Introduction to file operations

Organised & Supported by **RuggedBOARD**

Agenda

- File operations
- File open
- FILE structure
- fget(),fputc(),fclose()
- String(Line) I/O
- Text & Binary Files

File Operations

File Handling is the storing of data in a file using a program. In C programming language, the programs store results, and other data of the program to a file using *file handling* in C. Also, we can extract/fetch data from a file to work with it in the program.

- Creating a new file
- Opening an existing file
- Reading data from an existing file
- Writing data to a file
- Moving data to a specific location on the file
- Closing the file

File Open

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

The argument mode points to a string beginning with one of the following sequences :

r	Open text file for reading. The stream is positioned at the beginning of the file.
r+	Open for reading and writing. The stream is positioned at the beginning of the file.
w	Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
w+	Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
a	Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
a+	Open for reading and appending (writing at end of file). The file is created if it does not exist. Output is always appended to the end of the file. POSIX is silent on what the initial read position is when using this mode. For glibc, the initial file position for reading is at the beginning of the file, but for Android/BSD/MacOS, the initial file position for reading is at the end of the file.

```
# include <stdio.h>
#include <stdlib.h>
int main( )
{
        FILE *fp ;
        char ch ;
        fp = fopen ( "PR1.c","r");//"file1.txt"
        if(fp == NULL)
        {
                printf("Error in opening file\n");
                exit(0);
        }
        while ( 1 )
        {
                ch = fgetc ( fp ) ;
                if ( ch == EOF )
                        break ;
                printf ( "%c", ch ) ;
        }
        fclose ( fp ) ;
        return 0;
}
```

File Open

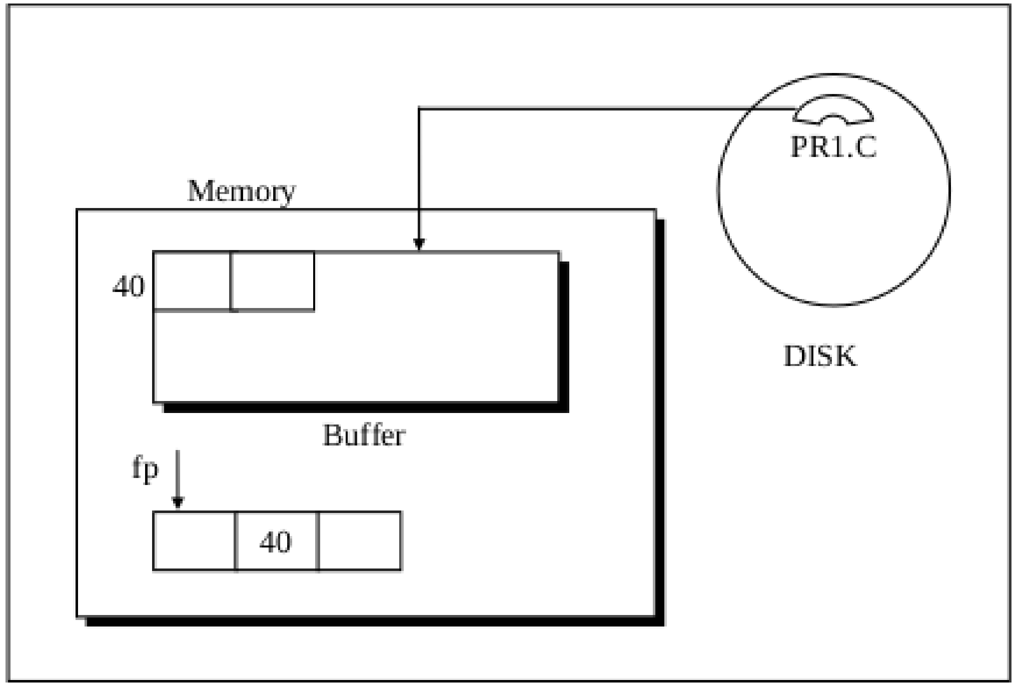
Opening a File

Before we can read (or write) information from (to) a file on a disk we must open the file.
To open the file we have called the function fopen( ).
Note that “r” is a string and not a character; hence the double quotes and not single quotes.

In fact fopen( ) performs three important tasks when you open the file in “r” mode:
(a) Firstly it searches on the disk the file to be opened.
(b) Then it loads the file from the disk into a place in memory called buffer.
(c) It sets up a character pointer that points to the first character of the buffer.
(d)It sets up the FILE structure and returns its address

Why do we need a buffer at all?
Imagine how inefficient it would be to actually access the disk every time we want to read a character from it.
Every time we read something from a disk, it takes some time for the disk drive to position the read/write head correctly. it would take a long time to complete the reading operation.

This is where a buffer comes in.



FILE structure

- To be able to successfully read from a file information like mode of opening, size of file, place in the file from where the next read operation would be performed, etc. has to be maintained.
- Since all this information is inter-related, all of it is gathered together by fopen( ) in a structure called FILE.
- fopen( ) returns the address of this structure, which we have collected in the structure pointer called fp.
- We have declared fp as FILE *fp ;

Note:The FILE structure has been defined in the header file “stdio.h” (standing for standard input/output header file). Therefore, it is necessary to #include this file.

fgetc(),fputc(), fclose()

int fgetc(FILE *stream);

fgetc() reads the next character from stream and returns it as an unsigned char cast to an int, or EOF on end of file or error.

int fputc(int c, FILE *stream);

fputc() writes the character c, cast to an unsigned char, to stream.

int fclose(FILE *stream);

The fclose() function flushes the stream pointed to by stream and closes the underlying file descriptor.

fgetc(),fputc(), fclose()

```
#include<stdio.h>
#include<stdlib.h>
int main( )
{
	FILE *fs,*ft ;
	char ch ;
	fs = fopen("PR1.c","r");//"file1.txt"
	if(fs == NULL)
	{
		printf("Error in opening file\n");
		exit(0);
	}
	ft = fopen("pr2.c", "w") ;
	if ( ft == NULL )
	{
		puts("Cannot open target filei\n") ;
		fclose(fs);
		exit(0) ;
	}
```

```
	while(1)
	{
		ch = fgetc(fs) ;
		if(ch == EOF)
		break ;
		else
		  putc(ch,ft) ;
	}
	fclose(fs);
	fclose(ft);
	return 0;
}
```

String(Line) I/O

char *fgets(char *s, int size, FILE *stream);

- fgets() reads in at most one less than size characters from stream and stores them into the buffer pointed to by s.
- Reading stops after an EOF or a newline.
- If a newline is read, it is stored into the buffer.
- A terminating null byte ('\0') is stored after the last character in the buffer.

int fputs(const char *s, FILE *stream);

- fputs() writes the strings to stream, without its terminating null byte ('\0').

```
#include <stdio.h>
int main()
{
  FILE *fptr = fopen("sample.txt","w");
  fputs("Phytec Embedded\n",fptr);
  fputs("C programming\n",fptr);
  fputs("Rugged Board\n",fptr);
  fclose(fptr);
  fptr = fopen("sample.txt","r");
  char string[30];
  while(fgets(string,30,fptr)!=NULL)
  {
    printf("%s",string);
  }
  fclose(fptr);
  return 0;
}
```

Text & Binary Files

- Text file contains only textual information like alphabets, digits and special symbols. In actuality the ASCII codes of these characters are stored in text files. A good example of a text file is any C program, say PR1.C.
- Binary file is merely a collection of bytes. This collection might be a compiled version of a C program (say PR1.EXE), or music data stored in a wave file or a picture stored in a graphic file.

In text mode, a newline character is converted into the carriage return-linefeed combination before being written to the disk.
Likewise, the carriage return-linefeed combination on the disk is converted back into a newline when the file is read by a C program.
In binary mode, as opposed to text mode, these conversions will not take place.

The second difference between text and binary modes is in the way the end-of-file is detected.
In text mode, a special character, whose ASCII value is 26, is inserted after the last character in the file to mark the end of file.
If this character is detected at any point in the file, the read function would return the EOF signal to the program.
There is no such special character present in the binary mode files to mark the end of file.
The binary mode files keep track of the end of file from the number of characters present in the directory entry of the file.

Note:File that has been written in text mode is read back only in text mode.
Similarly, the file that has been written in binary mode must be read back only in binary mode.

Text & Binary Files

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char* argv[])
{
	if(argc!=2)
	{
		printf("Invalid number of arguments entered\n");
		exit(1);
	}
	FILE* fp=fopen(argv[1],"r");
	if(fp==NULL)
	{
		perror("fopen() failed\n");exit(1);
	}
	int n;
	fscanf(fp,"%d, ",&n);
	while(feof(fp)==0)//end of file indicator
	{
		fprintf(stdout,"%d ",n);
		fscanf(fp,"%d, ",&n);
	 }
	 printf("\n");
	 fclose(fp);
	 return 0;
}
```

```
#include<stdio.h>
struct student
{
	int sno;  char sname [30]; float marks;
	char temp;
};
int main()
{
	struct student s[60];
	int i;
	FILE *fp;
	fp = fopen ("student1.txt", "wb");
	for (i=0; i<2; i++)
	{
		printf ("enter details of student %d\n", i+1);
		printf("student number:");
		scanf("%d",&s[i].sno);
		scanf("%c",&s[i].temp);
		printf("student name:");
		scanf("%[^\n]s",s[i].sname);
		printf("student marks:");
		scanf("%f",&s[i].marks);
		fwrite(&s[i], sizeof(s[i]),1,fp);
	}
	fclose (fp);
	fp = fopen ("student1.txt", "rb");
```

```
for (i=0; i<2; i++)
	{
		printf ("details of student
%d are\n", i+1);
		fread (&s[i], sizeof (s[i])
,1,fp);
		printf("student number =
%d\n", s[i]. sno);
		printf("student name =
%s\n", s[i]. sname);
		printf("marks = %f\n", s[i].
marks);
	}
	fclose(fp);
	return 0;
}
```

Thank You