C LAB 4: C I/O with files

ECSE 427/COMP 310 Winter 2023

TA: Alice Chang

McGill

Contents:

Definition of Files

Basic Operations on Files

Input/Output Stream Redirection

What is file?

In modern computer systems, a common (but not universal) model for a file is that it is a sequence of 8-bit bytes sitting on some sort of **non-volatile** storage device. The sequence may be very short or very long, but it must be **finite**.

A rough definition for non-volatile storage is: storage that holds data even when the power is turned off. Common forms of non-volatile storage are magnetic disk drives and “flash" memory systems. RAM in computers is **volatile**. RAM contents are lost when power is turned off.

What is file?

- When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device.

- **File handling** in C programming uses **file streams** as a means of communication between programs and data files.

File Streams:

• A stream refers to the flow of data (in bytes) from one place to another (from program to file or vice-versa).

There are two types of streams

1)**Text Stream**

• It consists of sequence of characters

• Each line of characters in the stream may be terminated by a newline character.

• Text streams are used for textual data, which has a consistent appearance
from one environment to another or from one machine to another

2)**Binary Stream**

• It is a series of bytes.

• Binary streams are primarily used for non-textual data, which is required
to keep exact contents of the file.

File Abstraction in C programming model: FILE

How do we describe and manipulate a file in C?

→ Use a marker **FILE ***, a file pointer

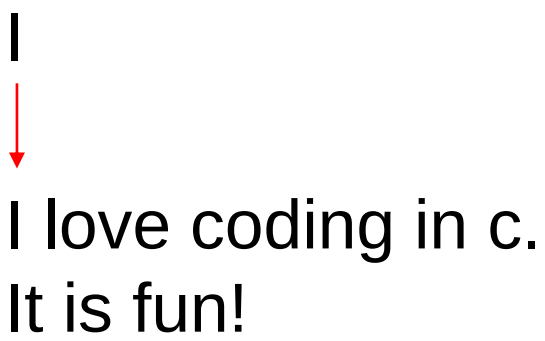
The file pointer marks the current position of the file from the beginning bytes while reading and write operation takes place on a file.

- Initially the marker is at the beginning of the file. It can be moved to any position in the file.
- The position can be specified as an offset from the beginning of the file.

File Abstraction in C programming model: FILE

→ Use a marker **FILE ***, a file pointer

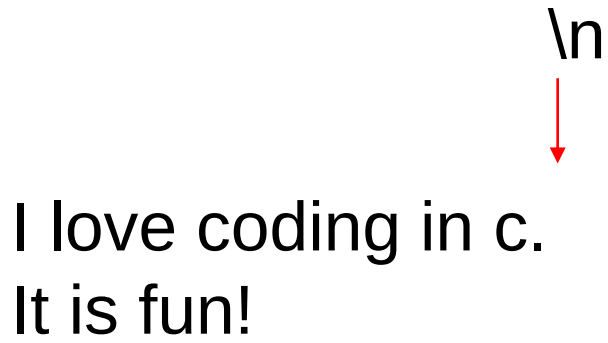
File: example.txt



File Abstraction in C programming model: FILE

→ Use a marker **FILE ***, a file pointer

File: example.txt



File Abstraction in C programming model: FILE

→ Use a marker **FILE ***, a file pointer

File: example.txt

I love coding in c.
It is fun!\0

EOF: the end of the file

Operations on FILES

1. Open a file:
   a. **fopen()**
2. Reading from file
3. Writing to a file
4. Moving to a specific location in a file
5. Closing a file

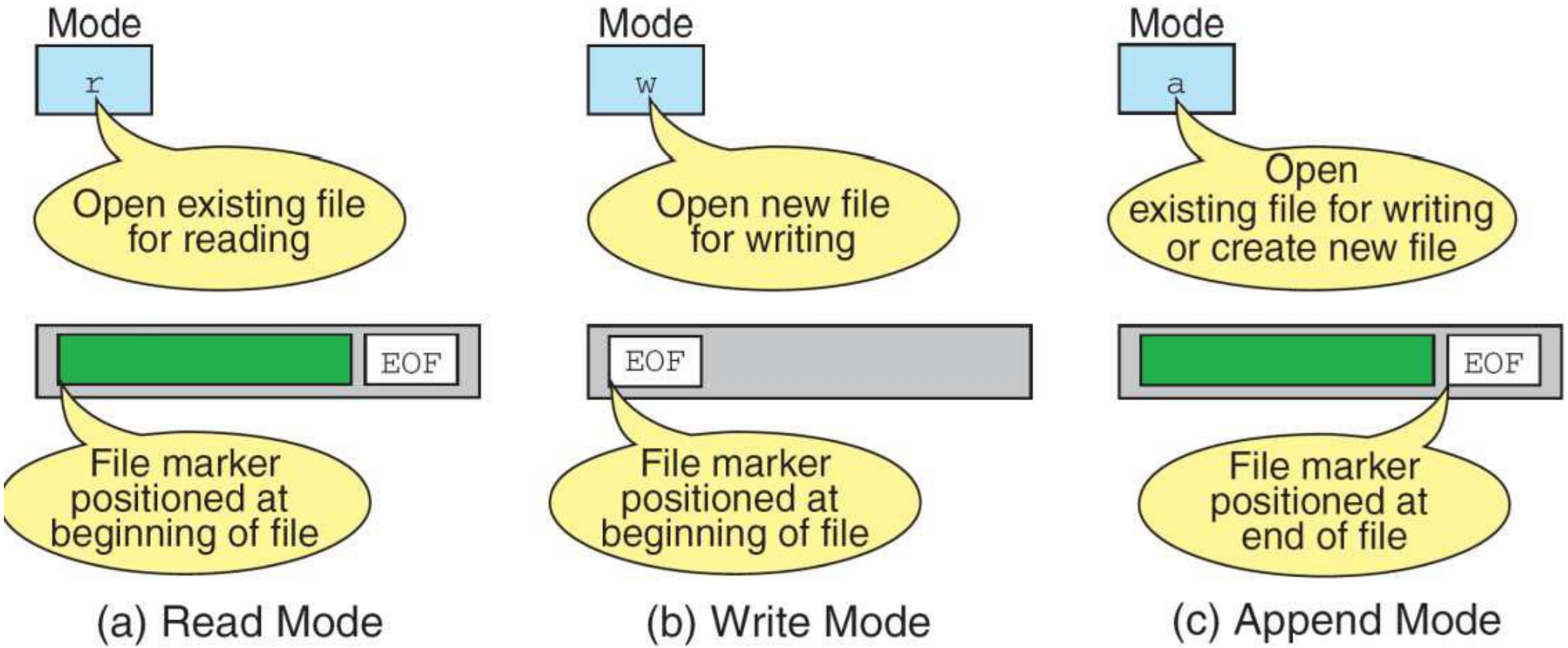
Open a file: fopen

```
FILE * fopen(const char *filename, const char *mode)
```

The C library function **fopen** opens the filename pointed to, by filename using the given mode.

The marker FILE is set to be pointing to the beginning of the file.

Fopen: Mode

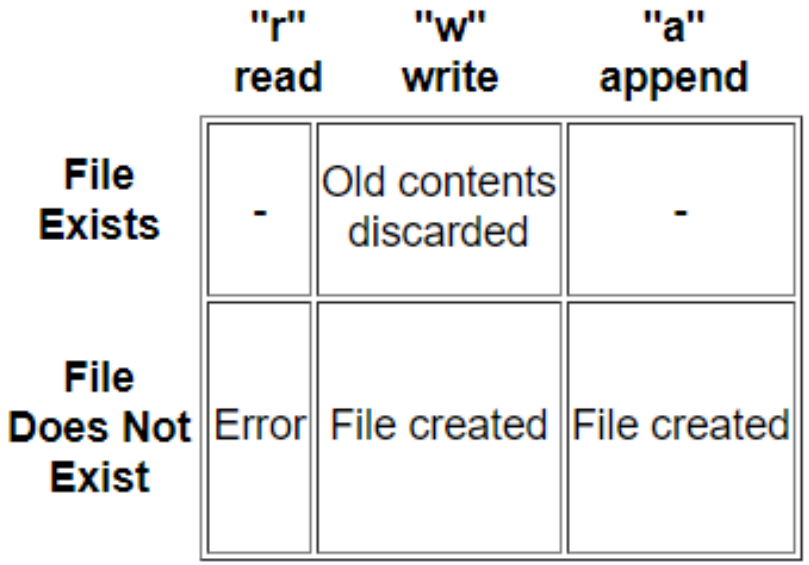


Graph By, Vrushali Solanke

Fopen: Mode

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

Table by Vrushali Solanke



Graph from: https://perugini.cps.udayton.edu/teaching/books/SPUC/www/lecture_notes/librariesIO.html

Input/output operations on FILES

1. Open a file
2. Reading from file:
   a. **fgetc()**
   b. **fgets()**
   c. **fscanf()**
3. Writing to a file
4. Moving to a specific location in a file
5. Closing a file

Reading from a file: fgetc & fgets

```
fgetc(FILE *stream);
```

Function fgetc gets the **next character** (an unsigned char) from the specified stream and advances the position indicator for the stream.

```
char *fgets(char *str, int n, FILE *stream)
```

Function fgets reads **a line** from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

Reading from a file: fgetc & fgets

```c
#include <stdio.h>
int main(){
    FILE * file;
    char str;
    if (file = fopen("hello.txt", "r")){
        while((str=fgetc(file))!=EOF)
        printf("%c",str);
    }
    fclose(file);
    return 0;
}
```

Output

i love programming in c

.

```c
#include <stdio.h>
int main(){
    FILE * file;
    char str[500];
    if (file = fopen("hello.txt", "r")){
        printf("%s", fgets(str, 50, file));
    }
    fclose(file);
    return 0;
}
```

Output

i love programming in c.

Reading from a file: fscanf

```
fscanf(FILE *stream, const char *format, ...)
```

The fscanf() function is used to read character set i.e strings from the file. It returns the **EOF**, when all the content of the file are read by it.

Reading from a file: fscanf

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
  char str1[10], str2[10], str3[10];
  int year;
  FILE * fp;
  fp = fopen ("file.txt", "w+");
  fputs("We are in 2012", fp);
  rewind(fp);
  fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
  printf("Read String1 |%s|\n", str1 );
  printf("Read String2 |%s|\n", str2 );
  printf("Read String3 |%s|\n", str3 );
  printf("Read Integer |%d|\n", year );
  fclose(fp);
  return(0);
}
```

Output

Read String1 | We |
Read String2 | are |
Read String3 | in |
Read Integer | 2012 |

Input/output operations on FILES

1. Open a file
2. Reading from file
3. Writing to a file:
   a. **fputc**
   b. **fputs**
   c. **fprintf**
4. Moving to a specific location in a file
5. Closing a file

Writing to a file: fputc

fputc(char character , FILE *stream);

The fputc() function is used to write a single character to the file.

```
#include <stdio.h>
int main(){
FILE * file;
if (file = fopen("hello.txt","w")){
if(fputs("C Lab", file) >= 0)
printf("String written to the file successfully...");
}
fclose(file);
return 0;
}
```

Output

String written to the file successfully...

Writing to a file: fputs

fputs(const char *string, FILE *stream);

The fputs() function in C can be used to write to a file. It is used to write a line (character line) to the file.

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt","w")){
       if(fputs("I/O with files", file) >= 0)
           printf("String written to the file
successfully...");
       }
   fclose(file);
   return 0;
}
```

Output
String written to the file successfully…

Writing to a file: fprintf

int fprintf(FILE *string, char *string[])

The fprintf() function is used to write data to a file. It writes a set of characters in a file.

```
#include <stdio.h>
int main(){
FILE * file;
if (file = fopen("hello.txt", "w")){
if(fprintf(file, "File Handling") >= 0)
printf("Write operation successful");
}
fclose(file);
return 0;
}
```

Output
Write operation successful

Input/output operations on FILES

1. Open a file
2. Reading from file
3. Writing to a file:
4. Moving to a specific location in a file
   a. **fseek()**
   b. **rewind()**
5. Closing a file

Moving file marker: fseek()

```
fseek ( fp, offset, position);
```

- The position fseek() repositions the file pointer.
- The prototype of fseek() function is defined in stdio.h.
- It is used to set the file position pointer for the given stream. The variable offset is an integer value that gives the number of bytes to move forward or backward in the file. The value of offset may be positive or negative, provided it makes sense.

Moving file marker: fseek()

Position can take any one of the following three values 0, 1 and 2.

Symbolic_Constant	Meaning
SEEK_SET	Beginning of File
SEEK_CUR	Current position of File
SEEK_END	End of File

fseek() options

Function	Meaning
feek(fp, 0L, SEEK_SET);	Moves to the beginning of the file
fseek(fp,0L, SEEK_CUR);	Stay at the current position of the file
fseek(fp,0L, SEEK_END);	Go to the End of File
fseek(fp,m,SEEK_CUR);	Move forward by m bytes in the file form the current location.
fseek(fp,-m, SEEK_CUR);	Moves backwards by m bytes in the file from the current location.
fseek_(fp,-m,SEEK_END);	Moves backwards by m bytes in the file from the end of the file

Moving file marker: fseek()

Return Value

This function returns zero if successful, or else it returns a non-zero value.

```c
#include <stdio.h>

int main () {
   FILE *fp;
   char c;
   fp = fopen("file.txt","w+");
   fputs("123456789", fp);

   fseek( fp, 7, SEEK_SET );
   c = fgetc(fp);
   printf("%c\n", c);
   fclose(fp);

   return(0);
}
```

Output

8

https://www.tutorialspoint.com/c_standard_library/c_function_fseek.htm

Moving file marker: rewind()

rewind(fp);

The rewind positions the file pointer to the beginning of a file.

```c
#include <stdio.h>
int main () {
   FILE *fp;
   char c;
   fp = fopen("file.txt","w+");
   fputs("123456789", fp);
   fseek( fp, 7, SEEK_SET );
   c = fgetc(fp);
   printf("%c\n", c);
   rewind(fp);
   c = fgetc(fp);
   printf("%c\n", c);
   fclose(fp);
   return(0);
}
```

Output:
8
1

Input/output operations on FILES

1. Open a file
2. Reading from file
3. Writing to a file:
4. Moving to a specific location in a file
5. Closing a file
   a. **fclose()**

Close a file

- When we finish with a mode, we need to close the file before ending the program or beginning another mode with that same file.
- To close a file, we use fclose and the pointer variable:

```
fclose(FILE *file);
```

File Operation: Serialized Write/Read: fread()

fread(<buffer>, <size>, <qty>, <file pointer>);

fread reads <qty> units of size <size> from the file pointed to and stores them in memory in a buffer(usually a array) pointed to by <buffer>..

```
Int arr[10];
fread(arr, sizeof(int), 10, ptr);
```

Another way to write the function of fgetc

```
char c;
fread(&c, sizeof(char), 1, ptr);
```

https://www.tutorialspoint.com/explain-the-functions-fread-and-fwrite-used-in-files-in-c

File Operation: Serialized Write/Read: fwrite()

fwrite(<buffer> , <size> , <qty> , <file pointer>);

- fwrite writes <qty> units of size <size> to the file pointed to by reading them from a buffer(usually an array) pointed to by <buffer>.
- The operation of the file pointer passed in as a parameter must be “w” for write or “a” for append. Or you will get an error.

The fwrite() function writes an entire record at a time.

```
Int arr[10];

fwrite(arr, sizeof(int), 10, ptr);
```

File Operation: Serialized Write/Read:

Problem

Write a C program for storing the details of 2 students into a file and print the same using fread() and fwrite()

Solution

The fread() function reads the entire record at a time.

https://www.tutorialspoint.com/explain-the-functions-fread-and-fwrite-used-in-files-in-c

File Redirect: stdin, stdout, and stderr

standard data streams created when you launch a Linux command.

You can use them to tell if your scripts are being piped or redirected.

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: stdin

stdin is the standard input stream.
This accepts text as its input.

```
./a.out < input.txt
```

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: stdout

In Linux, stdout is the standard output stream.
Text output from the command to the shell is delivered via the stdout (standard out) stream.

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: stderr

Error messages from the command are sent through the stderr (standard error) stream.

Example

redirect the output to a file:

SomeCommand > SomeFile.txt

./binary > file

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: Redirecting stdout and stderr

The error message that is delivered via stderr is still sent to the terminal window. We can check the contents of the file to see whether the stdout output went to the file.

```
cat capture.txt
```

The output from stdin was redirected to the file as expected.

cat fgetc.c > output.txt

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

Streams Are Handled Like Files

Each file associated with a process is allocated a unique number to identify it. This is known as the file descriptor. Whenever an action is required to be performed on a file, the file descriptor is used to identify the file.

Example

0: stdin
1: stdout
2: stderr

To explicitly redirect stdout, use this redirection instruction:

```
1>
```

To explicitly redirect stderr, use this redirection instruction:

```
2>
```

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: Redirecting stdout and stderr

Let’s try to our test again, and this time we’ll use 2>:

```
./error.sh 2> capture.txt
```

Let’s see what is in the capture.txt file.

```
cat capture.txt
```

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: Redirecting Both stdout and stderr

This command will direct stdout to a file called capture.txt and stderr to a file called error.txt.

```
./error.sh 1> capture.txt 2> error.txt
```

Let's check the contents of each file:

```
cat capture.txt
cat error.txt
```

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

File Redirect: Redirecting stdout and stderr to the Same File

We can achieve this with the following command:

```
./error.sh > capture.txt 2>&1
```

- *./error.sh*: Launches the error.sh script file.
- > *capture.txt*: Redirects the stdout stream to the capture.txt file. > is shorthand for 1>.
- *2>&1*: This uses the &> redirect instruction. This instruction allows you to tell the shell to make one stream got to the same destination as another stream. In this case, we're saying "redirect stream 2, stderr, to the same destination that stream 1, stdout, is being redirected to."

What Are stdin, stdout, and stderr on Linux? (howtogeek.com)

Thank you!