Example 12.1

Write a program to read data from the keyboard, write it to a file called **INPUT,** again read the same data from the **INPUT** file, and display it on the screen.

A program and the related input and output data are shown in Fig.12.1. We enter the input data via the keyboard and the program writes it, character by character, to the file **INPUT**. The end of the data is indicated by entering an **EOF** character, which is *control-Z* in the reference system. (This may be control-D in other systems). The file INPUT is closed at this signal.

WRITING TO AND READING FROM A FILE

Program

```
#include  <stdio.h>

main()
{
    FILE *f1;
    char c;
    printf("Data Input\n\n");
    /* Open the file INPUT */
    f1 = fopen("INPUT", "w");

    /* Get a character from keyboard   */
    while((c=getchar()) != EOF)

        /* Write a character to INPUT  */
        putc(c,f1);
    /* Close the file INPUT   */
    fclose(f1);
    printf("\nData Output\n\n");
    /* Reopen the file INPUT    */
    f1 = fopen("INPUT","r");

   /* Read a character from INPUT*/
    while((c=getc(f1)) != EOF)

       /* Display a character on screen */
        printf("%c",c);

    /* Close the file INPUT       */
    fclose(f1);
}
```

Output

```
Data Input
This is a program to test the file handling
features on this system^Z

Data Output
This is a program to test the file handling
features on this system
```

***Fig.12.1** Character oriented read/write operations on a file*

Example 12.2

A file named **DATA** contains a series of integer numbers. Code a program to read these numbers and then write all 'odd' numbers to a file to be called **ODD** and all `even' numbers to a file to be called **EVEN.**

The program is shown in Fig.12.2. It uses three files simultaneously and therefore we need to define three-file pointers **f1, f2** and **f3.**

First, the file DATA containing integer values is created. The integer values are read from the terminal and are written to the file **DATA** with the help of the statement

putw(number, f1);

Notice that when we type -1, the reading is terminated and the file is closed. The next step is to open all the three files, **DATA** for reading, **ODD** and **EVEN** for writing. The contents of **DATA** file are read, integer by integer, by the function **getw(f1)** and written to **ODD** or **EVEN** file after an appropriate test. Note that the statement

(number = getw(f1)) != EOF

reads a value, assigns the same to **number**, and then tests for the end-of-file mark.

Finally, the program displays the contents of ODD and EVEN files. It is important to note that the files **ODD** and **EVEN** opened for writing are closed before they are reopened for reading.

HANDLING OF INTEGER DATA FILES

Program

```
#include  <stdio.h>
main()
{
    FILE  *f1, *f2, *f3;
    int   number, i;

    printf("Contents of DATA file\n\n");
    f1 = fopen("DATA", "w");      /* Create DATA file    */
    for(i = 1; i <= 30; i++)
    {
       scanf("%d", &number);
       if(number == -1) break;
       putw(number,f1);
    }
    fclose(f1);

    f1 = fopen("DATA", "r");
    f2 = fopen("ODD", "w");
    f3 = fopen("EVEN", "w");

    /* Read from DATA file */
    while((number = getw(f1)) != EOF)
    {
        if(number %2 == 0)
```

```
            putw(number, f3);   /*  Write to EVEN file  */
          else
            putw(number, f2);   /*  Write to ODD file   */
      }
      fclose(f1);
      fclose(f2);
      fclose(f3);

      f2 = fopen("ODD","r");
      f3 = fopen("EVEN", "r");
      printf("\n\nContents of ODD file\n\n");

      while((number = getw(f2)) != EOF)
         printf("%4d", number);
      printf("\n\nContents of EVEN file\n\n");

      while((number = getw(f3)) != EOF)
         printf("%4d", number);

      fclose(f2);
      fclose(f3);

}
```

Output

```
Contents of DATA file
111 222 333 444 555 666 777 888 999 000 121 232 343 454 565 -1

Contents of ODD file
111 333 555 777 999 121 343 565

Contents of EVEN file
222 444 666 888   0 232 454
```

Fig.12.2 Operations on integer data

Example 12.3

Write a program to open a file named INVENTORY and store in it the following data:

Item name	Number	Price	Quantity
AAA-1	111	17.50	115
BBB-2	125	36.00	75
C-3	247	31.75	104

Extend the program to read this data from the file INVENTORY and display the inventory table with the value of each item.

The program is given in Fig.12.3. The filename INVENTORY is supplied through the keyboard. Data is read using the function **fscanf** from the file **stdin,** which refers to the terminal and it is then written to the file that is being pointed to by the file pointer **fp.** Remember that the file pointer **fp** points to the file INVENTORY.

After closing the file INVENTORY, it is again reopened for reading. The data from the file, along with the item values are written to the file **stdout,** which refers to the screen. While reading from a file, care should be taken to use the same format specifications with which the contents have been written to the file.…é

HANDLING OF FILES WITH MIXED DATA TYPES
(fscanf and fprinf)

Program

```
#include  <stdio.h>

main()
{
    FILE  *fp;
    int    number, quantity, i;
    float  price, value;
    char   item[10], filename[10];

    printf("Input file name\n");
    scanf("%s", filename);

    fp = fopen(filename, "w");

    printf("Input inventory data\n\n");
    printf("Item name  Number   Price   Quantity\n");
    for(i = 1; i <= 3; i++)
    {
       fscanf(stdin, "%s %d %f %d",
                     item, &number, &price, &quantity);
       fprintf(fp, "%s %d %.2f %d",
                     item, number, price, quantity);
    }
    fclose(fp);
    fprintf(stdout, "\n\n");

    fp = fopen(filename, "r");

    printf("Item name  Number   Price   Quantity    Value\n");
    for(i = 1; i <= 3; i++)
    {
       fscanf(fp, "%s %d %f d",item,&number,&price,&quantity);
       value = price * quantity;
       fprintf(stdout, "%-8s %7d %8.2f %8d %11.2f\n",
                      item, number, price, quantity, value);
    }
    fclose(fp);
}
```

Output

```
Input file name
INVENTORY
Input inventory data

Item name  Number   Price   Quantity
AAA-1  111  17.50  115
BBB-2  125  36.00  75
C-3    247  31.75  104

Item name  Number   Price   Quantity    Value
AAA-1        111    17.50      115     2012.50
BBB-2        125    36.00       75     2700.00
C-3          247    31.75      104     3302.00
```

Fig.12.3 *Operations on mixed data types*

Example 12.4

Write a program to illustrate error handling in file operations.

The program shown in Fig.12.4 illustrates the use of the **NULL** pointer test and **feof** function. When we input filename as TETS, the function call

fopen("TETS", "r");

returns a **NULL** pointer because the file TETS does not exist and therefore the message "Cannot open the file" is printed out.

Similarly, the call **feof(fp2)** returns a non-zero integer when the entire data has been read, and hence the program prints the message "Ran out of data" and terminates further reading.

ERROR HANDLING IN FILE OPERATIONS

Program

```
#include  <stdio.h>

main()
{
   char  *filename;
    FILE  *fp1, *fp2;
    int   i, number;

    fp1 = fopen("TEST", "w");
```

```
        for(i = 10; i <= 100; i += 10)
           putw(i, fp1);

        fclose(fp1);

        printf("\nInput filename\n");

    open_file:
        scanf("%s", filename);

        if((fp2 = fopen(filename,"r")) == NULL)
        {
           printf("Cannot open the file.\n");
           printf("Type filename again.\n\n");
           goto open_file;
        }

        else

        for(i = 1; i <= 20; i++)
        {  number = getw(fp2);
           if(feof(fp2))
           {
              printf("\nRan out of data.\n");
              break;
           }
           else
              printf("%d\n", number);
        }

        fclose(fp2);
    }
```

Output

```
Input filename
TETS
Cannot open the file.
Type filename again.

TEST
10
20
30
40
50
```

```
60
70
80
90
100

Ran out of data.
```

Fig.12.4 *Illustration of error handling*

Example 12.5

Write a program that uses the functions **ftell** and **fseek.**

A program employing **ftell** and **fseek** functions is shown in Fig.12.5. We have created a file **RANDOM** with the following contents:

Position ----> 0 1 2 25

Character
stored ----> A B C . . . . . .Z

We are reading the file twice. First, we are reading the content of every fifth position and printing its value along with its position on the screen. The second time, we are reading the contents of the file from the end and printing the same on the screen.

During the first reading, the file pointer crosses the end-of-file mark when the parameter **n** of **fsee(fp,n,0)** becomes 30. Therefore, after printing the content of position 30, the loop is terminated.

For reading the file from the end, we use the statement

fseek(fp,-1L,2);

to position the file pointer to the last character. Since every read causes the position to move forward by one position, we have to move it back by two positions to read the next character. This is achieved by the function

fseek(fp, -2L, 1);

in the while statement. This statement also tests whether the file pointer has crossed the file boundary or not. The loop is terminated as soon as it crosses it.

ILLUSTRATION OF fseek & ftell FUNCTIONS

Program

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

```
    fp = fopen("RANDOM", "w");
    while((c = getchar()) != EOF)
        putc(c,fp);

    printf("No. of characters entered = %ld\n", ftell(fp));
    fclose(fp);
    fp = fopen("RANDOM","r");
    n = 0L;

    while(feof(fp) == 0)
    {
        fseek(fp, n, 0);  /*  Position to (n+1)th character */
        printf("Position of %c is %ld\n", getc(fp),ftell(fp));
        n = n+5L;
    }
    putchar('\n');

    fseek(fp,-1L,2);     /*  Position to the last character */
      do
      {
          putchar(getc(fp));
      }
      while(!fseek(fp,-2L,1));
      fclose(fp);
}
```

Output

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ^Z
No. of characters entered = 26
Position of A is 0
Position of F is 5
Position of K is 10
Position of P is 15
Position of U is 20
Position of Z is 25
Position of   is 30

ZYXWVUTSRQPONMLKJIHGFEDCBA
```

Fig.12.5 *Illustration of* ***fseek*** *and* ***ftell*** *functions*

Example 12.6

Write a program to append additional items to the file INVENTORY and print the total contents of the file.

The program is shown in Fig.12.6. It uses a structure definition to describe each item and a function **append()** to add an item to the file.

On execution, the program requests for the filename to which data is to be appended. After appending the items, the position of the last character in the file is assigned to **n** and then the file is closed.

The file is reopened for reading and its contents are displayed. Note that reading and displaying are done under the control of a **while** loop. The loop tests the current file position against n and is terminated when they become equal.

APPENDING ITEMS TO AN EXISTING FILE

Program

```
#include  <stdio.h>

struct invent_record
{
    char   name[10];
    int    number;
    float  price;
    int    quantity;
};

main()
{
    struct invent_record item;
    char  filename[10];
    int   response;
    FILE  *fp;
    long  n;

    void append (struct invent_record 8x, file *y);

    printf("Type filename:");
    scanf("%s", filename);

    fp = fopen(filename, "a+");
    do
    {
       append(&item, fp);
       printf("\nItem %s appended.\n",item.name);
       printf("\nDo you want to add another item\
        (1 for YES /0 for NO)?");
       scanf("%d", &response);
    }  while (response == 1);

    n = ftell(fp);      /* Position of last character  */
    fclose(fp);

    fp = fopen(filename, "r");
```

```
        while(ftell(fp) < n)
        {
           fscanf(fp,"%s %d %f %d",
           item.name, &item.number, &item.price, &item.quantity);
           fprintf(stdout,"%-8s %7d %8.2f %8d\n",
           item.name, item.number, item.price, item.quantity);
        }
        fclose(fp);
    }
    void append(struct invent_record *product, File *ptr)
    {
        printf("Item name:");
        scanf("%s", product->name);

        printf("Item number:");
        scanf("%d", &product->number);

        printf("Item price:");
        scanf("%f", &product->price);

        printf("Quantity:");
        scanf("%d", &product->quantity);

        fprintf(ptr, "%s %d %.2f %d",
                     product->name,
                     product->number,
                     product->price,
                     product->quantity);
    }
```

Output

```
Type filename:INVENTORY
Item name:XXX
Item number:444
Item price:40.50
Quantity:34

Item XXX appended.
Do you want to add another item(1 for YES /0 for NO)?1
Item name:YYY
Item number:555
Item price:50.50
Quantity:45

Item YYY appended.
Do you want to add another item(1 for YES /0 for NO)?0
AAA-1        111    17.50      115
BBB-2        125    36.00       75
C-3          247    31.75      104
XXX          444    40.50       34
YYY          555    50.50       45
```

Fig.12.6 *Adding items to an existing file*

Example 12.7

Write a program that will receive a filename and a line of text as command line arguments and write the text to the file.

Figure 12.7 shows the use of command line arguments. The command line is

```
F12_7 TEXT AAAAAA BBBBBB CCCCCC DDDDDD EEEEEE FFFFFF GGGGGG
```

Each word in the command line is an argument to the **main** and therefore the total number of arguments is 9.

The argument vector **argv[1]** points to the string TEXT and therefore the statement

```
fp = fopen(argv[1], "w");
```

opens a file with the name TEXT. The **for** loop that follows immediately writes the remaining 7 arguments to the file TEXT.

COMMAND LINE ARGUMENTS

Program

```
#include  <stdio.h>

main(argc, argv)
int  argc;                    /*   argument count           */
char *argv[];                 /*   list of arguments        */
{
    FILE   *fp;
    int  i;
    char word[15];

    fp = fopen(argv[1], "w"); /* open file with name argv[1] */
    printf("\nNo. of arguments in Command line = %d\n\n",argc);
    for(i = 2; i < argc; i++)
       fprintf(fp,"%s ", argv[i]); /* write to file argv[1]  */
    fclose(fp);

/*  Writing content of the file to screen                    */

    printf("Contents of %s file\n\n", argv[1]);
    fp = fopen(argv[1], "r");
    for(i = 2; i < argc; i++)
    {
       fscanf(fp,"%s", word);
       printf("%s ", word);
    }
```

```
    fclose(fp);
    printf("\n\n");

/*  Writing the arguments from memory */

    for(i = 0; i < argc; i++)
       printf("%*s \n", i*5,argv[i]);
}
```

Output

```
C>F12_7 TEXT AAAAAA BBBBBB CCCCCC DDDDDD EEEEEE FFFFFF GGGGG

No. of arguments in Command line = 9

Contents of TEXT file

AAAAAA BBBBBB CCCCCC DDDDDD EEEEEE FFFFFF GGGGGG

C:\C\F12_7.EXE
 TEXT
    AAAAAA
         BBBBBB
              CCCCCC
                   DDDDDD
                        EEEEEE
                             FFFFFF
                                  GGGGGG
```

Fig.12.7 *Use of command line arguments*