K. V. N. Naik Shikshan Prasarak Sanstha's
Arts, Commerce & Science College, Nashik
Department of Computer Science

Class:-F.Y.B.Sc.(Comp.Sci.) Sem:II

Subject: Advance 'C' Programming

Chapter 4:- File Handling in C

Why files are needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all.
  However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of Files

When dealing with files, there are two types of files you should know about:

- Text files
- Binary files

1. Text files

- Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad.
- 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 the 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 a higher amount of data, are not readable easily, and provides better security than text files.
- Text file is human readable because everything is stored in terms of text. In binary file everything is written in terms of 0 and 1, therefore binary file is not human readable.

File Operations

In C, you can perform four major operations on files, either text or binary:

- Naming a file
- Creating a new file
- Opening an existing file

- Closing a file
- Reading from a file
- Writing information to a file

Working with files

- When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.

Syntax for declaring file pointer

```
FILE * fp;
```

- A pointer to FILE structure contains information, such as size, current file pointer position, type of file etc., to perform operation on the file.

Opening a file using fopen() function

- The fopen() function takes two arguments, the name of the file amd the mode in which the file is to be opened. Mode specify the purpose of opening the file i.e, whether for reading or writing.

Syntax for opening the file in C

```
fp = fopen(char *filename,char *mode);
```

When fopen() function opens a file in memory, it returns a pointer to this particular file. If fopen() function can't open the file then it will return NULL.

Example for opening the file in C

```
void main()
{
    FILE *fp;
    fp = fopen("file1.txt","r");
    if(fp == NULL)
    {
        printf("\nCan't open file or file doesn't exist.");
        exit(0);
    }
```

Mode	Meaning of Mode	Explain
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. Data is added to the end of the file.	If the file does not exist, it will be created.

ab	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, 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 exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Closing a File using fclose() function

When the reading or writing of a file is finished, the file should be closed properly using fclose() function. The fclose() function does the following tasks:

- Flushes any unwritten data from memory.
- Discards any unread buffered input.
- Frees any automatically allocated buffer
- Finally, close the file.

Syntax for closing the file in C

```
int fclose( FILE* );
```

Example for closing the file in C

```
void main()
{
    FILE *fp;

    fp = fopen("file1.txt","r");
    if(fp == NULL)
    {
        printf("\nCan't open file or file doesn't exist.");
        exit(0);
    }
    fclose(fp);

}
```

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

The fwrite() function

The fwrite() function is used to write records (sequence of bytes) to the file. A record may be an array or a structure.

Syntax of fwrite() function

```
fwrite( ptr, int size, int n, FILE *fp );
```

- **The fwrite() function takes four arguments.**
- **ptr : ptr is the reference of an array or a structure stored in memory.**
- **size : size is the total number of bytes to be written.**
- **n : n is number of times a record will be written.**
- **FILE* : FILE* is a file where the records will be written in binary mode.**

Example of fwrite() function

/*C program to write all the members of an array of structures to a file using fwrite(). */

```
#include<stdio.h>
#include<stdlib.h>
    struct Student
    {
        int roll;
        char name[25];
        float marks;
    };
    void main()
    {
        FILE *fp;
        int i,n;
     struct Student S[50];

     fp = fopen("stud.txt","w");
     printf("\nEnter how many records:");
     scanf("%d",&n);
     for(i = 0; i < n; i++)
    {
          printf("\nEnter Roll : ");
           scanf("%d",&S[i].roll);

           printf("Enter Name : ");
           scanf("%s",S[i].name);

           printf("Enter Marks : ");
           scanf("%f",&S[i].marks);
```

```
        }
            fwrite(&S,sizeof(S),1,fp);
        printf("\nData written successfully...");
        fclose(fp);
    }
```

The fread() function

The fread() function is used to read bytes form the file.

Syntax of fread() function

```
fread( ptr, int size, int n, FILE *fp );
```

- **The fread() function takes four arguments.**
- **ptr : ptr is the reference of an array or a structure where data will be stored after reading.**
- **size : size is the total number of bytes to be read from file.**
- **n : n is number of times a record will be read.**
- **FILE* : FILE* is a file where the records will be read.**

Example of fread() function

```
#include<stdio.h>
#include<stdlib.h>
    struct Student
    {
        int roll;
        char name[25];
        float marks;
    };
    void main()
    {
     FILE *fp;
     int i,n;
     struct Student S[50];
     fp = fopen("stud.txt","r");
     printf("\nEnter how many records:");
     scanf("%d",&n);
     printf("\n\tRoll\tName\tMarks\n");
     for(i = 0; i < n; i++)
    {
        fread(&S,sizeof(S),1,fp);
```

```
            printf("\n\t%d\t%s\t%f",S[i].roll,S[i].name,S[i].marks);
        }
         fclose(fp);
    }
```

Reading writing integer using putw() and getw() functions

The putw() function

The putw() function is used to write integers to the file.

Syntax of putw() function

```
putw(int number, FILE *fp);
```

The putw() function takes two arguments, first is an integer value to be written to the file and second is the file pointer where the number will be written.

```
#include<stdio.h>
 void main()
 {
        FILE *fp;
        int i,n;
        fp = fopen("file.txt","w");
        printf("\nEnter the integer data:\n");
         scanf("%d",&n);
         while(n!=0)
         {
           putw(n,fp);
           scanf("%d",&n);
         }
        printf("\nData written successfully...\n");
        fclose(fp);
    }
```

The getw() function

The getw() function is used to read integer value form the file.

Syntax of getw() function

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

The getw() function takes the file pointer as argument from where the integer value will be read and returns returns the *end-of-file* if it has reached the end of file.

Example of getw() function

```
#include<stdio.h>
```

```
void main()
{
        FILE *fp;
        int n;
        fp = fopen("file.txt","r");
        printf("\nReading data from file..\n");
         while((n=getw(fp))!=EOF)
         {
              printf("%d\n",n);
           }
        fclose(fp);
 }
```

Reading Writting characters using fgetc() and fputc() function

The fputc() function

The fputc() function is used to write characters to the file.

Syntax of fputc() function

```
fputc(char ch, FILE *fp);
```

The fputc() function takes two arguments, first is the character to be written to the file and second is the file pointer where the character will be written.

Example of fputc() function

```
#include<stdio.h>
    void main()
    {
        FILE *fp;
        char ch;
        fp = fopen("a.txt","w");
        while((ch=getchar())!=EOF)
           fputc(ch,fp);

        printf("\nData written successfully...");

        fclose(fp);
    }
```

The fgetc() function

The fgetc() function is used to read characters form the file.

Syntax of fgetc() function

```
char fgetc(FILE *fp);
```

The fgetc() function takes the file pointer indicates the file to read from and returns the character read from the file or returns the *end-of-file* character if it has reached the end of file.

Example of fgetc() function

```
#include<stdio.h>
void main()
  {
      FILE *fp;
      char ch;
      fp = fopen("a.txt","r");
      printf("\nData in file...\n");
      while((ch = fgetc(fp))!=EOF)
         printf("%c",ch);
      fclose(fp);
  }
```

Reading writing string using fputs() and fgets() functions

The fputs() function

The fputs() function is used to write string(array of characters) to the file.

Syntax of fputs() function

```
fputs(char str[], FILE *fp);
```

The fputs() function takes two arguments, first is the string to be written to the file and second is the file pointer where the string will be written.

Example of fputs() function

```
#include<stdio.h>
  void main()
{
    FILE *fp;
    char str[50];
    fp = fopen("string.txt","w");
    printf("\nEnter any string:");
      gets(str);
      fputs(str,fp);
    printf("\nData written successfully...");
    fclose(fp);
}
```

The fgets() function

The fgets() function is used to read string(array of characters) from the file.

Syntax of fgets() function

```
fgets(char str[],int n,FILE *fp);
```

The fgets() function takes three arguments, first is the string read from the file, second is size of string(character array) and third is the file pointer from where the string will be read.

The fgets() function will return NULL value when it reads EOF(end-of-file).

Example of fgets() function

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char str[80];

    fp = fopen("file.txt","r");
    if(fp == NULL)
    {
        printf("\nCan't open file or file doesn't exist.");
        exit(0);
    }
    printf("\nData in file...\n");

    while((fgets(str,80,fp))!=NULL)
        printf("%s",str);

    fclose(fp);
}
```

The fprintf() fscanf() Functions in C

The fprintf() function

So far we have seen writing of characters, strings and integers in different files. This is not enough if we need to write characters, strings and integers in one single file, for that purpose we use fprintf() function. The fprintf() function is used to write mixed type in the file.

Syntax of fprintf() function

```
fprintf(FILE *fp,"format-string",var-list);
```

The fprintf() function is similar to printf() function except the first argument which is a file pointer that specifies the file to be written.

Example of fprintf() function

```
#include<stdio.h>
```

```
void main()
{
      FILE *fp;
      int roll;
      char name[25];
      float marks;
      char ch;
      fp = fopen("print.txt","w");
            printf("\nEnter Roll : ");
            scanf("%d",&roll);
            printf("\nEnter Name : ");
            scanf("%s",name);
            printf("\nEnter Marks : ");
            scanf("%f",&marks);
            fprintf(fp,"%d%s%f",roll,name,marks);
         printf("\nData written successfully...");
         fclose(fp);
    }
```

The fscanf() function

The fscanf() function is used to read mixed type form the file.

Syntax of fscanf() function

```
fscanf(FILE *fp,"format-string",var-list);
```

The fscanf() function is similar to scanf() function except the first argument which is a file pointer that specifies the file to be read.

Example of fscanf() function

```
#include<stdio.h>
      void main()
      {
            FILE *fp;
            int roll;
            char name[25];
            float marks;
           fp = fopen("print.txt","r");
           printf("\nData in file...\n");
```

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

File Positioning Functions/Random Access to Files

File positioning functions are used to move the pointer in a file to the desired position, without closing and re-opning the file.

C provides followling three functions to move the pointer in the file:

- **fseek()**
- **ftell()**
- **rewind()**

The fseek() function in C

The fseek() function is used to move the cursor in the file to the desired position.

Syntax of fseek() function

```
int fseek( FILE *fp, long offset, int pos );
```

The fseek() function takes three arguments, first is the file pointer, second is the offset that specifies the number of bytes to moved and third is the position from where the offset will move. It will return zero if successfully move to the specified position otherwise return nonzero value.

There are three positions from where offset can move.

Constant Name	Constant Value	Description
SEEK_SET	0	The begining of file
SEEK_CUR	1	The current position in file
SEEK_END	2	The end of file

```
#include<stdio.h>
    struct Student
    {
        int roll;
        char name[25];
        float marks;
    };
```

```
void main()
{
      FILE *fp;
      char ch;
    long offset;
      struct Student Stu;
      fp = fopen("stud.txt","r");

    printf("\n\nLast record in file.\n");
      offset = sizeof(Stu)*-1;
      fseek(fp,offset,SEEK_END);
      fread(&Stu,sizeof(Stu),1,fp);
      printf("\n\tRoll : %d",Stu.roll);
      printf("\n\tName : %s",Stu.name);
      printf("\n\tMarks : %f",Stu.marks);
      printf("\n\nFirst record in file.\n");
      fseek(fp,0,SEEK_SET);
      fread(&Stu,sizeof(Stu),1,fp);
      printf("\n\tRoll : %d",Stu.roll);
      printf("\n\tName : %s",Stu.name);
      printf("\n\tMarks : %f",Stu.marks);

      printf("\n\nSecond record in file.\n");
          fseek(fp,0,SEEK_CUR);
          fread(&Stu,sizeof(Stu),1,fp);
          printf("\n\tRoll : %d",Stu.roll);
          printf("\n\tName : %s",Stu.name);
          printf("\n\tMarks : %f",Stu.marks);

          fclose(fp);
    }
```

The ftell() function

The ftell() function returns the current position of cursor in the file.

Syntax of ftell() function

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

The ftell() function takes the file pointer as argument and return the current position in long type. Bacaues the file may have more then 32767 bytes of data.

Example of ftell() function

```
#include<stdio.h>
    void main()
    {
        FILE *fp;
        char ch;
        long pos;
        fp = fopen("string.txt","r");
       printf("\nData in file...\n");
        while((ch = fgetc(fp))!=EOF)
        {
            if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
            {
                pos=ftell(fp);
                printf("\nPosition of %c is at %ld",ch,pos);
            }
        }
        fclose(fp);
}
```

The rewind() function

The rewind() function is used to move the cursor at the begining of the file. we can also use fseek() function to move the cursor at the begining of the file.

Syntax of rewind() function

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

The rewind() function takes the file pointer as argument.

Example of rewind() function

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

```
FILE *fp;
char ch;
fp = fopen("rewind.txt","a+");
printf("\nWrite some data :\n");
while((ch=getchar())!=EOF
   fputc(ch,fp);
   rewind(fp);
printf("\nData in file :\n");
while((ch = fgetc(fp))!=EOF)
   printf("%c",ch);
fclose(fp);
    }
```

```
#include<stdio.h>
void main()
    {
        FILE *fp;
        int roll;
        char name[25];
        float marks;
        char ch;
        fp = fopen("fprintf.txt","w+");

printf("\nEnter Roll : ");
scanf("%d",&roll);

printf("\nEnter Name : ");
scanf("%s",name);

printf("\nEnter Marks : ");
scanf("%f",&marks);
```

```
            fprintf(fp,"%d%s%f",roll,name,marks);

        printf("\nData written successfully...");

rewind(fp);

        printf("\nData in file...\n");

        while((fscanf(fp,"%d%s%f",&roll,name,&marks))!=EOF)
           printf("\n%d\t%s\t%f",roll,name,marks);

        fclose(fp);
    }

#include<stdio.h>

    void main()
    {
        FILE *fp;
        char str[50];

        fp = fopen("fputs.txt","w+");
        printf("\nEnter any string:");
            gets(str);
            fputs(str,fp);
        printf("\nData written successfully...");
rewind(fp);



        printf("\nData in file...\n");

        while((fgets(str,80,fp))!=NULL)
```

```
            printf("%s",str);

        fclose(fp);
    }

#include<stdio.h>
 void main()
 {
        FILE *fp;
        int i,n;
        fp = fopen("putw.txt","w+");
  printf("\nEnter the integer data:\n");
   scanf("%d",&n);
   while(n!=0)
   {
     putw(n,fp);
     scanf("%d",&n);
   }
  printf("\nData written successfully...\n");
rewind(fp);
  printf("\nReading data from file..\n");
   while((n=getw(fp))!=EOF)
   {
            printf("%d\n",n);

     }
        fclose(fp);
}
```