Object Oriented Programming

C++ Operator Overloading

Abeeda Akram

When Non-member functions needed?

- Member functions cannot be defined, if left operand of operation is not class object for example.

```
2 + p1; // left operand is int
cout << p1; // left operand is ostream class object
cin >> p1; //left operand if istream class object
```

- Therefore, Non-member functions can be used for such operations.

Non-Member Functions

- Non-member function cannot be defined inside the class
- They cannot access the private data members of a class
- Operators that cannot be overloaded through non-member functions are
  =, [], (), ->, &(address of operator)
- All other operators can be overloaded through non-member functions
- Unary operators
  - Non-member function, needs one argument
- Binary operators
  - Non-member function, needs two arguments
  - One argument must be class object or reference
- There is no **this** pointer in non-member functions

Non-Member Friend Functions

- `friend` function can access `private` and `protected` members of another class
  - `friend` functions are non-member functions of class
  - They are defined outside of class scope
  - Can only add prototype inside class definition for granting friendship
  - There is no **this** pointer in non-member friend functions
- Properties of friendship
  - **Friendship is granted, not taken**
  - **Not symmetric (if `B` a `friend` of `A`, `A` not necessarily a `friend` of `B`)**
  - **Not transitive (if `A` is `friend` of `B`, `B` is `friend` of `C`, `A` not necessarily a `friend` of `C`)**

Unary Operator Minus (-)

- Non-member function takes one argument that must be the class object.
- Can be called in two ways.

```
Point p1(3, 4);
operator-(p1);
// calls friend function
 Or
 -p1;
 Point p2 = -p1;
```

- Only add one function member or non-member friend to avoid conflict.

```
class Point {
   int x, y;
public:
   Point(int a=0, int b=0) { x=a; y=b;}
   friend Point operator-(const
   Point&); // prototype
};
Point operator-(const Point& p) {
     Point r(p);
     r.x = -p.x;
     r.y = -p.y;
     Return r;
}
```

Binary Operator Addition (+)

- **Both operands are class objects.**
- Non-Member function takes two arguments.
- Can be called in two ways.

```
    Point p1(3, 4), p2(3, 2);
    operator+(p1,p2);
     Or
     p1+p2;
     // both p1 and p2 are passed
   as arguments
     Point p3 = p1+p2;
   // cascaded call
```

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

Binary Operator Addition (+)

- One operand left one is class object.
- Non-member function takes two arguments
- Can be called in two ways.

```
    Point p1(3, 4);
    operator+(p1,3);
     Or
     p1+10;
 // both p1 and int 10 are passed
 as arguments
     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; }
friend Point operator+ (const Point& p, const Point&
q);
friend Point operator+ (const Point& p, const int &
n); // with int
};


Point operator+(const Point& p, const int& n){
    Point R;
    R.x = p.x + n;
    R.y = p.y + n;
    return R;
}
```

Binary Operator Addition (+)

- **One operand right one is class object**
- Must define non-member function.
- Non-member function takes two arguments
- Can be called in two ways.

```
Point p1(3, 4);
operator+(3,p1);
 Or
 10+p1;
// both p1 and int 10 are passed as
arguments
 int a = 10;
 Point p3 = a+p1;
// cascaded call
```

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

Binary Operator Addition (+)

- One operand **right** one is class object
- Must define non-member function.
- Non-member function takes two arguments
- Can be called in two ways.

```
    Point p1(3, 4);
    operator+(3,p1);
     Or
     10+p1;
   // both p1 and int 10 are passed as
  arguments
     int a = 10;
     Point p3 = a+p1;
  // cascaded call
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b; }
friend Point operator+ (const Point& p, const Point&
q);
friend Point operator+ (const Point& p, const int &
n);
// with int right operand
friend Point operator+ (const int& n, const Point& p);
// with int left operand
};
Point operator+(const int& n, const Point& p){
    return p + n;
    // Reuse code of right operand function
}
```

Binary Operator Stream insertion (<<)

- One operand **left** one is stream object and **right** one is class object
- Must define non-member function, which takes two arguments
  - First non constant reference of ostream object
  - Second const reference of class object
- Called as follows.

```
Point p1(3, 4), p2(1, 2);
cout << p1; // left operand
is ostream class object

cout << p1 << p2;
// cascading will not work
as return type is void
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b; }
friend void operator<< (ostream& , const
Point&);
};
// insert or write data of object member wise
in stream
void operator << (ostream & out, const Point&
p){
    out << "X:" << p.x << endl;
    out << "Y:" << p.y << endl;
}
```

Binary Operator Stream insertion (<<)

- One operand **left** one is stream object and **right** one is class object
- Must define non-member function, which takes two arguments
    - First non constant reference of ostream object
    - Second const reference of class object
- For cascading return ostream object by reference from function.

```
Point p1(3, 4), p2(1, 2);
cout << p1 << p2;
cout << ++p1 << p2+3;
// cascading will work
now
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b; }
friend ostream& operator<< (ostream& , const
Point&);
};
// insert or write data of object member wise in
stream
ostream& operator << (ostream & out, const Point&
p){
    out << "X:" << p.x << endl;
    out << "Y:" << p.y << endl;
        return out;
}
```

Binary Operator Stream extraction (>>)

- One operand **left** one is stream object and **right** one is class object
- Must define non-member function, which takes two arguments
  - First non constant reference of istream object
  - Second non constant reference of class object
- Called as follows.

```
Point p1, p2;
cin >> p1; // left operand is
istream class object

cin >> p1 >> p2;
// cascading will not work as
return type is void
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b; }
friend void operator>> (istream
& ,Point&);
};
// input data from stream and write
member wise in object
void operator >> (istream & in, Point&
p){
    in >> p.x;
    in >> p.y;
}
```

Binary Operator Stream extraction (>>)

- One operand **left** one is stream object and **right** one is class object
- Must define non-member function, which takes two arguments
  - First non constant reference of istream object
  - Second non constant reference of class object
- For cascading return istream object by reference from function.

```
Point p1, p2;
cin >> p1 >> p2;
// cascading will work now
```

```
class Point {
    int x, y;
public:
    Point(int a=0, int b=0) { x=a; y=b; }
friend istream& operator>> (istream & ,
Point&);
};
// input data from stream and write member
wise in object
istream& operator >> (istream & in, Point& p)
{
    in >> p.x;
    in >> p.y;
    return in;
}
```

Complete the implementation of myarray

```
class myArray{
    int size; // Array size  int *ptr; // Pointer for dynamic 1-D Array
public:
    myArray() { size=0; ptr=nullptr; }
    myArray(int size);
    myArray(int * arr, int size);
    myArray(const myArray&); // copy constructor
    ~myArray();
    myArray& operator=(const myArray &); // Assignment
    int& operator[](const int i);
    const int & operator[](const int i) const;
    myArray& operator++(); // increment data of all elements
    myArray operator++(int); // increment data of all elements
    bool operator==(const myArray &); //compare size and data of all elements
    bool operator!=(const myArray &);
    friend istream& operator>> (istream& , myArray&); //take size and data from console
    friend ostream& operator<< (ostream& , const myArray &); // Print data of array on
    console
    myArray operator+(const myArray &); // Return array containing data of both arrays merged
    friend myArray operator+(const int , const myArray &); // add int value to all elements
    of array
```