Class Note for Session 21: File Handling

Introduction

Session 21 of the "Elementary Programming in C" course introduces file handling in C, a critical skill for persistent data storage and retrieval. This session covers streams (text and binary), file pointers, file operations, and command-line arguments. By the end, students will be able to read from and write to files, manipulate file pointers, and handle command-line inputs, enabling robust program functionality.

1. Introduction to Streams

Explanation

A **stream** in C is an abstraction representing a sequence of data, typically linked to a file or input/output device. Streams provide a uniform way to handle data flow. C supports:

- **Text Streams**: Data is stored as human-readable characters, with newline (`\n`) translated to the system's line-ending convention (e.g., `\r\n` on Windows).
- **Binary Streams**: Data is stored in raw binary format without translation, suitable for non-text data like structures or images.
  Key functions include `fopen()` to open a stream and `fclose()` to close it.

Example

```c
#include <stdio.h>

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

    // Open file in write mode (text stream)
    fp = fopen("welcome.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 welcome.txt\n");

    return 0;
}
```

Output:

```
Data written to welcome.txt
```

File Content (welcome.txt):

```
Welcome to File Handling!
```

Classwork

1. Write a program to create a text file named `intro.txt` and write "Learning C is fun!" to it. Check for file opening errors.
2. Modify the program to append "Keep practicing!" to `intro.txt` using append mode (`"a"`).

2. Text Streams and Binary Streams

Explanation

- **Text Streams**: Store data as readable text, with automatic translation of newline characters. Use functions like `fprintf()` for writing and `fscanf()` for reading.
- **Binary Streams**: Store data in its raw form, ideal for structures, images, or other non-text data. Use `fwrite()` for writing and `fread()` for reading.
  The mode in `fopen()` determines the stream type: `"r"`, `"w"`, `"a"` for text; `"rb"`, `"wb"`, `"ab"` for binary.

Example

```c
#include <stdio.h>

struct Item {
    int id;
    float price;
};

int main() {
    FILE *fp;
    struct Item item = {101, 29.99};

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

    fwrite(&item, sizeof(struct Item), 1, fp);
    fclose(fp);

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

    fread(&read_item, sizeof(struct Item), 1, fp);
    printf("Item ID: %d, Price: $%.2f\n", read_item.id, read_item.price);
    fclose(fp);

    return 0;
}
```

Output:

```
Item ID: 101, Price: $29.99
```

Classwork

1. Write a program to store an array of 4 floats in a binary file named `values.bin` using `fwrite()`. Read and print the values.
2. Create a program to write a string to a text file (`data.txt`) and a structure (containing `name` and `age`) to a binary file (`person.bin`). Read and display both.

3. File Pointers and Various File Functions

Explanation

A **file pointer** (`FILE *`) references an open file. Key file functions include:

- `fopen(mode)` : Opens a file in specified mode.
- `fclose()` : Closes a file, freeing resources.
- `fscanf()` / `fprintf()` : Read/write formatted text.
- `fread()` / `fwrite()` : Read/write binary data.
- `fgetc()` / `fputc()` : Read/write single characters.
- `fgets()` / `fputs()` : Read/write strings.

  Error checking (e.g., `fp == NULL`) is crucial to handle file operation failures.

Example

```c
#include <stdio.h>

int main() {
    FILE *fp;

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

    fputc('X', fp);
    fputc('Y', fp);
    fputc('Z', fp);
    fclose(fp);

    // Read characters from file
    fp = fopen("letters.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: XYZ
```

Classwork

1. Write a program to write the characters 'A' to 'D' to a file (`chars.txt`) using `fputc()` . Read and print them using `fgetc()` .
2. Create a program to write a list of 3 names to a text file (`names.txt`) using `fputs()` . Read and display them using `fgets()` .

4. Current File Pointer

Explanation

The **current file pointer** tracks the position in a file where the next read or write occurs. Functions to manipulate it include:

- `fseek(fp, offset, origin)` : Moves the file pointer to a position. Origins are `SEEK_SET` (start), `SEEK_CUR` (current), `SEEK_END` (end).
- `ftell(fp)` : Returns the current position (in bytes).
- `rewind(fp)` : Resets the file pointer to the beginning.
  These functions enable random access within files.

Example

```c
#include <stdio.h>

int main() {
    FILE *fp;

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

    fprintf(fp, "This is a test string.");

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

    // Move to start and overwrite
    fseek(fp, 0, SEEK_SET);
    fputc('X', fp);

    // Reset to start and read
    rewind(fp);
    char buffer[50];
    fgets(buffer, 50, fp);
    printf("File contents: %s\n", buffer);

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

Output:

```
Current position: 21
File contents: Xhis is a test string.
```

Classwork

1. Write a program to write "File Handling in C" to a file, move the file pointer to the 6th byte using `fseek()`, and overwrite with "TEST". Read and print the updated file.
2. Create a program to write 10 integers to a file, use `fseek()` to read the 5th integer, and print it.

5. Command-Line Arguments

Explanation

Command-line arguments allow a program to accept inputs when executed. They are passed to `main()` as:

- `int argc` : Number of arguments (including program name).
- `char *argv[]` : Array of strings containing the arguments.

  For example, running `./program file.txt` sets `argc = 2`, `argv[0] = "./program"`, `argv[1] = "file.txt"`.

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, "File created via command-line: %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):

```
File created via command-line: output.txt
```

Classwork

1. Write a program that accepts a filename via a command-line argument and writes "Command-line file created!" to it.

2. Create a program that reads a text file specified via a command-line argument and prints its contents.

General Classwork

1. Write a program to create a text file containing 5 student names and grades (input by the user). Read the file and display students with grades above 75.
2. Create a program to store an array of 3 `Employee` structures (with `name`, `id`, and `salary`) in a binary file. Read and print all employee details.
3. Write a program that accepts a filename via a command-line argument, writes 5 floats to it (in binary), and uses `fseek()` to read and print the 3rd float.
4. Create a program to append a user-input string to a text file. Use `rewind()` to read and verify the entire file contents.

Objective Questions

1. **What is the role of a stream in C file handling?**
   a) To define a data type
   b) To represent a sequence of data for I/O operations
   c) To sort data in files
   d) To allocate memory
   Answer: b) To represent a sequence of data for I/O operations
2. **Which mode is used to open a file for writing in binary mode?**
   a) `"r"`
   b) `"w"`
   c) `"wb"`
   d) `"a"`
   Answer: c) `"wb"`
3. **What does `fclose()` do?**
   a) Opens a file
   b) Closes a file and frees resources
   c) Reads data from a file
   d) Moves the file pointer
   Answer: b) Closes a file and frees resources

4. **Which function reads a string from a file?**

   a) `fgetc()`

   b) `fgets()`

   c) `fread()`

   d) `fputc()`

   Answer: b) `fgets()`

5. **What does `fseek(fp, 0, SEEK_END)` do?**

   a) Moves the file pointer to the start

   b) Moves the file pointer to the end

   c) Closes the file

   d) Reads the current position

   Answer: b) Moves the file pointer to the end

6. **What is `argv[0]` in `main(int argc, char *argv[])`?**

   a) The first command-line argument

   b) The program name

   c) The file pointer

   d) The number of arguments

   Answer: b) The program name

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;
    fputs("Hello", fp);
    rewind(fp);
    fputc('X', fp);
    fclose(fp);
    return 0;
}
```

   File Content (test.txt):

   a) Hello

   b) Xello

   c) X

   d) HelloX

   Answer: b) Xello

8. **What happens if `fopen()` cannot open a file?**

   a) It returns `EOF`

b) It crashes the program

c) It returns `NULL`

d) It creates a new file

Answer: c) It returns `NULL`

This class note for Session 21 provides detailed explanations, practical examples, and exercises to reinforce file handling concepts in C. Students should complete the classwork and review the objective questions to solidify their understanding.