PES UNIVERSITY

ELECTRONIC CITY CAMPUS

#5 : File Management

Programming Using C

UE19BC151

2020 JAN – MAY

Objectives

Unit Objectives are –

- To understand the concept of file storage and file input output operations
- To learn the usage of streams and its library functions

Files & Streams

- C views each file as a **sequence of bytes**
  - *File ends with the end-of-file marker*
  - Or, file ends at a specified byte

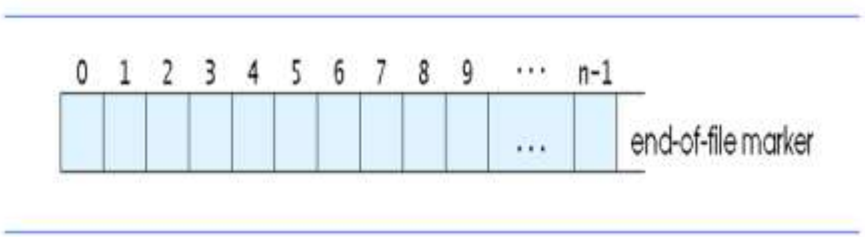


Fig. 11.2 C's view of a file of *n* bytes.

- File is a collection of data.
- It can be a database, a program, a letter or anything.
- We can create a file and access it using C.

- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a FILE structure
- Example file pointers:

stdin - standard input (keyboard)

stdout - standard output (screen)

stderr - standard error (screen)

File operations

Files & Streams

Basic File Operations

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

1) Opening Files

We can use the fopen( ) function to create a new file or to open an existing file. This call will initialize *an object of the type FILE,* which contains all the information necessary to control the stream.

Syntax

```
    FILE   *fp;

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

Files & Streams

- fp is declared as a pointer to the data type FILE.
- filename is a string - specifies the name of the file.
- fopen returns a pointer to the file which is used in all subsequent file operations.
- mode is a string which specifies the purpose of opening the file:
  - “r” :: open the file for reading only
  - “w” :: open the file for writing only
  - “a” :: open the file for appending data to it

Files & Streams

Examples:

```
FILE   *in,  *out ;
in  =  fopen (“mydata.txt”, “r”) ;
out  =  fopen (“result.txt”, “w”);

FILE   *empl ;
char  filename[25];
scanf  (“%s”, filename);
empl  =  fopen (filename, “r”) ;
```

Files & Streams

2) **Closing a File**

- After all operations on a file have been completed, it must be closed.

  Ensures that all file data stored in memory buffers are properly written to the file.

Syntax

```
    fclose  (file_pointer) ;
```

Eg -

```
 FILE   *xyz ;
 xyz  =  fopen (“test”, “w”) ;
 …….
 fclose (xyz) ;
```

Files & Streams

3) Read/Write Operations on Files

- The simplest file input-output (I/O) function are getc and putc.
- **getc is used to read a character from a file and return it.**

```
    char   ch;   FILE   *fp;

    .....

   ch  =  getc (fp) ;
```

- getc will return an end-of-file marker EOF, when the end of the file has been reached.
- **putc is used to write a character to a file.**

```
char   ch;   FILE   *fp;

 ......

putc  (ch, fp) ;
```

Read / Write functions

- **fgetc**
  - Reads one character from a file
  - Takes a FILE pointer as an argument
  - fgetc(stdin) equivalent to getchar()
- **fputc**
  - Writes one character to a file
  - Takes a FILE pointer and a character to write as an argument
  - fputc('a', stdout) equivalent to putchar('a')
- **Fgets / fputs**
  - Reads / writes a line from/to a file
- **fscanf / fprintf**
  - File processing equivalents of scanf and printf
- **fread / fwrite**
  - to write records (sequence of bytes) to the file. A record may be an array or a structure

Files & Streams

File open modes

•r	Open a file for reading.
•w	Create a file for writing. If the file already exists, discard the current contents.
•a	Append; open or create a file for writing at end of file.
•r+	Open a file for update (reading and writing).
•w+	Create a file for update. If the file already exists, discard the current contents.
•a+	Append; open or create a file for update; writing is done at the end of the file.
•rb	Open a file for reading in binary mode.
•wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
•ab	Append; open or create a file for writing at end of file in binary mode.

Files & Streams

rb+	Open a file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append; open or create a file for update in binary mode; writing is done at the end of the file.

Files & Streams

- **How to check EOF condition when using fscanf?**

  Use the function feof

```
if   (feof (fp))
   printf  ("\n Reached end of file") ;
```

- **How to check successful open?**

  For opening in "r" mode, the file must exist.

```
if   (fp == NULL)
   printf  ("\n Unable to open file") ;
```

Example Programs

EXAMPLE : writing the contents in file using fprintf()

```
FILE *fp;
fp= fopen(“sample.txt”, ”w”);
fprintf(fp, ”WELCOME TO C PROGRAMMING CLASS”);
fprintf (fp, ”this is a sample data”);
fclose (fp);
```

Note: after executing the program, check the contents of file sample.txt

EXAMPLE : writing the contents from the file to consol

```
FILE *fp;
char c;
fp= fopen(“sample.txt”, ”r”);
if(fp==NULL)
{
    printf(“file doesnot exist”);
    exit();
}
else
{
while(fp!=EOF)
{
    c=getc(fp);
    putchar(c);
}
}
fclose (fp);
```

Files & Streams

We can also use the file versions of **scanf and printf, called fscanf and fprintf**.

General format:

```
fscanf  (file_pointer, control_string, list) ;
fprintf  (file_pointer, control_string, list) ;
```

Examples:

```
fscanf  (fp, “%d %s %f”, &roll, dept_code, &cgpa) ;
fprintf  (out, “\nThe result is: %d”, xyz) ;
```

EAXMPLE: read data from the file using fscanf ( )

```
struct  student {
          int     roll;
           int dept_code[6];
          float  cgpa;
} ;
main()   {
   FILE   *fp;
   student  s;
   float  sum = 0.0;
   int  count = 0;
   fp =fopen (“student.txt”, “r”) ;

while (1)   {
      if  (feof (fp))   break;
      fscanf (fp, “%d  %d %f”, &s.roll,
                  &s.dept_code, &s.cgpa);
      count ++;
      sum = sum + s.cgpa;
   }
   printf (“\nThe average cgpa is %f”,
                  sum/count);
   fclose (fp);
}
```

Example: Write a Program to copy the contents of one file into another using fgetc and fputc function

```
void main() {
    FILE *fp1, *fp2;
    char a;
    clrscr();

    fp1 = fopen("test.txt", "r");
    if (fp1 == NULL) {
        puts("cannot open this file");
        exit(1);
    }

    fp2 = fopen("test1.txt", "w");
    if (fp2 == NULL) {
        puts("Not able to open this file");
        fclose(fp1);
        exit(1);
    }

    do {
        a = fgetc(fp1);
        fputc(a, fp2);
    } while (a != EOF);

    fcloseall();
    getch();
}
```

EXAMPLE PROGRAM FOR FILE OPEN, FILE WRITE AND FILE CLOSE IN C LANGUAGE USING FPUTS()

```
int main( ) {
    FILE *fp ;
    char data[50];
    printf( "Opening the file test.c in write mode" ) ;        // opening an existing file
    fp = fopen("test.c", "w") ;
    if ( fp == NULL ) {
        printf( "Could not open file test.c" ) ;
        return 1;   }
    printf( "\n Enter some text from keyboard to write in the file test.c" ) ;  // getting input
                                                                                   from user
   while ( strlen ( gets( data ) ) > 0 )  {
        fputs(data, fp) ;   // writing in the file
        fputs("\n", fp) ; }
    printf("Closing the file test.c") ; // closing the file
    fclose(fp) ;
    return 0;
}
```

Example: Open, Read and close a file: reading string by string using fgets()

```
int main( ) {
        FILE *fp ;
        char data[50] ;
        printf( "Opening the file test.c in read mode" ) ;
        fp = fopen( "test.c", "r" ) ;
        if ( fp == NULL)    {
                printf( "Could not open file test.c" ) ;
                return 1;       }
        printf( "Reading the file test.c" ) ;
        while( fgets ( data, 50, fp ) != NULL )
        printf( "%s" , data ) ;
        printf("Closing the file test.c") ;
        fclose(fp) ;
        return 0;
}
```

Example: Write a C program to convert the file contents in Upper-case & Write Contents in a output file. (using fgetc and fputc)

```c
void main() {
    FILE *fp1, *fp2;
    char a;
    clrscr();

    fp1 = fopen("test.txt", "r");
    if (fp1 == NULL) {
        puts("cannot open this file");
        exit(1);
    }

    fp2 = fopen("test1.txt", "w");
    if (fp2 == NULL) {
        puts("Not able to open this file");
        fclose(fp1);
        exit(1);
    }

    do {
        a = fgetc(fp1);
        a = toupper(a);
        fputc(a, fp2);
    } while (a != EOF);

    fcloseall();
    getch();
}
```

C Program to Append the Content of File at the end of Another

```
FILE *fsring1, *fsring2, *ftemp;
char ch, file1[20], file2[20], file3[20];
printf("Enter name of first file ");             gets(file1);
printf("Enter name of second file ");       gets(file2);
printf("Enter name to store merged file "); gets(file3);
fsring1 = fopen(file1, "r");     fsring2 = fopen(file2, "r");
if (fsring1 == NULL || fsring2 == NULL) { printf("file does not exist");
 exit(); }
ftemp = fopen(file3, "w");
if (ftemp == NULL) { printf (“ file does not exist ");                     exit();      }
while ((ch = fgetc(fsring1)) != EOF)
    fputc(ch, ftemp);
while ((ch = fgetc(fsring2) ) != EOF)
    fputc(ch, ftemp);
printf("Two files merged  %s successfully.\n", file3);
fcloseall();
```

PES UNIVERSITY
ELECTRONIC CITY CAMPUS

C Program to Find the Number of Lines in a Text File

```
FILE *fileptr;
    int count_lines = 0;
    char filechar[40], chr;
    printf("Enter file name: ");
    scanf("%s", filechar);
    fileptr = fopen(filechar, "r");
   chr = getc(fileptr); //extract character from file and store in chr
    while (chr != EOF)  {
            if (chr == 'n') //Count whenever new line is encountered
        {
            count_lines = count_lines + 1;
        }
     chr = getc(fileptr); //take next character from file.
    }
    fclose(fileptr); //close file.
    printf("There are %d lines in %s  in a file\n", count_lines, filechar);
```

Use of fwrite()

- The fwrite() function is used to write records (sequence of bytes) to the file. A record may be an array or a structure.
- The function take 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 pointer where the records will be written in binary mode.

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

Write to a file using fwrite()

```
struct Student     {          int roll;    char name[25];          float marks;    };
          FILE *fp;
          char ch;
          struct Student Stu;
          fp = fopen("Student.txt","w");
          if(fp == NULL) {
               printf("\nCan't open file or file doesn't exist.");  exit(0);  }
          do {
               printf("\nEnter Roll : ");   scanf("%d",&Stu.roll);
               printf("Enter Name : ");    scanf("%s",Stu.name);
               printf("Enter Marks : ");  scanf("%f",&Stu.marks);
               fwrite(&Stu,sizeof(Stu),1,fp);
               printf("\nDo you want to add another data (y/n) : ");
               ch = getche();
               }while(ch=='y' || ch=='Y');
             printf("\nData written successfully...");
          fclose(fp);
```

Use of fread()

- 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.

Read from a file using fread()

```
struct Student  {          int roll;          char name[25];      float marks;    };
        FILE *fp;
        char ch;
        struct Student Stu;
        fp = fopen("Student.txt","r");
        if(fp == NULL)
        {
              printf("\nCan't open file or file doesn't exist.");
              exit(0);
        }
        printf("\n\tRoll\tName\tMarks\n");
        while(fread(&Stu,sizeof(Stu),1,fp)>0)
                        printf("\n\t%d\t%s\t%f",Stu.roll,Stu.name,Stu.marks);

        fclose(fp);
```

Buffer in C

Files & Streams

Buffer in C

- Temporary storage area is called buffer.
- All standard input output devices are containing input output buffer.
- **In implementation when we are passing more than required number of values as a input then rest of all values will automatically holds in standard input buffer, this buffer data will automatically pass to next input functionality if it is exist.**

C library function - fflush()

- Description: The C library function **int fflush(FILE *stream)** flushes the output buffer of a stream.

- Following is the declaration for fflush() function.

  int fflush(FILE *stream)

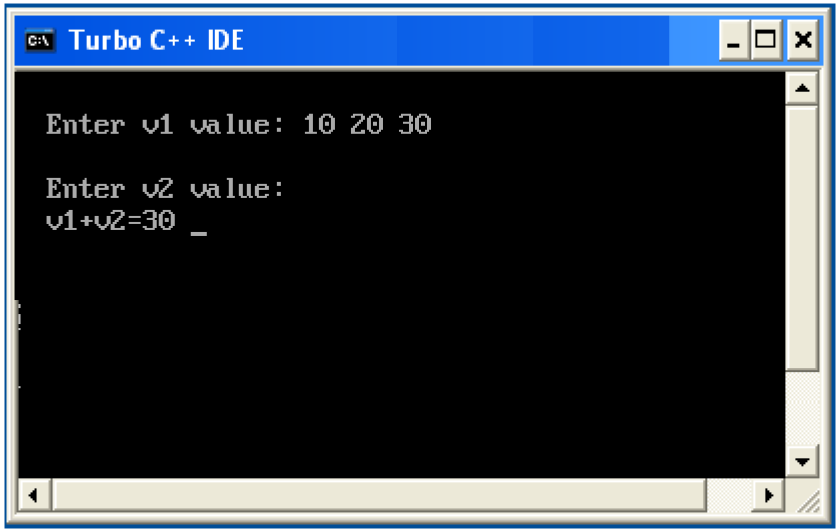
- Parameters **stream** − This is the pointer to a FILE object that specifies a buffered stream.

- Return Value: This function returns a zero value on success. If an error occurs, EOF is returned and the error indicator is set (i.e. feof).

NEED OF BUFFER

Eg –

```
    #include<stdio.h>
#include<conio.h>
void main()
{
int v1,v2;
clrscr();
printf("\n Enter v1 value: ");
scanf("%d",&v1);
printf("\n Enter v2 value: ");
scanf("%d",&v2);
printf("\n v1+v2=%d ",v1+v2);
getch();
}
```



```
Enter v1 value: 10 20 30

Enter v2 value:
v1+v2=30 _
```

In the above example we pass three input in v1 and we cannot pass any value in v2 but value of v1 is automatically pass in v2.

In implementation when we need to remove standard input buffer data then go for flushall() or fflush() function.

C Library Functions

flushall()

it is a predefined function which is declared in stdio.h by using flushall we can remove the data from standard input output buffer.

fflush()

it is a predefined function in "stdio.h" header file used to flush or clear either input or output buffer memory.

fflush(stdin)

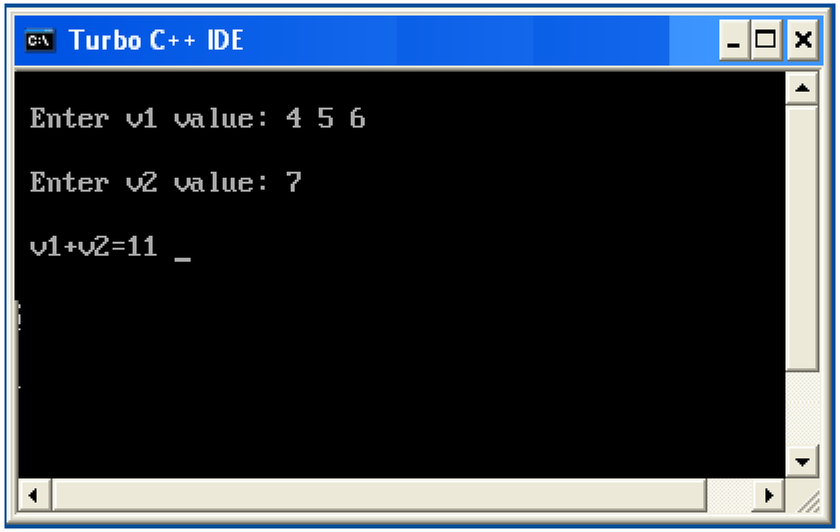
it is used to clear the input buffer memory. It is recommended to use before writing scanf statement.

fflush(stdout)

it is used to clear the output buffer memory. It is recommended to use before printf statement.

Use of fflush function example

```
#include<stdio.h>
#include<conio.h>
void main()
{
int v1,v2;
clrscr();
printf("\n Enter v1 value: ");
scanf("%d",&v1);
printf("\n Enter v2 value: ");
fflush(stdin);
scanf("%d",&v2);
printf("\n v1+v2=%d ",v1+v2);
getch();
}
```



```
Enter v1 value: 4 5 6

Enter v2 value: 7

v1+v2=11 _
```

Random access in file

Random access in file

- There is no need to read each record sequentially, if we want to access a particular record. C supports these functions for random access file processing.
  - fseek()
  - ftell()
  - rewind()

fseek(): This function is used for seeking the pointer position in the file at the specified byte.
Syntax: fseek(file pointer, displacement, pointer position);
file pointer : It is the pointer which points to the file.
Displacement : It is positive or negative. This is the number of bytes which are skipped backward (if negative) or forward(if positive) from the current position. This is attached with L because this is a long integer.
Pointer position: This sets the pointer position in the file. Value pointer position 0 Beginning of file. 1 Current position. 2 End of file

Examples of fseek()

1) fseek(p,10L,0)

0 means pointer position is on beginning of the file. From this statement pointer position is skipped 10 bytes from the beginning of the file.

2)fseek(p,5L,1)

1 means current position of the pointer position. From this statement pointer position is skipped 5 bytes forward from the current position.

3)fseek(p,-5L,1)

From this statement pointer position is skipped 5 bytes backward from the current position.

Random access in file

ftell()

This function returns the value of the current pointer position in the file. The value is count from the beginning of the file.

Syntax: ftell(fptr);

Where fptr is a file pointer.

rewind()

This function is used to move the file pointer to the beginning of the given file.

Syntax: rewind(fptr);

Where fptr is a file pointer.

Write a program to read last ‘n’ characters of the file using appropriate file functions(Here we need fseek() and fgetc()).

```
FILE *fp;
char ch;
 fp=fopen("file1.c", "r");
 if(fp==NULL)  printf("file cannot be opened");
else    {
          printf("Enter value of n  to read last ‘n’ characters");
          scanf("%d",&n);
          fseek(fp,-n,2);
          while((ch=fgetc(fp))!=EOF)
           {
               printf("%c\t",ch);
          }     } fclose(fp);
```

Program to understand rewind() and ftell function

```
FILE *fp;
fp=fopen("file10.c","r");
if(fp==NULL)
      printf("Error in opening file\n");

printf("Position pointer %ld\n",ftell(fp));
fseek(fp,0,2); //move to fp to EOF

printf("Position pointer %ld\n",ftell(fp));
rewind(fp); //move to fp to beginning of file

printf("Position pointer %ld\n",ftell(fp));
fclose(fp);
```

Outcomes

The students will be able to –

- Apply file operations to perform read/write functions in file
- Usage of fflush function in C