MODULE -5

FILE PROCESSING

Basics of File Handling in C

File handling in C is the process in which we create, open, read, write, and close operations on a file. C language provides different functions such as fopen(), fwrite(), fread(), fseek(), fprintf(), etc. to perform input, output, and many different C file operations in our program.

Why do we need File Handling in C?

So far the operations using the C program are done on a prompt/terminal which is not stored anywhere. The output is deleted when the program is closed. But in the software industry, most programs are written to store the information fetched from the program. The use of file handling is exactly what the situation calls for.

In order to understand why file handling is important, let us look at a few features of using files:

- **Reusability:** The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
- **Portability:** Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
- **Efficient:** A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
- **Storage Capacity:** Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

Types of Files in C

A file can be classified into two types based on the way the file stores the data. They are as follows:

- **Text Files**
- **Binary Files**

1. Text Files

A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.

- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with **.txt** file extension.
- Text files can also be used to store the source code.

2. Binary Files

A binary file contains data in **binary form (i.e. 0's and 1's)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

- The binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.
- They are generally stored with **.bin** file extension.

C File Operations

C file operations refer to the different possible operations that we can perform on a file in C such as:

1. Creating a new file – **fopen() with attributes as "a" or "a+" or "w" or "w+"**
2. Opening an existing file – **fopen()**
3. Reading from file – **fscanf() or fgets()**
4. Writing to a file – **fprintf() or fputs()**
5. Moving to a specific location in a file – **fseek(), rewind()**
6. Closing a file – **fclose()**

Opening Modes	Description
r	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the first character in it. If the file cannot be opened fopen( ) returns NULL.
rb	Open for reading in binary mode. If the file does not exist, fopen( ) returns NULL.
w	Open for writing in text mode. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file.
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	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It opens only in the append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file.

File Pointer in C

A file pointer is a reference to a particular position in the opened file. It is used in file handling to perform all file operations such as read, write, close, etc. We use the **FILE** macro to declare the file pointer variable. The FILE macro is defined inside **<stdio.h>** header file.

Syntax of File Pointer

```
FILE* pointer_name;
```

File Pointer is used in almost all the file operations in C.

Open a File in C

For opening a file in C, the fopen() function is used with the filename or file path along with the required access modes.

Syntax of fopen()

```
FILE* fopen(const char *file_name, const char *access_mode);
```

Parameters

- *file_name:* name of the file when present in the same directory as the source file. Otherwise, full path.
- *access_mode:* Specifies for what operation the file is being opened.

Return Value

- If the file is opened successfully, returns a file pointer to it.
- If the file is not opened, then returns NULL.

```
// C Program to illustrate file opening
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // file pointer variable to store the value returned by
        FILE* fptr;
    // opening the file in read mode
    fptr = fopen("filename.txt", "r");
    // checking if the file is opened successfully
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "now exit.");
        exit(0);
    }
    return 0;
}
```

Create a File in C

The fopen() function can not only open a file but also can create a file if it does not exist already. For that, we have to use the modes that allow the creation of a file if not found such as w, w+, wb, wb+, a, a+, ab, and ab+.

```
FILE *fptr;
fptr = fopen("filename.txt", "w");
```

```
// C Program to create a file
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // file pointer
    FILE* fptr;
    // creating file using fopen() access mode "w"
    fptr = fopen("file.txt", "w");

    // checking if the file is created
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "exit now");
        exit(0);
    }
    else {
        printf("The file is created Successfully.");
    }
      return 0;
}
```

Sequential Access File

Reading From a File

The file read operation in C can be performed using functions fscanf() or fgets(). Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file pointer.

Function	Description
fscanf()	Use formatted string and variable arguments list to take input from a file.
fgets()	Input the whole line from the file.
fgetc()	Reads a single character from the file.

Example:

```
FILE * fptr;
fptr = fopen("fileName.txt", "r");
fscanf(fptr, "%s %s %s %d", str1, str2, str3, &year);
char c = fgetc(fptr);
```

Write to a File

The file write operations can be performed by the functions fprintf() and fputs() with similarities to read operations. C programming also provides some other functions that can be used to write data to a file such as:

Function	Description
fprintf()	Similar to printf(), this function use formatted string and varible arguments list to print output to the file.
fputs()	Prints the whole line in the file and a newline at the end.

Example:

```
FILE *fptr ;
fptr = fopen("fileName.txt", "w");
fprintf(fptr, "%s %s %s %d", "We", "are", "in", 2012);
fputc("a", fptr);
```

Closing a File

The fclose() function is used to close the file. After successful file operations, you must always close a file to remove it from the memory.

Syntax of fclose()

```
fclose(file_pointer);
```

where the *file_pointer* is the pointer to the opened file.

Example:

```
FILE *fptr ;
fptr= fopen("fileName.txt", "w");
---------- Some file Operations -------
fclose(fptr);
```

Here is a C program that demonstrates **sequential access file handling** using fprintf(), fscanf(), fopen(), and fclose(). This program allows writing data to a file and then reading it sequentially.

```
#include <stdio.h>
int main() {
    FILE *file;
    char name[50];
    int age;
        // Writing data sequentially to a file
    file = fopen("data.txt", "w");  // Open file in write mode
```

```
    printf("Enter name and age: ");
    scanf("%s %d", name, &age);
    fprintf(file, "%s %d\n", name, age); // Writing data to file
    fclose(file); // Close the file
    // Reading data sequentially from the file
    file = fopen("data.txt", "r");  // Open file in read mode
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    printf("\nReading from file:\n");
    while (fscanf(file, "%s %d", name, &age) != EOF) {
// Reading data from file
        printf("Name: %s, Age: %d\n", name, age);
    }
    fclose(file);
 // Close the file
    return 0;
}
```

Random Access File

Random access file in C enables us to read or write any data in our disk file without reading or writing every piece of data before it.

Random access, ideal for large files, involves functions like ftell(), fseek(), and rewind().

1. The *fseek()* function moves the file position to the desired location.
2. *ftell()* is used to find the position of the file pointer from the starting of the file.
3. *rewind()* is used to move the file pointer to the beginning of the file.

ftell() in C

ftell() in C is used to find out the position of the file pointer in the file with respect to starting of the file.

The syntax of ftell() is:

```
long ftell(FILE *stream);
```

// C program to demonstrate use of ftell()

```
#include <stdio.h>
int main()
{
    /* Opening file in read mode */
    FILE* fp = fopen("file.txt", "r");
    /* Reading first string */
    char string[20];
    fscanf(fp, "%s", string);
    /* Printing position of file pointer */
    printf("%ld", ftell(fp));
    return 0;
}
```

fseek() in C

If we have multiple records inside a file and need to access a particular record that is at a specific position, so we need to loop through all the records before it to get the record. Doing this will waste a lot of memory and operational time. To reduce memory consumption and operational time we can use fseek() which provides an easier way to get to the required data. fseek() function in C seeks the cursor to the given record in the file.

Syntax for fseek():

```
int fseek(FILE *ptr, long int offset, int pos);
```

// C Program to demonstrate the use of fseek() in C

```
#include <stdio.h>

int main()

{

    FILE* fp;

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

    // Moving pointer to end

    fseek(fp, 0, SEEK_END);

    // Printing position of pointer

    printf("%ld", ftell(fp));

    return 0;

}
```

rewind() in C

The rewind() function is used to bring the file pointer to the beginning of the file. It can be used in place of fseek() when you want the file pointer at the start.

Syntax of rewind()

```
rewind (file_pointer);
```

// C program to illustrate the use of rewind

```
#include <stdio.h>

int main()

{

    FILE* fptr;

    fptr = fopen("file.txt", "w+");

    fprintf(fptr, "Girish Kumar\n");

    // using rewind()

    rewind(fptr);

    // reading from file

    char buf[50];

    fscanf(fptr, "%[^\n]s", buf);

    printf("%s", buf);

    return 0;

}
```

Command line arguments in C

Command-line arguments allow users to pass values to a C program when executing it from the terminal. These arguments are handled using main() function parameters:

```
int main(int argc, char *argv[])
```

- **`argc:`** Argument **count** (number of arguments passed, including program name).
- **`argv[]:`** Argument **vector** (array of strings representing the arguments).

Simple Example: Printing Command-Line Arguments

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Total Arguments: %d\n", argc);
        for (int i = 0; i < argc; i++) {
        printf("Argument %d: %s\n", i, argv[i]);
    }
       return 0;
}
```

Example: Adding Two Numbers Passed as Command-Line Arguments

```
#include <stdio.h>
#include <stdlib.h>  // Required for atoi()
int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Enter only 3 values”);
        return 1;  // Exit with error
    }
    int num1 = atoi(argv[1]);  // Convert argument to integer
    int num2 = atoi(argv[2]);
    printf("Sum: %d\n", num1 + num2);
    return 0;
}
```

Assignment Questions

1. what is the need for file handling in C and explain file opening modes
2. Explain the following sequential access file handling functions in C
    - fopen()
    - fscanf()
    - fprintf()
    - fclose()
3. Explain the following random access handling functions in C
    - ftell()
    - fseek()
    - rewind()
4. Explain command line arguments in C