# BAB XI
# OPERASI FILE

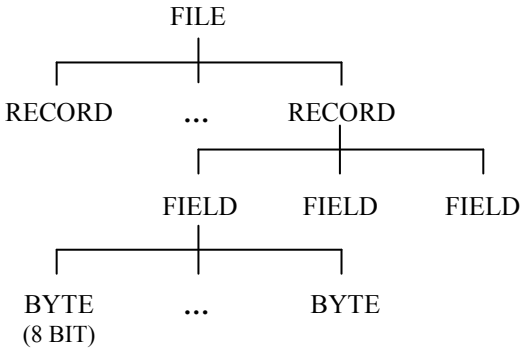
**Tujuan :**

1. Menjelaskan tentang struktur file
2. Menjelaskan tentang tahap-tahap operasi pada file
3. Menjelaskan tentang fungsi untuk penyimpanan dan pembacaan file per-karakter
4. Menjelaskan tentang file biner dan file teks
5. Menjelaskan tentang operasi penyimpanan dan pembacaan file per-int
6. Menjelaskan tentang operasi penyimpanan dan pembacaan file per-blok
7. Menjelaskan cara membaca dan menyimpan data string pada file
8. Menjelaskan cara mengakses file biner secara acak
9. Menjelaskan cara menghapus file
10. Menjelaskan cara mengganti nama file

## 11.1 Struktur File

Kebanyakan program melibatkan media disk sebagai tempat untuk membaca atau merekam data. Data sendiri disimpan dalam disk dalam bentuk suatu kesatuan yang disebut file. Suatu file merupakan organisasi dari sejumlah record. Masing-masing record dapat terdiri atas satu atau beberapa field dan setiap field terdiri atas satu atau beberapa byte. Adapun byte merupakan susunan dari 8 bit. Untuk lebih jelasnya, perhatikan gambar 11.1 di bawah ini.

Catatan : *record* adalah nama lain dari struktur *(struct)*.




Gambar 11.1 Struktur-data dari file

## 11.2 Tahapan Operasi File

Operasi pada file pada dasarnya meliputi tiga tahapan, yaitu :

1. Membuka/mengaktifkan file
2. Melaksanakan proses file
3. Menutup file

### 11.2.1 Membuka / Mengaktifkan File

Sebelum file dapat diakses (dibaca atau ditulisi), mula-mula file haruslah diaktifkan terlebih dahulu. Untuk keperluan ini fungsi yang digunakan yaitu *fopen()*. Bentuk deklarasinya adalah sebagai berikut :

```
FILE *fopen(char *namafile, char *mode);
```

dengan :

- **namafile** berupa nama dari file yang akan diaktifkan
- **mode** berupa jenis operasi yang akan dilakukan terhadap file
- prototipe ada pada file **stdio.h**

Jenis operasi file dapat berupa salah satu di antara mode berikut :

- **`r`** menyatakan file hanya akan dibaca.
  Dalam hal ini, file yang akan diproses haruslah sudah ada dalam disk pada current directory.
- **`w`** menyatakan bahwa file baru diciptakan. Selanjutnya operasi yang akan dilakukan terhadap file adalah operasi perekaman data. Seandainya file tersebut sudah ada dalam disk, isinya yang lama akan terhapus.
- **`a`** untuk membuka file yang sudah ada dalam disk, dan operasi yang akan dilakukan adalah penambahan data pada file. Data baru akan ditempatkan di bagian belakang dari file. Seandainya file belum ada, secara otomatis file akan diciptakan terlebih dahulu.
- **`r+`** untuk membuka file yang sudah ada, dan operasi yang akan dilakukan berupa pembacaan serta penulisan.
- **`w+`** untuk membuka file dengan tujuan untuk pembacaan atau penulisan. Jika file sudah ada, isinya akan dihapus.

- **a+** untuk membuka file, dengan operasi yang dapat dilakukan berupa perekaman maupun pembacaan. Jika file sudah ada, isinya tak akan dihapus.

Keluaran fungsi *fopen()* berupa pointer yang menunjuk ke tipe FILE (pointer to FILE), yaitu tipe struktur yang definisinya ada pada **stdio.h** (oleh karena itu program yang menggunakan *fopen()* harus melibatkan file **stdio.h**).

Berhasil tidaknya operasi pengaktifan file dapat dilihat pada keluaran fungsi *fopen()*. Jika keluaran fungsi berupa NULL (suatu makro yang didefinisikan pada file **stdio.h**), berarti operasi pengaktifan file gagal. Kejadian seperti ini bisa terjadi misalnya saat membuka file dengan mode "**r**" ternyata file yang dibuka tidak ada dalam disk.
Contoh pemakaian fungsi *fopen()* :

```
pf = fopen("COBA.TXT", "w");
```

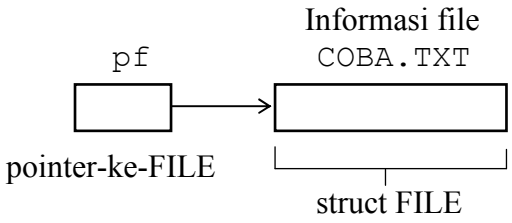
dengan variabel pf dideklarasikan sebagai berikut :

```
FILE *pf;
```

Maksud dari pernyataan `pf = fopen("COBA.TXT", "w");` adalah

- menciptakan dan mengaktifkan file bernama **"COBA.TXT"**
- dengan mode yaitu "**w**" (mode penulisan ke file)
- dan menempatkan pointer-ke-FILE ke variabel pointer **pf**

Dengan instruksi di atas, seandainya file **"COBA.TXT"** sudah ada dalam disk, maka isi file tersebut akan menjadi hilang (data lama akan terhapus).




Gambar 11.2 Pointer-ke-FILE **pf** menunjuk ke file **COBA.TXT**

Bentuk yang biasa dipakai untuk mengaktifkan file beserta pemeriksaan keberhasilannya adalah semacam berikut :

```
if (pf = fopen("COBA.TXT", "w") == NULL)
{
    printf("File tidak dapat diciptakan !\n");
    exit(1);                    /* keluar dari program */
}
```

Mula-mula **pf** diisi dengan keluaran dari fungsi *fopen()*. Seandainya nilainya adalah NULL (berarti operasi pengaktifan gagal), maka

- pada layar ditampilkan tulisan : `File tidak dapat diciptakan !`
- program dihentikan (selesai).

### 11.2.2 Menutup File

Apabila suatu file sudah tidak diproses lagi, maka file tersebut perlu ditutup. Hal seperti ini sangat penting terutama jika melakukan pemrosesan file yang jumlahnya lebih dari satu. Alasannya di antaranya adalah karena adanya keterbatasan jumlah file yang dapat dibuka secara serentak. Untuk menutup file, fungsi yang digunakan adalah *fclose()*, dengan bentuk deklarasi sebagai berikut :

```
int fclose(FILE *pf);
```

dengan prototipe ada pada **stdio.h.**

Pada waktu pemanggilan fungsi ini, **pf** haruslah berupa variabel pointer bertipe *FILE* yang digunakan dalam pengaktifan file. Fungsi *fclose()* menghasilkan keluaran berupa nol jika operasi penutupan berhasil dilakukan.

Di samping *fclose()*, terdapat pula fungsi bernama *fcloseall()* yang digunakan untuk menutup semua file yang sedang terbuka. Bentuk deklarasinya :

```
int fcloseall(void);
```

Fungsi ini menghasilkan nilai EOF (EOF didefinisikan pada **stdio.h**, yaitu bernilai –1) jika terjadi kegagalan. Sedangkan bila berhasil, keluaran fungsi berupa jumlah file yang ditutup.

## 11.3 Operasi Penyimpanan dan Pembacaan File Per Karakter

### 11.3.1 Fungsi*fputc()*

Sebuah karakter dapat disimpan/dituliskan ke dalam file dengan menggunakan fungsi *fputc()*. Bentuk deklarasi dari fungsi ini :

```
int fputc(char kar, FILE *ptr_file);
```

dengan **ptr_file** adalah pointer-ke-FILE yang berisi keluaran dari *fopen()*, dan kar berupa karakter yang akan disimpan dalam file. Sekalipun kar bertipe int (2 byte), sebenarnya hanya byte terendah dari kar yang akan disimpan ke dalam file. Byte tertinggi tak ikut disimpan.

Seandainya operasi *fputc()* berjalan dengan sempurna, keluaran fungsi sama dengan nilai kar. Bila tak berhasil melaksanakan penyimpanan, keluaran fungsi berupa EOF (-1). Contoh program untuk menciptakan file dan digunakan untuk menyimpan sejumlah karakter :

```
/* File program: fputc.c
Menciptakan & mengisi file dgn data karakter dr keyboard */

#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *pf;                       /* Pointer-ke-FILE */
    char kar;

    /* Ciptakan file */
    if ((pf = fopen("COBA.TXT","w")) == NULL)
    {
        printf("file tak dapat diciptakan!\r\n");
        exit(1);                    /* selesai */
    }

    printf("Ketikkan apa saja, akhiri dengan ENTER.\n");
    printf("Program akan membaca perkarakter\n");
    printf("dan menyimpannya dalam file COBA.TXT\n\n");

    while((kar=getchar()) != '\n')  /*baca kar dr keyboard*/
        fputc(kar, pf);     /*tulis ke file per karakter*/

    fclose(pf);                     /* tutup file */
}
```

**Contoh eksekusi :**

```
Ketikkan apa saja, akhiri dengan ENTER.
Program akan membaca perkarakter
dan menyimpannya dalam file COBA.TXT

Mencoba menulis ke file COBA.TXT
```

---

Program mula-mula menciptakan dan membuka file melalui pemanggilan fungsi *fopen()*, dengan mode file "w". Kalau keluaran fungsi bernilai NULL, program dihentikan melalui *exit( ).* Kalau file **COBA.TXT** berhasil dibuka, maka pernyataan

```
while((kar=getchar()) != '\n')
  fputc(kar, pf);
```

akan dijalankan, yang memungkinkan untuk memasukkan sejumlah karakter, sampai tombol ENTER ditekan (ENTER tidak ikut disimpan dalam file). Jika tombol ENTER ditekan, file akan ditutup dan eksekusi program selesai. Sedangkan file **COBA.TXT** yang dihasilkan oleh program di atas merupakan file teks, sehingga isinya bisa dilihat dengan menggunakan bantuan sebuah teks editor misalnya **Notepad**.

### 11.3.2 Fungsi*fgetc()*

Untuk melihat isi file hasil program di atas, bisa juga melalui program dengan memakai fungsi *fgetc()*, yang digunakan untuk pembacaan per karakter dari isi file. Prototipe dari fungsi ini ada di **stdio.h**. Bentuk deklarasi *fgetc()*:

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

Keluaran fungsi berupa nilai bertipe int dari sebuah karakter yang dibaca dari file. Jika akhir file ditemukan atau terjadi kegagalan membaca, keluaran fungsi berupa EOF.

Program berikut digunakan untuk membaca isi file **COBA.TXT** dengan langkah-langkah sebagai berikut :

1. Buka file COBA.TXT dengan mode “r”

   Jika tidak berhasil dibuka maka

   - beri keterangan pada layar bahwa file tak ada
   - selesai

2. Baca sebuah karakter dari file

   Jika karakter sama dengan EOF (tanda akhir file) maka ke langkah 4

3. Tampilkan karakter ke layar dan kembali ke langkah 2
4. Tutup file
5. Selesai

---

```
/* File program: fgetc.c
contoh membaca isi file per karakter */

#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *pf;
    char kar;

    if((pf=fopen("COBA.TXT","r")) == NULL )   /* buka file */
    {
        printf("file tak dapat dibuka !\r\n");
        exit(1);                              /* selesai */
    }

    while((kar=fgetc(pf)) != EOF)   /* baca kar dari file */
        putchar(kar);               /* tampilkan ke layar*/

    printf("\n");
    fclose(pf);                         /* tutup file */
}
```

**Contoh eksekusi :**

```
Mencoba menulis ke file COBA.TXT
```

---

## 11.4 File Biner dan File Teks

Pada saat file dibuka, file bisa diperlakukan sebagai file biner atau file teks. File biner adalah file yang pola penyimpanan di dalam disk berbentuk biner, yaitu seperti bentuk pada memori RAM (komputer). Misalnya data bertipe *int* selalu akan menempati ruang 4 byte (pada mesin 32 bit), berapapun harganya.

Sedangkan file teks merupakan file yang pola penyimpanan datanya dalam bentuk karakter. Bilangan bertipe *int* misalnya, bisa saja menempati ruang 1 byte, 2 byte atau lebih bergantung kepada nilai dari bilangannya. Sebagai contoh, bilangan 54 akan disimpan dalam 2 byte (berupa karakter 5 dan 4), tetapi bilangan 123 akan disimpan dalam 3 byte. File seperti ini bisa dilihat langsung dengan perintah **TYPE** melalui prompt DOS atau memakai editor teks (seperti **Notepad**).

File teks biasanya dipakai untuk menyimpan data bertipe karakter atau string. Sedangkan file biner dipakai untuk menyimpan data bilangan atau data kompleks, seperti struktur (*struct*).

Penambahan yang perlu dilakukan untuk menentukan mode teks atau mode biner berupa :

- **`t`** untuk mode teks
- **`b`** untuk mode biner

Contoh :

- `"rt"` Berarti mode file adalah teks dan file hendak dibaca
- `"rt+"` Berarti mode file adalah teks dan file bisa dibaca dan ditulisi.
  Bentuk penulisan yang lain (tetapi maknanya sama) : `"r+t"`
- `"rb"` Berarti mode file adalah biner dan file hendak dibaca.

Catatan :

- Jika pada mode file tidak terdapat karakter **`t`** atau **`b`**, mode file akan ditentukan oleh variabel global bernama *_fmode* (deklarasinya ada pada file **fcntl.h**). Jika *_fmode* tidak dilibatkan dalam program, maka mode file yang tak mengandung **`t`** atau **`b`** akan diperlakukan sebagai file teks (secara *default*).
- Variabel *_fmode* bisa diisi dengan **O_BINARY** untuk menyatakan file biner, atau **O_TEXT** untuk menyatakan file teks. Contoh :

```
_fmode = O_BINARY;
pf = fopen("TEST1", "r");
```

Berarti bahwa "TEST1" adalah file biner dan hendak dibaca.

- **O_TEXT** ataupun **O_BINARY** didefinisikan pada file **fcntl.h**

### 11.5 Operasi Penyimpanan dan Pembacaan File Per Int

Untuk keperluan menyimpan atau membaca membaca file bertipe int, C menyediakan fungsi *_putw()* dan *_getw()*. Betuk deklarasinya :

```
int _putw(int nilai, FILE *ptr_file);
int _getw(FILE *ptr_file);
```

Dengan prototipe ada pada **stdio.h**. Kegunaan masing-masing adalah :

- *_getw()* untuk membaca sebuah data bertipe int dari file
- *_putw()* untuk menyimpan sebuah data (yang disimpan dalam variabel **nilai**) yang bertipe *int* ke file.

Contoh berikut merupakan program untuk menyimpan sejumlah data bertipe *int* ke dalam file bertipe biner bernama **BILANGAN.DAT**. Dalam hal ini, file **BILANGAN.DAT** akan diperlakukan sebagai file biner.

---

```
/* File program: _putw.c
contoh menyimpan data bertipe int menggunakan putw() */

#include <stdio.h>
#include <stdlib.h>

main( )
{
    FILE *pf;                          /* ptr-ke-FILE */
    int nilai, sudah_benar;
    char jawab;

    if((pf=fopen("BILANGAN.DAT", "wb")) == NULL )
                                       /* ciptakan file*/
    {
        printf("file gagal diciptakan!\n");
        exit(1);
    }

    printf("MENYIMPAN DATA INTEGER KE FILE\n");
```

```
    do {
        printf("\nBilangan yang akan disimpan: ");
        scanf("%d", &nilai);    /* baca nilai dr keyboard */
        _putw(nilai, pf);       /* baca bilangan ke file */
        printf("memasukkan data lagi (Y/T)?   ");

        do
        {
            jawab = getchar(); /* baca jawaban dr keyboard */
            sudah_benar = ((jawab == 'Y') || (jawab == 'y')
                || (jawab == 'T') || (jawab == 't'));
        } while(! sudah_benar);
    }
    while  (jawab == 'y'|| jawab == 'Y');

    printf("\nOke. Data sudah disimpan dalam file.\n");
    fclose(pf);                          /* menutup file */
}
```

**Contoh eksekusi :**

```
Program untuk menyimpan data integer ke file

Bilangan yang akan disimpan: 60
Memasukkan data lagi (Y/T)? y

Bilangan yang akan disimpan: 998
Memasukkan data lagi (Y/T)? y

Bilangan yang akan disimpan: -75
Memasukkan data lagi (Y/T)? t
Oke. Data sudah disimpan dalam file.
```

Program yang digunakan untuk menampilkan isi file **BILANGAN.DAT** pada dasarnya sama dengan program **fgetc.c** yang menampilkan isi file teks **COBA.TXT** di atas. Hanya saja, untuk mendeteksi akhir dari file, diperlukan makro bernama *feof()***,** yang memiliki bentuk deklarasi

```
int feof(FILE *ptr_file);
```

dengan prototipe dan definisinya ada pada file**stdio.h**

Keluaran *feof()* berupa nilai nol (NULL) jika operasi pembacaan yang terakhir membaca tanda akhir file. Sebagai contohnya, perhatikan implementasi pada program di bawah ini.

```
/* File program : _getw.c
```

```
Contoh membaca isi file biner menggunakan getw() */

#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *pf;                   /* ptr ke file */
    int nilai, nomor = 0;
    /* Buka file biner untuk dibaca */
    if((pf=fopen("BILANGAN.DAT","rb")) == NULL)
    {
        printf("File gagal dibuka.\n");
        exit(1);
    }

    printf("Isi file BILANGAN.DAT : \n");

    while(1)                    /* file berhasil dibuka */
    {
        nilai = _getw(pf);      /* Baca sebuah int dr file */
        if (feof(pf) != 0)  /*Jika akhir file, keluar loop*/
            break;
                                /* Tampilkan ke layar */
        printf("%2d.  %d \n", ++nomor, nilai);
    }

    fclose(pf);                 /* Tutup file */
}
```

**Contoh eksekusi :**

```
Isi file BILANGAN.DAT :
1. 60
2. 998
3. -75
```

---

### 11.6 Operasi Penyimpanan dan Pembacaan File Per Blok

Ada dua fungsi yang memungkinkan untuk menyimpan atau membaca data file dalam bentuk kesatuan blok (sejumlah byte), misalnya untuk menyimpan data bertipe *float* atau data bertipe *struct*. Kedua fungsi tersebut adalah *fread()* dan *fwrite()* yang memiliki bentuk deklarasi sbb :

```
int fread(void *buffer, int n, FILE *ptr_file);
```

```
int fwrite(void *buffer, int jum_byte, int n,
  FILE *ptr_file);
```

dengan :

- **buffer** adalah
  - pointer yang menunjuk ke daerah memori yang akan ditempati data dari file disk (untuk *fread()*), atau
  - pointer yang menunjuk ke daerah memori yang akan berisi data yang akan disimpan ke file disk (untuk *fwrite()*.
- **jum_byte** menyatakan jumlah byte yang akan dibaca atau disimpan.
- **n** menentukan banyaknya blok data berukuran **jum_byte** yang akan ditulis atau dibaca.
- **ptr_file** berupa pointer-ke-FILE yang berisi nilai keluaran dari *fopen()*.

Program berikut ini memberikan contoh penyimpanan data bertipe struktur ke dalam file disk bernama **DAFBUKU.DAT**.

```
/* File program : fwrite.c
Menyimpan data bertipe struktur ke file memakai fwrite() */

#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *f_struktur;
    char jawaban;
    int sudah_benar;

    struct {
        char judul[26];
        char pengarang[20];
        int jumlah;
    } buku;             /* variabel buku bertipe struktur */

    /* Buka file */
    if((f_struktur  = fopen("DAFBUKU.DAT", "wb")) == NULL)
    {
        printf("File tidak dapat diciptakan !\n");
        exit(1);
    }

    do {
        fflush(stdin);  /* Hapus isi penampung keyboard */
        printf("Judul buku    : ");
        gets(buku.judul);
```

```
        printf("Nama pengarang     : ");
        gets(buku.pengarang);
        printf("Jumlah buku   : ");
        scanf("%d", &buku.jumlah);
        fflush(stdin);  /* Hapus isi penampung keyboard */

        /* Rekam sebuah data bertipe struktur */
        fwrite(&buku, sizeof(buku), 1, f_struktur);

        printf("\nMau merekam data lagi [Y/T] ?");

        do {
            jawaban = getchar();
            sudah_benar = ((jawaban == 'Y') || (jawaban ==
                'y')||(jawaban == 'T') || (jawaban == 't'));
        } while(!sudah_benar);

        printf("\n");

    } while(jawaban == 'Y' || jawaban == 'y');

    fclose(f_struktur);             /* Tutup file */
}
```

**Contoh eksekusi :**

```
Judul buku     : Relational Database Design
Nama pengarang : Igor T. Hawryszkiewycz
Jumlah         : 1
Mau merekam data lagi [Y/T] ? Y
Judul buku     : C Programming FAQs
Nama pengarang : Steve Summit
Jumlah         : 4
Mau merekam data lagi [Y/T] ? Y

Judul buku     : The C Programming Language
Nama pengarang : Brian WK & Dennis MR
Jumlah         : 2
Mau merekam data lagi [Y/T] ? T
```

---

Pada program di atas, instruksi untuk menyimpan sebuah data bertipe *struct* ke file adalah

```
fwrite(&buku, sizeof(buku), 1, f_struktur);
```

yang menyatakan data sebanyak 1 x ukuran variabel *struct* buku (dalam satuan *byte*) dari lokasi buku (dinyatakan dengan `&buku`) disimpan dalam file *f_struktur* (nama filenya

adalah **DAFBUKU.DAT**). Untuk membaca data yang ada pada file **DAFBUKU.DAT**, programnya adalah sbb :

```
/* File program : fread.c
Membaca data bertipe struktur ke file menggunakan fread() */

#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *f_struktur;
    int i=1;



    struct {
        char judul[30];
        char pengarang[30];
        int jumlah;
    } buku;             /* variabel buku bertipe struktur */

    /* Buka file */
    if((f_struktur  = fopen("DAFBUKU.DAT", "rb")) == NULL)
    {
        printf("File tidak dapat dibuka !\n");
        exit(1);
    }

    printf("%2s. %-30s %-30s %s\n\n", "No", "Judul Buku",
        "Nama Pengarang", "Jumlah");

    /* diulang selama masih ada record yg terbaca dlm file */
    while(fread(&buku, sizeof(buku), 1, f_struktur) == 1)
        printf("%2d. %-30s %-30s %4d\n", i++, buku.judul,
            buku.pengarang, buku.jumlah);

    printf("\n");
    fclose(f_struktur);     /* Tutup file */
}
```

**Contoh eksekusi :**

```
No Judul Buku                  Nama Pengarang          Jumlah
1. Relational Database Design  Igor T. Hawryszkiewycz     1
2. C Programming FAQs          Steve Summit               4
3. The C Programming Language  Brian WK & Dennis MR       2
```

## 11.7 Menyimpan dan Membaca Data String pada File

Dua fungsi yang dipakai untuk membaca data string pada file yaitu *fgets()* dan *fputs()*. Bentuk deklarasinya :

```
int fputs(char *str, FILE *ptr_file);
char fgets(char *str, int n, FILE *ptr_file);
```

dengan prototipe pada file **stdio.h**

Kegunaannya :

- *fputs()* untuk menyimpan string str ke dalam file.
- *fgets()* untuk membaca string dari file sampai ditemukannya karakter baris baru '\n' atau setelah (n-1) karakter, dengan n adalah panjang maksimal string yang dibaca per waktu-baca.

Keluaran fungsi :

- untuk *fputs()*: - Jika penyimpanan berhasil dilaksanakan, hasilnya berupa karakter yang terakhir ditulis ke file.
  - Jika gagal, hasilnya berupa EOF.
- untuk *fgets()* : - Jika pembacaan berhasil dilaksanakan, hasilnya berupa pointer yang menunjuk string yang ditunjuk oleh **str**.
  - Jika gagal, hasilnya berupa NULL.

Catatan :

- Pada saat menyimpan string ke file, *fputs()* tidak menambahkan karakter baris-baru ('\n') dengan sendirinya, dan karakter null tidak ikut disimpan.
- Pada saat pembacaan dengan *fgets()*, jika string yang dibaca mengandung karakter baris baru (CR/LF), hanya karakter LF yang akan disertakan pada string. Secara otomatis string akan diakhiri dengan karakter null
- Baik *fgets()* maupun *fputs()* digunakan untuk file teks.

Perhatikan program-program di bawah ini :

```
/* File program : fgets.c
Membaca isi file teks */
```

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

#define PANJANG 256


main()
{
    FILE *f_teks;
    char string[PANJANG];
    char namafile[65];

    printf("PROGRAM UNTUK MELIHAT ISI FILE TEKS\n\n");
    printf("Masukkan nama file : ");
    gets(namafile);

    printf("\nIsi file %s adalah sbb :\n", namafile);
    if((f_teks=fopen(namafile,"rt")) == NULL)
    {
        printf("File gagal dibuka\n");
        exit(1);
    }

    while(fgets(string, sizeof string, f_teks)!= NULL);
        printf("%s\n\n", string);

    fclose(f_teks);
}
```

**Contoh eksekusi :**

```
PROGRAM UNTUK MELIHAT ISI FILE TEKS

Masukkan nama file : coba.txt
Isi file %s adalah sbb :
Mencoba menulis ke file COBA.TXT
```

---

```
/* File program : fputs.c
Membaca kemudian menyalin isi file teks */

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

#define PANJANG 256
```

```
main()
{
    FILE *pf_input, *pf_output;
    char string[PANJANG];
    char namafile_inp[65], namafile_out[65];

    printf("PROGRAM UNTUK MENYALIN ISI FILE TEKS\n\n");
    printf("Masukkan nama file input : ");
    gets(namafile_inp);
    printf("Masukkan nama file output: ");
    gets(namafile_out);

    /* Buka file input */
    if((pf_input=fopen(namafile_inp,"r+")) == NULL)
    {
        printf("File input gagal dibuka\n");
        exit(1);
    }

    /* Buka file output */
    if((pf_output=fopen(namafile_out,"w+")) == NULL)
    {
        printf("File output gagal dibuka\n");
        exit(1);
    }

    /* menampilkan isi file input, merubahnya ke huruf besar
       & menyalinnya ke file output */
    while(fgets(string, sizeof string, pf_input) != NULL)
    {
        printf("\nIsi file %s adalah :\n",namafile_inp);
        printf("%s\n", string);
        strupr(string);        /* ubah menjadi huruf besar */
        fputs(string, pf_output); /*menyalin ke file output*/
    }

    fcloseall();

    /* Buka file output */
    if((pf_output=fopen(namafile_out,"r+")) == NULL)
    {
        printf("File output gagal dibuka\n");
        exit(1);
    }

    /* tampilkan isi file output */
    printf("\nIsi dari file %s adalah : \n",namafile_out);
    while(fgets(string, sizeof string, pf_output) != NULL)
        printf("%s\n\n",string);
    fclose(pf_output);
```

```
}
```

**Contoh eksekusi :**

```
PROGRAM UNTUK MENYALIN ISI FILE TEKS

Masukkan nama file input : coba.txt
Masukkan nama file output: out.txt

Isi file coba.txt adalah :
Mencoba menulis ke file COBA.TXT

Isi file out.txt adalah :
MENCOBA MENULIS KE FILE COBA.TXT
```

---

### 11.8 Mengakses File Biner secara Acak

C juga menyediakan fasilitas yang memungkinkan pembacaan file secara random (acak). Dengan adanya fasilitas ini, seandainya diinginkan untuk membaca data yang berada di tengah file, tidaklah perlu untuk membaca record demi record dimulai dari awal file. Oleh karenanya pengaksesan suatu data dapat dilaksanakan dengan cepat.

Untuk keperluan pengaksesan secara random, fungsi yang digunakan adalah *fseek()*. Bentuk deklarasinya :

```
int fseek(FILE *ptr_file, long int offset, int posisi);
```

dengan :

- **ptr_file** adalah pointer yang berasal dari keluaran *fopen()*
- **offset** menyatakan jumlah byte terhadap posisi
- **posisi** dapat diisi dengan salah satu nilai yang tertera pada tabel 11.1

Kegunaan fungsi *fseek()* yaitu untuk menempatkan penunjuk file ke suatu lokasi dalam file, berdasarkan offset dan posisi.

Tabel 11.1 Konstanta untuk menentukan posisi pada pengaksesan file secara acak

| Konstanta | Nilai | Lokasi file |
|---|---|---|
| SEEK_SET | 0 | Awal file |
| SEEK_CUR | 1 | Posisi penunjuk file saat ini |
| SEEK_END | 2 | Akhir file |

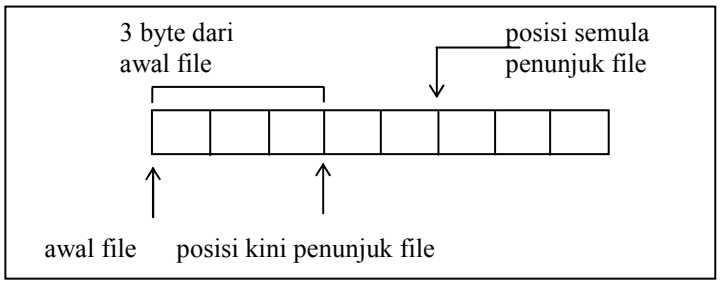
Catatan :

- Konstanta simbolis `SEEK_SET, SEEK_CUR` dan `SEEK_END` didefinisikan pada file **stdio.h**
- Prototipe *fseek()* ada pada **stdio.h**

Beberapa contoh :

```
(1) fseek(pf, 3, SEEK_SET);
```

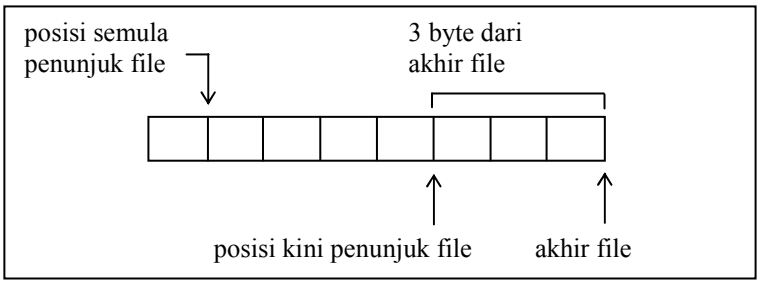
Pernyataan seperti ini akan menempatkan penunjuk file ke posisi 3 byte sesudah awal file (`SEEK_SET`).




Gambar 11.3 Ilustrasi penggunaan `SEEK_SET`

(2) `fseek(pf, 3, SEEK_END);`

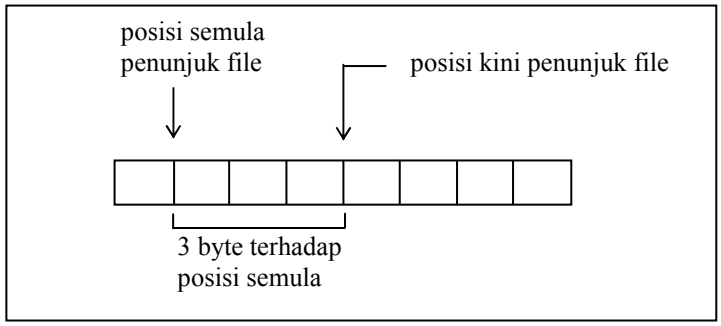
Pernyataan seperti ini akan menempatkan penunjuk file ke posisi 3 byte sebelum akhir file (`SEEK_END`).




Gambar 11.4 Ilustrasi penggunaan `SEEK_END`

(3) `fseek(pf, 3, SEEK_CUR);`

Pernyataan seperti ini akan menempatkan penunjuk file ke posisi 3 byte sesudah posisi penunjuk file sedang berada saat ini.




Gambar 11.5 Ilustrasi penggunaan SEEK_CUR

```
 /* File program : baca_acak.c
Membaca isi file secara random */

#include <stdio.h>
#include <stdlib.h>

main()
{
    struct {
        char judul[30];
        char pengarang[30];
        int jumlah;
    } buku;             /* variabel buku bertipe struktur */

    FILE *pf;
    char jawab;
    int i, no_record, sudah_benar;
    long int offset_byte;


    /* Buka file */
    if((pf  = fopen("DAFBUKU.DAT", "rb")) == NULL)
    {
        printf("File tidak dapat dibuka !\n");
        exit(1);
    }

    do
    {
        i = 1;
        printf("Nomor record dr data yg mau ditampilkan : ");
        scanf("%d", &no_record);

        offset_byte = (no_record-1) * sizeof(buku);
        fseek(pf, offset_byte, SEEK_SET);
```

```
        if(fread(&buku, sizeof(buku), 1, pf) == 0)
            printf("Nomor record tdk dikenali!\n");
        else
        {
            printf("\n%2s. %-30s %-30s %s\n\n", "No",
                "Judul Buku", "Nama Pengarang", "Jumlah");
            printf("%2d. %-30s %-30s %4d\n", i++, buku.judul,
                buku.pengarang, buku.jumlah);
        }
        printf("\nMau mencoba lagi (Y/T)? ");
        do
        {
            jawab = getchar();  /*baca jawaban dr keyboard */
            sudah_benar = ((jawab == 'Y') || (jawab == 'y')
                ||(jawab == 'T') || (jawab == 't'));
        } while(! sudah_benar);

    } while (jawab == 'y' || jawab == 'Y');

    printf("\n");
    fclose(pf);                 /* Tutup file */
}
```

**Contoh eksekusi :**

```
Nomor record dr data yg mau ditampilkan : 1

No. Judul Buku                  Nama Pengarang         Jumlah
1. Relational Database Design   Igor T. Hawryszkiewycz     1

Mau mencoba lagi (Y/T)? Y
Nomor record dari data yg mau ditampilkan : 3

No. Judul Buku                  Nama Pengarang         Jumlah
1. The C Programming Language   Brian WK & Dennis MR       2

Mau mencoba lagi (Y/T)? Y
Nomor record dari data yg mau ditampilkan : 7
Nomor record tidak dikenali!

Mau mencoba lagi (Y/T)? T
```

---

Mula-mula program menanyakan nomor record dari data yang ingin ditampilkan. Selanjutnya enunjuk file ditempatkan pada posisi data yang akan ditampilkan, melalui instruksi

```
    offset_byte = (no_record-1) * sizeof(buku);
    fseek(pf, offset_byte, SEEK_SET);
```

Langkah berikutnya, membaca data file dengan menggunakan *fread()*. Kalau keluaran *fread()* bernilai 0, maka di layar akan dimunculkan pesan : "Nomor record tidak dikenali!"

Kegunaan *fseek()* selain untuk membaca data secara random, juga memungkinkan untuk mengubah data secara acak, seperti pada program di bawah ini.

---

```
/* File program : gantirec.c
Mengganti isi suatu record secara random */

#include <stdio.h>
#include <stdlib.h>

#define SATU_RECORD 1

main()
{
    struct {
        char judul[30];
        char pengarang[30];
        int jumlah;
    } buku;             /* variabel buku bertipe struktur */

    FILE *pf;               /* pointer ke FILE */
    char jawab;
    int no_record, sudah_benar, hasil_baca;
    long int offset_byte;


    /* Buka file yg berisi data buku */
    if((pf  = fopen("DAFBUKU.DAT", "rb+")) == NULL)
    {
        printf("File tidak dapat dibuka !\n");
        exit(1);
    }

    /* Baca record secara random */
    do
    {
        printf("Nomor record dari data yg mau diubah : ");
        scanf("%d", &no_record);

        /* atur penunjuk posisi-file ke record tsb */
        offset_byte = (no_record-1) * sizeof(buku);
        fseek(pf, offset_byte, SEEK_SET);
```

```
        /*Baca record yg ditunjuk oleh penunjuk posisi_file*/
        hasil_baca = fread(&buku, sizeof(buku), SATU_RECORD,
            pf);

        if(hasil_baca == 0)
            printf("Nomor record tdk dikenali!\n");
        else
        {
            printf("\n%-30s %-30s %s\n\n","Judul Buku",
                "Nama Pengarang", "Jumlah");
            printf("%-30s %-30s %4d\n\n", buku.judul,
                buku.pengarang, buku.jumlah);

            printf("Jumlah buku tsb kini = ");
            scanf("%d", &buku.jumlah);

            /*Atur penunjuk posisi-file ke posisi seblmnya */
            fseek(pf, offset_byte, SEEK_SET);

            /* Rekam ulang */
            fwrite(&buku, sizeof(buku), SATU_RECORD, pf);
        }

        printf("\nMau mengubah lagi (Y/T)? ");
        do
        {
            jawab = getchar();  /*baca jawaban dr keyboard */
            sudah_benar = ((jawab == 'Y') || (jawab == 'y')
                ||(jawab == 'T') || (jawab == 't'));
        } while(! sudah_benar);

    } while (jawab == 'y' || jawab == 'Y');

    printf("\n");
    fclose(pf);                 /* Tutup file */
}
```

**Contoh eksekusi :**

```
Nomor record dari data yg mau diubah : 2

Judul Buku                    Nama Pengarang          Jumlah

C Programming FAQs            Steven Summit              4

Jumlah buku tsb kini = 5

Mau mengubah lagi (Y/T) ? T
```

Proses penggantian data record dilakukan dengan mula-mula menempatkan penunjuk file pada posisi dari data yang akan diganti. Selanjutnya data dibaca (dengan *fread()*), dan akan ditampilkan di layar. Setelah data jumlah buku yang baru dimasukkan dari keyboard, penunjuk file ditempatkan kembali ke posisi tempat data yang dibaca tadi. Kemudian data baru (satu record) direkam ulang dengan *fwrite()*.

### 11.9 Menghapus File

C menyediakan fungsi yang berguna untuk menghapus file yaitu *remove()*. Bentuk deklarasinya :

```
int remove (char *namafile);
```

dengan namafile adalah pointer yang menunjuk ke nama file yang akan dihapus.

Fungsi ini menghasilkan keluaran berupa nilai nol bila operasi penghapusan file berhasil dilaksanakan. Kalu terjadi kegagalan, keluaran fungsi berupa selain nol. Prototipe dari fungsi ini ada pada **stdio.h**

```
/* File program : hapusfile.c
Contoh program untuk menghapus file */

#include <stdio.h>
#include <stdlib.h>

#define PJG 65

main()
{
    int kode;
    char namafile[PJG];

    printf("Nama file yang akan dihapus : ");
    gets(namafile);

    kode = remove(namafile);
    if(kode == 0)
      printf("File sudah dihapus\n");

    else
      printf("Gagal dalam menghapus file\n");
}
```

**Contoh eksekusi :**

```
Nama file yang akan dihapus : bilangan.dat
File sudah dihapus
```

## 11.10 Mengganti Nama File

Untuk mengganti nama file, fungsi yang digunakan yaitu *rename()*. Bentuk deklarasinya :

```
int rename(char *namafilelama, char *namafilebaru);
```

Jika operasi penggantian nama file lama menjadi nama file baru ini berhasil, maka keluaran fungsi berupa nol. Jika terjadi kegagalan, keluaran fungsi berupa selain nol. Prototipe dari fungsi ini ada pada file **stdio.h**

```
/* File program : gantinama.c
Contoh program untuk mengganti nama file */

#include <stdio.h>
#include <stdlib.h>

#define PJG 65

main()
{
    int kode;
    char namafilelama[PJG], namafilebaru[PJG];

    printf("Nama file yang akan diganti : ");
    gets(namafilelama);
    printf("Nama file yang baru       : ");
    gets(namafilebaru);

    kode = rename(namafilelama, namafilebaru);
    if(kode == 0)
      printf("Nama file sudah diganti\n");
    else
      printf("Gagal dalam mengganti nama\n");
}
```

**Contoh eksekusi :**

```
Nama file yang akan diganti : bilangan.dat
Nama file yang baru         : bilangan1.dat
```

```
Nama file sudah diganti
```

**Kesimpulan :**

- File merupakan organisasi dari sejumlah record. Masing-masing record dapat terdiri atas satu atau beberapa field dan setiap field terdiri atas satu atau beberapa byte. Adapun byte merupakan susunan dari 8 bit.
- Operasi pada file pada dasarnya meliputi tiga tahapan, yaitu :
  a. Membuka/mengaktifkan file
  b. Melaksanakan proses file
  c. Menutup file
- Sebelum file dapat diakses (dibaca atau ditulisi), mula-mula file haruslah diaktifkan terlebih dahulu dengan menggunakan fungsi *fopen()*.
- Untuk menutup file, fungsi yang digunakan adalah *fclose()*.
- Sebuah karakter dapat disimpan dalam file dengan menggunakan fungsi *fputc()*.
- Untuk melihat isi file digunakan fungsi *fgetc()* yang digunakan untuk pembacaan per karakter dari isi file.
- File biner adalah file yang pola penyimpanan di dalam disk berbentuk biner, yaitu seperti bentuk pada memori RAM (komputer). Sedangkan file teks merupakan file yang pola penyimpanan datanya dalam bentuk karakter.
- Untuk keperluan menyimpan atau membaca membaca file bertipe int, digunakan fungsi *_putw()* dan *_getw()*.
- Fungsi yang digunakan untuk menyimpan atau membaca data file dalam bentuk kesatuan blok (sejumlah byte), misalnya untuk menyimpan data bertipe *float* atau data bertipe *struct* adalah *fread()* dan *fwrite()*.
- Dua fungsi yang dipakai untuk membaca data string pada file yaitu *fgets()* dan *fputs()*.
- Untuk keperluan pengaksesan secara random, fungsi yang digunakan adalah *fseek()*.
- Fungsi yang berguna untuk menghapus file yaitu *remove()*.
- Untuk mengganti nama file, fungsi yang digunakan yaitu *rename()*.

**atihan :**

**Buatlah potongan program untuk soal-soal di bawah ini**

1. Deklarasikan sebuah variabel untuk menangani pemasukan data ke file (misalkan nama variabelnya = **input_file**) yang merupakan sebuah pointer ke sebuah FILE *(pointer to a type FILE)*

2. Dengan menggunakan **input_file**, tuliskan pernyataan untuk membuka sebuah file (misalkan nama filenya = **result.dat**) dengan mode baca *(read)*

3. Tuliskan pernyataan-pernyataan dalam program C untuk mengecek apakah **input_file** berhasil membuka file **result.dat** dengan sukses. Jika tidak, tampilkan pesan kesalahannya dan keluar dari program (*exit*)

4. Tuliskan potongan program dalam bahasa C untuk membaca satu baris karakter (yang diakhiri dengan `\n`) dari **input_file** ke dalam *array of char* (misalkan nama array-nya = **buffer**). Array **buffer** diakhiri dengan NULL setelah mencapai akhir baris (membaca karakter `\n`). Terlebih dahulu deklarasikan semua variabel yang dipakai.

5. Tuliskan pernyataan untuk menutup file *(***result.dat***)* yang terasosiasi dengan **input_ file.**