C Programming Short Notes

Part 10

File Handling

- File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program.
- File handling in C involves performing operations such as **Creation of the new file & Opening an existing file, Reading from the file & Writing to the file.**
- The standard library in C provides functions for file handling.
- You need to include the standard input/output library <stdio.h>.

Modes in c

❖ **"r" - Read mode:**
- Opens the file for **reading only**.
- **The file must exist; otherwise, fopen() will return NULL.**

❖ **"w" - Write mode:**
- Opens the file for **writing only.**
- If the file exists, its **contents are overwritten.**
- If the file does not exist, it is created.
- **The file pointer is placed at the beginning of the file.**

❖ **"a" - Append mode:**
- Opens the file for **writing only.**
- **Data is appended to the end of the file.**
- If the file does not exist, it is created.
- **The file pointer is placed at the end of the file.**

❖ **"r+" - Read/update mode:**
- Opens the file for **both reading and writing.**
- **The file must exist; otherwise, fopen() will return NULL.**
- The file pointer is placed at the beginning of the file.

❖ **"w+" - Write/update mode:**
- Opens the file for **both reading and writing.**
- If the file exists, its contents are overwritten.
- If the file does not exist, it is created.
- The file pointer is placed at the beginning of the file.

❖ **"a+" - Append/update mode:**
- Opens the file for both reading and writing.
- Data is appended to the end of the file.
- If the file does not exist, it is created.
- The file pointer is placed at the end of the file.

1. `FILE`

- `FILE` is a **data type** in C used to **represent a file stream.**
- It's a structure defined in the **`<stdio.h>` header file**.

Syntax:

```
FILE *file_pointer;
```

Example:

```
#include <stdio.h>
FILE *fptr;
```

2. `fopen()`

- `fopen()` is used **to open a file.** It returns a **pointer to a `FILE` structure**.

Syntax:

```
FILE *fopen(const char *filename, const char *mode);
```

Example:

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

3. `fclose()`

- `fclose()` is used to **close an opened file.**

Syntax:

```
int fclose(FILE *stream);
```

Example:

```
fclose(fptr);
```

4. `fputs()`

- `fputs()` is used to **write a string to a file.**

Syntax:

```
int fputs(const char *str, FILE *stream);
```

Example:

```
fputs("Hello, world!", fptr);
```

5. `fgets()`

- `fgets()` is used to **read a line from a file.**

Syntax:

```
char *fgets(char *str, int n, FILE *stream);
```

Example:

```
char buffer[255];
fgets(buffer, 255, fptr);
```

6. `fputc()`

- `fputc()` is used to **write a character to a file.**

Syntax:

```
int fputc(int character, FILE *stream);
```

Example:

```
fputc('A', fptr);
```

7. `fgetc()`

- `fgetc()` is used to **read a character from a file.**

Syntax:

```
int fgetc(FILE *stream);
```

Example:

```
char ch;
ch = fgetc(fptr);
```

8. `fprintf()`

- `fprintf()` is used to **write formatted data to a file.**

Syntax:

```
int fprintf(FILE *stream, const char *format, ...);
```

Example:

```
fprintf(fptr, "Integer: %d, Float: %f\n", num, fnum);
```

9. `fscanf()`

- `fscanf()` is used to **read formatted data from a file.**

Syntax:

```
int fscanf(FILE *stream, const char *format, ...);
```

Example:

```
fscanf(fptr, "%d %f", &num, &fnum);
```

10. `fread`:

- `fread` is used to **read binary data from a file**. It takes four parameters:

1. Pointer to a memory block where the read data will be stored.
2. Size in bytes of each element to be read.
3. Number of elements to read.
4. File pointer pointing to the file from which the data will be read.

Syntax:

```
fread(void *ptr, size, nmemb, FILE *stream);
```

Example:

```
fread(buffer, 4, 10, file);
```

11. `fwrite`:

- `fwrite` is used to **write binary data to a file.** It takes four parameters:

1. Pointer to the data to be written.
2. Size in bytes of each element to be written.
3. Number of elements to write.
4. File pointer pointing to the file to which the data will be written.

Syntax:

```
fwrite(const void *ptr, size, nmemb, FILE *stream);
```

Example:

```
fwrite(data, sizeof(int), 5, file);
```

End of File(EOF) and Beginning of File (BOF)

EOF (End of File):

- It is a condition in a computer operating system where no more data can be read from a data source (such as a file or input stream).
- In C programming, the EOF constant is used to indicate the end of input when reading from files or streams.
- It is typically returned by functions like `fgetc()` or `fgets()` when they encounter the end of the file.
- `EOF` is used in the `while` loop condition to check whether the end of the file has been reached while reading characters from the file using `fgetc()`.

BOF (Beginning of File):

- It refers to the starting point of a file or data stream.
- When you open a file for reading in a program, the read pointer initially points to the beginning of the file.
- You can move this pointer to access different parts of the file.
- There's no direct use of `BOF` in the code you provided. However, when you open a file using `fopen()` in read mode ("r"), the file pointer is positioned at the beginning of the file, indicating the BOF.

Program to check File is present or not

```
#include<stdio.h>
int main()
{
   FILE *ptr;
   ptr = fopen("rsm.txt","r");
   if(ptr!=NULL)
   {
      printf("File open successful");
   }
   else
   {
      printf("File could not be opened.");
   }
}
```

Output:

File could not be opened.

Creating New File:

```
#include<stdio.h>
int main() {
   FILE *fptr;
   fptr = fopen("rsm.txt", "w");
   if (fptr == NULL) {
      printf("Error in opening the file.\n");
      return 1;
   }
   printf("File opened successfully.\n");
   fclose(fptr);
   return 0;
}
```

Output:

```
File opened successfully.
```

Reading a File:

```
#include<stdio.h>
int main() {
   FILE *fptr;
   fptr = fopen("rsm.txt", "r");
   if (fptr == NULL) {
      printf("Error in opening the file.\n");
      return 1;
   }
   printf("File opened successfully.\n");
   fclose(fptr);
   return 0;
}
```

Output:

```
File opened successfully.
```

Writing a Character in File:

```
#include <stdio.h>
int main() {
   FILE *filePtr;
   char c;
   c='A';
   filePtr = fopen("rsm.txt", "w");
   if (filePtr == NULL) {
      printf("Error opening the file!\n");
      return 1;
   }
   fputc(c,filePtr);
   fclose(filePtr);
   printf("Data written to rsm.txt successfully.\n");
   return 0;
}
```

Output:

```
Data written to rsm.txt successfully.
```

Reading a Character in File:

```
#include <stdio.h>
int main() {
   FILE *filePtr;
   char ch;
   filePtr = fopen("rsm.txt", "r");
   if (filePtr == NULL) {
      printf("Error opening the file!\n");
      return 1;
   }
   while ((ch = fgetc(filePtr)) != EOF) {
      putchar(ch);
   }
   fclose(filePtr);
   return 0;
}
```

Output:

```
A
```

Writing a String in File:

```
#include <stdio.h>
int main() {
   FILE *filePtr;
   filePtr = fopen("rsm.txt", "w");
   if (filePtr == NULL) {
      printf("Error opening the file!\n");
      return 1;
   }
   fputs("Welcome to Rasim Class",filePtr);
   fclose(filePtr);
   printf("Data written to rsmt.txt successfully.\n");
   return 0;
}
```

Output:

Data written to rsm.txt successfully.

Reading a String in File:

```
#include <stdio.h>
int main() {
   FILE *filePtr;
   char line[100];
   filePtr = fopen("rsm.txt", "r");
   if (filePtr == NULL) {
      printf("Error opening the file!\n");
      return 1;
   }
   while (fgets(line, sizeof(line), filePtr) != NULL) {
      printf("%s", line);
   }
   fclose(filePtr);
   return 0;
}
```

Output:

Welcome to Rasim Class

Writing Data/ Value in File:

```
#include <stdio.h>
int main() {
   FILE *file_pointer;
   int num = 10;
   file_pointer = fopen("rsm.txt", "w");
   if (file_pointer == NULL) {
      printf("File could not be opened.\n");
      return 1;
   }
   fprintf(file_pointer, "%d", num);
   fclose(file_pointer);
   printf("Data written to rsm.txt successfully.\n");
   return 0;
}
```

Output:

```
Data written to rsm.txt successfully.
```

Reading a Data/ Value in File:

```
#include <stdio.h>
int main() {
   FILE *file_pointer;
   int num;
   file_pointer = fopen("rsm.txt", "r");
   if (file_pointer == NULL) {
      printf("File could not be opened.\n");
      return 1;
   }
   fscanf(file_pointer, "%d", &num);
   printf("The number read from file: %d\n", num);
   fclose(file_pointer);
   return 0;
}
```

Output:

```
The number read from file: 10
```

Write Binary Data to a File:

```
#include <stdio.h>
int main() {
   FILE *file;
   int data[] = {1, 2, 3, 4, 5};
   file = fopen("rsm.bin", "wb");
   if (file == NULL) {
      perror("Error opening file");
      return 1;
   }
   size_t elements_written = fwrite(data, sizeof(int), 5, file);
   fclose(file);
   printf("Data written to rsm.bin successfully.\n");
   return 0;
}
```

Output:

```
Data written to rsm.bin successfully.
```

Read Binary Data to a File:

```
#include <stdio.h>
int main() {
   FILE *file;
   char buffer[100];
   file = fopen("rsm.bin", "rb");
   if (file == NULL) {
      perror("Error opening file");
      return 1;
   }
   size_t elements_read = fread(buffer, 4, 10, file);
   printf("Read %zu elements\n", elements_read);
   printf("Elements read from file: ");
   for (int i = 0; i < elements_read*4; i++) {
         if(buffer[i]!=0){
             printf("%d ", buffer[i]);
         }
   }
   printf("\n");

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

Output:

Read 5 elements
Elements read from file: 1 2 3 4 5