```
        fp = fopen("std.dat", "a");
        printf ("Enter No. of student records: ");
        scanf ("%d", &n);
        printf("Enter student info:\n<USN><Name><Marks>:\n");
        for (i = 0; i< n; i++)
        {
              scanf("%d%s%d", &s.USN,s.Name,&s.Marks);
              fwrite(&s, sizeof(s), 1, fp);
        }
        fclose(fp);

        /* display all student records */
        fp = fopen("std.dat", "r");
        if (!fp)
        {
              printf("Error in reading file\n");
              exit(1);
        }
        printf("Student Records in the file:\n");
        printf("----------------------------\n");

        /* reads one record from file */
        while (fread(&s,sizeof(s),1,fp))
              printf("%d %s %d\n", s.USN,s.Name,s.Marks);
        fclose(fp);
}
```

Sample Run

```
Enter No. of student records: 2
Enter student info:
<USN><Name><Marks>:
111 Nandagopalan 98
222 Deeapk 46
Student Records in the file:
----------------------------
111 Nandagopalan 98
222 Deeapk 46
```

(1) Since the records are stored sequentially (like an audio cassette), we should be able to add any new student record(s) with out destroying the old contents. Hence, the file *std.dat* is opened in "**a**" (append) mode instead of "**w**" (write) mode.
(2) Each student record read from the user is stored in the disk file using *fwrite()*.
(3) All the student records are displayed reading one record at a time using *fread()*.
(4) After writing to disk file using *fwrite()* don't try to display the contents of the file using DOS's `TYPE` command, because it is a binary write and not text write.

1.13.7 Formatted I/O – fscanf() and fprintf()

When we wish to read/write mixed type of data the best method is to use formatted I/O. This is achieved by using *fscanf()* and *fprintf()*. Assume that a data file contains some integer data (file has been created with an editor) that is to be processed. To read and store it back with integer (or any other standard data type) data type we use *fscanf()* and *fprintf()*. The syntax of these two functions are shown below:

```
int fscanf (FILE *fp, char *format, arg-list);
int printf (FILE *fp, char *format, arg-list);
```

Parameters:

`fp` – file pointer of opened stream.
`format` - *%d*, *%f*, etc.
`arg-list` – list of variables.

Return value:

⇒ On success, it returns number of variables read.
⇒ If returned value is 0, no fields were stored.

To understand the working of these two functions see Program 1.44

Program 1.44
To add two integers from a data file

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
     FILE *fp;
     int x, y, z = 0;
     fp = fopen("in.dat", "r");
     if (fp == NULL)
     {
      printf("Error in opening file <press Enter key>\n");
      exit(1);
     }
     while (!feof(fp))
     {
          fscanf(fp, "%d%d", &x, &y);
          z = x + y;
     }
     printf("Result=%d+%d=%d\n", x, y, z);
}
```

Sample Run

```
Assuming contents of <in.dat> as:
10 20
Result=10+20=30
```

You can use *fprintf* to store the result or any formatted data to a disk file.

1.13.8 Additional Examples

This section gives some more programs for I/O file operations.

Program 1.45
Merge two text files and store it in another file

```
/* Program to copy file1 & file2 into file3 */
/* file3 = file1 + file2                    */
#include <stdio.h>
#include <stdlib.h>
void main()
{
     FILE *infile1, *infile2, *outfile;
     char fname1[25],fname2[25];
     int c;

     printf("Enter the 1st file name : ");
     scanf("%s",fname1);
     infile1 = fopen(fname1,"r");
     if (infile1 == NULL)
     {
          printf("Error in opening file\n");
          exit(1);
     }

     printf("Enter the 2nd file name : ");
     scanf("%s", fname2);
     infile2 = fopen(fname2, "r");
     if (infile2 == NULL)
     {
          printf("Error in opening file\n");
          exit(1);
     }

     /* file is read, write and append */
```

```
    outfile = fopen("out.txt", "a+");
    if (outfile == NULL)
    {
         printf("Error in opening file\n");
         exit(1);
    }

    while ((c = getc(infile1)) != EOF)
         putc(c, outfile);

    while ((c = getc(infile2)) != EOF)
         putc(c, outfile);

    rewind(outfile);

    printf("The Merged file contents:\n\n");
    while ((c = getc(outfile)) != EOF)
         putc(c, stdout); /* printing on the console */

}
```

Sample Run

```
Enter the 1st file name : f1.txt
Enter the 2nd file name : f2.txt
The Merged file contents:

File-1 contents
File-2 contents
```

This program assumes that *f1.txt* contains "`File-1 contents`" and *f2.txt* contains "`File-2 contents`". The merged file out.txt also will contain (not shown) the same contents.

1.13.9 Random Access - fseek() function

fseek function is used to move the file pointer within the opened file. Using this function we can access records in a file randomly. The function *ftell()* gives the current position of the file pointer during run time. The syntax of *fseek()* is shown below:

```
int fseek(FILE *fp, long offset, int ref-origin);
```

Parameters:

`fp` – file pointer.

`offset` – location of file pointer mentioned in *long int*– positive or negative (explained below).
`ref-origin` – 0/1/2 (explained below).

The *fseek* moves the file pointer to a new position based on the following formula:

```
file pointer = ref-origin + offset
```

The following table gives the ref-origin values:

Ref-origin value	Constant	Location of fp
0	SEEK_SET	Beginning of the file
1	SEEK_CUR	Current position of the file pointer
2	SEEK_END	End of the file

For example,
(1) `fseek(fp, 20L, 0)` or `fseek(fp, 20L, SEEK_SET)` moves the file pointer `fp` to a position in the file after 20 bytes in the forward direction.
(2) `fseek(fp, -20L, SEEK_END)` repositions the file pointer to 20 bytes backward from end of the file.

Program 1.46
Demonstration of fseek()

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

     fp = fopen("t.txt","r");
     if (fp == NULL)
     {
          printf("Error in opening file\n");
          exit(1);
     }
     fseek(fp,10L, SEEK_SET);
     printf("The contents of file
               after seeking 10 bytes:\n");
     while ((c = getc(fp)) != EOF)
          putc(c,stdout);

     rewind(fp);
```

```
    /* positions fp to the beginning of the file */
    fseek(fp, -12L, SEEK_END);

    printf("The contents of file after
              seeking 12 bytes from backwards:\n");
    /* 12 bytes include the file terminator
         characters of 2 bytes */
    while ((c = getc(fp)) != EOF)
         putc(c,stdout);
}
```

Sample Run

```
Assuming contents of "t.txt" as:
Bangalore Institute of Technology
The contents of file after seeking 10 bytes:
Institute of Technology
The contents of file after seeking 12 bytes from    backwards:
Technology
```

1.13.10 ftell() function

The *ftell()* tells where the file pointer is? It accepts the file pointer of the opened stream as the only argument. See the below definition:

```
long ftell(FILE *fp);
```

The value returned by *ftell* can be used in a subsequent call to *fseek*.

Parameters:

fp – file pointer.

Return Value:

⇒ On success, returns the current file pointer position.
⇒ On error, returns -1L and sets **errno** to a positive value.

To illustrate the meaning of *ftell* take a look at the Program 1.47.

Program 1.47
Demonstration of ftell()

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main()
```

```
{
     FILE *fp;
     int c;

     clrscr();
     fp = fopen("t.txt","r");
     if (fp == NULL)
     {
          printf("Error in opening file\n");
          exit(1);
     }

     printf("Position of file pointer
               before fseek: %d\n",ftell(fp));
     fseek(fp, 6L, SEEK_SET);
     printf("char\tpos\n");
     while (!feof(fp))
          printf("%c\t%d\n", fgetc(fp), ftell(fp));

     /* print first 5 characters */
     fseek(fp,0L, SEEK_SET);
     printf("Printing first 5 characters:\n");
     while (ftell(fp) < 5)
          printf("%c", fgetc(fp));
     printf("\n");
}
```

Sample Run

```
Assuming contents of t.txt as:
Nanda Gopalan
Position of file pointer before fseek: 0
char    pos
G       6
o       7
p       8
a       9
l       10
a       11
n       12
Printing first 5 characters:
Nanda
```

As demonstrated in Program 1.47, *ftell* returns the current file pointer position as an integer that has been used in *printf*. Notice that the initial value of *fp* will be 0 as *ftell* tells in the first *printf* statement. To print the first 5 characters from the input file, we

have used *ftell* as an index or pointer to go through the file. We do not increment *ftell* as after *fgetc* the file pointer is automatically incremented and points to the next character in the file.

1.13.11 Error handling in files – ferror() and perror()

While operating with files there are several occasions for the programmer to make errors. Every programming language provides built-in error detection and default error handlers. For example, "*divide by zero*" is a common type of error that could be committed in data processing. Even during operations of files programmers may unknowingly write into a file when it is opened for reading and vice-versa.

C allows you to catch this type of error using a standard function called **ferror()**. The syntax is shown below:

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

ferror is a macro that tests the given file pointer for a read or write error. If the file's error indicator has been set, it remains set until *clearerr* or *rewind* is called, or until the file is closed.

Return Value:

ferror returns non-zero if an error was detected on the opened file.

In this category, another useful function is **perror** that prints to the console *error* message for the last library routine that produced the error. It prints the file name, a colon, and the message corresponding to the current value of *errno*, then a *newline*.

Generally, we pass the file name of the program as the argument to *perror()*. The example Program 1.48 demonstrates the working of both these functions.

Program 1.48
Catching error using ferror

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

     clrscr();
     fp = fopen("t.txt","r");
     if (fp == NULL)
```