy int lable
};
//implementation
Point Point:: operator++() {
    Point p(*this);
        x++;
        y++;
        return p;
}
```