**Chapter-10**

**File Handling in C**

**Concept of 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. It is a readymade structure. File is a collection of related data placed on the disk. C support the concept of files through which data can be stored on the disk or secondary storage device. The stored data can be read whenever required.

Input

Output

Memory

Data file stored in a secondary storage device

fputc

fputs

fprintf

fwrite

fgetc

fgets

fscanf

fread

scanf

printf

**Opening a File**

A file must be opened before any I/O operations performed on that file. The process of establishing a connection between the program and file is called opening the file. A structure named FILE is defined in the file stdio.h that contains all information about the file like name, status, buffer size, current position, end of file status etc. All these details are hidden from the programming and the operating system takes care of all these things.

A file pointer is a pointer to a structure of type FILE. Whenever a file is opened, a structure of type FILE is associated with it and a file pointer that points to this structure identifies this file.

Eg:

FILE \*fp;

The function fopen ( ) is used to open a file.

Eg:

fp=fopen(“path:\\filename.ext”,”file\_operation\_mode”);

#include<stdio.h>

main()

{

FILE \*fp;

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

if (fp == NULL)

{

printf("File does not exist, please check!\n");

}

fclose(fp);

}

Let us first discuss fopen(). This function accepts two arguments as strings. The first argument denotes the name of the file to be opened and the second signifies the mode in which the file is to be opened.

**Closing of file**

The fclose() function is used for closing opened files. The only argument it accepts is the file pointer.

If a program terminates, it automatically closes all opened files. But it is a good programming habit to close any file once it is no longer needed. This helps in better utilization of system resources, and is very useful when you are working on numerous files simultaneously. Some operating systems place a limit on the number of files that can be open at any given point in time.

The different file modes that can be used in opening file are:

|  |  |
| --- | --- |
| 1. “w” (write) | If the file doesn’t exist then this mode creates a new file for writing, and if the file already exists then the previous data is erased and the new data entered is written to the file. |
| 1. “a” (append) | If the file doesn’t exist then this mode creates a new file and if the file already exists then the new data entered is appended at the end of existing data. In this mode, the data existing in the file is not erased as in “w” mode. |
| 1. “r” (read) | This mode is used for opening an existing file for reading purpose only. The file to be opened must exist and the previous data of the file is not erased. |
| 1. “w+” (write + read) | This mode is same as “w” mode but in this mode we can also read and modify the data. If the file doesn’t exist then a new file is created and if the file exists then previous data is erased. |
| 1. “r+” (read + write) | This mode is same as “r” mode but in this mode we can also write and modify existing data. The file to be opened must exist and the previous data of file is not erased. Since we can add new data and modify existing data so this mode is also called updata mode. |
| 1. “a+” (append + read) | This mode is same as the “a” mode but in this mode we can also read the data stored in the file. If the file doesn’t exist, a new file is created and if the file already exists then new data is appended at the end of existing data. We cannot modify existing data in this mode. |
| 1. “wb” | Binary file opened in write mode. |
| 1. “ab” | Binary file opened in append mode. |
| 1. “rb” | Binary file opened in read mode. |
| 1. “wb+” | Create a binary file for read/write. |
| 1. “rb+” | Open a binary file for read/write. |
| 1. “ab+” | Append a binary file for read/write. |

**Text mode**

* In text mode every digit or text are stored as a character and while reading the content back, the conversion is required from character to appropriate format and takes lots of space.
* Character I/O, string I/O, and formatted I/O use text mode.
* If 3.14159 is stored in character mode file size would be 8 bytes (counts each character including decimal and EOF).

**Binary mode**

* In binary mode every digit or text is stored in binary format and while reading the content no conversion is necessary and takes little space.
* fread ( ) and fwrite ( ) are used in binary mode.
* If 3.14159 is stored in character mode file size would be 4 bytes.

**Input/Output Functions**

The functions used for file input/output are

1. Character I/O
2. fputc( )

This function writes a character to the specified file at the current file position and then increments the file position pointer.

1. fgetc( )

This function reads a single character from a given file and increments the file pointer.

1. getc( ) and putc( )

The operation of getc( ) and putc( ) are exactly similar to that of fgetc( ) and fputc( ) , the only difference is that the former two are defined as macros while the latter two are functions.

1. Integer I/O
2. putw( )

This function writes an integer value to the file pointed to by file\_pointer.

1. getw( )

This function returns the integer value from the file associated with file\_pointer.

1. String I/O
2. fputs( )

This function writes the null terminated string pointed by given character pointer to a file.

1. fgets()

This function is used to read characters from a file and these characters are stored in the string pointed by a character pointer.

1. Formatted I/O
2. fprintf( )

This function is same as the printf( ) function but it writes formatted data into the file instead of the standard output(screen). This function has same parameters as in printf( ) but it has one additional parameter which is a pointer of FILE type, that points to the file to which the output is to be written.

1. fscanf( )

This function is similar to the scanf( ) function but it reads data from file instead of standard input, so it has one more parameter which is a pointer of FILE type and it points to the file from which data will be read.

1. Block Read/Write
2. fwrite( )

This function is used for writing an entire block to a given file.

1. fread( )

This function is used to read an entire block from a given file.

**fscanf() and fprintf()**

The functions fprintf() and fscanf() are similar to printf() and scanf() except that these functions operate on files and require one additional and first argument to be a file pointer.

Example

#include <stdio.h>

main ()

{

FILE \*fp;

float total;

fp = fopen("data.txt", "w+");

if (fp == NULL)

{

printf("data.txt does not exist, please check!\n");

exit (1);

}

fprintf(fp, 100);

fscanf(fp, "%f", &total);

fclose(fp);

printf("Value of total is %f\n", total);

}

**getc() and putc()**

The functions getc() and putc() are equivalent to getchar() and putchar() functions, except that these functions require an argument which is the file pointer. Function getc() reads a single character from the file which has previously been opened using a function like fopen(). Function putc() does the opposite, it writes a character to the file identified by its second argument. The format of both functions is as follows :

getc(in\_file);

putc(c, out\_file);

Note: The second argument in the putc() function must be a file opened in either write or append mode.

Example

#include <stdio.h>

main ()

{

char in\_file[30], out\_file[30];

FILE \*fpin, \*fpout;

int c;

printf("This program copies the source file to the destination file \n\n");

printf("Enter name of the source file :");

scanf("%30s", in\_file);

printf("Enter name of the destination file :");

scanf("%30s", out\_file);

if((fpin=fopen(in\_file, "r")) == NULL)

printf("Error could not open source file for reading\n");

else if ((fpout=fopen(out\_file, "w")) == NULL)

printf("Error could not open destination file for reading\n");

else

{

while((c =getc(fpin)) != EOF)

putc(c, fpout);

printf("Destination file has been copied\n");

}

}

**Binary stream input and output**

The functions fread() and fwrite() are a somwhat complex file handling functions used for reading or writing chunks of data containing NULL characters ('\0') terminating strings.

The function prototype of fread() and fwrite() is as below :

size\_t fread(void \*ptr, size\_t sz, size\_t n, FILE \*fp)

size\_t fwrite(const void \*ptr, size\_t sz, size\_t n, FILE \*fp);

You may notice that the return type of fread() is size\_t which is the number of items read. You will understand this once you understand how fread() works. It reads n items, each of size sz from a file pointed to by the pointer fp into a buffer pointed by a void pointer ptr which is nothing but a generic pointer. Function fread() reads it as a stream of bytes and advances the file pointer by the number of bytes read. If it encounters an error or endof-file, it returns a zero, you have to use feof() or ferror() to distinguish between these two. Function fwrite() works similarly, it writes n objects of sz bytes long from a location pointed to by ptr, to a file pointed to by fp, and returns the number of items written to fp.

Let us rewrite program using fread() and fwrite() functions .

#include <stdio.h>

#define MAX\_SIZE 1024

main ()

{

FILE \*fp, \*gp;

char buf[MAX\_SIZE];

int i, total = 0;

if ((fp = fopen("data1.txt", "r") ) == NULL)

printf("Error in data1.txt file \n");

else if ((gp=fopen("data2.txt", "w")) == NULL)

printf("Error in data2.txt file \n");

else

{

while(i=fread(buf, 1, MAX\_SIZE, fp))

{

fwrite(buf, 1, MAX\_SIZE, gp);

total +=i;

}

printf("Total is %d\n", total);

}

fclose(fp);

fclose(gp);

}

**Random file processing**

In this technique the content are stored sequentially and read back the content from any position of the file. And it can be performed using some function like rewind( ), fseek( ) and ftell( ).

**rewind( )**

The rewind( ) function places the file pointer to the beginning of the file.

rewind(file\_pointer);

**fseek( )**

This function is used for setting the file position pointer at the specified byte.

fseek(file\_pointer,offset,offset\_initial);

**ftell( )**

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

position=ftell(fp); where position is long int.

* Write a program to write name, symbolno and address of ‘n’ students into a file named “student.txt” and display the record of students in appropriate format.

#include<stdio.h>

#include<conio.h>

void main()

{

struct student

{

int sno;

char name[15];

char add[15];

};

struct student st;

FILE \*fp;

fp=fopen("student.txt","w+");

int i,n;

clrscr();

printf("How many students are there: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\nEnter symbolno, name and address of student: ");

scanf("%d%s%s",&st.sno,st.name,st.add);

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

}

rewind(fp);

printf("\nSymbno\tName\tAddress");

while(fread(&st,sizeof(st),1,fp)==1)

{

printf("\n%d\t%s\t%s",st.sno,st.name,st.add);

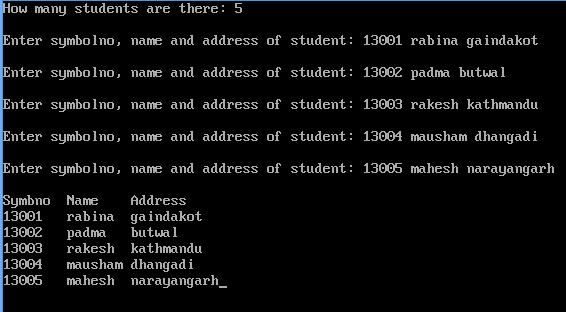
}

fclose(fp);

getch();

}

Output



* Write a program to delete and rename the file.

#include<stdio.h>

#include<conio.h>

void main()

{

char str[20];

FILE \*fp;

clrscr();

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

printf("\nInput string: ");

scanf("%s",str);

fprintf(fp,"%s",str);

fclose(fp);

rename("fil.txt","newfil.txt"); /\* rename the file name to newfil.txt \*/

remove("newfil.txt"); /\* removes(delete) file newfil.txt \*/

getch();

}

* Write a program to copy contents of one file to another

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fp1,\*fp2;

char ch;

clrscr();

fp1=fopen("abc.txt","r");

fp2=fopen("xyz.txt","w");

while((ch=getc(fp1))!=EOF)

{

putc(ch,fp2);

}

fclose(fp1);

fclose(fp2);

getch();

}

* Program to understand fputc( ) and fgetc( ).

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fp;

char ch;

clrscr();

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

printf("Enter text (Press ctrl+z to stop reading character)\n");

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

{

fputc(ch,fp);

}

fclose(fp);

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

while((ch=fgetc(fp))!=EOF)

{

printf("%c",ch);

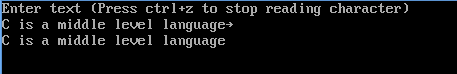
}

fclose(fp);

getch();

}

Output



* Program to understand putc( ) and getc( ).

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fptr;

char ch;

printf("\n Enter any character: ");

scanf("%c",&ch);

fptr=fopen("abc.txt","w");

putc(ch,fptr);

fclose(fptr);

fptr=fopen("abc.txt","r");

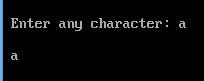
ch=getc(fptr);

printf("\n %c",ch);

getch();

}

Output



* Program to understand putw( ) and getw( )

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fp;

int value;

clrscr();

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

for(value=1;value<=10;value++)

{

putw(value,fp);

}

fclose(fp);

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

while((value=getw(fp))!=EOF)

{

printf("%d\n",value);

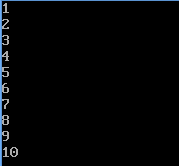
}

fclose(fp);

getch();

}

Output



* Program to understand fputs( ) and fgets( ).

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fptr;

char str[80];

fptr=fopen("abc.txt","w");

printf("Enter the text (Press ctrl+z to stop entering) \n");

while(gets(str)!=NULL)

fputs(str,fptr);

fclose(fptr);

fptr=fopen("abc.txt","r");

while(fgets(str,80,fptr)!=NULL)

{

puts(str);

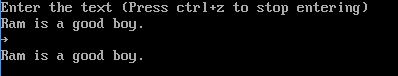
}

fclose(fptr);

getch();

}

Output



* Program to understand fprintf( ) and fscanf( )

#include<stdio.h>

#include<conio.h>

void main()

{

struct student

{

int roll;

char name[15];

};

struct student st;

FILE \*fp;

int i,n;

clrscr();

printf("How many records do you want to enter: ");

scanf("%d",&n);

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

for(i=0;i<n;i++)

{

printf("\n Enter rollno and name of student: ");

scanf("%d%s",&st.roll,st.name);

fprintf(fp,"\n%d%s",st.roll,st.name);

}

fclose(fp);

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

printf("\n Rollno\tName");

while(fscanf(fp,"%d%s",&st.roll,st.name)!=EOF)

{

printf("\n %d\t%s",st.roll,st.name);

}

fclose(fp);

getch();

}

Output

