ITEC243 – Lecture Session (11-May-2020)

Operator Overloading

+ (addition), - (subtraction), ++ (increment), -- (decrement), etc…

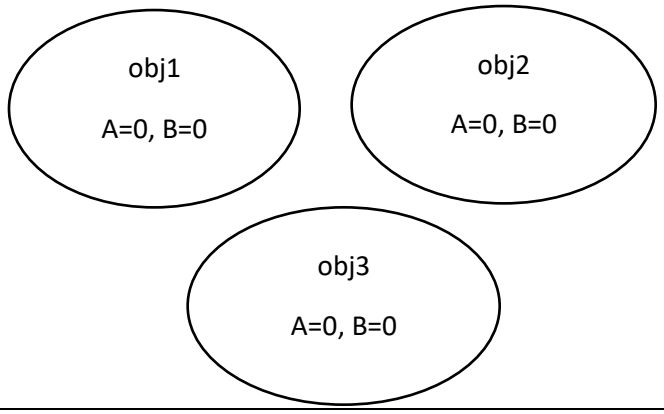
C++ allows us to redefine (change the behavior) how standard operators (that are listed above) work when used with class objects.

Example:

```
int a=5, b=10, c;
a++; //a is incremented by 1
b--; //b is decremented by 1
c=a+b; //holds the summation of a and b values
```

Assume we have a class called "test".

```
test obj1, obj2,obj3;
obj1++; //post-fix increment
--obj2; //pre-fix decrement
obj3=obj1+obj2;
```



Syntax:
<return type> operator<sign> (list of arguments/objects)

Question1: Why do we need the operator overloading while we can do the same task with simple method???
Answer: To use operator overloading feature is easier compare to the functions for the end-user point of view.

Question2: When do we need to use the operator overloading?
Answer: When we want to do the operation with help of operator and the operation is to be performed on **OBJECTS** not on built-in variables.

List of operators that can be overloaded:
+, - , /, *, new, delete, = ….

Operators that can be overloaded both unary and binary: +, -, *, &
The operators that cannot be overloaded: . (dot operator), :: (scope resolution operator), ….

Operator Overloading can be done with the help of;

1. Simple function
2. Friend function

There are two types of operator overloading:

1. Unary Operator (obj++, obj--,…)
2. Binary Operator (if two objects are involved into the operation, obj3=obj1+obj2)

Examples:

```
class example{
private:
	int a, b;
public:
	example()
	{
		this->a = 0;
		this->b = 0;
	}
	void setData(int a, int b)
	{
		this->a = a;
		this->b = b;
	}
	void display()
	{
		cout << "Value 1=" << this->a
			<< "\tValue 2=" << this->b << endl;
	}
	//prefix increment (++obj)
	void operator++()
	{
		this->a += 3;
		this->b += 3;
	}
	//postfix increment (obj++)
	//postfix unary operator always have a default parameter - int datatype (by default the value = 0)
	void operator++(int notused)
	{
		this->a += 5;
		this->b += 5;
```

```
	}
	//prefix decrement
	void operator--()
	{
		-- this->a;
		-- this->b;
	}
	//postfix decrement
	void operator--(int notused)
	{
		this->a -= 4;
		this->b -= 4;
	}
	//binary operator (+) usign simple function (you may use a friend function to do the
operation)
	example operator+(example obj)
	{
		example temp;
		temp.a = this->a + obj.a;
		temp.b = this->b + obj.b;
		return temp;
	}

};

#include<iostream>
using namespace std;
#include"example.h"
void main()
{
	example  obj1;
	obj1.setData(2, 2);
	obj1.display();

	//prefix increment
	cout << "Prefix increment" << endl;
	++obj1;
	obj1.display();

	//postfix increment
	cout << "Postfix increment" << endl;
	obj1++;
	obj1.display();
```

```
        //prefix decrement
        cout << "Prefix decrement" << endl;
        --obj1;
        obj1.display();

        //postfix decrement
        cout << "Postfix decrement" << endl;
        obj1--;
        obj1.display();

        example obj2, obj3;
        obj2.setData(3, 3);
        obj3 = obj1 + obj2;

        cout << endl << endl;
        obj1.display();
        obj2.display();

        cout << "Binary operator (+)"<<endl;
        obj3.display();

        system("pause");
}
```