I B.Tech CSE - PPSC

FILE HANDLING IN C

****

FILE HANDLING: Input and output – concept of a file, text files and binary files, Formatted I/o, file i/o operations, example programs.

****

Until now we have been using the functions such as scanf(), printf() to read and write data. These are console oriented I/O functions, which always use keyboard and screen as the target place. This works fine as long as the data is small. However, many real life problems involve large volumes of data and in such situations; the console oriented I/O operations pose two major problems.

1. It becomes cumbersome and time consuming to handle large volumes of data through terminals.

2. The entire data is lost when either the program is terminated or the computer is turned off.

It is therefore necessary to have a more flexible approach where data can be stored on the disks and read whenever necessary, without destroying the data. This method employs the concept of files to store data.

A file is a place on the disk where a group of related data is stored. Like more other languages, C supports a number of functions that have the ability to perform basic file operations, which include

- **Naming a file**
- **Opening a file**
- **Reading data from a file**
- **Writing data to a file**
- **Closing a file**

Defining and opening a file:

If we want to store data in a file in the secondary memory, we must specify certain things about the file, to the operating system. They include

1. File name
2. Data structure
3. Purpose

When we open a file , we must specify what we want to do with the file. For example, we may write data to the file or read the already existing data.
General format for declaring and opening a file:

```
FILE *fp;
fp=fopen("filename","mode");
```

The first statement declares the variable fp as a pointer to the data type FILE. FILE is a structure that is defined in the I/O library. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. This contains all the information about the file and is subsequently used as a communication link between the system and the program.

Mode can be one of the following.

r -- open the file for reading only.
w -- open the file for writing only
a -- open the file for appending data to it

Note that both filename and mode are specified as strings. They should be enclosed in double quotation marks.

```
FILE *fp1,*fp2;
fp1=fopen("data.dat","w");
fp2=fopen("stud.dat","r");
```

The file data.dat is opened for writing and
the file stud.dat is opened for reading.

MODE	OPERATION POSSIBLE
r+	Reading existing contents ,writing new contents, modifying existing contents of the file
w+	Writing new contents, reading them back and modifying existing contents of the file
a+	Reading existing contents, appending new contents to end of file. Cannot modify existing contents.

Closing a file: A file must be closed as soon as all operations on it have been completed. This ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.

fclose(file-pointer);

Input/Output operations on files:

The simplest file I/O functions are
getc() and
putc().

Assume that the file is opened in write mode and the file pointer fp1. The statement

putc(c,fp1);

Writes the character contained in the character variable c to the file associated with FILE pointer fp1.

Similarly getc() is used to read a character from a file that has been opened in read mode. For example

```
c=getc(fp2);
```

Would read a character from the file whose file pointer is fp2.

The getw() and putw() functions

The getw() and putw() are integer oriented functions. They are similar to getc and putc functions and are used to read and write integer values. These functions are used when we deal with only integer data.

```
putw(integer,fp);
ivar=getw(fp);
```

The fprintf() and fscanf() functions:

So far, we have seen functions which can handle only one character on integer at a time. fprintf() and fscanf() can handle a group of mixed data simultaneously.

The functions fprintf() and fscanf() perform I/O operations that are identical to the familiar printf() and scanf() functions, except of course that they work on files. The first argument of these functions is a file pointer which specifies the file to be used.

General format of fprintf() is:

```
fprintf(fp,"control-string",var-list);
```

Where fp is the file pointer associated with a file that has been opened for writing. The control string contains output specifications for the items in the list. The list may include variables, constants and strings.

The general format of fscanf() is

```
fscanf(fp,"control-string",var-list);
```

where fp is the file pointer associated with a file that has been opened for reading. This statement would cause the reading of the items in the list from the file specified by fp, according to the specifications contained in the control string.

feof() function can be used to test for an end of file condition. It takes a file pointer as its only argument and returns a non zero integer value if all of the data from the specified file has been read, and returns zero otherwise.

```
if (fp==null)
    printf("file could not be opened\n")
```

```
/*-----------------------------------------------------------------------------------------------------------
        program to create a data file containing student records
-----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
   FILE *fp;
   int stno;
   char stname[10];
   int m1,m2,m3,n,i;

   clrscr();
   fp=fopen("stu.dat","w");
   printf("enter  n:");
   scanf("%d",&n);

   for(i=1;i<=n;i++)
   {
       printf("stno:");
       scanf("%d",&stno);
       printf("stname:");
       scanf("%s",stname);
       printf("m1 m2 m3:");
       scanf("%d%d%d",&m1,&m2,&m3);
       fprintf(fp,"%d %s %d %d %d\n",stno,stname,m1,m2,m3);
   }
   fclose(fp);
   getch();
 }
```

OUTPUT:
enter n: 4
stno: 501
stname: rama
m1 m2 m3 : 50 60 70
stno: 502
stname: raja
m1 m2 m3 : 70 80 90
stno: 503
stname: anuradha
m1 m2 m3 : 40 30 70
Stno: 504
Stname: rajani
m1 m2 m3 : 70 80 50

stu.dat

501	rama	50 60 70
502	raja	70 80 90
503	anuradha	40 30 70
504	rajani	70 80 50

```
/*-------------------------------------------------------------------------------
        program to read a data file and print the contents
-------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
 {
        FILE *fp;
        int stno;
        char stname[10];
        int m1,m2,m3;

         clrscr();
         fp=fopen("stu.dat","r");
         fscanf(fp,"%d%s%d%d%d",&stno,stname,&m1,&m2,&m3);

          printf("The contents of data file are\n");
          while (!feof(fp))
          {
            printf("%d %s %d %d %d\n",stno,stname,m1,m2,m3);
            fscanf(fp,"%d%s%d%d%d",&stno,stname,&m1,&m2,&m3);
           }

         fclose(fp);
        getch();
 }
```

stu.dat

```
501 rama      50 60 70
502 raja      70 80 90
503 anuradha 40 30 70
504 rajani    70 80 50
```

```
/*-------------------------------------------------------------------------------------------------
        program to read a data file   "stu.dat" process the records and
        store the processed records in another data file  -- "stu.out"
-------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
 {
        FILE *fp1,*fp2;
        int  stno;
        char stname[10];
        int  m1,m2,m3,tot;
        float avg;
        char result[15];

         clrscr();

         fp1=fopen("stu.dat","r");
         fp2=fopen("stu.out","w");

         fscanf(fp1,"%d%s%d%d%d",&stno,stname,&m1,&m2,&m3);
          while (!feof(fp1))
          {
            tot=m1+m2+m3;
            avg=(float)tot/3.0;
                  if((m1<35)||(m2<35)||(m3<35))
                     strcpy(result,"Fail");
                  else
                      strcpy(result,"Pass");
            fprintf(fp2,"%d %s %d %d %d %d %f %s\n", stno,stname,m1,m2,m3,tot,avg,result);
            fscanf(fp1,"%d%s%d%d%d",&stno,stname,&m1,&m2,&m3);
           }

          fclose(fp1);
          fclose(fp2);
          getch();
}
```

Input file

```
stu.dat
501 rama      50 60 70
502 raja      70 80 90
503 anuradha 40 30 70
504 rajani    70 80 50
```

Output File

```
stu.out
501 rama      50 60 70  180  60.00   pass
502 raja      70 80 90  240  80.00   pass
503 anuradha 40 30 70  140  46.66   pass
504 rajani    70 80 50  200  66.66   fail
```

```
/*-----------------------------------------------------------------------------------------------------------
                    Program to create a data file containing only integers
-----------------------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <conio.h>
 void main()
 {
      FILE *fp;
      int i,num;

      clrscr();

      fp=fopen("int.dat","w");
      for (i=1;i<10;i++)
     {
           printf("enter an integer ");
           scanf("%d",&num);
           putw(num,fp);
     }
     fclose(fp);
     printf("integer data file created\n");

     printf("opening the file to read\n");
     fp=fopen("int.dat","r");

     printf("file contents are \n");
     num=getw(fp);

     while(!feof(fp))
     {
          printf("%d\n",num);
          num=getw(fp);
     }

     fclose(fp);
     getch();
 }
```

RANDOM ACCESS TO FILES:

- **ftell**
- **rewind**
- **fseek**

ftell() takes a file pointer and returns a number of type long, that corresponds to the current position. This function is useful in saving the current position of a file.

```
n=ftell(fp);
```

n would give the relative offset of the current position. This means that 'n' bytes have already been read.

rewind() takes file pointer and resets the position to the start of the file.

```
rewind(fp);
n=ftell(fp);
```

This would assign 0 to n because the file position has been set to the statement of the file by rewind.

fseek is used to move the file position to a desired location with in the file.

```
fseek(fileptr, offset,  position);
```

Offset is a number specifies the number of positions to be moved from the location specified by position.

Position can take one of the following three values.

Value	Meaning
0	Beginning of the file
1	Current position of the file
2	End of the file

The offset may be positive, meaning move forwards, or negative, meaning move backwards.

```
fseek(fp, 0L, 0)      go to the beginning
fseek(fp, 0L, 1)      stay at current position
fseek(fp, 0L, 2)      end of file.
fseek(fp, m, 0)       move to (m+1)th byte from begin
fseek(fp, m, 1)       move m bytes forward from current position
fseek(fp, -m, 1)      move m bytes backward from current position
fseek(fp, -m, 2)      move m bytes backward from the end
```

fseak(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 back by two positions to read the next character.

```
fseek(fp, -2l, 1);
```

```
/*-------------------------------------------------------------------------
      Random access - Program using fseak() and ftell() functions
-------------------------------------------------------------------------*/
#include <stdio.h>
#include <conio.h>
 void main()
 {
     FILE *fp;
     long n;
     char c;

     clrscr();
     fp=fopen("random.dat","w");
     printf("enter text terminated by ctrl+Z\n");
     while((c=getchar())!=EOF)
              putc(c,fp);

     printf("no of characters enterd =%d\n",ftell(fp));
     fclose(fp);

     fp=fopen("random.dat","r");
     n=0L;

     while(feof(fp)==0)
     {
        fseek(fp,n,0);
        printf("position of %c is %ld\n",getc(fp),ftell(fp));
        n=n+5L;
     }

     printf("printing the file in reverse direction\n");
     fseek(fp,-1L,2);
     do
     {
          putchar(getc(fp));
      }while(!fseek(fp,-2L,1));

     fclose(fp);
     getch();
 }
```

OUTPUT

```
Enter text terminated by ctrl+z
abcdefghijklmnopqrstuvwxyz<ctrl+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
Printing the file in reverse
 zyxwvutsrqponmlkjihgfedcba
```

COMMAND LINE ARGUMENTS:

argc -- argument counter
argv -- argument vector

argc counts the number of arguments on the command line. Argv is an argument vector and represents an array of character pointers that points to the command line arguments. In order to access the command line arguments, we must declare the main function and its parameters as follows.

```
void main(argc, argv)
int argc;
char *argv[];
{
      --
      --
      --
}
```

we can pass argument to a program i.e. main() using command line arguments. In fact main can take two arguments called argc and argv.

Ex:

```
C> program xfile   yfile↵
```

argv[0] → program
argv[1] → xfile
argv[2] → yfile
argc=3

```
/*----------------------------------pro1.c-----------------------------------------
        program to read data from command line and print the data
-------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main(argc,argv)
int argc;
char *argv[];
{
        int i;

        clrscr();

        printf("total no of arguments %d\n",argc);
        printf("the program name %s\n",argv[0]);

        for(i=1;i<argc;i++)
            printf("%s\n",argv[i]);

        getch();
}
```

OUTPUT

```
c>pro1 rama raja ravi
total no of arguments 4
the program name :pro1
rama
raja
ravi
```

```
/*----------------------------------pro.c-----------------------------------------------------------
Program to read data from command line and store the data in a file
      c>pro data aa bb cc dd ee
                  where pro is the name of the program
                        data is the file to be created
                        aa bb cc dd ee  is text to be stored in the file
-----------------------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<conio.h>
void main(argc,argv)
int argc;
char *argv[];
 {
      FILE *fp;
      int i;

      clrscr();

      printf("total no of arguments =%d\n",argc);
      printf("the program name =%s\n",argv[0]);
      printf("the data file name= %s\n",argv[1]);

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

      for(i=2;i<argc;i++)
      {
            fprintf(fp,"%s\n",argv[i]);
            printf("%s\n",argv[i]);
      }

      fclose(fp);

      getch();
 }
```

OUTPUT

```
c>pro data aa bb cc dd ee
Total number of arguments = 7
The program name = pro
The data file name = data
aa
bb
cc
dd
ee
```