rtp

week 5_2:

file handling

rajin teaches programming

INTRODUCTION:

File handling in C programming involves performing **operations** on **files** such as **creating**, **opening**, **reading**, **writing**, and **closing** files. It allows programs to store data **permanently** and **retrieve** it as needed.

– IMPORTANCE:

Data Persistence: Storing data permanently even after the program terminates.
Data Sharing: Sharing data between different programs or components.
Data Management: Organizing and managing large amounts of data efficiently.

– WHAT IS FILE HANDLING?

File **handling** refers to the process of performing **operations** on files. It includes **creating**, **opening**, **reading**, **writing**, and **closing files**, as mentioned previously. Files are used to **store** data **permanently** on a storage device.

File handling in C involves using **file pointers** and various **functions** provided by the C **standard library.** A **file pointer** is used to **keep track** of the **file** being **accessed.**

– HOW DO I USE FILES?

To use a **file** in C, you need to **declare** a **file pointer** and **open** the file using the **appropriate mode** (e.g., **read**, **write**, **append**).

SYNTAX:

DECLARATION AND OPENING

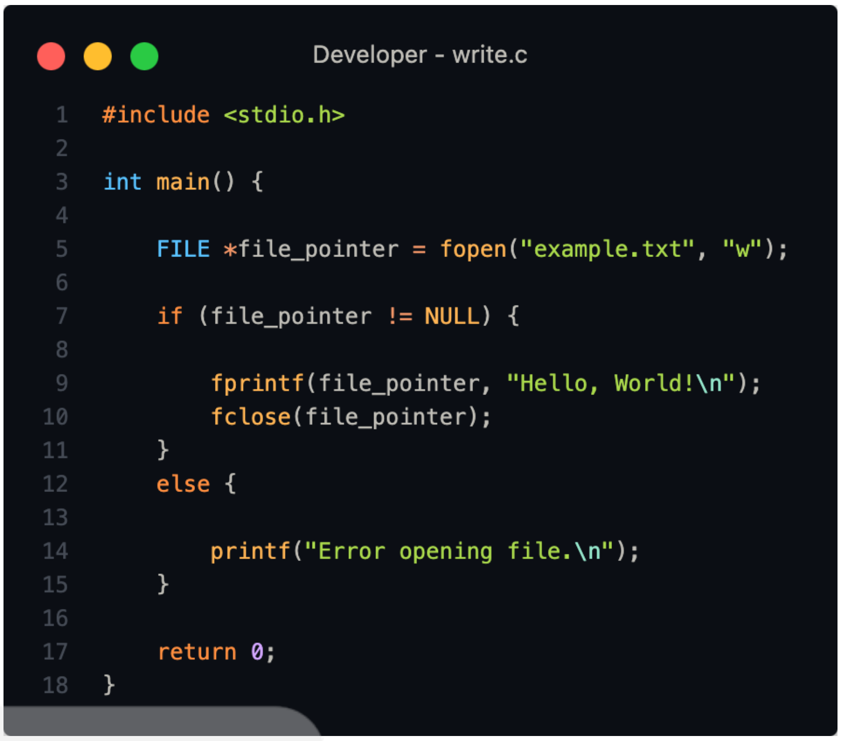
```c
FILE *file_pointer;
file_pointer = fopen("filename", "mode");
```

`filename`: Name of the file to open.
`mode`: Mode in which to open the file (`r`, `w`, `a`, `r+`, `w+`, `a+`).

mode	explanation
“r”	Read mode
“w”	Write mode
“a”	Append mode
“r+”	Read and Write mode
“w+”	Write and Read mode
“a+”	Append and Read mode

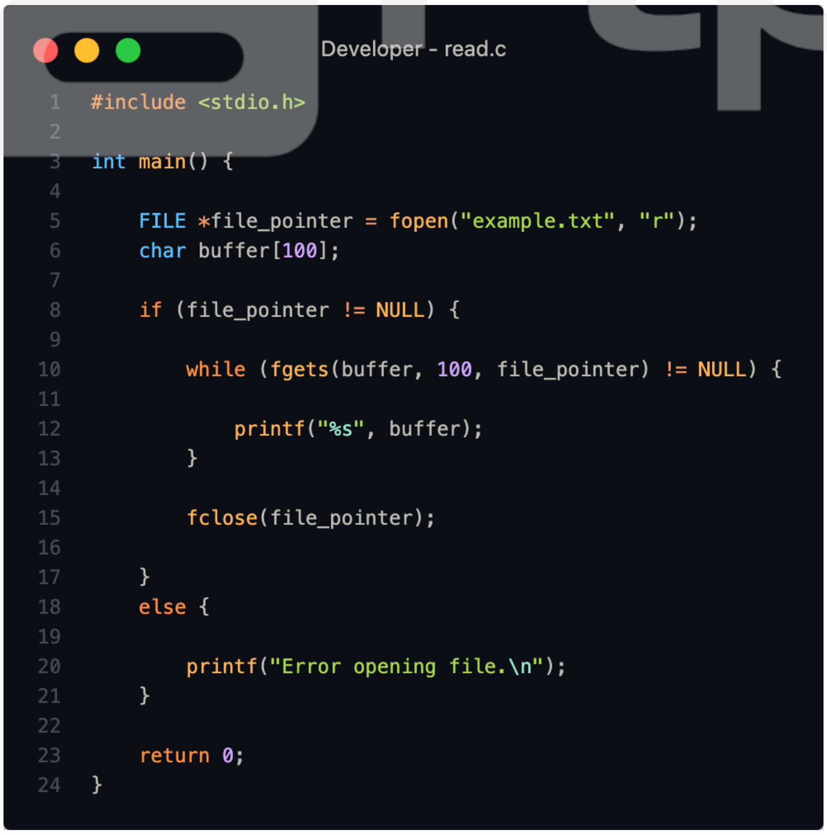
CLOSING FILES

```c
fclose(file_pointer);
```



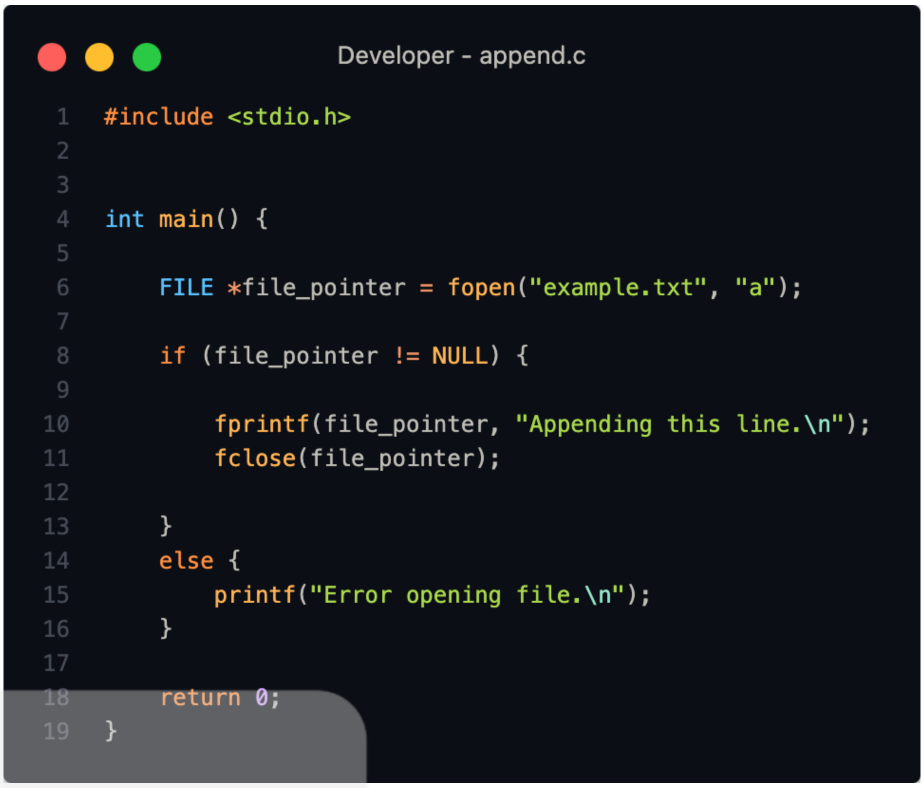
```c
#include <stdio.h>

int main() {

    FILE *file_pointer = fopen("example.txt", "w");

    if (file_pointer != NULL) {

        fprintf(file_pointer, "Hello, World!\n");
        fclose(file_pointer);
    }
    else {

        printf("Error opening file.\n");
    }

    return 0;
}
```

an example of how to write to a file in C. detailed explanation provided in class.



```c
#include <stdio.h>

int main() {

    FILE *file_pointer = fopen("example.txt", "r");
    char buffer[100];

    if (file_pointer != NULL) {

        while (fgets(buffer, 100, file_pointer) != NULL) {

            printf("%s", buffer);
        }

        fclose(file_pointer);

    }
    else {

        printf("Error opening file.\n");
    }

    return 0;
}
```

an example of how to read from a file in C. full explanation provided in class.



```c
#include <stdio.h>


int main() {

    FILE *file_pointer = fopen("example.txt", "a");

    if (file_pointer != NULL) {

        fprintf(file_pointer, "Appending this line.\n");
        fclose(file_pointer);

    }
    else {
        printf("Error opening file.\n");
    }

    return 0;
}
```

an example of how to append to a file in C.
detailed explanation provided in class.

file functions	**usage**
fopen()	Opens a file.
fclose()	Closes a file.
fread()	Reads data from a file.
fwrite()	Writes data to a file.
fprintf()	Writes formatted data to a file.
fputs()	Writes a string to a file.
fscanf()	Reads formatted data from a file.
fgets()	Reads a string from a file.

COMMON USE CASES:

Files are often used to log data from applications, such as error messages, user actions, or system events. Programs can also read configuration settings from files to determine their behavior. Furthermore, applications use files to store data such as user information, scores, or other persistent data.

BEST PRACTICES:

- ALWAYS CLOSE FILES:

Ensure that files are always closed after operations to prevent memory leaks and data corruption.

- CHECK FOR ERRORS:

Always check the return value of file operations to handle errors appropriately (this means checking if the file_pointer is NULL).

- AVOID HARDCODING FILE PATHS:

Use relative paths or configuration files to specify file paths, making your code more portable.

SUMMARY:

File handling allows for persistent data storage, retrieval, and management in C programming. Proper file handling involves opening, operating on, and closing files while checking for errors and considering performance impacts.

SOME FAQs:

How do I check if a file opened successfully?
Check if the file pointer returned by `` `fopen()` `` is `` `NULL` ``.

How can I read a file line by line?
Use `` `fgets()` `` in a loop to read each line.

How do I handle binary files?
Open the file in binary mode (**rb**, **wb**, **ab**) and use **fread()** and **fwrite()** for reading and writing.

What is the difference between fprintf() and fputs()?
fprintf(): Writes formatted data to a file.
fputs(): Writes a string to a file.

CONCLUSION:

Understanding file handling is essential for developing robust and efficient C programs that can manage data effectively. By mastering these concepts, you can leverage the power of file operations to build more sophisticated and functional applications.

rtp

next class X_X:

next topic name

rajin teaches programming

reach out for classes at: rajin.khan2001@gmail.com