One-Week-C-Primer (Day 6)

Day 6: File Handling and Preprocessors

Note : Refer to the Attached Document along with this for the codes

Files and File Handling in C

In C programming, files are used for reading and writing data to and from external storage, such as a hard disk, USB drive, or network location. Files are essential for data persistence and interaction with the external world.

File Operations:

File operations involve creating, opening, reading, writing, and closing files. To perform file operations, you need to use file pointers (`FILE*`) provided by the standard library (`<stdio.h>`).

1. Opening a File:

```c
#include <stdio.h>

int main() {
    FILE *filePointer;

    // Open a file in write mode
    filePointer = fopen("example.txt", "w");

    // Check if the file is opened successfully
    if (filePointer == NULL) {
        printf("Error opening the file.n");
        return 1;
    }

    // Perform file operations here...

    // Close the file
    fclose(filePointer);

    return 0;
}
```

2. Writing to a File:

```c
#include <stdio.h>

int main() {
    FILE *filePointer;

    // Open a file in write mode
    filePointer = fopen("example.txt", "w");

    // Check if the file is opened successfully
    if (filePointer == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }

    // Writing data to the file
    fprintf(filePointer, "Hello, World!\n");
    fprintf(filePointer, "This is a sample file.\n");

    // Close the file
    fclose(filePointer);

    return 0;
}
```

3. Reading from a File:

```c
#include <stdio.h>

int main() {
    FILE *filePointer;
    char buffer[100];

    // Open a file in read mode
    filePointer = fopen("example.txt", "r");

    // Check if the file is opened successfully
    if (filePointer == NULL) {
        printf("Error opening the file.n");
        return 1;
    }

    // Read data from the file line by line
    while (fgets(buffer, sizeof(buffer), filePointer) != NULL) {
        printf("%s", buffer);
    }

    // Close the file
    fclose(filePointer);
    return 0;
}
```

4. Using File Pointers for File Operations:

File pointers are essential for performing various file operations like reading, writing, and moving within the file.

```
#include <stdio.h>

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

    // Open a file in read mode
    filePointer = fopen("example.txt", "r");

    // Check if the file is opened successfully
    if (filePointer == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }

    // Read and print one character at a time
    while ((ch = fgetc(filePointer)) != EOF) {
        printf("%c", ch);
    }

    // Close the file
    fclose(filePointer);

    return 0;
}
```

5. Error Handling for File Operations:

Always check for errors during file operations to ensure proper execution and avoid unexpected behavior.

```
#include <stdio.h>

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

    // Open a file in read mode
    filePointer = fopen("example.txt", "r");
    // Check if the file is opened successfully
    if (filePointer == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }

    // Read and print one character at a time
    while ((ch = fgetc(filePointer)) != EOF) {
        printf("%c", ch);
    }

    // Check for read errors
    if (ferror(filePointer)) {
        printf("Error reading the file.n");
        fclose(filePointer);
        return 1;
    }

    // Close the file
    fclose(filePointer);

    return 0;
}
```

File handling in C is a critical aspect of programming, enabling interaction with external data and storage. Understanding file pointers, file operations, and error handling is essential for working with files effectively and ensuring the integrity of data read from and written to files. Always remember to close files after use to free up resources and prevent memory leaks.

Preprocessor Directives

Introduction to Preprocessor Directives:

Preprocessor directives in C are instructions to the compiler that begin with the `#` symbol. They are processed by the preprocessor before the actual compilation of the code starts. Preprocessor directives are used to modify the source code before it is compiled, enabling conditional compilation, file inclusion, macro definitions, and other preprocessing tasks.

Commonly Used Preprocessor Directives in C:

1. **`#include` Directive:**
    - Used to include header files in the source code.
    - Provides access to functions and declarations defined in the included header files.

- Syntax: `#include <header_file>` (for system-level headers) or `#include "header_file"` (for user-defined headers).

2. **`#define` Directive:**

    - Used to define constants and macros.
    - Replaces occurrences of defined names with the specified value during preprocessing.
    - Syntax: `#define MACRO_NAME value`.

3. **`#ifdef`, `#ifndef`, `#else`, and `#endif` Directives:**

    - Used for conditional compilation based on whether a macro is defined or not.
    - Syntax:

    ```
    #ifdef MACRO_NAME
        // Code to be included if MACRO_NAME is defined
    #else
        // Code to be included if MACRO_NAME is not defined
    #endif
    ```

4. **`#if`, `#elif`, and `#endif` Directives:**

    - Used for more complex conditional compilation based on integer expressions.
    - Syntax:

    ```
    #if expression
        // Code to be included if expression is true (non-zero)
    #elif expression
        // Code to be included if the previous condition is false and this expression
    is true (non-zero)
    #else
        // Code to be included if all previous conditions are false
    #endif
    ```

5. **`#undef` Directive:**

    - Used to undefine a previously defined macro.
    - Syntax: `#undef MACRO_NAME`.

6. **`#pragma` Directive:**

- Used to provide specific instructions to the compiler.
- The behavior of `#pragma` is compiler-specific, and different compilers may interpret it differently.

Example Usage:

```
#include <stdio.h>

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

#ifndef DEBUG
    #define DEBUG // Uncomment to enable debugging
#endif

int main() {
    #ifdef DEBUG
        printf("Debugging mode is enabled.\n");
    #else
        printf("Debugging mode is not enabled.\n");
    #endif

    printf("Area of a circle with radius 5: %.2f\n", PI * SQUARE(5));

    return 0;
}
```

Preprocessor directives are powerful tools that facilitate conditional compilation and code modification based on predefined macros and conditions. They enable code reusability, customization, and platform-specific handling in C programs. However, use preprocessor directives judiciously, as excessive use can lead to code maintenance and readability issues.
