

# Programlamaya Giriş

## HAFTA 12

### Dosyalar (Files)

**Prof. Dr. Cemil ÖZ**

**Doç. Dr. Cüneyt BAYILMIŞ**

**Dr. Öğretim Üyesi Gülüzar ÇİT**

# Konu & İçerik

- Dosyalama İşlemleri
- Dosya Modları
- Metin Dosyaları – İkili Dosyalar
- Dosya Veri Yazma İşlemi
- Dosya Veri Okuma İşlemi
- Dosya Konum İşaretçileri
- Rasgele Erişimli Dosya İşlemleri
- Kaynaklar



# Dosyalama İşlemleri

- Verilerin sürekli saklanması/depolanması için dosyalardan yararlanılır.
- Bilgisayarın işlediği tüm verilerin sonuçta 1 ve 0'lardan oluştuğu unutulmamalıdır.
- Bir dosya (file) birden çok kayıt (record) alanından oluşur.
- Kayıt alanları karakter grupları ile karakterler de bit grupları ile gösterilir.
- Programlar dosyalara iki farklı şekilde ulaşırlar; sıralı ve rasgele.
- Yazdığınız program sizin ulaşım seçiminize göre değişiklik gösterecektir.
- Dosyaya erişim modunuz, sizin dosyadan veriyi nasıl okuyacağınızı, dosyaya veriyi nasıl yazacağınızı, dosyadaki verileri nasıl değiştireceğinizi veriyi dosyadan nasıl sileceğinizi vb. durumları belirlemenizi sağlar.
- Bazı dosyalara iki şekilde de ulaşılabilir.

# Dosyalama İşlemleri...

- Dosyalamayla ilgili 3 temel işlem
  - Dosya açılır
  - Veri yazılır, ya da okunur
  - Dosya kapatılır
- Dosyalarla ilgili işlemlerde aşağıda bulunan ve **istream** ile **ostream** sınıflarından türetilen sınıflar kullanılır (cin, cout bu sınıflardan oluşturulan nesnelerdir.).
  - **ofstream**: Dosyalara yazmak için kullanılan stream sınıfı
  - **ifstream**: Dosyalardan okumak için kullanılan stream sınıfı
  - **fstream**: Dosyalara yazmak ve okumak için kullanılan stream sınıfı

# Dosyalama İşlemleri...

- Dosya açılırken dosya adı ve mod belirtilir

```
fstream dataFile("ornek.txt", ios::out | ios::app);
```

```
ofstream myFile;  
myFile.open("ornek.bin", ios::out | ios::app | ios::binary);
```

Sınıf Adı	Varsayılan Mod Parametresi
ofstream	ios::out
ifstream	ios::in
fstream	ios::in   ios::out

# Dosyalama İşlemleri...

## ➤ Dosya Modları

Mod Bayrağı	Görevi
<b>ios::in</b>	Varolan dosyanın okunmak üzere açılması. Dosya yok ise hata döndürülür
<b>ios::app</b>	Dosyaya yazmak için kullanılır. Dosya yok ise oluşturulur. Tüm ekleme işlemleri dosya sonuna yapılır.
<b>ios::ate</b>	ios::app gibidir. Farklı olarak yazma işlemi herhangi bir yere yapılabilir.
<b>ios::binary</b>	Dosyanın ikili modda açılmasını sağlar. Varsayılan mod text dir
<b>ios::out</b>	Yazma modu. Dosya var ise içeriği boşaltılır.
<b>ios::trunc</b>	Dosya var ise içeriği temizlenir. ios::out için varsayılan moddur.
<b>ios::noreplace</b>	Dosya var ise hata verir.
<b>ios::nocreate</b>	Dosya yok ise oluşturulmasın. Open fonksiyonu dosya yok ise hata verecektir.

# Metin Dosyaları – İkili Dosyalar

- İkili (binary) modda açılan dosya akışları giriş ve çıkış işlemlerini formattan bağımsız olarak gerçekleştirirler.
- Metin (text) dosyalarda ise bazı özel karakterlerin (satırbaşı ve satırbaşına dönüş karakterleri) formatından dolayı bazı dönüşüm işlemleri gerçekleşir.
- İkili dosyalar formatsızdır ve ASCII formatında saklanmazlar.

1	2	9	7	<EOF>
---	---	---	---	-------

1297 expressed in ASCII

49	55	57	55	<EOF>
----	----	----	----	-------

1297 as an integer, in binary

00000101	00010001
----------	----------

1297 as an integer, in hexadecimal

05	11
----	----

# Dosyaya Veri Yazma İşlemleri

- Dosya yazma işleminden önce dosya açılmalıdır.

```
① ofstream dosyaYaz("kayit.txt", ios::out);
```

ofstream      dosyaYaz      "kayit.txt"      ios::out

sınıf adı      dosya nesnesinin adı      kullanılan dosya      dosya açma modu

```
② ofstream dosyaYaz;  
dosyaYaz.open("kayit.txt"); veya ofstream dosyaYaz;  
dosyaYaz.open("kayit.txt", ios::out);
```

- Dosyaya veri yazma

```
dosyaYaz << "Dosyaya Yazıyorum... :-) \n";
```

- Dosya yazma işleminden sonra dosya kapatılmalıdır. Herhangi bir kapatma işlemi yapılmazsa işletim sistemi dosyayı program bitince kapatır

```
dosyaYaz.close();
```



# Dosyadan Veri Okuma İşlemleri

➤ Dosya okuma modunda açılmalıdır.

① `ifstream dosyaOku("kayit.txt", ios::in);`

`ifstream` `dosyaOku` `"kayit.txt"` `ios::in`

sınıf adı dosya nesnesinin adı kullanılan dosya dosya açma modu (default)

② `ifstream dosyaOku;`  
`dosyaOku.open("kayit.txt");` veya `ifstream dosyaOku;`  
`dosyaOku.open("kayit.txt", ios::out);`

➤ Dosyadan veri okuma

`dosyaOku >> degisken;` veya `dosyaOku.get(degisken);`

# Dosyadan Veri Okuma İşlemleri...

- String ifadeler gibi dosya sonunda bir işaretçi ile sonlanır.
- Dosya sonuna gelinip gelinmediği kontrol edilebilir

```
while (!dosyaOku.eof())    // end of file
```

veya

```
while (dosyaOku.good())
```

# Dosya İşlemleri

- Dosyanın açılıp açılmadığını kontrol etme işlemi

```
if (dosyaYaz.is_open())  
    cout << "Dosya açıldı";
```

# Dosya İşlemleri...

## ➤ ÖRNEK: ⇒ [1]\_dosya.cpp

```
// Dosya veri yazma işlemi
ofstream dosyaYaz;
dosyaYaz.open("kayit.txt", ios::out);

dosyaYaz << "Dosya Veri Yazma \n";
dosyaYaz.close();

// dosyadan okuma işlemi
ifstream dosyaOku("kayit.txt", ios::in);

if (dosyaOku.is_open())
    cout << "Dosya Acma Basarili\n";

char oku;

while (!dosyaOku.eof()) {
    dosyaOku >> oku;
    cout << oku << " ";
}

dosyaOku.close();
```

```
#include <fstream>
```

# Dosya İşlemleri...

- C++'da dosyaya herhangi bir yapı zorlanmaz.
- Dosyaları programcı formatlı veri girişi ile yapılandırır. Örneğin ad, tel bilgilerinin tek bir kişiye ait olması
- ÖRNEK: ⇒ [2]\_dosya.cpp

```
rehber rehberim;

char devam = 'e';

ofstream dosyaYaz;

dosyaYaz.open("Rehber.txt", ios::app);

do {
    cout << "ad ve telefon bilgileri giriniz\n";
    cin >> rehberim.ad >> rehberim.tel;
    dosyaYaz << rehberim.ad << "\t" << rehberim.tel << "\n";
    cout << "\n Yeni kayıt yapacak mısınız (e/h) ";
    cin >> devam;
} while (!(devam == 'h'));

dosyaYaz.close();
```

```
struct rehber {
    string ad;
    string tel;
};
```

# Dosya İşlemleri...

## ➤ ÖRNEK: ⇒ [2]\_dosya.cpp...

```
ifstream dosyaOku;

dosyaOku.open("Rehber.txt");

string isim;

cout << "Aranilacak Adi Giriniz: ";
cin >> isim;

while (!dosyaOku.eof()) {
    // while (dosyaOku>>rehberim.ad>>rehberim.tel)
    dosyaOku >> rehberim.ad >> rehberim.tel;
    if (isim == rehberim.ad)
        cout << rehberim.ad << "\n" << rehberim.tel;
}

dosyaOku.close();
```

```
struct rehber {
    string ad;
    string tel;
};
```

# Dosya Konum İşaretçileri

- Ardışık olarak bir dosyadan okuma işleminde, programlar işleme dosyanın başından başlar ve istenilen veri bulunana kadar art arda tüm verileri okur.
- Bir dosyanın her seferinde başından itibaren işleme sokulması gereksiz işlem ve zaman kaybına neden olur.
- Bu problem, hem okuma hem de yazma işlemi için dosya konum işaretçileri ile çözülür.
- Dosya konum işaretçileri işlem yapılacak (okuma/yazma) bir sonraki baytın numarasını tutar.

**ifstream** ⇒ **seekg** (seek get)

**ofstream** ⇒ **seekp** (seek put)

- Ayrıca, dosyada konumlandırmanın yönü belirtilmelidir.

- **beg** ⇒ başlangıca göre

- **cur** ⇒ geçerli/mevcut konuma göre

- **end** ⇒ sonuna göre

# Dosya Konum İşaretçileri...

## ➤ Dosya Konum İşaretçi Modları

Komut	Okuma/Yazma Pozisyonunu Etiketleme Durumu
<code>dosyaNesnesi.seekg ( n );</code>	Dosyanın n. baytına konumlandırır. n=0 ise dosya başına konumlandırır.
<code>dosyaNesnesi.seekp( n, ios::beg );</code> <code>dosyaNesnesi.seekp(32, ios::beg);</code>	Yazma pozisyonunu dosya başından n.(n=33) bayta kur.
<code>dosyaNesnesi.seekp( -10, ios::end );</code>	Dosya sonundan 11. bayta yazma pozisyonunu kur.
<code>dosyaNesnesi.seekp( 120, ios::cur );</code>	Geçerli pozisyondan 121. bayta yazma pozisyonunu kur.
<code>dosyaNesnesi.seekg ( 2, ios::beg );</code>	Dosya başından 3. bayta okuma pozisyonunu kur.
<code>dosyaNesnesi.seekg ( -100, ios::end );</code>	Dosya sonundan 101. bayta okuma pozisyonunu kur.
<code>dosyaNesnesi.seekg ( 40, ios::cur );</code>	Geçerli pozisyondan 41. bayta okuma pozisyonunu kur.
<code>dosyaNesnesi.seekg( 0, ios::end );</code>	Dosya sonuna okuma pozisyonunu kur.



# Dosya Konum İşaretçileri...

- Dosya üzerinde bulunan mevcut konumu öğrenme;

**ifstream** ⇒ **tellg** (tell get)

**ofstream** ⇒ **tellp** (tell put)

- Dosyadaki mevcut konumu öğrenme

```
long konum;
```

```
konum = dosyaNesnesi.tellg();
```

# Rasgele Erişimli Dosya İşlemleri

- Rasgele erişimli dosya ile ayrı kayıtlara, diğer kayıtları aramaya gerek kalmadan doğrudan ve hızlı bir şekilde erişilebilir.
- C++ ta bir dosya için belirli bir yapı zorunluluğu yoktur.
- Rasgele erişim için en kolay yöntemlerden biri dosyadaki tüm kayıtların/kayıt alanlarının aynı büyüklükte gerçekleştirilmesidir. Böylelikle bir kaydın konumu hızlı bir şekilde tespit edilebilir.
- Sabit kayıt alanı kullanımı ile dosyada değişiklikler/güncellemeler, diğer verilere zarar vermeden daha kolay gerçekleştirilebilir.

## ➤ Dosyaya veri yazma

```
dosyaNesnesi.write(reinterpret_cast <const char *> (&kayit), sizeof(kayit));
```

## ➤ Dosyadan veri okuma

```
dosyaNesnesi.read(reinterpret_cast <char *> (&kayit), sizeof(kayit));
```

# Rasgele Erişimli Dosya İşlemleri...

- Rasgele erişimli dosyalarda konum işaretçisini konumlandırma işlemi için ardışık dosyalarda da kullanılan **seekg** ve **seekp** komutları kullanılır.

```
dosyaNesnesi.seekp(sayi * sizeof(SinifAdi));
```

```
dosyaNesnesi.seekp(nesneAdi.fonksiyon() - 1) * sizeof(SinifAdi));
```

```
dosyaYaz.seekp ( rehberim.al () - 1) * sizeof (rehber)) ;
```

# Rasgele Erişimli Dosya İşlemleri...

➤ ÖRNEK: ⇒ [3]\_rasgele.cpp

```
struct Envanter
{
    char tanim[31];
    int kalite;
    float ucret;
};
```

```
fstream envanter("env.dat", ios::out | ios::binary);
Envanter kayit = { "", 0, 0.0 }; // {"sakarya", 54, 54.0 } sabit kayit

// Boş kayıtlar yazılıyor
for (int count = 0; count < 5; count++)
{
    cout << "Now writing record " << count << endl;
    envanter.write((char *)&kayit, sizeof(kayit));
}
envanter.close();
```

# Rasgele Erişimli Dosya İşlemleri...

## ➤ ÖRNEK: ⇒ [4]\_rasgele.cpp

```
fstream envanter("env.dat", ios::in | ios::binary);

Envanter kayit = { "", 0, 0.0 };

// kayıtlar okunuyor ve gösteriliyor
envanter.read((char *)&kayit, sizeof(kayit));

while (!envanter.eof())
{
    cout << "Tanim    : " << kayit.tanim << endl;
    cout << "Kalite    : " << kayit.kalite << endl;
    cout << "Ucret      : " << kayit.ucret << endl << endl;

    envanter.read((char *)&kayit, sizeof(kayit));
}

envanter.close();
```

# KAYNAKLAR

- Deitel, C++ How To Program, Prentice Hall
- Horstmann, C., Budd, T., Big C++, Jhon Wiley & Sons, Inc.
- Robert Lafore, Object Oriented Programming in C++, Macmillan Computer Publishing
- Prof. Dr. Celal ÇEKEN, Programlamaya Giriş Ders Notları
- Prof. Dr. Cemil ÖZ, Programlamaya Giriş Ders Notları