Day_10 C-Programming

(RECAP OF PREVIOUS DAY)
File Types, File operations, File pointer, File opening modes, File handling functions, Command Line Arguments, File handling through command line argument, Record I/O in files, Preprocessor directives: Macros and File inclusion

File Types in C

Files in C are used for permanent storage of data. C supports two types of files:

1. **Text Files**:
    - Contain data in human-readable format.
    - Example: `.txt`, `.csv`
2. **Binary Files**:
    - Contain data in machine-readable format (0s and 1s).
    - Example: `.bin`, `.dat`

File Operations in C

C provides a set of operations to handle files:

1. **Create a file**: Create a new file for storing data.
2. **Open a file**: Open an existing file for reading or writing.
3. **Read from a file**: Retrieve data from the file.
4. **Write to a file**: Write data to the file.
5. **Close a file**: Close the file to free resources.

File Pointer

- A file pointer is a pointer to a `FILE` structure, used to control file operations.

Syntax:

```
FILE *fp;
fp = fopen("example.txt", "r");
if (fp == NULL) {
    printf("File could not be opened\n");
}
fclose(fp);
```

File Opening Modes

C provides multiple modes for opening files:

Mode	Description
`"r"`	Open for reading. File must exist.
`"w"`	Open for writing. Creates/truncates.
`"a"`	Open for appending. Creates if absent.
`"r+"`	Open for reading and writing.
`"w+"`	Open for reading and writing.
`"a+"`	Open for reading and appending.
`"rb"`	Open for reading in binary mode.
`"wb"`	Open for writing in binary mode.

File Handling Functions

1. **Opening a file**:
    - `FILE *fopen(const char *filename, const char *mode);`

Example:

```
FILE *fp = fopen("data.txt", "r");
```

2. **Closing a file**:
    - `int fclose(FILE *fp);`

Example:

```
fclose(fp);
```

3. **Reading from a file**:
    - `fgetc()`: Read a single character.
    - `fgets()`: Read a string.
    - `fread()`: Read binary data.

Example:

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

4. **Writing to a file**:
    - `fputc()`: Write a single character.
    - `fputs()`: Write a string.
    - `fwrite()`: Write binary data.

Example:

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

5. **Positioning Functions**:
    - `fseek()`: Move the file pointer to the desired location.
    - `ftell()`: tells the current position of the file pointer.
    - `rewind()`: Set the file pointer to the start.
      We will see examples later in this session

Some File Programs

Example:
Program that writes content into a file and then reads it back to display on the screen

```
#include <stdio.h>

int main() {
    FILE *file;
    char content[] = "This is a sample text written into the file.\nThis is the second line.\n";

    // Open the file for writing
    file = fopen("sample.txt", "w");
    if (file == NULL) {
        printf("Error opening the file for writing.\n");
        return 1;
    }

    // Write content into the file
    fprintf(file, "%s", content);

    // Close the file after writing
    fclose(file);
    printf("Content has been written to the file.\n");

    // Open the file for reading
    file = fopen("sample.txt", "r");
    if (file == NULL) {
        printf("Error opening the file for reading.\n");
        return 1;
    }

    // Display the content of the file
    printf("\nReading the content from the file:\n");
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch);  // Print each character to the screen
    }

    // Close the file after reading
    fclose(file);

    return 0;
}
```

Example:
Write a program that takes 30 numbers as input, writes them into a file, then separates even and odd numbers into two different files named **`EVEN.txt`** and **`ODD.txt`**

```
#include <stdio.h>

int main() {
    FILE *inputFile, *evenFile, *oddFile;
    int numbers[30];
    int i;

    // Open the input file for writing
    inputFile = fopen("numbers.txt", "w");
    if (inputFile == NULL) {
        printf("Error opening numbers.txt file.\n");
        return 1;
    }

    // Taking 30 numbers as input from the user
    printf("Enter 30 numbers:\n");
    for (i = 0; i < 30; i++) {
        scanf("%d", &numbers[i]);
        fprintf(inputFile, "%d\n", numbers[i]);  // Write each number to the file
    }

    // Close the input file after writing
    fclose(inputFile);

    // Open the EVEN and ODD files for writing
    evenFile = fopen("EVEN.txt", "w");
    oddFile = fopen("ODD.txt", "w");
    if (evenFile == NULL || oddFile == NULL) {
        printf("Error opening EVEN.txt or ODD.txt file.\n");
        return 1;
    }

    // Separate the even and odd numbers and write them to respective files
    for (i = 0; i < 30; i++) {
        if (numbers[i] % 2 == 0) {
            fprintf(evenFile, "%d\n", numbers[i]);
        } else {
            fprintf(oddFile, "%d\n", numbers[i]);
        }
    }

    // Close the EVEN and ODD files after writing
    fclose(evenFile);
    fclose(oddFile);
```

```
    printf("The numbers have been separated into EVEN.txt and ODD.txt files.\n");

    return 0;
}
```

Example:
example of fseek, ftell, and rewind

```
#include <stdio.h>

int main() {
    // Open a file for writing
    FILE *file = fopen("example.txt", "w");

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

    // Write some data to the file
    fprintf(file, "Hello, World!\nThis is an example of fseek, ftell, and rewind.");

    // Move the file pointer 6 bytes from the beginning
    fseek(file, 6, SEEK_SET);
    printf("File pointer moved to position: %ld\n", ftell(file)); // Prints 6

    // Move the file pointer 5 bytes ahead from the current position
    fseek(file, 5, SEEK_CUR);
    printf("File pointer moved to position: %ld\n", ftell(file)); // Prints 11

    // Move the file pointer 5 bytes from the end
    fseek(file, -5, SEEK_END);
    printf("File pointer moved to position: %ld\n", ftell(file)); // Prints position 47 (based on file
size)

    // Rewind the file pointer to the start
    rewind(file);
    printf("File pointer after rewind: %ld\n", ftell(file)); // Prints 0

    // Close the file
    fclose(file);

    return 0;
}
```

Command Line Arguments

- Command-line arguments allow passing inputs to the program at runtime.

Syntax:

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

- **`argc`: Number of arguments.**
- **`argv`: Array of arguments.**

Example:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Program name: %s\n", argv[0]);
    if (argc > 1) {
        printf("Argument: %s\n", argv[1]);
    }
    return 0;
}
```

To Run it from command line

- Lets name it as MyProg.c
- Compile it from command line
- Run it without argument” MyProg.c
    - Output: Program name:MyProg.c
- Run it with argument: MyProg.c Hello
    - Output: Program name:MyProg.c
      Argument: Hello

File Handling Through Command Line Arguments

Example program to copy the contents of one file to another:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s source_file target_file\n", argv[0]);
        return 1;
```

```
    }

    FILE *src = fopen(argv[1], "r");
    FILE *dest = fopen(argv[2], "w");

    if (src == NULL || dest == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    char ch;
    while ((ch = fgetc(src)) != EOF) {
        fputc(ch, dest);
    }

    fclose(src);
    fclose(dest);
    printf("File copied successfully.\n");
    return 0;
}
```

Record I/O in Files

- Used to store and retrieve structured data (e.g., records of employees).

Example:

```
 #include <stdio.h>

struct Employee {
    int id;
    char name[50];
    float salary;
};

int main() {
    struct Employee e = {1, "Alice", 50000.0};

    FILE *fp = fopen("employee.dat", "wb");
    fwrite(&e, sizeof(struct Employee), 1, fp);
    fclose(fp);
```

```
    struct Employee e_read;
    fp = fopen("employee.dat", "rb");
    fread(&e_read, sizeof(struct Employee), 1, fp);
    fclose(fp);

    printf("ID: %d, Name: %s, Salary: %.2f\n", e_read.id, e_read.name, e_read.salary);
    return 0;
}
```

Preprocessor Directives

a. Macros:

- Macro is a piece of code in a program that is replaced by the value of the macro.
- Macros are constants or functions defined using the `#define` directive.

Example:

```
#define PI 3.14159
#define SQUARE(x) ((x) * (x))

int main() {
    printf("PI: %.2f\n", PI);
    printf("Square of 5: %d\n", SQUARE(5));
    return 0;
}
```

Program: Calculate Area of a Circle Using a Macro

```
#include <stdio.h>

#define PI 3.14159
#define AREA_OF_CIRCLE(radius) (PI * (radius) * (radius))

int main() {
    float radius;

    printf("Enter the radius of the circle: ");
    scanf("%f", &radius);

    float area = AREA_OF_CIRCLE(radius);
```

```
    printf("The area of the circle with radius %.2f is: %.2f\n", radius, area);

    return 0;
}
```

b. File Inclusion:

- Used to include header files or other files into the program.

Two main types of include file directives in C:

1. #include <filename.h>:
The use of this syntax is to include system header file in C program. It tells the compiler to look for or find the file in the standard system directory.

2. #include “filename”:
This syntax is used to add a header file. It tells the compiler to first look for the file in the current directory and then look in the system directory.

```
#include <stdio.h>
#include "myheader.h" // User-defined header file
```

Programs to practice(HW)

1. File Types and File Operations

- **Program 1**: Create a text file and write data into it using `fwrite()`.
- **Program 2**: Read data from a file using `fread()` and display it on the screen.
- **Program 3**: Append data to an existing file without overwriting using `fopen()` in append mode.

2. File Pointer and File Operations

- **Program 4**: Move the file pointer using `fseek()`, `ftell()`, and `rewind()`, and display the current file pointer position at various stages.
- **Program 5**: Implement a program that uses `fseek()` to search for a specific word or pattern in a file.
- **Program 6**: Read and display file content in reverse order using `fseek()`.

3. File Opening Modes

- **Program 7**: Open a file in different modes (`"r"`, `"w"`, `"a"`, `"r+"`, `"w+"`, `"a+"`) and handle errors for each mode.
- **Program 8**: Create a program that handles errors gracefully when trying to open a file that doesn't exist in `"r"` mode, or when trying to write in `"r"` mode.

4. File Handling Functions

- **Program 9**: Use `fopen()`, `fclose()`, `fgetc()`, and `fputc()` to read and write one character at a time in a file.
- **Program 10**: Implement a program that reads from a file line-by-line using `fgets()` and writes the content to another file.
- **Program 11**: Write a program that reads and writes a structured record (e.g., `struct` type) to/from a binary file using `fwrite()` and `fread()`.

5. Command Line Arguments

- **Program 12**: Write a program that takes two numbers from the command line, adds them, and displays the sum.
- **Program 13**: Create a program that takes a list of strings as command-line arguments and prints each string.
- **Program 14**: Implement a program that takes a file name as a command-line argument, reads the content of that file, and prints it.

6. File Handling through Command Line Arguments

- **Program 15**: Write a program that accepts a file name as a command-line argument, opens the file, reads the content, and displays it.
- **Program 16**: Create a program that accepts file names from the command line, and copies content from the source file to the destination file.
- **Program 17**: Implement a program that accepts a file name and a word from the command line and counts how many times the word appears in the file.

7. Record I/O in Files

- **Program 18**: Implement a program to write student records (name, roll number, marks) into a file using `struct` and binary file operations (`fwrite()`, `fread()`).
- **Program 19**: Write a program to read and display student records from a file, and update a student record (e.g., modify marks) in the file.
- **Program 20**: Create a program that writes multiple records (students) into a file, then reads the file and displays the records in tabular format.

8. Preprocessor Directives: Macros and File Inclusion

- **Program 21**: Implement a program that uses macros to find the maximum of two numbers.
- **Program 22**: Create a program with a header file (`.h`) that contains function prototypes for file operations, and implement the functions in a separate `.c` file.
- **Program 23**: Write a program that includes an external header file with preprocessor directives to define constants and use them in the program.
- **Program 24**: Implement a program that defines a macro for a factorial and uses it to compute the factorial of a number.

9. (Mixed Topics)

- **Program 25**: Write a program that takes a list of students' names and grades from the user, writes them to a file, and later reads and sorts the data in ascending order based on grades.
- **Program 26**: Create a program to count the number of lines, words, and characters in a file using `fgetc()` or `fgets()`.
- **Program 27**: Write a program that reads integers from a file, computes their average, and writes the result to a new file.