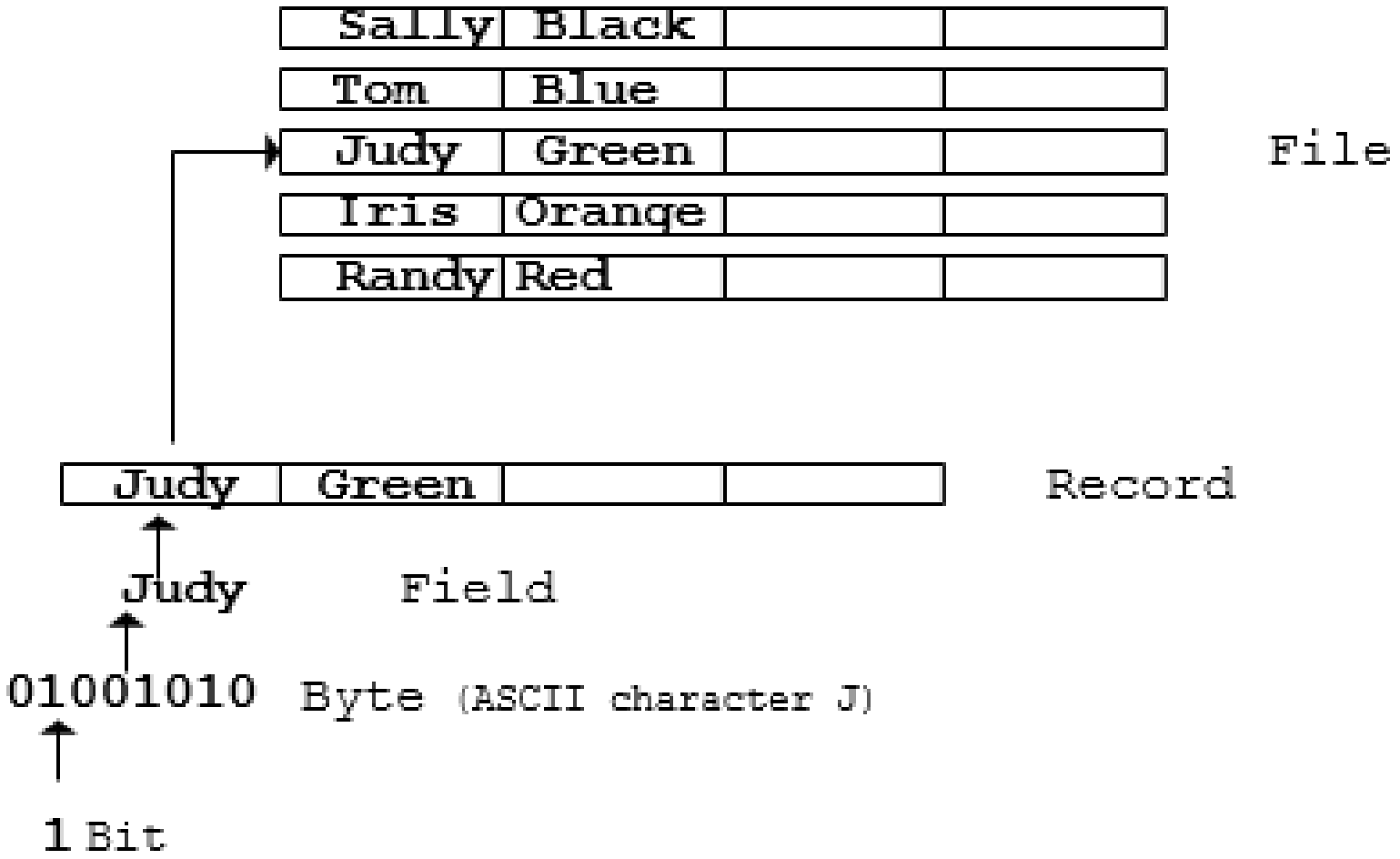
# Verileri Sıralı ve Random Erişimli Dosyalamak

# İçerik

Seri ve Rastgele erişimli dosyalar

Dosya sistemleri üzerinde çalışmak

C ve C# dilleri ile dosyalar ve akımlar üzerinde okuma/yazma işlemleri yapmak.


Sally Black
Tom Blue
Judy Green
Iris Orange
Randy Red
File
Judy Green
Record
Judy
Field
01001010
Byte (ASCII character J)
1 Bit

- Program sonlandırıldığında program içerisnde üzerinde işlem yapılan tüm veriler kaybolmaktadır.

- Eğer bu verilerin kaybolmadan saklanması isteniyorsa dosyalama işlemine gereksinim duyulur.

- C programlama dilinde I/O işlemleri için bir çok fonksiyon bulunmakdır. Bu fonksiyonlar 2 şekilde kategorize edilebilir.
  - Text Dosyalar
  - Binary Dosyalar

- Dosyalama İşlemleri
  - Yeni dosya oluşturma (Creating)
  - Varolan dosyanın açılması (Opening)
  - Dosyadan very okunması ve yazılması (Read and Write )
  - Dosyanın sonlandırılması (Closing file)

- Dosyalarla çalışırken, file türünde pointer tanımlanması gerekir. Bu tanımlama dosya ve program arasındaki iletişimin sağlanması açısından gereklidir.

```
FILE *ptr;
```

C'de, standart dosya yapısı tanımlanmıştır. Stdio.h başlık dosyasında FILE yapısı 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-çıkışiiçin kullanılan klavye ekran, printerve portlarıda birer dosya olarak tanımlamak mümkündür. Bu cihazlar sıralıerişimli dosya gibi davranırlar.

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

Ayrıca hata mesajlarının yazıldığıbir çıkışdosyasıvardır. Bu da stderr'dir.Bu dosyalar programın başında otomatik olarak tanımlanıp kullanım için açılırlar.

# Dosya açma/kapama (fopen, fclose)

- Dosya açmak için kullanılan yazım şekli

```
ptr= fopen("fileopen", "mode")

fopen("C:\\cfileoperations\program.txt", "w");
```

| File Mode | Anlamı | Dosyanın olmaması durumunda gerçekleşecekler |
|---|---|---|
| r | Okuma için açma | Eğer dosya mevcut değilse, fopen() NULL değeri döndürür. |
| w | Yazma için açma | Eğer dosya mevcutsa içeriğin üzerine yazılır. Tersi durumda (mevcut değilse) dosya oluşturulur. |
| a | Ekleme için açma. Veri dosyanın sonuna eklenir. | Eğer dosya mevcut değilse oluşturulur. |
| r+ | Yazma ve okuma için açılır. | Eğer dosya mevcut değilse, fopen() NULL değeri döndürür. |
| w+ | Yazma ve okuma için açılır. | Eğer dosya mevcutsa içeriğin üzerine yazılır. Tersi durumda (mevcut değilse) dosya oluşturulur. |
| a+ | Yazma ve okuma için açılır. | Eğer dosya mevcut değilse oluşturulur. |

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

- flush()
- fclose()

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. Tamamen kapatmak için fclose() veya fcloseall() kullanılmalıdır.

- flush (ptr) //Dönen değer 0 ise işlem başarılıdır.
- flush (void) // Açılmış olan bütün dosyaların tampon bellelerini diske aktarır.
- fclose(ptr) //Dosyayı kapatır.
- fcloseall(void) //Açık olan tüm dosyaları kapatır.

```c
#include <stdio.h>
int main()
{
    int n;
    FILE *fptr;
    fptr=fopen("C:\\program.txt","w");
    if(fptr==NULL){
        printf("Error!");
        exit(1);
    }
    printf("Enter n: ");
    scanf("%d",&n);
    fprintf(fptr,"%d",n);
    fclose(fptr);
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
   int n;
   FILE *fptr;
   if ((fptr=fopen("C:\\program.txt","r"))==NULL){
       printf("Error! opening file");
       exit(1);         /* Program exits if file pointer returns NULL. */
   }
   fscanf(fptr,"%d",&n);
   printf("Value of n=%d",n);
   fclose(fptr);
   return 0;
}
```

# fputc & fgetchar

- Heriki fonksiyonun yazım şekilleri

```
int fputc(int char, FILE *stream)

int fgetchar(FILE *stream)
```

```c
#include <stdio.h>
int main ()
{
   FILE *fp;
   int ch;

   fp = fopen("file.txt", "w+");
   for( ch = 33 ; ch <= 100; ch++ )
   {
      fputc(ch, fp);
   }
   fclose(fp);

   return(0);
}
```

```c
#include <stdio.h>
int main ()
{
   FILE *fp;
   int c;

   fp = fopen("file.txt","r");
   while(1)
   {
      c = fgetc(fp);
      if( feof(fp) )
      {
         break ;
      }
      printf("%c", c);
   }
   fclose(fp);
   return(0);
}
```

# Binary Dosyalar

- Eğer büyük miktarlarda veri depolama durumu sözkonusu ise text şeklinde depolama yeterli olmayacaktır. Böyle bir durumda binary olarak depolama tercih edilir.

- Binary şeklinde okuma/yazma text modunda okuma ve yazmaya oldukça benzer. Farklılık dosyayı açma modunda kendini gösterir.

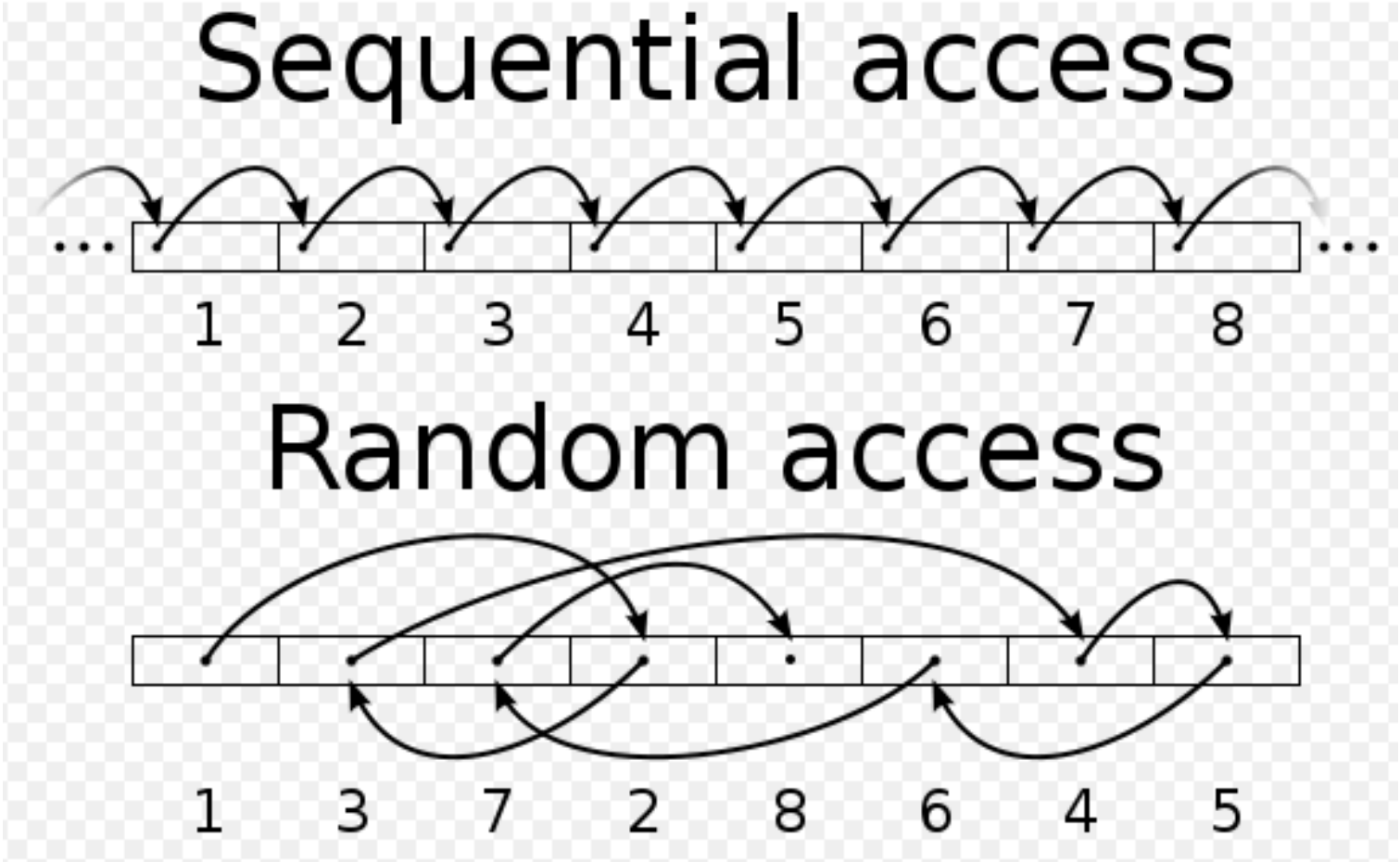
- Bu modlar; rb, rb+, wb, wb+, ab ve ab+ 'dir.

fread ve fwrite fonksiyonları binary dosyaları okuma ve yazma için sırasıyla kullanılır.

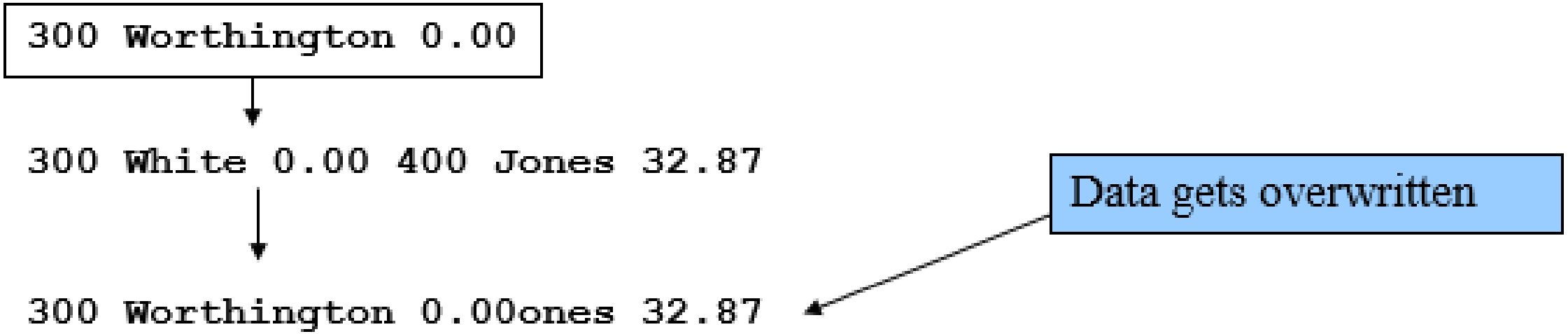
- fwrite'ın yazım şekli;

fwrite (address_data, size_data, numbers_data, pointer to file)

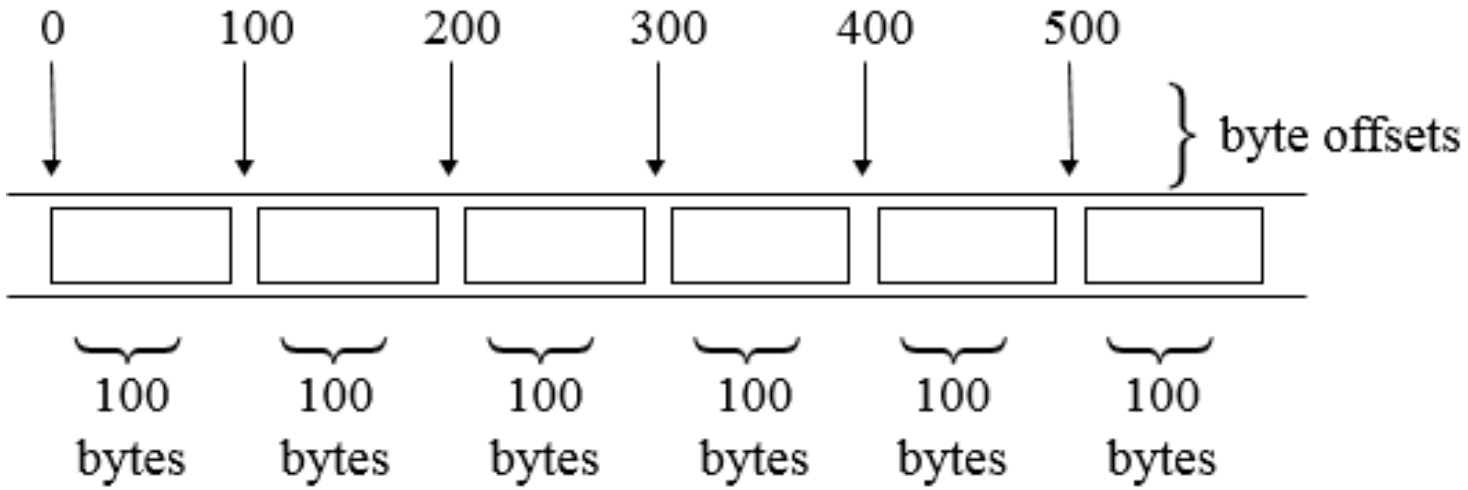
# Sequential & Random Access



- Sequential Files: Sıradüzensel olarak verilerin işlendiği durumlarda tercih edilir. Örn: text dosyadaki kelimelerin sayılması.
  - Kayıtlı verileri silerken önceki verilerde kayıplar meydana gelir.
  - Kayıtların boyutları çoğu durumda sabit değildir.



- Random Access Files: Verileri sıradüzensel olarak işlemek yerine dosyanın okunacağı ve/veya yazılacağı lokasyona erişim için kullanılır. Kayıtların boyutları sabittir.




- Çoğu durumda hybrid yaklaşım tercih edilir.

# 3 Temel Fonksiyon

- rewind()  //dosya pointerını dosyanın başlangıcına konumlandırır.
- fseek() //dosya pointerının pozisyonu
- ftell()  // dosya pointerının o anki konumunu geriye döndürür.

- Bu fonksiyonların herbiri C dosya işaretçisi üzerinde çalışırlar.
- Dosya içindeki herhangi bir kaydın konumu =Dosyanın başlangıcı + offset

# rewind()

- Hem sıralı hemde rastgele erişimli dosyalarda kullanılır.
- Dosya sistemine, dosya pointerının dosyanın başlangıcına konumlanmasını bildirir.
- Herhangi bir hata durumu sözkonusu ise bunu temizler.
- Herhangi bir değer geriye dönmez.
- fseek() fonksiyonu da benzer özellik için kullanılabilir.

# fseek & ftell

- Bu fonksiyon eğer kayıt boyutu (record size) bilindiği veya kaydın başlangıç ve bitiş adresleri bilindiği durumlarda çok fazla tercih edilir.

- 3 parametre almaktadır. Bunlar:

```
FILE *f     // dosya pointerı
Long offset       // offsetin pozisyonu
Int origin  // offsetin uygulanacağı nokta
```

FILE *f // dosya pointerı
Long offset // offsetin pozisyonu
Int origin // offsetin uygulanacağı nokta

Origin parametresi aşağıdaki 3 farklı değerden birini alabilir.

SEEK_SET // Başlangıçtan itibaren
SEEK_CUR // Şu anki bulunulan konumdan itibaren
SEEK_END // Sondan itibaren

- Dosyanın başlangıcına konumlanmak istenirse;

```
fseek(f, 0, SEEK_SET)  =~ rewind()
```

- Dosyanın sonuna kayıt eklenmek istenirse, dosya pointerı

```
fseek(f, 0, SEEK_END)
```

- Dosya pointerının bulunduğu konum bilgisi;

```
long file_marker = ftell(f)  //sonrasında fseek
fseek(f, file_marker, SEEK_SET)
```

# Error Handling

- Birçok C fonksiyonu hata durumunda -1 veya NULL geriye döndürür.

- Program çalışmasını başarılı bir şekilde sona erdirdiği durumda geriye 0(sıfır) döndürür.

# Global Variable errno

- Eğer fonksiyonun çalışması sırasında herhangi bir hata meydana gelirse herhangi bir int değer set edilir.

- errno' yu kullanabilmek için errno.h'ın programa dahil edilmesi ve extern olan extern int errno'nun çağrılması gerekir.

```c
#include <stdio.h>
#include <errno.h>
extern int errno;
int main () {
        FILE * fp;
        fp = fopen ("filedoesnotexist.txt", "rb");
        if (fp == NULL) {
                fprintf(stderr, "Value of errno: %d\n", errno);
        } else {
                fclose (fp);
        }
        return 0;
}
```

Value of errno is : 2

# strerror & perror

- errno ile ilişkili text içeriğin gösterilmesi için kullanılan iki fonksiyondur.
- strerror, errno ile ilişkili olarak görüntülenecek text içeriğin pointer bilgisini geri döndürür.
- perror ise programcının belirleyeceği text mesajı görüntüler.

# C#'da Dosyalama İşlemleri
# (File Operations in C#)

.NET, sürücüler, dosyalar ve klasörler üzerinde temel işlemleri gerçekleştirmek için sınıfları bünyesinde barındırır.

# Sürücüler

DriveInfo.GetDrives() (static) (System.IO NS)

```
class Program
  {
     static void Main(string[] args)
     {

        foreach (DriveInfo di in DriveInfo.GetDrives())
        {
           Console.WriteLine("{0} ({1})",di.Name,di.DriveType);
        }
     }
  }
```

- AvailableFreeSpace
- DriveFormat
- DriveType
- IsReady
- Name
- RootDirectory
- TotalFreeSpace
- VolumeLabel

# Dosya ve Klasör Yönetimi

```
class Program
  {
        static void Main(string[] args)
     {  DirectoryInfo dir = new DirectoryInfo(@"C:\Windows");
        Console.WriteLine("Klasörler");
        Console.WriteLine("**********");
        foreach (DirectoryInfo dirInfo in dir.GetDirectories())
        {  Console.WriteLine(dirInfo.Name);       }
        Console.WriteLine("\nDosyalar");
        Console.WriteLine("**********");
        foreach (FileInfo fi in dir.GetFiles())
        { Console.WriteLine(fi.Name); }
     }
  }
```

# Klasör Oluşturmak

```
class Program
  {
     static void Main(string[] args)
     {

        DirectoryInfo newDir = new DirectoryInfo(@"C:\bilgisayar");
        if (newDir.Exists)
           Console.WriteLine("Bu klasör zaten var");
        else
           newDir.Create();
     }
  }
```

# Dosya İşlemleri
## (Oluşturma, Kopyalama, Taşımak ve Silme)

```
class Program
    {
        static void Main(string[] args)
        {
        File.CreateText("yenidosya.txt");
        File.Copy("yenidosya.txt", "kopyadosya.txt");
        File.Move("kopyadosya.txt", "gecici.txt");
        }
    }
```

```
class Program
  {
     static void Main(string[] args)
     {
        FileInfo fi = new FileInfo("yenidosya.txt");
        fi.CreateText();
        fi.CopyTo("yenidosya2.txt");

        FileInfo fi2 = new FileInfo("yenidosya2.txt");
        fi2.MoveTo("yenidosya3.txt");
     }
  }
```

# Metin Dosyaları (Okuma/Yazma)

- Metin dosyalarını okuyabilmek için TextReader veya StreamReader sınıfları kullanılır. (StreamReader, TextReader'dan türemiştir.).

- Yazma işlemi için ise TextWriter ve StreamWriter kullanılır.

```
class Program
  {
     static void Main(string[] args)
     {
       TextReader tr = File.OpenText(@"C:\windows\win.ini");
       Console.WriteLine(tr.ReadToEnd());
                  tr.Close();
     }
  }
```

```
class Program
  {
     static void Main(string[] args)
     {
   StreamReader sr = new StreamReader(@"C:\windows\win.ini");
   string input;
   while((input = sr.ReadLine())!=null)
   Console.WriteLine(input);
   sr.Close();
     }
  }
```

# İkili Dosyalar (Binary Files)

Text olmayan değerleri okuyup yazabilmek için BinaryWriter ve BinaryReader sınıfları kullanılır.

(Her ikisi de bir dosya streamine ihtiyaç duyar)

```
class Program
  {
     static void Main(string[] args)
     {
        FileStream fs = new FileStream("data.bin", FileMode.Create);
        BinaryWriter w = new BinaryWriter(fs);
        for (int i = 0; i < 11; i++)
        {
           w.Write((int)i);
        }
        w.Close();
        fs.Close();
     }
  }
```

```
class Program
  {
     static void Main(string[] args)
     {
       FileStream fs = new FileStream("data.bin",
FileMode.Open,FileAccess.Read);
        BinaryReader r = new BinaryReader(fs);
        for (int i = 0; i < 11; i++)
        {
           Console.WriteLine(r.ReadInt32());
        }
        r.Close();
        fs.Close();
     }
  }
```

# String Okuma/Yazma

StringWriter ve StringReader sınıfları kullanılır.

```
class Program
  {
     static void Main(string[] args)
     {
        StringBuilder sb = new StringBuilder();
        StringWriter sw = new StringWriter(sb);
        sw.Write("Merhaba");
        sw.Write("Dünya");
        sw.Close();
     }
  }
```

```
class Program
  {
     static void Main(string[] args)
     {
        StringBuilder sb = new StringBuilder();
        StringReader sr = new StringReader(sb.ToString());
        sr.Close();
     }
  }
```

# Memory Stream

```
class Program
  {
     static void Main(string[] args)
     {
    MemoryStream ms = new MemoryStream();
    StreamWriter sw = new StreamWriter(ms);
    sw.WriteLine("Merhaba");
    sw.Flush();
    ms.WriteTo(File.Create("memory.txt"));
     sw.Close();
    ms.Close();
          }
  }
```

# Compressed Streams

GzipStream sınıfı kullanılır.

```
class Program
  {
     static void Main(string[] args)
     {
        GZipStream gzout = new GZipStream(File.Create("data.zip"),
CompressionMode.Compress);
        StreamWriter sw = new StreamWriter(gzout);
        for (int i = 0; i < 1000; i++)
        { sw.WriteLine("Merhaba");          }
        sw.Close();    gzout.Close();


        GZipStream gzin = new GZipStream(File.OpenRead("data.zip"),
CompressionMode.Decompress);
        StreamReader sr = new StreamReader(gzin);
        Console.WriteLine(sr.ReadToEnd());
        sr.Close();
        gzin.Close();
     }
  }
```

# Kaynak


Chapter 2