BTEP243 – Ders 8

Operatör Ekkullanımı

Fonksiyon ekkullanımına bağlı olan bir C++ özelliği de operatör ekkullanımıdır (operator overloading). C++ operatörlerinin çoğu kendi asıl işlevlerinden başka, tanımlandıkları sınıflara göre ek görevler yürütebilirler. Örneğin, **+** operatörü bir bağlantılı listeye (link list) yeni bir eleman ilave etmek maksadıyla kullanılabilir. Bu operatör iki karakter sözcüğünü birleştirmeye yarayabilir. Bir başka sınıf **+** operatörünü bambaşka bir amaçla kullabilir. Bir operatör ekkullanım yüklenince asıl işlevini kaybetmez; **+** operatörü yine sayıları toplama işlevini görür.

Bir operatöre ek görev yüklemek için, kullanıldığı sınıflara göre ne işlem yapacağı özel bir fonksiyonla belirtilir. Buna operatör fonksiyonu denir. Kullanım şekli:

```
dönüş_tipi operator### (parametre listesi)
{
        fonksiyon içeriği
}
```

Ekkullanımı mümkün olan operatörler:

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

Ekkullanımı tekli(unary) ve ikili(binary) olabilen operatörler:

+	-	*	&

Ekkullanımı mümkün olmayan operatörler:

.	.*	::	?:

Örnek 1:

```
class nokta{
        int x, y, z;
public:
        nokta()
        {
                x = y = z = 0;
        }
        nokta(int a, int b, int c)
        {
                x = a;
                y = b;
                z = c;
        }
        nokta operator+(nokta n)
        {
                nokta temp;
                temp.x = x + n.x;
                temp.y = y + n.y;
                temp.z = z + n.z;
                return temp;
        }
        nokta operator=(nokta n)
        {
                x = n.x;
                y = n.y;
                z = n.z;
                return *this;
        }
        void goster()
        {
                cout << x << ", "
                        << y << ", "
                        << z << endl;
        }
        void set(int nx, int ny, int nz)
        {
                x = nx;
                y = ny;
                z = nz;
        }
};
```

```
#include<iostream>
using namespace std;
#include "nokta.h"
void main()
{
        nokta a, b, c;
        a.set(1, 2, 3);
        b.set(5, 5, 5);
        a.goster();
        b.goster();
        c = a + b;
        c.goster();
        c = a + b + c;
        c.goster();
        c.set(1, 2, 3);
        c = b = c;
        c.goster();
        b.goster();
        system("pause");
}
```

Ekran çıktısı:

```
1, 2, 3
5, 5, 5
6, 7, 8
12, 14, 16
1, 2, 3
1, 2, 3
Press any key to continue . . .
```

Operatör Fonksiyonlarının Kısıtlamaları

Operatör fonksiyonları C++ programlarına esneklik getirmelerine karşın uyulması gerekli bazı kısıtlamalar vardır:

- **Yeni opertörler yaratılamaz.** Mevcut C++ operatörleri tanımlamak mümkün değildir. Örneğin, $ işareti bir operatör değildir. Bu nedenle de operator$ ( ) fonksiyonu tanımlanamaz.
- **Operatörlerin tip ve öncelikleri değiştirilemez.** Tekli operatörler ikili olarak kullanılamaz, ayni şekilde ikili bir operatör de tekli olarak kullanılamaz. Örneğin = operatörü ikilidir. Bu operatör tekli olarak tanımlanamaz.
- **Operatörlerin temel tipler üzerindeki işlevleri değiştirilemez.** Operatörler temel tipler için belli işlevler yapmak üzere tanımlıdır. Bu işlevleri fonksiyonlarla değiştirmek mümkün değildir. Örneğin, + operatörü iki sözcüğü birleştirmek için aşağıdaki şekilde kullanılamaz:

  char * operator+ (char *a, char *b);

  Bu nedenle bir operatör fonksiyonunun en az bir parametresi bir sınıf nesnesi olmalı ya da operatör fonksiyonunun kendisi bir sınıfın üyesi olmalıdır.
- **Operatörler başka operatörler oluşturmak için birleştirilemez.** İki operatörü yan yana getirerek iki işlemi birden yapacak yeni bir operatör yaratmak mümkün değildir. Örneğin operator+[]( ) isimli bir fonksiyon yaratılamaz.

Tekli (Unary) Operatörler

Tekli operatörler sadece bir parametre alırlar; ++ ve - - birer tekli operatördür.

Örnek2:

```
class abc{
private:
	int n1, n2;
public:
	abc()
	{
		n1=0; n2=0;
	}
	void set(int a,int b)
	{
		n1=a;
		n2=b;
	}
	void goster()
	{
		cout<<"Sayi1:"<<n1<<endl;
		cout<<"Sayi2:"<<n2<<endl;
	}

	//tekli ön operatör (prefix unary operator) (++obj1)
	void operator++()
	{
		n1=n1+3;
		n2=n2+3;
	}
	//tekli son operatör (postfix unary operator) (obj++)
	//son operator mutlaka int parametre içermelidir. Bu parametre asla kullanılmaz ve //varsayılan değeri her zaman sıfırdır.
	void operator++(int temp)
	{
		n1=n1+7;
		n2=n2+7;
	}
```

```
};
#include <iostream>
using namespace std;
#include "abc.h"
void main()
{
	abc obj1;
	abc.set(5,5);
	abc.goster();

	//ön arttırma
	cout<<"Ön arttırma"<<endl;
	++obj1;
	abc.goster();

	//son arttırma
	cout<<"Son Arttırma"<<endl;
	obj1++;
	obj1.goster();
	system("pause");
}
```

Örnek 3:

```
class test{
private:
	int n1,n2;
public:
	test(int a,int b)
	{
		n1=a;  n2=b;
	}
	void goster()
	{
		cout<<"Sayi 1:"<<n1<<endl;
		cout<<"Sayi 2:"<<n2<<endl;
	}
	void operator++(int n)
	{
		n1++;
		n2++;
	}
	void operator++()
	{
		n1=n1+10;
		n2=n2+20;
	}
};
#include <iostream>
using namespace std;
#include "test.h"
void main()
{
```

```
	test obj1(2,7);
	obj1.goster();
	++obj1;
	obj1.goster();

	obj1++;
	obj1.goster();
	system("pause");
}
```

Örnek 4:

```
class testsayac {
private:
	int sayac1,sayac2,sayac3;
public:
testsayac()
{
	sayac1=100; sayac2=0; sayac3=0;
}

void operator++()
{
	sayac1-=2 ; sayac2+=5; sayac3+=10;
}

void show()
{
	cout<<sayac1<<"  "<<sayac2<<"   "<<sayac3<<endl;
}
};

#include <iostream>
using namespace std;
#include "test.h"
void main()
{
	test t;
	t.show();
	++t;
	t.show();
	for ( int k=0;k<8;k++)
	{
		++t ;
		 t.show();
	}
}
/* OUTPUT :
100  0   0
98  5   10
96  10   20
94  15   30
92  20   40
```

```
90  25   50
88  30   60
86  35   70
84  40   80
82  45   90
*/
```

İkili (Binary) Operatörler

İki parametre ile kullanılan standard C operatörleri ekkullanımla yüklenebilirler. Bu tür operatör fonksiyonları **this** işaretçisi ile kullanılabilir.

Örnek 5:

```
class test
{
        int n1,n2;
public:
        test(int a, int b)
        {
                n1=a;
                n2=b;
        }
        void goster()
        {
                cout<<"("<<n1<<","<<n2<<")";
        }

//ikili (binary) operator
        test operator+(test obj)
        {
                test temp(0,0);
                temp.n1=n1+obj.n1;
                temp.n2=n2+obj.n2;
                return temp;
        }
};
#include<iostream>
using namespace std;
#include "test.h"
void main()
{
        test obj1(2,1), obj2(4,3),obj3(0,0);
        obj1.goster();
        cout<<"+";
        obj2.goster();
        cout<<"=";
        obj3=obj1+obj2;
        obj3.goster();
        system("pause");
}
```

Örnek 6:

Arkadaş Fonksiyon kullanımı ile İkili Operatör tanımı

```
class test
{
	int n1,n2;
public:
	test(int a, int b)
	{
		n1=a;
		n2=b;
	}
	void goster()
	{
		cout<<"("<<n1<<","<<n2<<")";
	}
	friend test operator+(test,test);
};
test operator+(test obj1,test obj2)
{
	test temp(0,0);
	temp.n1=obj1.n1+obj2.n1;
	temp.n2=obj1.n2+obj2.n2;
	return temp;
}
#include<iostream>
using namespace std;
#include "test.h"
void main()
{
	test obj1(3,2), obj2(2,4),obj3(0,0);
	obj1.goster();
	cout<<"+";
	obj2.goster();
	cout<<"=";
	obj3=obj1+obj2;
	obj3.goster();
	system("pause");
}
```

Örnek 7:

```
class testsayac{
	int sayac;
public:
	testsayac()
	{
		sayac=0;
	}
	testsayac(int s)
	{
		sayac=s;
	}
```

```
	void operator++()
	{
		++sayac;
	}
	void goster()
	{
		cout<<sayac<<endl;
	}
};
#include<iostream>
using namespace std;
#include “testsayac.h”
void main()
{
	testsayac t=10;
	t.goster();
	++t;
	t.goster();
	system("pause");
}
```

Örnek 8:

```
class testsayac{
	int sayac;
public:
	testsayac()
	{
		sayac=0;
	}
	testsayac(int s)
	{
		sayac=s;
	}
	testsayac operator++()
	{
		testsayac temp;
		++sayac;
		temp.sayac=sayac;
		return temp;
	veya
		testsayac temp;
		++sayac;
		return *this;
	}
	void goster()
	{
		cout<<sayac<<endl;
	}
};
```

```
#include<iostream>
using namespace std;
#include “testsayac.h”
void main()
{
	testsayac a=10, b;
	a.goster();
	b=++a;
	a.goster();
	++b;
	b.goster();
	system("pause");
}
```

Örnek 9:

Arkadaş fonksiyonlar ve tekli operatör kullanımı

```
class testsayac{
	int sayac;
public:
	testsayac()
	{
		sayac=0;
	}
	testsayac(int s)
	{
		sayac=s;
	}
	friend testsayac operator++(testsayac&);

	void goster()
	{
		cout<<sayac<<endl;
	}
};

testsayac operator++(testsayac& obj)
	{
		testsayac temp;
		temp.sayac=obj.sayac++;
		return temp;
```

veya

```
		++obj.sayac;
		return obj;
	}
```

```
#include<iostream>
using namespace std;
#include “testsayac.h”
void main()
{
	testsayac a=10, b;
	a.goster();
	b=++a;
	a.goster();
	++b;
	b.goster();
	system("pause");
}
```

Örnek 10:

```
class testsayac{
	int sayac;
public:
	testsayac()
	{
		sayac=0;
	}
	testsayac(int s)
	{
		sayac=s;
	}

	friend void operator++(testsayac&);

	void goster()
	{
		cout<<sayac<<endl;
	}
};
void operator++(testsayac& obj)
	{
		++obj.sayac;
	}
#include<iostream>
using namespace std;
#include “testsayac.h”
void main()
{
	testsayac a=10;
	a.goster();
	++a;
	a.goster();
	system("pause");
}
```

Örnek 11:

```
class btep{
private:
	int a,b;
public:
	btep(int a=0,int b=0)
	{
		this->a=a;
		this->b=b;
	}
	void goster()
	{
		cout<<"("<<a<<","<<b<<")";
	}

	void operator++(int t)
	{
		this->a+=2;
		this->b+=2;
	}
	void operator++()
	{
		++this->a;
		++this->b;
	}

	void operator--(int t)
	{
		this->a-=3;
		this->b-=3;
	}

	void operator--()
	{
		this->a-=2;
		this->b-=2;
	}
	friend btep operator+(exam,exam);
	friend btep operator-(exam,exam);
};
btep operator+(btep E1,btep E2)
{
	btep E3;
	E3.a=E1.a+E2.a;
	E3.b=E1.b+E2.b;
	return E3;
}

btep operator-(btep E1,btep E2)
{
	btep E3;
	E3.a=E1.a-E2.a;
```

```
	E3.b=E1.b-E2.b;
	return E3;
}

#include<iostream>
using namespace std;
#include"btep.h"
void main()
{
	btep eobj1(10,15),btep(5,5),btep;
	++eobj1;
	++eobj2;

	eobj3=eobj1-eobj2;

	eobj1.goster();
	cout<<"-";
	eobj2.goster();
	cout<<"=";
	eobj3.goster();
	cout<<endl;

	eobj1++;
	eobj2++;
	exam eobj4;

	eobj4=eobj1+eobj2;
	eobj1.goster();
	cout<<"+";
	eobj2.goster();
	cout<<"=";
	eobj4.goster();
	cout<<endl;

	system("pause");
}
```