18ES611 Embedded System Programming

Sarath tv

File Operations

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:

Text files

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, you can perform four major operations on the file, either text or binary:

- Creating a new file
- Opening an existing file
- Closing a file
- Reading from and writing information to a file

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

```
FILE *fptr;
```

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 beginning point

Opening a File or Creating a File

The fopen() function is used to create a new file or to open an existing file.

General Syntax :

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

Here **filename** is the name of the file to be opened and **mode** specifies the purpose of opening the file. Mode can be of following types,

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

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.

r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Difference between Append and Write Mode

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.

The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file. While in append mode this will not happen. Append mode is used to append or add data to the existing data of file(if any). Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file.

Closing a File

The fclose() function is used to close an already opened file.

General 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.

```
#include <stdio.h>
main ()
{
FILE *fp;
fp = fopen("data.txt", "r");
if (fp == NULL) {
printf("File does not exist,
please check!\n");
}
fclose(fp);
}
```

Input/Output operation on File

getc() and putc() are simplest functions used to read and write individual characters to a file.

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.

```
#include<stdio.h>
#include<conio.h>
main()
{
 FILE *fp;
 char ch;
 fp = fopen("one.txt", "w");
 printf("Enter data");
 while( (ch = getchar()) != EOF) {
    putc(ch,fp);
 }

 fclose(fp);
 fp = fopen("one.txt", "r");
 while( (ch = getc()) != EOF)
    printf("%c",ch);
 fclose(fp);
}
```

Reading and Writing in a Binary File

A Binary file is similar to the text file, but it contains only large numerical data. The Opening

modes are mentioned in the table for opening modes above.

fread() and **fwrite()** functions are used to read and write a binary file.

fwrite(data-element-to-be-written, size_of_elements, number_of_elements, pointer-to-file);

fread() is also used in the same way, with the same arguments like fwrite() function

The functions **fread() and fwrite()** are a somewhat 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);

return type of *fread()* is size_t which is the number of items read.

fread() -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 end-of-file, **it returns a zero**.

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*.

fseek(), ftell() and rewind() functions

fseek() - It is used to move the reading control to different positions using fseek function.

ftell() - It tells the byte location of current position of cursor in file pointer.

rewind() - It moves the control to beginning of the file.

fseek()

This function positions the next I/O operation on an open stream to a new position relative to the current position.

```
int fseek(FILE *fp, long int offset, int origin);
```

Here *fp* is the file pointer of the stream on which I/O operations are carried on, *offset* is the number of bytes to skip over. The *offset* can be either positive or negative, denting forward or backward movement in the file. *origin* is the position in the stream to which the offset is applied, this can be one of the following constants :

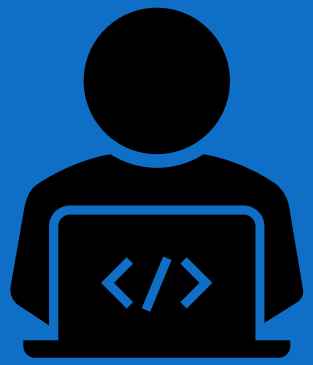
SEEK_SET : offset is relative to beginning of the file

SEEK_CUR : offset is relative to the current position in the file

SEEK_END : offset is relative to end of the file

Function *ftell()* returns the current position of the file pointer in a stream.

The return value is 0 or a positive integer indicating the byte offset from the beginning of an open file. A return value of -1 indicates an error.



THANK YOU!!!!!