Object-oriented Programming

Operator Overloading

CREATED BY: SYED ZAIN UL HASSAN

Operator Overloading

- Defining a new behavior for common operators of a language
- C++ enables you to overload most operators to be sensitive to the context in which they're used
- Using operator overloading makes a program clearer than accomplishing the same operations with function calls

Operator Overloading

- An operator is overloaded by writing a non-static member function definition or global function definition

- When operators are overloaded as member functions, they must be non-static

- To use an operator on class objects (as operands), that operator **“must” be overloaded**

Operator Overloading

- Operator overloading cannot change the arity (no. of operands) of an operator
- Operator overloading works when ***at least*** one operand of that operator is an object
- We cannot create new operators using operator overloading

Example

```
class Vector
{
	int x, y;
	public:
	Vector( int x = 0, int y = 0)
	{
		this->x = x; this->y = y;
	}
	void printXY( )
	{
		cout << "x: " << x << endl;
		cout << "y: " << y << endl;
	}
	Vector operator+(const Vector &ob)
	{
		Vector temp;
		temp.x = x + ob.x;
		temp.y = y + ob.y;
		return temp;
	}
};
```

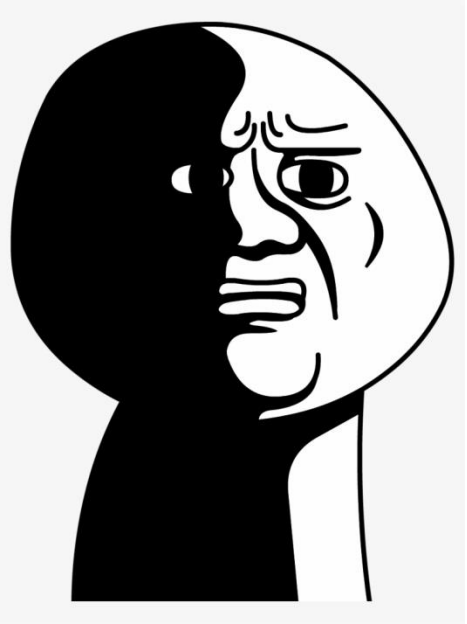
```
int main()
{
		Vector v1(10, 15);
		Vector v2(8, 6);
		Vector v3 = v1 + v2;

		v3.printXY();
}

// prints x: 18 & y: 21
```

Example

- We can also define overloads for other operators in our example **Vector** class



For += operator

```
Vector& operator += (const Vector &ob)
{
        x += ob.x;
        y += ob.y;
        return *this;
}
```

For Unary - operator (Negation)

```
Vector operator – ( ) const
{
        Vector temp;
        temp.x = -x;
        temp.y = -y;
        return temp;
}
```

For Prefix ++ operator

```
void operator ++ ( )
{
        ++x;
        ++y;
}
```

(Works the same way for prefix decrement operator)

For Postfix ++ Operator

```
Vector operator ++ ( int )
{
        Vector temp;
        temp.x = x++;
        temp.y = y++;
        return temp;
}
```

Restrictions on Operator Overloading

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

Operators that can be overloaded.

Operators that cannot be overloaded			
.	.*	::	?:

Operators that cannot be overloaded.

Friend Function

- A global friend function is one which can access private members of a class.
- Though a global friend function can access or modify the private instance members of a class directly, it still however needs an object of the class to do so.
- Access modifiers have not effect on friend function prototype inside class

Example

```
class A
{
        int  var;                       // private data member
        friend void func( );
};

void func( )
{
        A ob;
        ob.var = 50;                    // can directly access var
}
```

Friend Class

- Just like global friend functions, a friend class is one which can directly access private members of a class
- The private members of a class can be referenced directly in all member functions of its friend class
- However, an object would still be required to access the private members

Example

```
class A
{
        int  var;                       // private data member
        friend class B;
};

class B
{
        A ob;
        public:
        B( ) {
                ob.var = 10;            // can directly access var
        }
};
```

Next Topic

- Abstract class