File Handling in C Programming

Prof V.V.Subrahmanyam
Director, SOCIS

Date: 9th May, 2020 Time : 12 Noon

Introduction

- It is necessary to keep data in the permanent storage because it is difficult to handle the large volume of data by programs and after execution of program is over all the entered data will be lost because, the data stored in the variables are temporary.
- C supports the concept of file through which the data can be stored in the disk or secondary storage device.

File

- A file is a collection of data or text placed on the disk.
- A sequential stream of bytes ending with an *end-of-file* marker is what is called a ***file***.
- When the file is opened the stream is associated with the file.
- By default, three files and their streams are automatically opened when program execution begins –
  - The ***standard input***
  - ***The standard output***
  - ***The standard error***.
- Streams provide communication channels between files and programs.

Text Files and Binary Files

- C supports two kinds of files in which data can be stored in 2 ways either in characters coded in their ASCII character set or in binary format. They are
  - Text Files
  - Binary Files

Contd…

- In *text files*, everything is stored in terms of text
- *i.e.* even if we store an integer 54; it will be stored as a 3-byte string - “54\0”.
- In a text file certain character translations may occur. For example a *newline(\n)* character may be converted to a carriage return, linefeed pair. This is what Turbo C does. Therefore, there may not be one to one relationship between the characters that are read or written and those in the external device.
- A *binary file* contains data that was written in the same format used to store internally in main memory.

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: FILE *fptr;

File Operations

- Creation of a new file (**fopen**)
- Opening an existing file (**fopen**)
- Reading from file (**fscanf or fgets**)
- Writing to a file (**fprintf or fputs**)
- Moving to a specific location in a file (**fseek, rewind**)
- Closing a file (**fclose**)

Opening a File for Creation or Edit

- Opening a file is performed using the fopen() function defined in the stdio.h header file.

  The syntax for opening a file in standard I/O is:

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

  Example:

  fopen(“D:\\cprogram\\newprogram.txt","w");

File Opening Modes

- "**r**" – Searches file. If the file is opened successfully fopen() loads it into memory and sets up a pointer which points to the first character in it. If the file cannot be opened fopen() returns NULL.
- "**w**" – Searches file. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.
- "**a**" – Searches file. If the file is opened successfully fopen() loads it into memory and sets up a pointer that points to the last character in it. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.

- “**r+**” – Searches file. If is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the first character in it. Returns NULL, if unable to open the file.
- “**w+**” – Searches file. If the file exists, its contents are overwritten. If the file doesn’t exist a new file is created. Returns NULL, if unable to open file.
- “**a+**” – Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the last character in it. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open file.

Binary Files

- A binary file is a file that uses all 8 bits of a byte for storing the information .It is the form which can be interpreted and understood by the computer.

1.wb(write)

this opens a binary file in write mode.
SYNTAX:
fp=fopen(“data.dat”,”wb”);

2.rb(read)

this opens a binary file in read mode
SYNTAX:
fp=fopen(“data.dat”,”rb”);

3.ab(append)

this opens a binary file in a Append mode i.e. data can be added at the end of file.

SYNTAX:

fp=fopen(“data.dat”,”ab”);

4.r+b(read+write)

this mode opens preexisting File in read and write mode.

SYNTAX:

fp=fopen(“data.dat”,”r+b”);

5.w+b(write+read)

Example

```
FILE *fptr;
fptr = fopen(“abc.txt”, “w”);
```

File Close (fclose())

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

  fclose(fptr);

  Here, fptr is a file pointer associated with the file to be closed.

Contd…

- This function flushes any unwritten data for stream, discards any unread buffered input, frees any automatically allocated buffer, and then closes the stream.
- The return value is 0 if the file is closed successfully or a constant *EOF*, an end-of file marker, if an error occurred.

```
# include <stdio.h>
main ( )
{
 FILE *fp;
 if ((fp=fopen("file1.dat", "r"))==NULL)
       {
      printf("FILE DOES NOT EXIST\n");
      exit(0);
       }
else {
.....
.....     }
/* close the file */
fclose(fp);
}
```

INPUT OUTPUT USING FILE POINTERS

- Character input/output functions
- String input/output functions
- Formatted input/output functions
- Block input/output functions.

Character Input and Output in Files

- getc()
- putc()

getc( ) is used to read a character from a file and
putc( ) is used to write a character to a file.
Their syntax is as follows:

*int putc(int ch, FILE *stream);*
*int getc(FILE *stream);*

```
/*Program to copy one file to another */

 #include <stdio.h>
main( )
{
   FILE *fp1, *fp2;
   int ch;
   if((fp1=fopen("f1.dat","r")) == NULL)
          {
    printf("Error opening input file\n");
     exit(0);     }
       if((fp2=fopen("f2.dat","w")) == NULL)
   {
    printf("Error opening output file\n");
          exit(0);
        }
while (! eof(fp1))
        {
          ch=getc(fp1);
          putc(ch,fp2);
        }
        fclose(fp1);
        fclose(fp2);
}
```

String Input and Output Functions

- If we want to read a whole line in the file then each time we will need to call character input function, instead C provides some string input/output functions with the help of which we can read/write a set of characters at one time.
- *fgets( )*
- *fputs( )*

Syntax for fputs() and fgets()

These functions are used to read and write strings. Their syntax is:

*int fputs(char *str, FILE *stream);*

*char *fgets(char *str, int num, FILE *stream);*

```
/*Program to read a file and count the
    number of lines in the file */
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
   FILE *fp;
   int cnt=0;
   char str[80];
 /* open a file in read mode */
   if ((fp=fopen("lines.dat","r"))==NULL)
  {    printf("File does not exist\n");
       exit(0);
  }
/* read the file till end of file is encountered
    */
  while(!(feof(fp)))
  {  fgets(str,80,fp);      /*reads at most 80
                             characters in str */
      cnt++;                /* increment the
                               counter after reading
                                        a line */
  }
}/* print the number of lines */
printf("The number of lines in the file is
     :%d\n",cnt);
fclose(fp);
}
```

Formatted Input and Output Functions

- If the file contains data in the form of digits, real numbers, characters and strings, then character /string I/O functions are not enough as the values would be read in the form of characters.
- Hence C provides a set of formatted input/output functions.
  - fprintf()
  - fscanf()

Syntax

- The syntax for these functions is:

*int fscanf(FILE *fp, char *format,. . .);*

*int fprintf(FILE *fp, char *format,. . .);*

Both these functions return an integer indicating the number of bytes actually read or written.

```
#include<stdio.h>
main()
{
  int account;
  char name[30];
  double bal;
  FILE *fp;
 if((fp=fopen("bank.dat","r"))== NULL)
        printf("FILE not present \n");
     else
        do{
          fscanf(fp,"%d%s%lf",&account,name,&bal);
           if(!feof(fp)) {
               if(bal==0)
               printf("%d %s %lf\n",account,name,bal);
                 }
       }while(!feof(fp));
}
```

- This program opens a file "***bank.dat***" in the read mode if it exists, reads the records and prints the information (account number, name and balance) of the zero balance records.

Block Input and Output Functions

- Block Input / Output functions read/write a block (specific number of bytes from/to a file.
- A block can be a record, a set of records or an array.
- These functions are also defined in standard library and are described below.

  - *fread( )*
  - *fwrite( )*

Syntax

*int fread(void *buf, int num_bytes, int count, FILE *fp);*
*int fwrite(void *buf, int num_bytes, int count, FILE *fp);*

- In case of *fread()*, buf is the pointer to a memory area that receives the data from the file
- In *fwrite()*, it is the pointer to the information to be written to the file.
- *num_bytes* specifies the number of bytes to be read or written.
- These functions are quite helpful in case of binary files. Generally these functions are used to read or write array of records from or to a file.

Types of File Accesss

- We can also differentiate in terms of the type of file access as
  - Sequential access files
  - Random access files.
- Sequential access files allow reading the data from the file in sequential manner which means that data can only be read in sequence.

Positioning the File Pointer

- The function *fseek( )* is used to set the file position.

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

- The first argument is the pointer to a file.
- The second argument is the number of bytes to move the file pointer, counting from zero. (can be positive or negative depeding on the desired movement).
- The third parameter is a flag indicating from where in the file to compute the offset. It can have three values (0,1,2).

Contd…

- SEEK_SET(or value 0) the beginning of the file
- SEEK_CUR(or value 1) the current position
- SEEK_END(or value 2) the end of the file