C'den C++'a Geçiş

Dr. Metin Özkan

Amaç

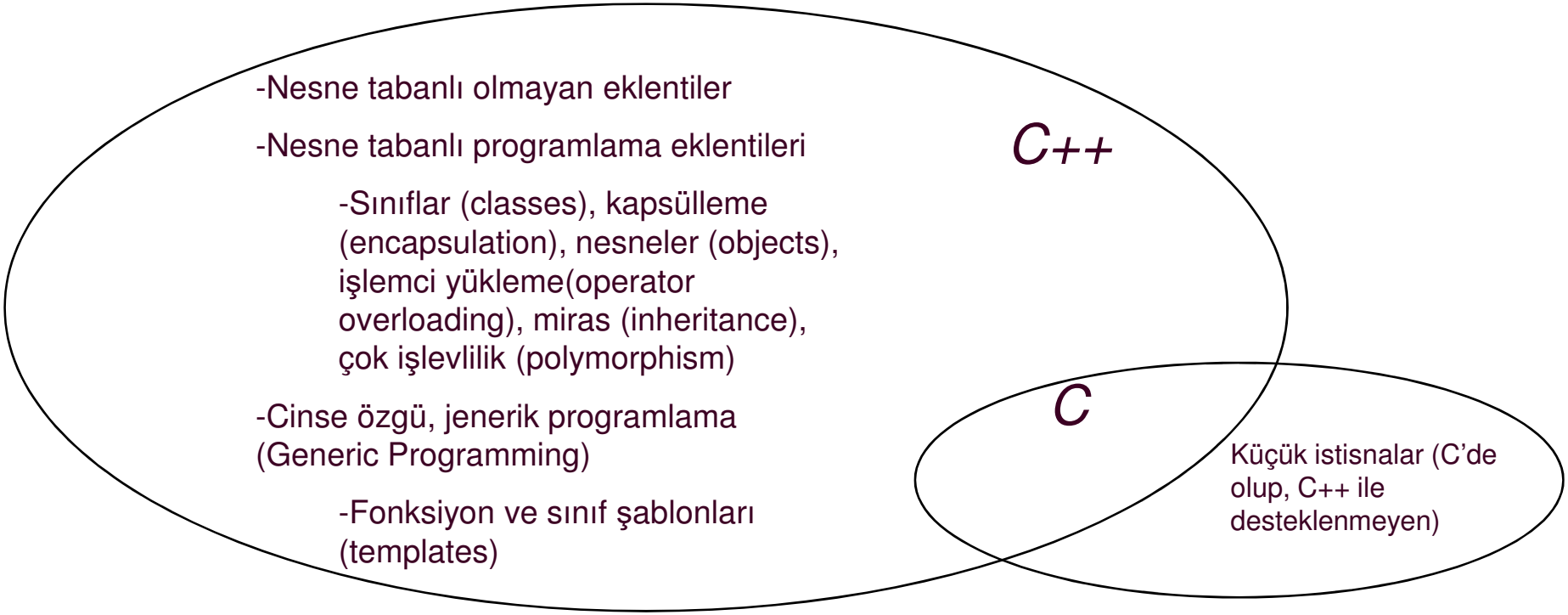
- Nesne tabanlı yaklaşımları kullanmadan, C++ programlama yapıları kullanarak prosedürel program yazmak,
- Nesne tabanlı olmayan, C üzerine C++ gelişmelerini öğrenmek.

Daha iyi bir C: C++

- C++, C programlama dilini kapsamaktadır.
- C++, C üzerine bazı özellikler ekler.
- C++, C'ye göre çok güçlü tip kontrolü sağlar ve C dilinin sahip olduğu hemen hemen tüm programlama stillerini kapsar.
- C++ için daha iyi bir C tanımlaması şu gerekçe ile yapılmaktadır:
  - *C++ ile, C programlama stili (procedural) kullanılarak program yazılabilir; ancak, daha iyi tip kontrolü ve daha fazla notasyon desteği (etkinlik kaybedilmeden) olacaktır.*

Daha iyi bir C: C++

- C++, C programlama diline bazı eklentiler kazandırmıştır.
  - *Nesne tabanlı olmayan eklentiler: Program kodlama esnasında kullanılan ve faydalanılan özellikler.*
  - *Nesne tabanlı programlamayı destekleyen özellikler*
  - *Jenerik programlamayı destekleyen özellik eklentileri*



Açıklama (Comment)

- "//" sembolü, satır sonuna kadar tüm yazılanları açıklama olarak değerlendirilir.

```
int number1, number2; //Bu bir açıklamadır...
                      // Bu da bir açıklamadır...
```

  - C++ programcısı, "/* … */" şeklinde C-stilindeki açıklama formunu da kullanabilir.

```
int number1, number2; /*  Bu bir açıklamadır...
                          Bu da bir açıklamadır... */
```

  - Büyük kod bölütlerini, açıklama haline getirmek için en iyi yol ön işlemci (preprocessor) kullanımıdır. #ifdef, #endif

```
#ifdef _identifier_
    a= b + c;
    v= a * c;
#endif
```

Beyan (Declaration) ve tanım (Definition)

- **Beyan (declaration):** Bir değişken, fonksiyon ya da sınıf için beyanda bulunulur. Bu isimde, bu tipte veya bu prototipe sahip bir değişken ya da fonksiyonun herhangi bir yerde bulunduğu derleyiciye beyan edilir. Derleyici, beyanda bulunulan şeyin tam tanımı olmaksızın kodlarda yer almasına izin verir. Özellikle birden çok kaynak dosya üzerinde çalışılıyorsa, her kaynak dosyada fonksiyonun kodlarını vermek yerine beyanı verilir.
- **Tanım (Definition):** Varlığın yaratılması için gereken tüm bilginin tanımlanmasını belirtir. Bir fonksiyonu tanımlamak demek, fonksiyonun işleteceği tüm kodların yazılması demektir.

Beyan (Declaration) ve tanım (Definition)

- Örneğin;

```
extern int i;          //Beyan (Decrelation)

struct Point{          //Beyan (Decrelation)
    float x;
    float y;
};

int func(int);         //Beyan (Decrelation)



int i;                 //Tanım (Definition)

Point p1,p2;           //Tanım (Definition)

int func(int i){       //Tanım (Definition)
    return i++;
}
```

- C'de, beyan ve tanımlar bir bloğun başında bulunur.
- C++'da, herkangi bir yere yerleştirilebilir ancak, kullanılmadan önce görünür olmalıdır.

Kapsam (Scope) Kuralları

- Beyanlar (Declarations), her hangi bir yerde olabilir, ancak kullanılmadan önce görünür olmalıdır.
- Bir beyanda bulunulan varlığın, kullanılabildiği program bölütü onun kapsamı (scope) olarak tanımlanır.

```
int x=1;                        // global değişken beyan ve tanımı
void function1(int);            // fonksiyon beyanı (declaration)

int main()
{
    int x=5;                   //local değişken beyanı(declaration), scope
                               //of main
    for(int i=0;i<5;i++){ // scope of for-loop
            x++;
    }
    int y=1;
    {                          //Yeni bir kapsam başlat (start new scope)
            int y=1;
            y= y+x;
    }                          //Yeni kapsamı kapat (end new scope)
    ::x++;                     // global x değişkeni (::, kapsam
                          //işlemcisidir <scope resolution operator>)
}
```

İsim Alanları (Namespaces)

- İsim alanları (Namespaces), sınıf, nesne, değişken ve fonksiyon gibi tanımlamaların bir isim altında gruplamayı sağlamaktadır. Böylece, isim benzerliği nedeniyle hatalı kod işletilmesi engellenerek, kodlamaların tutarlı olması sağlanabilmektedir.
- Farklı programcı ya da kod tedarikçilerinden temin edilen kütüphane fonksiyonlarının isim benzerliğinden çakışma olması ya da hatalı fonksiyon çağrısı gerçekleştirilmesinin önüne geçilmektedir.

```
//Vendor1.h
namespace Vendor1 {              //Vendor1,adlı tedarikçinin kodları
    int a;                       // Vendor1'de tanımlı değişken
    void f(){                    // Vendor1'de tanımlı fonksiyon
            .
            .
    }
}
//Vendor2.h
namespace Vendor2 {               //Vendor2,adlı tedarikçinin kodları
    int a;                        // Vendor1'de tanımlı değişken
}
```

İsim Alanları (Namespaces)

- İsim alanları içerisinde tanımlı, değişken ve fonksiyonlara erişim için farklı yaklaşımlar vardır.
  - *İlk yaklaşım, değişken ya da fonksiyona ait olduğu isim alanı belirtilerek erişmek.*

```
Vendor1::a=1;  //Vendor1's a
Vendor2::a=5;  //Vendor2's a
Vendor1::f();  // Vendor1's function f()
```

  - *Diğer yaklaşım, ´using´ direktifi kullanarak erişim.*

```
using namespace Vendor1;
a=1;                                    //Vendor1's a
f();                                    //Vendor1's function f()

Ya da

using Vendor2::a;
a=5;                                    //Vendor2's a
```

C++ Standart Kütüphane ve Başlık Dosyaları (C++ Standard Library and Header Files

- Program yazılırken, standart kütüphanede bulunan kodlar yoğun olarak kullanılır. Kullanılan kodun bulunduğu başlık dosyasının kodda tanımlanması gerekir.
- C, dosya isimlerinde genellikle .c uzantısı kullanılır. C++ dosya isimlerinde, birkaç farklı uzantı kullanılabilmektedir: .cpp, .cxx, .C
- Eski stilde, standart kütüphane başlık dosyası
  - *# include <iostream.h>*

  olarak belirtilmekte ,iken C++'da
  - *# include <iostream>*

  olarak belirtilmektedir.
- Ayrıca bazı başlık dosyası isimlerinde de değişiklik olmuştur.
  - *<math.h>* → *<cmath>*, *<stdlib.h>* → *<cstdlib>*

  , *<stdio.h>* → *<cstdio>*

Boolean Değişkeni (true/false)

- C++'da bool olarak ifade edilen değişken tipi bulunmaktadır. Doğru/yanlış (true/false) şeklinde mantıksal değer alır.
- Kodlamada kullanılan bu değişken, derleyici tarafından int tipine dönüştürülür. Sıfırdan farklı değer true, sıfır değer false olarak dönüştürülür.

```
// Addition program of two numbers
#include <iostream>
int main(){
          bool isEqual;                    //Değişken tanımlama

          isEqual=true;                    //Değişkene değer atama
          if(isEqual)
                    .
                    .

          return 0;
} // end function main
```

Giriş/Çıkış (Input/Output)

- Printf ve scanf fonksiyonları yerine cin ve cout nesneleri << ve >> operatörleri ile birlikte kullanılabilir.

```
// Addition program of two numbers
#include <iostream>       // allows program to perform I/O
int main()
{
        int number1;      // first integer to add

        std::cout << "Enter first integer: "; //prompt user for first int
        std::cin >> number1;      // read first int

        int number2;      // second int to add
        int sum;          // sum of two number

        using namespace std;

        cout << "Enter second integer: ";//prompt user for second int
        cin >> number2;                  // read second int

        sum=number1 + number2; //add the numbers
        cout <<"Sum is " << sum << endl;  //display sum; end line

        return 0; //indicate that program ended successfully
} // end function main
```

Dinamik Bellek Tahsisi (Dynamic Memory Allocation)

- C'de, dinamik bellek tahsisi "malloc" ve "free" adlı standart kütüphane fonksiyonları ile sağlamaktadır.
- C++'da ise, "new" and "delete" işlemcileri (operators) ile daha kolay bir kullanım ile bu işlem yapılmaktadır.

```
int *ptr;      // a pointer to integer
ptr= new int; // memory allocation for pointer
. . .
delete ptr;    // Releasing the memory
```

```
int *ptr;          // a pointer to integer
ptr= new int(1);   // memory allocation by setting initial value of 1
. . .
delete ptr;
```

```
int *ptr;          // a pointer to integer
ptr= new int[5];   // memory allocation for 5 integers
. . .
delete [] ptr;     // Releasing whole memory
```

Sabitler (Constants)

- C'de, sabitler (constants) ön işlemci (preprocessor) kullanımı ile tanımlanmaktadır.
  - `#define PI 3.14`
- C++'da, "const" anahtar kelimesi ile tanımlanmaktadır.

```
int const a=10;        //a is constant with a value of 10, cannot be changed.

              ya da

const int a=10;        //the same as above.


a=5;                   //Derleme hatası, çünkü a bir sabittir.
```

- Değişkenler, sabit olarak tanımlanmasa da program aynen çalışır. Ancak, sabitler program yazımı sırasında hata yapılma olasılığını azaltır.

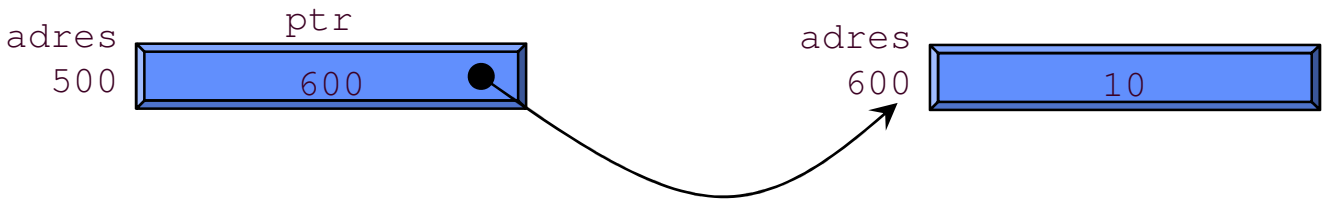
Sabitler (Constants)

- Sabitlerin göstergelerle (pointer) kullanımı da sözkonusudur.

```
const int *ptr= new int(10); // data constant, however pointer is not constant
int const *ptr= new int(10); //the same as above.

*ptr=5;      // error
ptr++;       // OK
```

```
int * const ptr= new int(10); // data may change, however pointer is constant

*ptr=5;              // OK
ptr++;               // error
```

```
const int * const ptr= new int(10); // both data and pointer are constant
int const * const ptr= new int(10); // the same as above

*ptr=5;              // error
ptr++;               // error
```



Satıriçi Fonksiyonlar (Inline Functions) (Macros)

- C'de, makrolar #define direktifi ile tanımlanmaktadır.
- C++'da, ´inline´ anahtar kelimesi ile fonksiyonların makro olarak tanımlanması mümkün olmaktadır.

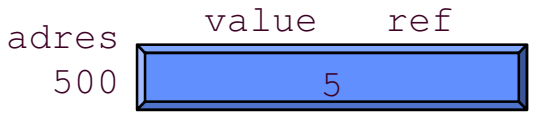
```
#include <iostream>       // allows program to perform I/O
using namespace std;
inline double cube (const double side)
{
        return side*side*side;
}
int main()
{
        double sideValue=2;       // first integer to add
        cout<< "Volume of the cube with the side length of"
                 << sideValue << "is" << cube(sideValue)<< endl;
} // end function main
```

- Derleyici, bu fonkisyonun kullanıldığı yerlere fonksiyon kodlarını kopyalar. Fonksiyon çağrısı gerçekleşmez.
- Programın çalışma hızı artar; ancak, program boyutu da artar.

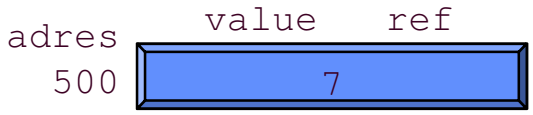
Referans Değişken

- Referans değişken, bir değişkeni farklı bir isim ile kullanmak için tanımlanır. Farklı isimler ile aynı değişkene erişilebilir.
- Referans değişken, & (reference) operatörü ile tanımlanır.
- Örneğin;

```
int value = 5;
int &ref = value;
```




```
ref = 7;
```



`ref` adlı değişkene 7 değeri atandığında, `value` adlı değişkenin de değeri 7'dir. Çünkü, her ikisi de aynı adresteki değişkene verilen iki farklı isimdir.

Fonksiyon Parametreleri

- Bir fonksiyona, iki şekilde parametre değerleri aktarılabilir: Değer parametresi (pass-by-value) ve referans parametresi (pass-by-reference).
  - *Değer parametresi (pass-by-value): Fonksiyon parametresi olarak fonksiyon içerisinde kullanılacak değişken için bellekte yer açılır ve gerçek parametre değerinin kopyası yollanır. Fonksiyon bitiminde, bellekteki alan yok olur ama gerçek parametre bundan etkilenmez.*

```
int increment(int i){
      return ++i;
}
int main(){
      int value=5, result;
      result=increment(value);
      return 0;
}
```

Fonksiyon Parametreleri

- *referans parametresi (Pass-by-reference): Fonksiyon parametresi olarak gerçek değişkenin referansı gönderilir. Fonksiyonda, parametre değişken için bellekte yeni bir alan açılmaz. Fonksiyon içerisindeki parametre değişkeninde yapılan değişiklik gerçek değişkene yansır.*

```
int increment(int& i){
      return ++i;
}
int main(){
      int value=5, result;
      result=increment(value);
      return 0;
}
```

- *referans parametresi (Pass-by-reference) kullanılmasının önemli bir avantajı vardır. Fonksiyon parametresi olarak gönderilen verinin boyutu büyük ise, bellek alan kullanımı ve işlem süresi oldukça çok artar.*

Fonksiyon Parametreleri (Referans/Değer)

```
// Comparing pass-by-value and pass-by-reference with references.
#include <iostream>
using namespace std;

int squareByValue( int ); // function prototype (value pass)
void squareByReference( int & ); // function prototype (reference pass)

int main()
{
   int x = 2; // value to square using squareByValue
   int z = 4; // value to square using squareByReference

   // demonstrate squareByValue
   cout << "x = " << x << " before squareByValue\n";
   cout << "Value returned by squareByValue: "
      << squareByValue( x ) << endl;
   cout << "x = " << x << " after squareByValue\n" << endl;

   // demonstrate squareByReference
   cout << "z = " << z << " before squareByReference" << endl;
   squareByReference( z );
   cout << "z = " << z << " after squareByReference" << endl;
} // end main
```

Function illustrating pass-by-value

Function illustrating pass-by-reference

Variable is simply mentioned by name in both function calls

Fonksiyon Parametreleri (Referans/Değer)

```
// squareByValue multiplies number by itself, stores the
// result in number and returns the new value of number
int squareByValue( int number )
{
   return number *= number; // caller's argument not modified
} // end function squareByValue

// squareByReference multiplies numberRef by itself and stores the result
// in the variable to which numberRef refers in function main
void squareByReference( int &numberRef )
{
   numberRef *= numberRef; // caller's argument modified
} // end function squareByReference
```

Receives copy of argument in `main`

Receives reference to argument in `main`

Modifies variable in `main`

Output:

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

Fonksiyonlar (Varsayılan Parametreler -Default Arguments)

- Bir varsayılan değer, fonksiyonun parametresi olarak tanımlanabilir. Bu durumda, gereken parametre için değer yollanmazsa, fonksiyon varsayılan değeri kullanır.

```
void print(int i=1, int j=2){          //Varsayılan değerler ile iki parametre
     cout << i << " " << j <<endl;
}
int main(){
     print(5, 10);     //2 gerçek parametre verilerek fonksiyon çağrısı
     print(5);         //1 gerçek parametre verilerek fonksiyon çağrısı
     print();          //Gerçek parametresiz fonksiyon çağrısı
     return 0;
}
```

- Fonksiyonun parametre listesinde, varsayılan değerler en sağda bulunmalıdır.

```
     void print(int i=1, int j);        //Hatalı
     void print(int i, int j=2);        //uygun
```

Fonksiyonlar (Varsayılan Parametreler -Default Arguments)

- Varsayılan değerler, fonksiyonun beyanında (declaration) verilmelidir.

```
void print(int, int =2);   //Beyan(declaration), fonksiyon prototipi
        yada
void print(int i, int j=2){  //Beyan(declaration) ve tanımı (definition)
        …
}
```

- Fonksiyonun hem beyan hem de tanımında varsayılan değer verilirse hatalı olacaktır.

Fonksiyonlar
(Varsayılan Parametreler -Default Arguments)

```
// Using default arguments.
#include <iostream>
using namespace std;

// function prototype that specifies default arguments
int boxVolume( int length = 1 , int width = 1, int height = 1 );

int main()
{
   // no arguments--use default values for all dimensions
   cout << "The default box volume is: " << boxVolume();

   // specify length; default width and height
   cout << "\n\nThe volume of a box with length 10,\n"
      << "width 1 and height 1 is: " << boxVolume( 10 );

   // specify length and width; default height
   cout << "\n\nThe volume of a box with length 10,\n"
      << "width 5 and height 1 is: " << boxVolume( 10, 5 );

   // specify all arguments
   cout << "\n\nThe volume of a box with length 10,\n"
      << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
      << endl;
} // end main
```

Default arguments

Calling function with no arguments

Calling function with one argument

Calling function with two arguments

Calling function with three arguments

Fonksiyonlar
(Varsayılan Parametreler -Default Arguments)

```
// function boxVolume calculates the volume of a box
int boxVolume( int length, int width, int height )
{
   return length * width * height;
} // end function boxVolume
```

Note that default arguments were specified in the function prototype, so they are not specified in the function header

Output:

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

Fonksiyonlar (Fonksiyon Yükleme-Function Overloading)

- C++, aynı isme sahip birden çok fonksiyonun bulunmasına imkan vermektedir. Ancak, fonksiyonların parametre sayıları ve/veya tipleri ile farklılaşması gerekmektedir.
  - *Yüklenmiş fonksiyonlar,*
    - *Aynı isme*
    - *Farklı parametre kümesine sahiptir.*
- Derleyici, fonksiyon çağrısı sırasında verilen parametre sayısı ve tipine göre uygun fonksiyonu seçer ve işletir.

Fonksiyonlar (Fonksiyon Yükleme-Function Overloading) Örnek

```
// Overloaded functions.
#include <iostream>
using namespace std;

// function square for int values
int square( int x )
{
   cout << "square of integer " << x << " is ";
   return x * x;
} // end function square with int argument

// function square for double values
double square( double y )
{
   cout << "square of double " << y << " is ";
   return y * y;
} // end function square with double argument

double square( double y,double x)
{
   cout << "square of double + double " << y << " is ";
   return y * x;
} // end function square with double argument
```

Defining a **square** function for **int**s

Defining a **square** function for **double**s

Fonksiyonlar (Fonksiyon Yükleme-Function Overloading) Örnek

```
int main()
{
   cout << square( 7 ); // calls int version
   cout << endl;
   cout << square( 7.5 ); // calls double version
   cout << endl;
         cout << square( 7.5,3.2 ); // calls double version
   cout << endl;

} // end main
```

Output:

```
square of integer 7 is 49
square of double 7.5 is 56.25
```

Output confirms that the proper function was called in each case

Operatör Yükleme (Operator Overloading)

- C++'da, operatörler (+,-,=, ++, v.b.) programcının tanımladığı veri tipleri ile birlikte kullanılması mümkündür. Ancak, operatör fonksiyonunun kodlanması gerekir.
- Örneğin, + operatörünü iki karmaşık sayının toplanmasında kullanmak isteyelim.

```
struct Complex{
     double Re;
     double Im;
};
Complex operator+ (const Complex &a, const Complex &b){
     Complex sum;
     sum.Re=a.Re+b.Re;
     sum.Im=a.Im+b.Im;
     return sum;
}
int main()
{
     Complex s,a1={1,2},a2={3,4};
     s= a1 + a2; // similar to "s=operator+ (a1, a2);"
} // end function main
```

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.