Chapter 10

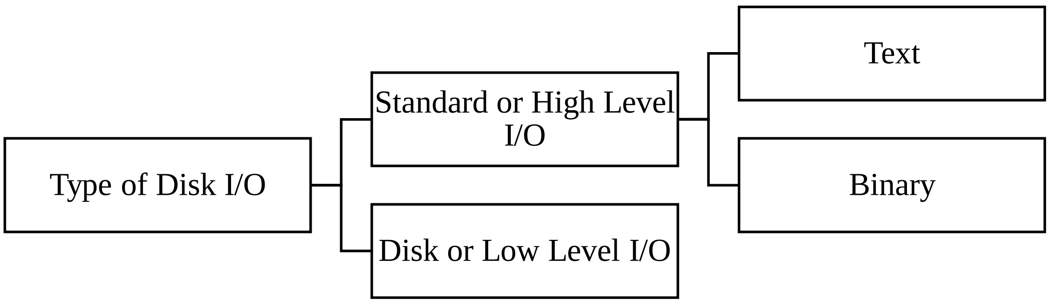
Data Files

A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices.

The input/output functions like *printf(), scanf(), getchar(), putchar(), gets(), puts()* are known as ***console oriented I/O functions*** which always use keyboard for input device and computer screen or monitor for output device.

Many applications require the information to be written or read from an auxiliary memory device. Such information is stored on the memory device in the form of a data file. Thus, data files allow us to store information permanently, and to access and alter that information whenever necessary.

Disk File I/O



Type of Disk I/O

1. Standard or High Level I/O
    a. Text
        i. Record I/O, Formatted I/O, String I/O, Character I/O
    b. Binary
2. Disk or Low Level I/O

Types of I/Os are summarized as:

	Record I/O	**Formatted I/O**	**String I/O**	**Character I/O**
Standard I/O	fread()	fscanf()	fgets()	fgetc()
	fwrite()	fprintf()	fputs()	fputc()
System I/O	fread()			
	fwrite()			

In C, an extensive set of library functions are available for creating and processing these files. There are two different types of data files:

1. Stream-oriented (or standard) data files
    a. Text files
    b. Unformatted data files

2. System-oriented (or low-level) data files

1. **Stream-oriented data files** are divided into two categories:
    - **a. Text files**
        - Consists of consecutive characters
        - Characters can be interpreted as individual, or as components of strings or numbers
    - **b. Unformatted data files**
        - Organizes data into blocks containing contiguous bytes of information
        - These blocks represents more complex data structures, such as array and structures.
2. **System-oriented data files**
    - are more closely related to the computer's operating system
    - more complicated to work with
    - a separate set of procedures, library functions are require to process these files

When dealing with files, there are mainly two types of files: Text files & Binary files

1. Text files

Text files are the normal .txt files that you can easily create using Notepad or any simple text editors. When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents. They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold higher amount of data, are not readable easily and provides a better security than text files.

Library Functions for Reading/Writing from/to a File

The most common functions are shown below. They require the header file **stdio.h** to be included:

Name	**Function**
fopen()	Opens a file
fclose()	Closes a file
putc()	Writes a character to a file
fputc()	Same as *putc()*
getc()	Reads a character from a file
fgetc()	Same as *getc()*
fgets()	Sends a string from a file to program
fputs()	Writes a string to a file
fseek()	Seeks to a specified byte in a file
ftell()	Returns the current file position
fprintf()	is to a file what *printf()* is to the console
fscanf()	is to a file what *scanf()* is to the console
feof()	Returns true if end-of-file is reached
ferror()	Returns true if an error has occurred
rewind()	Resets the file position indicator to the beginning
remove()	Erases a file
fflush()	Flushes a file

File Operations

In C, following operations can be performed on the file, either text or binary:

- Creating a new file
- Opening an existing file
- Closing a file
- Reading from a file
- Writing information to a file
- Searching information in a file

The FILE Pointer

The FILE pointer is the common thread that unites the C I/O system. A file pointer is a pointer to a structure of type FILE. It points the information that defines various things about the file, including its name, status, and the current position of the file.

In order to read or write files, your program needs to use file pointers.

*FILE *fp;*

Opening a file - for creation and edit

Opening a file is performed using the library function in the "stdio.h" header file: fopen(). The syntax for opening a file in standard I/O is:

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

For Example:

```
fopen("E:\\cprogram\\newprogram.txt","w");

fopen("E:\\cprogram\\oldprogram.bin","rb");
```

- Let's suppose the file newprogram.txt doesn't exist in the location E:\cprogram. The first function creates a new file named newprogram.txt and opens it for writing as per the mode 'w'. The writing mode allows you to create and edit (overwrite) the contents of the file.
- Now let's suppose the second binary file oldprogram.bin exists in the location E:\cprogram. The second function opens the existing file for reading in binary mode 'rb'. The reading mode only allows you to read the file, you cannot write into the file.

File Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Note: *Strings like "rb+" may also be represented as "r+b"*

Closing a File

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using library function fclose().

```
fclose(fptr); //fptr is the file pointer associated with file to be closed.
```

Reading and Writing to a Text File

For reading and writing to a text file, we use the functions fprintf() and fscanf(). They are just the file versions of printf() and scanf(). The only difference is that, fprint and fscanf expects a pointer to the structure FILE.

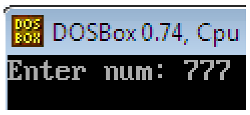
Writing to a text file

Example#1: Write to a text file using fprintf()

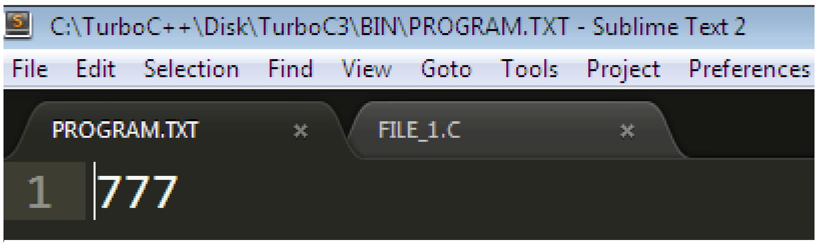
```
#include <stdio.h>
#include <conio.h>
void main(){
        int num;
        FILE *fptr;
        clrscr();
        fptr = fopen("program.txt","w");

        if(fptr == NULL){
          clrscr();
          printf("Error!");
          getch();
          exit(1);
        }
        printf("Enter num: ");
        scanf("%d",&num);

        fprintf(fptr,"%d",num);
        fclose(fptr);
        getch();
}
```



```
Enter num: 777
```


C:\TurboC++\Disk\TurboC3\BIN\PROGRAM.TXT - Sublime Text 2
File Edit Selection Find View Goto Tools Project Preferences
PROGRAM.TXT
FILE_1.C

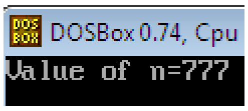
```
777
```

This program takes a number from user and stores in the file program.txt. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open the file, you can see the integer you entered.

Reading from a text file

Example 2: Read from a text file using fscanf()

```
#include <stdio.h>
#include <conio.h>
void main(){
        int num;
        FILE *fptr;
        clrscr();
        if ((fptr = fopen("program.txt","r")) == NULL){
          printf("Error! opening file");
          exit(1); // Program exits if the file pointer returns NULL.
        }
        fscanf(fptr,"%d", &num);
        printf("Value of n=%d", num);
        getch();
}
```



```
Value of n=777
```

This program reads the integer present in the program.txt file and prints it onto the screen. If you successfully created the file from Example 1, running this program will get you the integer you entered. Other functions like fgetchar(), fputc() etc. can be used in similar way.

Reading and Writing to a Binary File

Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the function fwrite(). The functions takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

```
fwrite(address_data,size_data,numbers_data,pointer_to_file);
```

Example 3: Writing to a binary file using fwrite()

```
#include <stdio.h>
#include <conio.h>
struct threeNum{
  int n1, n2, n3;
};
void main(){
        int n;
        struct threeNum num;
        FILE *fptr;
        if ((fptr = fopen("program.bin","wb")) == NULL){
          printf("Error! opening file");
          exit(1); // Program exits if the file pointer returns NULL.
        }
        for(n = 1; n < 5; ++n){
          num.n1 = n;
          num.n2 = 5*n;
          num.n3 = 5*n + 1;
          fwrite(&num, sizeof(struct threeNum), 1, fptr);
        }
        fclose(fptr);
        getch();
}
```

In this program, you create a new file program.bin in the C drive. We declare a structure threeNum with three numbers - n1, n2 and n3, and define it in the main function as num. Now, inside the for loop, we store the value into the file using fwrite. The first parameter takes the address of num and the second parameter takes the size of the structure threeNum. Since, we're only inserting one instance of num, the third parameter is 1. And, the last parameter *fptr points to the file we're storing the data. Finally, we close the file.

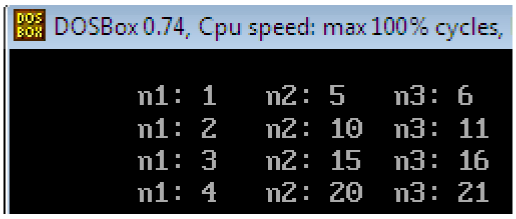
Reading from a binary file

Function fread() also take 4 arguments similar to fwrite() function as above.

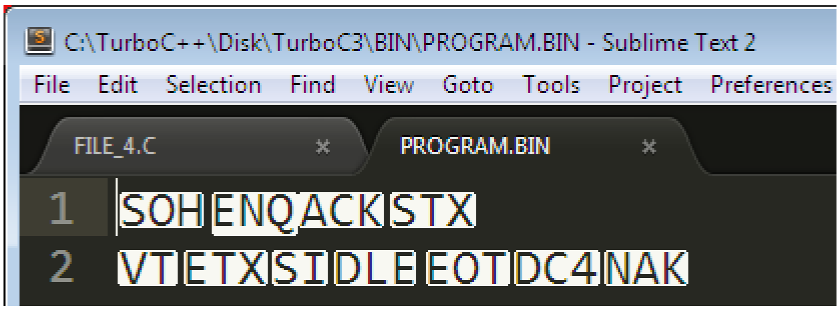
```
fread(address_data,size_data,numbers_data,pointer_to_file);
```

Example 4: Reading from a binary file using fread()

```
#include <stdio.h>
#include <conio.h>
struct threeNum{
  int n1, n2, n3;
};
void main(){
        int n;
        struct threeNum num;
        FILE *fptr;
        clrscr();
        if ((fptr = fopen("program.bin","rb")) == NULL){
          printf("Error! opening file");
          exit(1); // Program exits if the file pointer returns NULL.
        }
        for(n = 1; n < 5; ++n){
          fread(&num, sizeof(struct threeNum), 1, fptr);
          printf("\n\tn1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);
        }
        fclose(fptr);
        getch();
}
```



```
        n1: 1   n2: 5   n3: 6
        n1: 2   n2: 10  n3: 11
        n1: 3   n2: 15  n3: 16
        n1: 4   n2: 20  n3: 21
```


C:\TurboC++\Disk\TurboC3\BIN\PROGRAM.BIN - Sublime Text 2
File Edit Selection Find View Goto Tools Project Preferences
FILE_4.C
PROGRAM.BIN

```
SOHENQACKSTX
VTETXSIDLEEOTDC4NAK
```

In this program, you read the same file program.bin and loop through the records one by one. In simple terms, you read one threeNum record of threeNum size from the file pointed by **fptr* into the structure *num*. You'll get the same records you inserted in Example 3.

Character Input/Output functions in file

getc(), putc(), fgetc(), and fputc() are charcter input output functions. As a practical use of these character input output functions we can read line of text in lowercase and convert it to uppercase, as demonstrated in the following example.

use of getc( ) and putc( ) functions to read a line of text in lowercase and store in data file after converting it to uppercase.

Example#5:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<ctype.h>
void main(){
        char ch;
        FILE *fp;
        fp=fopen("file1.txt","a+");
        if(fp==NULL){
                printf("cannot open file");
                fclose(fp);
                exit(1);
        }
        do{
                ch=toupper(getchar());
                putc(ch,fp);
        }while(ch!='\n');
        clrscr();
        rewind(fp);
        while((ch=getc(fp))!=EOF){
                printf("%c",ch);
        }
        fclose(fp);
        getch();
}
```

Coping contents of one file into another. This program takes the contents of a text file and copies them into another text file character by character

Example#5:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<ctype.h>
void main(){
	char ch;
	FILE *fs,*ft;
	fs=fopen("file1.txt","r");
	if(fs==NULL){
		printf("can not open source file\n");
		exit(1);
	}
	ft=fopen("file2.txt","w");
	if(ft==NULL){
		printf("Cannot open target file");
		fclose(ft);
		exit(1);
	}
	while((ch=fgetc(fs))!=EOF){
		fputc(ch,ft);
	}
	fclose(fs);
	fclose(ft);
	getch();
}
```

In this program file1.txt is a source file which is opened in read mode and file2.txt is a destination file which is opened in write mode.

Using feof() function

The function feof() determines when the end of the file has been encountered. The feof() function has this prototype:

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

feof() returns true(1) if the end of the file has been reached; otherwise, it returns 0.

e.g.

```
	while(!feof(fp))

	ch=getc(fp);
```

The operation of feof() is illustrated in the Example 3.

String input/output function in file

Reading or writing strings of characters from and to files is easy as reading and writing individual characters. fputs() and fgets() are string(line) input/output function in C.

The **fputs()** function writes the string pointed to by string variable to the specified stream. It returns EOF if an error occurs.Since fputs() does not automatically add a new line character to the end if the string; we must do this explicitly to read the string back from the file.

The **fgets()** function reads a string from the specified stream until either a new line character is read or length-1 character have been read. The function fgets() takes three arguments: the first is the address where the string is stored, the second is the maximum length of the string, and the third is the pointer to the structure FILE. When all the lines from the file have been read, if we attempt to read one more time it will return NULL.

Here is a sample program that writes strings to a file using the function fputs() and reads string from the file using fgets() function.

Example#7:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
void main(){
        char string[25];
        FILE *fp;
        fp=fopen("file3.txt","a+");
        if(fp==NULL){
                puts("can not open file");
                exit(1);
        }
        do{
                printf("Enter a string to store in data file:");
                gets(string);
                strcat(string,"\n"); //add new line, since fgets() does not automatically add new line
character to the end of the string
                fputs(string,fp);
        }while(*string!='\n');
        rewind(fp);
        while(!feof(fp)){
                fgets(string,2,fp);
                puts(string);
        }
        fclose(fp);
        getch();
}
```

In this example, while writing to a data file do while loop is used, which allows you to enter any number of strings. To stop entering you have to press enter key at the beginning (i.e. new line character should appear at the beginning). Similarly strings can be read from the data file until the end-of-file is reached.

Using rewind() function

The rewind() function resets the file position indicator to the beginning of the file specified as its arguments. That is, it "rewinds" the file. Its prototype is

```
void rewind(FILE *fp);
```

where, fp is a valid file pointer.

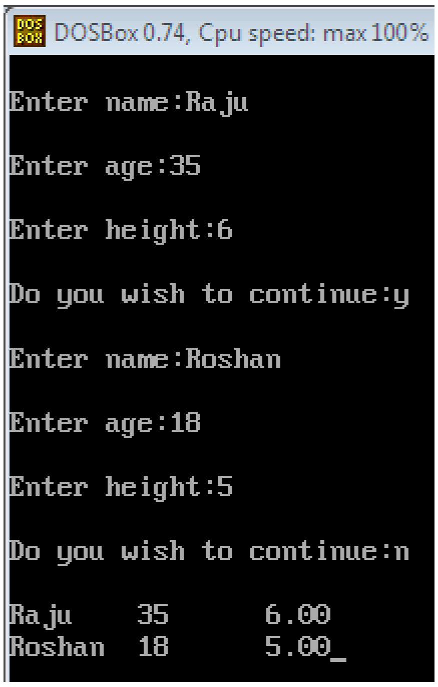
In Example 1 and Example 3 the rewind() function is used after input is complete to move file position indicator at the beginning so that data can be read from the beginning.

Formatted Disk I/O Functions in file

For formatted reading and writing of characters, strings, integers, float, there exist **two** functions: **fscanf()** and **fprintf().** Here is a sample program which illustrates the use of these functions.

Example#8: Program to write information to a file and read from using formatted Input/Output

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main(){
        char choice;
        char name[40];
        int age;
        float height;
        FILE *fp;
        clrscr();
        fp=fopen("myname.txt","w+");
        do{
                printf("\nEnter name:");
                scanf("%s",name);
                fflush(stdin);
                printf("\nEnter age:");
                scanf("%d",&age);
                fflush(stdin);
                printf("\nEnter height:");
                scanf("%f",&height);
                fflush(stdin);
                printf("\nDo you wish to continue:");
                scanf("%c",&choice);
                fprintf(fp,"%s\t%d\t%f",name,age,height);
        }while(choice=='Y'||choice=='y');
        rewind(fp);
        while(!feof(fp)){
                fscanf(fp,"%s%d%f",name,&age,&height);
                printf("\n%s\t%d\t%0.2f",name,age,height);
        }
        fclose(fp);
        getch();
}
```



```
Enter name:Raju

Enter age:35

Enter height:6

Do you wish to continue:y

Enter name:Roshan

Enter age:18

Enter height:5

Do you wish to continue:n

Raju    35      6.00
Roshan  18      5.00_
```

In this program the function **fprintf()** writes the values of three variables to the file. This function is similar to **printf(),** except that a FILE pointer is included as the first argument.The function **fflush()** is designed to remove or flush out any data remaining in the buffer. The argument to **fflush()** must be the buffer which we want to flush out. Here we use 'stdin', which means buffer related with standard input device, the keyboard. The **fscanf()** function is used to read the data from the disk. This function is similar to **scanf(),** except that, as with **fprintf(),** a pointer to FILE is included as the first argument.

Record Input/Output functions in file

In above cases we found that storing number in the format provided by formatted Input/Output functions take up a lot of disk space, because each digit is stored as a character. Similarly, formatted Input/Output has another problem: there is not direct way to read and write complex data types such as arrays and structures. Arrays can be handled but inefficiently by writing each array element one at a time.

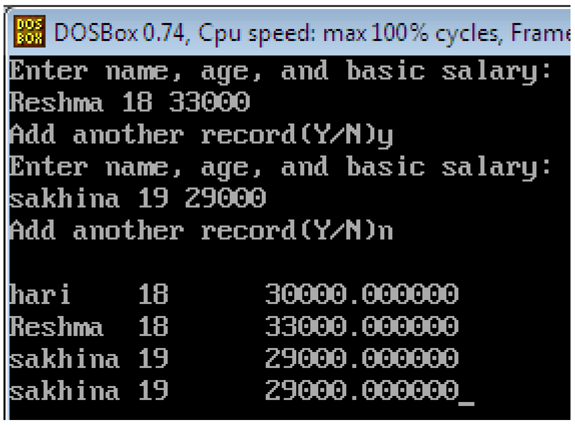
A possible solution for this problem is record input/output, which is sometimes called "block input/output". Record input/output writes number to disk file in binary(or "untranslated") format, so that integers are stored in 4 bytes, floating point number in 4 bytes, and so on for the other numerical types. Record I/O also permits writing any amount of data at once. Array, structures, and other data constructions can be written with a single statement. The functions used for this purpose are fread() and fwrite().

fread() and fwrite()

To read and write data types that are longer than one byte, the C file system provides two functions: **fread()** and **fwrite().** These functions allow the reading and writing of blocks of any type of data (they are used for reading/writing records with binary file). The following program writes and reads data to and from binary file.

Example#9: Program to write and read the data of structure to and from data file

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct employee{
        char name[40];
        int age;
        float basic_sal;
};
void main(){
        struct employee e;
        FILE *fp;
        char choice;
        clrscr();
        fp=fopen("employee.txt","a+b");
        if(fp==NULL){
                printf("Can not open file!");
                exit(0);
        }
        do{
                printf("Enter name, age, and basic salary:\n");
                scanf("%s%d%f",e.name,&e.age,&e.basic_sal);
                fwrite(&e,sizeof(e),1,fp);
                fflush(stdin);
                printf("Add another record(Y/N)");
                scanf("%c",&choice);
        }while(choice=='Y'||choice=='y');
        rewind(fp); //Moves file pointer to the beginning to read from the beginning
        while(!feof(fp)){
                fread(&e,sizeof(e),1,fp);
                printf("\n%s\t%d\t%f",e.name,e.age,e.basic_sal);
        }
        fclose(fp);
        getch();
}
```



```
Enter name, age, and basic salary:
Reshma 18 33000
Add another record(Y/N)y
Enter name, age, and basic salary:
sakhina 19 29000
Add another record(Y/N)n

hari    18      30000.000000
Reshma  18      33000.000000
sakhina 19      29000.000000
sakhina 19      29000.000000_
```

The information obtained about the employee from the keyboard is placed in the structure variable e. Then, the following statement writes the structure to the file.

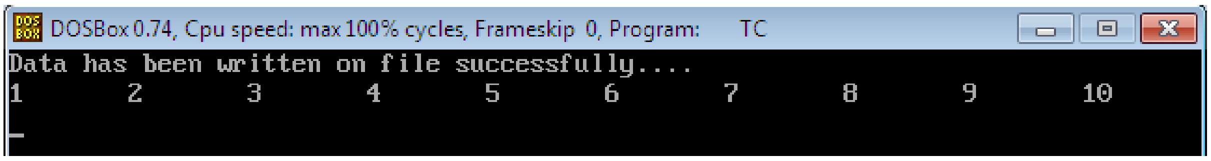
```
fwrite(&e,sizeof(e),1,fp);
```

Here, the first argument is the address of the structure to be written to the disk. The second argument is the size of the structure in bytes. Instead of counting the bytes occupied by the structure ourselves, we let the program do it for us by using sizeof() operator, which gives the size of the variable in bytes. The third argument is the number of such structures that we want to write at one time. In this case, we want to write only one structure at a time. The last argument is the pointer to the file we want to write to.

The fread() function causes the data read from the disk to be placed in the structure variable e. The format of fread() is same as fwrite(). The function fread() returns the number of records read. If we have reached the end of file, since fread() can not read anything, it returns 0.

Example#10: Program on reading and wrting arrays with record I/O functions

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int table[10]={1,2,3,4,5,6,7,8,9,10};
void main(){
        FILE *ftp;
        int i;
        clrscr();
        ftp=fopen("table.txt","wb+");
        if(ftp==NULL){
                printf("Can not open file");
                exit(0);
        }
        else{
                fwrite(table,sizeof(table),1,ftp);
        }
        printf("Data has been written on file successfully....\n");
        rewind(ftp);
        while(!feof(ftp))
        fread(table,sizeof(table),1,ftp);
        for(i=0;i<10;i++){
                printf("%d\t",table[i]);
        }
        fclose(ftp);
        getch();
}
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Data has been written on file successfully....
1       2       3       4       5       6       7       8       9       10
_
```

ftell() function

ftell() returns the current file pointer for stream. The offset is measured in bytes from the beginning of the file (if the file is binary). Its prototype is

```
long int ftell(FILE *stream);
```

The value returned by ftell can be used in a subsequent call to fseek. ftell returns the current file pointer position on success. It returns -1L on error and sets the global variable errno to a positive value. In the event of an error return the global variable errno is set to one of the following values:

EBADF	Bad file pointer
ESPIPE	Illegal seek on device

Example#11: /* ftell example */

```
#include<stdio.h>
void main(){
  FILE *stream;
  clrscr();
  stream = fopen("MYFILE.TXT", "w+");
  fprintf(stream, "This is a test");
  printf("The file pointer is at byte %ld\n", ftell(stream));
  fclose(stream);
  getch();
}
```

fseek() function - Repositions a file pointer on a stream.

fseek() sets the file pointer associated with stream to a new position that is offset bytes from the file location given by origin.

Syntax: *int fseek(FILE *stream, long offset, int origin);*

offset is the number of bytes from the origin that will become the new current position.

For text mode streams offset should be 0 or a value returned by ftell(). Origin must be one of the values 0, 1, or 2 which represent three symbolic constants (defined in stdio.h) as follows:

Constant	Origin	File location
SEEK_SET	0	File beginning
SEEK_CUR	1	Current file pointer position
SEEK_END	2	End-of-file

After fseek the next operation on an update file can be either input or output.

Return Value - fseek returns 0 if the pointer is successfully moved and nonzero on failure.

Example#8: Random File Processing

```
#include<stdio.h>
#include<conio.h>
struct employee{
        char name[10];
        char address[15];
        long int phone;
        char qualification[10];
}emp[5],emp1;
void main(){
        int end_position,current_position,total_no,n,i;
        char choice;
        FILE *fptr;
        clrscr();
        fptr=fopen("random.txt","a+");
        for(i=0;i<5;i++){
                printf("Enter the name:");
                scanf("%s",emp[i].name);
                printf("Enter the address:");
                scanf("%s",emp[i].address);
                printf("Enter Phone No:");
                scanf("%ld",&emp[i].phone);
                printf("Enter Qualification:");
                scanf("%s",emp[i].qualification);
                fwrite(&emp[i],sizeof(struct employee),1,fptr);
                clrscr();
        }
        rewind(fptr);
        fseek(fptr,0,SEEK_END);
        end_position=ftell(fptr);
        total_no=end_position/sizeof(struct employee);
        do{
                printf("There are %d structures,which u want to view?",total_no);
                scanf("%d",&n);
                current_position=(n-1)*sizeof(struct employee);
                fseek(fptr,current_position,SEEK_SET);
                fread(&emp1,sizeof(struct employee),1,fptr);
                printf("\n The name is %s\n",emp1.name);
                printf("The address is %s\n",emp1.address);
                printf("The phone no is %ld\n",emp1.phone);
                printf("The qualification is %s\n",emp1.qualification);
                fflush(stdin);
                printf("Do u want to see some other records(Y/N):");
                scanf("%c",&choice);
                clrscr();
        }while(choice=='Y'||choice=='y');
        fclose(fptr);
        getch();
}
```