CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 25

File Management in C (3)

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
www.mijanrahman.com

Contents

File Management in C

fseek and rewind Functions

- In C, fseek() and rewind() functions are used for file handling operations, particularly for positioning the file pointer within a file.
- **fseek():** This function is used to set the file position indicator for the specified stream (FILE pointer) to a new position. It allows us to move the file pointer to a specific byte offset within the file.
- **rewind():** This function is used to move the file pointer to the beginning of the file. It is equivalent to calling fseek() with an offset of 0 bytes from the beginning of the file.
- Both fseek() and rewind() are defined in the standard C library <stdio.h>.

fseek and rewind Functions

- **fseek():** The syntax for fseek() is:

```
int fseek(FILE *stream, long int offset, int origin);
```

  - stream: Pointer to a FILE object, which specifies the file to set the file position indicator.
  - offset: Number of bytes to offset from the position specified by origin.
  - origin: It specifies the position from where the offset is added. It can take one of three values: SEEK_SET (beginning of the file), SEEK_CUR (current position of the file pointer), and SEEK_END (end of the file).

fseek and rewind Functions

- **fseek():**
- The return value of fseek() is zero if successful, and nonzero otherwise.

- For example:

```
FILE *fp;
fp = fopen("example.txt", "r");
fseek(fp, 10, SEEK_SET); // Moves the file pointer to 10th byte from the beginning of the file
```

fseek and rewind Functions

- **rewind():** The syntax for rewind() is:

```
void rewind(FILE *stream);
```

  - stream: Pointer to a FILE object, which specifies the file to rewind.

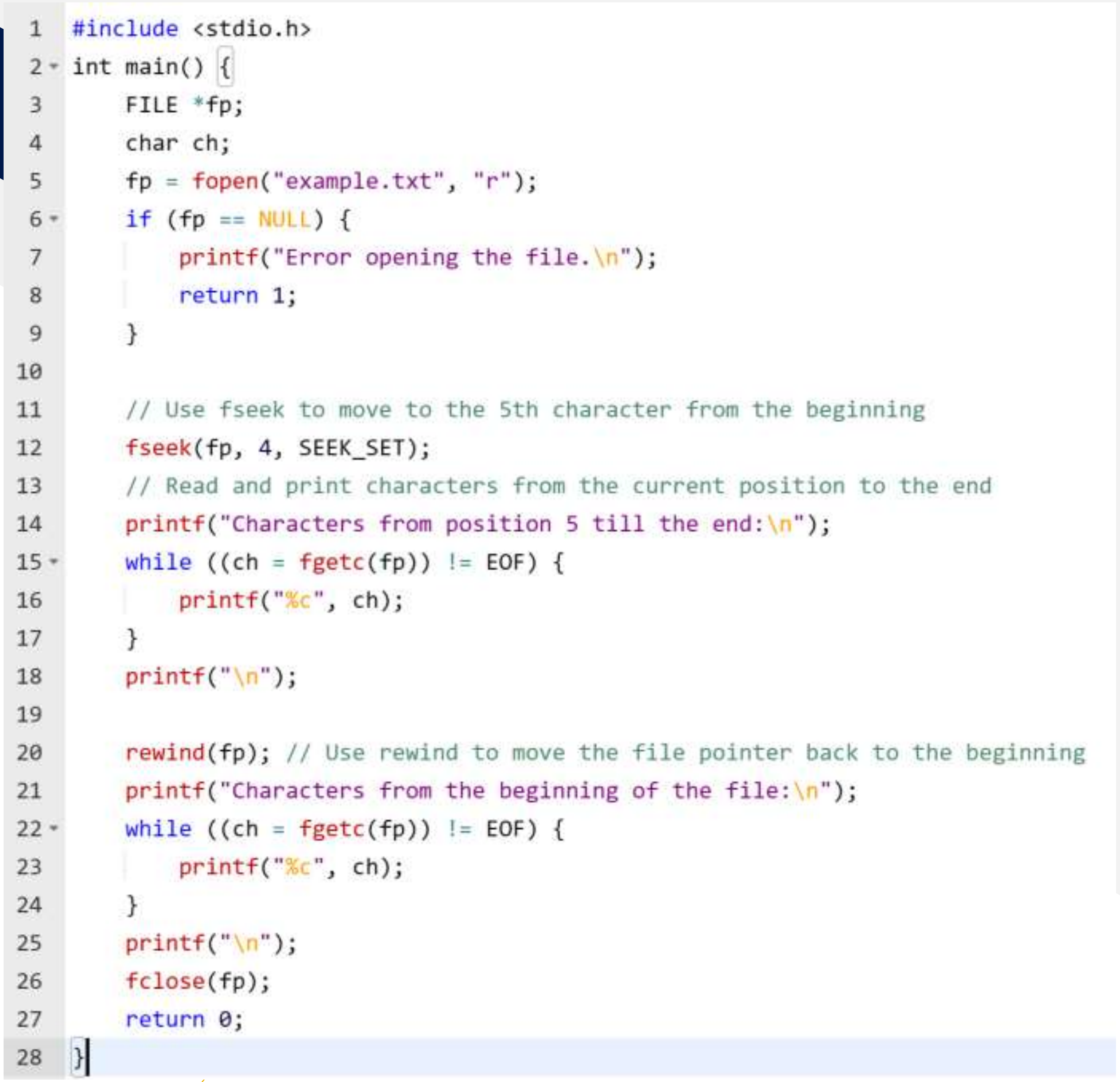
- For example:

```
FILE *fp;
fp = fopen("example.txt", "r");
rewind(fp); // Moves the file pointer to the beginning of the file
```

fseek and rewind

- Here's a simple C program that demonstrates the usage of fseek() and rewind() functions:

- Make sure you have a file named example.txt in the same directory as the program, containing some text, to run this program.
- This program will first move the file pointer to the 5th character from the beginning using fseek(), then it will read and print characters till the end of the file. After that, it uses rewind() to move the file pointer back to the beginning of the file and reads and prints characters from the beginning.



```c
#include <stdio.h>
int main() {
    FILE *fp;
    char ch;
    fp = fopen("example.txt", "r");
    if (fp == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }

    // Use fseek to move to the 5th character from the beginning
    fseek(fp, 4, SEEK_SET);
    // Read and print characters from the current position to the end
    printf("Characters from position 5 till the end:\n");
    while ((ch = fgetc(fp)) != EOF) {
        printf("%c", ch);
    }
    printf("\n");

    rewind(fp); // Use rewind to move the file pointer back to the beginning
    printf("Characters from the beginning of the file:\n");
    while ((ch = fgetc(fp)) != EOF) {
        printf("%c", ch);
    }
    printf("\n");
    fclose(fp);
    return 0;
}
```

Ftell Function

- In C, the ftell() function is used to determine the current position of the file pointer within a file. It returns the current value of the file position indicator associated with the specified stream (FILE pointer).
- The syntax for ftell() function is:

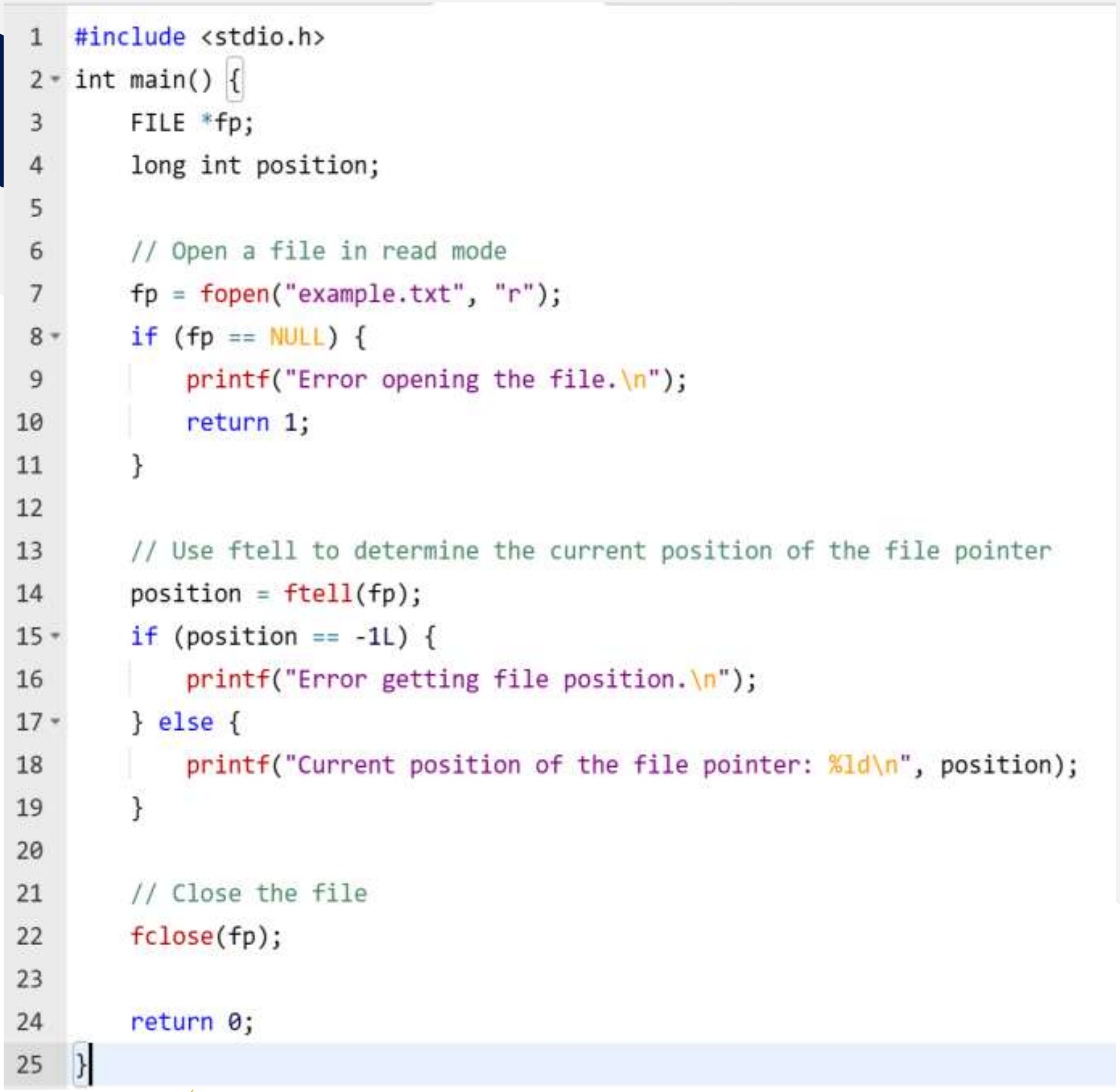
```
long int ftell(FILE *stream);
```

  - stream: Pointer to a FILE object, which specifies the file whose current position is to be determined.
- The return value of ftell() is the current position of the file pointer if successful, and -1L if an error occurs.

Ftell Function

- Here's a simple example demonstrating the usage of ftell():

- This program opens a file named example.txt, then uses ftell() to determine the current position of the file pointer within the file. It prints the position to the console. Finally, it closes the file.



```c
#include <stdio.h>
int main() {
    FILE *fp;
    long int position;

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

    // Use ftell to determine the current position of the file pointer
    position = ftell(fp);
    if (position == -1L) {
        printf("Error getting file position.\n");
    } else {
        printf("Current position of the file pointer: %ld\n", position);
    }

    // Close the file
    fclose(fp);

    return 0;
}
```

Error Handling during File Operations

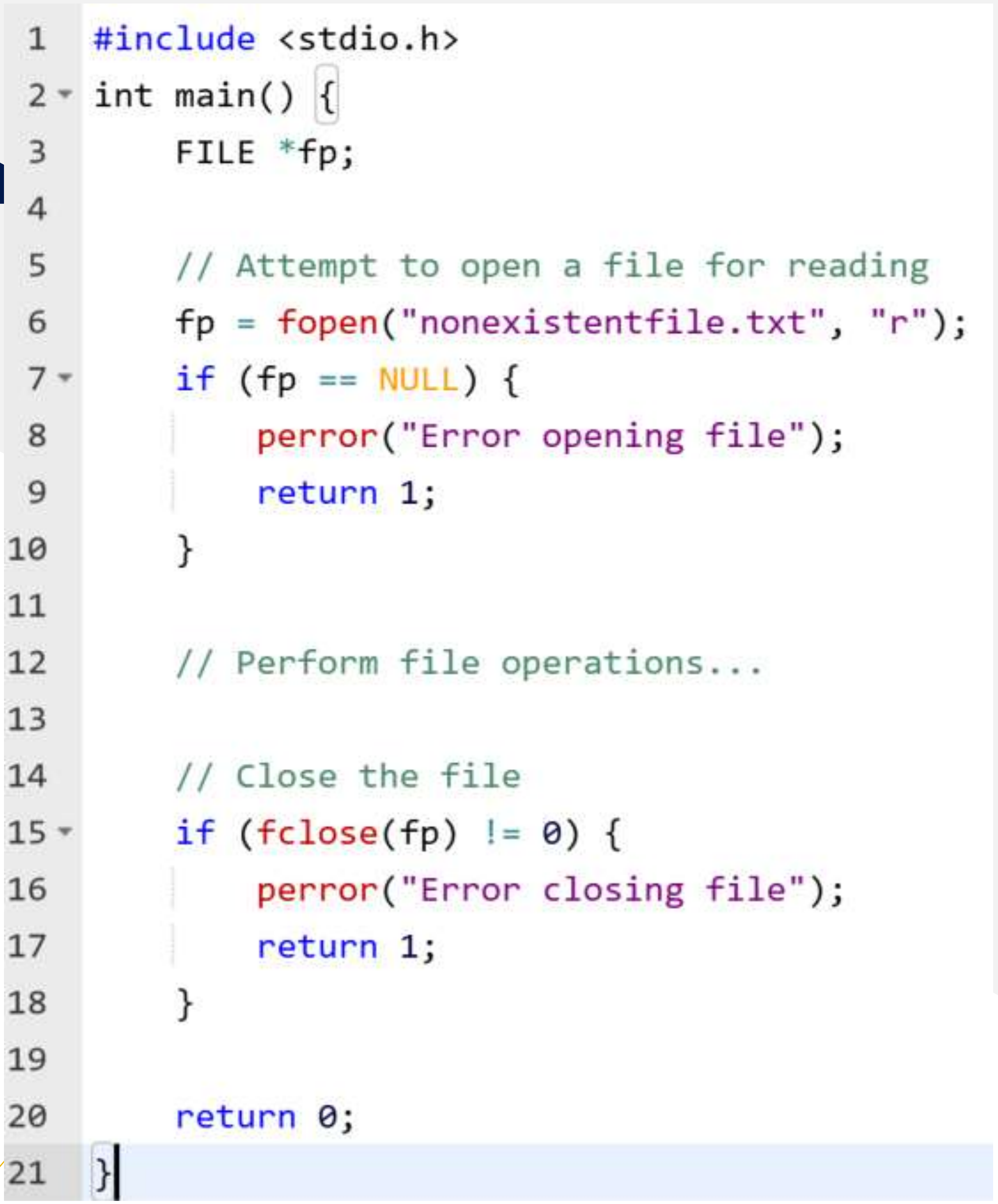
- Error handling during file operations in C is crucial to ensure the robustness and reliability of the program, especially when dealing with file I/O.
- Here's how we can handle errors effectively:
    - **Check Return Values:** Most file-related functions return a special value (NULL or -1, for example) to indicate an error condition. Always check these return values after calling file-related functions to detect errors.
    - **Print Error Messages:** When an error occurs, print a descriptive error message to the console or log file. This helps in debugging and provides valuable information for understanding what went wrong.

Error Handling during File Operations

- **Close Files Properly:** If a file operation fails, close any open files before exiting the program. This ensures that system resources are released properly and prevents potential resource leaks.
- **Use perror():** The perror() function can be used to print a descriptive error message corresponding to the last error encountered during a file operation. It provides additional information about the error, such as the error code and a description.
- **Handle Specific Errors:** Different file operations may encounter different types of errors. Handle specific error conditions appropriately. For example, if a file cannot be opened due to a permissions error, notify the user and possibly prompt for corrective action.

Error Handling: Example

- Here's an example demonstrating error handling during file operations:

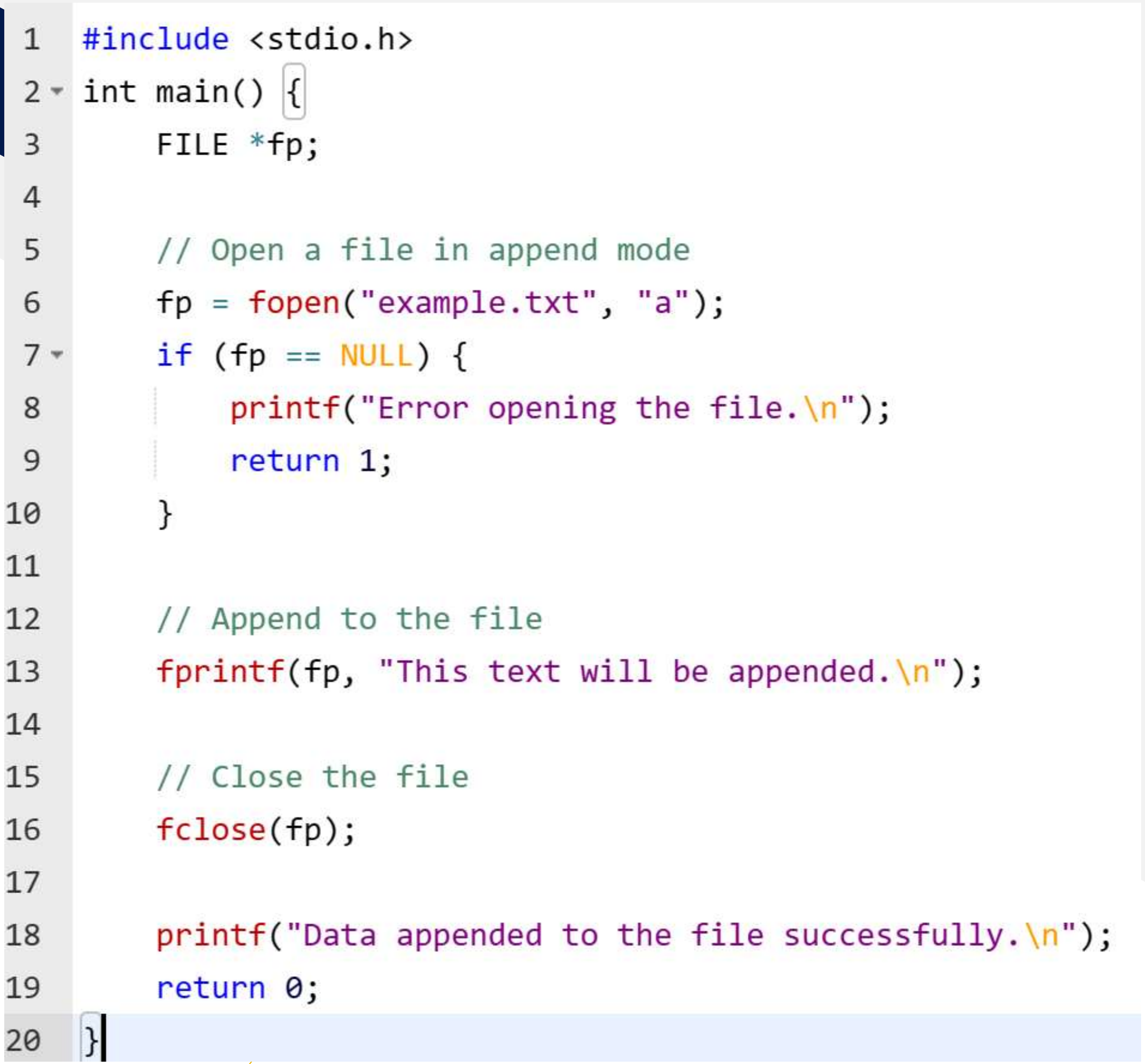


```c
#include <stdio.h>
int main() {
    FILE *fp;

    // Attempt to open a file for reading
    fp = fopen("nonexistentfile.txt", "r");
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }

    // Perform file operations...

    // Close the file
    if (fclose(fp) != 0) {
        perror("Error closing file");
        return 1;
    }

    return 0;
}
```

- In this example, if the file "nonexistentfile.txt" does not exist or cannot be opened for reading, fopen() will return NULL, indicating an error. We then use perror() to print a descriptive error message. Similarly, when closing the file with fclose(), if an error occurs, perror() is used to print the error message.

Program Examples

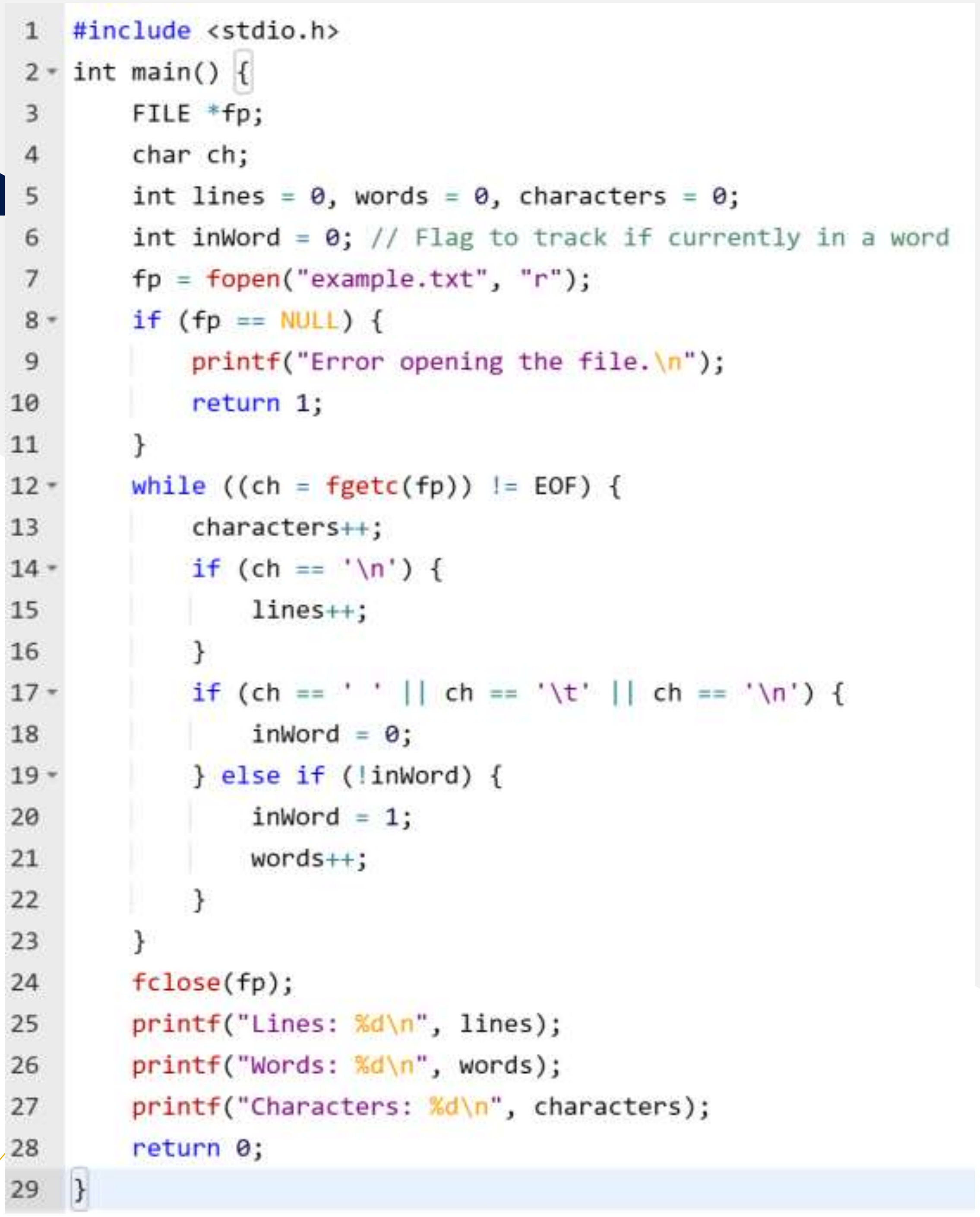
- Appending to a File:



```c
#include <stdio.h>
int main() {
    FILE *fp;

    // Open a file in append mode
    fp = fopen("example.txt", "a");
    if (fp == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }

    // Append to the file
    fprintf(fp, "This text will be appended.\n");

    // Close the file
    fclose(fp);

    printf("Data appended to the file successfully.\n");
    return 0;
}
```

Program Examples

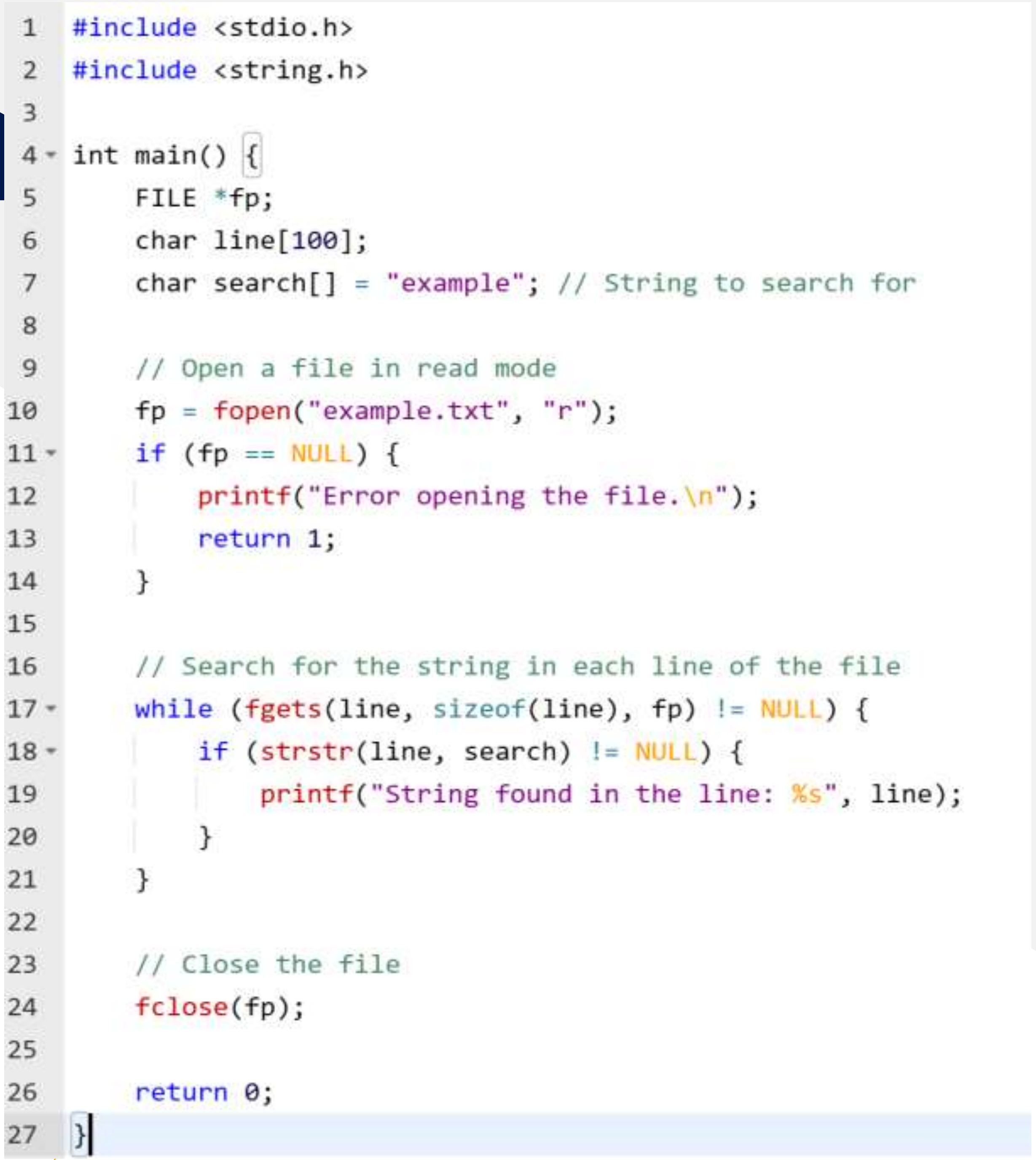
- Counting Lines, Words, and Characters in a File:



```c
#include <stdio.h>
int main() {
    FILE *fp;
    char ch;
    int lines = 0, words = 0, characters = 0;
    int inWord = 0; // Flag to track if currently in a word
    fp = fopen("example.txt", "r");
    if (fp == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }
    while ((ch = fgetc(fp)) != EOF) {
        characters++;
        if (ch == '\n') {
            lines++;
        }
        if (ch == ' ' || ch == '\t' || ch == '\n') {
            inWord = 0;
        } else if (!inWord) {
            inWord = 1;
            words++;
        }
    }
    fclose(fp);
    printf("Lines: %d\n", lines);
    printf("Words: %d\n", words);
    printf("Characters: %d\n", characters);
    return 0;
}
```

Program Examples

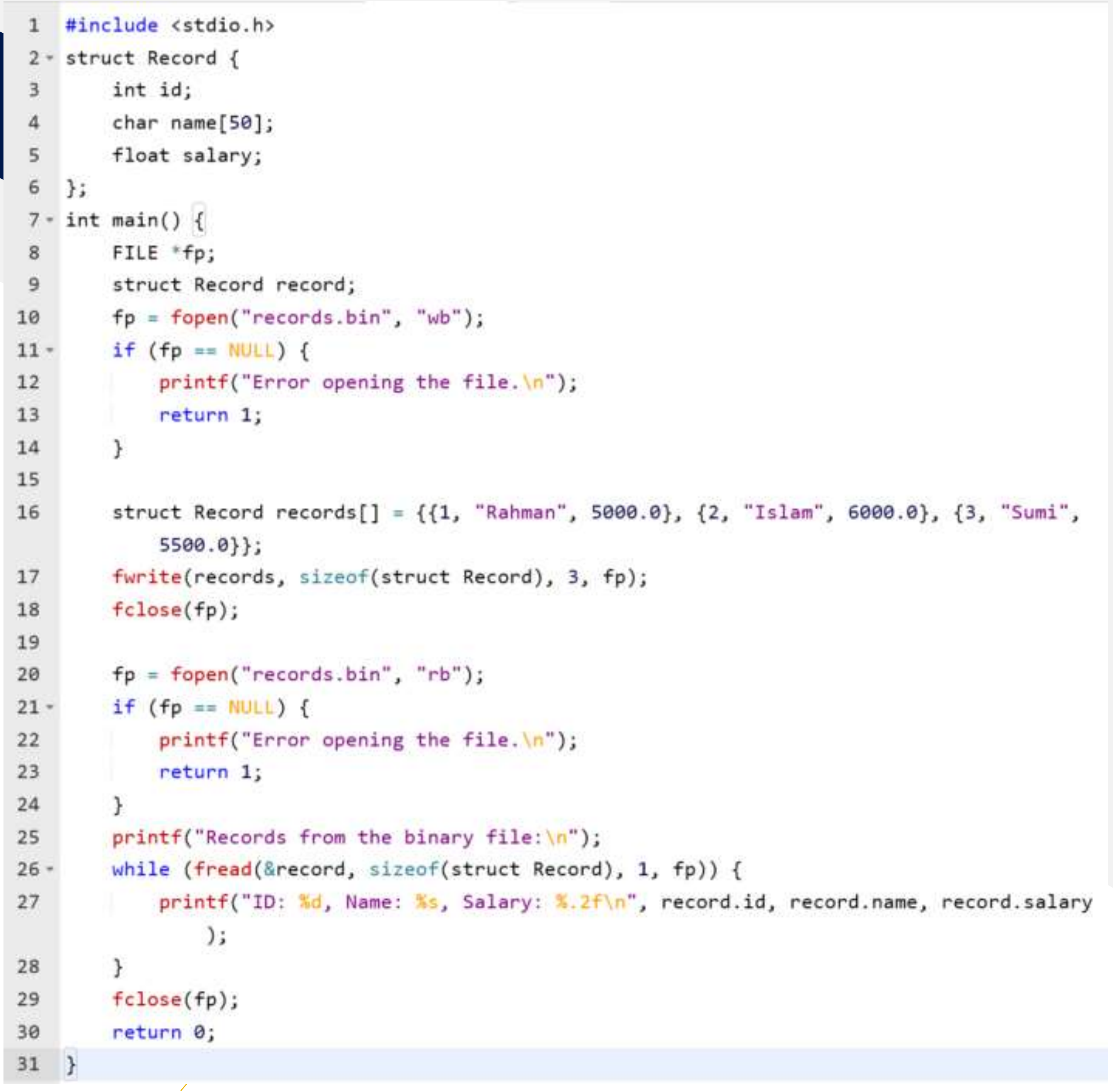
- Searching for a Specific String in a File:



```c
#include <stdio.h>
#include <string.h>

int main() {
    FILE *fp;
    char line[100];
    char search[] = "example"; // String to search for

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

    // Search for the string in each line of the file
    while (fgets(line, sizeof(line), fp) != NULL) {
        if (strstr(line, search) != NULL) {
            printf("String found in the line: %s", line);
        }
    }

    // Close the file
    fclose(fp);

    return 0;
}
```

Program Examples

- Reading and Writing Binary Files:



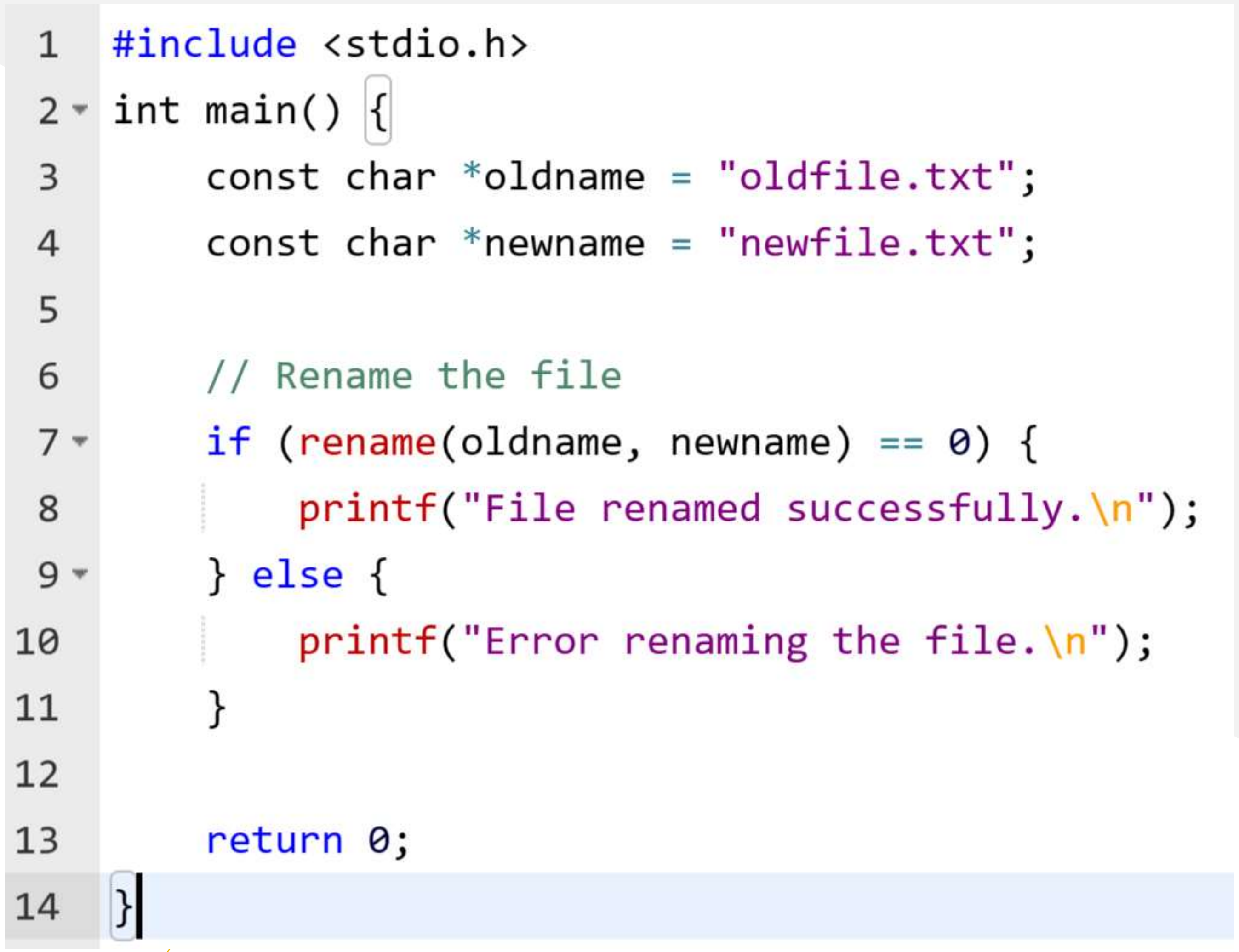
```c
#include <stdio.h>
struct Record {
    int id;
    char name[50];
    float salary;
};
int main() {
    FILE *fp;
    struct Record record;
    fp = fopen("records.bin", "wb");
    if (fp == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }

    struct Record records[] = {{1, "Rahman", 5000.0}, {2, "Islam", 6000.0}, {3, "Sumi", 5500.0}};
    fwrite(records, sizeof(struct Record), 3, fp);
    fclose(fp);

    fp = fopen("records.bin", "rb");
    if (fp == NULL) {
        printf("Error opening the file.\n");
        return 1;
    }
    printf("Records from the binary file:\n");
    while (fread(&record, sizeof(struct Record), 1, fp)) {
        printf("ID: %d, Name: %s, Salary: %.2f\n", record.id, record.name, record.salary);
    }
    fclose(fp);
    return 0;
}
```

- This program demonstrates reading and writing binary files using structures. It writes records of employees to a binary file and then reads and prints them.

Program Examples

- Renaming a File:

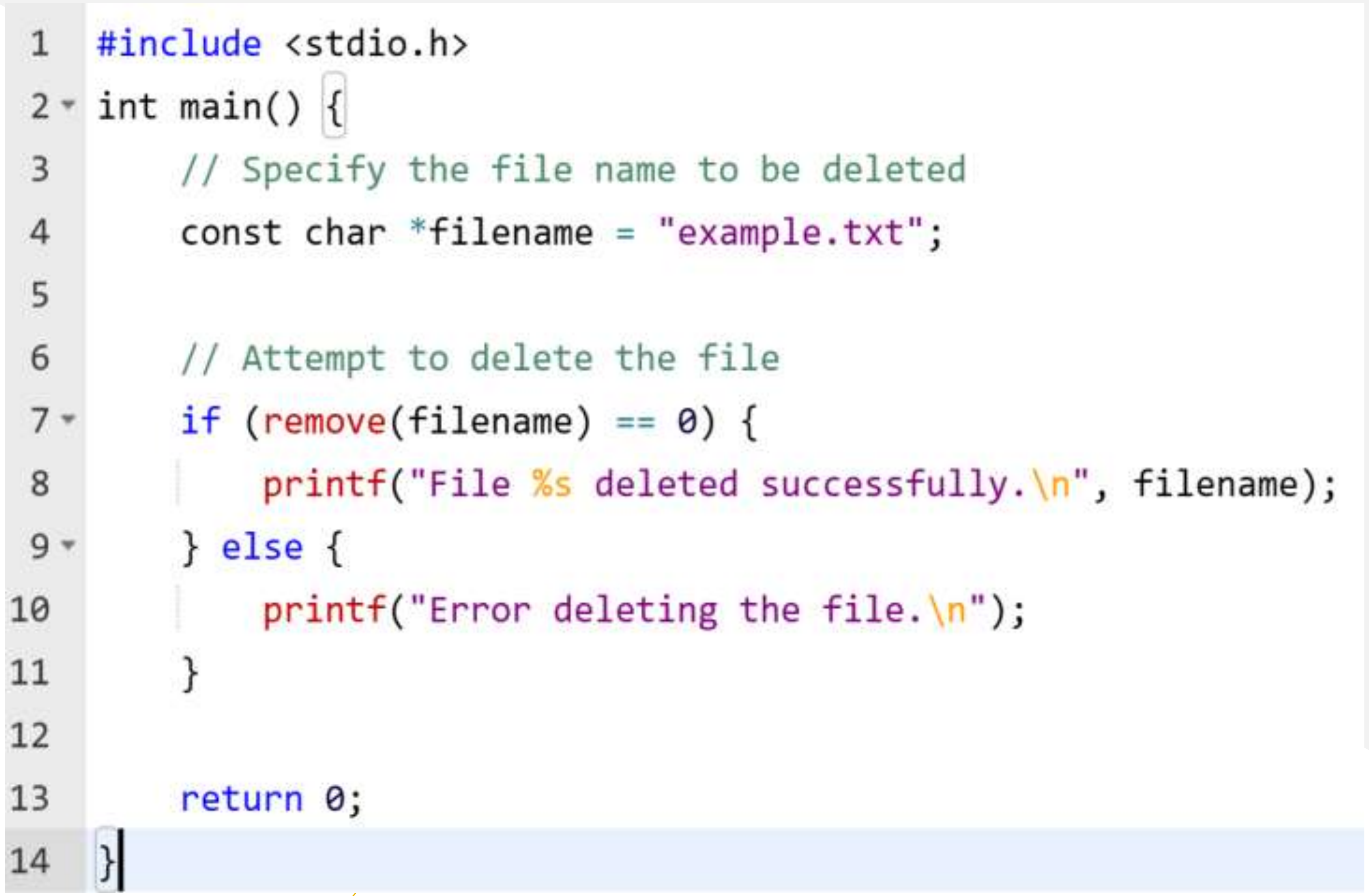
- This program demonstrates renaming a file from "oldfile.txt" to "newfile.txt".



```c
#include <stdio.h>
int main() {
    const char *oldname = "oldfile.txt";
    const char *newname = "newfile.txt";

    // Rename the file
    if (rename(oldname, newname) == 0) {
        printf("File renamed successfully.\n");
    } else {
        printf("Error renaming the file.\n");
    }

    return 0;
}
```

Program Examples

- Deleting a File:



```c
#include <stdio.h>
int main() {
    // Specify the file name to be deleted
    const char *filename = "example.txt";

    // Attempt to delete the file
    if (remove(filename) == 0) {
        printf("File %s deleted successfully.\n", filename);
    } else {
        printf("Error deleting the file.\n");
    }

    return 0;
}
```

?

THE END