Files

File

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data.

In C language, we use a structure **pointer of file type** to declare a file.

FILE *fp;

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

Function	Description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

fopen()

The fopen () function opens a file.

Syntax:

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

The fopen() function opens a file whose name is pointed to by 'filename' and returns the stream that is associated with it. The type of operation that will be allowed on the file are defined by the value of mode. The legal values of modes are shown in the following table.

File Type	Meaning
“r”	Open an existing file for reading only
“w”	Open a new file for writing only. If a file with the specified file-name currently exists, it will be destroyed and a new file created in its place.
“a”	Open an existing file for appending (i.e., for adding new information at the end of the file). If a file with the specified file-name currently does not exist, a new file will be created.
“r+”	Open an existing file for both reading and writing.
“w+”	Open a new file for both reading and writing. If a file with the specified file-name currently exists, it will be destroyed and a new file created in its place.
“a+”	Open an existing file for both reading and appending. If a file with the specified file-name currently does not exist, a new file will be created.
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.

If fopen () is successful in opening the specified file, a FILE pointer is returned. If the file cannot be opened, a NULL pointer is required. The following code uses fopen() to open a file named TEST for output.

```
FILE *fptr;
fptr = fopen("TEST","w");
```

Although the preceding code is technically correct, the following code fragment illustrates the correct method of opening the file.

```
FILE *fptr;
if ((fptr = fopen ("TEST","w"))==NULL)
     {
     printf ("Cannot open file\n");
     exit (1);
    }
```

This method detects any error in opening a file, such as write-protected or a full disk before attempting to write to it. If no file by that name exists, one will be created. Opening a file for read operations require that the file exists.

fclose()

The fclose function closes the file.

Syntax:

```
int fclose (FILE *stream);
```

The fclose() function closes the file associated with stream and flushes its buffer (i.e., it writes any data still remaining in the disk buffer to the file). After a call to fclose(), stream is no longer connected with the file, and any automatically allocated buffers are de allocated.

If fclose () is successful, zero is returned; otherwise EOF is returned.

For example, the following code opens and closes a file:

```
#include<stdio.h>
#include<stdlib.h>
void main()
    {
    FILE *fptr;
    if((fptr = fopen("TEST","w"))==NULL)
          {
```

```
        printf("Cannot open file\n");
        exit(1);
        }
    if(fclose(fptr))
        printf("File close error\n");
    }
```

getc()

The getc() function returns the next character from the specified input stream and increment file position indicator. The character is read as an unsigned char that is converted to an integer.

If the end-of-file is reached, getc() returns EOF. If getc() encounters an error, EOF is also returned. The function getc() and fgetc() are identical.

Syntax:

```
int getc(FILE *stream);
```

For example, the following program reads and displays the contents of a text file.

```
#include <stdio.h>
#include <stdlib.h>
void main()
    {
    FILE *fptr;
    char c;
    clrscr();

    if((fptr = fopen("TEST5.txt","r"))==NULL)
        {
        printf("Cannot open file\n");
        exit(1);
        }

    while((c=getc(fptr))!=EOF)
        putchar(c);

    if(fclose(fptr))
        printf("File close error\n");

    getch();
    }
```

Output

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("\n hello world");
getch();
}
```

putc()

Syntax:

int putc(int ch, FILE *stream);

The putc() function writes the character ch to the specified stream at the current file position and then advance the file position indicator. Even though the ch is declared to be an int, it is converted by putc() into an unsigned char.

The value returned by the putc() is the value of the character written . If an error occurs, EOF is returned. The putc() and fputc() are identical.

For example, the following program writes a string to the specified stream.

```
#include <stdio.h>
#include <stdlib.h>
void main()
    {
    FILE *fptr;
    char text[100];
     int i=0;
     clrscr();

     printf(“Enter a text:\n”);
     gets(text);
     if((fptr = fopen(“TEST”,”w”))==NULL)
          {
          printf(“Cannot open file\n”);
          exit(1);
          }

    while(text[i]!=’\0’)
    putc(text[i++],fptr);

    if(fclose(fptr))
         printf(“File close error\n”);
```

```
getch();
}
```

Output

Enter a text:
abcdef
open the file test6.txt, it will show
abcdef

fprintf()

Syntax:

fprintf (FILE *stream, const char *format,…..);

The fprintf() function outputs the values of the arguments that makes up the argument list as specified in the format string to the stream pointed to by stream. The operations of the format control string and command are identical to those in printf().

The following program uses the fprintf() and fscanf() function to prepare the results of ‘n’ students.

```
#include<stdio.h>
 void main()
      {
      int regno,m[5],tot,i,j,k,n;
      char name[15];
      float avg;
      FILE * fptr;
      if((fptr = fopen(“MARK”,”w”))==NULL)
              {
              printf(“Cannot open file\n”);
              exit(1);
             }
     clrscr();
     printf(“How many students?”);
     scanf(“%d”,&n);
     for(i=0;i<n;i++)
             {
             clrscr();
             printf(“Enter the details of Student-%d\n\n”,i+1);
             printf(“Name?”);
```

```
            scanf(%[^\n],name);
            printf("Register Number?");
            scanf(%d,&regno);
            for(tot=0,j=0;j<5;++j)
                  {
                  printf("Mark in paper-%d?",j+1);
                  scanf(%d,&m[j]);
                  tot+=m[j];
                  }
           avg=(float)tot/5.0;
           fprintf(fptr,"%15s%d%d%d%d%d%d%f\n",name,regno,m[0], m[1],m[2], m[3],
m[4],tot,avg);
           }
    clrscr();
    fclose(fptr);
    if((fptr = fopen("MARK","r"))==NULL)
           {
           printf("Cannot open file\n");
           exit(1);
           }
     printf("NAME\t\tReg.No\TM1 M2 M3 M4 M5 TOTAL AVERAGE RES");
     for(i=0;i<n;i++)
           {
           fscanf(fptr,"%-15s%d%d%d%d%d%d%f",&name,&regno,&m[0], &m[1],&m[2],
&m[3], &m[4],&tot,&avg);
           printf("%15s%d%d%d%d%d%d%f\n",name,regno,m[0], m[1],m[2], m[3],
m[4],tot,avg);
           if(avg>35.0)
                  printf("Passed");
           else
                  printf("Failed");
           }
   fclose(fptr);
   getch();
  }
```

fscanf()

Syntax:

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

The fscanf function works exactly like the scanf function except that it reads the information from the stream specified by stream instead of standard input device.

fgets()

Syntax:

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

The fgets() function reads up to num-1character from stream and store them into a character array pointed to by str. Characters are read until either a newline or an EOF is received or until the specified limit is reached. After the character has been read, a null is stored in the array immediately after the last character is read. A newline character will be retained and will be a part of the array pointed to by str.

If successful, fgets() returns str; a null pointer is returned upon failure. If a read error occurs, the content of the array pointed to by str are indeterminate.

For example, the following program uses fgets() to display the content of a file.

```
#include<stdlib.h>
#include <stdio.h>
void main()
    {
    FILE *fptr;
    char text[100];
    clrscr();
    if((fptr = fopen("TEST","r"))==NULL)
            {
            printf("Cannot open file\n");
            exit(1);
            }
     fgets(text,100,fptr);
     puts(text);
     if(fclose(fptr))
              pritf("File close error\n");
     getch();
     }
```

fputs()

Syntax:

char * fputs(const char *str , FILE *stream);

The fputs() function writes the content of the string pointed to by str to the specified stream. The null terminator is not written. The fputs() function returns non negative on success and EOF on failure.

For example, the following program uses fputs() to write a string into a file.

```
#include<stdio.h>
#include<stdlib.h>
int main()
    {
    FILE *fptr;
    char text[100];
    int i=0;
    clrscr();
    printf("Enter a text:\n");
    gets(text);
    if((fptr = fopen("TEST","w"))==NULL)
            {
            printf("Cannot open file\n");
            exit(1);
            }
    fputs(text,fptr);
    if(fclose(fptr))
              pritf("File close error\n");
     getch();
     return 0;
     }
```

fgetc()

Syntax:

int fgetc(FILE *stream);

The fgetc() function returns the next character from the specified input stream and increments the file position indicator. The character is read as an unsigned char that is converted to an integer. If the end-of-file is reached, fgetc() returns EOF. If fgetc() encounters an error, EOF is also returned.

For example, the following program reads and displays the content of text file.

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

```
#include<stdlib.h>
int main()
    {
    FILE *fptr;
    if((fptr = fopen("TEST","r"))==NULL)
          {
          printf("Cannot open file\n");
          exit(1);
          }
    while((c=fgetc(fptr))!=EOF)
          puchar(c);
    if(fclose(fptr))
          pritf("File close error\n");
    return 0;
   }
```

fputc()

Syntax:

int fputc(int ch,FILE *stream);

The fputc() function writes the character ch to the specified stream at the current file position and then advance the file position indicator. Even though the ch is declared to be an int , it is converted by fputc() into an unsigned char.

The value returned by the fputc() is the value of the character written . if an error occurs, EOF is returned.

For example, the following program writes to the specified stream.

```
#include <stdio.h>
#include<stdlib.h>
void main()
     {
     FILE *fptr;
     char text[100];
     int i=0;
     clrscr();
     printf("Enter a text:\n");
     gets(text);
     if((fptr = fopen("TEST","w"))==NULL)
          {
              printf("Cannot open file\n");
              exit(1);
```

```
    }
while(text[i]!='\0')
        fputc(text[i++],fptr);

if(fclose(fptr))
        pritf("File close error\n");
getch();
}
```