File Handling

Unit-3

Random Access to a file

What is a Computer File ?

- 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.

  – 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 opening a file

- Creating a file.
- Opening an existing file.
- Reading from or Writing to a file.
- Closing a file.

DEFINING, OPENING AND CLOSING OF FILES

- File can defined by declaring a pointer of type **FILE**.
- For ex.

FILE *ptr;

- The keyword **FILE** is 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 its 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.
  - function initializes a variable of the type **FILE**, which contains all the information necessary to control the information stored inside it.
- The syntax:

  FILE *fopen(const char *filename, const char *access_mode);
  - where filename is string, used to identify the file.

Possible values of access_mode

- ***access_mode*** can possess one of the following values:
  - **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.
  - **a** Opens a text file for writing in append mode.

Contd..

- **r+** Opens a text file for both reading and writing.

  File must exist.

- **rb+** Opens a binary file for reading. File must exist
- **wb+** Opens a binary file for writing

Closing a File

- Any file that is opened for either for reading or writing should be closed.
- fclose() function can be used to close a file:

  int fclose(FILE *fp);

- The fclose( ) function returns **zero** on success or returns **EOF** (special character generally **-1**) if end of file is reached or there is an error.
- Function flushes the contents of the buffer and releases any buffers that the system reserved.

INPUT/OUTPUT OPERATIONS

- Basically three functions may be used to write to a File:
  - **fputc(), fputs() & fprintf()**
- Other three functions may be used to read from a file:
  - **fgetc(), fgets() & fsanf()**

fputc(): write a character to a file

- The function **fputc** can be used to write individual characters to the file pointed to by the file pointer **fp**:

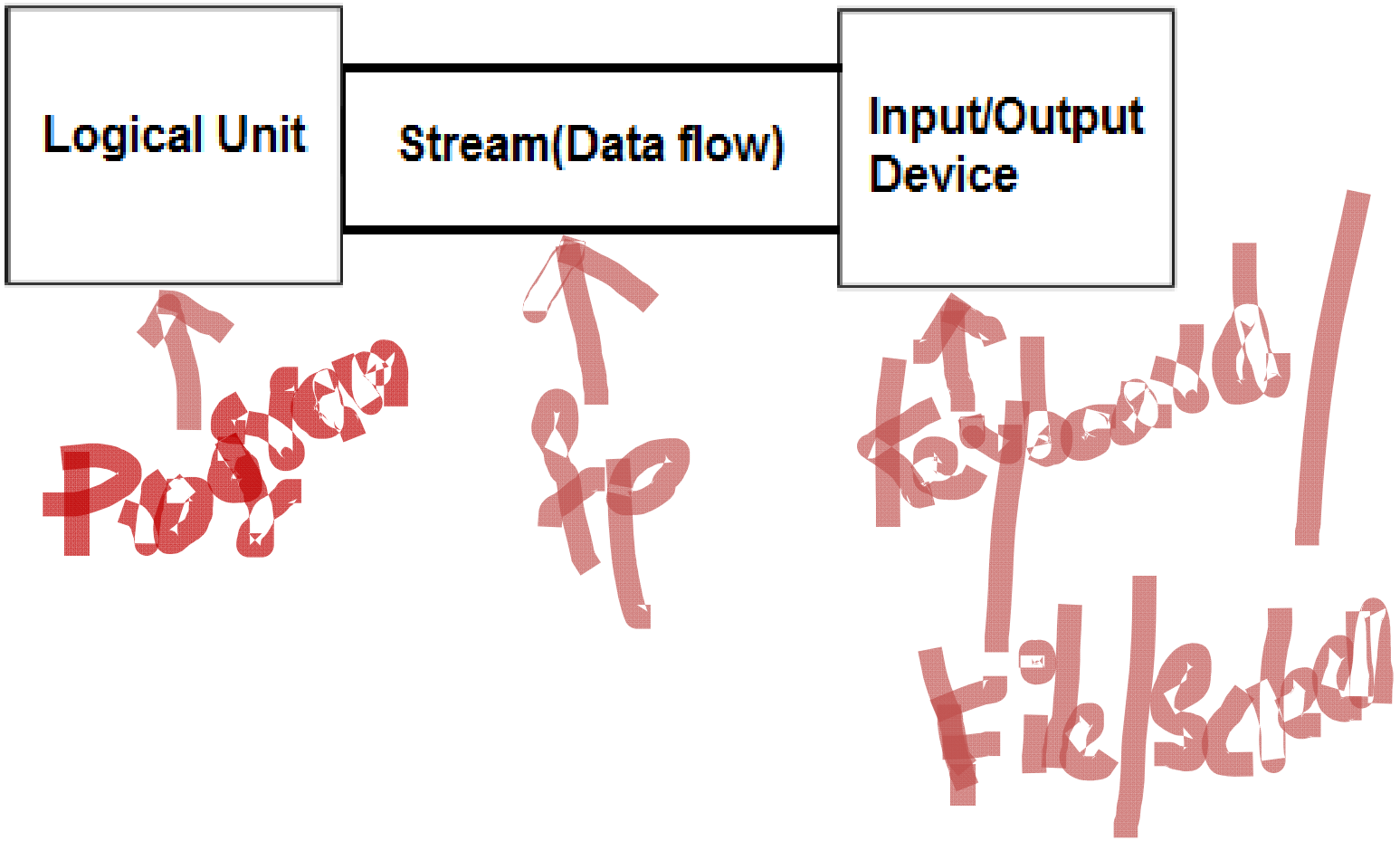
  int fputc (int c, FILE *fp);

  – 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 (-1)*** if there is an error.

Stream: flow of data

FILE ***fp= fopen(FILE_NAME, MODE_OF_ACCESS)**



Program using fputc()

```
#include <stdio.h>
#include <string.h>
void main()
{    int     i=0;
     char str[]={“Its never too late to start on a new path of success.”};
/* following statement creates a file Sample.txt if it doesn’t exist in the path
     D:\DemoC\Sample.txt   */
     FILE   *fp=fopen("D:\\DemoC\\Sample.txt","w+");
     if(fp !=NULL ){
              while(i < strlen(str) )
              {  fputc ( str[i], fp );    //writes the value of str[i] to the file into the
                 i++;                     // location pointed to by the file pointer fp
              }
   }
   fclose(fp);   // Closes the file pointer fp to the stream
 }
```

fgetc(): to read a char from a file

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

- Reads a single character from a file at a time.
- It returns the character present at the position indicated by the file pointer.
- After reading the character, the file pointer is automatically advanced to next character.

Contd..

- If pointer reaches the end of file or if an error occurs EOF file is returned.

```
//Program using fgetc() to read &  display the contents of the file
#include <stdio.h>
void main()
{    int    w, i;
     char  c;
// following statement opens the file Sample.txt created earlier
     FILE  *fp=fopen("D:\\DemoC\\Sample.txt",“r");
     if(fp!=NULL){
            while(!feof(fp))
            {           c=fgetc(fp);
                        printf("%c",c);
            }
      }
      else
         printf("Error !!! Opening the File.");
     fclose(fp);
   }
```

- **Output**

File Sample.txt created successfully with 53 characters.

The content of the file read are:

==========

Its never too late to start on a new path of success.

==========

```
//Another Sample program to demonstrate the use of fgetc
#include <stdio.h>
int main()
{   int   w, i;
    char ch;
    // following statement creates the file ASCII.txt if it doesn't
    // exist.
    FILE  *fp=fopen("D:\\DemoC\\ASCII.txt","w+");
    if(fp !=NULL ){
          for(i=65;i<91;i++)
                { w=fputc ( i, fp ); //writes the value of i to the file
                                     //pointed to the location by the
                                     //pointer fp
                printf("%c => %d\n",w,w);
                }
    }
   fclose(fp);
   return 0;
}
```

```
//Reading from the file created above and displaying it to the screen
   fp=fopen("D:\\DemocC\\ASCII.txt","r");
   if(fp!=NULL){
         while(!feof(fp))
         {
               ch=fgetc(fp);
               printf("%c",ch);
         }
   }
   else
          printf("\n Error !!! Opening the file.");
   fclose(fp);
    return 0;
}
```

Output:

A	**65**
B	**66**
C	**67**
D	**68**
E	**69**
F	**70**
G	**71**
H	**72**
I	**73**
J	**74**
K	**75**
L	**76**
M	**77**

N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90

fputs()

- The function **fputs()** can be used to write a string to the specified stream up to but not including the null character:

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

  - The function **fputs()** writes the string ***str*** to the file referenced by ***fp***.
  - This function returns a non-negative value on success; otherwise returns ***EOF*** if there is an error.

fgets()

The function **fgets()** can be used to read a string from a stream:

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

- **str** – The pointer to an array of chars where the string read is stored.
- **n** – The maximum number of characters to be read (including the final null-character). The length of the array passed as **str** is used.
- **stream** – The pointer to a **FILE** object that identifies the stream where characters are read from.

- Reads a line from the specified stream and stores it into the string pointed to by **str**.
- It stops when either (n-1) characters are read, or the newline character (‘**\n’**) is read, or the end-of-file (-1) is reached.
- On success, the function returns the same **str** parameter.
- If the **EOF** is encountered or an error occurs a null pointer is returned.

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{   int  val;
   char  *buf, *str="Welcome to the world of C Programming.";
   FILE *fp=fopen("D:\\DemoC\\SampleStr.txt","w+");
   if(fp!=NULL) {
      val=fputs(str,fp); // writes the string str to the stream fp
      if(val<0) {
          printf("Failed to write to the file !!!\n");
          exit(1);
      }
   }
   fclose(fp);
   return 0;
}
```

```
//Reading from the file created above and displaying it to
// the screen
fp=fopen("D:\\DemoC\\SampleStr.txt","r");
if(fp!=NULL){
    while(!feof(fp))
        {   fgets(buf,50,fp);   //reads 50 char's into buf
            printf("%s\n",buf); //prints the content of buf
        }
}
else
        printf("\n Error !!! Opening the file.");
fclose(fp);
return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{   int  val;
    char  *buf, *str={“Why should you learn to write programs
     Writing programs (or programming) is a very creative and
     rewarding activity.  You can write programs for manreasons,
     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.“};
    FILE *fp=fopen("D:\\DemoC\\SampleStr.txt","w+");
```

```
if(fp!=NULL) {
    val=fputs(str,fp); // writes the string str to
  the stream fp
    if(val<0) {
      printf("Failed to write to the file !!!\n");
            exit(1);
    }
  }
  fclose(fp);
```

```
//Reading from the file created above and displaying it to
// the screen
fp=fopen("D:\\DemoC\\SampleStr.txt","r");
if(fp!=NULL){
    while(!feof(fp))
        {   fgets(buf,50,fp);   //reads 50 char's into buf
            printf("%s\n",buf); //prints the content of buf
        }
}
else
        printf("\n Error !!! Opening the file.");
fclose(fp);
```

Output:

Why should you learn to write programs Writing programs (or programming) is a very creative and rewarding activity. You can write programs for man 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 New found skills.

fprintf()

- The function **fprintf()** can be used to write a string to a file:

  int fprintf(FILE *fp, const char *format, ...);
- **fp** – The pointer to a **FILE** object that identifies the stream where characters are written.
- **format** – The string that contains the text to be written to the stream.
- **[…]** – indicates an optional argument(s)
- On success, it **returns** the total number of characters written to the file. On error, it returns **EOF.**

fscanf()

- The function **fscanf()** can be used to read a formatted input from a stream:

int fscanf(FILE *stream, const char *format, ...);

- **stream** – The pointer to a FILE object that identifies the stream.
- **format** – The string that contains one or more of the following items – *Whitespace character, Non-whitespace character* and *Format specifiers*.
- returns the number of items assigned successfully otherwise EOF on error.

```
//Program using fprintf to write details of students to a file
#include <stdio.h>
int main()
{
   int  i, rollno, marks;
   char name[25];

   FILE *fp=fopen("D:\\DemoC\\StdData.txt","w+");

   if(fp!=NULL){
      for(i=1;i<=5;i++)
      {  printf("Enter the RollNo:");
         scanf("%d",&rollno);
```

```
            fflush(stdin);
            printf("Enter Name:");
            scanf("%15[^\n]s",name);
            printf("Enter the Percentage Marks:");
            scanf("%d",&marks);
            fprintf(fp,"%-4d%-15s%3d\n", rollno, name, marks );
        }
    }
    else
        printf("Error !!! Opening file");
    fclose(fp);
}
```

```
// Program reads from the file SampleData.txt & displays the
// students details it to the screen.
#include <stdio.h>
int  main()
{   int  rollno,marks;
    char name[25];
    FILE *fp=fopen("D:\\DemoC\\Sampledata.txt","w+");
    if(fp!=NULL){
          printf("Rollno\tName\tMarks\n");

    while(fscanf(fp,"%4d%15s%3d",&rollno,name,&marks)!=EOF) {
          printf("%d\t%s\t%d\n",rollno,name,marks);
    }
    }
```

```
    else
        printf("\n Its NULL");
fclose(fp);
return 0;
}
```

OUTPUT:

Rollno	Name	Marks
101	Amar S	88
102	Amit K	90
103	Devika J	90
104	Firoz S	75
105	Gayatri S	92

Append Mode: Appending 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.

Contd..

- **“a+”** : when plus sign is used with append it indicates that one can append data to the file as well as can ***read*** from the file. **fopen** creates a file if does not exist.

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.

```
//Program to append to a text file the contents of another file
#include <stdio.h>
int main()
{   int ch;
    FILE *fp=fopen("D:\\DemoC\\Message.txt","a+");
    FILE *fc=fopen("D:\\DemoC\\Partmsg.txt","r");

    if(fp!=NULL || fc !=NULL ){
        while(1)
        {           fflush(stdin);
                    ch=fgetc(fc);
                    if(ch==EOF)
                                break;
                     else{   printf("%c",ch);
                                fputc(ch,fp);
                    }
        }
```

```
        printf("\nData Appended to the file Successfully.\n");
    }
else
        printf("Error !!! Opening file");

   fclose(fp);
   fclose(fc);
}
```

Storing of newline character (LF)

- Before writing a newline character (**LF)** 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.
- 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 that stores strings 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.
- Binary files have faster read and write times than text files, because a binary image of the record is stored directly from memory to disk (or vice versa).

- 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.

    1234, 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. The numbers with more digits would require more disk space.

```
//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);
}
```

```
//Program to create a copy of binary file
#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 a infinite loop
        {  fflush(stdin);
           if ( ch=fgetc(fp)== EOF ) //break 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);
}
```

Random Access to a file

- Function to reposition the file pointer to a byte location inside the file.

  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
- **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**

- **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
```

ftell() : Current byte location

long ftell (FILE *pointer);

- Accepts a stream ***pointer*** as an argument.
- Returns the current location in bytes of **long** type.

rewind(): resets the pointer to the BOF

- Resets the file pointer to the beginning of the file.

  void rewind(FILE *pointer);

- **Pointer**: to a FILE object that identifies the stream.

```
/* Program to read last 50 bytes of a file Message.txt  */
#include <stdio.h>
int main()
{   int ch;
   FILE *fp=fopen("D:\\DemoC\\Message.txt","r+");
   if(fp!=NULL ){
//Positions the file pointer to 50 bytes from the End of File
fseek(fp,-50L,SEEK_END);     // L suffix identifies long type value
        while(1) {  flush(stdin);
                    ch=fgetc(fp);
                    if(ch==EOF)
                              break;
```

```
      else{
                              fflush(stdin);
                              printf("%c",ch);
                    }
      } /* end of while loop */
    }  /* end of if condition */
else
    printf("Error !!! Opening file");
    fclose(fp);
    printf("\n\n");
}
```

- **OUTPUT:**

what you want to do with your new found skills.

```
//Program to print the number of bytes in a file
#include <stdio.h>
int main()
{   int ch,cnt=0;
    long byteno;
    FILE *fp=fopen("D:\\DemoC\\Message.txt","r+");

     if(fp!=NULL ){
            // Positioning the pointer at the end of the file
            fseek(fp,0L,SEEK_END);
            // Prints the position of the pointer
            printf("%ld bytes", ftell(fp));
     }
```

```
 else
       printf("Error !!! Opening a file");
   printf("\n\n");
   fclose(fp);
}
```

Summary

- File is a computer resource stored on a digital storage permanently and identified by a name. For Ex. .txt, .bin, .c etc
- Calling any one of these functions declared in stdio.h, such as fputc(), fputs() or fprintf() causes a chunk of data to be copied to the buffer and then to file.
- Calling any one of these functions declared in stdio.h, such as fgetc(), fgets() or fscanf() causes a chunk of data to be copied from the file to the buffer.
- File can be randomly accessed using the functions such as fseek, ftell and rewind. It allows to reposition the stream pointer randomly to any location