12. SINIF VE NESNELER

Giriş

Bu bölümde sınıf ve nesne kavramlarını ele alacağız. Önce basit sınıflardan başlayarak giderek daha karmaşık sınıflar yazacak ve bu sınıflardan nesneler tanımlayacağız. Bölüm tamamlandığında nesne yönelimli programlama (***Object Oriented Programming***) kavramları ve teknikleri ile ilgili temel bilgileri öğrenmiş olacağız.

Nesne yönelimli programlamanın esası, belirli özelliklere ve yöntemlere sahip bir nesne oluşturmaktır. Nesneye yönelik programlamada (***Object Oriented Programming***) programı tasarlarken, tüm dünyayı nesne şeklinde görmeye çalışmalıyız. Örneğin araba, renk, kapı sayısı ve benzerleri gibi belirli özelliklere sahip bir nesnedir. Ayrıca, hızlanma, fren vb. gibi belirli işlevlere sahiptir. Nesneye yönelik programlamanın temelini oluşturan bazı kavramlar vardır. Bu kavramlar sınıf, nesne, soyutlama, Paketleme, kalıtım ve çok biçimlilik şeklinde sıralanabilir. ***Bu kavramlardan soyutlama, Paketleme, kalıtım ve çok biçimlilik nesneye yönelik programlama teorisinin dört temel özelliği olarak anılır.*** Konunun başında bu kavramları açıklamanın konun anlaşılmasına katkı sağlayacağı düşüncesiyle önce bu kavramların tanımlarını yapalım.

Nesne (Object)

Nesne, nesne yönelimli programlamanın en temel birimidir ve ait olduğu sınıfın bir örneğidir. Bu yaklaşımda hem veriler hem de veriler üzerinde çalışan işlevler nesne olarak adlandırılan bir birim olarak paketlenir.

Sınıf (Class)

Sınıf tanımlandığında, bir nesne için bir plan tanımlanır. Sınıfın tanımını yapmak aslında herhangi bir veri tanımlamaz, ancak sınıf adının ne anlama geldiğini, yani sınıftaki bir nesnenin ne içereceğini ve böyle bir nesne üzerinde hangi işlemlerin gerçekleştirilebileceğini tanımlar.

Soyutlama (Abstraction)

Veri soyutlama, dış dünyaya sadece gerekli bilgileri sağlamak ve arka plan ayrıntılarını gizlemek, yani ayrıntıları sunmadan programdaki gerekli bilgileri temsil etmek anlamına geldiğini de söyleyebiliriz.

Örneğin, bir veritabanı sistemi, verilerin nasıl saklandığına, nasıl oluşturulduğuna ve nasıl korunduğuna ilişkin belirli ayrıntıları gizler. Benzer şekilde, C ++ sınıfları bu yöntemler ve veriler hakkında iç ayrıntı vermeden dış dünyaya farklı yöntemler sunar.

Paketleme (Encapsulation)

Paketleme, verileri ve bu veriler (değişkenler) üzerinde çalışan işlevleri (fonksiyonlar) aynı isim altında tanımlamak, paketlemek anlamına gelir. Prosedürel dillerle çalışırken, hangi işlevlerin hangi değişkenler üzerinde çalıştığı her zaman açık değildir, ancak nesne yönelimli programlama size verileri ve ilgili işlevleri aynı nesneye yerleştirmek için önemli bir çerçeve sağlar.

Nesneye yönelik programlama nesnelerin özelliklerini (değişkenleri) ve işlevlerini (fonksiyonlarını – metotlarını) diğer nesnelerden saklanmasını sağlar. Bu, özellik ve işlevleri public (açık), private (özel) ve protected (korumalı) şeklinde tanımlayarak gerçekleştirilir.

- Public (genel) : Özelliklere ve işlevlere her nesneden erişilebilir.
- Private (özel): Özelliklere ve işlevlere yalnızca işlevin tanımlandığı sınıfa ait nesnelerle erişilebilir.
- Protected (korumalı): Özelliklere ve işlevlere yalnızca metodun tanımlandığı sınıfa ait nesnelerle erişilebilir ve bu sınıftan türetilmiş alt nesnelerinden erişilebilir.

Kalıtım (Inheritance)

Nesneye yönelik programlamanın en kullanışlı yönlerinden biri de kodların yeniden kullanılabilir olmasıdır. Adından da anlaşılacağı gibi kalıtım, temel sınıf olarak adlandırılan mevcut bir sınıftan yeni bir sınıf oluşturma işlemidir. Bu, nesne yönelimli programlama için çok önemli bir kavramdır, çünkü bu özellik kod boyutunu azaltmaya, kodun okunabilir olmasına önemli katkılar sağlar.

Çok Biçimlilik (Polymorphism)

Nesneye yönelik programlamada, oluşturulan nesnelerin gerektiğinde başka bir nesne gibi davranabilme özelliğine çok biçimlilik denmektedir. Çok biçimlilik kavramı için nesnelerin aynı mesaja farklı cevap vereme özelliğidir de diyebiliriz.

Nesneye yönelik programlama yaklaşımı günümüzde düzenli kod yazmak, kod bloklarına daha çabuk erişim sağlamak, kod tekrarını önlemek, güvenlik düzeyi daha yüksek programlar yazılmasını sağlamak ve vb. nedenlerle, özellikle büyük ölçekli projelerde kullanmaktan kaçmayacağımız bir tekniktir. Başlangıçta sadece C++ gibi, Java gibi programlama dilleri tarafından desteklenen nesneye yönelik programlama yaklaşımı, günümüzde hemen bütün programlama dilleri tarafından desteklenmektedir. Bu diller, yukarıda sayılan özellikleri ile programcılara daha sonradan bakımını daha düşük maliyetlerle yapılabilecekleri (geliştirilebilir), daha az zamanda tamamlayabilecekleri, güvenlik düzeyi yüksek, karmaşıklığı daha az ve bilgisayarın fiziksel kaynaklarını daha verimli kullanan projeler yapmalarını sağlayacak programlama ortamları sunmaktadır. Nesneye yönelik programlama yaklaşımı diğer bir yönüyle kendisinden önce kullanılan yapısal programlama yaklaşımını ve başlangıçtan günümüze kullanılan daha önceki programlama yaklaşımlarını da kapsayan bir yaklaşımdır.

Nesneye Yönelik programlama yaklaşımından önce yazılım projelerinde yapısal programlama yaklaşımı kullanılmaktaydı (Bu bölüme kadar biz de örneklerimizi yapısal programlama tekniklerini kullanarak oluşturduk). C, Pascal ve benzeri prosedürel dillerin desteklediği yapısal programlama yaklaşımında, bir program dosyası aslında bir komut listesi olarak hazırlanmaktaydı. Zaman içerisinde bu listeler, elektronik ortama taşınacak işlerin hacminin artmasıyla çok fazla uzadı ve sonradan ortaya çıkan bakım maliyetleri katlanılamaz boyutlara ulaştı. Daha sonra yine aynı yapısal programlama yaklaşımı içerisinde kalınarak program içerisindeki tekrarlardan kurtulmak, programı bilgisayar ortamında koştururken bilgisayarın fiziksel bileşenlerini daha verimli kullanabilmek gibi nedenlerle programlar fonksiyonlar halinde yazılmaya başlandı. Ancak, yine de eklenen bu önemli yeni özelliğine rağmen yapısal programlama yaklaşımı gerçek hayat problemlerini temsil etmekte çok başarılı olamadı ve de en önemlisi yapısal programlama yaklaşımı ile geliştirilen yazılım projelerinde programcılar güvenli projeler oluşturduklarından hiçbir zaman emin olamadı. Yukarıda sayılan bu ve benzeri eksikliklerine rağmen yapısal programlama yaklaşımı, kendi döneminde önemli yazılım projelerinde kullanılarak programlama dilleri tarihi içerisinde yerini almıştır.

Nesneye yönelik programlama yaklaşımında sınıflar, aslında yapısal programlamadan miras aldığı fonksiyonlar (Nesneye yönelik programlamada **metot** olarak anılacaktır.) ile verileri (özellikleri) bir çatı altında toplar. Daha doğru bir ifadeyle sınıflar, metotlar ile özellikleri bir çatı altında toplayan tanımlamalardır.

C++ ve nesneye yönelik benzer programlama dillerini kullanan programcılar, sınıf adı verilen kendilerine ait kullanıcı tanımlı tiplerini oluşturur. Programcı tarafından oluşturulan sınıflar veriyi yöneten metotların kümesi ve verilerden oluşur. Daha sonra program içerisinde oluşturulan bu sınıfa ait nesneler oluşturulur.

Bu bölümde yukarıda da belirtildiği gibi C++ programlama dilini kullanarak sınıfların ve bu sınıflara ait nesnelerin tanımlanmasını program içerisinde bu yaklaşımın nasıl kullanılacağını öğreneceksiniz.

12.1. Sınıf ve Nesnelerin Tanımlanması ve Kullanılması

12.1.1. Sınıf ve Nesnelerin Tanımlanması

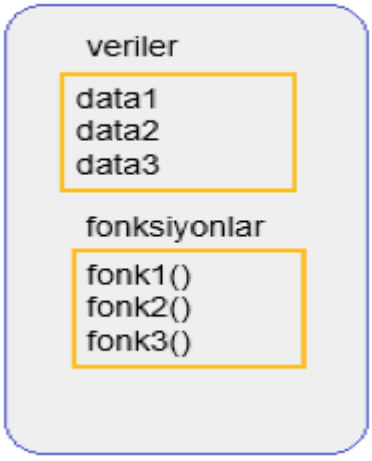
Aşağıda kaynak kodu verilen C++ programında, bir sınıf ve bir de nesne tanımlanmıştır. Bu program basit olmakla birlikte, C++'ta sınıflar ve nesneler hakkında birçok bilgi içermektedir. Programda ***Kare*** isimli bir sınıf tanımlanmış ve Kare isimli bu sınıfta tek bir veri öğesi yer almıştır. Bundan başka sınıf içerisinde üç tane fonksiyon (metot) yazılmıştır. Fonksiyonlardan ilki, veri öğesine bir değer atamak için, ikinci fonksiyon karenin çevresini yazdırmak için ve üçüncü fonksiyon da karenin alanı yazdırmak için yazılmıştır.

İpucu: Veri ve fonksiyonları tek bir bütün halinde ele almak nesne yönelimli programlamanın en önemli özelliğidir.

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

class Kare
{
    private:
        int karekenar;

    public:
        void KenarUzunlukata(int sayi)
            { karekenar = sayi; }

        void cevreyaz()
            { cout << 4*karekenar << endl; }
        void alanyaz()
        {
            cout<<karekenar*karekenar<<endl;}
    };

int main()
{
    Kare kare1;
    kare1.KenarUzunlukata(3);
    kare1.cevreyaz();
    kare1.alanyaz();

    return 0;
}
```

Program 12.1 Karenin Çevresi ve Alanı

Veri ve fonksiyonları tek bir bütün haline getirmek nesne yönelimli programlamanın temel yaklaşımıdır.



Veri ve fonksiyonların sınıf içerisinde gösterilmesi

Yukarıdaki programda, ilk bölümde ***Kare*** isimli bir sınıf tanımlanmış daha sonra ***main()*** bölümünde Kare ***sınıfına*** ait ***kare1*** isimli bir örnek(nesne) tanımlanmıştır.

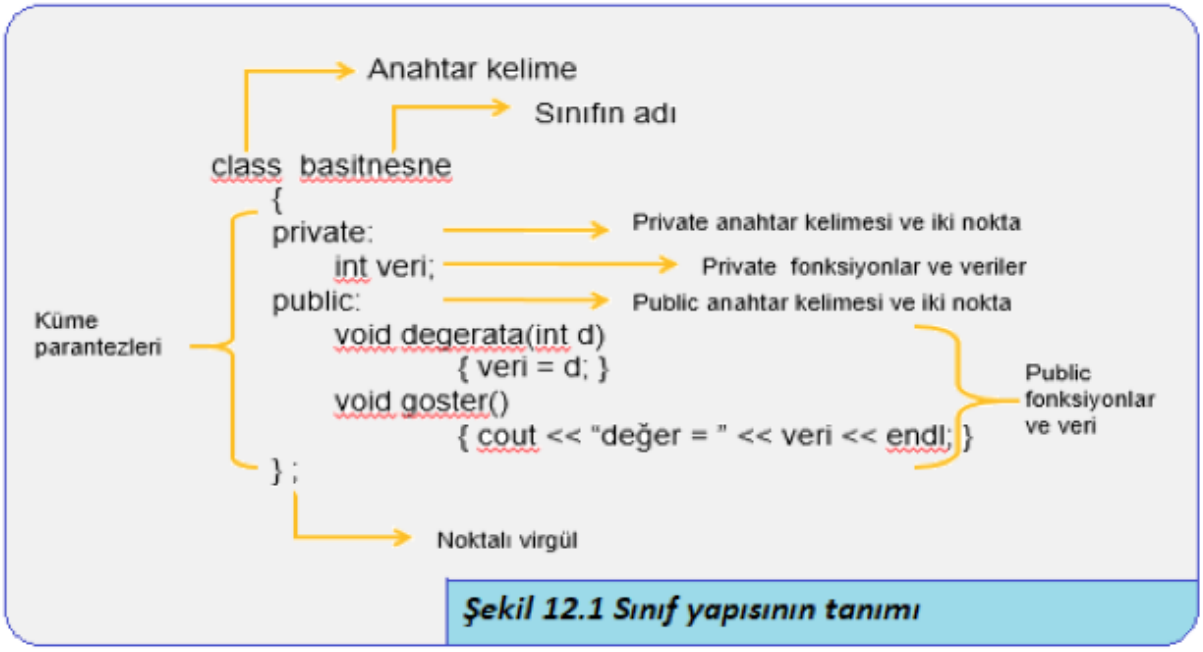
Program 12.1'in kaynak kodundan alınan sınıf tanımlama bölümü, aşağıda gösterilmiştir. C++ programlarında sınıf tanımı **class** bildirimi ile başlar ve bildirimden sonra sınıfın adı **(Aşağıdaki örnekte bu ad, *Kare* olarak belirlenmiştir.)** yazılır. Daha sonra sınıf bloğunda yapılacak tanımlamalar, aşağıda da gösterildiği gibi, küme parantezleri arasına yazılır. Kapanan küme parantezi sonunda noktalı virgül konulur.

```
class Kare
{
    private:
        int karekenar;

    public:
        void KenarUzunlukata(int sayi)
            { karekenar = sayi; }

        void cevreyaz()
            { cout << 4*karekenar << endl; }
        void alanyaz()
        {
            cout<<karekenar*karekenar<<endl;}
    };
```

Program 12.1'den Alınan Sınıf Tanımı Bölümü

İpucu: Sınıf içerisinde tanımlanan verilere ve fonksiyonlara o sınıfın üyeleri(elemanları) denilir.

Program 12.1'e ait sınıf tanımından da görülebileceği gibi, nesneye yönelik programların gövdesinde ilki ***private***(özel), ikincisi ***public***(genel) olmak üzere, daha önceki bölümlerde kullanmadığımız iki yeni bildirim kullanılmıştır. Nesneye yönelik programlamanın en temel özelliklerinde birisi verinin saklanmasıdır. Nesneye yönelik programlamada verinin saklanması, veriyi sınıf içerisinde ***private*** olarak tanımlayarak gerçekleştirilir. Bu şekilde tanımlanan verilere, sadece o sınıfın fonksiyonları erişebilir. Dolayısı ile veriler, diğer sınıflara ait fonksiyonlar tarafından erişilip değiştirilemez ve veriler güvenlik altına alınmış olur. Verilerin güvenlik altına alınması, programın içerisinde o verilere ulaşması gerekmeyen bölümlerden ulaşılmasını önlemek anlamına gelmektedir.



Şekil 12.1 Sınıf yapısının tanımı

Veriler ve fonksiyonlar sınıf gövdesinde private olarak tanımlanabildiği gibi public olarak da tanımlanabilir. Bu şekilde tanımlanan veri ve fonksiyonlara programın her yerinden ulaşılabilir.

Program 12.1'de ***int*** tipinde tek bir veri öğesi tanımlanmıştır. Ancak, bir sınıfta istenildiği kadar veri öğesi tanımlanabilir. Sınıf içerisinde tanımlanan veri öğelerine ***üye veriler*** veya ***veri üyeleri*** denilir.

İpucu: Sınıf gövdesinde private bildirimi içerisinde verilerin tanımı yapılabildiği gibi, ihtiyaç duyulması halinde bu bildirim altında fonksiyonların tanımlamaları da yapılabilir.

İpucu: *private* veri veya fonksiyon tanımlanması gerekiyorsa *private* yazmaya gerek yoktur. Sınıf adından sonra {} içine direk veri ve fonksiyon yazıldığında bu tanımlamalar otomatik olarak private kabul edilir.

Bir sınıfın içerisinde yer alan fonksiyonlara o sınıfın ***üye fonksiyonları*** denilir. Bazen de fonksiyonlar **metot** olarak ta isimlendirilir. Program 4.1'de Kare sınıfının üç üye fonksiyonu(metot) vardır. Bu üye fonksiyonlar aşağıda gösterilmiştir.

```
public:
    void KenarUzunlukata(int sayi)
        { karekenar = sayi; }

    void cevreyaz()
        { cout << 4*karekenar << endl; }
    void alanyaz()
    {
        cout<<karekenar*karekenar<<endl;}
```

Program 4.1'deki üye fonksiyonlar

Tanımlamaya yakından bakıldığında, üye fonksiyonların(metotlar) **public** anahtar kelimesi altında yazıldığı görülmektedir. Genellikle bir sınıf tanımlanırken veriler **private** altında, üye fonksiyonlar da public altında yazılır. Fakat bu bir kural değildir. Bazen üye veriler **public** altında, üye fonksiyonlar da **private** altında tanımlanabilir.

İpucu: Üye veriler, yanlışlıkla değiştirilmemeleri için, güvenlik nedeniyle private altında tanımlanır. Üye fonksiyonlar da, üye veriler üzerinde işlem yaptıklarından, sınıf dışından erişilebilmeleri için public olarak tanımlanırlar.

12.1.2. Tanımlanan Sınıfın Kullanılması

Bir sınıf tanımlamak yapı tanımlamak gibidir. Tanımlanan sınıf, kendisine ait bir nesne oluşturulana kadar sadece oluşturulacak nesnenin neye benzeyeceğini tanımlar. Karenin çevresini ve alanını hesaplayan programda main() fonksiyonu içerisindeki ilk ifadede Kare sınıfına ait bir nesne tanımlanmıştır:

Kare kare1;

İfadesiyle programda bir nesne tanımlanmıştır. Bu işleme örnek oluşturmak denir. Nesnelere bazen örnek değişken(***instance variable***) adı da verilebilir.

12.1.3. Üye Fonksiyonların Çağırılması

Üye fonksiyonlar(metotlar) bir nesne üzerinde işlem yapmak için çağırılır. Başka bir şekilde ifade etmek gerekirse, sınıfın üye fonksiyonlarına sadece o sınıfa ait nesneler tarafından erişilir. Sınıfa ait nesne ile yine o sınıfa ait bir üye fonksiyonu çağırmak için, nesne ile üye fonksiyonu ***nokta operatörü*** ile ilişkilendirmek gerekir. Nokta operatörüne ***sınıf üyesi erişim operatörü*** de denir. Ele aldığımız örnekte, üç üye fonksiyon da tanımlanan nesne tarafından aşağıdaki şekilde çağırıldığını görüyoruz:

kare1.KenarUzunlukata(3);

Programda çalışma sırası yukarıdaki ifadeye geldiğinde, *kare1* nesnesinin *KenarUzunlukata()* üye fonksiyonu çalışır. Bu fonksiyon kenar1 değişkeninin değerini üç(3) yapar.

İkinci çağrıda;

kare1.çevreyaz();
kare1 nesnesinin çevresi ekrana yazılır.

Üçüncü çağrıda;

kare1.alanyaz();
kare1 nesnesinin alanı ekrana yazılır.

> İpucu: Nesne yönelimli dillerde üye fonksiyonları çağırma işlemine ***mesaj*** denir. Bu nedenle ***kare1.alanyaz();*** *çağrısı* ***kare1****'e, verisini göndermesini bildiren bir* ***mesaj*** gönderimi *olarak düşünülebilir.*

12.1.4. Program 12.2

Bir Kare sınıfı tanımlayınız. Kare sınıfında, sınıf elemanlarının tamamı public olarak tanımlanmış olsun. Sınıfa ait bir nesne oluşturunuz ve klavyeden nesneye kenar uzunluk ölçüsünün girişinin yapılmasını sağlayınız. Program çalıştırıldığında karenin çevresini ve alanını ekrana yazdırınız?

Çözüm:

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

class kare
{
    public:
        int kareKenar;
    int cevre(){

    return 4*kareKenar;}

    int alan(){
    return kareKenar*kareKenar;}

};x;
int main() {

    cin>>x.kareKenar;
    cout<<"Çevre    :"<<x.cevre()<<endl;
    cout<<"Alan     :"<<x.alan()<<endl;
    return 0;
}
```

Program 12.2 Üye fonksiyonların ve üye verilerin public olduğu C++ programı

Program 12.2'de üye veriler ve üye fonksiyonlar ***public*** olarak tanımlanmıştır. Bundan dolayı sınıfın bütün elemanlarına dışarıdan erişilebilmektedir. Biz de üye veriye(***KareKenar***) public olarak tanımlandığı için,

tanımlanan nesne üzerinden, dışarıdan(***main() içerisinden***) doğrudan erişilebildik. Bu bir kolaylık gibi görünse de verinin güvenliğini tehdit edici bir yaklaşımdır.

Bundan başka, Program 12.2'de nesnenin tanımlandığı yere de dikkat ediniz. Program 12.1'de nesne ***main()*** fonksiyonunda tanımlanmıştır. Program 12.2'de ise bu tanımlama, sınıf gövdesinin bittiği noktada yapılmıştır. Bu iki yaklaşım da doğrudur. C++'ta Nesnelerin tanımı, sınıfın tanımlandığı yerde veya ***main()*** fonksiyonu içerisinde yapılabilir.

12.1.5. Program 12.3

Bilgisayara stok kodu, ürün adı, ürünün birim fiyatı girişinin klavyeden yapıldığı ve girilen verilerin ekranda gösterildiği programı nesneye yönelik programlama yaklaşımı ile yazınız.

Çözüm:

```
#include <iostream>
#include <string>
using namespace std;
class Stok  //sinifi tanimla
    {
    private:
        int stokKodu;   //Stok kodu
        string urunAdi;     //Ürün adı
        float brimFiyat;    //Birim fiyatı
    public:
        void stokAta(int stk, string uad, float bf) //deger atama fonsiyonu
        {
        stokKodu = stk;
        urunAdi = uad;
        brimFiyat = bf;
        }
        void goster()   //veriyi görüntüleyen fonksiyon
        {
        cout << "Stok Kodu   : " << stokKodu<<endl;
        cout << "Urun Adi    : " << urunAdi<<endl;
        cout << "Birim Fiyat : " << brimFiyat << endl;
        }
    };
int main()
    {
    Stok  stok1; string ua; int sn; float bf;   //Stok sinifina ait nesneyi tanimla
    cout<<"stok kodu   : ";cin>>sn;
    cout<<"Urun adi    : ";cin>>ua;
    cout<<"Birim Fiyat : ";cin>>bf;

    stok1.stokAta(sn, ua, bf);  //üye foksiyonu çagir
    cout<<"----------------------"<<endl;
    stok1.goster();          //üye foksiyonu çagir
    return 0;
    }
```

Program 12.3 Stok Programı

12.1.6. Program 12.4

Öğrencilerin öğrenci numarası, ad, soy ad ve yaşlarının bilgisayara girişinin klavyeden yapıldığı ve girilen verilerin ekranda görüntülendiği C++ programını yazınız?

Çözüm:

```
#include <iostream>
#include<string>
#include <locale.h>
using namespace std;
class ogrenci
{
string ogrNo,ogrAd,ogrSoyad;
int ogrYas;
public:
   void ogr(string oNo, string oAd, string oSad,int oYas)
   {
    ogrNo=oNo;
    ogrAd=oAd;
    ogrSoyad=oSad;
    ogrYas=oYas;
    }
    void goruntule()
    {
        cout<<"Öğrenci No    : "<<ogrNo<<endl;
        cout<<"Öğrenci Ad    : "<<ogrAd<<endl;
        cout<<"Öğrenci Soyad : "<<ogrNo<<endl;
        cout<<"ogrYas        : "<<ogrYas<<endl;
      }
};

int main() {
    setlocale(LC_ALL,"Turkish");
    ogrenci ogr1,ogr2;
    ogr1.ogr("00001","Ahmet","Kara",20);
    ogr2.ogr("00002","Ayşe","Ak",19);
    ogr1.goruntule();
    ogr2.goruntule();
    return 0;
}
```

Program 12.4 Sınıf ve Nesne Örneği_ Öğrenci Uygulaması

Sıra Sizde: Aşağıda, Program 12.4 için bir çözüm sunulmuştur. Fakat üye verilere değerleri, main() fonksiyonu içerisinde atama yoluyla gerçekleştirilmiştir. Programa ait soru metninde veri girişlerinin klavyeden yapılması istenmiştir. Program üzerinde veri girişlerinin klavyeden yapılabilmesi için gerekli değişiklikleri yapınız.

12.2. Kurucu Fonksiyonlar

C++'da nesnenin otomatik olarak ilk kullanıma hazırlama işlemi kurucu(constructor) fonksiyon tarafından gerçekleştirilir. Ne zaman bir nesne oluşturulursa kurucu fonksiyon otomatik olarak çalıştırılır.

Bir program içerisinde kullanılmak üzere bir sayaç sınıfı oluşturulduğunu düşünelim. Her zaman sayaçların ilk oluşturulduklarında sıfır değerini almasını isteriz.

Bunu sağlamak için sayaca ilk değeri atayan bir üye fonksiyon yazabiliriz ve sayaç nesnesi tanımlandığında bu üye fonksiyonu çağırarak sıfır değeri atayabiliriz. Ancak, her nesne oluşturulduğunda bu üye fonksiyonun da bir defa çalıştırılması gerekir. Bu işlem hataya açık bir durumdur ve bu nedenle sınıf içerisinde başlangıç değeri verme işlemi yapılmaz.

Ancak, kurucu fonksiyonları kullanarak üye değişkenlere ilk değer atanabilir. Her nesne oluşturulduğunda, otomatik olarak kurucu fonksiyonla kendisini ilk kullanıma hazırlaması daha uygun ve güvenilirdir.

Kurucu (Yapıcı) Fonksiyonların Özellikleri:

a. Kurucu fonksiyonlar üyesi olduğu sınıfla aynı isme sahip olmalıdır. Bu özellik derleyicinin fonksiyonun kurucu olduğunu anlamasına neden olur.
b. Sınıfa ait nesne oluşturulduğunda kurucu fonksiyon otomatik olarak çağırılır.
c. Bir sınıf içerisinde birden fazla kurucu fonksiyon tanımlanabilir. Bir sınıf içerisinde birden fazla kurucu fonksiyon tanımlanmışsa bu fonksiyonların isimleri aynı olmalıdır.
d. Kurucu fonksiyonlar değer döndürmezler.
e. Kurucu fonksiyonlar nesne oluşturulduğu zaman sadece bir kere çalıştırılırlar.
f. Eğer bir sınıfa ait kurucu fonksiyon tanımlanmaz ise derleyici tarafında otomatik oluşturulur.

12.2.1. Program 12.5

Bir C++ uygulaması geliştiriniz. Uygulamada üye değerlere kurucu fonksiyon tarafından ilk değer ataması yapılsın.

Çözüm:

```cpp
#include <iostream>
using namespace std;
class ucret
{
    int calismaSaati;
    int saatUcreti;
    int brut;
    public:
        ucret():calismaSaati(0),saatUcreti(0),brut(0)
        {}
        void goster( )
        {
            cout<<calismaSaati<<endl;
            cout<<saatUcreti<<endl;
            cout<<brut<<endl;
        }
};

int main() {

    ucret prsl;
    prsl.goster();
    return 0;
}
```

Program 12.5 Kurucu Fonksiyon örneği

```
Calisma Sati : 0
Saat Ucreti  : 0
Brut Ucret   : 0
```

Program 12.5'in Ekran Çıktısı

İlk değer atama işlemi üye fonksiyon bildiriminden sonra, fakat fonksiyon gövdesinden önce yapılır. Atanacak değerden önce iki nokta gelir. Değer üye verinin peşinden parantez içerisinde yazılır. Eğer birden fazla üyeye değer atanacaksa, bu değerler virgül ile ayrılır. Program 12.5'teki örnek kurucu fonksiyon tarafından birden fazla üyeye ilk değer ataması yapılmasına örnek olarak verilmiştir.

Sıra Sizde: Program 12.5'te aşağıdaki değişiklikleri yapıp programı tekrar çalıştırınız. Yukarıdaki programın ekran çıktıları ve aşağıdaki değişiklikleri yaptıktan sonra çalıştıracağınız programdan elde edilen ekran çıktılarını karşılaştırınız.

1. Kurucu fonksiyonda üye değerlere sıfır yerine 10, 20, 30 gibi farklı değerler atayınız ve programı tekrar çalıştırınız.
2. Kurucu fonksiyonu kaldırınız ve programı tekrar çalıştırınız?

12.2.2. Program 12.6

Bir C++ uygulaması geliştiriniz. Geliştirdiğiniz bu programda kurucu fonksiyon ile programın üye değişkeni sıfırlansın.

Çözüm:

```
#include <iostream>
using namespace std;
class sayac
    {
private:
    unsigned  int  sayici;
public:
 sayac():sayici(0)
        {    }
   void artir_sayici()
        {sayici++;}

   int  al_sayici()
        {return sayici;}
    };
int main()
    {
sayac  s1,s2;
cout<<"\n s1 degeri "<< s1.al_sayici();
cout<<"\n s2 degeri "<<s2.al_sayici();

s1.artir_sayici();
s2.artir_sayici();
s2.artir_sayici();

cout<<"\n s1 degeri "<< s1.al_sayici();
cout<<"\n s2 degeri "<< s2.al_sayici();
cout<<endl;
system("pause");
return 0;
    }
```

Program 12.6 Kurucu Fonksiyonla Üye değ. Sıfırlanması

```
s1 degeri 0
s2 degeri 0
s1 degeri 1
s2 degeri 2
```

Program 12.6 'nın Ekran çıktısı

12.3. Yıkıcı Fonksiyonlar

Yıkıcı Fonksiyonlar nesneler yok edileceği zaman çalışan fonksiyonlardır. Eğer programcı tarafından tanımlanmazsa otomatik olarak tanımlanırlar. Yıkıcı fonksiyonların da isimleri kurucu fonksiyonlarda olduğu gibi sınıf ismi ile aynı olmalıdır. Yıkıcı fonksiyonların isimlerinin başında, kurucu fonksiyonlardan farklı olarak ~(Tilda) işareti bulunur. Bir sınıf içerisinde sadece bir yıkıcı fonksiyon olabilir. Yıkıcı fonksiyonlar parametre almazlar ve değer döndürmezler.

Yıkıcı fonksiyonlar yaygın olarak, bir nesne için kurucu fonksiyonların ayırdığı bellek alanını iade etmek için kullanılır.

12.3.1 Program 12.7

İçerisinde yıkıcı fonksiyon bulunan bir C++ programı yazınız?

Çözüm:

```
using namespace std;
class yoket
    {
        int sayi;
        public:
                yoket();
                ~yoket();
                void goruntule();

    }sayil;

    yoket::yoket(){
        cout<<"kurucu fonksiyon calisti"<<endl;
        sayi=5;
    }
    yoket::~yoket(){
        cout<<"Yıkıcı fonksiyon calisti"<<endl;
            }
    void yoket::goruntule(){
        cout<<"sayi degeri : "<<sayi<<endl;
    }
int main() {
    sayil.goruntule();
    return 0;
}
```

Program 12.7 İçerisinde yıkıcı fonksiyon bulunan C++ programı

```
kurucu fonksiyon calisti
sayi degeri : 5
Yikici fonksiyon calisti
```

Program 12.7'nin Ekran Çıktısı

12.4. Bir Nesneyi Başka Bir Nesneye Atamak

Bazı problemlerin çözümünde kullanılmak üzere, bir sınıftan birden fazla nesne oluşturabiliriz ve bu nesnelerden birinin değerlerini, diğer başka nesnelere atamak isteyebiliriz. C++ ta bu işlem atama operatörü kullanılarak yapılmaktadır.

12.4.1. Problem 12.8

Nesneden nesneye atama işlemin Problem 12.3'ün sınıf yapısını değiştirmeden *main()* fonksiyonu içerisinde yeni bir nesne oluşturup(*stok2*), daha önceden oluşturulan stok1'in değerleri stok2'ye aktarılmıştır. Programın son kısmında da göster fonksiyonu yardımıyla her iki nesne de görüntülenmiş ve aynı değerlere sahip olduğu görülmüştür.

```
class Stok  //sinifi tanimla
    {
    private:
        int stokKodu;    //Stok kodu
        string urunAdi;      //Ürün adı
        float brimFiyat;     //Birim fiyatı
    public:
        void stokAta(int stk, string uad, float bf) //deger atama fonsiyonu
        {
        stokKodu = stk;
        urunAdi = uad;
        brimFiyat = bf;
        }
        void goster()    //veriyi görüntüleyen fonksiyon
        {
        cout << "Stok Kodu    : " << stokKodu<<endl;
        cout << "Urun Adi     : " << urunAdi<<endl;
        cout << "Birim Fiyat  : " << brimFiyat << endl;
        }
    };
int main()
    {
    string ua; int sn; float bf;
    Stok  stok1,stok2;  //Stok sinifina ait nesneyi tanimla
    cout<<"stok kodu    : ";cin>>sn;
    cout<<"Urun adi     : ";cin>>ua;
    cout<<"Birim Fiyat  : ";cin>>bf;
    stok1.stokAta(sn, ua, bf);  //üye foksiyonu çagir
    cout<<"----------------------"<<endl;
    stok1.goster();           //üye foksiyonu çagir
    stok2=stok1;
    cout<<"----------------------"<<endl;
    stok2.goster();
    return 0;    }
```

Program 12.8 Bir nesneyi Başka Bir nesneye Atama Örneği

```
Stok kodu    : 0423
Urun adi     : Bilgisayar
Birim Fiyat  : 5090
--------------------
Stok Kodu    : 423
Urun Adi     : Bilgisayar
Birim Fiyat  : 5090
--------------------
Stok Kodu    : 423
Urun Adi     : Bilgisayar
Birim Fiyat  : 5090
```

Program 14.8'in ekran çıktısı

12.5. const Fonksiyonlar ve const Nesneler

Daha önceki bölümlerde değişkenlerin program içerisinde değerlerinin değişmez olmasını sağlamak, programı veri açısından güvenlik düzeyini arttırmak için ***const*** bildirimini kullanmıştık. Nesneye yönelik programlamada da const nesneler ve const fonksiyonlar kullanarak aynı amaçları gerçekleştirebiliriz.

12.5.1. Problem 12.9

Bir C++ programı yazınız. Program içerisinde tanımlanan sınıfın içerisinde const bir üye fonksiyon bulunsun. Ayrıca, programda iki nesne tanımlayınız. Tanımlanan nesnelerin birincisi const diğeri const olmasın. Programın aşağıdaki ihtimallerden hangisinde hata verdiğini kontrol ediniz.

	const		const	Sonuç
Nesne	evet	Fonksiyon	evet	?
Nesne	hayır	Fonksiyon	hayır	?
Nesne	hayır	Fonksiyon	evet	?
Nesne	evet	Fonksiyon	hayır	?

```
1  #include <iostream>
2  using namespace std;
3  class prs{
4      public:
5          double maas;
6          prs(){
7          maas=0.0;
8          }
9          void maasAta(double m){
10             maas=m;
11         }
12         float geridon() const{
13         return maas;
14         }
15 };
16 int main() {
17 const prs prs1;
18 prs prs2;
19 prs1.geridon();
20 prs1.maasAta(7500.85);
21 prs2.
22 prs2.maasAta(8500.25);
```

public void prs::maasAta (double m)

(2) Resources | Compile Log | Debug | Find Results | Close

File	Message
C:\Users\Yedek\Desktop\Dersİçeriği\12_Sınıflar Nesn...	In function 'int main()':
C:\Users\Yedek\Desktop\Dersİçeriği\12_Sınıflar Nesneler...	[Error] passing 'const prs' as 'this' argument of 'void prs::maasAta(double)' discards qualifiers [-fpermissive]

Program 12.9A const fonksiyon ve const nesneler (20. Satır Hatalı, Nesne const-Fonks. Değil

Yapacağınız kontrollerin sonunda ihtimallerden sonuncusunda hata alacağınızı program 12.9A'dan görebilirsiniz. Denemeleriniz sonucunda tablo aşağıdaki gibi oluşacaktır:

	const		const	Sonuç
Nesne	evet	Fonksiyon	evet	Hata Yok
Nesne	hayır	Fonksiyon	hayır	Hata Yok
Nesne	hayır	Fonksiyon	evet	Hata Yok
Nesne	evet	Fonksiyon	hayır	Hata

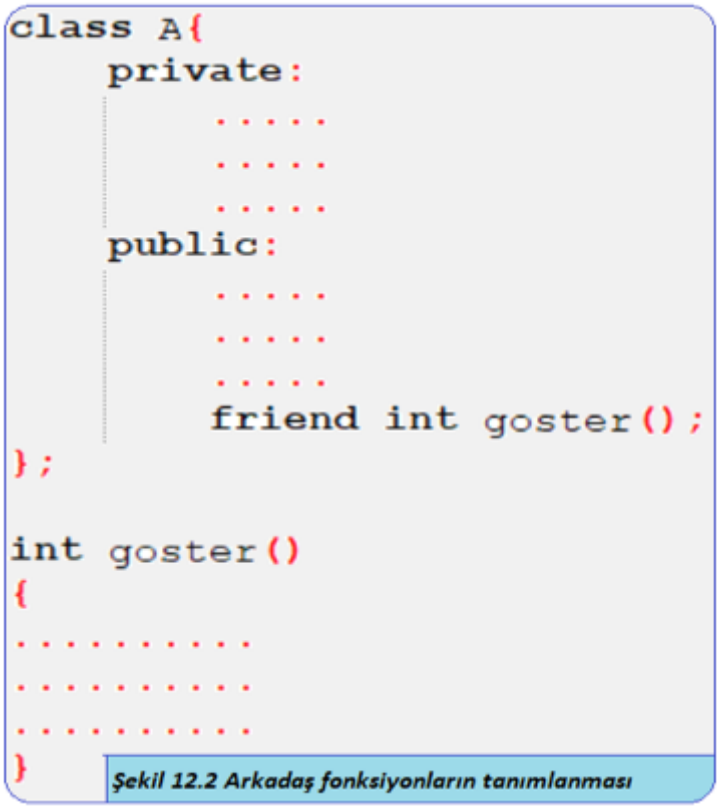
Program 12.9B, 12.9A'dan 20. Satırın kaldırılmasıyla oluşturulmuştur. Program bu yeni şekli ile derlendiğinde hata vermemiş ve beklenen ekran çıktısı alınmıştır.

```cpp
#include <iostream>
using namespace std;
class prs{
    public:
        double maas;
        prs(){
        maas=0.0;
        }
        void maasAta(double m){
            maas=m;
        }
        float geridon() const{
        return maas;
        }
};
int main() {
const prs prs1;
prs prs2;
prs1.geridon();
//prs1.maasAta(7500.85);
prs2.geridon();
prs2.maasAta(8500.25);
cout<<prs2.maas<<endl;
    return 0;}
```

Problem 12.9B const fonksiyon ve const nesne örneği

12.6. Arkadaş Fonksiyonlar

Arkadaş fonksiyonlar sınıfın dışında tanımlanan ve sınıfın içerisindeki private veriler ulaşmak için kullanılan fonksiyonlardır. Bir sınıfa ait arkadaş bir fonksiyon tanımlanacak ise ***friend*** anahtar kelimesinden yararlanılır. Arkadaş fonksiyonların bildirimleri sınıf içerisinde private veya public alanlarının içerisinde yapılabilir. Fonksiyon tanımlaması ise sınıf dışında yapılır. Aşağıda şekil 12.2'de arkadaş fonksiyonların bildiriminin prototipi gösterilmiştir.



Şekil 12.2 Arkadaş fonksiyonların tanımlanması

C++ ta arkadaş fonksiyon tanımlanabildiği gibi arkadaş sınıflar da tanımlanabilir. Arkadaşlık bir sınıf tarafından verilir ve bir fonksiyon veya başka bir sınıf tarafından alınır. İki sınıfın kendi aralarında arkadaş olması ve bu sınıflardan birinin başka bir sınıfla arkadaşlık ilişkisi bulunması, bu üçüncü sınıfın diğer sınıfla arkadaş olacağı sonucunu doğurmaz.

Aşağıda içerisinde arkadaş sınıf tanımlaması yapılmış ve kullanılmış bir C++ programı verilmiştir.

```cpp
#include <iostream>
using namespace std;
class A {
private:
    int a;
  public:
    A(){
        a=4;
    }
    friend int kare();
}k;
int kare(){
    return k.a*k.a;
}
int main() {
    cout<<" Sayinin Karesi : "<<kare()<<endl;
    return 0;
}
```

Program 12.10 Arkadaş fonksiyonlar

```
Sayinin Karesi : 16

--------------------------------
Process exited after 0.5454 seconds with return value 0
Press any key to continue . . .
```

Program 12.10'un ekran çıktısı

12.7. Nesne Dizileri

Nesne dizilerini ele almadan önce program 12.11' inceleyelim. Program 12.11'de önce Ogrenci isimli bir sınıf oluşturulmuştur. Bu sınıfta ***private*** üye olarak iki özellik (***string Ad, int Ortalama***) ve ***public*** üye olarak üç fonksiyon (***void getAd(), void getOrtalama, void goruntule()***) yer almaktadır.

Sınıfı yukarıda açıklandığı gibi tanımladıktan sonra Ogrenci sınıfına ait üç nesne oluşturuldu (***ogr1, ogr2, ogr3***). Daha sonra ***Ogrenci*** sınıfına ait ***getAd(), getOrtalama()*** isimli üye fonksiyonlar çağırılarak nesnelerin private olan özelliklerinin klavyeden girişleri yapıldı. Programın sonunda da **goruntule()** üye fonksiyonu çağırılarak bu üç nesneni bilgisayara girilen özellikleri ekranda gösterildi.

```
#include <iostream>
#include <locale.h>
using namespace std;
class Ogrenci
{
    string Ad;
    int Ortalama;
    public:
        void getAd(){
            getline( cin, Ad);
        }
        void getOrtalama(){
            cin >> Ortalama;
        }
        void goruntule(){
            cout << "Ad       : " << Ad << endl;
            cout << "Ortalama : " << Ortalama << endl;
        }
};
int main(){
    setlocale(LC_ALL,"turkish");
    Ogrenci ogr1,ogr2,ogr3; int i=1;
    cout << "Öğrenci " << i << endl;
    cout << "Öğrenci Adı         : ";cin.ignore();ogr1.getAd();
    cout << "Öğrenci Ortalaması : ";ogr1.getOrtalama();i+=1;
    cout << "Öğrenci " << i << endl;
    cout << "Öğrenci Adı         : "; cin.ignore(); ogr2.getAd();i+=1;
    cout << "Öğrenci Ortalaması : "; ogr2.getOrtalama();
    cout << "Öğrenci " << i << endl;
    cout << "Öğrenci Adı         : ";cin.ignore(); ogr3.getAd();
    cout << "Öğrenci Ortalaması : ";ogr3.getOrtalama();
    ogr1.goruntule();ogr2.goruntule();ogr3.goruntule();
    return 0;}
```

Program 12.11 Nesne Dizisi Kullanılmadan Öğrenci Bilgilerinin Bilgisayara Girilmesi

Yukarıda verilen örnekte sadece üç öğrencinin verilerinin bilgisayara girilmesi istenmişti. Her zaman bu şekilde aynı sınıfa ait nesnelerin bir veya bir kaçıyla ilgili işlem yapmamız gerekseydi, problemlerimizi program 12.11'deki gibi bir yaklaşım belirleyerek çözebilirdik.

Fakat bazen 50 bazen 100 bazen binler hatta on binlerce öğrencinin bilgilerinin bilgisayara girilmesi gerekebiliyor. Böyle bir durumda program 12.11'i kullanmamız gerekseydi 10 000 nesne tanımlayıp her bir nesnenin özelliklerinin girişini yapabilmek için 12 ifade yazmamız gerekecekti. Yazılması gereken toplam ifade sayısının ne kadar artacağını kestirmeyi size bıraktık.

Anlatılan öyküden bu ve benzeri işlemler için program 12.11'de kullanılan yaklaşımdan başka bir yaklaşım bulma ihtiyacında olduğumuz kesin olarak anlaşılmaktadır. Bunun için program 12.12 hazırlandı ve bu programda öğrenciler için nesnelerinden oluşan bir dizi tanımlanarak yine 3 öğrencinin adının ve ortalamasının bilgisayara girilmesini sağlayan örnek gösterildi. Program 12.12'yi inceleyerek çözümün ne kadar kısaldığını ve pratik hale geldiğini anlamaya çalışınız. Ayrıca, çok küçük müdahalelerle bu programla istenildiği kadar öğrencinin bilgelerinin bilgisayara girileceği açıktır.

```cpp
#include <iostream>
using namespace std;
class Ogrenci
{
    string Ad;
    int Ortalama;
    public:
        void getAd(){
            getline( cin, Ad);
        }
        void getOrtalama(){
            cin >> Ortalama;
        }
        void goruntule(){
            cout << "Ad       : " << Ad << endl;
            cout << "Ortalama : " << Ortalama << endl;
        }
};
int main(){
    setlocale(LC_ALL,"turkish");
    int sayi=3;Ogrenci ogr[sayi];
    for( int i=0; i<sayi; i++ ){
        cout <<endl<< "Öğrenci " << i+1 << endl;
        cout << "Öğrenci Adı         : ";cin.ignore();ogr[i].getAd();
        cout << "Öğrenci Ortalaması : ";ogr[i].getOrtalama();
    }
    for(int i=0;i<sayi;i++) {
        cout <<endl<<"Öğrenci " << i+1 << endl;
        ogr[i].goruntule();
    }
    return 0;}
```

Program 12.12 Nesne Dizileri

Program 12.12 incelendiğinde nesne dizilerinin tanımlanması ve kullanılmasının daha önce kullandığımız dizilere çok benzediği görülmektedir. Bundan başka nesne dizilerinin kullanılmasının çok daha kısa, okunabilir program yazmak için çok pratik faydalar sağladığı görülmektedir.

Bölüm Özeti

C++ programlama dili aslında birçok yönden C programlama diline benzemektedir. C++ programlama dilinin C programlama dilinden farklı en temel özelliği nesneye yönelik programlama yaklaşımını destekliyor olmasıdır. Nesneye yönelik programlamanın ilk adımı ve en önemli aracı sınıflardır. Bu bölümde Bir sınıfın, bir nesnenin biçimini belirtmek için kullanıldığını verilerle, verileri işlemek için kullanılacak işlevleri tek bir düzgün pakette birleştirdiğini öğrendik. Bundan sonrası için sınıf içindeki verilerin (özelliklerin) ve işlevlerin (fonksiyonların) sınıfın üyeleri olarak isimlendirileceğini biliyoruz.

On ikinci bölümün devamında, program içerisinde tanımı yapılmış olan sınıfa ait nesnelerin nasıl tanımlanacağını ele aldık. Nesneye yönelik programlama yaklaşımında nesnelerin tanımlanması işlemine ***örnek(instance) oluşturmak***, işlem neticesinde oluşturulan nesnelere de ***örnek değişken***(***instance variable***) adı da verilmektedir. Esasında programlar içerisinde nesne tanımlamanın, herhangi bir temel veri tipindeki değişken tanımlamaya çok benzediğine dikkat ediniz. Ayrıca, kullanıcı tarafından tanımı yapılmış bir yapı ile yapı değişkeni tanımlama işleminin de tanımı yapılmış sınıfla nesne tanımlanmasına benzediğini hatırdan çıkarmayınız.

Bu bölümde daha sonra, tanımlanmış olan nesneler aracılığı ile o sınıfa ait üye fonksiyonlara erişimin nasıl yapılacağı örneklerle açıklandı. Üye fonksiyonların çağırılması için oluşturulan C++ ifadelerinde üye

fonksiyon ile nesnenin isminin nokta operatörünün ilişkilendirdiği gösterildi.

C++'da nesnelerin otomatik olarak ilk kullanıma hazırlanması işleminin kurucu (constructor) fonksiyonlar ile yapıldığı yine bu bölümde ele alındı ve kurucu fonksiyonların isimlerinin sınıflarla aynı olması gerekliliği vurgulandı. Benzer şekilde çalışması sona ermiş nesneler için bilgisayar belleğinde ayrılmış olan adreslerin serbest bırakılması amacıyla da de yıkıcı fonksiyonların kullanıldığı örneklerle açıklandı.

Sınıflar ve Nesneler bölümünün son dört konusu Bir nesneyi Başka Bir Nesneye Atamak, const nesneler ve const sınıflar, Arkadaş fonksiyonlar ve nesne dizileri oldu.

Bir nesneyi başka bir nesneye kopyalamak başlığı altında aynı sınıfa ait nesnelerin birbiri arasında, değişkenlerde olduğu kopyalanabileceği örnekle gösterildi.

Daha önceki bölümlerde değişkenlerin program içerisinde değerlerinin değişmez olmasını sağlayarak, programın veri açısından güvenliğinin arttırılması gerektiğini, bunun için const bildiriminin kullanılabileceğini öğrenmiştik. Bu bölümde de nesneye yönelik programlamada **const** nesneler kullanarak aynı amaçların gerçekleştirebileceği bir örnekle açıklandı.

Arkadaş fonksiyonların sınıfın dışında tanımlandığı ve sınıfın içerisindeki private verilere ulaşmak için kullanıldığı, bir sınıfa ait arkadaş fonksiyonun ***friend*** anahtar kelimesinden yararlanarak tanımlanabileceği ve arkadaş fonksiyonların bildirimlerinin sınıf içerisinde private veya public alanlarının içerisinde yapılabileceği yine bir örnek üzerinden gösterildi.

Bu bölümün son konusu nesne dizileri oldu. Nesne dizilerinin tanımlanması ve kullanılmasının daha önceki bölümlerde kullanılan dizilere çok benzediği ve daha kısa kod yazarak program yazmaya önemli katkı sağlayacağı gösterildi.

Harvey M. Deitel, Paul J. Deitel, C ve C++, Sistem Yayıncılık,2004

Dr. Rifat Çölkesen, Veri Yapıları ve Algoritmalar, Papatya,2007

H. Burak Tungut, Algoritma ve Programlama Mantığı, KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti, 2019.

Duygu Arbatlı Yağcı, Nesne Yönelimli C++ Programlama Kılavuzu, Alfa Basım Yayım Dağıtım Ltd. Şti, 2016.

G. Murat Taşbaşı, C programlama, Altaş yayıncılık ve Elektronik Tic. Ltd. Şti. 2005.

Muhammed Mastar, Süha Eriş, C++, KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti, 2012.

Sabahat Karaman, Pratik C++ Programlama, Pusula Yayınevi,2004