Operator Overloading

CS(217) Object Oriented Programming

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 Operator)
- Binary Operators:
  - Arithmetic: -, +, *, /, % , +=, -=, *=, /=, %=
  - Relational: ==, !=, >=,<=, <, >
  - Assignment: =
  - Logical: &&, ||
  - Subscript: []
  - Member access: ->
  - Stream operators: can be overload for file stream or command line stream
    - << (stream insertion), >> (stream extraction)
  - Bitwise: &, |, >> (shift right), <<(shift left), ^ (XOR)
  - 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 other 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;
}
```

Binary Operator + Addition

- Both operands are class objects.
- Member function takes right operand of operation as one argument.
- Called on left operand must be class object.
- Can be called in two ways.

```
Point p1(3, 4), p2(3, 2);
p1.operator+(p2); // called on p1
Or
p1+p2;
// called on p1, p2 passed as argument
Point p3 = p1+p2;
// cascaded call
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b;}
    Point operator+ (const Point&);
};
//implementation
Point Point:: operator+(const Point& p){
        Point R;
        R.x = x + p.x;
        R.y = y + p.y;
        return R;
}
```

Binary Operator + Addition

- One operands left one is class object.
- Member function takes right operand of operation as one argument.
- Called on left operand must be class object.
- Can be called in two ways.

```
Point p1(3, 4);
p1.operator+(3); // called on p1
Or
p1+10; // called on p1, int 10 is passed as argument
int a = 10;
Point p3 = p1+a; // cascaded call
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b;}
    Point operator+ (const Point&);
    Point operator+ (const int);
    // with int
};
//implementation
Point Point:: operator+(const int n){
        Point R;
        R.x = x + n;
        R.y = y + n;
        return R;
}
```

Binary Operator == is equal to

- Both operands should be class objects.
- Member function takes right operand of operation as one argument.
- Called on left operand must be class object.
- Can be called in two ways.

```
Point p1(3, 4), p2(3, 2);
p1.operator==(p2); // called on p1
Or
cout << (p1==p2);
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b;}
    bool operator==(const Point&);
};
//implementation
bool Point:: operator==(const Point& p){
        if (x == p.x && y == p.y)
                return true;
        else
                return false;
}
```

Binary Operator != is not equal to

- Both operands should be class objects.
- Member function takes right operand of operation one argument.
- Called on left operand must be class object.
- Can be called in two ways.

```
Point p1(3, 4), p2(3, 2);
p1.operator!=(p2); // called on p1
Or
cout << (p1!=p2);
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b;}
    bool operator==(const Point&);
    bool operator!=(const Point&);
};
//Reuse == operator function
bool Point:: operator!=(const Point& p){
        return !((*this) == p) ;
}
```

Binary Operator = Assignment

- Member function is compulsory for assignment.
- Both operands should be class objects
- Member function takes right operand of operation as argument.
- Called on left operand that must be class object.
- Check state of both left and right object's data members carefully.
    1. If they are pointers address issues, due to different constructors, nullptr or valid memory address.
    2. Dynamic arrays size mismatch issues.
    3. Self assignment issue with pointer data members.
- Can be called in two ways.

```
Point p1(3, 4), p2(3, 2), p3;
p1.operator=(p2);
// called on p1
Or
p1=p2; // called on p1
p1=p2=p3; // cascaded call
```

Left Operand	Right Operand
nullptr	nullptr
nullptr	Address
Address	nullptr
Address (Single variable)	Address (Single variable)
Address (Array Size check)	Address (Array Size check)

Binary Operator = Assignment

```
class Point {
    int x, *y;
public:
    Point() { x=0; y=nullptr; }
    Point(int a, int b) {
        x=a;
        y=new int(b);
    }
    Point& operator=(const Point& p);
};
```

```
//implementation
Point& Point:: operator=(const Point& p){
  if (this != &p) {
        x= p.x;
        if(y==nullptr && p.y!=nullptr)
                y = new int(*(p.y));
        else if(y!=nullptr && p.y==nullptr){
                delete y;
                y = nullptr;
        }
        else if(y!=nullptr && p.y!=nullptr)
                *y = *(p.y);
      }
  return *this;
}
```