Operatör Yükleme (Operator Overloading)

Dr. Metin Özkan

Giriş

- Programlama dillerinde, bir çok operatör kullanılmaktadır.
  - *Matematiksel operatörler (+, -, /, *, v.b.)*
  - *Mantıksal operatörler (!, ||, &&, v.b.)*
  - *İlişkisel operatörler (==, <, > , v.b.)*
- C++ içindeki operatörler, farklı fonksiyonların çağrılmasını sağlar.
- Bu fonksiyonlar, standart C++ değişken tipleri için tanımlıdır. Ancak, programcı tarafından beyan edilmiş değişkenler (nesneler) için, operatör yükleme (operator overloading) ile yeni anlamlar yüklenebilir.
- Operatör yükleme C++ programlamaya yeni bir yetenek kazandırmaz. Ancak, programların kolay yazılmasını, okunmasını ve anlaşılmasını sağlar.

Giriş

- Örneğin, + operatörü, a + b şeklinde (a ve b int tipinde olsun) yazılan bir kodda bir fonksiyonun çağrılmasını sağlar. Bu fonksiyon, operatörün sağ ve solunda bulunan değişkenleri parametre olarak alıp, bunların toplamını döndürür.

- Eğer, a ve b değişkenleri int tipinde değil de, birer Point tipinde nesne ise hangi fonksiyon çağrılacaktır?

- Bu durumda, operatör yükleme fonksiyonu yazılarak, a ve b değişkenlerinin birer Point tipinde nesne olması durumunda toplama işleminin nasıl yapılacağı kodlanmalıdır.

Giriş

- Operatör fonksiyonunun yazımı için sözdizimi (syntax) şu şekildedir:

```
veriTipi operator operatörSembolü (formal parametre listesi)
returnType operator operatorSymbol (formal parameter list)
```

- Bir çok operatör için yükleme mümkündür..

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

- Bazı operatörlere yükleme yapılamaz.

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

Bazı Kısıtlar

- Operatörün öncelik durumu değiştirilemez.
- Operatörlerin çağrımsal sırası (örneğin, operatör sağdan sola ya da sağdan sola uygulanabilir) değiştirilemez.
- Operatör fonksiyonlarında, varsayılan parametre kullanılamaz.
- Bir operatör için gereken parametre sayısı değiştirilemez.
- Yeni bir operatör yaratılamaz.

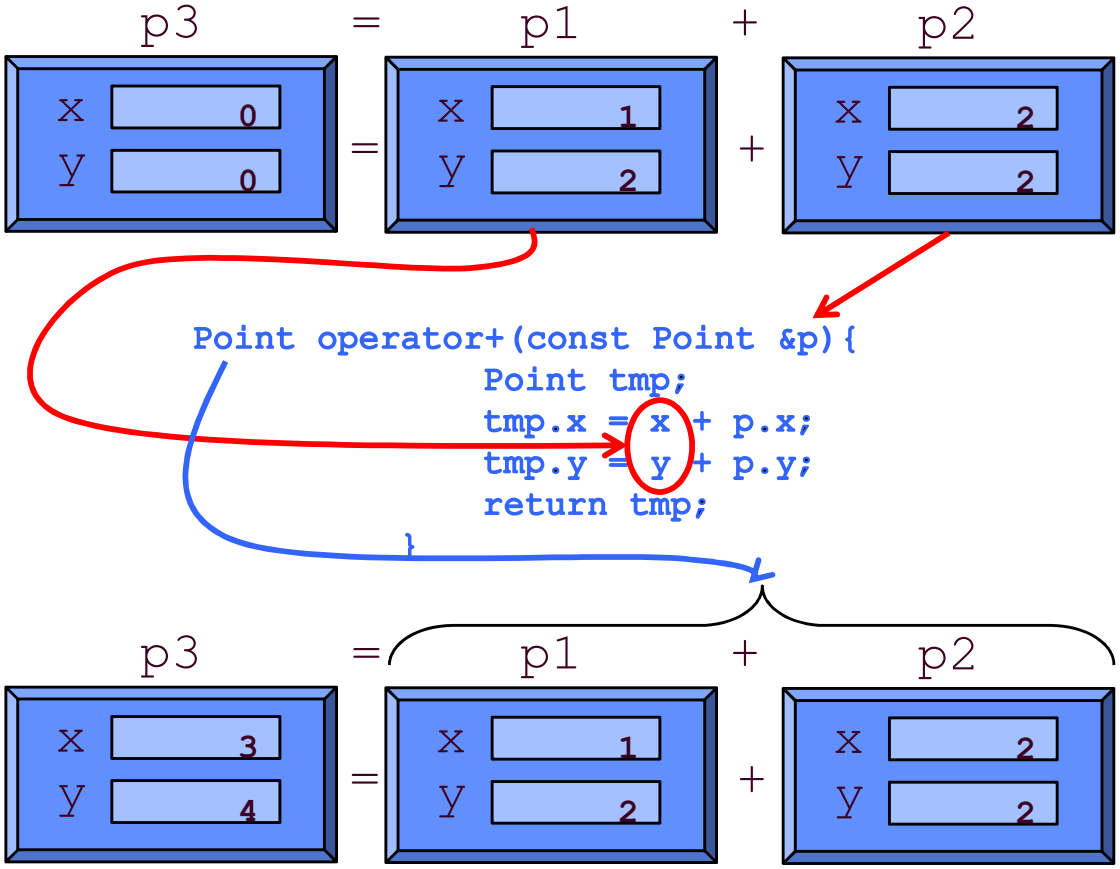
Sınıf Üyesi & Global Fonksiyon olarak Operatör Fonksiyonları

- Operatör fonksiyonları, bir sınıf üyesi (member functions) ya da global fonksiyon (global functions) olarak tanımlanabilir.

- `()`, `[]`, `->` ya da herhangi atama operatörleri (assignment operators), sınıf üyesi olarak beyan edilmelidir. Bunlar dışındaki operatörler üye fonksiyon da olabilir global fonksiyon da olabilir.

Sınıf Üyesi olarak Operatör Fonksiyonları

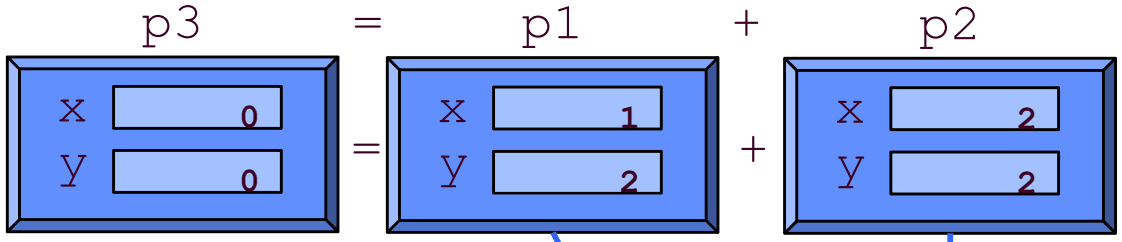
- Programda, `p3=p1.add(p2);` komutu ile `p3=p1+p2;` komutu aynı işlemi yürütmektedir. Ancak, operatör kullanımı program yazımını ve programın okunurluğunu kolaylaştırmaktadır.




```
#include <iostream>
using namespace std;
class Point{
            int x;
            int y;
public:
            Point(int X=0,int Y=0)
            :x(X),y(Y){}
            Point add(const Point &p){
                Point tmp;
                tmp.x=x+p.x;
                tmp.y=y+p.y;
                return tmp;
            }
            Point operator+(const Point &p){
                Point tmp;
                tmp.x=x+p.x;
                tmp.y=y+p.y;
                return tmp;
            }
};
int main(){
            Point p1(1,2),p2(2,2),p3;
            p3=p1.add(p2);
            p3=p1+p2;
            return 0;
}
```

Global Fonksiyon olarak Operatör Fonksiyonları

- Burada, + operatör fonksiyonunun global fonksiyon olarak gerçekleştirilmiş hali bulunmaktadır.




```
Point operator+(const Point &l, const Point &r)
{
        Point tmp;
        tmp.x=l.x+r.x;
        tmp.y=l.y+r.y;
        return tmp;
}
```

```
#include <iostream>
using namespace std;
class Point{
        friend Point operator+(const Point&
                               ,const Point &);
        int x;
        int y;
public:
        Point(int X=0,int Y=0)
        :x(X),y(Y){}
        Point add(const Point &p){
            Point tmp;
            tmp.x=x+p.x;
            tmp.y=y+p.y;
            return tmp;
        }
};
Point operator+(const Point &l, const Point &r)
{
        Point tmp;
        tmp.x=l.x+r.x;
        tmp.y=l.y+r.y;
        return tmp;
}
int main(){
        Point p1(1,2),p2(2,2),p3;
        p3=p1.add(p2);
        p3=p1+p2;
        return 0;
}
```

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

- C++ dilinde, temel girdi/çıktı işlemleri >> ve << operatörleri ile sağlanmakta ve bu operatör fonksiyonları, istream ve ostream sınıfları içinde tanımlıdır.

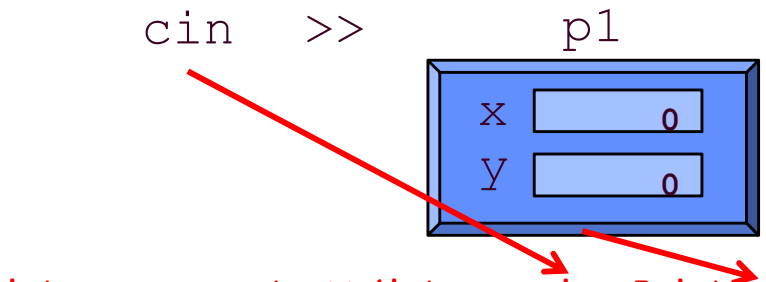
- Bu operatör ile sizin tanımladığınız nesneler üzerinde işlem yürütmek için bu operatör fonksiyonlarını yüklemeniz gerekecektir.

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

- `cin >> p1;` komutu çalıştığında, kullanıcıdan arada boşluk ile iki sayı girmesi beklenir.

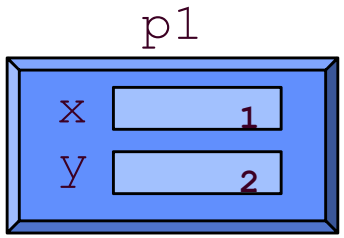
```
1 2 ↵
```

- Bu giriş sonrası, ilk değer p1 nesnesinin x değişkenine, ikinci değer y değişkenine atanır.




```
istream &operator>>(istream& in, Point& p)
{
        in >> p.x >> p.y;
        return in;
}
```

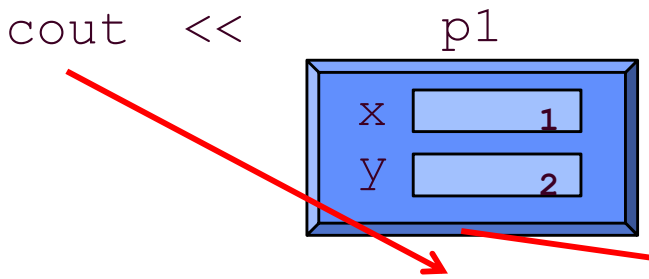
Sonrasında :




```
#include <iostream>
using namespace std;
class Point{
friend ostream &operator<<(ostream&,const Point&);
friend istream &operator>>(istream&, Point&);
        int x;
        int y;
public:
        Point(int X=0,int Y=0,int i=0)
        :x(X),y(Y){}
};
ostream &operator<<(ostream& out,const Point& p)
{
        out << "(" << p.x << "," << p.y << ")";
        return out;
}
istream &operator>>(istream& in, Point& p)
{
        in >> p.x >> p.y;
        return in;
}
int main(){
        Point p1;
        cin >> p1;
        cout << p1;
        return 0;
}
```

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

- `cout << p1;` komutu çalıştığında, p1 nesnesinin x ve y değişkenleri fonksiyonda belirtilen formatta ekrana yazdırılır.
- Bu giriş sonrası, ilk değer p1 nesnesinin x değişkenine, ikinci değer y değişkenine atanır.




```
ostream &operator<<(ostream& out,const Point& p)
{
        out << "(" << p.x << "," << p.y << ",";
        return out;
}
```

Sonrasında : (1,2)

```
#include <iostream>
using namespace std;
class Point{
friend ostream &operator<<(ostream&,const Point&);
friend istream &operator>>(istream&, Point&);
        int x;
        int y;
public:
        Point(int X=0,int Y=0,int i=0)
        :x(X),y(Y){}
};
ostream &operator<<(ostream& out,const Point& p)
{
        out << "(" << p.x << "," << p.y << ")";
        return out;
}
istream &operator>>(istream& in, Point& p)
{
        in >> p.x >> p.y;
        return in;
}
int main(){
        Point p1;
        cin >> p1;
        cout << p1;
        return 0;
}
```

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

- PhoneNumber sınıfı için, girdi (stream insertion) ve çıktı (extraction) operatörleri yüklenmektedir.

```
// PhoneNumber.h
// PhoneNumber class definition
#ifndef PHONENUMBER_H
#define PHONENUMBER_H

#include <iostream>
#include <string>
using namespace std;

class PhoneNumber
{
   friend ostream &operator<<( ostream &, const PhoneNumber & );
   friend istream &operator>>( istream &, PhoneNumber & );
private:
   string areaCode; // 3-digit area code
   string exchange; // 3-digit exchange
   string line; // 4-digit line
}; // end class PhoneNumber

#endif
```

The class ostream defines member functions that assist in formatting and writing output to sequences controlled by a stream buffer.

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

```
// overloaded stream insertion operator; cannot be
// a member function if we would like to invoke it with
// cout << somePhoneNumber;
ostream &operator<<( ostream &output, const PhoneNumber &number )
{

   output << "(" << number.areaCode << ") "
          << number.exchange << "-" << number.line;
   return output; // enables cout << a << b << c;

} // end function operator<<
```

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

setw(): Sets the number of characters to be used as the field width for the next insertion operation.

ignore(): Extracts characters from the input sequence and discards them.

```
// overloaded stream extraction operator; cannot be
// a member function if we would like to invoke it with
// cin >> somePhoneNumber;
istream &operator>>( istream &input, PhoneNumber &number )
{
   input.ignore(); // skip (
   input >> setw( 3 ) >> number.areaCode; // input area code
   input.ignore( 2 ); // skip ) and space
   input >> setw( 3 ) >> number.exchange; // input exchange
   input.ignore(); // skip dash (-)
   input >> setw( 4 ) >> number.line; // input line
   return input; // enables cin >> a >> b >> c;
} // end function operator>>
```

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

```
// main.cpp
// Demonstrating class PhoneNumber's overloaded stream insertion
// and stream extraction operators.
#include <iostream>
#include "PhoneNumber.h"
using namespace std;

int main()
{
   PhoneNumber phone; // create object phone

   cout << "Enter phone number in the form (123) 456-7890:" << endl;

   // cin >> phone invokes operator>> by implicitly issuing
   // the global function call operator>>( cin, phone )
   cin >> phone;

   cout << "The phone number entered was: ";

   // cout << phone invokes operator<< by implicitly issuing
   // the global function call operator<<( cout, phone )
   cout << phone << endl;
} // end main
```

Çıktı (Stream Insertion) ve Girdi (Extraction Operators) (<< , >>)

```
Enter phone number in the form (123) 456-7890:
(800) 555-1212
The phone number entered was: (800) 555-1212
```

Tek parametreli operatörler, Unary Operators (++, --, !)

- Bu operatörlerin yanında bir değişken bulunur.
  - *Üye fonksiyon olarak tanımlanırsa, parametre almaz*
  - *global fonksiyon olarak tanımlanırsa, bir parametre alır.*
- Önek (prefix) ve sonek(postfix) versiyonları bulunan artırma ve eksiltme operatörleridir.
  - *Önek(Prefix) formda (**Sınıf üye fonksiyonu olarak**)*
    - *Fonksiyon prototipi:*

```
className operator++ ();
```

    - *Fonksiyon tanımı:*

```
className className::operator++ ()
{
   //increment the value of the object by 1
   return *this;
}
```

Tek parametreli operatörler, Unary Operators (++, --, !)

- *Sonek(Postfix) formda (**Sınıf üye fonksiyonu olarak**)*
  - *Fonksiyon Prototipi:*

```
className operator++ (int);
```

  - *Fonksiyon tanımı :*

```
className className::operator++ (int u)
{
   className temp = *this; //use this pointer to copy
   //the value of the object
   //increment the object
   return temp; //return the old value of the object
}
```

Tek parametreli operatörler, Unary Operators (++, --, !)

- *Önek (Prefix) (**Global fonksiyon olarak**)*
  - *Fonksiyon Prototipi :*

```
friend className operator++ (className&);
```

  - *Fonksiyon tanımı :*

```
className operator++ (className& incObj)
{
   //increment the incObj by 1
   return incObj;
}
```

- *Sonek (Postfix) (**Global fonksiyon olarak**)*
  - *Fonksiyon Prototipi :*

```
friend className operator++ (className&, int);
```

  - *Fonksiyon tanımı :*

```
className operator++ (className& incObj, int u)
{
className temp = incObj;
//increment the incObj by 1
return temp;
}
```

İkili parametreli operatörler ,Binary Operators (+, -, ==)

- Bu operatörlerin yanında iki değişken bulunur.
  - *Üye fonksiyon olarak tanımlanırsa, bir parametre alır.*
  - *global fonksiyon olarak tanımlanırsa, iki parametre alır.*

- Üye fonksiyon olarak
  - *Fonksiyon Prototipi :*

```
returnType operator< (const className&) const;
```

  - *Fonksiyon tanımı :*

```
returnType className::operator< (const className& otherObject) const
{
   //algorithm to perform the operation
   return value;
}
```

Binary Operators (+, -, ==, <, etc.)

- Global fonksiyon olarak
  - *Fonksiyon Prototipi :*

```
friend returnType operator< (const className&, const className&);
```

  - *Fonksiyon tanımı :*

```
returnType operator< (const className& firstObject,
                      const className& secondObject)
{
   //algorithm to perform the operation
   return value;
}
```

Örnek Çalışma: Array Sınıfı

- Bu örnekte, güçlü bir dizi (Array) sınıfı yaratılmaktadır:
  - *Değer dizisi denetimi sağlar,*
  - *Bir array nesnesinin, bir başka array nesnesine atama operatörü ile kopyalanmasını sağlar.*
  - *Nesne, kendi boyutunu bilir,*
  - *Operatör kullanımı ile array nesnesine girdi ve çıktı işlemleri uygulanabilir.*
  - *Eşitlik karşılaştırma operatörleri == ve != ile array nesnelerinin karşılaştırması mümkün olur.*
- C++ Standart Şablon Kütüphanesi (Standard Template Library, STL), `vector` olarak adlandırılan ve burada ele alınan nesnenin çok daha kabiliyetlisi ve güçlüsünü sağlamaktadır.

Örnek Çalışma: Array Sınıfı (Sınıf Tanımı, Class Definition)

```
// Array.h
// Array class definition with overloaded operators.
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
using namespace std;

class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );
public:
   Array( int = 10 ); // default constructor
   Array( const Array & ); // copy constructor
   ~Array(); // destructor
   int getSize() const; // return size

   const Array &operator=( const Array & ); // assignment operator
   bool operator==( const Array & ) const; // equality operator
```

Örnek Çalışma: Array Sınıfı (Sınıf Tanımı, Class Definition)

```
   // inequality operator; returns opposite of == operator
   bool operator!=( const Array &right ) const
   {
      return ! ( *this == right ); // invokes Array::operator==
   } // end function operator!=

   // subscript operator for non-const objects returns modifiable lvalue
   int &operator[]( int );

   // subscript operator for const objects returns rvalue
   int operator[]( int ) const;
private:
   int size; // pointer-based array size
   int *ptr; // pointer to first element of pointer-based array
}; // end class Array

#endif
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// Array.cpp
// Array class member- and friend-function definitions.
#include <iostream>
#include <iomanip>
#include <cstdlib> // exit function prototype
#include "Array.h" // Array class definition
using namespace std;

// default constructor for class Array (default size 10)
Array::Array( int arraySize )
{
   size = ( arraySize > 0 ? arraySize : 10 ); // validate arraySize
   ptr = new int[ size ]; // create space for pointer-based array

   for ( int i = 0; i < size; i++ )
      ptr[ i ] = 0; // set pointer-based array element
} // end Array default constructor
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// copy constructor for class Array;
// must receive a reference to prevent infinite recursion
Array::Array( const Array &arrayToCopy )
   : size( arrayToCopy.size )
{
   ptr = new int[ size ]; // create space for pointer-based array

   for ( int i = 0; i < size; i++ )
      ptr[ i ] = arrayToCopy.ptr[ i ]; // copy into object
} // end Array copy constructor

// destructor for class Array
Array::~Array()
{
   delete [] ptr; // release pointer-based array space
} // end destructor

// return number of elements of Array
int Array::getSize() const
{
   return size; // number of elements in Array
} // end function getSize
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// overloaded assignment operator;
// const return avoids: ( a1 = a2 ) = a3
const Array &Array::operator=( const Array &right )
{
          if ( &right != this ) // avoid self-assignment{
                    // for Arrays of different sizes, deallocate original
                    // left-side array, then allocate new left-side array
                    if ( size != right.size ){
                              delete [] ptr; // release space
                              size = right.size; // resize this object
                              ptr = new int[ size ]; // create space for array copy
                    } // end inner if

                    for ( int i = 0; i < size; i++ )
                              ptr[ i ] = right.ptr[ i ]; // copy array into object
          } // end outer if

          return *this; // enables x = y = z, for example
}
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// determine if two Arrays are equal and
// return true, otherwise return false
bool Array::operator==( const Array &right ) const
{
          if ( size != right.size )
                    return false; // arrays of different number of elements

          for ( int i = 0; i < size; i++ )
                    if ( ptr[ i ] != right.ptr[ i ] )
          return false; // Array contents are not equal

          return true; // Arrays are equal
}
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// overloaded subscript operator for non-const Arrays;
// reference return creates a modifiable lvalue
int &Array::operator[]( int subscript )
{
          // check for subscript out-of-range error
          if ( subscript < 0 || subscript >= size )
          {
                    cerr << "\nError: Subscript " << subscript
                              << " out of range" << endl;
                    exit( 1 ); // terminate program; subscript out of range
          } // end if

          return ptr[ subscript ]; // reference return
} // end function operator[]
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// overloaded subscript operator for const Arrays
// const reference return creates an rvalue
int Array::operator[]( int subscript ) const
{
          // check for subscript out-of-range error
          if ( subscript < 0 || subscript >= size )
          {
                    cerr << "\nError: Subscript " << subscript
                              << " out of range" << endl;
                    exit( 1 ); // terminate program; subscript out of range
          } // end if
          return ptr[ subscript ]; // returns copy of this element
} // end function operator[]
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// overloaded input operator for class Array;
// inputs values for entire Array
istream &operator>>( istream &input, Array &a )
{
          for ( int i = 0; i < a.size; i++ )
                    input >> a.ptr[ i ];
          return input; // enables cin >> x >> y;
} // end function
```

Örnek Çalışma: Array Sınıfı (Sınıf İcrası, Class Implementation)

```
// overloaded output operator for class Array
ostream &operator<<( ostream &output, const Array &a )
{
          int i;
          // output private ptr-based array
          for ( i = 0; i < a.size; i++ )
          {
                    output << setw( 12 ) << a.ptr[ i ];
                    if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
                              output << endl;
          } // end for
          if ( i % 4 != 0 ) // end last line of output
                    output << endl;
          return output; // enables cout << x << y;
} // end function operator<<
```

Örnek Çalışma: Array Sınıfı (Test Programı)

```
// Array class test program.
#include <iostream>
#include "Array.h"
using namespace std;
int main()
{
          Array integers1( 7 ); // seven-element Array
          Array integers2; // 10-element Array by default
          // print integers1 size and contents
          cout << "Size of Array integers1 is "
                    << integers1.getSize()
                    << "\nArray after initialization:\n" << integers1;
          // print integers2 size and contents
          cout << "\nSize of Array integers2 is "
                    << integers2.getSize()
                    << "\nArray after initialization:\n" << integers2;
```

Örnek Çalışma: Array Sınıfı (Test Programı)

```
// input and print integers1 and integers2
cout << "\nEnter 17 integers:" << endl;
cin >> integers1 >> integers2;
cout << "\nAfter input, the Arrays contain:\n"
          << "integers1:\n" << integers1
          << "integers2:\n" << integers2;
// use overloaded inequality (!=) operator
cout << "\nEvaluating: integers1 != integers2" << endl;
if ( integers1 != integers2 )
          cout << "integers1 and integers2 are not equal" << endl;
// create Array integers3 using integers1 as an
// initializer; print size and contents
Array integers3( integers1 ); // invokes copy constructor
cout << "\nSize of Array integers3 is "
          << integers3.getSize()
          << "\nArray after initialization:\n" << integers3;
```

Örnek Çalışma: Array Sınıfı (Test Programı)

```
        // use overloaded assignment (=) operator
        cout << "\nAssigning integers2 to integers1:" << endl;
        integers1 = integers2; // note target Array is smaller
        cout << "integers1:\n" << integers1
             << "integers2:\n" << integers2;
        // use overloaded equality (==) operator
        cout << "\nEvaluating: integers1 == integers2" << endl;
        if ( integers1 == integers2 )
             cout << "integers1 and integers2 are equal" << endl;
        // use overloaded subscript operator to create rvalue
        cout << "\nintegers1[5] is " << integers1[ 5 ];
        // use overloaded subscript operator to create lvalue
        cout << "\n\nAssigning 1000 to integers1[5]" << endl;
        integers1[ 5 ] = 1000;
        cout << "integers1:\n" << integers1;
        // attempt to use out-of-range subscript
        cout << "\nAttempt to assign 1000 to integers1[15]" << endl;
        integers1[ 15 ] = 1000; // ERROR: out of range
} // end main
```

Örnek Çalışma: Array Sınıfı (Test Programı Çıktısı)

```
Size of Array integers1 is 7
Array after initialization:
           0          0          0          0
           0          0          0

Size of Array integers2 is 10
Array after initialization:
           0          0          0          0
           0          0          0          0
           0          0

Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the Arrays contain:
integers1:
           1          2          3          4
           5          6          7
integers2:
           8          9         10         11
          12         13         14         15
          16         17

Evaluating: integers1 != integers2
integers1 and integers2 are not equal
```

Örnek Çalışma: Array Sınıfı (Test Programı Çıktısı)

```
Size of Array integers3 is 7
Array after initialization:
           1          2          3          4
           5          6          7

Assigning integers2 to integers1:
integers1:
           8          9         10         11
          12         13         14         15
          16         17
integers2:
           8          9         10         11
          12         13         14         15
          16         17

Evaluating: integers1 == integers2
integers1 and integers2 are equal

integers1[5] is 13

Assigning 1000 to integers1[5]
integers1:
           8          9         10         11
          12       1000         14         15
          16         17
```

Örnek Çalışma: Array Sınıfı (Test Programı Çıktısı)

```
Attempt to assign 1000 to integers1[15]

Error: Subscript 15 out of range
```

Örnek Çalışma: Date Sınıfı (Sınıf Tanımı, Class Definition)

- Bu örnek, bir tarih (**`Date`**) sınıfını sunar,
  - *Arttırma operatörünün önek (prefix) ve sonek(postfix) formlarını kapsar.*

Örnek Çalışma: Date Sınıfı (Sınıf Tanımı, Class Definition)

```
#include <iostream>
using namespace std;
class Date
{
          friend ostream &operator<<( ostream &, const Date & );
public:
          Date( int m = 1, int d = 1, int y = 1900 ); // default constructor
          void setDate( int, int, int ); // set month, day, year
          Date &operator++(); // prefix increment operator
          Date operator++( int ); // postfix increment operator
          const Date &operator+=( int ); // add days, modify object
          static bool leapYear( int ); // is date in a leap year?
          bool endOfMonth( int ) const; // is date at the end of month?
private:
          int month;
          int day;
          int year;
```

Örnek Çalışma: Date Sınıfı (Sınıf Tanımı, Class Definition)

```
        static const int days[]; // array of days per month
        void helpIncrement(); // utility function for incrementing date
}; // end class Date
#endif
```

Örnek Çalışma: Date Sınıfı (Sınıf İcrası, Class Implementation)

```
// Date.cpp
// Date class member- and friend-function definitions.
#include <iostream>
#include <string>
#include "Date.h"
using namespace std;
// initialize static member; one classwide copy
const int Date::days[] =
          { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
// Date constructor
Date::Date( int m, int d, int y )
{
          setDate( m, d, y );
} // end Date constructor
// set month, day and year
void Date::setDate( int mm, int dd, int yy )
{
          month = ( mm >= 1 && mm <= 12 ) ? mm : 1;
          year = ( yy >= 1900 && yy <= 2100 ) ? yy : 1900;
```

Örnek Çalışma: Date Sınıfı (Sınıf İcrası, Class Implementation)

```
            // test for a leap year
            if ( month == 2 && leapYear( year ) )
                        day = ( dd >= 1 && dd <= 29 ) ? dd : 1;
            else
                        day = ( dd >= 1 && dd <= days[ month ] ) ? dd : 1;
} // end function setDate
// overloaded prefix increment operator
Date &Date::operator++()
{
            helpIncrement(); // increment date
            return *this; // reference return to create an lvalue
} // end function operator++
// overloaded postfix increment operator; note that the
// dummy integer parameter does not have a parameter name
Date Date::operator++( int )
{
            Date temp = *this; // hold current state of object
            helpIncrement();
            // return unincremented, saved, temporary object
            return temp; // value return; not a reference return
} // end function operator++
```

Örnek Çalışma: Date Sınıfı (Sınıf İcrası, Class Implementation)

```
// add specified number of days to date
const Date &Date::operator+=( int additionalDays )
{
        for ( int i = 0; i < additionalDays; i++ )
                helpIncrement();
        return *this; // enables cascading
} // end function operator+=
// if the year is a leap year, return true; otherwise, return false
bool Date::leapYear( int testYear )
{
        if ( testYear % 400 == 0 ||
                        ( testYear % 100 != 0 && testYear % 4 == 0 ) )
                return true; // a leap year
        else
                return false; // not a leap year
} // end function leapYear
```

Örnek Çalışma: Date Sınıfı (Sınıf İcrası, Class Implementation)

```
// determine whether the day is the last day of the month
bool Date::endOfMonth( int testDay ) const
{
          if ( month == 2 && leapYear( year ) )
                    return testDay == 29; // last day of Feb. in leap year
          else
                    return testDay == days[ month ];
} // end function endOfMonth

// function to help increment the date
void Date::helpIncrement()
{
          // day is not end of month
          if ( !endOfMonth( day ) )
                    day++; // increment day
          else
                    if ( month < 12 ) // day is end of month and month < 12
                    {
                              month++; // increment month
                              day = 1; // first day of new month
                    } // end if
                    else // last day of year
                    {
```

Örnek Çalışma: Date Sınıfı (Sınıf İcrası, Class Implementation)

```
                    year++; // increment year
                    month = 1; // first month of new year
                    day = 1; // first day of new month
          } // end else
} // end function helpIncrement
// overloaded output operator
ostream &operator<<( ostream &output, const Date &d )
{
          static string monthName[ 13 ] = { "", "January", "February",
                    "March", "April", "May", "June", "July", "August",
                    "September", "October", "November", "December" };
          output << monthName[ d.month ] << ' ' << d.day << ", " << d.year;
          return output; // enables cascading
} // end function operator<<
```

Örnek Çalışma: Date Sınıfı (Test Programı)

```
// main.cpp
// Date class test program.
#include <iostream>
#include "Date.h" // Date class definition
using namespace std;
int main()
{
          Date d1; // defaults to January 1, 1900
          Date d2( 12, 27, 1992 ); // December 27, 1992
          Date d3( 0, 99, 8045 ); // invalid date
          cout << "d1 is " << d1 << "\nd2 is " << d2 << "\nd3 is " << d3;
          cout << "\n\nd2 += 7 is " << ( d2 += 7 );
          d3.setDate( 2, 28, 1992 );
          cout << "\n\n d3 is " << d3;
          cout << "\n++d3 is " << ++d3 << " (leap year allows 29th)";
```

Örnek Çalışma: Date Sınıfı (Test Programı)

```
         Date d4( 7, 13, 2002 );
         cout << "\n\nTesting the prefix increment operator:\n"
              << " d4 is " << d4 << endl;
         cout << "++d4 is " << ++d4 << endl;
         cout << " d4 is " << d4;
         cout << "\n\nTesting the postfix increment operator:\n"
              << " d4 is " << d4 << endl;
         cout << "d4++ is " << d4++ << endl;
         cout << " d4 is " << d4 << endl;
} // end main
```

Örnek Çalışma: Date Sınıfı (Test Programı Çıktısı)

```
d1 is January 1, 1900
d2 is December 27, 1992
d3 is January 1, 1900

d2 += 7 is January 3, 1993

  d3 is February 28, 1992
++d3 is February 29, 1992 (leap year allows 29th)

Testing the prefix increment operator:
  d4 is July 13, 2002
++d4 is July 14, 2002
  d4 is July 14, 2002

Testing the postfix increment operator:
  d4 is July 14, 2002
d4++ is July 14, 2002
  d4 is July 15, 2002
```

Standart Kütüphane Sınıfı (`string`)

- `String` sınıfı bir çok yüklenmiş operatör içerir. Ayrıca, `empty`, `substr` ve `at` gibi kullanışlı üye fonksiyonları vardır.

Standart Kütüphane Sınıfı (`string`) (Test Program)

```
// Standard Library string class test program.
#include <iostream>
#include <string>
using namespace std;
int main()
{
          string s1( "happy" );
          string s2( " birthday" );
          string s3;
          // test overloaded equality and relational operators
          cout << "s1 is \"" << s1 << "\"; s2 is \"" << s2
                    << "\"; s3 is \"" << s3 << '\"'
                    << "\n\nThe results of comparing s2 and s1:"
                    << "\ns2 == s1 yields " << ( s2 == s1 ? "true" : "false" )
                    << "\ns2 != s1 yields " << ( s2 != s1 ? "true" : "false" )
                    << "\ns2 > s1 yields " << ( s2 > s1 ? "true" : "false" )
                    << "\ns2 < s1 yields " << ( s2 < s1 ? "true" : "false" )
                    << "\ns2 >= s1 yields " << ( s2 >= s1 ? "true" : "false" )
                    << "\ns2 <= s1 yields " << ( s2 <= s1 ? "true" : "false" );
```

Standart Kütüphane Sınıfı (`string`) (Test Program)

```
// test string member function empty
cout << "\n\nTesting s3.empty():" << endl;
if ( s3.empty() )
{
          cout << "s3 is empty; assigning s1 to s3;" << endl;
          s3 = s1; // assign s1 to s3
          cout << "s3 is \"" << s3 << "\"";
} // end if
// test overloaded string concatenation operator
cout << "\n\ns1 += s2 yields s1 = ";
s1 += s2; // test overloaded concatenation
cout << s1;
// test overloaded string concatenation operator with C-style string
cout << "\n\ns1 += \" to you\" yields" << endl;
s1 += " to you";
cout << "s1 = " << s1 << "\n\n";
```

Standart Kütüphane Sınıfı (`string`) (Test Program)

```
// test string member function substr
cout << "The substring of s1 starting at location 0 for\n"
          << "14 characters, s1.substr(0, 14), is:\n"
          << s1.substr( 0, 14 ) << "\n\n";
// test substr "to-end-of-string" option
cout << "The substring of s1 starting at\n"
          << "location 15, s1.substr(15), is:\n"
          << s1.substr( 15 ) << endl;
// test copy constructor
string s4( s1 );
cout << "\ns4 = " << s4 << "\n\n";
```

Standart Kütüphane Sınıfı (`string`) (Test Program)

```
         // test overloaded assignment (=) operator with self-assignment
         cout << "assigning s4 to s4" << endl;
         s4 = s4;
         cout << "s4 = " << s4 << endl;
         // test using overloaded subscript operator to create lvalue
         s1[ 0 ] = 'H';
         s1[ 6 ] = 'B';
         cout << "\ns1 after s1[0] = 'H' and s1[6] = 'B' is: "
              << s1 << "\n\n";
         // test subscript out of range with string member function "at"
         cout << "Attempt to assign 'd' to s1.at( 30 ) yields:" << endl;
         s1.at( 30 ) = 'd'; // ERROR: subscript out of range
} // end main
```

Standart Kütüphane Sınıfı (`string`) (Test Program Çıktısı)

```
s1 is "happy"; s2 is " birthday"; s3 is ""

The results of comparing s2 and s1:
s2 == s1 yields false
s2 != s1 yields true
s2 >  s1 yields false
s2 <  s1 yields true
s2 >= s1 yields false
s2 <= s1 yields true

Testing s3.empty():
s3 is empty; assigning s1 to s3;
s3 is "happy"

s1 += s2 yields s1 = happy birthday

s1 += " to you" yields
s1 = happy birthday to you

The substring of s1 starting at location 0 for
14 characters, s1.substr(0, 14), is:
happy birthday
```

Standart Kütüphane Sınıfı (`string`) (Test Program Çıktısı)

```
The substring of s1 starting at
location 15, s1.substr(15), is:
to you

s4 = happy birthday to you

assigning s4 to s4
s4 = happy birthday to you

s1 after s1[0] = 'H' and s1[6] = 'B' is: Happy Birthday to you

Attempt to assign 'd' to s1.at( 30 ) yields:

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

Kaynaklar

- T.C. Lethbridge and R. Laganiere, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.