Unit - 3

File Handling

File Management

A computer file is a resource that stores digital data or information on a computer device such as hard disk, magnetic tape or any external electronic storage device. A file is identified by a file name.

For ex. A file could be any of the types such as Text, Audio, Video or any other format.
• For example: A source code written in C program is stored in a file with **.c** extension, any other text information may be stored with .doc or .txt extension, any image may be stored in a .bmp or .jpeg or .png extensions etc.

Steps involved in a File Operation
1) Creating a new file
2) Opening an existing file
3) Reading from and writing information to a file
4) Closing a file

DEFINING, OPENING AND CLOSING OF FILES
File can defined by declaring a pointer of type **FILE**. For ex.

FILE *ptr; **// defined in the header file <stdio.h>**

The **FILE** is considered as opaque ***datatype*** i.e. its implementation is hidden. Although, it is type of **struct typedef** as **FILE**. However, the library knows it's internally type and how to use it.

Opening of a File
- **fopen( )** function can be used to create a new file or to open an existing file.
- This function initializes an object of the type **FILE**, which contains all the information necessary to control the information stored inside it.
- The syntax is shown below:
  FILE *fopen(const char *filename, const char *access_mode);
  where filename is string literal, which is used to name the file while access_mode can be one of the following values:

Mode Description
- **r** Opens an existing text file for reading purpose

- **w** Opens a text file for writing, if it does not exist then a new file is created. If a file already exists then it truncates or erases the contents
- **a** Opens a text file for writing in append mode
- **r+** Opens a text file for both reading and writing if the file exists
- **rb:** Opens a binary file in read mode
- **wb:** Opens a binary file in write mode

Closing a File

Any file that is opened should be closed after reading/writing a file.

- **fclose( )** function can be used to close a file.
- The syntax is shown below:
    - **int fclose(FILE *fp);**
- The function also flushes the entire buffer memory and closes all the references to the file pointer.
- The **fclose** function returns **zero** on success or returns **EOF** (special character generally **-1**) if there is an error.

Concept of beginning and end of file

Whenever a file is opened by **FILE *fp= fopen(.....)** function the file/stream pointer is positioned at the beginning of the file as shown in the fig. 3.2(a) below i.e. it is pointing to the first character '***I***' of the word ***Its.*** Similarly, whenever the file pointer reaches the end of the file as shown in the fig. 3.2(b), it points to the EOF character -1.

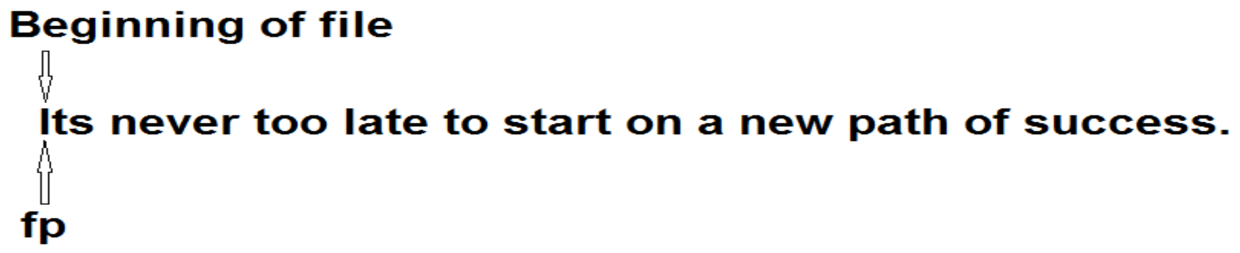


Fig. 3.2 (a): file pointer at the beginning of the file

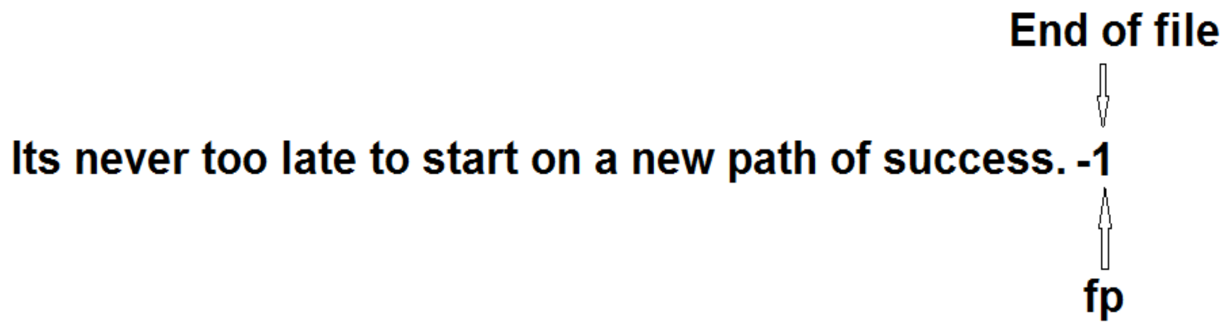


Fig. 3.2 (b): file pointer at the beginning of the file

INPUT/OUTPUT OPERATIONS

Writing to a File: **fputc(), fputs() & fprintf()**

The following function can be used to write individual characters to a stream:

int fputc(int c, FILE *fp);

- The function fputc() writes the character value of argument **c** to the output stream pointed by fp.
- This function returns the written character on success; otherwise returns EOF if there is an error.
- The following function can be used to write a null-terminated string to a stream:

int fputs(const char *s, FILE *fp);

- The function fputs() writes the string s to the file referenced by fp.
- This function returns a non-negative value on success; otherwise returns EOF if there is an error.
- You can use formatted output function to write a string into a file:

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

Ex.1. using fputs

```
#include <stdio.h>
void main()
{       FILE *fp;
        fp = fopen("sample.txt", "w+");
        fputs( "This message is added by fputs...\n",fp);
        fclose(fp);
}
```

- When the above code is compiled and executed, it creates a new file sample.txt in the current directory and writes the line using **fputs** function.

Ex.2. using fprintf

```
#include <stdio.h>
void main()
{
        FILE *fp;
        fp = fopen("sample1.txt", "w+");
        fprintf(fp, "This message is added by fprintf...\n");
        fclose(fp);
}
```

- When the above code is compiled and executed, it creates a new file sample1.txt in the current directory and writes a line using **fprintf** function.

Ex.3. using fprintf

```
#include <stdio.h>
void main()
{
	FILE *fp;
	char str[]=”Pr0gramming is fun”;
	fp = fopen("sample2.txt", "w+");
	fprintf(fp, "%s”,str);
	fclose(fp);
}
```

- When the above code is compiled and executed, it creates a new file sample2.txt in the current directory and writes the string pointed to by str using **fprintf** function.

Reading a File:

fgetc() , fgets() & fscanf()

• The following function can be used to read a text file character by character:

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

• The fgetc() function reads a character from the input file referenced by fp.

• This function returns the character being read on success; otherwise returns EOF if there is an error.

• The following function can be used to read a string from a stream:

```
char *fgets( char *buf, int n, FILE *fp );
```

• The functions fgets() reads up to n-1 characters from the input stream referenced by fp.

• It copies the read string into the buffer buf, appending a null character to terminate the string.

• If this function encounters a newline character '\n’, then it returns only the characters read up to that point including new line character.

int fscanf(FILE *fp, const char *format, ...) function can be used to read strings from a file but it stops reading after the first space character is encountered.

Please Note: Refer presentation slides for more details of the functions.

Assume that the **sample.txt** is already created In the path /Desktop/terms and contains the text : **Twinkle twinkle little star, How I wonder what you are?**

Ex. 1: using fscanf function

```
#include <stdio.h>
void main()
{       FILE *fp;
        char str[255];
        fp = fopen("D:\\DemoC\\Sample.txt", "r");
        fscanf(fp, "%s", str);       // reads a string into the array str
        printf("After fscanf: \n%s \n", str); // displays the first word of the string
                                              //to the output screen
        fclose(fp);
}
```

Output:

```
After fscanf:
Twinkle
```

Ex. 2: using fgets

```
#include <stdio.h>
void main()
{       FILE *fp;
        char str[255];
        fp = fopen("D:\\DemoC\\sample.txt", "r");
        fgets(str, 255, fp); // reads string from the file into the string str
        printf("After fgets:\n:%s \n",str);
        fclose(fp);
}
```

Output:

```
After fgets:
Twinkle twinkle little star, How I wonder what you are?
```

Programming Examples using files:

P1: Write a C program to open a file “data.txt” and count the number of characters present in it. Assume that data.txt is already present in the path D:\DemoC\.

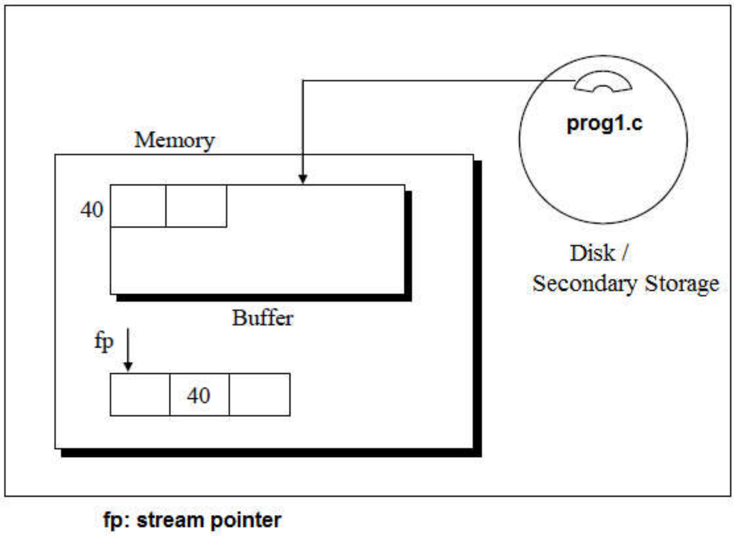
Solution: prog1.c

```
#include <stdio.h>
void main()
{       FILE *fp;
```

```
        int cnt=0;
        char ch;
        fp = fopen("D:\\DemoC\\data.txt", "r");
        while(1){
                fflush(stdin);
                ch=fgetc( fp); // reads string from the file into the string str
                if(ch==EOF)
                        break;
                cnt++;
        }
        printf("\n The number of characters present %d",cnt-1);
        fclose(fp);
}
```

The above program reads a character at a time from the file data.txt located in the path **D:\DemoC\** and counts the number of character's using the while loop until it reaches the EOF character equal to -1 to break the loop.

The function **fflush** mentioned above the line **ch=fgetc(fp);** indicates that the buffer a memory which is located between the program and file needs to be flushed. The below figure fig. 3.1 shows that the file pointer reads from the buffer (generally the buffer size varies from system to system 512KB , 4KB to 16KB size) depending upon the availability of the System memory.



fp: stream pointer

Fig. 3.1 : Buffer memory between the program and the Disk

P2: Write a C program to open a file “data.txt” in write mode and write a string “Hello !!! Welcome to the world of computing in C” to the file. Assume that data.txt needs to be created in the path D:\DemoC\.

Solution:

```
#include <stdio.h>
void main()
{       FILE *fp;
        int cnt=0;
        char str[]=”Hello !!! Welcome to the world of computing in C”;
        fp = fopen("D:\\DemoC\\data1.txt", "w");
        fprintf(fp,”%s”,str);
        fclose(fp);
}
```

Append to a file:

A file opened in append mode allows to add (append) data to the end of an existing file.

Gen. Syntax:

FILE *fopen(Filename, mode);

For Ex.

FILE *fp=fopen(“Message.txt, “a”);

The second argument to function **fopen** is the mode and specifying “**a**” means ***append*** i.e data can be added to the end of an existing file or if file is not present creates an empty file and then adds the data to it. When While **“a+”** plus sign is used with append it indicates that one can append data to the file as well as can ***read*** from the file.

Difference between write and append mode:

- The difference between opening a file in ***write mode*** and ***append*** is that in write mode, if the file already exist then the function **fopen** erases all the contents and then resets the file pointer to beginning of the file. Or if file does not exist it creates a new empty file.
- In case of append it opens an existing file and adds data to the end of the file. It does not erase the content of that file. And if the file is not present then creates an empty file and then adds the data to it.

P3: Write a C program to open a file "data.txt" in append mode and append a string "**Thi s is an appending string to the end portion of the file**" to the file. Assume that data.txt is already present in the path D:\DemoC\.

Solution:

```
#include <stdio.h>
void main()
{       FILE *fp;
        char str[]="This is an appending string to the end portion of the file";
        fp = fopen("D:\\DemoC\\data1.txt", "a+");
        fprintf(fp,"%s",str);
        fclose(fp);
}
```

Writing a newline character (LF) to a file

Before writing a newline character (**LF) ('\n')** to a text file on a disk in windows the newline character is converted into the carriage return & line feed ('\r' and '\n') combination. While reading from the text file the behavior is reversed i.e. the carriage return-linefeed combination ('\r','\n') on the disk is converted back into a newline ('\n').
For Ex.

- Windows uses CR+LF because DOS (Disk Operating System).
- Mac OS used CR for years until it switched to LF.
- Unix/Linux used just a single LF

Binary File(s):

A binary file is a collection of bytes, a string of 0s and 1s. For example it may be a compiled version of a *C program* a .exe file or a Media Player, Photoshop or any .bin file etc. A binary file contains 0s and 1s whenever read and not the text.

Text and binary mode files differ in

- Handling of newlines
- Storage of numbers

Handling of newlines
In text mode, a newline character is converted into the carriage return-linefeed (newline) ('\r' and '\n') combination before being written to the disk.
The behavior is reversed i.e. the carriage return-linefeed combination (\r\n) on the disk is converted back into a newline when the file is read by a C program. In case of binary file(s) these conversions do not take place.

Storage of numbers

Numbers are stored as strings of characters. For ex 1234, an integer although it occupies two bytes in memory, when transferred to the disk using fprintf(), would occupy ***four bytes***, ***one byte per character***.

While the floating-point number 1256.78 would occupy 7 bytes on disk ***(%0.2f***). The numbers with more digits would require more disk space.

P4. Write a C program to create a binary file MsgBinary.bin and display it to the screen.

```
#include <stdio.h>
void main()
{   int ch;
    char str[]={“Welcome to the world of C programming”};
    FILE *fp=fopen("D:\\DemoC\\MsgBinary.bin",“wb+");
     if(fp!=NULL) {
          while(1)  {              // creates a infinite loop
             fflush(stdin);
            ch=fgetc(fp);
           if ( ch==EOF)                    //break the loop on encountering EOF
                      break;
            fputc(ch,fp);     //writes byte read into ch to the file stream fp }
}
else
       printf("Error !!! Opening file");
rewind(fp);
if(fp!=NULL ){
          while(1){
                        fflush(stdin);
                        chr=fgetc(fp);
                if(chr==EOF)
                        break;
                cnt++;
                printf("%c ",chr);
          }
         }
printf("\ncnt=%d\n\n",cnt);
 fclose(fp);
}
```

P5. Write a C program to create a copy of binary file ***layout.bin*** available in the path D***:\DemoC\*** as ***layoutCopy.bin***.

Solution:

```
#include <stdio.h>
int main()
{   int ch;
   FILE *fp=fopen("D:\\DemoC\\layout.bin","rb");
   FILE *fc=fopen("D:\\DemoC\\layoutCopy.bin","wb+");

    if(fp!=NULL) {
        while(1)                          // Creates an infinite loop
        {         fflush(stdin);
           if ( ch=fgetc(fp)== EOF ) //Breaks the loop on encountering EOF
                 break;
           else
                 fputc(ch,fp);  // Writes the byte read into ch to the file stream fc
      }
    printf("\n File  layoutCopy.bin created successfully. \n");
    }
else
      printf("Error !!! Opening file");
   fclose(fp);
   fclose(fc);
}
```

The above program reads ***a character*** (***or a byte***) at a time from the file ***layout.bin*** located in the path **D:\DemoC\** and copies to the file ***layoutcopy.bin*** using the while loop until it reaches the ***EOF*** character equal to -1 to break the loop.

Random access to a file:

Unlike a sequential access, functions in C allow repositioning the file pointer to a byte location inside the file randomly. A file can be randomly accessed by using the functions such as ***fseek*** and ***rewind***.

int fseek(FILE *pointer, long Offset, int Position);

pointer:	pointer to a FILE object that identifies the stream.
offset:	number of bytes to offset from position of long type
position:	from where ***offset*** value is added to or subtracted w.r.t the position value.

On success returns **0** or ***non-zero*** value of **error.**

- The argument ***Position*** can take any of the three values:

SEEK_SET or a value **0** :	indicates beginning of the file
SEEK_CUR or a value **1** :	indicates current position of the file
SEEK_END or a value **2** :	indicates end of the file

For Ex.

fseek(fp, 0L, SEEK_SET);	// Sets file pointer to the beginning of the file
fseek(fp, 10L, 0);	// Sets file pointer 10 bytes from BOF the file
fseek(fp, 2L, SEEK_CUR);	// Advances 2 bytes from the current position
fseek(fp, 0L, SEEK_END);	// Sets file pointer to the end of the file
fseek(fp, -10L, 2);	// Sets file pointer 10 bytes prior to end of the file

P6. Write a C program to read last 100 bytes of a file ***Message.txt*** available in the path ***D:\demoC\\.*** Assume the following as the content of the file ***Message.txt***.

Why should you learn to write programs?
Writing programs (or programming) is a very creative and rewarding activity. You can write programs for many reasons, ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem. This book assumes that everyone needs to know how to program, and that once you know how to program you will figure out what you want to do with your newfound skills.

Solution:
```
#include <stdio.h>
int main()
{   int ch;
   FILE *fp=fopen("D:\\DemoC\\Message.txt","r+");
   if(fp!=NULL ){
        //Positions the file pointer to 100 bytes from the End of File
            fseek(fp,-100L,SEEK_END);      // The suffix L identifies long type value
             while(1) {  flush(stdin);
                  ch=fgetc(fp);
                  if(ch==EOF)
```

```
                        break;
          else{ fflush(stdin);
                    printf("%c",ch);
          }
   }
}
else
          printf("Error !!! Opening file");
fclose(fp);
printf("\n\n");
}
```

OUTPUT:
at once you know how to program you will figure out what
you want to do with your newfound skills.

ftell() : Returns current byte location in bytes as **long** type.

long ftell (FILE *pointer);

Accepts a stream pointer as an argument and returns the current byte location of the file pointer in the file as **long** type.

P7. Write a C program to print the number of bytes in a file (size of file) pointed to by the file pointer after positioning the pointer to the 0th (zero) offset from end of the file.

Solution:

```
#include <stdio.h>
int main()
{   FILE *fp=fopen("D:\\DemoC\\Message.txt","r+");
          if(fp!=NULL ){
                    fseek(fp,0L,SEEK_END);           // Positioning the pointer at the end of the file
                    printf("%ld bytes", ftell(fp));   // Prints the location in bytes  of the file pointer
          }
else
       printf("Error !!! Opening a file");
printf("\n\n");
fclose(fp);
}
```

***P.S: For more programs based on files handling -problem solving based on lab exercises refer power point slides used in the video lectures.**