Unit 10
File Handling in C

What is File Handling in C?

- File handling refers to the method of storing data in the C program in the form of an output or input that might have been generated while running a C program in a data file, i.e., a binary file or a text file for future analysis and reference in that very program.

What is a File in C?

- A file refers to a source in which a program stores the information/data in the form of bytes of sequence on a disk (permanently).
- The content available on a file isn't volatile like the compiler memory in C.
- But the program can perform various operations, such as creating, opening, reading a file, or even manipulating the data present inside the file.
- This process is known as file handling in C.

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.
- You can easily create text files using any simple text editors such as Notepad.
- 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 the 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 a higher amount of data, are not readable easily, and provides better security than text files.

File Operations

- In C, you can perform four major operations on files, either text or binary:
  1. Creating a new file
  2. Opening an existing file
  3. Reading from and writing information to a file
  4. Closing a file

Working with files (File Pointer)

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

```
FILE *fptr
```

Opening a file - for creation and edit

- Opening a file is performed using the fopen() function defined in the stdio.h header file.
- The syntax for opening a file in standard I/O is:

Syntax:

```
ptr = fopen("fileopen","mode");
```

Example:

```
fopen("E:\\cprogram\\newprogram.txt","w");

fopen("E:\\cprogram\\oldprogram.bin","rb");
```

Opening Modes in Standard I/O

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. Data is added to the end of the file.	If the file does not exist, it will be created.
ab	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, 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 exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Closing a File

- The file (both text and binary) should be closed after reading/writing.
- Closing a file is performed using the fclose() function.

```
fclose(fptr);
```

- Here, fptr is a file pointer associated with the file to be closed.

Reading and writing to a text file

- 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().
- The only difference is that fprintf() and fscanf() expects a pointer to the structure FILE.

Example: Write to a text file

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num;
    FILE *fptr;

    // use appropriate location if you are using MacOS or Linux
    fptr = fopen("G://Lab/CH10/lab10.txt","w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);
```

```
        return 0;
    }
```

Output:

```
    Enter num: 20
```

Example: Read from a text file

```
    #include <stdio.h>
    #include <stdlib.h>
    int main()
    {
        int num;
        FILE *fptr;

        if ((fptr = fopen("G://Lab/CH10/lab10.txt","r")) == NULL)
        {
            printf("Error! opening file");
            exit(1);
        }

        fscanf(fptr,"%d", &num);

        printf("Value of n=%d", num);
        fclose(fptr);

        return 0;
    }
```

Output:

```
    Value of n=20
```

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.

Writing to a binary file

- To write into a binary file, you need to use the fwrite() function.
- The functions take four arguments:
    a) address of data to be written in the disk
    b) size of data to be written in the disk
    c) number of such type of data
    d) pointer to the file where you want to write.

```
fwrite(addressData, sizeData, numbersData, pointerToFile);
```

Example: Write to a binary file using fwrite()

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum
{
        int n1, n2, n3;
};

int main()
{
        int n;
        struct threeNum num;
        FILE *fptr;

        if ((fptr = fopen("G://Lab/CH10/BinaryFile.bin","wb")) == NULL)
        {
                printf("Error! opening file");
                exit(1);
        }

        for(n = 1; n < 5; ++n)
        {
                num.n1 = n;
                num.n2 = 5*n;
                num.n3 = 5*n + 1;
                fwrite(&num, sizeof(struct threeNum), 1, fptr);
        }
        fclose(fptr);

        return 0;
}
```

Output:

Output written on the file.

Reading from a binary file

- Function fread() also take 4 arguments similar to the fwrite() function as above.

```
fread(addressData, sizeData, numbersData, pointerToFile);
```

Example: Read from a binary file using fread()

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

```
struct threeNum
{
        int n1, n2, n3;
};

int main()
{
        int n;
        struct threeNum num;
        FILE *fptr;

        if ((fptr = fopen("G://Lab/CH10/BinaryFile.bin","rb")) == NULL)
        {
                printf("Error! opening file");
                exit(1);
        }

        for(n = 1; n < 5; ++n)
        {
                fread(&num, sizeof(struct threeNum), 1, fptr);
                printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
        }
        fclose(fptr);

        return 0;
}
```

Output

```
n1: 1   n2: 5   n3: 6
n1: 2   n2: 10  n3: 11
n1: 3   n2: 15  n3: 16
n1: 4   n2: 20  n3: 21
```

fgetc() and fputc() Functions

fputc() Function:

- fputc() is used to write a single character at a time to a given file.
- It writes the given character at the position denoted by the file pointer and then advances the file pointer.
- This function returns the character that is written in case of successful write operation else in case of error EOF is returned.

Syntax:

```
int fputc(int char, FILE *pointer)
```

Example:

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

```
int main()
{
        FILE *fp;
        int ch;
        fp = fopen("file.txt", "w+");
        for( ch = 65 ; ch <= 90; ch++ )
        {
                fputc(ch, fp);
        }
        fclose(fp);
        return(0);
}
```

Output:

```
good bye
```

fgetc() Function:

- fgetc() is used to obtain input from a file single character at a time.
- This function returns the ASCII code of the character read by the function.
- It returns the character present at position indicated by file pointer.
- After reading the character, the file pointer is advanced to next character.
- If pointer is at end of file or if an error occurs EOF file is returned by this function.

Syntax:

```
int fgetc(FILE *pointer)
```

Example:

```
// C program to illustrate fgetc() function
#include<stdio.h>
#include<stdlib.h>
int main()
{
        // open the file
        FILE *fp = fopen("test.txt","r");
        // Return if could not open file
        if(fp == NULL)
        {
                printf("File error");
                exit(1);
        }

        do
        {
                // Taking input single character at a time
                char c = fgetc(fp);
                // Checking for end of file
```

```
            if (feof(fp))
                break ;
            printf("%c", c);
        } while(1);

        fclose(fp);
        return 0;
    }
```

Output:

Hello from file

fgets() and fputs() Functions

- The fgets() and fputs() functions in C are used for reading and writing strings respectively, from/to a file or standard input/output (stdin/stdout).

fputs() Function:

- The fputs() function writes a string of characters to a file or to standard output. It takes two arguments:
  1. A pointer to the string of characters to be written.
  2. A pointer to the file or output stream to write to.

Example:

```
#include<stdio.h>
#include<string.h>
int main ()
{
    FILE *fp;
    fp = fopen("file.txt", "w+");
    char str[100];
    strcpy(str,"This is c programming.");
    fputs(str, fp);
    fclose(fp);
    return 0;
}
```

Output:

On the file: This is c programming.

fgets() Function:

- The fgets() function reads a string of characters from a file or from standard input and stores it in a buffer. It takes three arguments:
  1. A pointer to a buffer where the string read will be stored.
  2. The maximum number of characters to be read.
  3. A pointer to the file or input stream to read from.

Example:

```
#include <stdio.h>
int main ()
{
        FILE *fp = fopen("file.txt" , "r");
        char str[60];
        fgets(str, 60, fp);
        puts(str);
        fclose(fp);
        return 0;
}
```

Output:

This is c programming.

Random Access in File

- Random access in file refers to the ability to read or write data at any position within a file.
- In C, we can perform random access in a file using the functions fseek(), ftell() and rewind().

fseek()

- The fseek() function allows us to set the position of the file pointer to a specified offset from a given location.
- The function takes three arguments:
  1. A pointer to the file to be modified.
  2. An offset value to set the position of the file pointer.
  3. A predefined location to use as the reference point for the offset, which can be one of the following:

     SEEK_SET: The beginning of the file.

     SEEK_CUR: The current position of the file pointer.

     SEEK_END: The end of the file.

Example:

```
#include <stdio.h>
int main ()
{
        FILE *fp;
        fp = fopen("file.txt","w+");
        fputs("This is Java Programming Language", fp);
        fseek( fp, 7, SEEK_SET);
```

```
        fputs(" C Programming Language", fp);
        fclose(fp);
        return(0);
}
```

Output:

On Printed as: This is C Programming language

ftell()

- The ftell() function returns the current position of the file pointer, relative to the beginning of the file.
- It takes a single argument, which is a pointer to the file to be queried.

Example:

```
// C program to demonstrate use of ftell()

#include<stdio.h>
int main()
{
        /* Opening file in read mode */
        FILE *fp = fopen("test.txt","r");

        /* Reading first string */
        char string[20];
        fscanf(fp,"%s",string);

        /* Printing position of file pointer */
        printf("%ld", ftell(fp));
        return 0;
}
```

Output:

5

rewind()

- The rewind() function in C is used to set the file position indicator to the beginning of the file.
- It is typically used to move the file position indicator back to the beginning of the file after reaching the end of the file, so that the file can be read again from the beginning.
- The rewind() function takes a single argument, which is a pointer to the file to be rewound.

Example:

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

```
int main()
{
	FILE *fp;
	char c;
	fp=fopen("file.txt","r");
	while((c=fgetc(fp))!=EOF)
	{
		printf("%c",c);
	}
	rewind(fp);	//moves the file pointer at beginning of the file
	while((c=fgetc(fp))!=EOF)
	{
		printf("%c",c);
	}
	fclose(fp);
	return 0;
}
```

Output:

This is c programming.This is c programming.

feof() function

- The feof() function takes a single argument, which is a pointer to a FILE object, and returns a non-zero value if the end-of-file indicator has been set on the file stream, or zero otherwise.

Example:

```
#include <stdio.h>
int main()
{
	FILE *fp = fopen("file.txt", "r");
	int c;
	while ((c = fgetc(fp)) != EOF)
	{
		putchar(c);
	}

	if (feof(fp))
	{
		printf("\nEnd of file reached.");
	}
	else
	{
		printf("Error reading file.");
	}

	fclose(fp);
```

```
        return 0;
    }
```

Output:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
End of file reached.
```

ferror() function

- The ferror() function takes a single argument, which is a pointer to a FILE object, and returns a non-zero value if an error condition has occurred on the file stream, or zero otherwise.

Example:

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("file.txt", "r");
    int c;

    while ((c = fgetc(fp)) != EOF) {
        putchar(c);
    }

    if (ferror(fp)) {
        printf("Error reading file.\n");
    }

    fclose(fp);
    return 0;
}
```

Output:

If error occurred in reading file then if condition will execute.

Exercise

1. Suppose a file named “Num.txt” contains a list of integers. Write a program to extract the prime numbers only from that file and write them on “Prime.txt” file. (5) [TU 2079]
2. Write a program that simply reads data from a file. (5) [TU 2078]
3. Explain different file I/O functions with example. (5) [TU 2077]
4. Why do we need data files? What are the different file opening modes? Write a program that reads data from a file “input.txt” and writes to “output.txt” file. (10) [TU 2075]
5. Write a program to read and print data stored in a file input.txt. (5) [TU 2074]