MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

- **Kullanılacak kaynaklar:**
  http://www.cplusplus.com/doc/tutorial/
  Published by Juan Soulié

- **C++ ile ileri programlama**
  Paul Deitel
  Harvey Deitel

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

```
#include <iostream>
using namespace std;
class Date
{
public:
   Date( int = 1, int = 1, int = 2000 ); // default constructor
   void print();
private:
   int month;
   int day;
   int year;
}; // end class Date

Date::Date( int m, int d, int y )
{
   month = m;
   day = d;
   year = y;
} // end constructor Date

// print Date in the format mm/dd/yyyy
void Date::print()
{
   cout << month << '/' << day << '/' << year;
} // end function print
```

```
#include <iostream>
#include "Date.h" // include definition of class Date from Date.h
using namespace std;

int main()
{
   Date date1( 7, 4, 2004 );
   Date date2; // date2 defaults to 1/1/2000

   cout << "date1 = ";
   date1.print();
   cout << "\ndate2 = ";
   date2.print();

   date2 = date1; // default memberwise assignment

   cout << "\n\nAfter default memberwise assignment, date2 = ";
   date2.print();
   cout << endl;
} // end main
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Date.h

```
#include <iostream>
using namespace std;
class Date
{
public:
   static const int monthsPerYear = 12; // number of months in a
year
   Date( int = 1, int = 1, int = 1900 ); // default constructor
   void print() const; // print date in month/day/year format
   ~Date(); // provided to confirm destruction order
private:
   int month; // 1-12 (January-December)
   int day; // 1-31 based on month
   int year; // any year
   // utility function to check if day is proper for month and year
   int checkDay( int ) const;
}; // end class Date// constructor confirms proper value for month;
calls
// utility function checkDay to confirm proper value for day
Date::Date( int mn, int dy, int yr )
{
   if ( mn > 0 && mn <= monthsPerYear ) // validate the month
      month = mn;
   else
   {
      month = 1; // invalid month set to 1
      cout << "Invalid month (" << mn << ") set to 1.\n";
   } // end else
   year = yr; // could validate yr
   day = checkDay( dy ); // validate the day
   // output Date object to show when its constructor is called
   cout << "Date object constructor for date ";
   print();
   cout << endl;
} // end Date constructor
// print Date object in form month/day/year
void Date::print() const
{
   cout << month << '/' << day << '/' << year;
} // end function print
// output Date object to show when its destructor is called
Date::~Date()
{
   cout << "Date object destructor for date ";
   print();
   cout << endl;
} // end ~Date destructor

// utility function to confirm proper day value based on
// month and year; handles leap years, too
int Date::checkDay( int testDay ) const
{
   static const int daysPerMonth[ monthsPerYear + 1 ] =
      { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

   // determine whether testDay is valid for specified month
   if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
      return testDay;

   // February 29 check for leap year
   if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
      ( year % 4 == 0 && year % 100 != 0 ) ) )
      return testDay;

   cout << "Invalid day (" << testDay << ") set to 1.\n";
   return 1; // leave object in consistent state if bad value
} // end function checkDay
```

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Employee.h

```
#include <string>
#include "Date.h" // include Date class definition
#include <iostream>
using namespace std;

class Employee
{
public:
   Employee( const string &, const string &,
      const Date &, const Date & );
   void print() const;
   ~Employee(); // provided to confirm destruction order
private:
   string firstName; // composition: member object
   string lastName; // composition: member object
   const Date birthDate; // composition: member object
   const Date hireDate; // composition: member object
}; // end class Employee
```

```
// constructor uses member initializer list to pass initializer
// values to constructors of member objects
Employee::Employee( const string &first, const string &last,
   const Date &dateOfBirth, const Date &dateOfHire )
   : firstName( first ), // initialize firstName
     lastName( last ), // initialize lastName
     birthDate( dateOfBirth ), // initialize birthDate
     hireDate( dateOfHire ) // initialize hireDate
{
   // output Employee object to show when constructor is called
   cout << "Employee object constructor: "
      << firstName << ' ' << lastName << endl;
} // end Employee constructor

// print Employee object
void Employee::print() const
{
   cout << lastName << ", " << firstName << "  Hired: ";
   hireDate.print();
   cout << "  Birthday: ";
   birthDate.print();
   cout << endl;
} // end function print

// output Employee object to show when its destructor is called
Employee::~Employee()
{
   cout << "Employee object destructor: "
      << lastName << ", " << firstName << endl;
} // end ~Employee destructor
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

```
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

int main()
{
   Date birth( 7, 24, 1949 );
   Date hire( 3, 12, 1988 );
   Employee manager( "Temel", "Temeloglu", birth, hire );

   cout << endl;
   manager.print();

   cout << "\nTest Date constructor with invalid values:\n";
   Date lastDayOff( 14, 35, 1994 ); // invalid month and day
   cout << endl;
} // end main
```

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Overloading operators

Sınıflar aslında C++ kodlarında yeni tipte değişkenler tanımlar. C++ ta Tipler sadece atamalar ve kurucular vasıtasıyla degil operatorler vasıtasıyla da kodlariçinde etkileşmeye girer. Örnek olarak aşağıdaki temel operasyonu alalım.

```
int a, b, c;
a = b + c;
```

Bu örnekte int tipindeki değişkenlerin birbiri ile toplanması ve birbirlerine atanması görülmektedir. Burda toplama operatörünün Yapacağı işlem açıktır.

```
struct myclass {
  string product;
  float price;
} a, b, c;
a = b + c;
```

Bu örnekte ise toplama operatörünün b ve c üzerindeki etkisi belirsizdir. Ve bu islem derleme sırasında hata verecektir. C++ operatörlerin sınıflar ve butun tipler icin overload(aşırı yüklenmesine) izin vermektedir.

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

Overloading için kullanım şekli:
type operator sign (parameters) { /*... body ...*/ }

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

```
// overloading operators örneği
#include <iostream>
using namespace std;
class CVector {
  public:
    int x,y;
    CVector () {};
    CVector (int a,int b) : x(a), y(b) {}
    CVector operator + (const CVector&);
};
CVector CVector::operator+ (const CVector& param) {
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return temp;
}
int main () {
  CVector foo (3,1);
  CVector bar (1,2);
  CVector result;
  result = foo + bar;
  cout << result.x << ',' << result.y << '\n';
  return 0;
}
```

```
 //  CVector isimlifonksiyon(constructor)
CVector (int, int) : x(a), y(b) {}
//  CVector  tipinde sonuç döndüren fonksiyon
CVector operator+ (const Cvector&);
```

Cvector için Aşırı yüklenmiş + operatörü iki şekilde kullanılabilir:

```
c = a + b;
c = a.operator+ (b);
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Expression	Operator	Member function	Non-member function
@a	+ - * & ! ~ ++ --	A::operator@()	operator@(A)
a@	++ --	A::operator@(int)	operator@(A,int)
a@b	+ - * / % ^ & | < > == != <= >= << >> && || ,	A::operator@(B)	operator@(A,B)
a@b	= += -= *= /= %= ^= &= |= <<= >>= []	A::operator@(B)	-
a(b,c...)	()	A::operator()(B,C...)	-
a->b	->	A::operator->()	-
(TYPE) a	TYPE	A::operator TYPE()	-

```
// non-member operator overloads
#include <iostream>
using namespace std;
class CVector {
  public:
    int x,y;
    CVector () {}
    CVector (int a, int b) : x(a), y(b) {}
};
CVector operator+ (const CVector& lhs, const CVector& rhs) {
  CVector temp;
  temp.x = lhs.x + rhs.x;
  temp.y = lhs.y + rhs.y;
  return temp;
}
int main () {
  CVector foo (3,1);
  CVector bar (1,2);
  CVector result;
  result = foo + bar;
  cout << result.x << ',' << result.y << '\n';
  return 0;
}
```

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

this

Anahtar kelimesi

```
//  this anahtar kelimesi için örnek
#include <iostream>
using namespace std;
class Dummy {
  public:
    bool isitme (Dummy& param);
};
bool Dummy::isitme (Dummy& param)
{
  if (&param == this) return true;
  else return false;
}
int main () {
  Dummy a;
  Dummy* b = &a;
  if ( b->isitme(a) )
    cout << "Evet, &a == b\n";
  return 0;
}
```

This anahtar kelimesi program içerisinde fonksiyonu çalıştırılan sınıfın işaretçisini çağırır. This genellikle opertör fonksiyonlarda da kullanılır

```
CVector& CVector::operator= (const
CVector& param)
{
  x=param.x;
  y=param.y;
  return *this;
}
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Static Üyeler:

Bir sınıf için Statik üyeler o sınıflın bütün üyeleri için ortak olan üyelerdir

```
// Sınıflarda static üyeler
#include <iostream>
using namespace std;
class Dummy {
  public:
    static int n;
    Dummy () { n++; };
};
int Dummy::n=0;
int main () {
  Dummy a;
  Dummy b[5];
  cout << a.n << '\n';
  Dummy * c = new Dummy;
  cout << Dummy::n << '\n';
  delete c;
  return 0;
}
```

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Const üye fonksiyonlar

```
const MyClass myobject;
```

Eğer bir sınıf yukardaki gibi const olarak tanımlanmışsa elemanlarına sadece okuma hakkı ile erişilebilir. Değişiklik yapılamaz

```
// constructor on const object
#include <iostream>
using namespace std;
class MyClass {
  public:
    int x;
    MyClass(int val) : x(val) {}
    int get() {return x;}
};
int main() {
  const MyClass foo(10);
// foo.x = 20;            // Geçerli değil: x cannot be
modified
  cout << foo.x << '\n';  // ok: data member x can be read
  return 0;
}

Bir sınıf bileşenini const olarak tanımlamak:
int get() const {return x;}


int get() const {return x;}        // const üye fonksiyon
const int& get() {return x;}       // bir const& döndüren üye  fonksiyon
const int& get() const {return x;} // bir const& döndüren const üye  fonksiyon
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 3
MSGSU Fizik Bölümü
Ferhat ÖZOK

Örnek

```
// const objects
#include <iostream>
using namespace std;
class MyClass {
    int x;
  public:
    MyClass(int val) : x(val) {}
    const int& get() const {return x;}
};
void print (const MyClass& arg) {
  cout << arg.get() << '\n';
}
int main() {
  MyClass foo (10);
  print(foo);
  return 0;
}
```