Operatörlere İşlev Yükleme

Binnur Kurt
kurt@ce.itu.edu.tr

Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi

3

Operatörler

• C tip-duyarlı ve -odaklı bir dildir. Her operatör belirli tiplerde operand alır.
• C'de temel tiplerden ve türetilmiş tiplerden yeni tipler türetilebilir.
• Türetilen tipler için mevcut operatörleri kullanabilir miyiz ?
• C'de operatörleri aldıkları operand sayılarına göre
- Tekli Operatörler
  -, +, !, ++, --, new, delete, sizeof()
- İkili Operatörler
  +, -, *, /, %, ^, &, |, ~, ==, <, >, +=, -=, *=, /=, %=, ^=, &=, |=, <<, >>, <<=,>>=, <=, >=, &&, ||, ->*, ->, [], (), :: , .
- Üçlü operatör ?:

şeklinde sınıflandırabiliriz.

```
/* A class to define complex numbers */
class ComplexNumber {
    double realPart,imaginaryPart;
  public:
    // Other member functions
    ComplexNumber operator+(ComplexNumber&); // header of operator+
function
};
/* The Body of the function for operator + */
ComplexNumber ComplexNumber::operator+(ComplexNumber& z)
{
   ComplexNumber result;
   result.realPart      = realPart + z.realPart;
   result.imaginaryPart = imaginaryPart + z.imaginaryPart ;
   return result;
}
void main()
{
   ComplexNumber z1,z2,z3;
    // Other operations
    z3=z1+z2;                                   z3 = z1.operator+(z2);
}
```


İşlev Yükleme

Yerel olarak geçici nesne yaratmaktan kaçının.

Aşağıdaki fonksiyon daha az bellek kullanır ve daha hızlıdır.

Neden?

ComplexNumber operator+(ComplexNumber&);

```
/* The Body of the function for operator + */
ComplexNumber ComplexNumber::operator+(ComplexNumber& z)
{
   double myRealPart,myImaginaryPart ;
   myRealPart      = this->realPart + z.realPart;
   myImaginaryPart  = this->imaginaryPart + z.imaginaryPart;
   return ComplexNumber(myRealPart,myImaginaryPart);
}
```

Atama İşlevi

Genellikle bir sınıfın birkaç üyesini kopyalamak istediğimizde atama operatörüne işlev yükleriz :

```
void ComplexNumber::operator=(ComplexNumber& z)
{
      this->realPart      = z.realPart;
      this->imaginaryPart = z.imaginaryPart;
}
```


Atama İşlevi

• Atama operatörü, diğer operatörlerden bir özelliği ile ayrılır:

Diğer operatörlerin varsayılan bir davranışı yoktur. Bunun sonucu olarak, ilgili operatör, tanımlanan sınıf nesneleri üzerinde tanımlı olması için mutlaka işlev yüklemek gerekir.

• Atama operatörünün varsayılan davranışı sınıf üyelerini bire bir kopyalamaktır.

• Bu durumda, atama operatörüne bu davranıştan farklı bir işlem yapılacaksa işlev yüklemek uygun olur.

• İzleyen örnekte bir dinamik bellek kullanımı var.

```
class IntegerArray
{
   int size;
   int *elements;
 public:
  // Assignment Operator
   void operator=(IntegerArray &);
   // Other methods
};

void IntegerArray::operator=(IntegerArray &s)
{
   size = s.size; delete[] elements ;
   elements = new int[size];
   memcpy(elements, s.elements,size*sizeof(int));
 }
```


Atama İşlevi

Bir önceki örnekteki atama operatörü ile

s1=s2=s3 ;

şeklinde iç içe atamalar yapılamaz. Çünkü işlev yüklediğimiz atama operatörü aynı sınıftan bir nesne döndürmemektedir.

```
IntegerArray IntegerArray::operator=(IntegerArray &s)
{
   this->size = s.size;
   elements = new int[size];
   memcpy(elements, s.elements,size*sizeof(int));
   return *this ;
 }
```

Tekil İşlev Yükleme

- Tekli Operatörler : Tek bir operand alırlar

  -, +, !, ++, --

Bu nedenle herhangi bir değer döndürmezler.

```
void ComplexNumber::operator++()
{
    realPart=realPart+1.0;
}
void main()
{
  ComplexNumber z(1.2, 0.5);
  ++z;
  cout << z ;
}
```



```
class ComplexNumber {
   double realPart,imaginaryPart;
 public:
   // Other member functions
   friend istream& operator  >>(istream& , ComplexNumber&);
   friend ostream& operator <<(ostream&, ComplexNumber&);
};

istream& operator >>(istream& stream, ComplexNumber& z)
{
  cout << "Enter real part:";
  stream >> z.realPart;
  cout << "Enter imaginer part:";
  stream >> z.imaginaryPart;
  return stream;
};
```

```
ostream& operator <<(ostream& stream,ComplexNumber& z)
{
  stream << "("
         << z.realPart
          <<","
         << z.imaginaryPart
         << ") \n";
  return stream;
};
```


Önceden Arttırma Operatörü

```
ComplexNumber operator++();
```

```
ComplexNumber ComplexNumber::operator++()
{
  realPart ++;
  return *this;
}
```

Sonradan Arttırma Operatörü

```
ComplexNumber operator++ ( int );
```

gerçek bir tip bildirimi değildir.

```
ComplexNumber ComplexNumber::operator++(int)
{
  double u,v  ;
  u = realPart ;
  v = imaginaryPart ;
  realPart++ ;
  return ComplexNumber(u,v);
}
```


İpucu - 1

```
void ComplexNumber::operator+=(ComplexNumber& z)
{
  realPart += z.realPart;
  imaginaryPart += z.imaginaryPart ;
}
```

```
ComplexNumber z1,z2 ;
z1 =  z2 + z1;
```

```
ComplexNumber z1,z2 ;
z1 +=  z2 ;
```

Daha Hızlı Çalışır

İpucu - 2

```
ComplexNumber z1,z2,z3 ;
 z3 =  z1 + z2;
```

```
ComplexNumber z1,z2,z3 ;
 z3.add(z1,z2) ;
```

Daha Hızlı Çalışır

```
void ComplexNumber::add(ComplexNumber& z1, ComplexNumber& z2)
{
  realPart = z1.realPart+z2.realPart;
  imaginaryPart = z1.imaginaryPart + z2.imaginaryPart;
}
```