# Temel Dosya İşlemleri

# Dosyaların Temel İşlemleri

- Bilgiler dosyada belirli bir düzen içerisinde yer alırlar. Örn:

**ALAN** **THARP** ←
100 100 100 ←
**JOHN** **BISHOP** ←
70 80 75 ←
**PAUL** **AUSTER** ←
40 100 70 ←
**MICHEAL** **IRON** ←
90 90 90 ←
**JEAN** **HILL** ←
30 60 45 ←

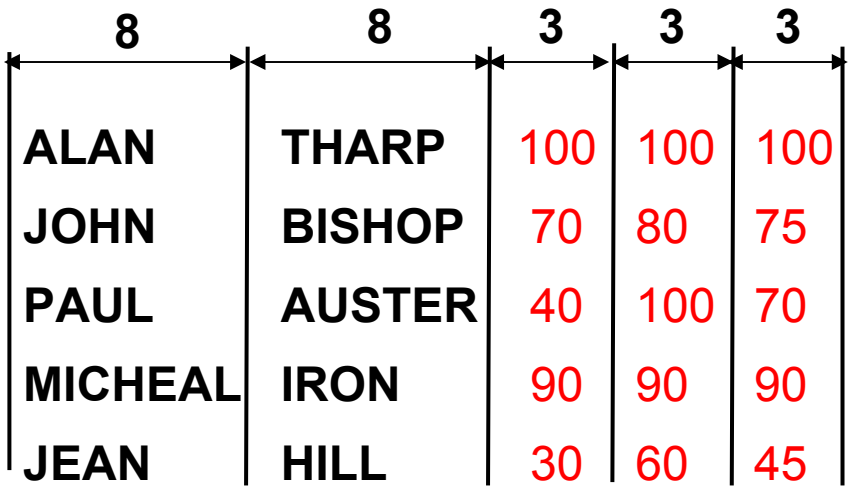
Bir satırda olabilecek karakter sayısı konusunda bir sınırlama yoktur.

Satır bitişini ← karakteri gösterir.

Bu tarz bir dosyada 3.kişi hakkında bilgi edinmek istenirse kendisinden önceki ilk 2 kaydı da mutlaka okumak gerekir (veya bu kayıtlar önceden okunmuş olmalıdır)

Dosyadaki belirli bir kayda ulaşmak için dosyanın başından başlayarak istenilen kayda doğru okumanın sırasıyla yapılması gerekir.

Bundan dolayı, bu tür dosyalara sıralı erişimli dosyalar (*sequential files*) denir. Bu tür dosyalar ayrıca text dosyalar olarak da bilinir.




Her bir kayıt 25 karakterden oluşmaktadır.

Kayıt uzunluğu sabit olduğu için böyle bir dosyada herhangi bir anda herhangi bir kayda ulaşmak mümkündür. Örneğin 4. kaydı öğrenmek için baştan itibaren 76. karakterden başlayarak kaydı okumak mümkündür. Bu tür dosyalara rastgele erişimli (Random access files) denir.

# Dosyalarla İlgili Komutlar

Dosyalarla ilgili komutları 6 grup altında toplamak mümkündür. Bunlar :

- Hata kontrol komutları
- Dosyanın açılması
- Dosyanın kapatılması
- Dosyaya Yazma
- Dosya Okuma
- Diğer Dosya Kullanım Komutları

# Bilgisayarın Dosyalara Erişimi

İşlemler buffer denilen bir tampon üzerinde gerçekleştirilir.

Buffer, bellek üzerinde belli bir uzunlukta olan byte dizisidir. (Aksi belirtilmedikçe uzunluğu 512 byte'tır)

- Dosya okumak için açıldığında ilk 512 byte'ı bu tampon belleğe yerleştirilir.
- Yazma işleminde ise yazılan karakterler tampon bellekte tutulur. Tampon bellek dolduğunda tüm bilgiler diske yazılır.

Not: Bir dosyaya erişim yapılırken, yazılacakların hangi satır ve sütuna gerçekleştirileceğini belirleyen **K**ayıt **İ**şaretçisi (**Kİ**) kullanılır.

C’de, standart bir dosya tipi tanımlanmıştır. Ancak stdio.h başlık dosyası içinde FILE yapısal veri tipi aşağıdaki gibidir.

```
typedef struct
    {
        short level;
        unsigned flags;
        char fd;
        unsigned char hold;
        short bsize;
        unsigned char *buffer;
        unsigned char *curp;
        unsigned istemp;
        short token;
} FILE;
```

Bu yapısal veri tipi kullanılarak:

**FILE *A_dosyası, *B_dosyası;**

tanımlanır.

```
typedef struct
    {
     short level;
     unsigned flags;
     char fd;
     unsigned char hold;
     short bsize;
     unsigned char *buffer;
     unsigned char *curp;
     unsigned istemp;
     short token;
} FILE;
```


Tampon belleğin dolu olup olmadığını kontrol eder.


Dosyanın durumunu gösterir.


Dosya belirticisidir.


Ungetch() fonksiyonu için ayrılmış bir tamsayıdır.


Tampon belleğin uzunluğunu verir. (Default: 512 byte)


Tampon belleğin başlangıç adresini verir.


Tampon bellek üzerinde okunacak yada yazılacak karakterlerin pozisyonunu belirtir.


Geçici dosya göstericisidir.


Kontrol için kullanılır.

# Standart Dosyalar

Bilgisayara veri giriş-çıkışı için kullanılan klavye ekran, printer ve portları da birer dosya olarak tanımlamak mümkündür. Bu cihazlar sıralı erişimli dosya gibi davranırlar.

Klavye → stdin
Ekran → stdout
Printer → stdprn
Yrd. Port→ stdaux

Ayrıca hata mesajlarının yazıldığı bir çıkış dosyası vardır. Bu da stderr'dir.

Bu dosyalar programın başında otomatik olarak tanımlanıp kullanım için açılırlar.

# Dosyalarla İlgili İşlemler ve Hata Kontrolü

3 tane hata yakalama fonksiyonu verilmiştir.

**feof**

Dosya okumak için açıldığında dosya sonuna ulaşılıp ulaşılmadığını kontol etmek amacıyla kullanılır.

Prototipi :

```
İnt feof (FILE *dosya);
```

Geri dönen değer 0 ise dosya sonuna ulaşılmamıştır.

## ferror

Dosyalarla ilgili daha genel hataların olup olmadığının belirlenmesi amacıyla kullanılır.

Prototipi :

```
int ferror (FILE *dosya);
```

Geri dönen değer 0 ise dosyada hata yok demektir.

## clearerr

Hata kontrolü yapıldıktan sonra, hatalı durum ortadan kalktığında, daha sonraki kontrollerin doğru yapılabilmesi için, önceki hataların sıfırlanması gerekir. Bunu da clearerr fonksiyonu sağlar.

Prototipi :

```
void clearerr (FILE *dosya);
```

# Dosya Oluşturma ve Dosya Açma

**fopen ( )**

Prototipi

```
FILE *fopen (char *dosya_adi, char *mod)
```

Mod seçeneği dosyanın açılış şeklini belirler.

**'r'** → Varolan bir dosyanın sadece okumak için açılmasını sağlar.

**'w'**→ Yazmak için yeni bir dosya oluşturur. Dosya daha önceden mevcut ise silinerek yazılmak için yeniden açılır.

**'a'** → Varolan dosyanın sonuna yeni bilgiler eklemek için kullanılır. Dosya mevcut değilse yeni bir dosya açılır.

## fopen ( )

Prototipi

```
FILE *fopen (char *dosya_adi, char *mod)
```

Mod seçeneği dosyanın açılış şeklini belirler.

**"r+"** → Varolan bir dosyayı üzerinde hem okuma hem de yazma yapmak için açar.

**"w+"**→Dosyayı okumak ve yazmak için yeni bir dosya oluşturur.

**"a+"** → Okumak ve yazmak için yeni bir dosya oluşturur ve en son pozisyonda bekler.

Bu modlara ilave olarak '**t**' ve '**b**' karakterleri de ilave edilir.

**'t'** → Dosyanın text olduğunu belirtir. İlave edilmezse dosya default text olarak işlem görür.

**'b'** →Dosyanın doğrudan binary düzeyde işlendiğini belirtir.

fopen( ), kendisini çağırana bir adres gönderir. Dosya açılamazsa NULL; açılırsa dosya bilgilerinin saklanıdığı topluluğun başlangıç adresini gönderir. Bu adres, daha sonraki erişimlerde kullanılacağı için, işaretçi olarak tanımlanmış bir değişkene atanır.

```
# include <stdio.h>
.
.
FILE *di ;  /*Dosya işaretçisi */
if ( (di=fopen("dosya.txt","w")) ==NULL)
 {
  puts("Dosya açılamadı\n");
 }
.
.
.
```

# Dosya Kapatma Fonksiyonları

Dosyanın kapatılması, dosya tampon belleğinin diske kaydedilmesi ve dosya ile ilişkinin kesilmesi demektir. 2 farklı şekilde dosya kapatılabilr;

## i) flush

Yazmak için açılan dosyanın tampon belleğindeki bilgileri diske aktarır. Okumak için açılan dosyalarla bir ilgisi yoktur. **Fonksiyondan sonra dosya ile olan ilişki hala devam eder.**

Prototipi **a)**

```
int flush (FILE *dosya);
```

Geri dönen değer 0 ise işlem başarılmıştır.

**b)**

```
int flushall (void);
```

Açılmış bütün dosyaların tampon belleklerini diske aktarır.

## ii) fclose

Okumak ve yazmak için açılmış herhangi bir dosyanın kapatılmasını sağlar.

Prototipi **a)** `int fclose (FILE *dosya) ;`

Geri dönen değer 0 ise işlem başarılmıştır.

**b)** `int fcloseall (void);`

Açık olan tüm dosyaları kapatır.

# Örnek

```
#include <stdio.h>
int main ()
FILE *di=fopen(“deneme.txt”,”r”);
char c;
While (!feof(di)) {
.....
.....
.....
}
fclose (di);
return 0;
}
```

Kayıtlar üzerinde işlemler

# Dosyaya Karakter Düzeyinde Erişmek

### i) fputc( )

Dosyaya tek bir karakter yazmak için kullanılır. Prototipi ;

```
int fputc (int c, FILE *dosya) ;
```

Dosyaya yazılan bu karaktere ait ASCII kodudur. Yazma işleminin hatasız yapılması halinde fputc, yazılan karakterin ASCII kodunu geri gönderir.

### ii) putc( )

fputc fonksiyonu ile aynıdır. Prototipi;

```
int putc (int c, FILE *dosya) ;
```

### iii) fgetc( )

Dosyadan bir karakter okumak için kullanılır. Prototipi;

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

Okunan karakter bir tamsayı olarak geri dönmektedir. Bu sayı 0-255 arasındadır. 0'dan küçük olması durumunda dosya okumada bir hata vardır.

### iv) getc( )

fgetc fonksiyonu ile aynıdır.

```
int getc (FILE *dosya) ;
```

Not: Yazma işlemleri  Kayıt işaretçisinin (Kİ) konumlandığı yere göre yapılır.

# Örnek

```
# include <stdio.h>
FILE *di;
main ()
{ FILE *di;
char kr;
if ( (di=fopen(“deneme”,”w”))==NULL)
{ printf (“Dosya açılamadı \n”);
}
while ((kr=getchar())!=‘q’)
putc(kr,di);
fclose(di);
}
```

Klavyeden girilen karakter q olmadığı müddetçe klavyeden aldığı değeri deneme.txt isimli dosyaya yazacaktır.

# Örnek

```
# include <stdio.h>
FILE *di;
main ()
{ FILE *di;
char kr;
if ( (di=fopen(“deneme”,”w”))==NULL)
{ printf (“Dosya açılamadı \n”);
}
while (!feof(di)) {
kr=getc(di)
putchar(kr);
fclose(di);
}
```

Dosyadan karakterler okunmakta ve ekrana yazılmaktadır.

Dosya sonuna gelinip gelinmediğinin sınanması 2 farklı şekilde gerçekleştirilebilir :

i) while (!feof(di))
{.....
.....
}

ii) while ( ( kr=getchar() ) !=EOF)
{.....
.....
}

# Dosyaya Karakter Katarı Düzeyinde Erişmek

### i) fputs( )

Dosyaya karakter dizisi yazmak için kullanılır. Prototipi ;

```
int fputs (char *s, FILE *dosya) ;
```

s karakter dizisini dosya ile belirtilen yere yazar. Karakter dizisi \0 ile son bulmasına rağmen, bu karakter dosyaya yazılmaz. Başarılı bir yazma işlemi sonucunda 0 değeri geri döner.

## ii) fgets( )

Dosyadan karakter dizisi okumak için kullanılır. Prototipi ;

```
char *fgets (char *DIZI,int N, FILE *dosya);
```

**DIZI ;** okumanın yapılacağı dizinin başlangıç adresidir.

**N ;** dizinin maksimum uzunluğudur.

**Not :** Okuma esnasında satır sonu ile karşılaşılırsa dizi sonuna \0 karakteri ilave edilerek okuma tamamlanır. Geri dönen pointerın NULL olması durumunda dosyada okunacak başka bilgi kalmamıştır. Başarılı olarak okuma yaptığında okunanları yerleştirği katar değişkenin başlangıç adresini gönderir.

# Örnek

```
#include <stdio.h>
main ()
{
.
.
FILE *di;
char ad[10];
.
.
gets (ad);
.
.
fputs(ad,di);
.
.
.
}
```

Yazma işlemi, ayrılan yer kadar değil, NULL karakteri ile karşılaşılana kadar yapılır.

# Örnek

```
# include <stdio.h>
main()
{
.
.
FILE *di;
int n;
char ad[10];
.
.
while          (fgets(ad,n,di)
    !=NULL)
puts(ad);
.
.
}
```

En falza n -1 karakter okuyup, ad dizi değişkenine yerleştirir. Dönen değer NULL ise dosya sonuna gelinmiştir.

# Dosyalara Formatlı Erişim

**i) fprintf( )**

printf fonksiyonuna benzemektedir. Prototipi:

```
int printf (FILE *dosya,const char *format[,arguman listesi]);
```

Geri dönüş değeri dosyaya yazılan karakter sayısıdır.

**ii) fscanf( )**

scanf fonksiyonuna benzemektedir. Prototipi ;

```
int scanf (FILE *dosya,const char *format,degisken adresleri listesi);
```

Geri dönen değer ise, kaç tane değişken okunduğunu gösterir.

# Dosyalara Toplu Erişim

### i) fwrite( )

Belleğin belli bir bölümünü bir defade dosyaya yazmak amacıyla kullanılır. Prototipi;

```
int fwrite(void *Bellek,int UZUNLUK, int N, FILE *dosya)
```

Bu fonksiyon, BELLEK adresinden başlayan ve herbirinin uzunluğu UZUNLUK kadar olan N adet veriyi dosyaya aktarır.

### ii) fread( )

Bilgiyi verilen bir alana transfer eder. Prototipi;

```
fread(void *Bellek,size_t UZUNLUK, size_T ADET, FILE *dosya)
```

BELLEK okuma yapılacak bellek bölümünün başlangıç adresidir. Bu bölümün uzunluğu UZUNLUK x ADET kadardır.

# Örnek

```
#include<stdio.h>
main()
{
.
.
FILE *di;
İnt dizi[ ]={10,15,18,.....};
int tane;
.
fwrite(dizi,sizeof(int),n,di);
.
.
}
```

```
#include<stdio.h>
main()
{
.
.
FILE *di;
int dizi[ ]
.
fread(dizi,sizeof(int),10,di);
.
.
}
```

# Diğer Kullanım Fonksiyonları

### i) ftell( )

Dosya göstericisinin pozisyonunu belirlemek için kullanılanılır. Prototipi

```
long ftell (FILE *dosya)
```

Dosya göstericisinin pozisyonunun kaçıncı byte'ı gösterdiğini dönüş bilgisi olrak verir.

### ii) fseek( )

Dosya göstericisinin pozisyonunu değiştirmek için kullanılanılır. Prototipi

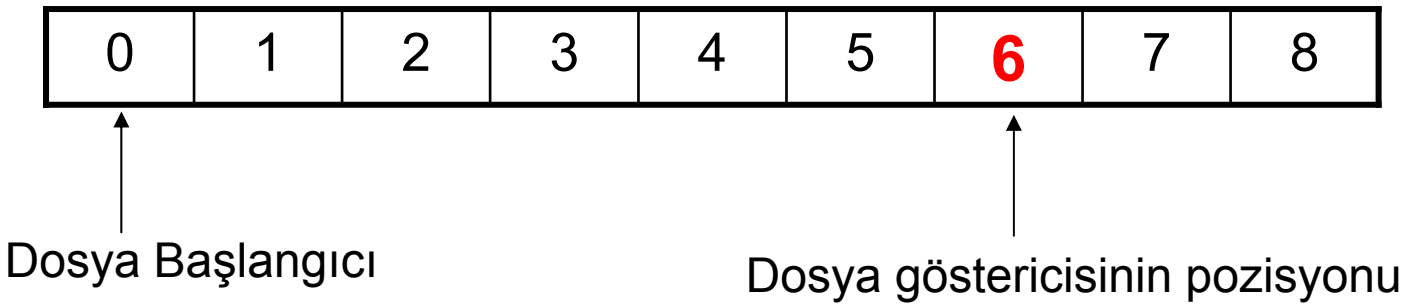
```
int fseek (FILE *dosya, long Konum, int Baslama_sekli )
```

Konum ; Dosya göstericisinin hangi byte'ı göstereceğini belirleyen değerdir.

Baslama_sekli ; Belirlenen başlangıç byte'ıdır. 3 değeri mevcuttur.

a) **SEEK_SET**; hesaplamanın dosyanın başından itibaren yapılmasını sağlar.

**Örn:** fseek(veri,7,SEEK_SET)




b) **SEEK_END**; hesaplamanın dosyanın sonundan itibaren yapılmasını sağlar.

b) **SEEK_CUR**; hesaplamanın dosya işaretçisinin bulunduğu konumdan itibaren yapılmasını sağlar. Konumun 0 olması dosya işaretçisinin konumunu değiştirmez. 0'dan büyük olması durumunda ileri, küçük olması durumunda geriye doğru sıçrar.

### iii) Rewind( )

Dosya göstericisini dosyanın ilk byte'ına getirmek amacıyla kullanılır. Prototipi:

```
void rewind (FILE *dosya)
```

### iv) fgetpos( )

Dosya gösterici pozisyonun değerini pozisyon değişkenine yerleştirir. Prototipi

```
int fgetpos (FILE *dosya, fpos_t  *pozisyon)
```

fpos_t tipi stdio.h dosyasında tanımlanmıştır.

### v) fsetpos( )

Dosya değişkeninin belirttiği dosyanın pozisyon ile verilen byte'ına ulaşmak için kullanılır.  Prototipi:

```
int fsetpos (FILE *dosya, const fpos_t *pozisyon)
```