UNIT-V

Basics of File Handling in C, File Handling Functions –Defining, Opening a file,Closing a file,I/O Operations on files, Command line Arguments

***********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.

If you're interested in working with files and integrating them into larger data structures, the **C Programming Course Online with Data Structures** provides detailed lessons on file management in C.
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.

*********** File Handling Functions –Defining**

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.

Functions that We Use for File Handling in C

We can use a variety of functions in order to open a file, read it, write more data, create a new file, close or delete a file, search for a file, etc. These are known as file handling operators in C.

Here's a list of functions that allow you to do so:

Description of Function	**Function in Use**
used to open an existing file or a new file	fopen()
writing data into an available file	fprintf()
reading the data available in a file	fscanf()
writing any character into the program file	fputc()
reading the character from an available file	fgetc()
used to close the program file	fclose()
used to set the file pointer to the intended file position	fseek()
writing an integer into an available file	fputw()

used to read an integer from the given file	fgetw()
used for reading the current position of a file	ftell()
sets an intended file pointer to the file’s beginning itself	rewind()

Note: It is important to know that we must declare a file-type pointer when we are working with various files in a program. This helps establish direct communication between a program and the files.

*********** I/O Operations on files**

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()**

The highlighted text mentions the C function used to perform the file operations.
File Pointer is used in almost all the file operations in C.

1 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.

File opening modes in C

File opening modes or access modes specify the allowed operations on the file to be opened. They are passed as an argument to the fopen() function. Some of the commonly used file access modes are listed below:

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

Opening Modes	**Description**
	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.
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+	Searches file. It is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the first character in it. Returns NULL, if unable to open the file.
rb+	Open for both reading and writing in binary mode. If the file does not exist, fopen( ) returns NULL.
w+	Searches file. 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 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+	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 the file in both reading and append mode. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open the file.
ab+	Open for both reading and appending in binary mode. If the file does not exist, it will be created.

As given above, if you want to perform operations on a binary file, then you have to append ‘b’ at the last. For example, instead of “w”, you have to use “wb”, instead of “a+” you have to use “a+b”.

Example of Opening a File

```
// C Program to illustrate file opening
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // file pointer variable to store the value returned by
    // fopen
    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;
}
```

Output

```
The file is not opened. The program will now exit.
```

The file is not opened because it does not exist in the source directory. But the fopen() function is also capable of creating a file if it does not exist. It is shown below

**************Closing a file – fclose()**

In C language, fclose() is a standard library function used to close a file that was previously opened using fopen(). This function is the itegral part of the file handling in C which allows the users to free the memory occupied by the file once all the operations are done on it.
In this article, we will learn about the fclose() function, its syntax and how it behaves in different scenarios.

Syntax of fclose()

The syntax of fclose is straigforward:

```
fclose(file_pointer);
```

Parameters

- file_pointer: A pointer to a FILE object that identifies the stream to be closed.

Return Value

This function only returns two values:

- **0**: When the file is successfully closed.
- **EOF**: When an error occurred while closing the file.

Example of fclose() in C

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

```
int main() {
  FILE *file = fopen("example1.txt", "w");
  if (file == NULL) {
      perror("Error opening file");
      return 1;
  }
  fprintf(file, "Hello, World!\n");
  if (fclose(file) == EOF) {
      perror("Error closing file");
      return 1;
  }
  printf("File closed successfully.\n");
  return 0;
}
```
Output

```
File closed successfully.
```

********************Command Line Arguments in C**

The most important function of C is the main() function. It is mostly defined with a return type of int and without parameters.

```
int main() {
   ...
}
```

We can also give command-line arguments in C. Command-line arguments are the values given after the name of the program in the command-line shell of Operating Systems. Command-line arguments are handled by the main() function of a C program.

To pass command-line arguments, we typically define main() with two arguments: the first argument is the **number of command-line arguments** and the second is a **list of command-ine arguments.**

Syntax

```
int main(int argc, char *argv[]) { /* ... */ }
          or
int main(int argc, char **argv) { /* ... */ }
```

Here,

- **argc (ARGument Count)** is an integer variable that stores the number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, the value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non-negative.
- **argv (ARGument Vector)** is an array of character pointers listing all the arguments.
- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

Example
The below example illustrates the printing of command line arguments.
C

```
// C program named mainreturn.c to demonstrate the working
// of command line arguement
#include <stdio.h>

// defining main with arguments
int main(int argc, char* argv[])
{
    printf("You have entered %d arguments:\n", argc);

    for (int i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

Output

```
You have entered 4 arguments:
./main
geeks
for
geeks
```

for Terminal Input

```
$ g++ mainreturn.cpp -o main $ ./main geeks for geeks
```

Note: *Other platform-dependent formats are also allowed by the C standards; for example, Unix (though not POSIX.1) and Microsoft Visual C++ have a third argument giving the program's environment, otherwise accessible through getenv in stdlib.h. Refer C program to print environment variables for details.*

Properties of Command Line Arguments in C

1. They are passed to the main() function.
2. They are parameters/arguments supplied to the program when it is invoked.
3. They are used to control programs from outside instead of hard coding those values inside the code.
4. argv[argc] is a NULL pointer.
5. argv[0] holds the name of the program.
6. argv[1] points to the first command line argument and argv[argc-1] points to the last argument.

Note: *You pass all the command line arguments separated by a space, but if the argument itself has a space, then you can pass such arguments by putting them inside double quotes "" or single quotes ''.*