

Dosya İşlemleri-2

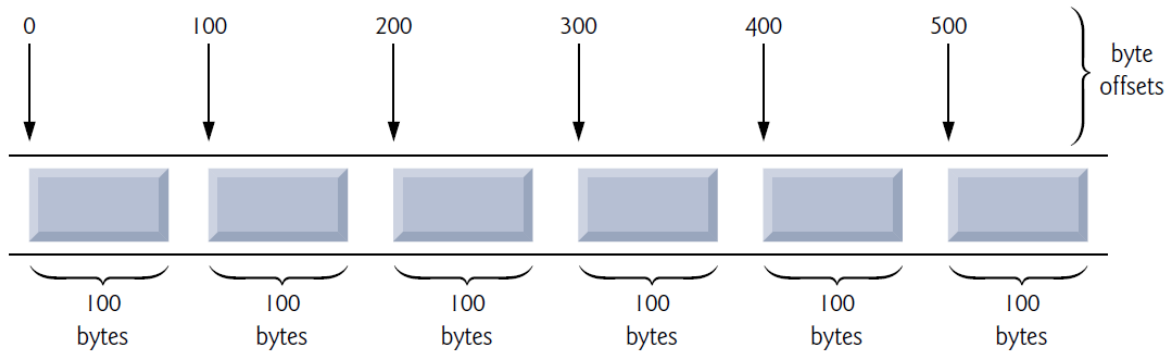
Rastgele Erişimli Dosyalar

Daha önce belirttiğimiz gibi, fprintf biçimlendirilmiş çıktı fonksiyonu ile oluşturulan bir dosyadaki kayıtların aynı uzunlukta olması gerekmez. Ancak, rasgele erişimli bir dosyaya yazdığınız ve buradan okuduğunuz bireysel kayıtların uzunluğu normalde sabittir ve diğer kayıtlar aranmadan doğrudan (ve dolayısıyla hızlı bir şekilde) erişilebilir.

Bu, rastgele erişim dosyalarını havayolu rezervasyon sistemleri, bankacılık sistemleri, satış noktası sistemleri ve belirli verilere hızlı erişim gerektiren diğer hesap işleme sistemleri için uygun hale getirir.

Rastgele erişimli dosyaları uygulamanın başka yolları da vardır, ancak şimdilik sabit uzunluklu kayıtları kullanan bu basit yaklaşımla sınırlayacağız. Bir rasgele erişim dosyasındaki her kayıt normalde aynı uzunluğa sahip olduğundan, bir kaydın dosyanın başlangıcına göre tam konumu, kayıt anahtarının bir fonksiyonu olarak hesaplanabilir. Bu da, büyük dosyalarda bile belirli kayıtlara anında erişimi kolaylaştırır.

Şekil 11.9, bir rasgele erişim dosyasını uygulamanın bir yolunu göstermektedir.



Şekil 11.9 | Rastgele erişimli bir dosyanın C görünümü.

Sabit uzunluklu kayıtlar, dosyadaki diğer verileri yok etmeden verilerin rastgele erişimli bir dosyaya eklenmesini sağlar. Daha önce saklanan veriler, dosyanın tamamı yeniden yazılmadan da güncellenebilir veya silinebilir. Bunun için;

- bir rasgele erişim dosyası oluşturulur,
- veriler girilir,
- veriler hem sırayla hem de rastgele okunabilir,
- veriler güncellenebilir,
- gerekli olmayan veriler silinebilir.

Rastgele Erişimli Dosya Oluşturma

fwrite fonksiyonu, bellekte belirli bir konumdan başlayarak belirli sayıda baytı bir dosyaya aktarır. Veriler, dosya konumu işaretçisi tarafından belirtilen dosya konumundan başlayarak yazılır. fread fonksiyonu, dosya konum işaretçisi tarafından belirtilen dosyadaki konumdan belirli sayıda baytı, belirtilen adresle başlayan bellekteki bir alana aktarır.

Örneğin, dört baytlık bir tamsayı yazarken,

```
fprintf(fPtr, "%d", number);
```

fonksiyonunu kullanmak yerine, aşağıdaki gibi fPtr ile temsil edilen dört baytlık tamsayılara sahip bir sistemde her zaman dört bayt yazan aşağıdaki fonksiyonu kullanabiliriz.

```
fwrite(&number, sizeof(int), 1, fPtr);
```

Daha sonra, bu dört baytı okumak için fread kullanılabilir.

fwrite ve fread fonksiyonları dosyalara veri dizilerini yazabilir ve dosyalardan okuyabilir. Hem fread hem de fwrite'ın üçüncü argümanı, dizideki bir dosyadan okunması veya dosyaya yazılması gereken öğelerin sayısıdır.

Yukardaki fwrite fonksiyon çağrısı, bir dosyaya tek bir tamsayı yazar, bu nedenle üçüncü bağımsız değişken 1'dir (sanki bir dizinin bir öğesi yazılıyormuş gibi). Aslında dosya işleme programları nadiren bir dosyaya tek bir alan yazar. Normalde, aşağıdaki örneklerde gösterdiğimiz gibi, her seferinde bir struct yapısı yazarlar.

Aşağıdaki problemi ele alalım:

100 adede kadar sabit uzunlukta kaydı depolayabilen bir hesap işleme sistemi oluşturalım. Her kayıt, kayıt anahtarı olarak kullanılacak bir hesap numarası, bir soyadı, bir ad ve bir bakiyeden oluşmalıdır. Ortaya çıkan program, bir hesabı güncelleyebilmeli, yeni bir hesap kaydı ekleyebilmeli, bir hesabı silebilmeli ve tüm hesap kayıtlarını yazdırılmak üzere biçimlendirilmiş bir metin dosyasında listeleyebilmelidir. Bunun için rastgele erişimli bir dosya kullanalım.

Şekil 11.10, rastgele erişimli bir dosyanın nasıl açılacağını, bir struct yapısı kullanarak bir kayıt formatının nasıl tanımlanacağını, dosyaya nasıl veri yazılacağını ve dosyanın nasıl kapatılacağını gösterir.

Bu program, "accounts.dat" dosyasının tüm 100 kaydını fwrite işlevini kullanarak boş yapılarla başlatır. Her boş yapı, hesap numarası için 0, soyadı için "" (boş dize), ad için "" (boş dize) ve bakiye için 0,0 içerir. Dosya, dosyanın depolanacağı alanı oluşturmak ve bir kaydın veri içerip içermediğini belirlemeyi mümkün kılmak için bu şekilde başlatılır.

```

1 // Fig. 11.10: fig11_10.c
2 // Creating a random-access file sequentially
3 #include <stdio.h>
4
5 // clientData structure definition
6 struct clientData {
7     unsigned int acctNum; // account number
8     char lastName[15]; // account last name
9     char firstName[10]; // account first name
10    double balance; // account balance
11 };
12
13 int main(void)
14 {
15     FILE *cfPtr; // accounts.dat file pointer
16
17     // fopen opens the file; exits if file cannot be opened
18     if ((cfPtr = fopen("accounts.dat", "wb")) == NULL) {
19         puts("File could not be opened.");
20     }
21     else {
22         // create clientData with default information
23         struct clientData blankClient = {0, "", "", 0.0};
24
25         // output 100 blank records to file
26         for (unsigned int i = 1; i <= 100; ++i) {
27             fwrite(&blankClient, sizeof(struct clientData), 1, cfPtr);
28         }
29
30         fclose (cfPtr); // fclose closes the file
31     }
32 }

```

Şekil 11.10 | Adım adım rastgele erişimli bir dosya oluşturma.

fwrite fonksiyonu bir dosyaya bir bayt bloğu yazar. Satır 27 sizeof(struct clientData) boyutunda blankClient yapısının cfPtr tarafından işaret edilen dosyaya yazılmasını sağlar. sizeof operatörü parantez içinde işleneninin boyutunu bayt cinsinden döndürür (bu durumda struct yapısı clientData).

fwrite fonksiyonu aslında bir nesne dizisinin birkaç öğesini yazmak için kullanılabilir. Önceki ifadede, dizi öğesi olmayan tek bir nesneyi yazmak için fwrite kullanılmıştı. Tek bir nesne yazmak, bir dizinin bir öğesini yazmaya eşdeğerdir, dolayısıyla fwrite çağrısında 1 olur.

[Not: Şekil 11.11 ve 11.14, Şekil 11.10'da oluşturulan veri dosyasını kullanır, bu nedenle Şekil 11.10'u diğer programlardan önce çalıştırmalısınız.]

Şekil 11.11, verileri "accounts.dat" dosyasına yazar. Verileri dosyadaki belirli konumlarda depolamak için fseek ve fwrite kombinasyonunu kullanır. fseek fonksiyonu dosya konumu işaretçisini dosyada belirli bir konuma ayarlar, ardından fwrite verileri yazar. Örnek bir veri yazma uygulaması, Şekil 11.11'de gösterilmiştir.

```
1 // Fig. 11.11: fig11_11.c
2 // Writing data randomly to a random-access file
3 #include <stdio.h>
4
5 // clientData structure definition
6 struct clientData {
7     unsigned int acctNum; // account number
8     char lastName[15]; // account last name
9     char firstName[10]; // account first name
10    double balance; // account balance
11 }; // end structure clientData
12
13 int main(void)
14 {
15     FILE *cfPtr; // accounts.dat file pointer
16
17     // fopen opens the file; exits if file cannot be opened
18     if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL) {
19         puts("File could not be opened.");
20     }
21     else {
22         // create clientData with default information
23         struct clientData client = {0, "", "", 0.0};
24
25         // require user to specify account number
26         printf("%s", "Enter account number"
27             " (1 to 100, 0 to end input): ");
28         scanf("%d", &client.acctNum);
29
30         // user enters information, which is copied into file
31         while (client.acctNum != 0) {
32             // user enters last name, first name and balance
33             printf("%s", "\nEnter lastname, firstname, balance: ");
34
35             // set record lastName, firstName and balance value
36             fscanf(stdin, "%14s%9s%lf", client.lastName,
37                 client.firstName, &client.balance);
38
39             // seek position in file to user-specified record
40             fseek(cfPtr, (client.acctNum - 1) *
41                 sizeof(struct clientData), SEEK_SET);
42
43             // write user-specified information in file
44             fwrite(&client, sizeof(struct clientData), 1, cfPtr);
45
46             // enable user to input another account number
47             printf("%s", "\nEnter account number: ");
48             scanf("%d", &client.acctNum);
49         }
50
51         fclose(cfPtr); // fclose closes the file
52     }
53 }
```

Şekil 11.11 | Rastgele erişimli bir dosyaya rastgele veri yazma.

```
Enter account number (1 to 100, 0 to end input): 37
Enter lastname, firstname, balance: Barker Doug 0.00
Enter account number: 29
Enter lastname, firstname, balance: Brown Nancy -24.54
Enter account number: 96
Enter lastname, firstname, balance: Stone Sam 34.98
Enter account number: 88
Enter lastname, firstname, balance: Smith Dave 258.34
Enter account number: 33
Enter lastname, firstname, balance: Dunn Stacey 314.33
Enter account number: 0
```

Şekil 11.12 | Şekil 11.11'deki programın örnek uygulaması.

40–41 satırları, cfPtr tarafından başvuru dosya için dosya konumu işaretçisi

(client.accountNum - 1) * sizeof(struct clientData)

tarafından hesaplanan bayt konumuna konumlandırır.

Bu ifadenin değeri ofset veya yer değiştirme olarak adlandırılır. Hesap numarası 1 ile 100 arasında olup dosyadaki byte konumları 0 ile başladığından kaydın byte konumu hesaplanırken hesap numarasından 1 çıkarılır. Böylece, kayıt 1 için, dosya konumu işaretçisi dosyanın 0. baytına ayarlanır.

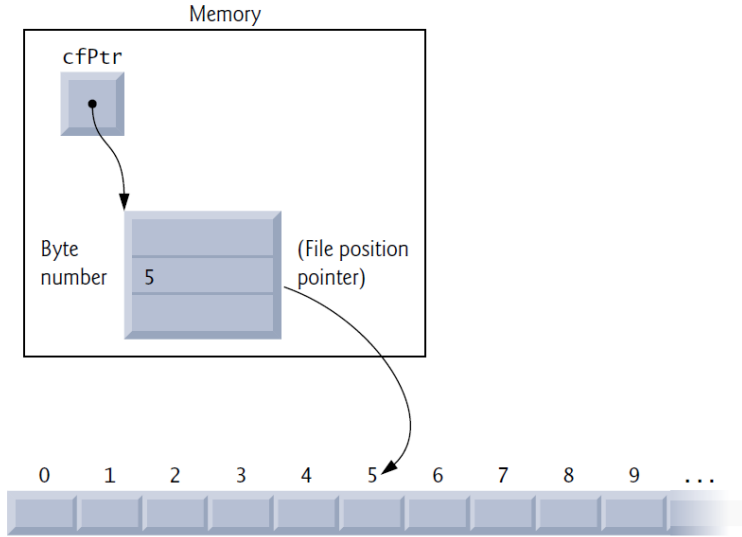
SEEK_SET sembolik sabiti, dosya konumu işaretçisinin, ofset miktarına göre dosyanın başlangıcına göre konumlandırıldığını gösterir. Yukarıdaki ifadenin de belirttiği gibi, dosyada hesap numarası 1'in aranması, hesaplanan bayt konumu 0 olduğu için dosya konumu işaretçisini dosyanın başlangıcına ayarlar.

Şekil 11.13, bellekteki bir FILE yapısına atıfta bulunan dosya işaretçisini göstermektedir. Bu şemadaki dosya konumu işaretçisi, okunacak veya yazılacak bir sonraki baytın dosyanın başlangıcından itibaren 5nci bayt olduğunu gösterir.

fseek için fonksiyon prototipi;

int fseek(FILE *stream, **long int** offset, **int** whence);

burada ofset, akışın işaret ettiği dosya olan whence değişkeninde aranacak bayt sayısıdır; pozitif bir ofset ileriye doğru, negatif bir ofset ise geriye doğru arar. whence bağımsız değişkeni, aramanın başladığı konumu gösteren SEEK_SET, SEEK_CUR veya SEEK_END değerlerinden biridir (hepsi <stdio.h> içinde tanımlanmıştır). SEEK_SET, aramanın dosyanın başında başladığını belirtir; SEEK_CUR, aramanın dosyadaki geçerli konumda başladığını belirtir ve SEEK_END, aramanın dosyanın sonundan itibaren yapıldığını belirtir.



Şekil 11.13 | Dosyanın başlangıcından itibaren 5 baytlık bir kaymayı gösteren dosya konumu işaretçisi.

Basit olması açısından, bu bölümdeki programlar hata denetimi gerçekleştirmez. Endüstriyel programlarda, `fscanf` (Şekil 11.11, satır 36–37), `fseek` (satır 40–41) ve `fwrite` (satır 44) gibi fonksiyonların dönüş değerlerini kontrol ederek doğru çalışıp çalışmadığını belirlemelidir. `fscanf` fonksiyonu, başarıyla okunan veri öğelerinin sayısını veya verileri okurken bir sorun oluşursa EOF değerini döndürür. `fseek` fonksiyonu ile arama işlemi gerçekleştirilemezse (örneğin, dosyanın başlangıcından önceki bir konumu aramaya çalışmak gibi) sıfır olmayan bir değer döndürür. `fwrite` fonksiyonu, başarıyla çıktısını aldığı öğe sayısını döndürür.

`fread` fonksiyonu, bir dosyadan belirli sayıda baytı belleğe okur. Örneğin,

```
fread(&client, sizeof(struct clientData), 1, cfPtr);
```

`cfPtr` tarafından başvuru dosyadan `sizeof(struct clientData)` tarafından belirlenen bayt sayısını okur, verileri `client`'ta saklar ve okunan bayt sayısını döndürür.

Baytlar, dosya konumu işaretçisi tarafından belirtilen konumdan okunur. `fread` fonksiyonu, öğelerin depolanacağı diziye bir işaretçi sağlayarak ve okunacak öğe sayısını belirterek birkaç sabit boyutlu dizi öğesini okuyabilir. Önceki ifade bir öğeyi okur. Birden fazla veri okumak için, `fread`'in üçüncü argümanı olarak eleman sayısının belirtilmesi gerekir.

Şekil 11.14, "accounts.dat" dosyasındaki her kaydı sırayla okur, her kaydın veri içerip içermediğini belirler ve veri içeren kayıtlar için biçimlendirilmiş verileri görüntüler. `feof` fonksiyonu, dosyanın sonuna ne zaman ulaşılacağını belirler ve `fread` fonksiyonu, verileri dosyadan `clientData` struct yapısı olan `client`'a aktarır.

```

1 // Fig. 11.14: fig11_14.c
2 // Reading a random-access file sequentially
3 #include <stdio.h>
4
5 // clientData structure definition
6 struct clientData {
7     unsigned int acctNum; // account number
8     char lastName[15]; // account last name
9     char firstName[10]; // account first name
10    double balance; // account balance
11 };
12
13 int main(void)
14 {
15     FILE *cfPtr; // accounts.dat file pointer
16
17     // fopen opens the file; exits if file cannot be opened
18     if ((cfPtr = fopen("accounts.dat", "rb")) == NULL) {
19         puts("File could not be opened.");
20     }
21     else {
22         printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
23             "First Name", "Balance");
24
25         // read all records from file (until eof)
26         while (!feof(cfPtr)) {
27             // create clientData with default information
28             struct clientData client = {0, "", "", 0.0};
29
30             int result = fread(&client, sizeof(struct clientData), 1, cfPtr);
31
32             // display record
33             if (result != 0 && client.acctNum != 0) {
34                 printf("%-6d%-16s%-11s%10.2f\n",
35                     client.acctNum, client.lastName,
36                     client.firstName, client.balance);
37             }
38         }
39
40         fclose(cfPtr); // fclose closes the file
41     }
42 }

```

NOT: Kitapta fig_11_11 deki programda yazdığımız dosyadaki verilerin okunabilmesi için 18 nci satırda dosya adı "credit.txt" yazılmış. Ama daha önce açtığımız dosyanın okunabilmesi için bunun yerine "accounts.dat" yazılmalı

Acct	Last Name	First Name	Balance
29	Brown	Nancy	-24.54
33	Dunn	Stacey	314.33
37	Barker	Doug	0.00
88	Smith	Dave	258.34
96	Stone	Sam	34.98

Şekil 11.14 | Rastgele erişimli bir dosyayı sırayla okuma.