DATE: LAB REPORT NO.: **10** SET: **A**
TITLE OF THE PROGRAM: **File Handling Basic**

OBJECTIVES

- Understand File Operations: Gain familiarity with fundamental file operations in C, including file opening (**fopen()**), writing (**fprintf()** and **putc()**), reading (**fscanf()**, **getc()**, and **fgetc()),** and closing (**fclose()).**
- Practice Writing to Files: Learn how to write different types of data (strings, characters, integers) to files using appropriate functions (**fprintf()**, **putc()**, **putw()**).
- Practice Reading from Files: Learn how to read data from files using various methods (**fscanf()**, **getc()**, **getw()).**
- Handle Multiple Records: Write programs that handle multiple records, such as student information (name and age), and write these records to a file.
- Continuous Input Handling: Implement programs that continuously accept user input until terminated (**e.g., Ctrl+Z**), and store this input as records in a file.
- Understanding **EOF**: Understand the usage of **EOF (End-of-File)** in file handling operations to control input loops.
- File Error Handling: Implement error handling for file operations to manage situations where files cannot be opened or accessed.
- Memory Efficiency: Appreciate the efficiency of file handling in managing data storage, especially when dealing with large datasets or structured data records.

REQUIREMENTS

1. C Compiler (e.g., GCC)
2. Computer System
3. IDE or Text Editor
4. OS compatible with the software

THEORY

File handling in C involves performing operations on files, such as reading from or writing to them. Files are essential for storing and retrieving data persistently, making them crucial for many applications.

File Pointers

In C, file operations are managed using file pointers (**FILE *).** These pointers are used to keep track of the current position within the file during read and write operations.

Steps in File Handling

1. Opening a File: Before reading from or writing to a file, it must be opened using **fopen()** function:

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

- "**mode**" specifies the type of operations allowed ("**r**" for reading, "**w**" for writing, "**a**" for appending, "r+" for reading and writing, etc.).
- If the file cannot be opened, **fopen()** returns **NULL.**

2. Writing to a File: Data can be written to a file using functions like **fprintf()**, **fputc()**, or **fwrite()**:

```
fprintf(fptr, "Hello, World!");

char ch = 'A';
fputc(ch, fptr);
```

- **fprintf()** formats data before writing.
- **fputc()** writes a single character.
- **fwrite()** is used for writing blocks of data.

3. Reading from a File: Data can be read from a file using functions like **fscanf()**, **fgetc()**, or **fread()**:

```
fscanf(fptr, "%s", buffer);

char ch = fgetc(fptr);
```

- **fscanf()** reads formatted data.
- **fgetc()** reads a single character.
- **fread()** reads blocks of data.

4. Closing a File: After finishing operations on a file, it should be closed using **fclose():**

```
fclose(fptr);
```

PROCEDURE (Program Code, Comment, and Output)

1. C Program to Implement File Handling by writing “Welcome to Nepal”

Program Code:

```
#include <stdio.h>
int main()
{
   FILE *fp;
   char str[] = "Welcome to Nepal";

   fp = fopen("nepal.txt", "w");
   if (fp == NULL) {
      printf("Error opening file!\n");
      return 1;
   }

   fprintf(fp, "%s", str);
   fclose(fp);

   printf("Data written to file successfully!\n");
   return 0;
}
```

Output

```
Data written to file successfully!
```

Explanation:

- Opens a file "**nepal.txt**" in write mode using **fopen().**
- Checks if the file is opened successfully.
- Writes the string "Welcome to Nepal" to the file using **fprintf()**.
- Closes the file using **fclose().**

2. Writing Characters to a File using putc()

```
#include <stdio.h>

int main() {
   FILE  *fp;
   char ch;

   fp = fopen("data-1.txt", "w");
```

```
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    printf("Enter characters (Ctrl+Z to end):\n");
    ch = getchar();
    while (ch != EOF) {
        putc(ch, fp);
        ch = getchar();
    }

    fclose(fp);
    printf("Data written to file successfully!\n");

    return 0;
}
```

Output

```
Enter characters (Ctrl+Z to end):
This is a test.
Data written to file successfully!
```

Explanation

- Opens a file "**data-1.txt**" in write mode using **fopen().**
- Checks if the file is opened successfully.
- Reads characters from the user using **getchar()** until **Ctrl+Z (EOF)** is entered.
- Writes each character to the file using **putc().**
- Closes the file using **fclose().**

3. Reading Characters from a File using getc()

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char ch;

    fp = fopen("data-1.txt", "r");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
```

```
    printf("Contents of the file:\n");
    ch = getc(fp);
    while (ch != EOF) {
        printf("%c", ch);
        ch = getc(fp);
    }

    fclose(fp);
    printf("\nFile reading complete!\n");

    return 0;
}
```

Output

```
Contents of the file:
This is a test.
File reading complete!
```

Explanation

- Opens the file "**data-1.txt**" in read mode using **fopen().**
- Checks if the file is opened successfully.
- Reads characters from the file using **getc()** until **EOF** is encountered.
- Prints each character to the console.
- Closes the file using **fclose().**

4. Writing and Reading Integers to/from a File using putw() and getw()

```
#include <stdio.h>

int main() {
    FILE  *fp;
    int x, n, i;

    printf("How many numbers do you want to enter: ");
    scanf("%d", &n);

    fp = fopen("numbers.txt", "w");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    for (i = 0; i < n; i++) {
```

```
        printf("Enter number %d: ", i + 1);
        scanf("%d", &x);
        putw(x, fp);
    }

    fclose(fp);
    fp = fopen("numbers.txt", "r");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    printf("Numbers from file:\n");
    while ((x = getw(fp)) != EOF) {
        printf("%d\n", x);
    }
    fclose(fp);
    return 0;
}
```

Output

```
How many numbers do you want to enter: 3
Enter number 1: 12
Enter number 2: 45
Enter number 3: 78
Numbers from file:
12
45
78
```

Explanation

- Prompts the user for the number of integers to enter.
- Writes each integer to the file "**numbers.txt**" using **putw().**
- Reads integers from the file using **getw()** until **EOF**.
- Prints each integer to the console.

CONCLUSION

In conclusion, file handling is an essential aspect of programming that allows for efficient data

storage and retrieval. By using file operations such as opening, reading, writing, and closing files, we can manage large datasets and structured records effectively. Understanding the usage of file pointers, handling multiple records, and implementing error handling are crucial for developing robust programs. Additionally, the ability to control file operations using EOF (End-of-File) ensures that input loops are properly terminated, enhancing the reliability and functionality of the programs. Through these exercises, we've gained practical experience in managing data in a persistent and organized manner, which is fundamental to many real-world applications.