C++ ‘a Giriş 4

Sınıflar

Nesneye Yönelik Programlama

(Object Oriented Programming)

- Sınıflar(classes)
  - Genel
  - Başlangıç ve bitiş fonksiyonları
  - Overloading başlangıç fonksiyonları
  - Default başlangıç fonksiyonları
  - Sınıf olarak göstericiler
  - Struct ve union ile sınıf tanımlama
  - Overloading operatörleri
  - “this” anahtar kelimesi(the keyword this)
  - Durağan üyeler(static members)

Genel

- C++'a özgü yapılardır
- Önce bir sınıf bildirimi yapmak gerekir
- Eğer access specifier tanımlanmaz ise otomatik olarak private kabul edilir

```
class [sınıf_ismi] {
  [private:]     //aynı ve ya arkadaş sınıfın üyeleri
                 Üye1;
  [protected:] //aynı,arkadaş,türetilmiş sınıf üyeleri
                 Üye2;
  [public:]      //object'in göründüğü heryer
                 Üye3;
};
```

Genel

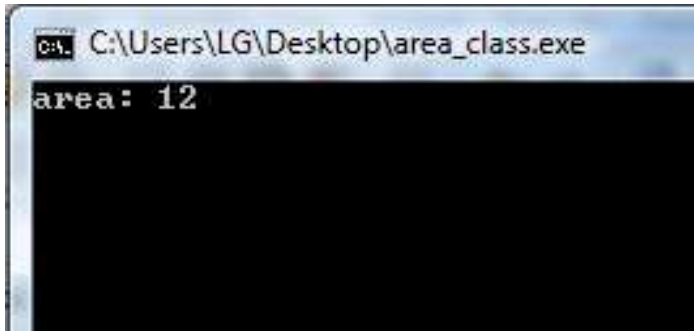
- Örnek:

```
class CRectangle {     //CRectangle : ismi
int x, y;              //Private Access'li üyeler.
public:                //Access specifier
void set_values (int,int);  //public access'li üye fonksiyon
int area (void);            // public access'li üye fonksiyon
    } rect;             // rect : object
```

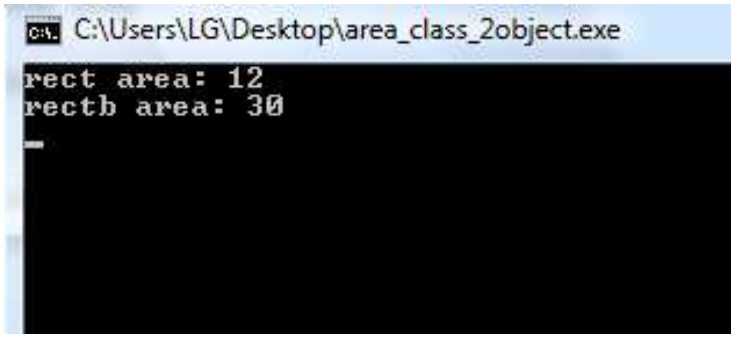
- CRectangle sınıfının 4 üyesi var
- Nasıl ki bir fonksiyonun türü oluyordu ; sınıfların isimleri de objelerinin türü oluyor.
- Programın gövdesinde rect nesnesinin public üyelerini normal bir fonksiyon ve ya değişken gibi çağırabilirim.

```
rect.set_values (3,4);
myarea = rect.area();
```

```cpp
// classes example
#include <iostream>

using namespace std;

class CRectangle {
int x, y;
public:
void set_values (int,int);
int area () {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
x = a;
y = b;
}

int main () {
CRectangle rect;
rect.set_values (3,4);
cout << "area: " << rect.area();
return 0;
}
```



```cpp
// example: one class, two objects
#include <iostream>

using namespace std;

class CRectangle {
int x, y;
public:
void set_values (int,int);
int area () {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
x = a;
y = b;
}

int main () {
CRectangle rect, rectb;
rect.set_values (3,4);
rectb.set_values (5,6);
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;
return 0;
}
```



!!!Nesneye yönelik programlamanın en önemli özelliklerinden birini gördük :
veri ve fonksiyonlar objelerin üyeleridir.

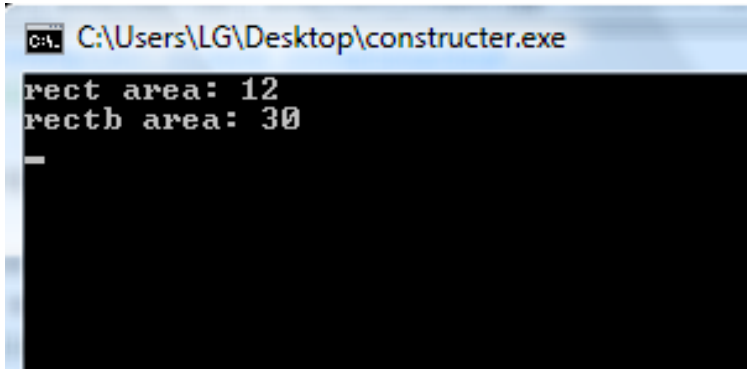
Başlangıç ve Bitiş fonksiyonları

(Constructers and destructors)

Başlangıç Fonksiyonları :

- Bir sınıf tanımlandığında derleyici tarafından otomatik olarak çağırılan fonksiyona sınıfın başlangıç fonksiyonu denir.
- Yerel bir sınıf nesnesi, programın akışı tanımlama noktasına geldiğinde, global bir sınıf nesnesiyse program belleğe yüklenir yüklenmez yaratılır.
- Başlangıç fonksiyonunun ismi sınıf ismiyle aynı olmalıdır.
- Başlangıç fonksiyonlarının geri dönüş değeri gibi bir kavramı yoktur. Yani geri dönüş türü yerine bir şey yazılmaz. Bu durum int ya da void anlamına gelmez.
- Başlangış fonksiyonları içerisinde return anahtar sözcüğü kullanılabilir, ancak yanına bir ifade yazılamaz. C++'ta farklı parametre yapısına sahip birden fazla başlangıç fonksiyonu olabilir.

```cpp
// example: class constructor
#include <iostream>

using namespace std;

class CRectangle {
int width, height;
public:
CRectangle (int,int);
int area () {return (width*height);}
};

CRectangle::CRectangle (int a, int b) {
width = a;
height = b;
}

int main () {
CRectangle rect (3,4);
CRectangle rectb (5,6);
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;

return 0;
}
```



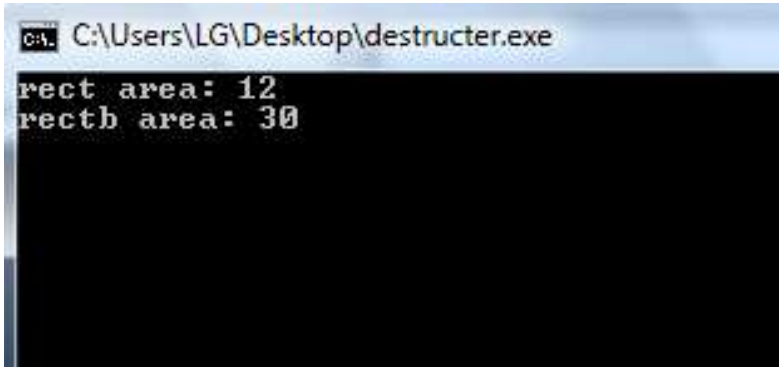
Başlangıç ve Bitiş fonksiyonları

(Constructers and destructors)

Bitiş Fonksiyonları :

- Bir nesne faaliyet alanını bitirmesiyle bellekten silinir. Yerel değişkenler programın akışında tanımlandıkları bloğun sonunda, global değişkenler ise programın bitimiyle bellekten silinirler.
- Bir sınıf nesnesi bellekten silineceği zaman otomatik olarak çağırılan fonksiyona bitiş fonksiyonu denir.
- Bitiş fonksiyonunun ismi sınıf ismiyle aynıdır, ancak başına bir ~ sembolü getirilir.
- Bitiş fonksiyonunun da geri dönüş değeri gibi bir kavramı yoktur. Bitiş fonksiyonu en az ve en fazla bir tane olabilir.
- Parametresi void olmak zorundadır.
- Global bir sınıf nesnesine ait bitiş fonksiyonu programın sonucunda main bittikten sonra yani main'in sonunda çalıştırılır.

```cpp
// example on constructors and destructors
#include <iostream>

using namespace std;

class CRectangle {
int *width, *height;
public:
CRectangle (int,int);
~CRectangle ();
int area () {return (*width * *height);}
};

CRectangle::CRectangle (int a, int b) {
width = new int;
height = new int;
*width = a;
*height = b;
}

CRectangle::~CRectangle () {
delete width;
delete height;
}

int main () {
CRectangle rect (3,4), rectb (5,6);
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;
return 0;
}
```



Overloading başlangıç fonksiyonları

- Başlangıç fonksiyonları c++ da diğir fonksiyonlar gibi aynı isim ile farklı fonksiyon çağırabilir, parametreleri de farklı olabilir, parametere sayısı da farklı olabilir.

```
// overloading class constructors
#include <iostream>

using namespace std;

class CRectangle {
int width, height;
public:
CRectangle ();
CRectangle (int,int);
int area (void) {return (width*height);}
};

CRectangle::CRectangle () {
width = 5;
height = 5;
}

CRectangle::CRectangle (int a, int b) {
width = a;
height = b;
}

int main () {
CRectangle rect (3,4);
CRectangle rectb;
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;
return 0;
}
```

CRectangle rectb; // right
CRectangle rectb(); // wrong!

Default başlangıç fonksiyonları

- Eğer sınıf tanımlarken bir başlangıç fonksiyonu çağırılmaz ise program bir default(olmayan) başlangıç fonksiyonu varmış gibi davranır.

```
class CExample {
public:                    //program otomatik olarak bir default başlangıç
int a,b,c;                 //fonksiyonu varmış gibi davranacak.
void multiply (int n, int m) { a=n; b=m; c=a*b; };
          };

CExample ex;          //"ex" CExample sınıfının objesi.ve bu yazım doğru.
```

- Eğer bir başlangıç fonksiyonu tanımladısak ileride çağırıken fonksiyonun prototipine uymalıyız.

```
class CExample {
public:
int a,b,c;
CExample (int n, int m) { a=n; b=m; };    //burada constructer tanımladık.
void multiply () { c=a*b; };
          };

CExample ex (2,3);     //main'in içinde artık tip'e uyarak çağıacağız.
```

Sınıf olarak göstericiler

- Bir sınıf'ı gösteren bir gösterici tanımlayabiliriz.

  Örnek:

  CRectangle * prect; //prect sınıfın bir objesinin göstericisidir.
- Bir gösterici atanan objenin üyesine refer etmek istiyorsak ok operatörünü kullanabiliriz (->)

  Örnek: Canlı göstereceğim ☺

expression	can be read as
*x	pointed by x
&x	address of x
x.y	member y of object x
x->y	member y of object pointed by x
(*x).y	member y of object pointed by x (equivalent to the previous one)
x[0]	first object pointed by x
x[1]	second object pointed by x
x[n]	(n+1)th object pointed by x

Struct ve union ile sınıf tanımlama

- Sınıfları class kelimesini kullanarak tanımlayacağımız gibi struct ve union kelimelerini kullanarak ta tanımlayabiliriz.
- Hatırlayacağınız gibi class ile tanımladığım bir sınıfta eğer belirtmez isem değişkenlerim private ulaşılabilirliğinde oluyordu. Struct ile tanımlamamın tek farkı bu değişkenler public ulaşılabilirliğinde oluyorlar.
- Union biraz daha farklı :
  Union türündeki sınıf içinde sadece 1 veri üyesi tutabiliyor. Ayrıca bu sınıfın da aksi belirtilmediği takdirde veri üyeleri public ulaşılabilirliğine sahip oluyor.

Overloading operatörleri

- Objeler üzerinde toplama çıkarma gibi işlemler yapabilmemi sağlıyor.
  Örnek :

```
struct {
string product;
float price;
} a, b, c;
a = b + c;      //böyle bir kodu direk yazarsam hata olur ama önce tanımlarsam olmaz.
```

Aşağıda istediğim işlemi overloading operatörü olarak tanımlayabilirim.

type operator sign (parameters) { /*...*/ }

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