CS217 – Object Oriented Programming (OOP)

Week – 10

April 12-16, 2021

Instructor: **Basit Ali**

Friend Function

- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class.

- Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

Friend Function

- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows

Implementation

```
class A
{
        int x;

        public:
            display()
            {
                cout<<x<<endl;
            }

            friend void set();

            friend void set2(A a);
};
```

```
void set()
{
    A a;
    a.x = 10;
    a.display();
}

void set2(A a)
{
    a.x = 15;
    a.display();
}
```

Operator Overloading

- In C++, we can make operators to work for user defined classes.

- This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

Operator Overloading (Continue..)

- For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +

- Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.

Operator Overloading (Continue..)

- Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined

```
ClassName operator - (ClassName c2)
{
    ... .. ...
    return result;
}

int main()
{
    ClassName c1, c2, result;
    ... .. ....
    result = c1-c2;
    ... .. ...
}
```

```
A operator +(A a)
{
    A a3;
    a3.x = x + a.x;
    a3.y = y + a.y;
    return a3;
}
```

Operator Overloading (Continue..)

```
2  #include<iostream>
3  using namespace std;
4
5  class Complex {
6
7  private:
8      int real, imag;
9
10 public:
11     Complex(int r = 0, int i =0)  {real = r;   imag = i;}
12
13     // This is automatically called when '+' is used with
14     // between two Complex objects
15     Complex operator + (Complex const &obj) {
16         Complex res;
17         res.real = real + obj.real;
18         res.imag = imag + obj.imag;
19         return res;
20     }
21 };
```

```
23 int main()
24 {
25     Complex c1(10, 5), c2(2, 4);
26     // An example call to "operator+"
27     Complex c3 = c1 + c2;
28 }
```

Operator Overloading (Continue..)

```
operator ++()
{
    real++;
}
```

++object;

```
operator ++(int)
{
    real++;
}
```

object++;

Global Operator Function

```
1  #include<iostream>
2  using namespace std;
3
4  class Complex {
5  private:
6      int real, imag;
7  public:
8      Complex(int r = 0, int i =0)  {real = r;   imag = i;}
9      void print() { cout << real << " + i" << imag << endl; }
10
11 // The global operator function is made friend of this class so
12 // that it can access private members
13 friend Complex operator + (Complex const &, Complex const &);
```

Global Operator Function

```
19  Complex operator + (Complex const &c1, Complex const &c2)
20  {
21      return Complex(c1.real + c2.real, c1.imag + c2.imag);
22  }
23
24  operator ++(Complex x)
25  {
26  }
27
28  int main()
29  {
30      Complex c1(10, 5), c2(2, 4);
31      Complex c3 = c1 + c2; // An example call to "operator+"
32      c3.print();
33      return 0;
34  }
```

Overload-able Operators

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

Non-overload-able Operators

::	.*	.	?: