CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia
Assistant Professor

Department of Computer Science
Banaras Hindu University
Varanasi
Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

February 27, 2021

Function basics in C I

✓ C provides a wide range of functions in the header file stdio.h for reading and writing data to and from the file.

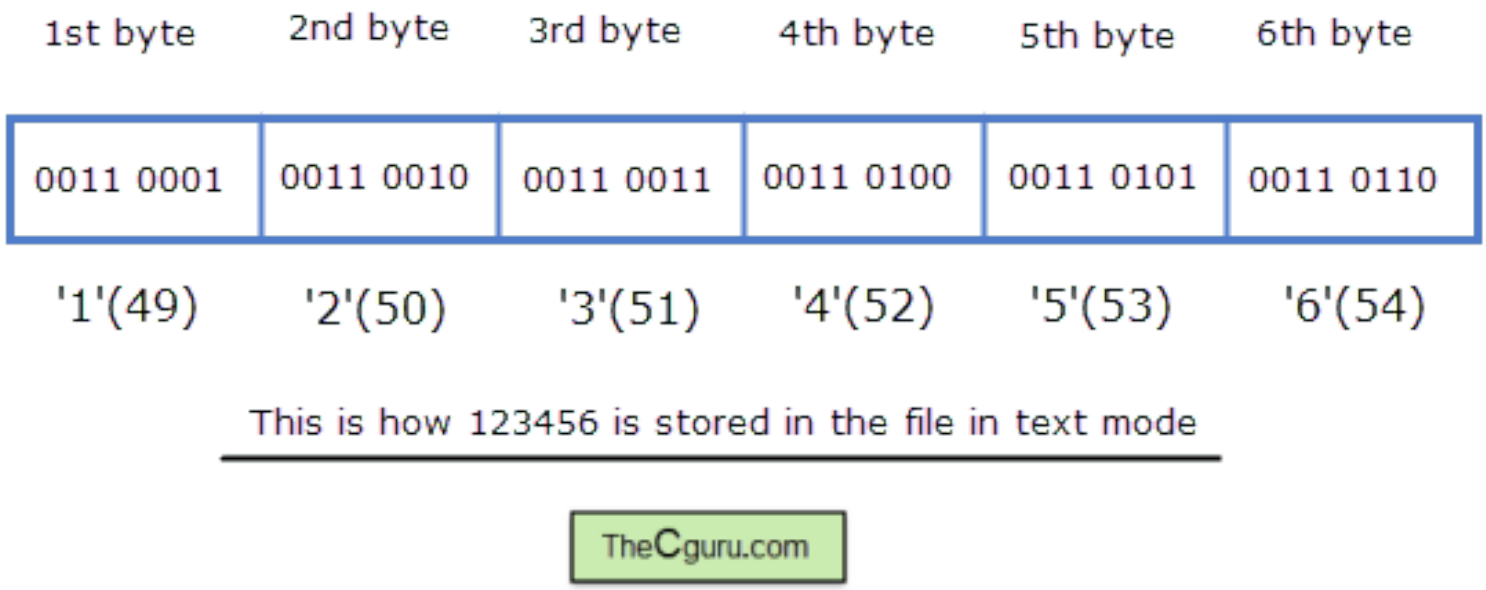
✓ Text and Binary Mode We can store data into files in two ways:

1. Text mode.
2. Binary mode.

✓ In Text mode, data is stored as a line of characters terminated by a newline character *(”)* where each character occupies 1 byte. To store 1234 in a text file would take 4 bytes, 1 byte for each character.

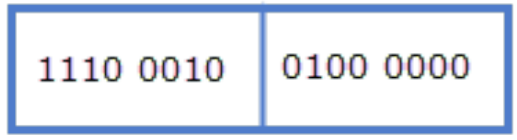
✓ The important thing to note in the Text mode what gets stored in the memory is that binary equivalent of the ASCII number of the character. Here is how 123456 is stored in the file in text mode.

Function basics in C II



As you can see it takes 6 bytes to store 123456 in text mode.
In binary mode, data is stored on a disk in the same way as it is represented in computer memory. As a result, storing 123456 in a binary mode would take only 2 bytes. In Linux systems, there is no difference between text mode and binary mode. Here is how 123456 is stored in the file in binary mode.

Function basics in C III



This is how 123456 is stored in the file in binary mode

TheCguru.com

By using binary mode, we can save a lot of disk space.

✓ In the text file, each line is terminated by one or two special characters.

✓ In Windows system, each line ends with a carriage return (’
r’) immediately followed by a newline (’
n’) character.

✓ On the other hand in Unix, Linux and Mac operating systems each line ends with as single newline (’
n’) character.

Function basics in C IV

✓ Text file as well as binary files keep track of the length of the file and also track end of the file. But in the text file, there is one more way to detect the end of the file.

✓ The text file may contain a special end of file marker having ASCII value of 26. All input/output functions stop reading from a file when this character is encountered and return an end of file signal to the program.

✓ No input/output functions insert this character automatically. In Windows, you can insert this character into the file using Ctrl+Z.

✓ It is important to note that there is no requirement that Ctrl+Z must be present, but if it is, it is considered as the end of the file.

✓ Any characters after Ctrl+Z will not be read. On the other hand, Operating systems based upon Unix have so such special end of file character.

✓ Another important thing you need to remember is that text files are portable. It means you can create a text file using Windows and open it a Linux system without any problem.

✓ On the other hand, the size of the data types and byte order may vary from system to system, as a result, binary files are not portable.

Buffer Memory ✓ Reading and writing to and from the files stored in the disk is relatively slow process when compared to reading and writing data stored in the RAM.

✓ As a result, all standard input/output functions uses something called buffer memory to store the data temporarily.

✓ The buffer memory is a memory where data is temporarily stored before it is written to the file.

✓ When the buffer memory is full the data is written(flushed) to the file. Also when the file is closed the data in the buffer memory is automatically written to the file even if the buffer is full or not.

✓ This process is called flushing the buffer.

✓ You don't have to do any anything to create buffer memory. As soon as you open a file, a buffer memory will be created for you automatically behind the scenes.

✓ However, there are some rare occasions when you have to flush the buffer manually. If so, you call use fflush() function.

Opening a File

✓ Before any I/O (short of input/output) can be performed on a file you must open the file first. fopen() function is used to open a file.

Function basics in C VI

Syntax:

```
FILE *fopen(const char *filename, const char *mode);

filename: string containing the name of the file.

mode: It specifies what you want to do with the file i.e read, write, append.
```

✓ On Success fopen() function returns a pointer to the structure of type FILE. FILE structure is defined in stdio.h and contains information about the file like name, size, buffer size, current position, end of file etc.

On error

```
fopen() function returns NULL.
```

Function basics in C VII

w	(write) - This mode is used to write data to the file. If the file doesn't exist this mode creates a new file. If the file already exists then this mode first clears the data inside the file before writing anything to it.
a	(append) - This mode is called append mode. If the file doesn't exist this mode creates a new file. If the file already exists then this mode appends new data to end of the file.
r	(read) - This mode opens the file for reading. To open a file in this mode file must already exist. This mode doesn't modify the contents of the file in anyway. Use this mode if you only want to read the contents of the file.
w+	(write + read) - This mode is a same as"w" but in this mode, you can also read the data. If the file doesn't exist this mode creates a new file. If the file already exists then previous data is erased before writing new data.
r+	(read + write) - This mode is same as "r" mode, but you can also modify the contents of the file. To open the file in this mode file must already exist. You can modify data in this mode but the previous contents of the file are not erased. This mode is also called update mode.
a+	(append + read) - This mode is same as "a" mode but in this mode, you can also read data from the file. If the file doesn't exist then a new file is created. If the file already exists then the new data is appended to the end of the file. Note that in this mode you can append data but can't modify existing data.

Mode	Description
"wb" Open the file in binary mode	
"a+b" or "ab+" Open the file in append + read in binary mode	

Mode	Description
"w" or "wt"	Open a file for writing in text mode.
"r" or "rt"	Open a file for reading in text mode.

Note: mode is a string so you must always use double quotes around it.

```
1 fopen("somefile.txt", 'r'); // Error
2 fopen("somefile.txt", "r"); // Ok
```

✓ If you want to open several files at once, then they must have their own file pointer variable created using a separate call to fopen() function.

```
File fp1 = fopen("readme1.txt", 'r');
File fp2 = fopen("readme2.txt", 'r');
File fp3 = fopen("readme3.txt", 'r');
```

Checking for Errors

✓ As already said fopen() returns NULL when an error is encountered in opening a file. So you before doing any operation on the file you must always check for errors.

```
//Checking for errors

File *fp;
fp = fopen("somefile.txt", "r");

if(fp == NULL)
{
     // if error opening the file show error message and exit the program
     printf("Error opening a file");
     exit(1);
}
```

✓ We can also give the full path name to fopen() function. Suppose we want to open a file list.txt located in /home/downloads/ .Here is how to specify pathname in fopen().

```
fp = fopen("/home/downloads/list.txt", "r");
```

✓ t is important to note that Windows uses backslash character (’ ́) as a directory separator but since C treats backslash as the beginning of escape sequence we can’t directly use ’ćharacter inside the string. To solve this problem use two ’ín place of one ’ ́.

```
fp = fopen("C:\home\downloads\list.txt", "r"); // Error
fp = fopen("C:\\home\\downloads\\list.txt", "r"); // ok
```

✓ The other technique is to use forward slash (’/’) instead of a backward slash (’ ́).

```
fp = fopen("C:/home/downloads/list.txt", "r");
```

Closing a File:

✓ When you are done working with a file you should close it immediately using the fclose() function.

Syntax: int fclose(FILE *fp);

✓ When a file is closed all the buffers associated with it are flushed i.e data in the buffer is written to file.

✓ Closing the file through fclose() is not mandatory, because as soon as the program ends all files are closed immediately. But it is good practice to close the file as soon you are finished working with it. Otherwise, in a large program, you would be wasting a lot of space to unused file pointers.

✓ On success fclose() returns 0, on error, it returns EOF i.e End Of File which is a constant defined in stdio.h and its value is -1.

```
1 fclose(fp1);
2 fclose(fp2);
```

✓ You can also use fcloseall() function to close all opened files at once.

✓ On success fcloseall() returns the number of files closed, on error, it returns EOF.

```c
// check whether all files are closed or not
n = fcloseall();
if(n == EOF)
{
     printf("Error in closing some files \n");
}
else
{
     printf("%d files closed \n", n);
}
```

End Of File - EOF:

✓ File reading functions need to know when the end of the file is reached. So when the end of file is reached the operating system sends an end of file or EOF signal to the program to stop reading.

✓ When the program receives the signal the file reading functions stops reading the file and returns EOF.

✓ As already said EOF is a constant defined in stdio.h header file and has a numerical value of -1.

Function basics in C XIII

✓ It is important to understand that EOF is not a character present at the end of the file instead it is returned by the file reading functions when the end of the file is reached.

```c
// Basic workflow of a file program in C
int main()
{
     FILE *fp; // declare file pointer variable
     fp = fopen("somefile.txt", "w"); // fopen() function called

     /*
     do something here
     */
     fclose(fp); // close the file
}
```

: Syntax: int fputc(int ch, FILE *fp);

✓ The fputc() function is used to write a single character specified by the first argument to a text file pointed by the fp pointer.

✓ After writing a character to the text file, it increments the internal position pointer.

✓ If the write is successful, it returns the ASCII value of the character that was written. On error, it returns EOF i.e -1.

✓ Although, the formal definition of fputc() says "it writes a single character to the file" that's not how it is implemented.

✓ In practice writing a single character one by one would be very inefficient and slow.

✓ Instead if writing characters one by one into the file, they are accumulated in a buffer memory.

✓ Once the number of characters reaches a reasonable number, they are written to the file in one go.

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

int main()
{
     int ch;
     FILE *fp;
     fp = fopen("myfile.txt", "w");

     if(fp == NULL)
     {
          printf("Error opening file\n");
          exit(1);
     }

     printf("Press Ctrl+Z in DOS and Ctrl+D\n\
     in Linux to stop reading more characters\n\n");

     printf("Enter text: ");

     while( (ch=getchar()) != EOF )
     {
```

```
23              fputc(ch, fp);
24          }
25
26          fclose(fp);
27
28          return 0;
29
30  }
```

fgetc() Function in C:

Syntax: int fgetc(FILE *fp);

✓ This function is complementary to fputc() function. It reads a single character from the file and increments the file position pointer.

✓ To use this function file must be opened in read mode. On success, it returns the ASCII value of the character but you can also assign the result to a variable of type char.

✓ On failure or end of file, it returns EOF or -1.

✓ Just as fputc() this function uses buffer memory too.

✓ So instead of reading a single character from the file one by one, a whole block of characters from the file is read into the buffer.

✓ The characters are then handed over one at a time to the function fgetc(), until the buffer is empty.

✓ If there are still some characters left to read in the file then again a block of characters is read into the buffer.

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

int main()
{
     int ch;
     FILE *fp;
     fp = fopen("myfile.txt", "r");

     if(fp == NULL)
     {
          printf("Error opening file\n");
          exit(1);
     }

     printf("Reading contents of myfile.txt: \n\n");

     while( (ch=fgetc(fp)) != EOF )
     {
          printf("%c", ch, ch);
     }

```

```
23      fclose(fp);
24      return 0;
25  }
```

fputs() Function in C:

Syntax: int fputc(const char *str, FILE *fp);

✓ This function is used to print a string to the file. It accepts two arguments pointer to string and file pointer.

✓ It writes a null-terminated string pointed by str to a file.

✓ The null character is not written to the file. On success, it returns 0. On error, it returns EOF or -1.

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

int main()
{
     char str[50];
     FILE *fp;
     fp = fopen("myfile2.txt", "w");

     if(fp == NULL)
     {
          printf("Error opening file\n");
          exit(1);
     }
```

```
15
16      printf("Testing fputs() function: \n\n");
17      printf("To stop reading press Ctrl+Z in windows and Ctrl+D in Linux :");
18
19      while( gets(str) != NULL )
20      {
21           fputs(str, fp);
22      }
23
24      fclose(fp);
25      return 0;
26  }
```

✓ gets() function converts newline character entered to null character (’
0’).

✓ When the end of file character is encountered gets() returns NULL.

Difference between puts() and fputs():

✓ The important difference between fputs() and puts() is that, the puts() converts the null character(’
0’) in the string to the newline (’
n’) character whereas fputs() doesn’t not.

fgets() Function in C:

Syntax: char *fgets(char *str, int n, FILE *fp);

✓ The function reads a string from the file pointed to by fp into the memory pointed to by str.

✓ The function reads characters from the file until either a newline (’n’) is read or n-1 characters is read or an end of file is encountered, whichever occurs first.

✓ After reading the string it appends the null character (’0’) to terminate the string. On success, it returns a pointer to str.

✓ On error or end of the file it returns NULL.

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int main()
5  {
6       char str[50];
7       FILE *fp;
8       fp = fopen("myfile2.txt", "r");
9
10      if(fp == NULL)
```

```
11      {
12           printf("Error opening file\n");
13           exit(1);
14      }
15
16      printf("Testing fgets() function: \n\n");
17      printf("Reading contents of myfile.txt: \n\n");
18
19      while( fgets(str, 30, fp) != NULL )
20      {
21           puts(str);
22      }
23
24      fclose(fp);
25
26      return 0;
27 }
```

Difference between fgets() andgets() function:

✓ The gets() reads the input from standard input while fgets() reads from the file.

✓ When gets() reads the input from standard input it converts the newline (’

n’) to the null character (’

0’) on the other hand when fgets() reads newline (’

n’) character from the file it doesn’t convert it into null character (’

0’), it is retained as it is.

fprintf() Function in C:

Syntax: int fprintf(FILE *fp, const char *format [, argument, ...]);

✓ The fprintf() function is same as printf() but instead of writing data to the console, it writes formatted data into the file.

✓ Almost all the arguments of fprintf() function is same as printf() function except it has an additional argument which is a file pointer to the file where the formatted output will be written.

✓ On success, it returns the total number of characters written to the file. On error, it returns EOF.

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

int main()
{
     FILE *fp;
     char name[50];
     int roll_no, chars, i, n;
     float marks;

     fp = fopen("records.txt", "w");

```

```
13      if(fp == NULL)
14      {
15           printf("Error opening file\n");
16           exit(1);
17      }
18
19      printf("Testing fprintf() function: \n\n");
20
21      printf("Enter the number of records you want to enter: ");
22      scanf("%d", &n);
23
24      for(i = 0; i < n; i++)
25      {
26           fflush(stdin);
27           printf("\nEnter the details of student %d \n\n", i +1);
28
29           printf("Enter name of the student: ");
30           gets(name);
31
32           printf("Enter roll no: ");
33           scanf("%d", &roll_no);
34
35           printf("Enter marks: ");
```

```
36          scanf("%f", &marks);
37
38          chars = fprintf(fp, "Name: %s\t\tRoll no: %d\t\tMarks: %.2f\n",
39          name, roll_no, marks);
40          printf("\n%d characters successfully written to the file\n\n", chars);
41      }
42
43      fclose(fp);
44      return 0;
45  }
```

fscanf() Function in C:

Syntax: int fscanf(FILE *fp, const char *format [, argument, ...]);

✓ The fscanf() function is used to read formatted input from the file.

✓ It works just like scanf() function but instead of reading data from the standard input it reads the data from the file.

✓ In fact, most of the arguments of fscanf() function are same as scanf() function, except it just needs an additional argument obviously enough a file pointer.

✓ On success, this function returns the number of values read and on error or end of the file it returns EOF or -1.

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

int main()
{
     FILE *fp;
     char name[50];
     int roll_no, chars;
     float marks;

     fp = fopen("records.txt", "r");

     if(fp == NULL)
     {
          printf("Error opening file\n");
          exit(1);
     }

     printf("Testing fscanf() function: \n\n");
     printf("Name:\t\tRoll\t\tMarks\n");

     while( fscanf(fp, "Name: %s\t\tRoll no: %d\t\tMarks: %f\n"
```

```
23      , name, &roll_no, &marks) != EOF )
24      {
25           printf("%s\t\t%d\t\t%.2f\n", name, roll_no ,marks);
26      }
27
28      fclose(fp);
29      return 0;
30 }
```

fwrite() Function in C:

Binary Input and Output:

✓ fread() and fwrite() functions are commonly used to read and write binary data to and from the file respectively. Although we can also use them with text mode too.

fwrite() function:

```
Syntax: size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp); \\

The fwrite() function writes the data specified by the void pointer ptr to the file
     .

ptr: it points to the block of memory which contains the data items to be written.

size: It specifies the number of bytes of each item to be written.

n: It is the number of items to be written.

fp: It is a pointer to the file where data items will be written.
```

✓ On success, it returns the count of the number of items successfully written to the file.

✓ On error, it returns a number less than n. Notice that two arguments (size and n) and return value of fwrite() are of type size_t which on the most system is unsigned int.

Example 1: Writing a variable

```
float *f = 100.13;

fwrite(&p, sizeof(f), 1, fp);
```

This writes the value of variable f to the file.

Example 2: Writing an array

```
int arr[3] =
{
     101, 203, 303
}
;

fwrite(arr, sizeof(arr), 1, fp);
```

This writes the entire array into the file.

Example 3: Writing some elements of array:

```
int arr[3] =
{
     101, 203, 303
}
;

fwrite(arr, sizeof(int), 2, fp);
```

This writes only the first two elements of the array into the file.

Example 4: Writing structure:

```
struct student
{
     char name[10];
     int roll;
     float marks;
}
;

struct student student_1 =
{
     "Tina", 12, 88.123
}
;

fwrite(&student_1, sizeof(student_1), 1, fp);
```

This writes contents of variable student_1 into the file.

Example 5: Writing array of structure:

```c
struct student
{
    char name[10];
    int roll;
    float marks;
}
;

struct student students[3] =
{

    {
        "Tina", 12, 88.123
    }
    ,

    {
        "Jack", 34, 71.182
    }
    ,

    {
```

```
            "May", 12, 93.713
        }

}
;

fwrite(students, sizeof(students), 1, fp);
```

This writes the whole array students into the file.

✓ Let's say we don't' want to write all elements of the array into the file, instead, we want is to write only 0th and 1st element of the array into the file.

```
fwrite(students, sizeof(struct student), 2, fp);
```

Now you have understood how fwrite() function works. Let's create a program using fwrite() function.

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

struct employee
{
     char name[50];
     char designation[50];
     int age;
     float salary
}
 employee;

int main()
{
     int n, i, chars;
     FILE *fp;

     fp = fopen("employee.txt", "wb");

     if(fp == NULL)
     {
          printf("Error opening file\n");
```

```
23          exit(1);
24      }
25
26      printf("Testing fwrite() function: \n\n");
27
28      printf("Enter the number of records you want to enter: ");
29      scanf("%d", &n);
30
31      for(i = 0; i < n; i++)
32      {
33          printf("\nEnter details of employee %d \n", i + 1);
34
35          fflush(stdin);
36
37          printf("Name: ");
38          gets(employee.name);
39
40          printf("Designation: ");
41          gets(employee.designation);
42
43          printf("Age: ");
44          scanf("%d", &employee.age);
45
```

```
46          printf("Salary: ");
47          scanf("%f", &employee.salary);
48
49          chars = fwrite(&employee, sizeof(employee), 1, fp);
50          printf("Number of items written to the file: %d\n", chars);
51      }
52
53      fclose(fp);
54      return 0;
55  }
```

fread() Function in C:

✓ The fread() function is the complementary of fwrite() function. fread() function is commonly used to read binary data. It accepts the same arguments as fwrite() function does.

The syntax of fread() function is as follows:

Syntax:

```
size_t fread(void *ptr, size_t size, size_t n, FILE *fp); \\
```

✓ The ptr is the starting address of the memory block where data will be stored after reading from the file.

✓ The function reads n items from the file where each item occupies the number of bytes specified in the second argument.

✓ On success, it reads n items from the file and returns n. On error or end of the file, it returns a number less than n.

Example 1: Reading a float value from the file

```
int val;

fread(&val, sizeof(int), 1, fp);
```

This reads a float value from the file and stores it in the variable val.

Example 2: Reading an array from the file

```
int arr[10];

fread(arr, sizeof(arr), 1, fp);
```

This reads an array of 10 integers from the file and stores it in the variable arr.

Example 3: Reading the first 5 elements of an array

```
int arr[10];

fread(arr, sizeof(int), 5, fp);
```

This reads 5 integers from the file and stores it in the variable arr.

Example 4: Reading the first 5 elements of an array:

```
int arr[10];

fread(arr, sizeof(int), 5, fp);
```

This reads 5 integers from the file and stores it in the variable arr.

Example 5: Reading the structure variable:

```
struct student
{
     char name[10];
     int roll;
     float marks;
}
;

struct student student_1;

fread(&student_1, sizeof(student_1), 1, fp);
```

This reads the contents of a structure variable from the file and stores it in the variable student_1.

Example 6: Reading an array of structure:

```
struct student
{
     char name[10];
     int roll;
     float marks;
}
;

struct student arr_student[100];

fread(&arr_student, sizeof(struct student), 10, fp);
```

This reads first 10 elements of type struct student from the file and stores them in the variable arr_student.

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

struct employee
{
     char name[50];
     char designation[50];
     int age;
     float salary
}
 emp;

int main()
{
     FILE *fp;
     fp = fopen("employee.txt", "rb");

     if(fp == NULL)
     {
          printf("Error opening file\n");
          exit(1);
     }
```

```
23
24      printf("Testing fread() function: \n\n");
25
26      while( fread(&emp, sizeof(emp), 1, fp) == 1 )
27      {
28           printf("Name: %s \n", emp.name);
29           printf("Designation: %s \n", emp.designation);
30           printf("Age: %d \n", emp.age);
31           printf("Salary: %.2f \n\n", emp.salary);
32      }
33
34      fclose(fp);
35      return 0;
36  }
```

fseek()

The C library function int fseek(FILE *stream, long int offset, int whence) sets the file position of the stream to the given offset.

```
1 int fseek(FILE *stream, long int offset, int whence)
```

Parameters:

stream - This is the pointer to a FILE object that identifies the stream.

offset - This is the number of bytes to offset from whence.

whence - This is the position from where offset is added. It is specified by one of the following constants

Sr.No.	Constant & Description
1	SEEK_SET Beginning of file
2	SEEK_CUR Current position of the file pointer
3	SEEK_END End of file

Return Value: This function returns zero if successful, or else it returns a non-zero value.

```c
#include <stdio.h>

int main ()
{
     FILE *fp;

     fp = fopen("file.txt","w+");
     fputs("This is tutorialspoint.com", fp);

     fseek( fp, 7, SEEK_SET );
     fputs(" C Programming Language", fp);
     fclose(fp);

     return(0);
}
```

```c
#include <stdio.h>

int main ()
{
     FILE *fp;
     int c;

     fp = fopen("file.txt","r");
     while(1)
     {
            c = fgetc(fp);
            if( feof(fp) )
          {
               break;

          }
            printf("%c", c);
     }
     fclose(fp);
     return(0);
}
```

```c
// C Program to demonstrate the use of fseek()
#include <stdio.h>
int main()
{
      FILE *fp;
      fp = fopen("test.txt", "r");
      // Moving pointer to end
      fseek(fp, 0, SEEK_END);
      // Printing position of pointer
      printf("%ld", ftell(fp));
      return 0;
}
```

```c
#include <stdio.h>
int main ()
{
     FILE *fp;
     char data[60];
     fp = fopen ("test.c","w");
     fputs("Fresh2refresh.com is a programming tutorial website", fp);
     fgets ( data, 60, fp );
     printf(Before fseek - %s", data);

     //To set file pointet to 21th byteor character in the file
     fseek(fp, 21, SEEK_SET);
     fflush(data);
     fgets ( data, 60, fp );
     printf("After SEEK_SET to 21 - %s", data);

     //To find backward 10 bytes from current position
     fseek(fp, -10, SEEK_CUR);
     fflush(data);
     fgets ( data, 60, fp );
     printf("After SEEK_CUR to -10 - %s", data);

```

```
23      //To find 7th byte before the end of file
24      fseek(fp, -7, SEEK_END);
25      fflush(data);
26      fgets ( data, 60, fp );
27      printf("After SEEK_END to -7 - %s", data);
28
29      //To set file pointer to the beginning of the file
30      fseek(fp, 0, SEEK_SET); // We can use rewind(fp); also
31
32       length = ftell(fp); // Cursor position is now at the end of file
33      /* You can use fseek(fp, 0, SEEK_END); also to move the
34      cursor to the end of the file */
35      rewind (fp); // It will move cursor position to the beginning of the file
36
37
38      fclose(fp);
39      return 0;
40  }
```