Dosya kavramı

İşletim sistemlerinde bir dosya, sabit disk, SSD veya USB bellek gibi ikincil bellek olarak tanımlanan cihazlarda saklanan ve belirli bir amaç için düzenlenmiş veri veya bilgilerin topluluğudur. Dosyalar, belgeler, resimler, videolar veya yazılım programları gibi farklı türlerde olabilir. Her dosyanın bir ismi vardır ve bu isim dosyanın işletim sistemi tarafından tanınmasını sağlar. Dosya isimlendirme kuralları işletim sistemine göre değişiklik gösterebilir; örneğin, bazı sistemler uzun dosya isimlerini desteklerken, bazıları belirli bir karakter sınırına sahiptir. Dosyalar oluşturulduktan sonra, üzerlerine veri yazılarak güncellenebilir veya okunarak içeriği kullanılabilir. Oluşturulan her hangi bir dosyaya veri yazmak (çıkış) ya da veri okumak (giriş) için disk erişimi gereklidir.

Bütün dosya işlemleri tamamen işletim sistemi tarafından yapılmaktadır. İşletim sistemi, kullanıcıların bu dosyaları yönetmesine olanak tanır ve bir program olarak, çeşitli işlevlerin birbirlerini çağırması biçiminde çalışır. Örneğin, komut satırında bir programın adının yazılarak çalıştırılması, aslında birden fazla sistem işlevinin sırasıyla çağrılması ile gerçekleşir. Bu süreçte, komut satırından girilen komutu alan, diskte dosyayı arayan, onu belleğe yükleyen ve bellekteki programı çalıştıran işlevler işletim sistemi tarafından düzenli olarak devreye girer.

Tüm dosya işlemleri, hangi programlama dili kullanılırsa kullanılsın, temelde işletim sisteminin sistem işlevleri aracılığıyla gerçekleştirilir. Windows işletim sisteminde bu işlevler **API (Application Programming Interface)** olarak adlandırılırken, UNIX benzeri işletim sistemlerinde **sistem çağrıları (system calls)** terimi kullanılır.

C dilinde, dosya işlemleri için doğrudan bir deyim bulunmaz. Bunun yerine, işlemler ya kütüphanede yer alan fonksiyonlar kullanılarak ya da sistem çağrıları aracılığıyla gerçekleştirilir. Bu yöntemler, programcıya dosyalama süreçlerinde esneklik sağlarken, kullanılan G/Ç yönteminin yapısına göre farklı performans ve kontrol düzeyleri sunar.

C programlama dilinde, disk dosyalarına erişim iki farklı yöntemle gerçekleştirilir:

- **üst düzey G/Ç (high level I/O) veya tamponlanmış G/Ç (buffered I/O) ve**
- **alt düzey G/Ç (low level I/O) ya da UNIX benzeri G/Ç (UNIX like I/O).**

Dosya erişim işlemleri

Üst düzey giriş/çıkış işlemleri kolaylık ve performans sunarken, alt düzey giriş/çıkış işlemleri düşük seviyede daha fazla kontrol sağlar. Hangisinin kullanılacağı, uygulamanın ihtiyaçlarına ve işlem yapılacak veri büyüklüğüne ve içeriğine bağlıdır.

Üst Düzey Giriş/Çıkış İşlemleri

Üst düzey giriş/çıkış işlemleri, tamponlanmış G/Ç (buffered I/O) olarak da adlandırılır ve genellikle daha kullanıcı dostu bir yöntemdir. Bu yaklaşımda, standart C kütüphanesinde yer alan fonksiyonlar kullanılarak işlem yapılır. Örneğin, fopen, fprintf, fscanf, fgets, fputc gibi fonksiyonlar üst düzey G/Ç işlemlerinde kullanılır. Üst düzey G/Ç, tamponlama mekanizması sayesinde, verileri doğrudan diske yazmak yerine önce bellekteki bir tamponda saklar ve belirli bir doluluğa ulaşıldığında topluca diske yazar. Bu yöntem, diske erişim sayısını azalttığı için daha verimli çalışır ve genellikle performans açısından avantaj sağlar.

Tamponlama, aynı zamanda veri güvenilirliğini de artırır. Örneğin, bir dosyaya yazma işlemi sırasında bir hata oluşursa, tamponda tutulan veriler kaybolmaz ve kullanıcıya hata kontrolü yapma fırsatı tanır. Üst düzey giriş/çıkış işlemleri, platformdan bağımsız çalışacak uygulamalar geliştirmek için idealdir.

Alt Düzey Giriş/Çıkış İşlemleri

Alt düzey giriş/çıkış işlemleri, tamponsuz G/Ç (unbuffered I/O) veya UNIX benzeri G/Ç (UNIX-like I/O) olarak bilinir ve işletim sistemi seviyesinde daha doğrudan bir kontrol sağlar. Bu yöntem, C programlama dilinde standart kütüphaneye bağımlı kalmadan, işletim sisteminin sağladığı sistem çağrıları (system calls) ile dosya işlemlerini gerçekleştirir. Örneğin, dosyaları açmak için **open**, dosyalardan veri okumak için **read**, veri yazmak için **write** ve dosyayı kapatmak için **close** fonksiyonları kullanılır.

En önemlisi alt düzey giriş/çıkış işlemleri bir programcının sistem kaynaklarını yönetmede daha fazla sorumluluk almasını gerektirir ve hata ayıklama sürecini üst düzey G/Ç'ye göre daha karmaşık hale getirebilir. Bu nedenle, genellikle daha fazla performans ve kontrol gerektiren durumlarda tercih edilir.

Dosya Türleri

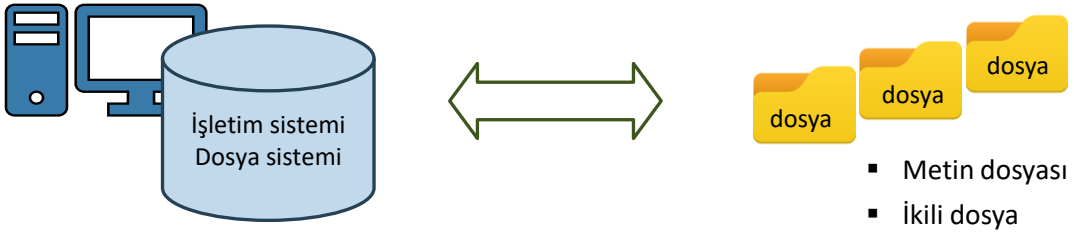
Bilgisayarda kullanılan dosya çeşitleri çok farklı olmakla beraber dosyalar içerik bakımdan genel olarak **metin dosyası (text file)** ve **ikili dosya (binary file)** olarak iki ana türde sınıflandırılabilir.

Metin dosyaları

Genellikle insan tarafından okunabilen ve düzenlenebilen dosya türleridir. Bu dosyalar, yazı veya metin biçimindeki verileri ASCII veya Unicode gibi karakter kodlama sistemlerini kullanarak saklar. Metin dosyaları, düz metin (plain text) ya da yapılandırılmış metin (structured text) biçiminde olabilir. Düz metin dosyaları yalnızca karakterlerden oluşurken, yapılandırılmış metin dosyaları verileri belirli bir format veya düzenle (örneğin, CSV veya JSON) saklar. Metin dosyalarının en büyük avantajı, küçük boyutlu olmaları ve hemen hemen her cihazda kolayca açılabilmeleridir. Bunun yanı sıra, basit bir düzenleme gerektirdiği için herhangi bir metin düzenleyicisi kullanılarak oluşturulabilir ve düzenlenebilir.

İkili dosyalar

Verilerin ikili (binary) formatta, yani 0 ve 1'lerden oluşan bir yapıda saklandığı dosyalardır. Bu dosyalar, makine tarafından okunabilir ve genellikle insanlar için doğrudan anlaşılır değildir. İkili dosyalar, verinin sıkıştırılmamış veya sıkıştırılmış formda saklanması için daha kompakt bir yapı sağlar ve bu nedenle grafik, ses, video, yürütülebilir dosyalar gibi geniş bir kullanım alanına sahiptir. Bu dosya türünde, veri sıkça ham bit (raw bit) seviyesinde saklanır, bu da daha hızlı okuma ve yazma işlemlerine olanak tanır. İkili dosyaların işlenmesi genellikle bir programlama dilindeki özel kütüphaneler veya fonksiyonlarla gerçekleştirilir.



Dosya İşlemleri

Dosya işlemleri dosyanın içeriği üzerinde olduğu gibi bulunduğu yer ve özellikleriyle ilgili işlemler de olabilmektedir. Dosya içeriği ile ilgili işlemler veri okumak, yazmak, düzenlemek ve yönetmek gibi çeşitli adımları içerir. Yer ve özellikleriyle ilgili işlemler ise dosya silme, kopyalama, taşıma ve erişim izinleri gibi adımlar sayılabilir.

Dosya içeriği üzerine yapılan işlemler:

Dosya Oluşturma: Yeni bir dosya oluşturulması, genellikle veri kaydedilecek bir alan sağlamak için yapılır. Bu işlemde dosyanın adı ve türü belirlenir.

Dosya Açma: Dosya açma işlemi, dosya üzerinde işlem yapmaya başlamak için yapılır. Dosya, okuma (giriş), yazma (çıkış) veya ekleme (append) gibi çeşitli modlarda açılabilir.

Dosya Kapatma: Dosya kapama işlemi, dosya ile yapılan tüm işlemler tamamlandıktan sonra yapılır. Dosya kapatıldığında, işletim sistemi dosyanın kaynaklarını serbest bırakır ve değişiklikler kaydedilir. Bu işlem, dosya işlemlerinin sonlandırılması için gereklidir.

Dosya Okuma: Dosya okuma, bir dosyanın içeriğinin program tarafından alınması işlemidir. Bu işlemde, dosya belirli bir formatta okunur ve içerik işlenmek üzere belleğe aktarılır.

Dosya Yazma: Dosya yazma, yeni veri eklemek veya mevcut veriyi değiştirmek amacıyla yapılan işlemdir. Yazma işlemi, dosyanın üzerine yazmak veya dosyaya yeni veri eklemek şeklinde olabilir.

Dosya yer ve özellikleri üzerine yapılan işlemler:

Dosya Silme: Dosya silme işlemi, bir dosyanın sistemden tamamen kaldırılmasını sağlar. Bu işlem, artık kullanılmayan veya geçersiz dosyaları sistemden temizlemek için yapılır.

Dosya Kopyalama ve Taşıma: Dosya kopyalama, bir dosyanın içeriğini başka bir yere çoğaltarak bir kopyasını oluşturmak anlamına gelir. Taşıma ise, dosyayı bir konumdan başka bir konuma fiziksel olarak yer değiştirme işlemidir.

Dosya İzinlerini Değiştirme: Dosya izinlerini değiştirme, dosyanın kimler tarafından okunabileceği, yazılabileceği veya çalıştırılabileceği gibi erişim haklarının belirlenmesi işlemidir ve güvenlik için önemlidir.

Dosyanın açılması – sistem

Bir dosyanın açılması sırasında, dosya ile ilgili ilk işlemler işletim sistemi tarafından gerçekleştirilir. Bu süreç, dosyanın yönetimi ve izlenmesi için gerekli bilgilerin hazırlanmasını içerir. İşletim sistemi, dosyaya erişim sağlandığında, dosya ile ilgili tüm temel bilgileri **"Dosya Tablosu" (File Table)** adı verilen özel bir yapıya kaydeder. Dosya tablosu, işletim sisteminin içinde yer alan ve dosyaların durumu, konumu ve özelliklerini takip eden bir veri yapısıdır. Her dosya açıldığında, bu tabloya dosyayla ilgili yeni bir giriş eklenir ve bu giriş, dosya kapatılana kadar kullanılmaya devam eder.

Dosya tablosunun biçimi, kullanılan işletim sistemine bağlı olarak değişiklik gösterebilir. Ancak genellikle bu tabloda dosya adı, dosyanın fiziksel konumu (disk üzerindeki başlangıç noktası), dosya boyutu, erişim hakları, açma modu (okuma, yazma vb.) ve dosya ile ilişkili işaretçiler gibi bilgiler bulunur. Örneğin, bir dosya tablosu aşağıdaki gibi düzenlenebilir:

Sıra No	Dosya ismi	Dosya Konumu	Dosyanın Özellikleri	Erişim Hakları
0				
1				
.				
12	AUTOEXEC.BAT	...	...	...
.				

İşletim sisteminin sistem işlevleri, herhangi bir programlama fonksiyonu gibi parametre değişkenleri ve geri dönüş değerlerine sahiptir. Örneğin, bir dosyayı açmak için kullanılan **"Dosya Aç"** sistem işlevi, açılacak dosyanın ismini bir parametre olarak alır. Bu işlev, dosya başarıyla açıldığında, dosya tablosunda o dosya için ayrılmış bilgilerin yer aldığı sıra numarasıyla geri döner. Bu numara, **"file handle"** olarak adlandırılır ve dosya üzerinde yapılacak sonraki işlemlerde, o dosyayı tanımlamak için bir referans olarak kullanılır. Örneğin, dosya üzerinde okuma, yazma veya kapatma gibi işlemler yapılırken, bu **"file handle"** değeri diğer dosya işlevlerine parametre olarak iletilir.

Dosya Konum Göstericisi

Bir dosya, ardışık baytların bir araya gelmesinden oluşur. Dosyadaki her bayta, sıfırdan başlayarak artan bir sıra numarası atanır. Bu numaraya, **baytın ofset numarası** denir. **Dosya konum göstericisi** ise, işletim sistemi tarafından dahili olarak tutulan ve bir **long türü** değişken olarak temsil edilen bir değerdir. Bu gösterici, dosyada hangi ofsetten (yani hangi bayttan) işlem yapılacağını belirler. Dosya okuma ve yazma işlemleri, her zaman dosya konum göstericisinin işaret ettiği konumdan başlar. Bu nedenle, standart okuma ve yazma işlevleri, verinin dosyada nereye yazılacağı veya nereden okunacağı bilgisini ayrı bir parametre olarak istemez.

Örneğin, dosya konum göstericisi 100. ofseti işaret ediyorsa ve 10 bayt okunmak isteniyorsa, işlem 100. ofsetten başlar ve 10 bayt ilerler. Dosya üzerinde belirli bir konumdan işlem yapmak gerektiğinde, dosya konum göstericisinin uygun bir konuma ayarlanması gerekir. İşletim sistemi, dosya konum göstericisini güncellemek için özel bir sistem işlevi sağlar. Bir dosya ilk açıldığında, gösterici varsayılan olarak 0. ofseti (dosyanın başını) işaret eder.

Eğer bir dosyanın 100. baytından itibaren 10 bayt okunmak isteniyorsa, şu adımlar izlenir:

1. Dosya açılır
2. Dosya konum göstericisi, 100. ofsete ayarlanır
3. Dosyadan 10 bayt okunur
4. İşlem tamamlandığında Dosya kapatılır.

Modern işletim sistemlerinde dosya konum göstericisi, dosyada okuma veya yazma işlemlerinin başladığı yeri belirtir ve bu gösterici dosya işlemleri sırasında otomatik olarak güncellenir. Ancak, konumlandırma işlevlerinin adlandırılması ve davranışlarında bazı farklılıklar olabilir. Örneğin, UNIX tabanlı sistemlerde **lseek** işlevi kullanılarak dosya konum göstericisi ayarlanırken, Windows sistemlerinde aynı işlev **SetFilePointer** veya **SetFilePointerEx** olarak adlandırılır.

Dosyanın açılması – C dilinde

Bir dosyaya okuma veya yazma yapmak (erişmek) için onun **açılması** gerekir. Bu işlem için **fopen()** fonksiyonu kullanılır. Bu fonksiyon, belirli bir dosyayı açmak ve bu dosya üzerinde işlem yapabilmek için kullanılır. fopen, dosya adını ve açma modunu belirten iki parametre alır ve başarılı bir açma işlemi yapıldığında, dosya ile etkileşime geçmek için bir dosya işaretçisi (file pointer) döner. Eğer dosya açılamazsa, fopen fonksiyonu NULL döner. Yazım düzeni şöyledir.

```
FILE *fopen(char dosya ismi, char mode);
```

```c
#include <stdio.h>
int main() {
    FILE *file = fopen("C:\\source\\example.txt", "r"); // Dosyayı okuma modunda aç
    if (file == NULL) {
        printf("Dosya açılamadı.\n");
    } else {
        printf("Dosya başarıyla açıldı.\n");
        fclose(file); // Dosyayı kapat
    }
    return 0;
}
```

Yaygın Olarak Kullanılan Modlar:

- "r": Dosya yalnızca okuma amacıyla açılır. Dosya mevcut olmalı, aksi takdirde NULL döner.
- "w": Dosya yalnızca yazma amacıyla açılır. Eğer dosya mevcutsa, içerikleri silinir. Dosya mevcut değilse, yeni bir dosya oluşturulur.
- "a": Dosya ekleme amacıyla açılır. Dosya varsa, veriler dosyanın sonuna eklenir. Dosya yoksa, yeni bir dosya oluşturulur.

Dosyanın açılması – devam

Dosya modların ayrıca **"+"** sembolü **"r"**, **"w"** ve **"a"** modlarıyla birlikte kullanılabilmektedir. Burada **"+"** sembolü dosyanın hem okuma hem yazma amacıyla açıldığını belirtmektedir. **"r+"** şeklindeki dosya modu kullanımında, dosya esas olarak okuma modunda açılmasına rağmen **"+"** sembolünün kullanılması sebebiyle aynı zamanda dosyaya yazma işlemi de yapılabilmektedir.

Metin ve ikili tipte dosyayı destekleyen işletim sistemlerinde **"r"**, **"w"** ve **"a"** ya da **"r+"**, **"w+"** ve **"a+"** modlarının sonuna **"b"** harfi eklenerek, işlem yapılacak dosyanın ikili (binary) tipte olacağı belirtilebilir. Bu durumda modların kullanımı **"rb"** ya da **"r+b"** şeklinde belirtilebilir.

```
FILE *file = fopen("program.exe", "rb");      // İkili dosyayı okuma modunda aç
FILE *file = fopen("calisma.dat", "w+b");   // İkili dosyayı okuma/yazma modunda aç
```

Dosya işlemleri esnasında işlem yapılacak dosya çeşitli sebeplerden dolayı açılamayabilir. Bu durumda **fopen** işlevi NULL adresine geri döner. İşlevin geri dönüş değeri kesinlikle kontrol edilmelidir. Tipik bir sınama işlemi aşağıdaki gibi yapılabilir:

```
/* ön ayarlamalar */
FILE *f;
if ((f = fopen("mektup.txt", "r")) == NULL) {
    printf("Dosya açılamadı...\n");
    exit(EXIT_FAILURE);
}
```

Yukarıdaki örnekte ismi mektup.txt olan bir dosya okuma amacıyla açılmaya çalışılıyor. Dosya açılmaz ise ekrana bir hata iletisi verilerek, standart **exit** işlevi ile program sonlandırılıyor. Şüphesiz fopen işlevi ile açılmak istenen bir dosyanın açılamaması durumunda programın sonlandırılması zorunlu değildir.

Dosyanın kapatılması

Her hangi bir işlem için **fopen** fonksiyonu ile açılmış bir dosya **fclose** fonksiyonu ile güvenli bir şekilde kapatılır. Bir dosya üzerinde yapılan işlemler tamamlandıktan sonra, fclose fonksiyonunun çağrılması, dosyanın sistem kaynaklarından serbest bırakılmasını ve değişikliklerin dosyaya kaydedilmesini sağlar. Fonksiyonun geri dönüş değeri int türü bir tamsayıdır. "0" değeri dosyanın başarıyla kapatıldığını ,"-1 (EOF)" değeri ise her hangi bir kapatma hatası oluştuğunda dönen değerlerdir. Hata durumda hata türü hakkında daha fazla bilgi almak için **errno** değeri kontrol edilebilir. **fclose** fonksiyonunun yazım düzeni şöyledir.

```
fclose(FILE *dosya işaretçisi);
```

```c
#include <stdio.h>
int main() {
    FILE *file = fopen("example.txt", "r"); // Dosyayı okuma modunda aç
    if (file != NULL) {
        printf("DİKKAT: Dosya mevcuttur.\n");
        return 1;
    }

    fprintf(file, "Merhaba, dünya!\n");    // Dosyaya yazma işlemi

    if (fclose(file) == 0) { // Dosyayı kapatma
        printf("Dosya başarıyla kapatıldı.\n");
    } else {
        printf("Dosya kapatma sırasında bir hata oluştu.\n");
    }

    return 0;
}
```

Dosya okuma

Bir program içinde ihtiyaç duyulan veri miktarına ve uygulama gereksinimlerine göre her hangi bir dosyadan tek bir karakter ya da bir karakter dizisi okunma durumu olabilir. C dilinde böyle tekli ya da çoklu karakter okuma işlemleri için ayrı fonksiyonlar oluşturulmuştur. **fgetc** fonksiyonu bir seferde yalnız bir karakter, **fgets** , **fscanf** ve **fread** fonksiyonları tek seferde birden fazla karakter okumak için kullanılmaktadır.

fgetc() fonksiyonu

Dosya işaretçisinin gösterdiği konumdan bir karakter okur ve bu karakteri **int** türünde döndürür. Eğer dosyanın sonuna ulaşılırsa veya bir hata meydana gelirse, **EOF** döner. Basit ve anlaşılır yapısıyla çoğunlukla küçük boyutlu verilerle çalışmak için kullanılır. Yazım düzeni şöyledir.

```
int fgetc(FILE dosya işaretçisi);
```

```c
#include <stdio.h>
int main() {
    FILE *file = fopen("example.txt", "r");
    if (file == NULL) {
        printf("Dosya açılamadı.\n");
        return 1;
    }

    char ch;
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch); // int değerin karakterini ekrana yazdır
    }
    fclose(file);
    return 0;
}
```

fgets() fonksiyonu

Bir dosyadan bir satır veya belirtilen karakter sayısından 1 eksiği kadar okur ve bunu bir karakter dizisine aktarır. Satır sonu karakterini (örneğin, \n) okursa, bu karakter de diziye eklenir. Fonksiyon nispeten hızlıdır, çünkü bir seferde birden fazla karakter okur ve büyük miktarda veri işlemek için oldukça uygundur. Yazım düzeni şöyledir.

```
char fgets(char dizgi_adı, int n, FILE dosya işaretçisi);
```

```
#include <stdio.h>
int main() {
    FILE *file = fopen("example.txt", "r");
    char buffer[100];
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("%s", buffer); // Okunan satır terminal yazılır
    }
    fclose(file);
    return 0;
}
```

Programda metin tipi tampon olarak 100 karakter **"buffer"** isimli bir değişken tanımlanmıştır. **"fgets"** fonksiyonu her defasında dosyadan 100 karakterlik veri okuyup bunu "buffer" değişkenine atar. Sonraki adım okunan metin dizisi terminale yazdırılmaktadır. Ancak dosya sonuna gelindiğinde ya da yeni satır karakteri okunduğunda ya da boş bir değer okuduğundan okuma işlemi sona erer ve dosya kapatılarak program sona erer.

fscanf() fonksiyonu

Bir dosyadan biçimlendirilmiş formatta veri okumak için kullanılır. Konsoldan giriş alan scanf fonksiyonunun dosya işlevine uyarlanmış halidir. Dosyadan biçimlendirilmiş veri okumayı kolaylaştırarak verinin doğrudan değişkenlere atamasına imkan verir. Ancak okuma biçimi dosya içeriğiyle tam eşleşmezse hata meydana gelebilir. Böyle durumlar için ek kontrol mekanizmaları ayarlanması gereklidir. Yazım düzeni şöyledir.

```
int fscanf(FILE dosya işaretçisi, char yazım formatı, değişken listesi);
```

```c
#include <stdio.h>
int main() {
    FILE *file = fopen("data.txt", "r");
    char isim;
    float odeme;
    fscanf(file, "isim: %s\nödeme: %.2f\n", &isim, &odeme);
    printf("Okunan isim: %s", isim);
    printf("Okunan Ödeme: %s", odeme);
    fclose(file);
    return 0;
}
```

fscanf() fonksiyonunda ilk parametre veri okunacak dosyanın dosya işaretçisidir, ikinci parametre okunacak verilerin formatını gösteren metin dizisidir, üçüncü parametre ise okunacak değişkenlerin listesidir.

Yukarıdaki programda "data.txt" dosyasından "İsim: %s\nÖdeme: %.2f\n" şeklindeki okuma düzenine uygun olarak "isim" ve "odeme" değişkenlerine değer atanmaktadır. Daha sonra bu değerler terminale yazılmaktadır.

fread() fonksiyonu

Dosyadan belirli bir boyutta veri okur ve bir bellek bölgesine yazar. Örneğin, karakter dizisi okuma işlemlerinde büyük miktarda veri almak için kullanılabilir. Satır bazlı işlemler için daha az esnek olabilir. Tampon boyutunun dikkatlice ayarlanması gerekir. Yazım düzeni şöyledir.

```
size_t fread(void dizgi_adı, size_t tür, size_t n, FILE dosya işaretçisi);
```

```c
#include <stdio.h>
int main() {
    FILE *file = fopen("example.txt", "r");
    char buffer[50];
    size_t bytesRead = fread(buffer, sizeof(char), sizeof(buffer) - 1, file);
    buffer[bytesRead] = '\0';     // Sonlandırıcı ekle
    printf("Okunan veri: %s\n", buffer);
    fclose(file);
    return 0;
}
```

fread() fonksiyonunda ilk parametre okunacak dizi değişkenin ismidir, ikinci parametre kullanılan veri tipinin büyüklüğüdür (int 2 byte, float 4 byte, char 1 byte), üçüncü parametre okunacak veri sayısıdır, son parametre dosya işaretçisidir.

Yukarıdaki programda "char" veri türünden 50 karakterlik bir tampon değişkeni tanımlanmıştır. fread() fonksiyonu belirtilen dosyadan "buffer" değişkeni büyüklüğü kadar 50 karakterlik veriyi okumaktadır. Daha sonra metin dizisinin sonlandırıcı karakteri en sona eklenmektedir. Ardında okunan veri terminale yazılmaktadır.

Dosya yazma

Okuma işlemlerinde olduğu gibi bir dosyaya yazma için de her hangi bir dosyaya tek bir karakter ya da bir karakter dizisi yazılma durumu olabilmektedir. C dilinde böyle tekli ya da çoklu karakter yazma işlemleri için ayrı fonksiyonlar oluşturulmuştur. **fputc** fonksiyonu bir seferde yalnız bir karakteri, **fputs**, **fprintf** ve **fwrite** fonksiyonları ise tek seferde birden fazla karakteri yazmak için kullanılmaktadır.

fputc() fonksiyonu

Fonksiyona verilen karakteri dosyaya yazar ve başarılı olursa karakterin kendisini, hata durumunda ise **EOF** döner. Basit ve anlaşılır yapısıyla çoğunlukla küçük boyutlu verilerle çalışmak için kullanılır. Yazım düzeni şöyledir.

```
int fputc(int karakter, FILE dosya işaretçisi);
```

```
#include <stdio.h>
int main() {
    FILE *file = fopen("example.txt", "w");
    if (file == NULL) {
        printf("Dosya açılamadı.\n");
        return 1;
    }

    char ch = 'A';
    if (fputc(ch, file)) != EOF) {
        printf("Karakter başarıyla yazıldı.\n");
    } else {
        printf("Karakter yazılamadı.\n");
    }

    fclose(file);
    return 0;
}
```

fputs() fonksiyonu

Bir karakter dizisini (string) dosyaya yazar. Yazma işlemi NULL karakterine kadar devam eder ve başarılı olursa 0, hata durumunda ise EOF döner. Fonksiyon nispeten hızlıdır, çünkü bir seferde birden fazla karakter yazar ve büyük miktarda veri işlemek için oldukça uygundur. Yazım düzeni şöyledir.

```
int fputs(char dizgi_adı, FILE dosya işaretçisi);
```

```
#include <stdio.h>
int main() {
    FILE *file = fopen("example.txt", "r");

    char mesaj[] = "Merhaba, dünya!";

    if (fputs(mesaj, file) != EOF) {
        printf("Metin başarıyla yazıldı.\n");
    } else {
        printf("Metin yazılamadı.\n");
    }

    fclose(file);
    return 0;
}
```

fputs() fonksiyonunda ilk parametre dosyaya yazılacak dizi değişkenin ismidir, ikinci parametre ise dosya işaretçisi-dir. Fonksiyonu geri dönüş parametresi tamsayı türünde bir değerdir.

Yukarıdaki programda "mesaj" metin veri türünden bir metin dizi tanımlanmıştır. fputs() fonksiyonu belirtilen dosyaya "mesaj" verisini yazmaktadır. Eğer yazma işleminde dosya sonuna erişilmemişse yazım başarılı olarak gerçekleştirilir aksi durumda yazma işlemi yapılamaz. Yazma işlemi tamamlandıktan sonra dosya kapatılır.

fprintf() fonksiyonu

Bir dosyaya biçimlendirilmiş formatta veri yazmak için kullanılır. Ekrana çıktı almak için kullanılan printf fonksiyonunun dosya işlevine uyarlanmış halidir. Veriyi kolayca biçimlendirme imkânı vermekte olup çok çeşitli veri türlerini destekler. Ancak biçimlendirme hatalarında yazma işlemi istenmeyen sonuçlara sebep olabilir. Yazım düzeni şöyledir.

```
int frintf(FILE dosya işaretçisi, char yazım formatı, değişken listesi);
```

```
#include <stdio.h>
int main() {
    FILE *file = fopen("data.txt", "w");
    int yas = 25;
    float ucret = 2500.55;
    fprintf(file, "Yaş: %d\nMaaş: %.2f\n", yas, ucret);
    fclose(file);
    return 0;
}
```

fprinf() fonksiyonunda ilk parametre veri yazılacak dosyanın dosya işaretçisidir, ikinci parametre verilerin formatını gösteren metin dizisidir, üçüncü parametre ise yazılacak değişkenlerin listesidir.

Yukarıdaki programda "yas" ve "ucret" değişkenleri ile tanımlanan değerler, fopen fonksiyonu ile belirtilen "data.txt" dosyasına yazdırılmaktadır. Burada verilerin çıktı formatı "fprintf" fonksiyonunda istenilen şekilde ayarlanmıştır. Format metin dizisi içinde tanımlanan yer tutucu sayısı kadar değişken adları format metninden sonra verilmiştir.

fwrite() fonksiyonu

Belirtilen bellek adresinden belirten türde verilen (n) sayı kadar baytlık veri alır ve bunu dosyaya yazar. Yazılan toplam veri miktarını döndürür. Büyük miktarda veri yazmak için oldukça idealdir ve tampon bellek kullanımı sayesinde daha hızlıdır. Ancak tek satır veya karakter bazlı yazma işlemleri için biraz karmaşık olabilir. Yazım düzeni şöyledir.

```
size_t fwrite(void dizgi_adı, size_t tür, size_t n, FILE dosya işaretçisi);
```

```c
#include <stdio.h>
int main() {
    FILE *file = fopen("sonuc.txt", "w");
    char buffer[] = "C programlama dilinde dosya işlemleri.";
    size_t bytesWritten = fwrite(buffer, sizeof(char), sizeof(buffer) - 1, file);
    if (bytesWritten == sizeof(buffer) - 1) {
        printf("Veri başarıyla yazıldı.\n");
    } else {
        printf("Veri yazılırken bir hata oluştu.\n");
    }
    fclose(file);
    return 0;
}
```

fwrite() fonksiyonunda ilk parametre okunacak dizi değişkenin ismidir, ikinci parametre kullanılan veri tipinin büyüklüğüdür (int 2 byte, float 4 byte, char 1 byte), üçüncü parametre okunacak veri sayısıdır, son parametre dosya işaretçisidir.

Yukarıdaki programda "buffer" ile tanımlanan metin dizisi "sonuç.txt" dosyasına bellek kullanılarak yazılmaktadır. Belleğe yazılan byte sayısı (bytesWritten) yazılacak metin dizisinin byte sayısı ile aynıysa yazma işlemi başarılıdır.

Örnek uygulama

Bir uygulama örneği olarak verilen bir dosyayı istenilen isimde kopyasını yazan bir kodu inceleyelim.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define  MAX_FILE_NAME_LEN   256

int main(int argc, char **argv)) {
    FILE *fsource, *fdest;
    char source_name[MAX_FILE_NAME_LEN];
    char dest_name[MAX_FILE_NAME_LEN];
    int ch;
    int byte_counter = 0;

    if (argc != 3) {
        printf("kopyalanacak dosyanın ismi : ");
        gets(source_name);
        printf("kopya dosyanın ismi : ");
        gets (dest_name);
    } else {
        strcpy(source_name, argv[1]);
        strcpy(dest_name, argv[2]);
    }

    if ((fsource = fopen(source_name, "r")) == NULL) {
        printf("%s dosyasi acilamiyor\n", source_name);
        exit(EXIT_FAILURE);
    }
    printf("%s dosyasi acildi!\n", source_name);
```

```c
if ((fdest = fopen(dest_name, "w")) != NULL) {
    printf("%s dosyasi mevcuttur\n", dest_name);
    fclose(fsource);
    exit(EXIT_FAILURE);
}
printf("%s dosyasi yaratildi!\n", dest_name);

while ((ch = fgetc(fsource)) != EOF) {
    fputc(ch, fdest);
    byte_counter++;
}
fclose(fsource);

printf("%s dosyasi kapatildi!\n", source_name);
fclose(fdest);

printf("%s dosyasi kapatildi!\n", dest_name);
printf("%d uzunlugunda %s dosyasinin %s isimli
kopyasi cikarildi!\n", byte_counter, source_name,
dest_name);

return 0;
}
```

Program "kopyala.c" adıyla kaydedilip derlendiğinde

kopyala kaynak_dosya_ismi hedef_dosya_ismi

şeklinde çalıştırılır.

Dosyalarla ilgili bazı fonksiyonlar

Dosyalar üzerinde belirli işlemlerde kullanılan bazı faydalı fonksiyonlar aşağıda listelenmiştir.

fseek() fonksiyonu: Dosya konum göstericisini belirtilen bir konuma ayarlamak için kullanılır. Başarı durumunda 0, hata durumunda -1 değerini geri döner.

```
int fseek(FILE dosya işaretçisi, long ofset, int başlangıç);
```

feof() fonksiyonu: Dosya okuma sırasında dosya sonuna ulaşılıp ulaşılmadığını kontrol eder. Dosya sonuna ulaşıldıysa 1, aksi takdirde 0 değerini geri döner.

```
int feof(FILE dosya işaretçisi);
```

ftell() fonksiyonu: Dosya konum göstericisinin mevcut konumunu bayt cinsinden döner. İşlem başarılı ise konum değeri, aksi takdirde -1 değerini geri döner.

```
long ftell(FILE dosya işaretçisi);
```

rewind() fonksiyonu: Dosya konum göstericisini dosyanın başlangıcına sıfırlar. Geri dönüş değer yoktur.

```
void rewind(FILE dosya işaretçisi);
```

rename() fonksiyonu: Bir dosyanın adını değiştirmek için kullanılır. Başarı durumunda 0, hata durumunda -1 değerini geri döner.

```
int rename(char eski isim, char yeni isim);
```

Çalışma soruları

1. Kullanıcıdan dosya adını aldıktan sonra 1'den 10'a kadar sayıları sırasıyla "Sayı=1", "Sayı=2" şeklinde dosyaya yazdıran C kodunu hazırlayınız.

2. 6-7 satırdan oluşan ve her bir satır kişi isimleri olan "isimler.txt" dosyasından bu isimleri sırayla okutup terminale yazdıran kodu hazırlayınız.

3. Kullanıcıdan bir döngü içinde aldığı ad ve yaş bilgilerini **"İsim: --, Yaş: --"** yazım düzeni ile "kişiler.txt" dosyasına yazdıran kodu hazırlayınız. Döngüden çıkış boş karakter ile sağlanacaktır.