

File Management In C

File is a set of records that can be accessed through the set of library functions.

Stream means reading and writing of data. The streams are designed to allow the user to access the files efficiently. A stream is a file or physical device like keyboard, printer and monitor. The FILE object uses these devices.

The FILE object contains all the information about stream like current position, pointer to any buffer, error and EOF (end of file).

There are two different types of data files called **stream oriented** (standard) data files, and **system-oriented** (or low-level) data files.

Stream-oriented data files can be subdivided into two categories.

The first category are **text files** consisting of consecutive characters. These characters can be interpreted as individual data items or as components of strings or numbers.

The second category are referred to as **unformatted data files**, organizes data into blocks containing contiguous bytes of information. These blocks represent more complex data structures, such as arrays and structures.

System-oriented data files are more closely related to the computer's operating system than stream-oriented data files. They are more complicated to work with, though they are efficient for certain kind of applications.

Sequential access and random access to files

Sequential access to a data file means that the computer system reads or writes information to the file sequentially, starting from the beginning of the file and proceeding step by step. If we want to read the last record of the file we need to read all the records before that record. It takes more time. For example if we desired to access the 10 th record then the first 9 records should be read sequentially for reaching to the 10 th record.

Random access to a file means that the computer system can read or write information anywhere in the data file. This type of operation is also called “**Direct Access**” because the computer system knows where the data is stored and hence directly reads the data.

Sequential access has advantages when the data is accessed in the same order all the time. It is faster than random access.

Random access has the advantage that we can search through and find the data more easily (using indexing).

A sequential file is like a video tape (have to watch movie in a sequence) and a random access file is like a DVD (where we can skip to different chapters).

File operations

A file is a place on the disk where a group of **related data** are stored.

The basic file operations include

- naming a file
- opening a file
- reading data from a file
- writing data to a file
- closing a file

To perform the above operations on a file we have number of file handling function in <stdio.h> header file.

The C programming language supports many more file handling functions that are available in standard library. These functions are listed in the below table.

Function	Operation
fopen()	Creates a new file for read/write operation
fclose()	Closes a file associated with file pointer
closeall()	Closes all opened files with file pointer
fgetc()	Reads the character from current pointer position and advances the pointer to next character.
getc()	Same as fgetc()
fputc()	Writes character one by one to a file
putc()	Same as fputc()

gets()	Reads string from the file
puts()	Writes string to the file
fgetw()	Reads integer from the file
fputw()	Writes integer to the file
fprintf()	Writes all types of data values to the file.
fscanf()	Reads all types of data values from a file
fseek()	Sets the pointer position anywhere in the file
ftell()	Return current position
feof()	Detects the end of the file
rewind()	Set pointer at the beginning
ferror()	Reports error occurred while read/ write operation
perror()	Prints compiler error messages along with user defined messages.
rename()	Changes the name of the file.

Defining and Opening a file

To store data in a file in secondary memory, the following must be specified.

1. Filename
2. Data structure
3. Purpose

Filename is a string of characters that make up a valid file name for the operating system. It may contain two parts, a primary name and an optional period with extension

Eg: student.c, Text.out

Data structure of a file is defined as FILE in the library of standard I/O function definitions. Therefore, all files should be declared as type FILE before they are used. FILE is a defined data type.

We must also specify the purpose, that is, what we want to do with the file, write data to the file or read the already existing data.

Following is the general format for declaring and opening a file

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

The first statement declares the variable fp as a pointer to the data type FILE. FILE is a structure defined in the I/O library. The second statement

opens the file named filename and assigns an identifier to the file pointer fp. It also specifies the purpose of opening this file. **Mode** can be one of the following.

r open the file for reading only
w open the file for writing only
a open the file for appending(adding) data to it

1. When the mode is writing, a file with the specified name is created if the file does not exist. The contents are deleted, if the file already exists
2. When the mode is reading, if the file does not exist, fopen() returns NULL
3. If the mode is appending, the file is created if it does not exist.

r+ open an exiting file for both reading and writing
w+ open a new file for both reading and writing.

a+ open an exiting file for both reading and appending.

Example

```
fp = fopen("newpgm.txt", "w");
```

If the file newpgm.txt does not exist, the function creates a new file named newpgm.txt and opens it for writing as per the mode "w"

Closing a file

The file should be closed after reading or writing. Closing a file is performed using the fclose() function.

Following is the general format for closing a file.

```
fclose(fp);
```

fp is the file pointer associated with the file to be closed.

Once the file is closed, we can no longer read from it unless it is reopened. On closing the file, the buffer associated with the file is removed from memory.

Reading from a file

To read the file's contents from memory, there exists functions like `fgetc()`, `fgets()` and `fscanf()`.

`fgetc()` is used to read a character from a file. It reads a single character at a time.

`fgets()` reads lines from a file into character arrays.

Declaration of `fgetc()`

```
int fgetc(FILE *stream)
```

`stream` – This is the pointer to a `FILE` object that identifies the stream on which the operation is to be performed.

Example program using `fgetc()` to read from a file

To run the following program, create a file "sample.txt" with the content "I am learning file operations in C". The objective of the program is to read from the file and print data on screen

Algorithm is as given below

Start

- 1) Open the file "sample.txt" in read mode.
- 2) Check if any error occurs in file opening.
- 3) Read each character from the file one by one using `fgetc()` till **end of file**(EOF) is reached
- 4) Print each character on the screen.
- 5) Close the file for reading.

Stop.

Program

```

#include <stdio.h>

int main () {
    FILE *fp;
    char ch;


    /* opening file for reading */
    fp = fopen("sample.txt" , "r");

    if(fp == NULL) {
        perror("Error opening file");
        return(-1);
    }
    while( ch!=EOF ) {
        /* reading character from file one by one */
        ch = fgetc(fp);
        printf("%c", ch);
    }
    fclose(fp);

    return(0);
}

```

Output

 "F:\Document\c programs\ex_getc.exe"

```

I am learning file operations in C.
Process returned 0 (0x0)   execution time : 0.890 s
Press any key to continue.

```

Declaration of fgetc()

```
char *fgetc(char *str, int n, FILE *stream)
```

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

Example program using fgets() to read from a file

To run the following program, create a file "file1.txt " with the content "I am doing C Programming". The objective of the program is to read from the file and print data on screen

Algorithm is as given below

Start

- 1) Open the file "file1.txt" in read mode.
- 2) Check if any error occurs in file opening.
- 3) Read the entire string from the file using fgets().
str is the string to be read with maximum size of 60.
Usage is fgets(str, length of string, file pointer)
- 4) Print the string on the screen using puts() function.
- 5) Close the file for reading.

Stop.

```
#include <stdio.h>

int main () {
    FILE *fp;
    char str[60];

    /* opening file for reading */
    fp = fopen("file1.txt" , "r");
    if(fp == NULL) {
        perror("Error opening file");
        return(-1);
    }
    if( fgets (str, 60, fp)!=NULL ) {
        /* writing content to stdout */
        puts(str);
    }
    fclose(fp);

    return(0);
}
```

Output

Use while statement to print multiple lines

Declaration of fscanf()

```
fscanf(FILE * stream, "control string", arg1,arg2,...argn)
```

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

control string specifies the field format in which data is entered and the arguments arg1,arg2....argn specify the address of locations where data is stored.

Example program using fscanf() to read from a file

The objective of the program is to create a file "file.txt" with content "We are in 2020". Then read from the file and print data on screen

Algorithm is as given below

Start

- 1) Open the file "file.txt" for reading and writing ("w+" mode).
- 2) Write the string "We are in 2020" to the file using fputs().
- 3) Use rewind() to move the file pointer to beginning of file.
- 4) Read data from the file using fscanf(), %s is used to read string, %d is used to read integer values.
- 5) Print the string and integer values on the screen.
- 6) Close the file.

Stop.


```

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

int main () {
    char str1[10], str2[10], str3[10];
    int year;
    FILE * fp;
    /* w+ mode is used to open the file for both reading and writing*/
    fp = fopen ("file.txt", "w+");
    fputs("We are in 2020", fp);

    rewind(fp);
    fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);


    printf("Read String1 - %s\n", str1 );
    printf("Read String2 - %s\n", str2 );
    printf("Read String3 - %s\n", str3 );
    printf("Read Integer - %d\n", year );

    fclose(fp);

    return(0);
}

```

Output

 "F:\Document\c programs\ex_fscanf.exe"

```

Read String1 - We
Read String2 - are
Read String3 - in
Read Integer - 2020

Process returned 0 (0x0)   execution time : 4.459 s
Press any key to continue.

```

Writing to a file

fputc() , fputs() and fprintf() functions are used to write to a file.

Declaration of fputc()

```
int fputc(int char, FILE *stream)
```

- **char** – This is the character to be written. This is passed as its int promotion.
- **stream** – This is the pointer to a FILE object that identifies the stream where the character is to be written

Example program using fputc() to write to a file

The objective of the program is to create a file “new.txt” containing capital letters from A to Z.

Algorithm is as given below

Start

- 1) Open the file “new.txt” for writing.
- 2) Check if any error occurs in file opening.
- 3) Write the ASCII equivalent of characters A to Z to the file using fputc(). ASCII equivalent of A is 65 and Z is 90.
- 5) All the characters from A to Z are written on the file new.txt
- 6) Close the file.

Stop.

```
#include <stdio.h>

int main () {
    FILE *fp;
    char ch;

    /* opening file for reading */
    fp = fopen("new.txt" , "w");

    if(fp == NULL) {
        perror("Error opening file");
        return(-1);
    }
    for( ch = 65 ; ch <= 90; ch++ ) {
        fputc(ch, fp);
    }
    fclose(fp);

    return(0);
}
```

Output

A file new.txt is created with the following content.
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Declaration of fprintf()

```
fprintf(FILE * stream, "control string", arg1,arg2,...argn)
```

Control string consists of three types of items

- Characters that will be printed on the screen as they appear
- Format specifications that define the output format for display of each item
- Escape sequence characters such as \n,\t and \b.

Example program using fprintf() to write to a file

The objective of the program is to create a file “file.txt” containing the string “We are in 2020”

Algorithm is as given below

Start

- 1) Open the file “file.txt” for writing.
- 2) Write the string “We are in 2020” using fprintf(). %s is used to write a string, %d is used to write integer values.
- 3) Close the file.

Stop.

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

int main () {
    FILE * fp;

    fp = fopen ("file.txt", "w");
    fprintf(fp, "%s %s %s %d", "We", "are", "in", 2020);

    fclose(fp);

    return(0);
}
```

Output

A file named **file.txt** is created with the following content.
We are in 2020.

Declaration of fputs()

```
int fputs(const char *str, FILE *stream)
```

Example program using fputs() to write to a file

The objective of the program is to create a file “file1.txt” containing two strings “This is C programming” and “I am learning file management in C”

Algorithm is as given below

Start

- 1) Open the file "file1.txt" for writing.
- 2) Write the string "This is C programming" followed by "I am learning file management in C" using fputs().
- 3) Close the file.

Stop.

```
#include <stdio.h>

int main () {
    FILE *fp;

    fp = fopen("file1.txt", "w");

    fputs("This is C programming.\n", fp);
    fputs("I am learning file management in C.", fp);

    fclose(fp);

    return(0);
}
```

Output

A file named **file1.txt** is created with the following content.

This is C programming.

I am learning file management in C.

Write a program to read name and marks of n number of students and store them in a file.

Start

- 1) Open the file "file1.txt" for writing.
- 2) Read number of students, n
- 3) Read name and marks, store them into file using a loop
- 4) Close the file.

Stop.

rewind(), fseek(), ftell(), feof(), fread(), fwrite() functions

a) rewind()

Declaration

```
void rewind(FILE *stream)
```

stream – This is the pointer to a FILE object that identifies the stream

rewind() function is used to move file pointer position to the beginning of the file.

In a C program, rewind is used as
rewind(fp);

Example program to read the contents of a file and write the contents to the screen twice.

Algorithm is as given below

Start

- 1) Open the file "sample.txt" for reading.
- 2) Check if any error occurs in file opening.
- 3) Read the characters from the file one by one using fgetc() and print the characters on the screen.
- 4) Use rewind() to move the file pointer to beginning of file.
- 5) Again repeat reading the characters from the file one by one using fgetc() and print the characters on the screen .
- 6) The same content is repeatedly written two times on the screen.
- 7) Close the file.

Stop.

```

int main () {
    FILE *fp;
    char ch;
    /* opening file for reading */
    fp = fopen("sample.txt" , "r");

    if(fp == NULL) {
        perror("Error opening file");
        return(-1);
    }
    while( ch!=EOF ) {
        /* reading character from file one by one */
        ch = fgetc(fp);
        printf("%c", ch);
    }

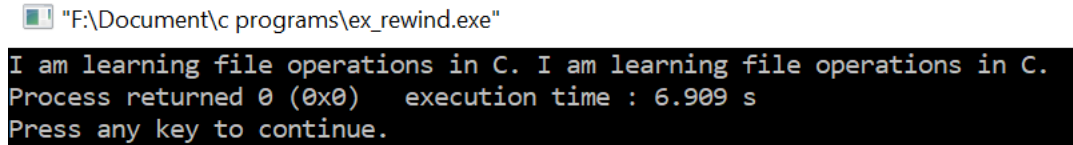
    rewind(fp);

    while((ch=fgetc(fp)) !=EOF) {
        /* writing character to the screen */
        printf("%c", ch);
    }

    fclose(fp);
    return(0);
}

```

Output



```

F:\Document\c programs\ex_rewind.exe
I am learning file operations in C. I am learning file operations in C.
Process returned 0 (0x0) execution time : 6.909 s
Press any key to continue.

```

b) fseek()

Declaration

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

fseek() function is used to move file pointer position to the given location. It is used to write data into file at desired location.

The function fseek() accept 3 arguments

- **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

SEEK_SET – moves file pointer to the beginning of file

SEEK_CUR – moves file pointer to given location

SEEK_END – moves file pointer to the end of file

Example program to read the contents of a file and write the contents to the screen twice.

Example program to overwrite the contents of a file from a specific location. The objective of the program is to overwrite the contents of the file that contains "This is Singapore" with "This is C Programming Language".

Algorithm is as given below

Start

- 1) Open the file "myfile.txt" for writing.
- 2) Check if any error occurs in file opening.
- 3) Write the string "This is Singapore" to the file using fputs()
- 4) Using fseek(), move the file pointer from the beginning of the file by 8 locations. This will set the pointer position to the letter "S" in Singapore.
- 5) Overwrite the contents from the above position with the string "C Programming language" using fputs().
- 6) The original content changes to "This is C Programming Language".
- 7) Close the file.

Stop.

```

#include <stdio.h>

int main () {
    FILE *fp;

    /* opening file for reading */
    fp = fopen("myfile.txt" "w");
    if(fp == NULL) {
        perror("Error opening file");
        return(-1);
    }
    fputs ("This is Singapore", fp);

    fseek(fp,8,SEEK_SET);

    fputs ("C Programming Language", fp);

    fclose(fp);

    return(0);
}

```

Output (Content of myfile.txt)

This is C Programming Language

c) ftell()

Declaration

```
long int ftell(FILE *stream)
```

stream – This is the pointer to a FILE object that identifies the stream

ftell() function is used to get current position of the file pointer. ftell() can be used to get the total size of a file after moving the file pointer at the end of file.

SEEK_END can be used with fseek() to move the file pointer to the end of file.

Example program to find the size of a file in bytes.

To run the following program, create a file “file1.txt “ with the content “We are in 2022”

Algorithm is as given below

Start

1) Open the file “file.txt” for reading.

- 2) Check if any error occurs in file opening.
 - 3) Using fseek(), move the file pointer to the end of the file.
 - 4) When file pointer is positioned at the end of file, ftell() function returns the current position, which is the size of the file in bytes.
 - 5) The return value from ftell() gives the size of file in bytes.
 - 6) Close the file.
- Stop.

```
#include <stdio.h>

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

    fp = fopen("file.txt", "r");
    if( fp == NULL ) {
        perror ("Error opening file");
        return(-1);
    }
    fseek(fp, 0, SEEK_END);

    len = ftell(fp);
    fclose(fp);

    printf("Total size of file.txt = %d bytes\n", len);

    return(0);
}
```

file.txt contains the content “We are in 2020”

Output

```
"F:\Document\c programs\ex_ftell.exe"
Total size of file.txt = 14 bytes
Process returned 0 (0x0) execution time : 4.969 s
Press any key to continue.
```

d) feof()

Declaration

```
int feof(FILE *stream)
```

stream – This is the pointer to a FILE object that identifies the stream

feof() function is used to find the end of file.

Example program to read the contents of a file and print to the screen till end of file is reached.

To run the following program, create a file “file.txt “ with the content “**We are in 2022**”

Algorithm is as given below

Start

- 1) Open the file “file.txt” for reading.
- 2) Check if any error occurs in file opening.
- 3) Using fgetc() function, read each character from the file.
- 4) Check if end of file is reached using feof(). If yes, then break from the loop. Go to step (7)
- 5) If no, print the character on screen
- 6) Repeat from step (3)
- 7) Print the statement “Closing the file file.txt as end of file is reached” on the screen.
- 8) Close the file.

Stop.

```

#include <stdio.h>

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

    fp = fopen("file.txt", "r");
    if(fp == NULL) {
        perror("Error in opening file");
        return(-1);
    }

    while(1) {
        c = fgetc(fp);
        if( feof(fp) ) {
            break ;
        }
        printf("%c", c);

    }

    printf("\nClosing the file file.txt as end of file is reached");
    fclose(fp);

    return(0);
}

```

file.txt contains the content “We are in 2020”

Output

```

F:\Document\c programs\ex_feof.exe
We are in 2020
Closing the file file.txt as end of file is reached
Process returned 0 (0x0) execution time : 1.443 s
Press any key to continue.

```

we
are
in
2022

e) fread() and fwrite()

For writing to a file, fprintf(), fputs() or fputc() can be used. But there will be difficulty in writing the contents of a structure. fwrite() and fread() make the task easier when we want to write and read blocks of data.

The fwrite() function is used to write records (sequence of bytes) to the file. A record may be an array or a structure.

Declaration

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
```

The function accept 4 arguments.

ptr - This is pointer to array of elements to be written.

size - This is the size in bytes of each element to be written.

nmemb - This is the number of elements, each one with a size of

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

fread()

The fread() function is used to records (sequence of bytes) to the file. A record may be an array or as structure.

Declaration

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
```

ptr - This is pointer to a block of memory with a minimum size of nmemb bytes.

size - This is the size in bytes of each element to be written.

nmemb - This is the number of elements, each one with a size of size bytes.

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

Following is a programming example illustrating the use of fread() and fwrite()

Program to write details of two students to a file, then read from the file to print the data on the screen

Algorithm is as given below

Start

- 1) Create a structure Student to declare variables inp1 and inp2. Each variable contains roll_no and name.
- 2) Open file c1.txt to write.
- 3) Check if any error occurs in file opening.

- 4) Initialize the structure variables inp1 and inp2 with data.
- 5) If file open successfully, write each structure inp1 and inp2 using fwrite(). The 4 arguments passed to fwrite() are address of structure variable, size of the structure variable, number of structure variables to write, file pointer
- 6) Check the return value of fwrite() to verify if data written Successfully.
- 6) Close the file for writing.
- 7) Declare a structure variable inp to read from the file.
- 7) Open the file to read.
- 8) Check if any error occurs in file opening.
- 9) If file open successfully, read from the file using fread().

The 4 arguments passed to fread() are same as that of fwrite().

address of structure variable, size of the structure variable, number of structure variables to read, file pointer

- 10) Check if any error occurs in reading the file.
- 11) Print the data on the screen.
- 12) Close the file for reading.

Stop.

Program

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Student {
    int roll_no;
    char name[20];
};
int main () {
    FILE *of;
    of= fopen ("cl.txt", "w");
    if (of == NULL) {
        fprintf(stderr, "\nError to open the file\n");
        exit (1);
    }
    struct Student inp1 = {1, "Ram"};
    struct Student inp2 = {2, "Shyam"};
    fwrite (&inp1, sizeof(struct Student), 1, of);
    fwrite (&inp2, sizeof(struct Student), 1, of);
    if(fwrite != 0)
        printf("Contents to file written successfully !\n");
    else
        printf("Error writing file !\n");
    fclose (of);
    FILE *inf;
    struct Student inp;
    inf = fopen ("cl.txt", "r");
    if (inf == NULL) {
        fprintf(stderr, "\nError to open the file\n");
        exit (1);
    }
    while(fread(&inp, sizeof(struct Student), 1, inf))
        printf ("roll_no = %d name = %s\n", inp.roll_no, inp.name);
    fclose (inf);
}

```

Output

```

F:\Document\c programs\ex_fread_write.exe
Contents to file written successfully !
roll_no = 1 name = Ram
roll_no = 2 name = Shyam

Process returned 0 (0x0)   execution time : 3.071 s
Press any key to continue.

```

Handling errors during I/O operations

While performing read or write operations sometimes we do not get the result successfully. The reason may be that the attempt of reading or writing the operation may not be correct.

The C-Language provides the following standard library functions to handle these type of errors during I/O operations.

`ferror()`

It is used to detect any error that might occur during read/write operation on a file. It returns '0' when the attempt is successful otherwise non-zero in case of failure.

`perror()`

It is a standard library function which prints the error messages specified by the compiler. The following program illustrates the above two functions.....

Segmentation fault(SIGSEGV) and **Bus error(SIGBUS)** are signals generated when serious program error is detected by the operating system and there is no way the program could continue to execute because of these errors.

1) Segmentation Fault (also known as SIGSEGV and is usually signal 11) occur when the program tries to write/read outside the memory allocated for it or when writing memory which can only be read. In other words when the program tries to access the memory to which it doesn't have access to. SIGSEGV is abbreviation for "Segmentation Violation".

Few cases where SIGSEGV signal generated are as follows,

- > Using uninitialized pointer
- > De-referencing a NULL pointer
- > Trying to access memory that the program doesn't own (eg. trying to access an array element out of array bounds).
- > Trying to access memory which is already de-allocated (trying to use dangling pointers).

2) Bus Error (also known as SIGBUS and is usually signal 10) occur when a process is trying to access memory that the CPU cannot physically address. In other words the memory tried to access by the program is not a valid memory address. It caused due to alignment issues with the CPU (eg. trying

to read a long from an address which isn't a multiple of 4). SIGBUS is abbreviation for "Bus Error".

The main difference between Segmentation Fault and Bus Error is that Segmentation Fault indicates an invalid access to a valid memory, while Bus Error indicates an access to an invalid address.

Write a program to count the number of characters, spaces, tab, new lines in a file.

```
#include<stdio.h>
main( )
{
FILE *fp;
char ch;
int lines=0,tab=0,space=0, characters=0;
fp=fopen("text.c","r");
while(1)
/* infinite loop*/
{
ch = fgetc(fp);
if(ch==EOF)
break;
characters++;
if(ch== 32)
space++;
if(ch== '\n')
lines++;
if(ch== '\t ')
tab++;
}
fclose(fp);
printf("number of lines = %d\ntabs = %d\ncharacters = %d\nspaces = %d",lines,characters,spaces);
getch( );
}
```

Write a program to receive some strings from keyboard and print it to a file

```
#include<stdio.h>
main( )
{
```



```

FILE *fp;
char s[50];
fp=fopen("text.c","w");
/*opening file*/
if(fp==NULL)
/*to check opening errors*/
{
puts("file opening error");
exit();
}
printf("Enter some text");
while(strlen(gets(s))>0)
{
fputs(s,fp);
fputs("\n",fp);
}
fclose(fp);
}

```

Write a program to read all the strings from file and print it on screen

```

#include<stdio.h>
main( )
{
FILE *fp;
char s[50];
fp=fopen("text.c","r");
/*opening file*/
if(fp==NULL)
/*to check opening errors*/
{
puts("file opening error");
exit();
}
while(fgets(s,49,fp)!= NULL)
printf("%s",s);
fclose(fp);
getch( );
}

```

Write a c program to print contents of a file in reverse order.

```
int pos=0;
fseek(fp,0,SEEK,END)    //moves the file pointer to the end.
pos=ftell(fp);

while(i<pos)
{
    fseek(fp,-i,SEEK,END)
    printf("%c",fgetc(fp));
    i++;
}

fclose(fp);
```