Class Note for Session 20: Lab on File Handling

Introduction

Session 20 is a lab session designed to provide hands-on practice with file handling in C, building on the theoretical concepts introduced in Session 21 of the "Elementary Programming in C" book. File handling allows programs to read from and write to files, enabling persistent data storage. This session focuses on practical exercises involving text and binary streams, file pointers, and file operations, reinforcing the skills needed to manipulate files effectively.

1. Introduction to Streams

Explanation

A **stream** in C is an abstraction that represents a sequence of data, typically associated with a file or input/output device. Streams can be **text streams** (human-readable data, such as text files) or **binary streams** (raw data, such as images or serialized structures). The C standard library provides functions like `fopen()`, `fclose()`, `fread()`, and `fwrite()` to work with streams.

Example

```c
#include <stdio.h>

int main() {
    FILE *fp;
    char data[] = "Hello, File Handling!";

    // Open file in write mode (text stream)
    fp = fopen("example.txt", "w");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    // Write to file
    fprintf(fp, "%s", data);

    // Close file
    fclose(fp);
    printf("Data written to file successfully.\n");

    return 0;
}
```

Output:

```
Data written to file successfully.
```

File Content (example.txt):

```
Hello, File Handling!
```

Classwork

1. Write a program to create a text file named `greeting.txt` and write the string "Welcome to C Programming!" to it. Check if the file was opened successfully.
2. Modify the program to append the string "Let's learn file handling!" to the same file using the append mode (`"a"`).

2. Text Streams and Binary Streams

Explanation

- **Text Streams**: Data is stored as human-readable characters, with newline characters (`\n`) translated to the system's line-ending convention (e.g., `\r\n` on Windows). Functions like `fprintf()` and `fscanf()` are used.
- **Binary Streams**: Data is stored in its raw binary format, with no translation. Functions like `fwrite()` and `fread()` are used, suitable for non-text data like structures or images.

Example

```c
#include <stdio.h>

struct Record {
    int id;
    float value;
};

int main() {
    FILE *fp;
    struct Record data = {1, 99.99};

    // Write to binary file
    fp = fopen("data.bin", "wb");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fwrite(&data, sizeof(struct Record), 1, fp);
    fclose(fp);

    // Read from binary file
    struct Record read_data;
    fp = fopen("data.bin", "rb");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fread(&read_data, sizeof(struct Record), 1, fp);
    printf("Read: ID = %d, Value = %.2f\n", read_data.id, read_data.value);
    fclose(fp);

    return 0;
}
```

Output:

```
Read: ID = 1, Value = 99.99
```

Classwork

1. Write a program to store an array of 3 integers in a binary file named `numbers.bin` using `fwrite()`. Read the data back and print it.
2. Create a program to write a string to a text file and a structure (with an integer and a float) to a binary file. Read and display both.

3. File Pointers and Various File Functions

Explanation

A **file pointer** (`FILE *`) is used to reference an open file. Key file functions include:

- `fopen(mode)` : Opens a file in modes like `"r"` (read), `"w"` (write), `"a"` (append), `"rb"` , `"wb"` , etc.
- `fclose()` : Closes a file, releasing resources.
- `fscanf()` / `fprintf()` : Read/write formatted data in text mode.
- `fread()` / `fwrite()` : Read/write raw data in binary mode.
- `fgetc()` / `fputc()` : Read/write single characters.

Example

```c
#include <stdio.h>

int main() {
    FILE *fp;

    // Write characters to a file
    fp = fopen("chars.txt", "w");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fputc('A', fp);
    fputc('B', fp);
    fputc('C', fp);
    fclose(fp);

    // Read characters from the file
    fp = fopen("chars.txt", "r");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    printf("File contents: ");
    char ch;
    while ((ch = fgetc(fp)) != EOF) {
        printf("%c", ch);
    }
    fclose(fp);

    return 0;
}
```

Output:

```
File contents: ABC
```

Classwork

1. Write a program to write 5 single characters (A to E) to a file using `fputc()`. Read and print them using `fgetc()`.
2. Create a program to read a text file containing a list of names (one per line) and print them to the console using `fscanf()`.

4. Current File Pointer

Explanation

The **current file pointer** indicates the position in the file where the next read or write operation will occur. Functions like `fseek()` and `ftell()` manipulate or query this position:

- `fseek(fp, offset, origin)`: Moves the file pointer to a specific position. Origins are `SEEK_SET` (start), `SEEK_CUR` (current), or `SEEK_END` (end).
- `ftell(fp)`: Returns the current position.
- `rewind(fp)`: Resets the file pointer to the beginning.

Example

```c
#include <stdio.h>

int main() {
    FILE *fp;

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

    fprintf(fp, "File Handling Example");

    // Get current position
    long pos = ftell(fp);
    printf("Current file position: %ld\n", pos);

    // Move to start
    fseek(fp, 0, SEEK_SET);
    printf("After fseek to start, position: %ld\n", ftell(fp));

    // Read and print first 4 characters
    char buffer[5];
    fread(buffer, sizeof(char), 4, fp);
    buffer[4] = '\0';
    printf("Read: %s\n", buffer);

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

Output:

```
Current file position: 20
After fseek to start, position: 0
Read: File
```

Classwork

1. Write a program to write a string to a file, move the file pointer to the 5th character using `fseek()`, and read the rest of the file.
2. Create a program to append data to a file, then use `fseek()` to read the file from the beginning and print its contents.

5. Command-Line Arguments

Explanation

Command-line arguments allow a program to accept input parameters when executed. They are passed to `main()` as `int argc` (argument count) and `char *argv[]` (argument vector). `argc` indicates the number of arguments, and `argv` is an array of strings containing the arguments (including the program name).

Example

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "w");
    if (fp == NULL) {
        printf("Error opening file %s!\n", argv[1]);
        return 1;
    }

    fprintf(fp, "Written using command-line argument: %s", argv[1]);
    fclose(fp);
    printf("Data written to %s\n", argv[1]);

    return 0;
}
```

Command to run:

```
./program output.txt
```

Output:

```
Data written to output.txt
```

File Content (output.txt):

```
Written using command-line argument: output.txt
```

Classwork

1. Write a program that accepts a filename as a command-line argument and writes "Command-line file handling" to that file.
2. Create a program that reads a text file specified via a command-line argument and prints its contents to the console.

General Classwork

1. Write a program to create a text file containing student names and marks (input by the user). Read the file and display students with marks above 80.
2. Create a program to store an array of 3 `Student` structures (with `name` and `marks`) in a binary file. Read the file and print all students' details.
3. Write a program that accepts a filename via a command-line argument, writes 10 integers to it (in binary), and then reads and prints them in reverse order using `fseek()`.
4. Create a program to append user-input text to an existing file. Use `fseek()` to read the file from the beginning and verify the appended content.

Objective Questions

1. **What is a stream in C?**
   a) A data type for arrays

b) A sequence of data associated with a file or device

c) A pointer to a structure

d) A function for sorting data

Answer: b) A sequence of data associated with a file or device

2. **Which function is used to open a file in C?**

   a) `fclose()`

   b) `fopen()`

   c) `fread()`

   d) `fseek()`

   Answer: b) `fopen()`

3. **What is the difference between text and binary streams?**

   a) Text streams store data as raw bytes, binary streams as characters

   b) Text streams translate newline characters, binary streams do not

   c) Binary streams are used for text files only

   d) Text streams are faster than binary streams

   Answer: b) Text streams translate newline characters, binary streams do not

4. **Which function moves the file pointer to a specific position?**

   a) `ftell()`

   b) `fseek()`

   c) `fclose()`

   d) `fputc()`

   Answer: b) `fseek()`

5. **What does `argc` represent in `main(int argc, char *argv[])`?**

   a) The number of command-line arguments

   b) The name of the program

   c) The size of the file

   d) The file pointer

   Answer: a) The number of command-line arguments

6. **What happens if `fopen()` fails to open a file?**

   a) It returns `NULL`

   b) It crashes the program

   c) It returns `EOF`

   d) It creates a new file

   Answer: a) It returns `NULL`

7. **What is the output of the following code?**

```c
#include <stdio.h>
int main() {
    FILE *fp = fopen("test.txt", "w");
    if (fp == NULL) return 1;
    fprintf(fp, "Test");
    fseek(fp, 0, SEEK_SET);
    fputc('X', fp);
    fclose(fp);
    return 0;
}
```

File Content (test.txt):

a) Test

b) Xest

c) X

d) TestX

Answer: b) Xest

8. **Which function reads a single character from a file?**

a) `fscanf()`

b) `fread()`

c) `fgetc()`

d) `fwrite()`

Answer: c) `fgetc()`