Files

COP 3223C – Introduction to Programming with C

Fall 2025

Yancy Vance Paredes, PhD

Recall

Previously, we learned about **pointers**

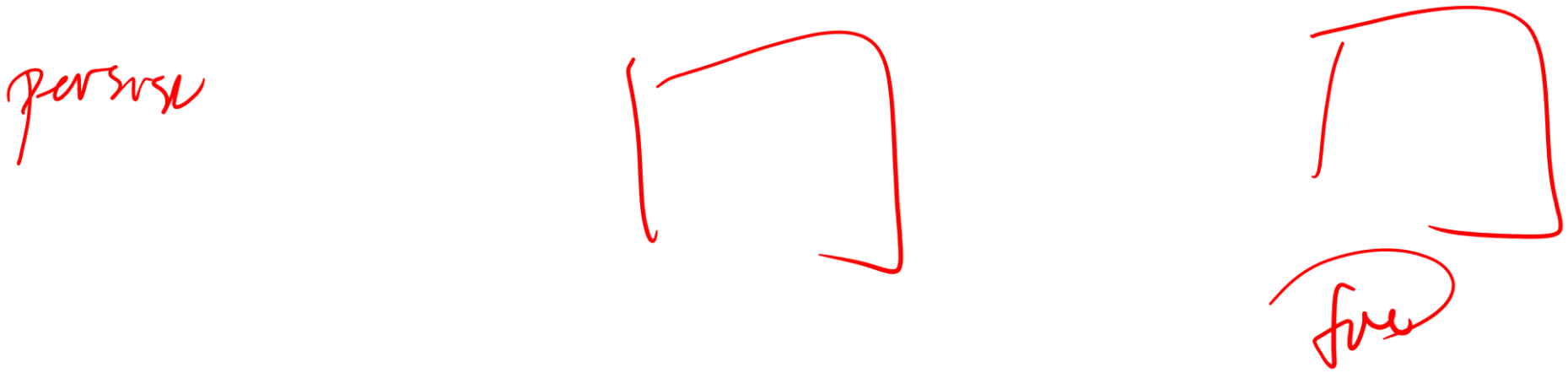
Discussion

- What if the program is **complex**?
- What if we have a large input to our program?

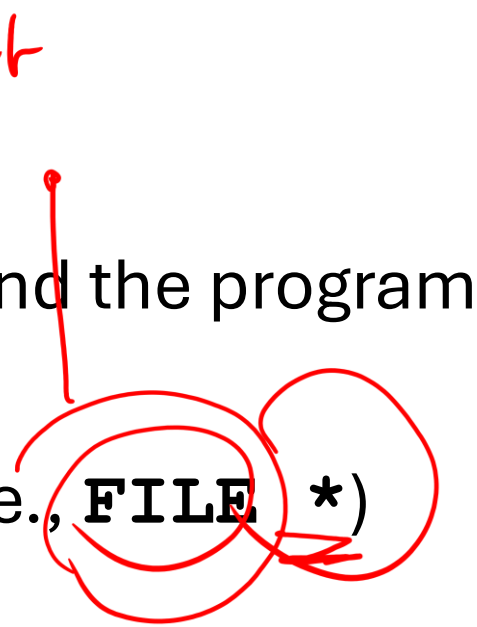
file redirection <

Files /1

- We can write programs that access files (e.g., text files)
- Two common modes are **reading** and **writing**



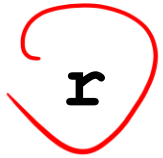
Files /2



- We want to create a connection between the file and the program
- This connection is represented by a **file pointer** (i.e., `FILE *`)
- We use this pointer to read from or write to the file

Common Type of File Access

Read Mode



Write Mode

Other Types of File Access

Append Mode

`a`

Both Reading and Writing

`r+, w+, a+`

Overview

Mode	Meaning	File must exist?	File created if missing?	File truncated (emptied)?	Cursor position
`"r"`	Read text only	Yes	No	No	Beginning
`"w"`	Write text only	No	Yes	Yes	Beginning
`"a"`	Append text only	No	Yes	No	End (writes only at end)
`"r+"`	Read and write text	Yes	No	No	Beginning
`"w+"`	Read and write text (overwrite)	No	Yes	Yes	Beginning
`"a+"`	Read and write text (append)	No	Yes	No	End (writes only at end)

File Handling in C /1

We need to keep track of the address where the file is loaded

Syntax:

```
FILE *ifile;
```

File Handling in C /2

Open the file using the appropriate mode (note: double quotes)

Syntax:

```
ifile = fopen("data.txt", "r");
```

Practice

- Open a file in reading mode
- Where should the file be located?*
- What if the file doesn't exist?

absolute path

relative path

input/data.txt

.c

.exe

data.txt

main.exe

input

└ data.txt

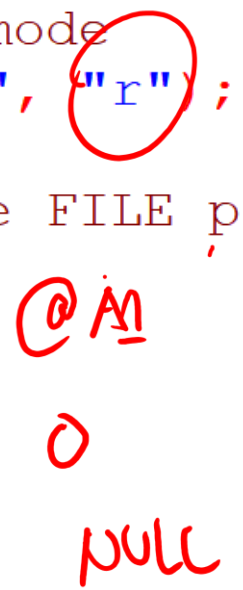
```c
#include <stdio.h>

int main(void) {
    FILE *ifile;

    // open file in reading mode
    ifile = fopen("input.txt", "r");

    // check the value of the FILE pointer
    printf("%p\n", ifile);

    return 0;
}
```



Notes

- If the file doesn’t exist and it is in **read** mode, the address is **`NULL`**
- The filename is a *string* that can be **relative** or absolute* path

** OS-dependent and often stricter in terms of where to put the file*

Best Practices

memory leak

- When we are **reading** a file, check if **`fopen()`** was successful

- Further, whenever we open a file (any mode), we must **close** it

Syntax:

```
fclose ( ifile );
```

Notes

We produced our **boilerplate** if we want to open a file for reading

```c
#include <stdio.h>

int main(void) {
    FILE *ifile;

    // open file in reading mode
    ifile = fopen("input.txt", "r");

    // check the value of the FILE pointer
    printf("%p\n", ifile);

    // check if the file was actually read
    if(ifile == NULL) {
        printf("Error: Cannot Open File!");
        return 1;
    }

    // do some file processing

    // close the file
    fclose(ifile);

    return 0;
}
```

Reading from a File

Use the **`fscanf()`** function

It is similar to **`scanf()`**

Syntax:

```
int num;
scanf("%d", &num);
fscanf(ifile, "%d", &num);
```

Practice

- Read from a file called `input.txt`
- It contains two whole numbers separated by a **newline** `'\n'`
- Print the sum of the two numbers on the screen

```c
#include <stdio.h>

int main(void) {
    FILE *ifile;

    // open file in reading mode
    ifile = fopen("input.txt", "r");

    // check the value of the FILE pointer
    printf("%p\n", ifile);

    // check if file was loaded correctly
    if(ifile == NULL) {
        printf("Error. Unable to read file.\n");
        return 1;
    }

    // if successful, then the program should
    // reach this point
    // TODO: process file
    int num1, num2;
    fscanf(ifile, "%d", &num1);
    fscanf(ifile, "%d", &num2);

    // print the sum on the screen
    printf("%d\n", num1+num2);

    // close the file
    fclose(ifile);

    return 0;
}
```

How File Reading Works?

- When a file is opened, a **cursor** (i.e., file position indicator) is placed at the **beginning of the file**

- Each time you read data (e.g., using **`fscanf`**), it moves forward

- When it reaches the end, there is nothing left to read, it is said that you have reached the **end of file (EOF)**

Code Tracing /1

Code Fragment

```
FILE *ifile;
ifile = fopen("data.txt", "r");

int count;
fscanf(ifile, "%d", &count);

int num1, num2;
fscanf(ifile, "%d", &num1);
fscanf(ifile, "%d", &num2);
```

data.txt

```
2
10
20
```

Code Tracing /2

Code Fragment

```
FILE *ifile;
ifile = fopen("data.txt", "r");

int count;
fscanf(ifile, "%d", &count);

int num1, num2;
fscanf(ifile, "%d", &num1);
fscanf(ifile, "%d", &num2);
```

data.txt

```
2
10
20
```

Code Tracing /3

Code Fragment

```
FILE *ifile;
ifile = fopen("data.txt", "r");

int count;
fscanf(ifile, "%d", &count);

int num1, num2;
fscanf(ifile, "%d", &num1);
fscanf(ifile, "%d", &num2);
```

data.txt

```
2
10
20
```

Code Tracing /4

Code Fragment

```
FILE *ifile;
ifile = fopen("data.txt", "r");

int count;
fscanf(ifile, "%d", &count);

int num1, num2;
fscanf(ifile, "%d", &num1);
fscanf(ifile, "%d", &num2);
```

data.txt

```
2
10
20
```

Code Tracing /5

Code Fragment

```
FILE *ifile;
ifile = fopen("data.txt", "r");

int count;
fscanf(ifile, "%d", &count);

int num1, num2;
fscanf(ifile, "%d", &num1);
fscanf(ifile, "%d", &num2);
```

data.txt

```
2
10
20
```

EOF

Notes

You can rewind the cursor by calling the **`rewind(fp)`** function and pass the file pointer

Practice

Write a program that reads from a file called **`test.txt`**. The first line of input indicates **`T`**, the number of test cases that will be processed by this program. Afterward, **`T`** lines follow. Each line will contain a whole number. The program should print out the square of each number (one per line).

Sample Run

test.txt

```
5
10
20
30
40
50
```

Sample Output

```
100
400
900
1600
2500
```

```c
#include <stdio.h>

int main(void) {
    FILE *ifile;

    // open file in reading mode
    ifile = fopen("test.txt", "r");

    // check the value of the FILE pointer
    printf("%p\n", ifile);

    // check if file was loaded correctly
    if(ifile == NULL) {
        printf("Error. Unable to read file.\n");
        return 1;
    }

    // if successful, then the program should
    // reach this point
    // TODO: process file
    int T;
    fscanf(ifile, "%d", &T);

    int num;
    for(int i = 0; i < T; i++) {
        fscanf(ifile, "%d", &num);
        printf("%d\n", num*num);
    }

    // close the file
    fclose(ifile);

    return 0;
}
```

Scenario

What if we don’t know how many numbers to read? For example, the file contains an arbitrary number of integers.

The `scanf()` and `fscanf()` Functions

- Both return an integer indicating how many values were read
- It returns a **0** if there was a mismatch
- Returns a **–1** if the **End Of File** (**`EOF`**) is reached

`EOF` Macro Constant

- Predefined value of **`-1`** to indicate the **end of file**

- It is defined in **`stdio.h`**

Notes

To manually signal **`EOF`** when your program is running in a terminal:

For Windows: Ctrl + Z

For Mac or Unix: Ctrl + D

Discussion

```c
#include <stdio.h>

int main(void) {
    int count;
    int a, b, c;

    count = scanf("%d%d%d", &a, &b, &c);
    printf("%d\n", count);

    return 0;
}
```

Practice /1

Write a program that reads from a file called **`numbers.txt`**. The file consists of multiple lines where each line will contain a whole number. The program should print out the square of each number (one per line).

Practice /2

numbers.txt

```
10
20
30
40
50
```

Sample Output

```
100
400
900
1600
2500
```

```c
#include <stdio.h>

int main(void) {
    FILE *ifile;

    // open file in reading mode
    ifile = fopen("numbers.txt", "r");

    // check the value of the FILE pointer
    printf("%p\n", ifile);

    // check if file was loaded correctly
    if(ifile == NULL) {
        printf("Error. Unable to read file.\n");
        return 1;
    }

    // if successful, then the program should
    // reach this point
    // TODO: process file
    int num;
    while( fscanf(ifile, "%d", &num) != EOF ) {
        printf("%d\n", num);
    }

    // close the file
    fclose(ifile);

    return 0;
}
```

Writing to a File

Use the **`fprintf()`** function

It is similar to **`printf()`**

Don’t forget to close the file!

Syntax:

Practice

- Write to a file called `output.txt`

- It should contain the sum of the two numbers from `input.txt`

```c
#include <stdio.h>

int main(void) {
    FILE *ifile;
    FILE *ofile;

    // open file in reading mode
    ifile = fopen("input.txt", "r");

    // open file in writing mode
    ofile = fopen("output.txt", "w");

    // check the value of the FILE pointer
    printf("%p\n", ifile);

    // check if file was loaded correctly
    if(ifile == NULL) {
        printf("Error. Unable to read file.\n");
        return 1;
    }

    // if successful, then the program should
    // reach this point
    // TODO: process file
    int num1, num2;
    fscanf(ifile, "%d", &num1);
    fscanf(ifile, "%d", &num2);

    printf("%d\n", num1+num2);
    fprintf(ofile, "%d\n", num1+num2);

    // close the file
    fclose(ifile);
    fclose(ofile);


    return 0;
}
```

Practice

Write a program that reads floating-point numbers from a file named `readings.txt`. The program should format each number to three decimal places, applying proper rounding. The formatted values should then be written to a new file named `readings_cleaned.txt`.

Sample Files

readings.txt

```
10.50
90.9
75
87.5
43.7897
2.9344
```

readings_cleaned.txt

```
10.500
90.900
75.000
87.500
43.790
2.934
```

%.3f

Your Turn!

Write a program that reads a list of English letters from a file named `letters.txt`. The program should convert all the letters to uppercase and write the results to a new file named `letters_upper.txt`.

Sample Files

letters.txt

```
a
b
c
d
e
f
```

letters_upper.txt

```
A
B
C
D
E
F
```

Challenge

Write a program that reads a file named `unsorted.txt`, which contains a list of integers (one integer per line). You may assume that the file contains at most 100 integers, and each integer fits within the range of a standard **`int`** data type. The program should then create a new file named `sorted.txt` that stores the same integers sorted in ascending order.

Sample Files

unsorted.txt

```
50
40
10
20
20
30
```

sorted.txt

```
10
20
20
30
40
50
```

Questions?