Software Engineering Project 1
COMP10050

Lecture 2

Objectives

- Introduce you to Assignment 1
- Use 2-D Arrays to store sentences
- Open, Read and Write Files
- Create Modules in C

Let’s Have a Look at Assignment 1

Assignment 1

1. Read/write strings from/to files

2. Sort strings

3. Identify anagrams

4. Identify missing anagrams

Assignment 1

1. Read/write strings from/to files
   → (Week 2)
2. Sort strings
   → (Week 3)
3. Identify anagrams
   → (Week 4-5)
4. Identify missing anagrams
   → (Week 6)

Use 2D Arrays To Store Strings

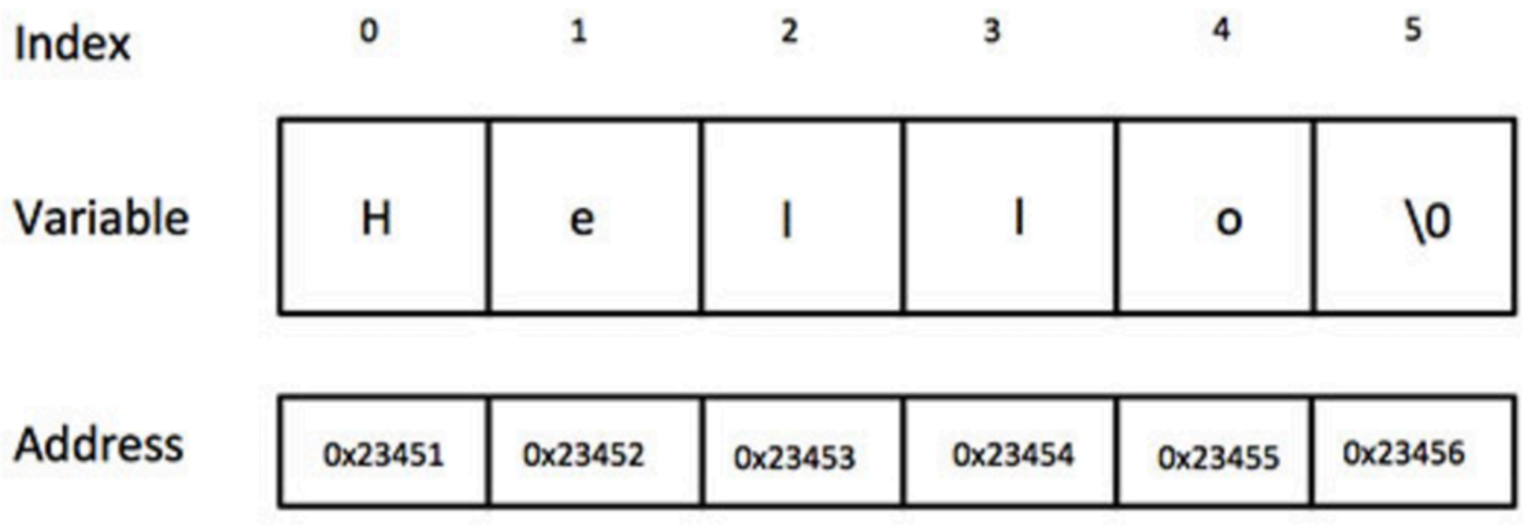
Strings – Representation

- Strings are represented as an array of characters
- C does not restrict the length of the string. The end of the string is specified using \0.

Strings – Representation

- Strings are represented as an array of characters
- C does not restrict the length of the string. The end of the string is specified using \0.

For instance “Hello” is represented as follows



Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

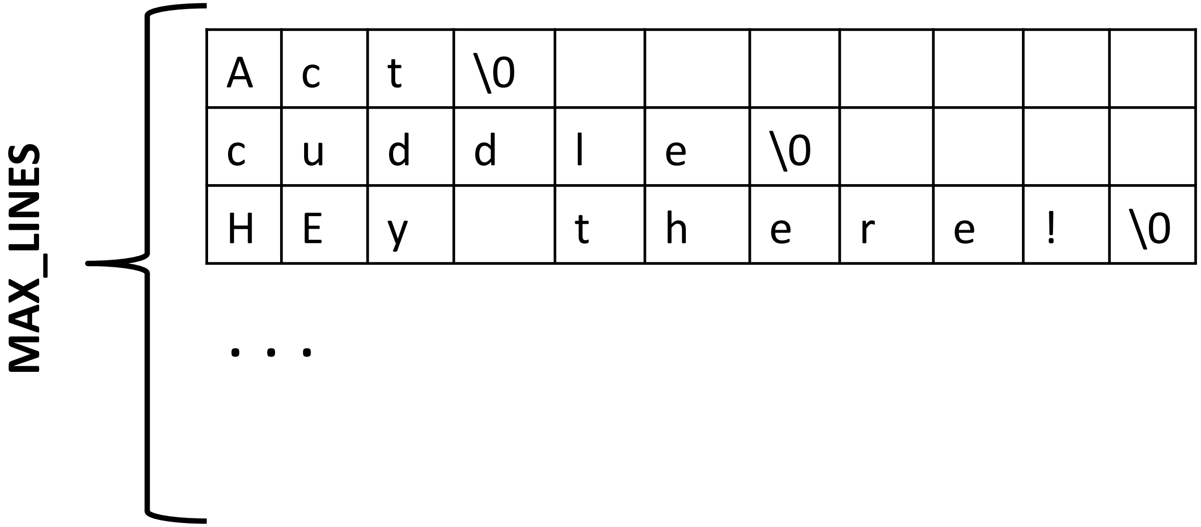
Strings – Declaration Examples

```
char str[] = “hello”;
/*compiler takes care of size */
```

But How can we store more than one string?

Store a List of Strings

```
#define MAX_LINES 20
#define MAX_CHAR 60

char sentences[MAX_LINES][MAX_CHAR];
```

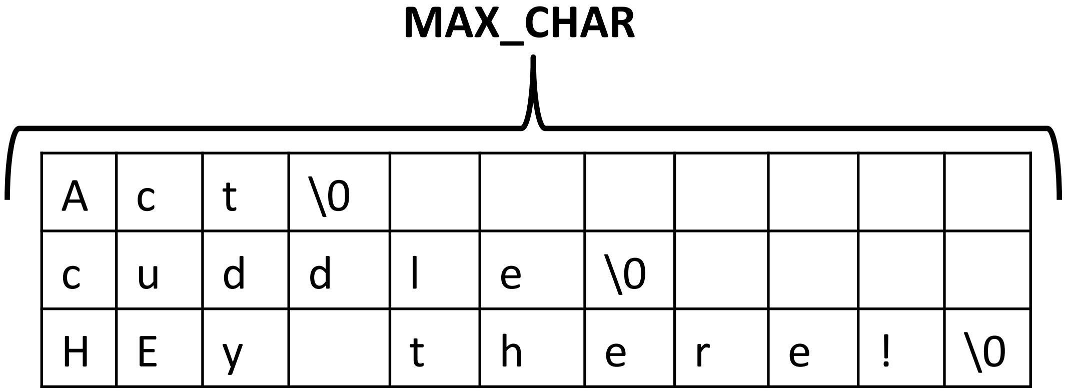


A	c	t	\0							
c	u	d	d	l	e	\0				
H	E	y		t	h	e	r	e	!	\0

. . .

Store a List of Strings

```
#define MAX_LINES 20
#define MAX_CHAR 60

char sentences[MAX_LINES][MAX_CHAR];
```



MAX_CHAR

A	c	t	\0							
c	u	d	d	l	e	\0				
H	E	y		t	h	e	r	e	!	\0

. . .

Print Sentences

```
#define MAX_LINES 20
#define MAX_CHAR 60

char sentences[MAX_LINES][MAX_CHAR];
```

A	c	t	\0							
c	u	d	d	l	e	\0				
H	E	y		t	h	e	r	e	!	\0

. . .

```
printf("Character [1][2]: %c", sentences[1][2]);
```

```
Character [1][2]: d
```

Print Sentences

```
#define MAX_LINES 20
#define MAX_CHAR 60
char sentences[MAX_LINES][MAX_CHAR];
```

A	c	t	\0							
c	u	d	d	l	e	\0				
H	E	y		t	h	e	r	e	!	\0

. . .

```
printf(“%s”, sentences[1]);
```

```
cuddle
```

File Input and Output

Opening a File

FILE *fopen(const char *path, const char *mode)

```
fopen(FILE_PATH, “r+”)
```

- **Mode** can be “**r**” (read only), “**w**” (write only), “**a**” (append)
- **fopen** returns a pointer to the file stream if it exists or NULL otherwise
- No need to know the details of the FILE data type
- **Important:** The standard input and output are also FILE* datatypes (stdin,stdout).
- **Important:** stderr corresponds to standard error output (different from stdout)

Closing a File

int fclose(FILE* fp)

```
fclose(fp)
```

- Closes the stream (releases OS resources).
- Returns 0 if the stream is closed successfully. On failure, EOF is returned.
- fclose() is automatically called on all open files when program terminates.

File Input

int getc(FILE* fp)

- Reads a single character from the stream
- Returns the character read or EOF on error/end of file.

Note: getchar simply uses the standard input to read a character. We can implement it as follows:

#define getchar() getc(stdin)

File Input

char* fgets(char *line, int maxlen, FILE *fp)

- **line –** pointer to an array of chars where the string read is stored
- Reads a single line (up to a number of characters given by **maxlen**) from the file input stream **fp** (including linebreak).
- Returns a pointer to the character array that stores the line (read-only)
- Return NULL if end of stream

File Output

int putc(int c, FILE* fp)

- Writes a single character **c** to the output stream
- Returns the character written or EOF on error

Note: putchar simply uses the standard output to write a character.

We can implement it as follows:

#define putchar() putc(c, stdout)

File Output

int fputs(char *line, FILE *fp)

- Writes a single line to the output stream
- Returns zero on success, EOF otherwise

Exercise 1

Create a new file and write string “Hello World” in it.

Exercise 1

Create a new file and write string “Hello World” in it.

```
FILE *fp;
char myString[80];
```

Exercise 1

Create a new file and write string “Hello World” in it.

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”,”rw”);
```

Exercise 1

Create a new file and write string “Hello World” in it.

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”,”rw”);
```

File needs to be read and written

Exercise 1

Create a new file and write string “Hello World” in it.

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”,”rw”);
fputs(“Hello World”, fp);
```

Writes “Hello World in the file”

Exercise 1

Create a new file and write string “Hello World” in it.

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”,”rw”);
fputs(“Hello World”, fp);

fgets(myString, 80, fp);
```

Reads the first 80 characters from the first line of the file.

Exercise 1

Create a new file and write string “Hello World” in it.

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”,”rw”);
fputs(“Hello World”, fp);

fgets(myString, 80, fp);

printf(“The context f myfile is: %s”, myString);
```

Exercise 1

Is there anything missing?

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”,”rw”);
fputs(“Hello World”, fp);

fgets(myString, 80, fp);

printf(“The context f myfile is: %s”, myString);

          fclose(fp);
```

Exercise 1

Will the program still read string “Hello World” correctly?

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”, “w+”);
fputs(“Hello World”, fp);

fgets(myString, 80, fp);

printf(“The context f myfile is: %s”, myString);
```

Exercise 1

Will the program still read string “Hello World” correctly?

```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”, “w+”);
fputs(“Hello World”, fp);

fseek(fp, 0, SEEK_SET);

fgets(myString, 80, fp);

printf(“The context f myfile is: %s”, myString);
```

Moves the file pointer position to the beginning of the file

Exercise 1

Will the program still read string “Hello World” correctly?

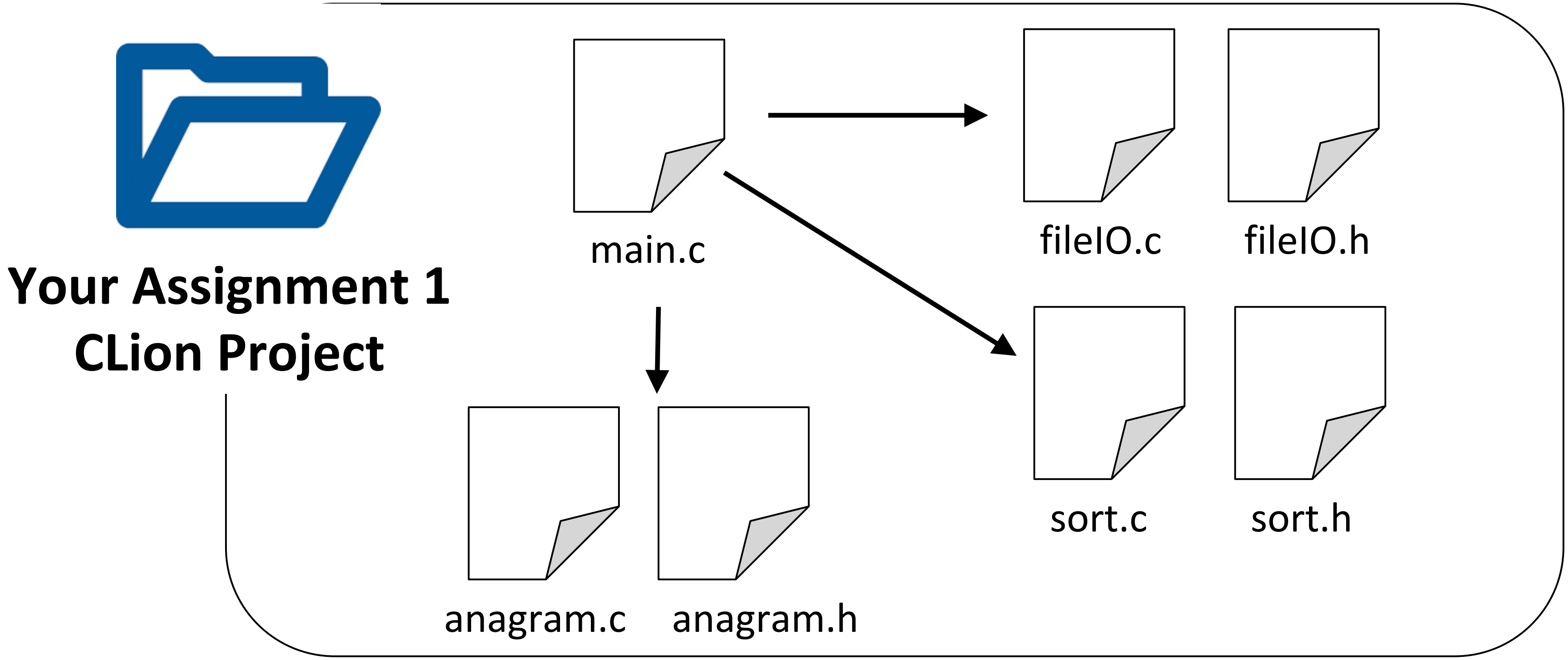
```
FILE *fp;
char myString[80];

fp = fopen(“/Users/liliana1/myfile.txt”, “w+”);
fputs(“Hello World”, fp);

fseek(fp, 0, SEEK_SET);

fgets(myString, 80, fp);

printf(“The context f myfile is: %s”, myString);
```

SEEK_CUR : move the pointer to the current position
SEEK_END : move the pointer to the end of the file

Create Modules in C

Modularity

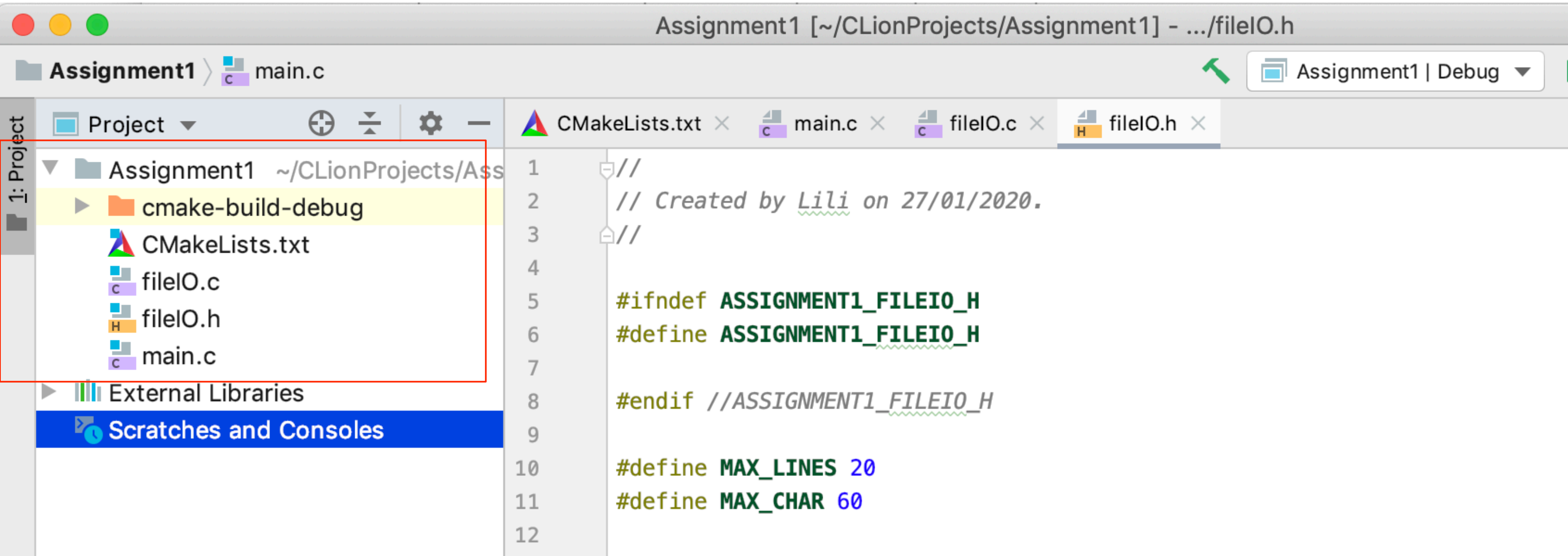
Your assignment should be modular and should not be implemented in 1 single file and using a single main function!



Creating and Using Libraries

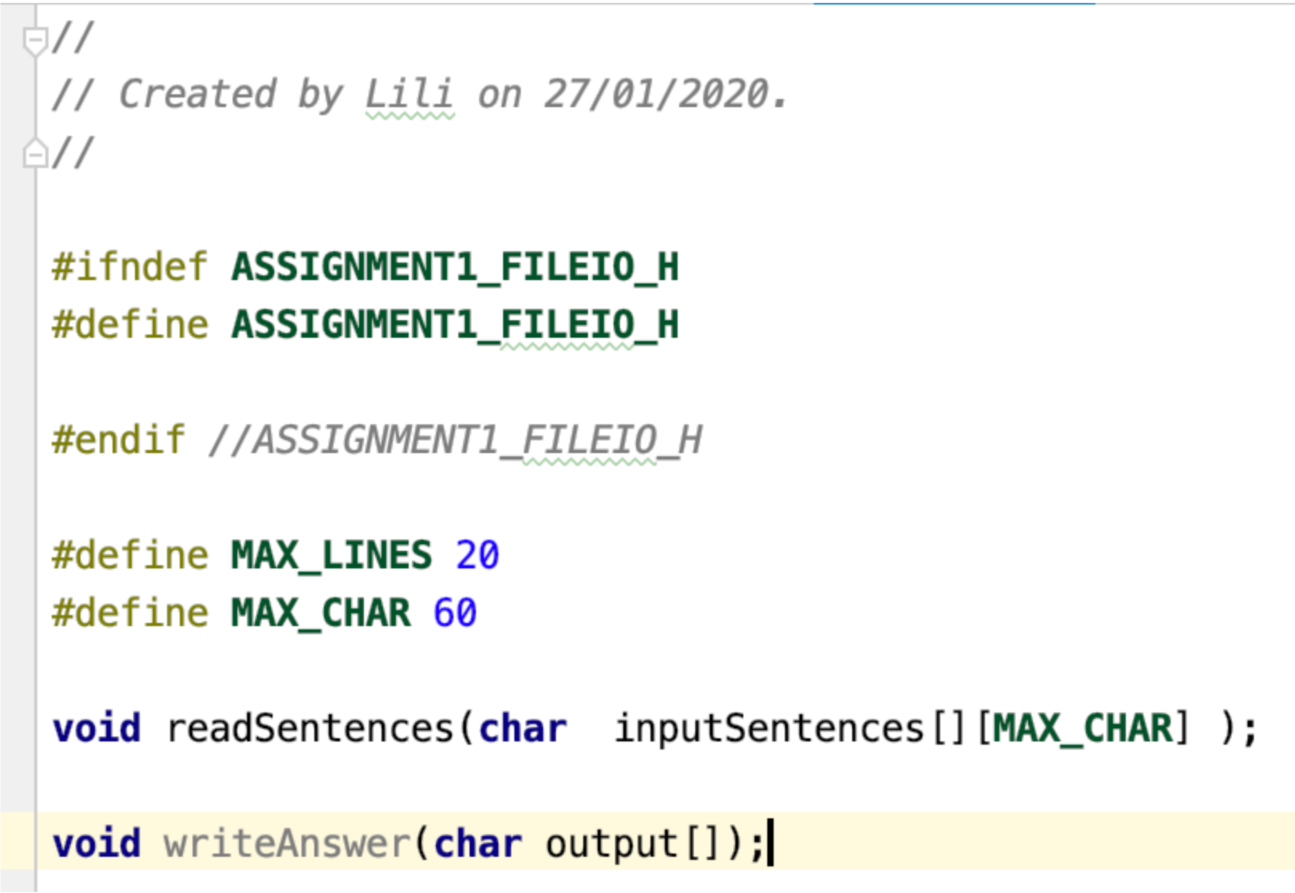
fileIO.h ➔ It is a library containing the prototypes of the methods necessary to read and write to/from files

fileIO.c ➔ It is a source file containing the implementation of the methods listed in fileIO.h



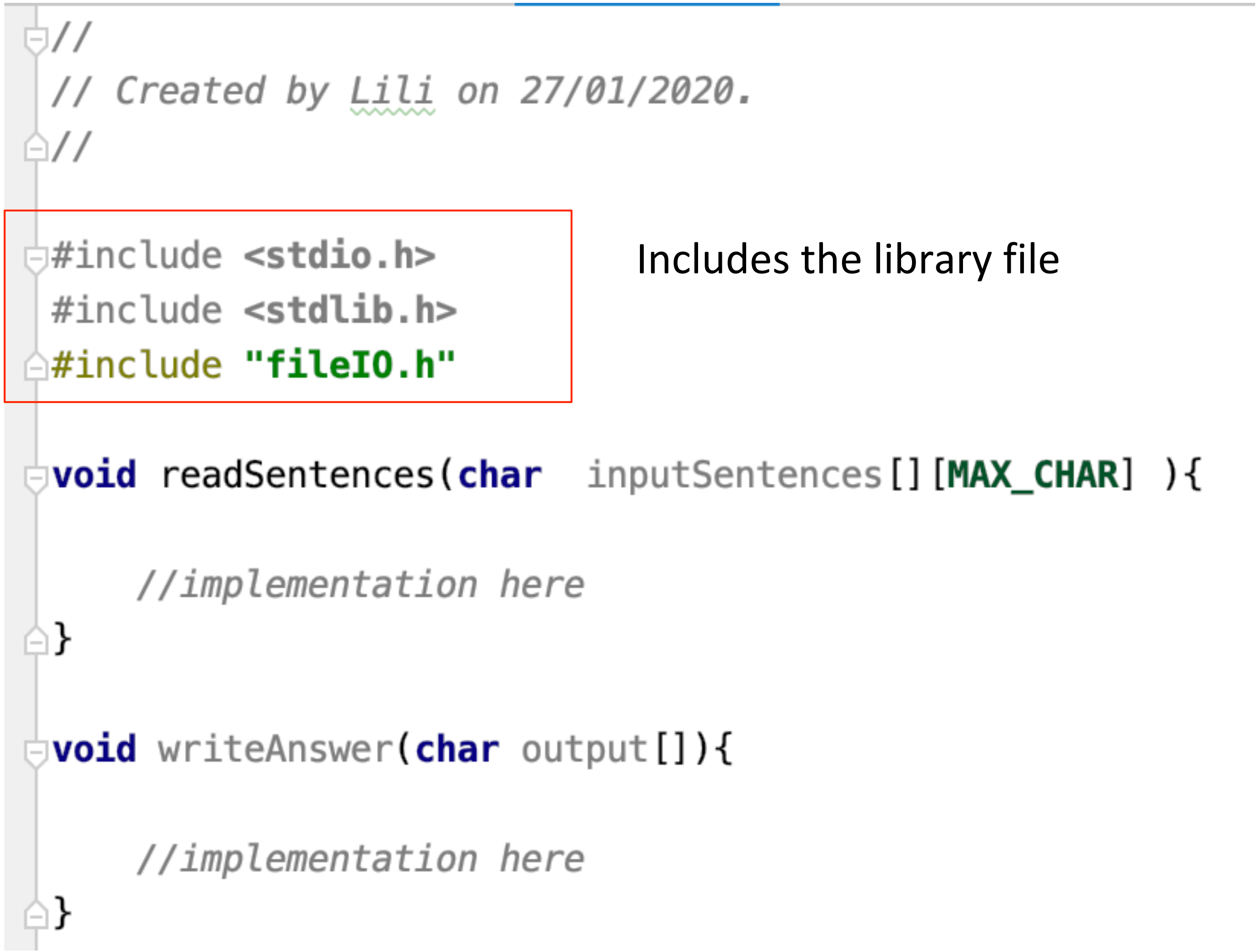
```
//
// Created by Lili on 27/01/2020.
//

#ifndef ASSIGNMENT1_FILEIO_H
#define ASSIGNMENT1_FILEIO_H

#endif //ASSIGNMENT1_FILEIO_H

#define MAX_LINES 20
#define MAX_CHAR 60

```

fileIO.h



```
//
// Created by Lili on 27/01/2020.
//

#ifndef ASSIGNMENT1_FILEIO_H
#define ASSIGNMENT1_FILEIO_H

#endif //ASSIGNMENT1_FILEIO_H

#define MAX_LINES 20
#define MAX_CHAR 60

void readSentences(char  inputSentences[][MAX_CHAR] );

void writeAnswer(char output[]);
```

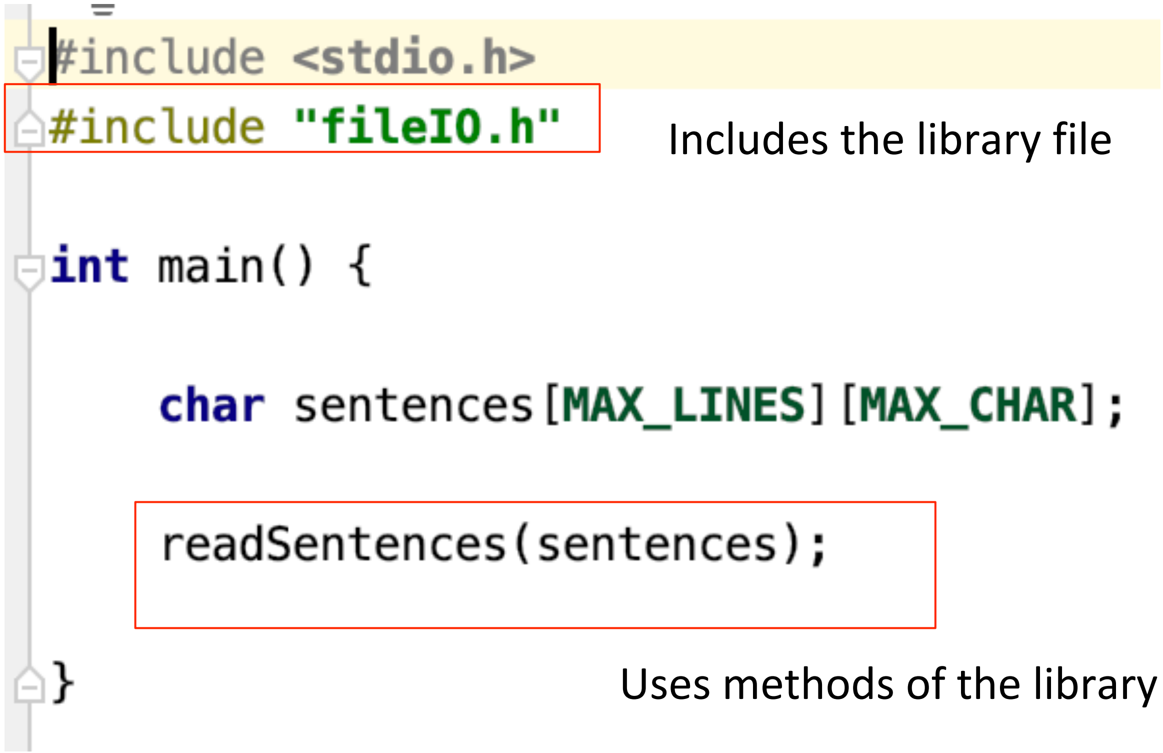
fileIO.c



```c
//
// Created by Lili on 27/01/2020.
//

#include <stdio.h>
#include <stdlib.h>
#include "fileIO.h"

void readSentences(char  inputSentences[][MAX_CHAR] ){

    //implementation here
}

void writeAnswer(char output[]){

    //implementation here
}
```

Includes the library file

main.c



```
#include <stdio.h>
#include "fileIO.h"

int main() {

    char sentences[MAX_LINES][MAX_CHAR];

    readSentences(sentences);

}
```

Includes the library file

Uses methods of the library

Library methods can also be used from other source files.

The important is to declare the library at the beginning!