File Handling In ‘C’

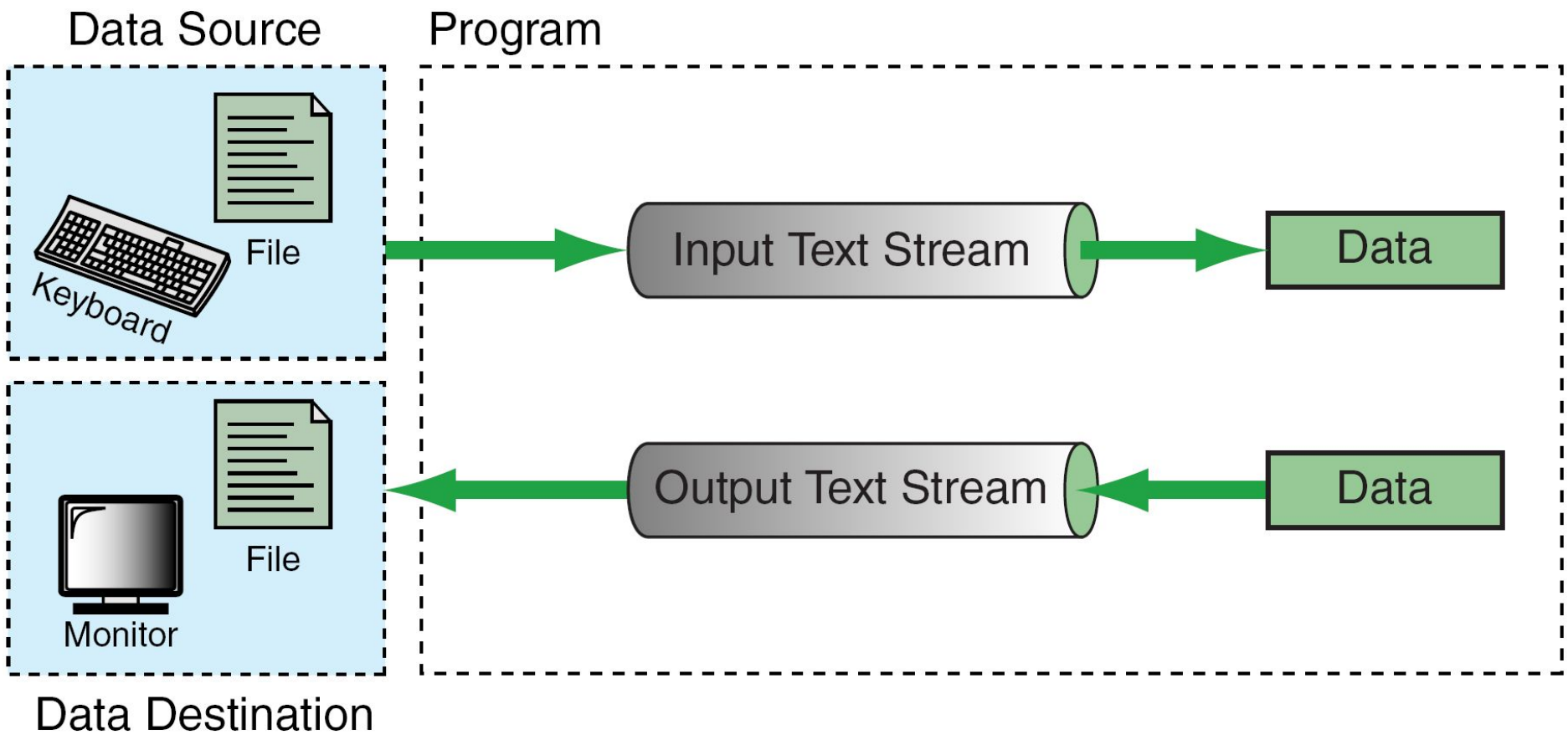
Ritu Devi

Files

- *A file is an external collection of related data treated as a unit. The primary purpose of a file is to keep a record of data. Since the contents of primary memory are lost when the computer is shut down, we need files to store our data in a more permanent form.*

- *data is input to and output from a stream. A stream can be associated with a physical device, such as a terminal, or with a file stored in auxiliary memory.*

Streams


Data Source
Program
File
Keyboard
Input Text Stream
Data
Output Text Stream
Data
File
Monitor
Data Destination

Text And Binary Modes

Text Mode	Binary Mode
In text format, data is stored as lines of characters with each line terminated by a newline character('\n')	In binary format, data stored on the disk in the same way as it is represented in the computer memory
in human readable form and they can be created and read using any text editor	not in human readable form and they can be created and read only by specific program written for them
newline is stored as a combination of carriage return '\r'(ASCII 13) and linefeed '\n'(ASCII 10)	newline is stored only as '\n'(ASCII 10)
format some conversions take place while transferring data between memory and file.(e.g., newline('\n') converted to a combination of ‘\r’ and '\n')	no conversions take place while transferring of data between memory and file. Therefore, input and output operations take less time
More portable than binary format data	Data written in these formats are not very portable since the size of data types and byte order may be different on different machines

File Operations

- Creation of a new file
- Opening an existing file
- Reading from a file
- Writing to a file
- Moving to a specific location in a file (seeking)
- Closing a file

Buffer

- is an area in memory where the data is temporarily stored before being written to the file.
- When we open a file, a buffer is automatically associated with its file pointer.
- The content of buffer is written back to the file in any of the following case
    - The Buffer is full
    - The program is terminated
    - Closing the file using fclose()
    - Using fflush().
- Improves the efficiency

Opening a File

- The process of establishing a connection between the program and file is called *opening the file.*
- A structure named FILE contains all information about the file like *name, status, buffer size, current position, end of file status* etc.

```
typedef  struct{
                    ……….
                    ……….
                    ……….
                }FILE;
```

- **Declaration:** FILE *fopen(const char *filename, const char *mode);

- On success, fopen( ) returns a pointer of type FILE and on error it returns NULL

```
FILE *fp;
fp=fopen(“file1.dat","w");
if(fp==NULL)
{
        printf("Error in opening file");
        exit (1) ;
 }
```

- Error may happen if
  - File does not exist
  - no space on the disk or we don't have write permission
  - OS limits the number of files that can be opened at a time

File Mode

Mode	r	w	a	r+	w+	a+
Open State	read	write	write	read	write	write
Read Allowed	yes	no	no	yes	yes	yes
Write Allowed	no	yes	yes	yes	yes	yes
Append Allowed	no	no	yes	no	no	yes
File Must Exist	yes	no	no	yes	no	no
Contents of Existing File Lost	no	yes	no	no	yes	no

Closing a File

- After closing the file,
  - connection between file and program is broken.
  - all the buffers associated with it are flushed (means all data in buffer written to the file)
- **Declaration:** int fclose(FILE *fptr);
  - returns EOF on error and 0 on success.
- We can also close multiple files by calling a single function fcloseall( ).
- **Declaration:** int fcloseall(void);
  - On error, it returns EOF and on success it returns the number of files closed

- During read operation, when the end of file is reached, the operating system sends an end-of-file signal to the program.
- When the program receives this signal, the *file reading functions* return EOF, which is a constant defined in the file Stdio.h and its value is -1

Structure of a General File Program

```
main( )
{
    FILE  *fp;
    fp=fopen("filename","mode");
    ……..
    ……..
    ……..
    fclose  (  fp  );
}/*End  of  main  */
```

Functions used for file I/O are

Purpose	Functions used
Character I/O	fgetc( ), fputc( ), getc( ), putc( )
String I/O	fgets( ), fputs( )
Integer I/O	getw( ), putw( )
Formatted I/O	fscanf( ), fprintf( )
Record I/O	fread( ), fwrite( )

The stdio.h header file contains several different input/output function declarations.

Character I/O

- **Declaration**: int fputc(int c, FILE *fptr);
  - writes a character to the specified file at the current file position

```
main()
{
  FILE *fp;
   int ch;
   if (( fp=fopen("myfile.txt", "w") )==NULL)
    {
            printf ("File does not exist\n");
            exit( ) ;
    }
   else
    {
          printf ("Enter text : \n" ) ;
          while ( (ch=getchar ( ) ) ! =EOF)
          fputc (ch, fp);
    }
  fclose (fp);
}
```

- **Declaration:** int fgetc(FILE *fptr);

```
main( )
{
    FILE  *p;
    char  ch;
    if((p=fopen("myfile.txt","r"))==NULL)
        printf("This  file  doesn't  exist\n");
    else
    {
        while((ch=fgetc(p))!=EOF)
            printf("%c",ch);
    }
    fclose(p);
}
```

```
ch=fgetc(p);
while(ch!=EOF)
{
    printf("%c",ch);
    fgetc(p);
}
```

Integer I/O

- **Declaration:** int putw(int value, FILE *fptr);
- program to write integers from 1 to 30 into the file "num.dat"

```
Main()
{
 FILE *fptr;
 int value;
  fptr=fopen("num.dat", "wb");
  for(value=1;value<=30;value++)
        putw(value,fptr);
  fclose (fptr) ;
}
```

- **Declaration:** int getw(FILE *fptr);

```
Main()
{
 FILE * fptr;
 int value;
 fptr=fopen("num.dat", "rb");
 while( (value=getw(fptr)) !=EOF)
       printf("%d\t",value);
 fclose(fptr);
}
```

- If *getw( )* is used with text files then it will stop reading if integer 26 is encountered in the file. And program will work correctly if -1 is also not present in file.
- Solution is to use feof() and ferror().

String I/O

- **Declaration:** int fputs(const char *str, FILE *fptr);

```
main ( )
{
 FILE *fptr;
 char str[80j;
 fptr=fopen("test.txt", "w");
 printf ( "Enter the text \n" ) ;
 while (gets (str) ! =NULL)
      fputs(str,fptr);
 fclose(fptr) ;
}
```

- Suppose we enter this text after running the program

  Yesterday is history
  Tomorrow is mystery
  Today is a gift
  That's why it is called Present

- function *gets( )* converts the *newline character* to the *null character* and the array *str* contains "Yesterday is history"(20 characters +1 null character). .
- The null character is not written to the file, so only 20 characters are written. Immediately after the first line of text, the second one is written.
- *puts( )* translates null character to a newline

- **Declaration:** char *fgets(char *str, int n, FILE *fptr);
- reads characters from the file until either a newline or an end of file is encountered till n-1 characters have been read.

```
Main()
{
 FILE * fptr;
 char str[80];
 fptr=fopen("test.txt“, "r");
 while (fgets (str, 80, fptr) ! =NULL)
        puts(str);
 fclose(fptr) ;
}
```

Output: Yesterday is historyTomorrow is mysteryToday is a giftThat's why it is called P
resent

- fgets( ) appends null character after reading first 80 characters
- function puts( ) converts the null character of string to a newline

Formatted I/O

- generally used when there is a need to display data on terminal or print data in some format.
- **Declaration**: fprintf (FILE *fptr, const char *format [, argument, ...]);

```
main()
{
    FILE  *fp;
    char  name[10];
    int  age;
    fp=fopen("rec.dat","w");
    printf("Enter  your  name  and  age  :  "  );
    scanf("%s%d",name,&age);
    fprintf(fp,"My  name  is  %s  and  age  is  %d",name,age);
    fclose(fp);
}
```

```c
#include<stdio.h>
struct student
{   char name[20];
    float marks;
}stu;
main()
{
    FILE *fp;
    int i,n;
    fp=fopen("students.dat","w");
    printf("Enter number of records : ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("Enter name and marks : ");
        scanf("%s%f",stu.name,&stu.marks);
        fprintf(fp,"%s %f",stu.name,stu.marks);
    }
}
```

- **Declaration:** fscanf (FILE *fptr, const char *format [, address, ...]);

```
struct  student
{
    char  name[20];
    float  marks;
}stu;
main( )
{
    FILE  *fopen( ),*fp;

    fp=fopen("students.dat","r");
    printf("NAME\tMARKS\n");

    while(fscanf(fp,"%s%f",stu.name,&stu.marks)!=EOF)
        printf("%s\t%f\n",stu.name,stu.marks);
    fclose(fp);
}
```

Predefined File Pointers

- Predefined constant file pointers are opened automatically when the program is executed.

File pointer	Device
stdin	Standard input device(Keyboard))
stdout	Standard output device(Screen)
stderr	Standard error output device (Screen)

- If we use stdout and stdin in the fprintf( ) and fscanf( ) functions, then these function calls become equivalent to printf( ) and scanf( ).
  - fprintf(stdout, "My age is %s" , age); is equivalent to printf("My age is %d", age);
  - fscanf(stdin, "%s%d"', name, &age); is equivalent to scanf("%s%d", name, &age);

Block Read / Write

- It is easy to read the entire block from file or write the entire block to the file using fread( ) and fwrite( ) functions.
- We can read or write any type of data varying from a single character to arrays and structures using these functions.
- **Declaration:** size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fptr);
  - size_ t is defined in stdio.h as: typedef unsigned int size_t ;
    - On success, fwrite( ) will write n items. On error or end of file it will return a number less than n.

- **Declaration:** size_t fread(void *ptr, size_t size, size_t n, FILE *fptr);

Variable Declaration:

```
float fval;
int arr[10];
struct record {
        char name[20];
        int roll;
        float marks;
}student;
struct record stu_arr[200];
```

Blocks to used in I/O	Writing to file	Reading from file
a single float value	fwrite(&fval, sizeof(float), 1, fp);	fread(&fval, sizeof(float), I, fp);
an array of integers	fwrite(arr, sizeof(arr), 1, fp);	fread(an, sizeof(arr), I, fp);
only first 5 elements	fwrite(arr, sizeof(int), 5, fp);	fread(arr, sizeof(int), 5, fp);
a single structure variable	fwrite(&student,sizeof(student),1,fp);	fread(&student, sizeof(student),1 ,fp);
an array of structures (e.g., stu_arr[200])	fwrite(stu_arr, sizeof(stu_arr), 1, fp);	stu_arr, sizeof(stu_arr), 1, fp);

Random Access To File

- random access file processing in c is performed by using these functions:
    - Fseek()
    - Ftell()
    - Rewind()
- File position pointer points to a particular byte in the file.
- This pointer automatically moves forward when a read or write operation takes place.
- To access the data randomly, we'll have to take control of this position pointer

fseek()

- **Declaration:** int fseek(FILE *fp, long displacement, int origin);

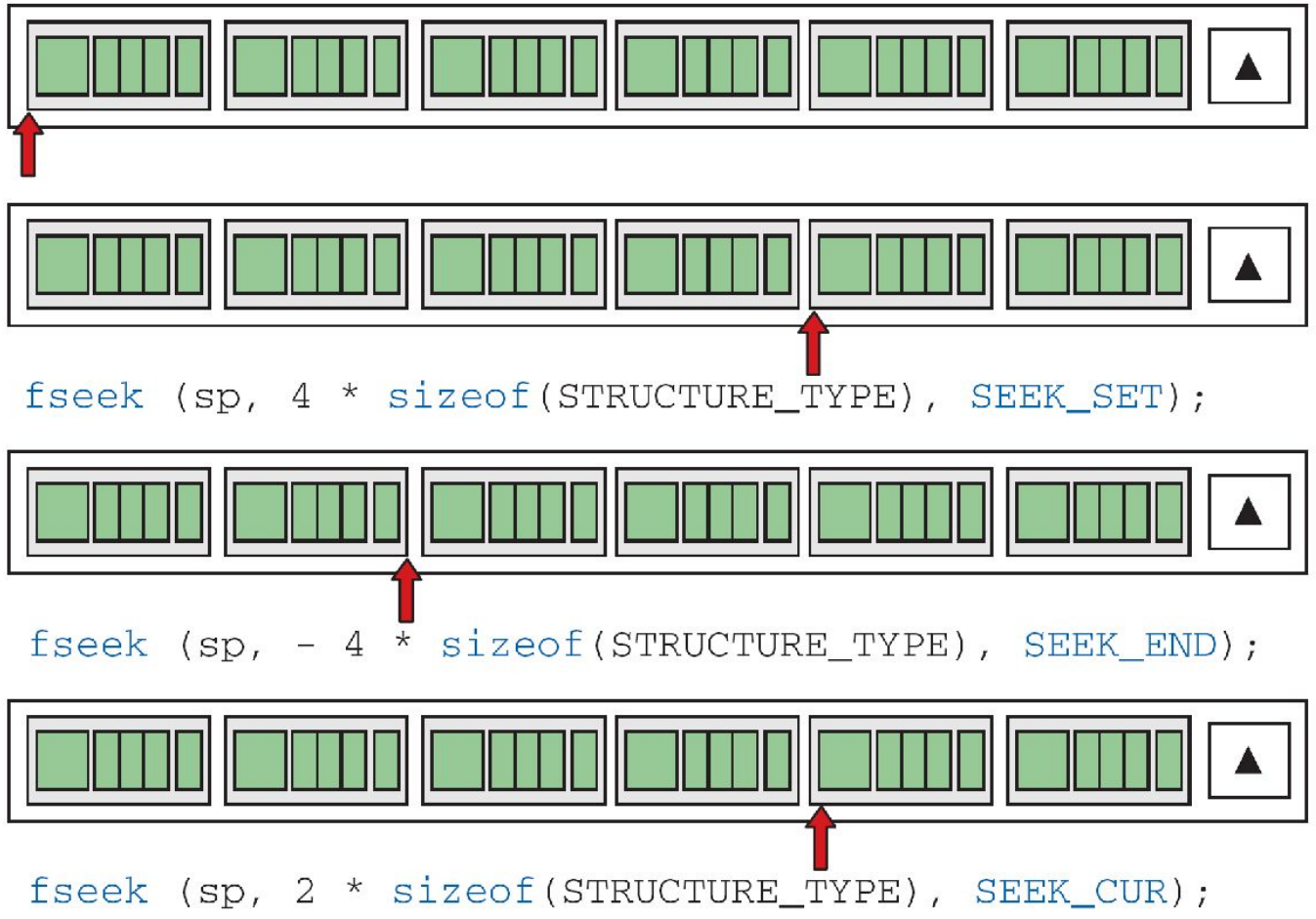


Fig: File Seek Operation

ftell()

- **Declaration:** long ftell(FILE *fptr);

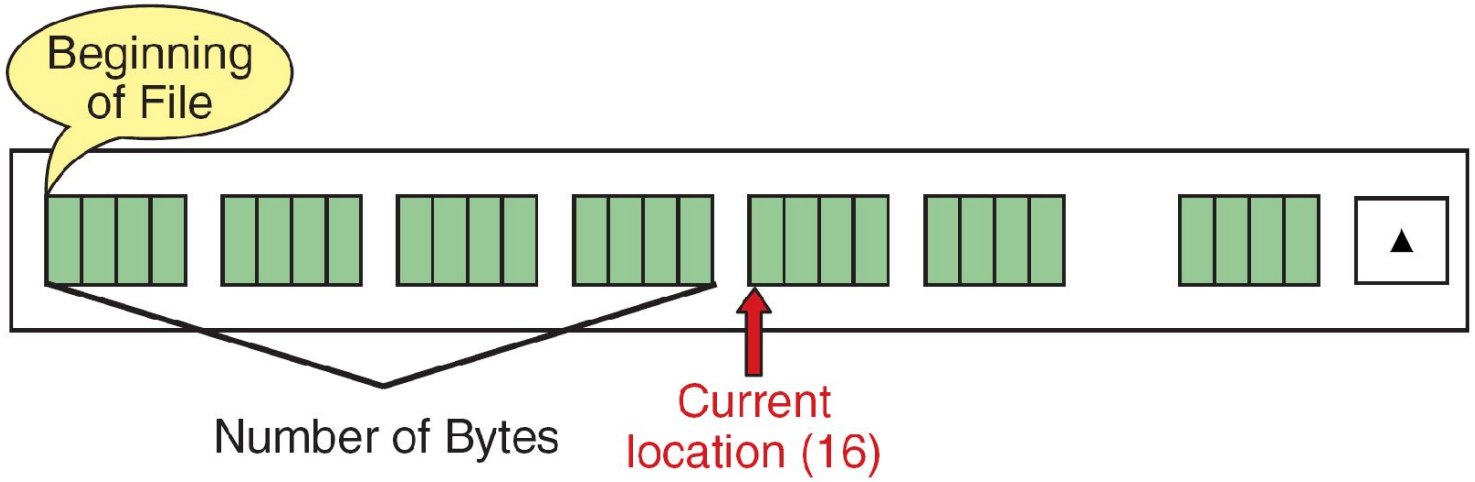
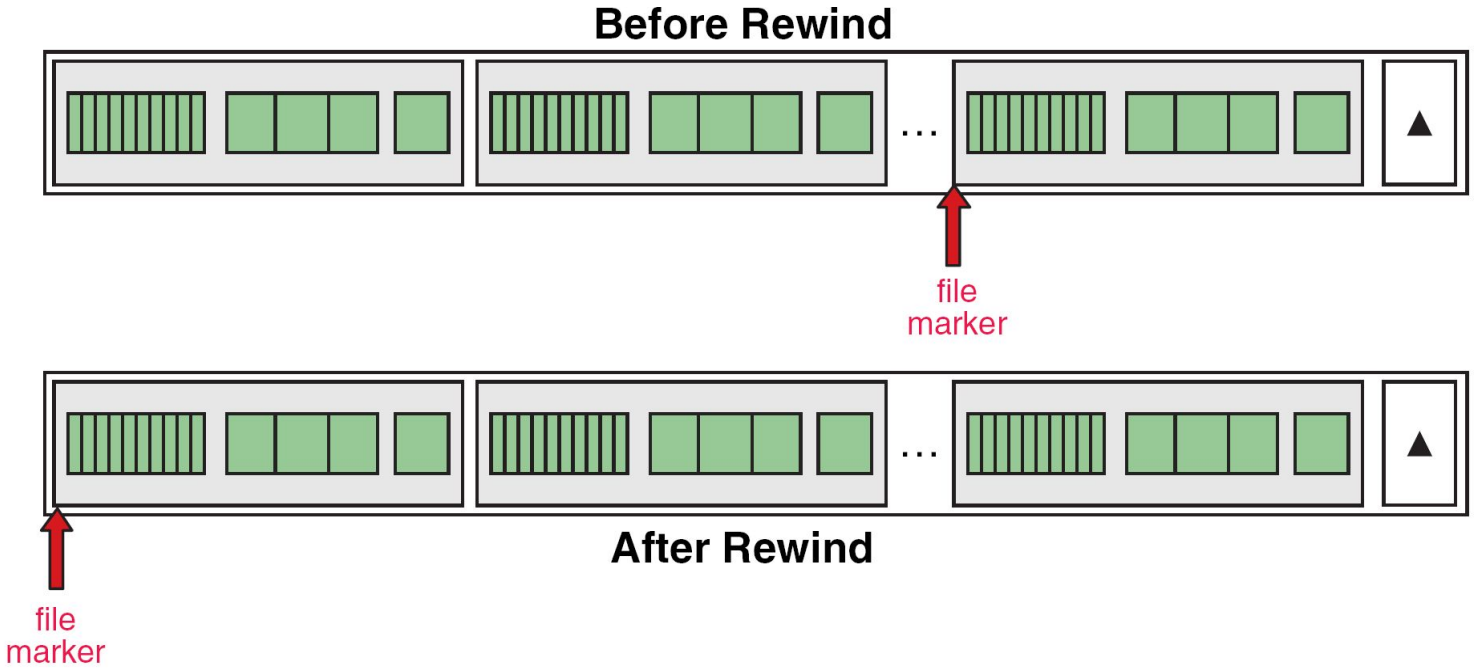


Fig: Current Location (*ftell*) Operation

rewind()

- **Declaration:** rewind(FILE *fptr);



Exercises

1. Write a C language program to read “mark.dat” file containing rollno, name,marks of three subjects and calculate total mark, result in grade and store same in “result.dat” file. (Note : Make use of fread and fwrite functions)
2. Write a C language program to read a cust.dat file containing meter number, name, current reading & previous reading. Read the same file. Calculate unit and total amounts according to the following rules —

Unit	**rate**
0-50	1.00
51-100	1.50
> 100	2.00

Store meter number ,name ,unit & amount in master.dat file.