

In C/C++ programming, **FILE** is a place on your physical disk where information is stored.

Why File are needed?

- When a program is terminated, the entire data is lost. Storing in a **FILE** will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a **FILE** containing all the data, you can easily access the contents of the **FILE** using few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of files

When dealing with FILES, there are two types of FILES you should know about:

1. Text FILES
2. Binary FILES

1. Text files

Text FILES are the normal .txt FILES that you can easily create using Notepad or any simple text editors .When you open those FILES, you'll see all the contents within the FILE as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

Binary FILES are mostly the .bin FILES in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold higher amount of data, are not readable easily and provides a better security than text FILES.

FILE Operations

In C/C++, you can perform four major operations on the FILE, either text or binary:

1. Creating a new FILE
2. Opening an existing FILE
3. Closing a FILE
4. Reading from and writing information to a FILE

In C/C++, A binary FILE is a series of characters or bytes for C/C++, attaches no special meaning. Any structure to the data is determined by the application that reads from or writes to the FILE. A text FILE, in contrast, is assumed to have only printable characters and a small set of control or formatting characters. The Formatted characters are the binary equivalentents of the escape sequences listed in following table.

Escape Sequences	Meaning	Decimal Code	Octal	Hexadecimal
<code>\r</code>	Line Feed	10	012	0A or <code>\x0A</code>
<code>\n</code>	Carriage return	13	015	00 or <code>\x0F</code>
<code>\t</code>	Horizontal tab	9	011	09 or <code>\x09</code>
<code>\v</code>	Vertical tab	11	013	0B or <code>\x0B</code>
<code>\f</code>	Form feed	12	012	0C or <code>\x0C</code>

Different operating system use different escape sequences to identify the end of a line. UNIX/Linux platforms use the `\n` carriage return escape sequence, and Macintosh application usually use the `\r` line feed escape sequence, and windows operating system use the `\n` carriage return escape sequence followed by the `\r` line feed escape sequence.

- Escape sequence "`\n`" use for **end-of-line** in Unix/Linux Operating System.
- Escape sequences "`\n\r`" combine use for **end-of-line** in Windows Operating System. □ Escape sequence "`\r`" use for **end-of-line** in Macintosh OS.

Working with files

When working with FILES in C, you need to declare a pointer of type FILE. This declaration is needed for communication between the FILE and program.

Syntax: `FILE *fp`

C/C++ provides a number of functions that helps to perform basic FILE operations. Following are the functions,

Function	Description
<code>fopen()</code>	create a new FILE or open a existing FILE
<code>fclose()</code>	closes a FILE
<code>getc()</code>	reads a character from a FILE
<code>putc()</code>	writes a character to a FILE
<code>fscanf()</code>	reads a set of data from a FILE
<code>fprintf()</code>	writes a set of data to a FILE
<code>fgets()</code>	reads string specific length from a FILE
<code>fputs()</code>	writes string to a FILE
<code>fseek()</code>	set the position to desire point
<code>ftell()</code>	gives current position in the FILE
<code>rewind()</code>	set the position to the beginning point

Opening a FILE: (for creation and edit)

To open a **FILE** you need to use the **fopen** function, which returns a FILE pointer. Once you've opened a FILE, you can use the FILE pointer to let the compiler perform input and output functions on the **FILE**.

Syntax: `*fp = FILE *fopen(const char *FILENAME, const char *mode);`

Mode	Description
r	opens a text FILE in reading mode
w	Opens or create a text FILE in writing mode.
a	opens a text FILE in append mode
r+	opens a text FILE in both reading and writing mode
w+	opens a text FILE in both reading and writing mode
a+	opens a text FILE in both reading and writing mode
rb	opens a binary FILE in reading mode
wb	opens or create a binary FILE in writing mode
ab	opens a binary FILE in append mode
rb+	opens a binary FILE in both reading and writing mode
wb+	opens a binary FILE in both reading and writing mode
ab+	opens a binary FILE in both reading and writing mode

Example:

```
#include <stdio.>
void main()
{
    FILE *fp;
    fp=fopen("c:\\test.txt", "r");
}
```

This code will open test.txt for reading in text mode. To open a **FILE** in a binary mode you must write "rb" to the mode string.

Closing a FILE:

To close a FILE, use the **fclose()** function.

Syntax: `int fclose(FILE *fp);`

Here `fclose()` function closes the FILE and returns zero on success, or EOF if there is an error in closing the FILE. This EOF is a constant defined in the header FILE *stdio.h*.

Write one character into mytextFILE.txt FILE.

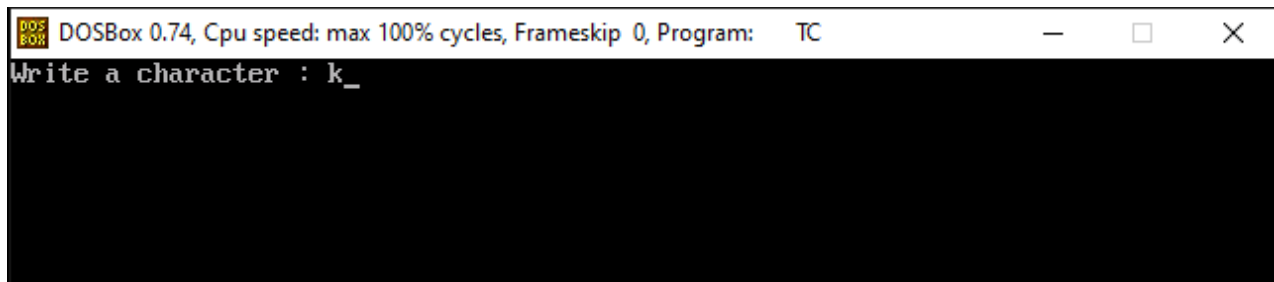
`putc()` is simplest functions used to write individual characters to a FILE.

Syntax: `fputc(character,FILEpointer)`

Example:

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

    clrscr();
    fp=fopen("mytextfile.txt","w");
    printf("Write a character : ");
    ch=getche();
    putc(ch,fp);
    fclose(fp);
}
```



In this program you will see create one FILE name is *mytextfile.txt* in source folder and one character k put in this FILE.

Read one character from mytextfile.txt FILE.

`getc()` is simplest functions used to read individual characters to a FILE.

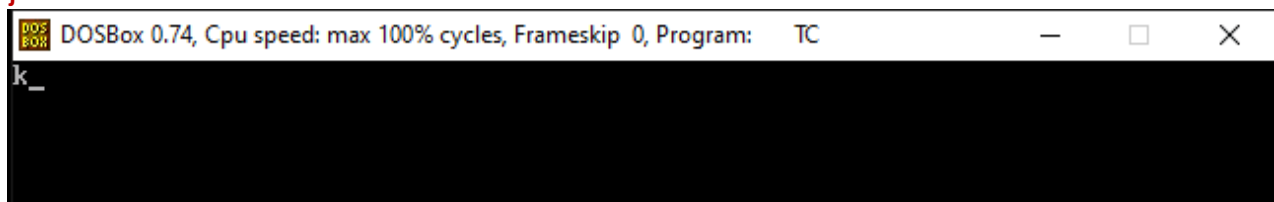
Syntax: `variable = getc(Filepointer)`

Example:

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

    clrscr();
    fp=fopen("mytextfile.txt","r");

    ch=getc(fp);
    printf("%c",ch);
    fclose(fp);
getche();
}
```

**Input/Output operation on FILE**

In the above table we have discussed about various FILE I/O functions to perform reading and writing on FILE. `getc()` and `putc()` are simplest functions used to read and write individual characters to a FILE.

Example:

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

    clrscr();
    fp = fopen("one.txt", "w");
    printf("Enter Character data : ");

    while( (ch = getchar()) != 13)
    {
        putc(ch,fp);
    }

    fclose(fp);
}
```

```

fp = fopen("one.txt", "r");
printf("\n\n Read one by one character from FILE name is one.txt \n");
while( (ch = getc(fp) != EOF)
    printf("%c",ch);

fclose(fp);
}

```

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter characters data : My name is Sir Masood

Read one by one charcerter from file name is one.txt
My name is Sir Masood_

```

(Reading and Writing from FILE using fprintf() and fscanf())

For reading and writing to a text **FILE**, we use the functions **fprintf()** and **fscanf()**. They are just the **FILE** versions of **printf()** and **scanf()**.

Example Data Store :

```

#include<stdio.h>
#include<conio.h>

void main()
{
    int Roll, Age;
    char Name[20], key;
    FILE *fp1;
    fp1 = fopen("Data.txt", "w");
    clrscr();

    while(1)
    {
        printf("\nEnter Roll Number ..... : ");
        scanf("%d", &Roll);
        printf("Enter Student Name ..... : ");
        scanf("%s", &Name);
        printf("Enter Student Age ..... : ");
        scanf("%d", &Age);
    }
}

```

```

    fprintf(fp1,"%d %s %d\n", Roll,Name,Age);

    printf("\nDo you want to add more Record [y/n] ");
    key=getche();
    if (key=='y' || key=='Y')
        continue ;
    else
        break;
}
fclose(fp1);
}

```

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter Roll Number ..... : 10
Enter Student Name ..... : imran
Enter Student Age ..... : 19

Do you want to add more Record [y/n] y
Enter Roll Number ..... : 20
Enter Student Name ..... : farhan
Enter Student Age ..... : 20

Do you want to add more Record [y/n] y
Enter Roll Number ..... : 30
Enter Student Name ..... : Muhammad
Enter Student Age ..... : 18

Do you want to add more Record [y/n] n

```

Example Data Read :

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int Roll,Age,a=0;
    char Name[20];
    FILE *fp2;
    fp2 = fopen("Data.txt", "r");
    clrscr();

    printf("roll Name Age \n");
    printf("=====\n");
    do
    {
        a++;
        fscanf(fp2,"%d %s %d\n",&Roll,&Name,&Age);
        printf("%d %s %d \n", Roll,Name,Age);
    }
}

```

```

}
while( a != 3 );
fclose(fp2);
getche();
}

```



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
roll Name Age
=====
10 Imran 18
20 Farhan 19
30 Muhammad 20

```

In this program, we have create two FILE pointers and both are referring to the same FILE but in different modes. **fprintf()** function directly writes into the FILE, while **fscanf()** reads from the FILE, which can then be printed on console using standard **printf()** function.

Reading and Writing from FILE using **fputs()** and **fgets ()** functions

The C library function **int fputs(const char *str, FILE *stream)** writes a string to the specified stream up to but not including the null character.

Syntax : int fputs(const char *str, FILE *stream)

- **str** – This is an array containing the null-terminated sequence of characters to be written.
- **stream** – This is the pointer to a FILE object that identifies the stream where the string is to be written

Example

```

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

    fp = fopen("Stringfile.txt", "w");
    clrscr();
    fputs("This is c programming. ", fp);
    fputs("This is a system programming language.", fp);
    printf("This is c programming and This is a system programming language.\n Above program has
been saved in Stringfile.txt");
    fclose(fp);
}

```


Let us compile and run the above program, this will create a **FILE Stringfile.txt** with the following content

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
This is C Programming and this is System programming language
Above data has been saved in Stringfile.txt_
```

Reading String Data:

The C/C++ library function `char *fgets(char *str, int n, FILE *stream)` 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, the newline character is read, or the end-of-FILE is reached, whichever comes first.

Syntax: `char *fgets(char *str, int n, FILE *stream)`

- `str` – This is the pointer to an array of chars 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:

Now let's see the content of the above **FILE** using the following program

```
#include <stdio.h>
#include <conio.h>
void main ()
{ char st[81];
  FILE *fp;
  fp = fopen("StringFILE.txt", "r");

  clrscr();
  fgets(st,20,fp); // read 20 characters
  puts(st);
  fgets(st,60,fp); // read 60 characters
  puts(st);
  fclose(fp);
}
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
This is c programmi
ng. This is a system programming language.
```

Reading and writing to a binary file

Functions **fread()** and **fwrite()** are used for reading from and writing to a **FILE** on the disk respectively in case of binary **FILEs**.

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

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

1. **ptr** : ptr is the reference of an array or a structure stored in memory.
2. **size** : size is the total number of bytes to be written.
3. **n** : n is number of times a record will be written.
4. **FILE*** : FILE* is a file where the records will be written in binary mode.

Example

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

```
struct Student
{
    int roll;
    char name[25];
    float per;
};
```

```
void main()
{
```

```
    FILE *fp;
    char key,temp[20];
    struct Student Stu;
    clrscr();
    fp = fopen("Student.dat","w");       //Statement 1
```

```
    if(fp == NULL)
    {
        printf("\nCan't open file or file doesn't exist.");
        exit(0);
    }
```

```
    do
    {
        printf("\n\tEnter Roll : ");
        gets(temp);
        Stu.roll = atoi(temp);
```

```
printf("\tEnter Name : ");
gets(Stu.name);

printf("\tEnter Percentage : ");
gets(temp);
Stu.per = atof(temp);

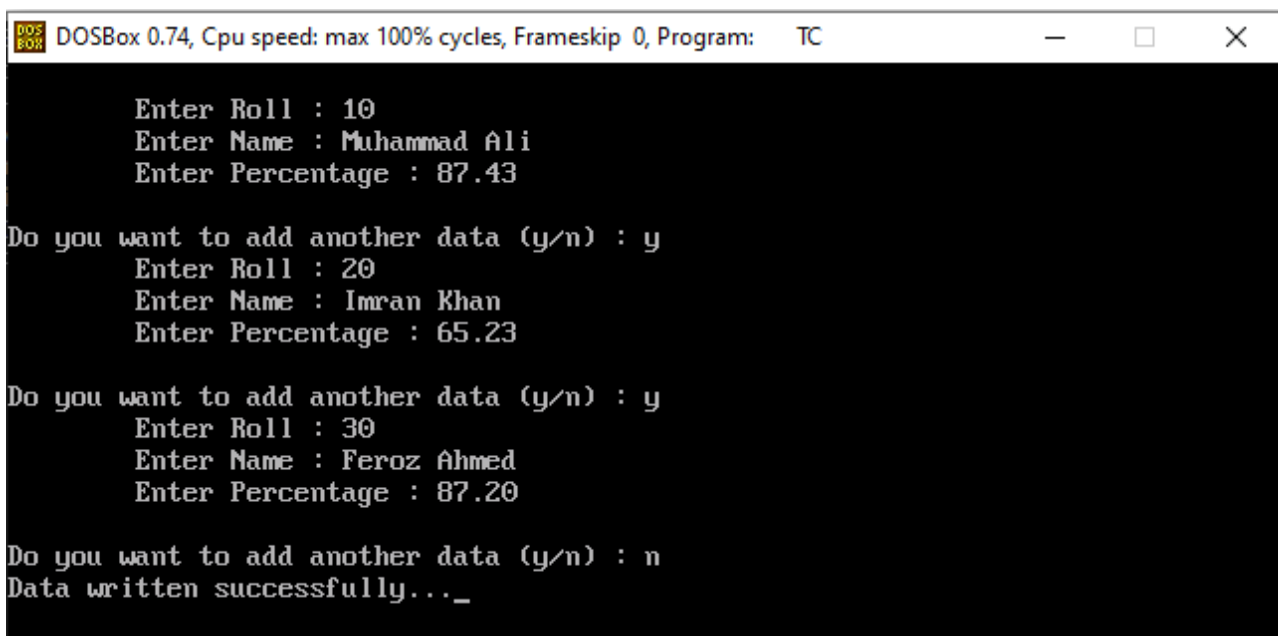
fwrite(&Stu,sizeof(Stu),1,fp);

printf("\nDo you want to add another data (y/n) : ");
key = getche();

}while(key=='y' || key=='Y');

printf("\nData written successfully...");

fclose(fp);
getche();
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter Roll : 10
Enter Name : Muhammad Ali
Enter Percentage : 87.43
Do you want to add another data (y/n) : y
Enter Roll : 20
Enter Name : Imran Khan
Enter Percentage : 65.23
Do you want to add another data (y/n) : y
Enter Roll : 30
Enter Name : Feroz Ahmed
Enter Percentage : 87.20
Do you want to add another data (y/n) : n
Data written successfully..._
```

Example: Read all record form using fread()

The fread() function is used to read bytes form the file. The fread() function takes four arguments.

Syntax: `read(ptr, int size, int n, FILE *fp);`

1. **ptr** : ptr is the reference of an array or a structure where data will be stored after reading.
2. **size** : size is the total number of bytes to be read from file.
3. **n** : n is number of times a record will be read.
4. **FILE*** : FILE* is a file where the records will be read.

Example

```
#include <stdio.h>
#include <conio.h> // for use clrscr() and getche() functions
#include <stdlib.h> // for use exit(0) function
struct Student
{
    int roll;
    char name[25];
    float per;
};

void main()
{
    FILE *fp;
    char ch;
    struct Student Stu;
    clrscr();
    fp = fopen("Student.dat","r"); //Statement 1

    if(fp == NULL)
    {
        printf("\nCan't open file or file doesn't exist.");
        exit(0);
    }

    printf("\n\tRoll\tName\t\tPercentage\n");

    while(fread(&Stu,sizeof(Stu),1,fp)>0)
        printf("\n\t%d\t%s\t%.3f",Stu.roll,Stu.name,Stu.per);

    fclose(fp);
    getche();
}
```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Roll	Name	Percentage
10	Muhammad Ali	87.430
20	Imran Khan	65.230
30	Feroz Ahmed	87.200

Example: Search Record (linear search)

```
#include <stdio.h>
#include <conio.h> // for use clrscr() and getch() functions
#include <stdlib.h> // for use exit(0) function

struct Student
{
    int roll;
    char name[25];
    float per;
};

void main()
{
    FILE *fp;
    int mroll,found;
    char key;
    struct Student Stu;
    clrscr();
    fp = fopen("Student.dat","r"); //Statement 1

    if(fp == NULL)
    {
        printf("\nCan't open file or file doesn't exist.");
        exit(0);
    }

    while(1)
    {
        clrscr();
        found=0;
        printf("Enter Roll number For Search : ");
        scanf("%d",&mroll);
```

```

printf("\n\tRoll\tName\t\tPercentage\n");

while(fread(&Stu,sizeof(Stu),1,fp)>0)
{
    if(mroll == Stu.roll)
    { printf("\n\t%d\t%s\t%.3f",Stu.roll,Stu.name,Stu.per);
      found=1;

      break;
    }
}
rewind(fp);
if(found==0)
    printf(" \n\n \t\t Record is nout found ");
printf("\n\n Search Another Record [y/n] ");
key=getche();
if(key=='y' || key=='Y')
    continue;
else
    break;
}
fclose(fp);
}

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```

Enter Roll number For Search : 20

      Roll      Name          Percentage
      20       Imran Khan      65.230

Search Another Record [y/n] _

```

If record is not found then display following screen view

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```

Enter Roll number For Search : 60

      Roll      Name          Percentage

      Record is nout found

Search Another Record [y/n]

```

The Fseek() Function

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

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

1. **stream** – This is the pointer to a FILE object that identifies the stream.
2. **offset** – This is the number of bytes to offset from whence.
3. **whence** – This is the position from where offset is added. It is specified by one of the following constants (`SEEK_SET` for beginning of file , `SEEK_CUR` for current position of file and `SEEK_END` for end of file)

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

```
void 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);
```

```
}
```

Let us compile and run the above program that will create a file `file.txt` with the following content. Initially program creates the file and writes `This is tutorialspoint.com` but later we had reset the write pointer at 7th position from the beginning and used `fputs()` statement which over-write the file with the following content –

```
This is C Programming Language
```

Exercise**Theory question**

1. Why file are needed.
2. How many types of file.
3. How many operation in the file.
4. How many modes of file opening with detail.

Practical question

1. Write a program to copy one file to another.
2. Write a program to count total number of blank spaces in a file.
3. Write a program to count total characters, words and line in a file.
4. Write a menu driven program to add, display, search, update and delete the student

Objective MCQ's