CS 332/532 Systems Programming

Lecture 14

-Standard I/O Libraries-

Professor : Mahmut Unan – UAB CS

Agenda

- getc
- getline
- getdelim
- fgets
- fscanf
- fprintf
- …..

Standard I/O Library

- This library is specified by ISO C standard because it has been implemented on many OS
- The standard I/O library handles such details as buffer allocation and performing I/O in optimal-sized chunks, obviating our need to worry about using the correct block size

Recall - File Descriptor

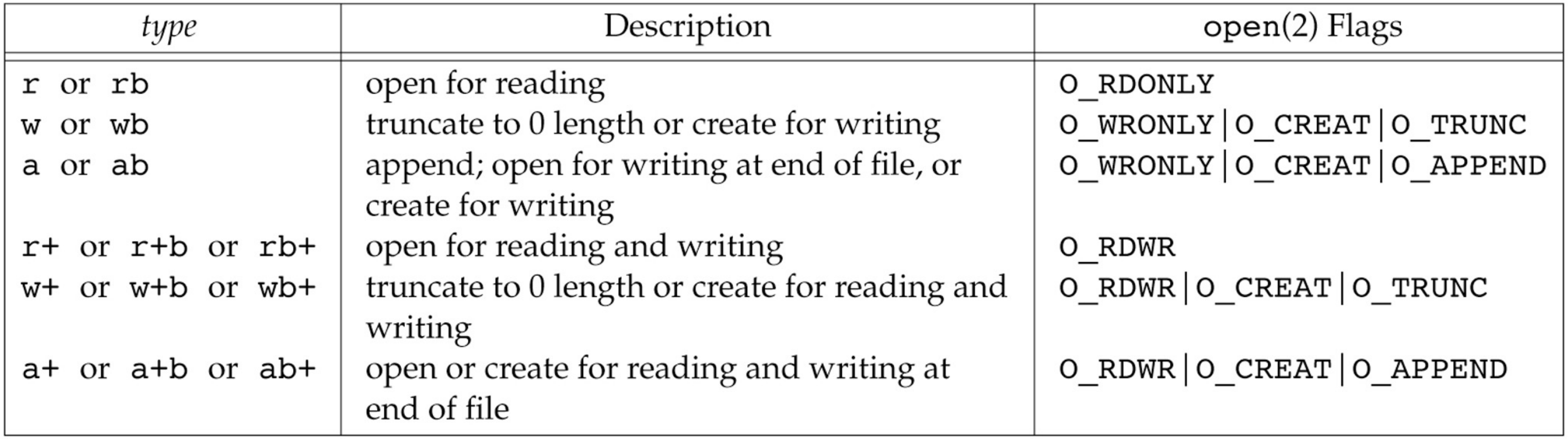
- When a file opened
  - nonnegative int assigned
  - this int is used in all operations

- Streams
  - with the standard I/O library, the discussion centers on streams
  - Open or Create a file → associate with a stream

I/O Stream

- *Open I/O Stream:* The standard I/O stream allows you to open a file in read, write, or append modes. This mode can be combined in a single open function call (see Figure 5.2 in Section 5.5 of the textbook for a complete list of options that can be specified). For example:

```
FILE *fptr;
fptr = fopen("listings.csv", "r+");
```



type	Description	open(2) Flags
`r` or `rb`	open for reading	`O_RDONLY`
`w` or `wb`	truncate to 0 length or create for writing	`O_WRONLY|O_CREAT|O_TRUNC`
`a` or `ab`	append; open for writing at end of file, or create for writing	`O_WRONLY|O_CREAT|O_APPEND`
`r+` or `r+b` or `rb+`	open for reading and writing	`O_RDWR`
`w+` or `w+b` or `wb+`	truncate to 0 length or create for reading and writing	`O_RDWR|O_CREAT|O_TRUNC`
`a+` or `a+b` or `ab+`	open or create for reading and writing at end of file	`O_RDWR|O_CREAT|O_APPEND`

Figure 5.2 The *type* argument for opening a standard I/O stream

Input / Output Stream:

- ***Input Stream*****:** The standard I/O stream allows us to read from the open file. These functions allow us to read a file character by character – getchar(), line by line – fgets(), or with specific size – fread()
- ***Output Stream*****:** The standard I/O stream allow you to write to an open file. These functions allow you to write to a file character by character – putchar(), line by line – fputs(), or with specific size – fwrite()

Exercise 1

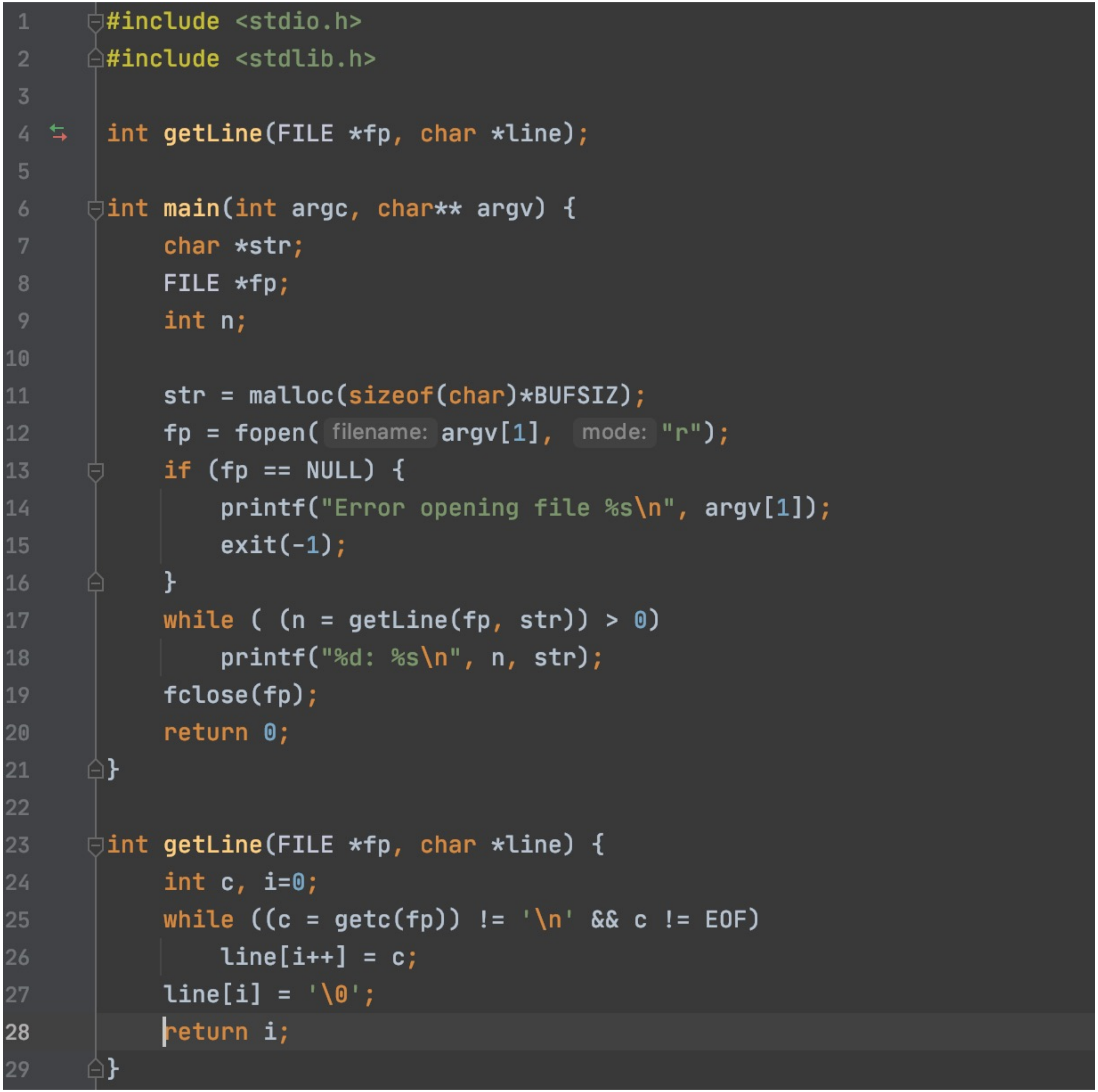
- Now let’s use these functions and write a program. We will use APIs available in Linux and C to develop different versions of this program

getc()

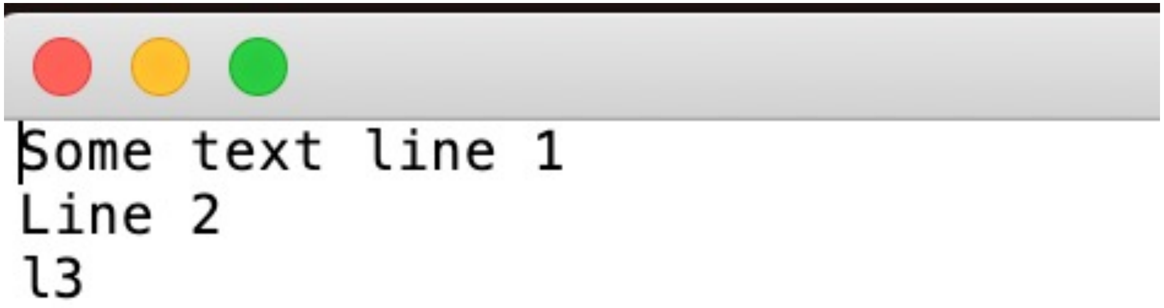
```
int getc(FILE *stream)
```

- Gets the next character (an unsigned char) from the specified stream
- Advances the position indicator for the stream.

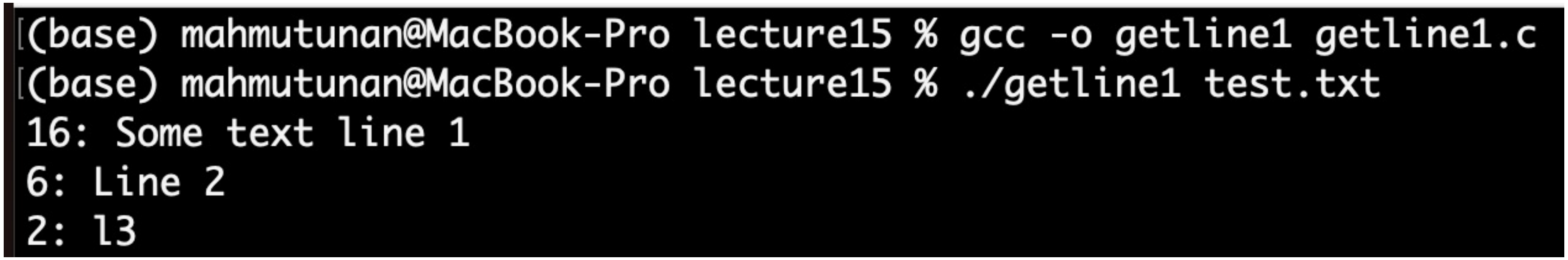
getline1.c



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

int getLine(FILE *fp, char *line);

int main(int argc, char** argv) {
    char *str;
    FILE *fp;
    int n;

    str = malloc(sizeof(char)*BUFSIZ);
    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        printf("Error opening file %s\n", argv[1]);
        exit(-1);
    }
    while ( (n = getLine(fp, str)) > 0)
        printf("%d: %s\n", n, str);
    fclose(fp);
    return 0;
}

int getLine(FILE *fp, char *line) {
    int c, i=0;
    while ((c = getc(fp)) != '\n' && c != EOF)
        line[i++] = c;
    line[i] = '\0';
    return i;
}
```



```
Some text line 1
Line 2
l3
```



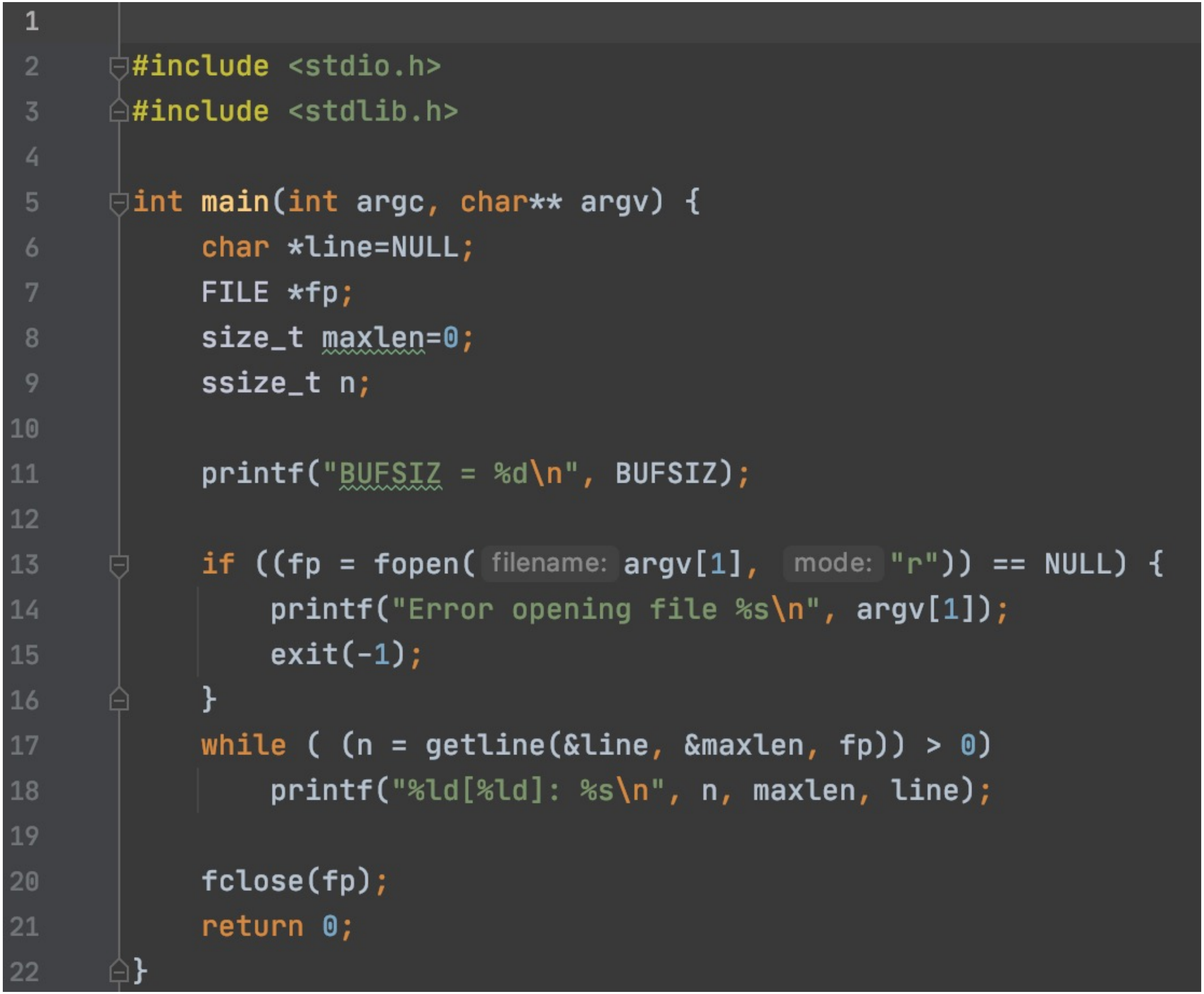
```
(base) mahmutunan@MacBook-Pro lecture15 % gcc -o getline1 getline1.c
(base) mahmutunan@MacBook-Pro lecture15 % ./getline1 test.txt
16: Some text line 1
6: Line 2
2: l3
```

getline()

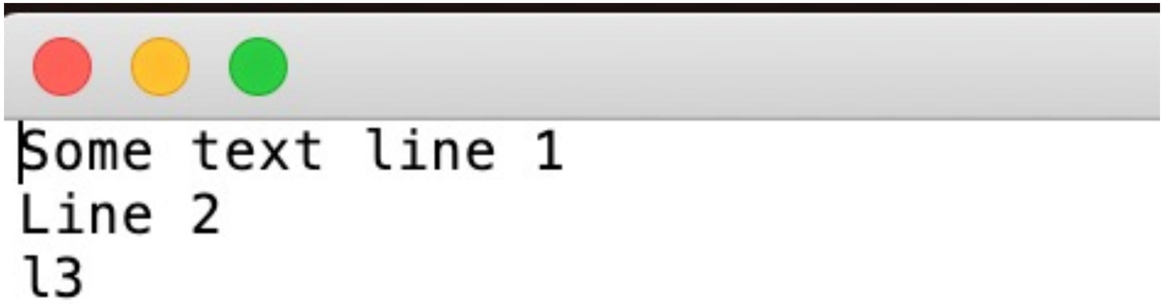
```
ssize_t getline(char **lineptr, size_t *n, FILE
*stream);
```

- **getline**() reads an entire line from *stream*, storing the address of the buffer containing the text into **lineptr*. The buffer is null-terminated and includes the newline character, if one was found.
- On success, **getline**() returns the number of characters read, including the delimiter character, but not including the terminating null byte ('\0')

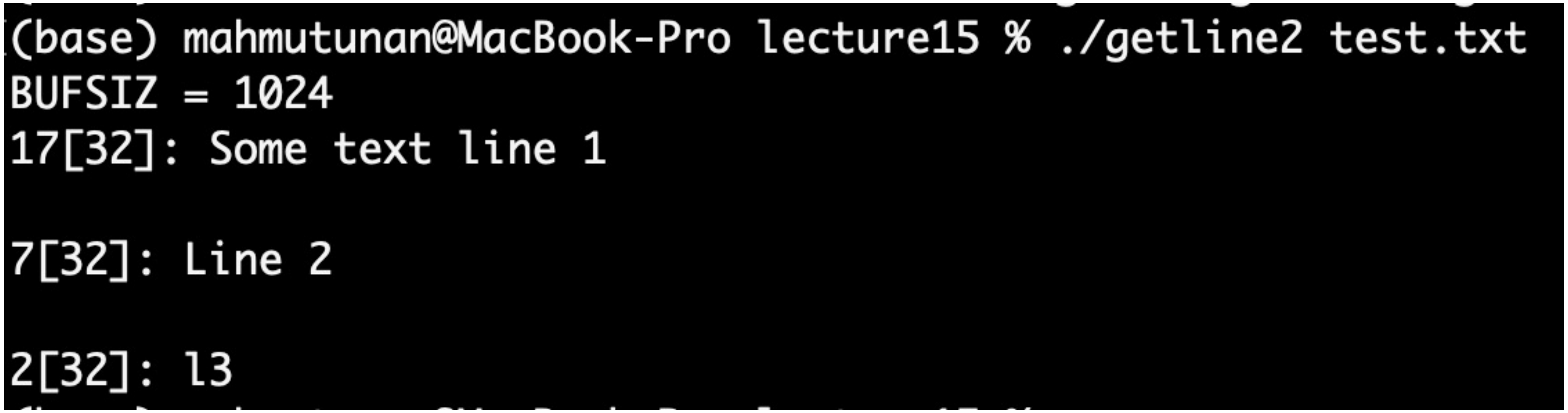
getline2.c



```

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

int main(int argc, char** argv) {
    char *line=NULL;
    FILE *fp;
    size_t maxlen=0;
    ssize_t n;

    printf("BUFSIZ = %d\n", BUFSIZ);

    if ((fp = fopen( filename: argv[1],  mode: "r")) == NULL) {
        printf("Error opening file %s\n", argv[1]);
        exit(-1);
    }
    while ( (n = getline(&line, &maxlen, fp)) > 0)
        printf("%ld[%ld]: %s\n", n, maxlen, line);

    fclose(fp);
    return 0;
}
```



```
Some text line 1
Line 2
l3
```



```
(base) mahmutunan@MacBook-Pro lecture15 % ./getline2 test.txt
BUFSIZ = 1024
17[32]: Some text line 1

7[32]: Line 2

2[32]: l3
```

getdelim()

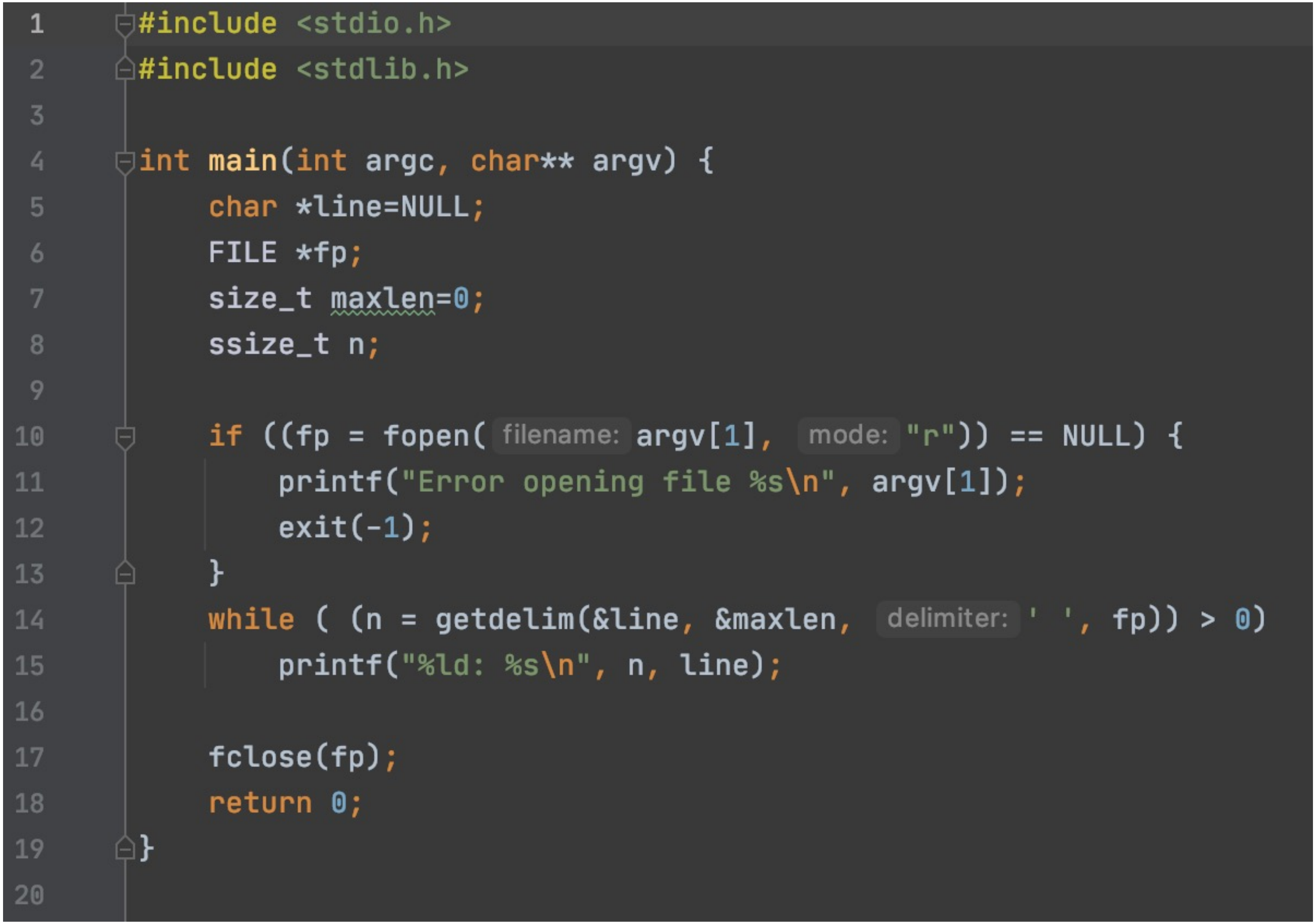
- `ssize_t getdelim(char **lineptr, size_t *n, int delim, FILE *stream);`

- getdelim() works like getline(), except that a line delimiter other than newline can be specified as the *delimiter* argument.

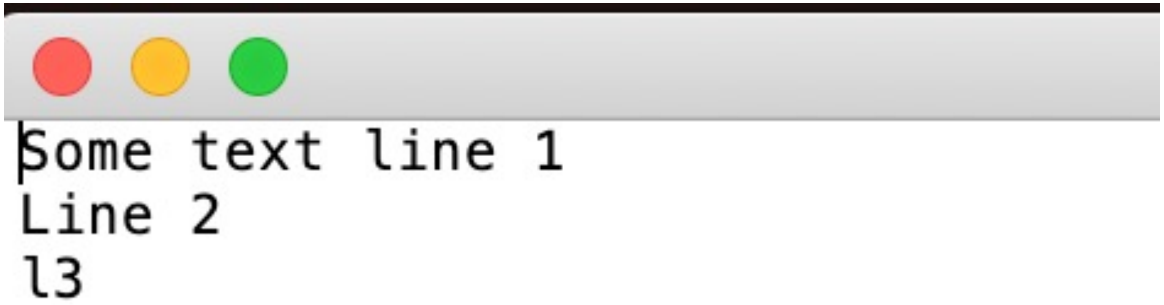
- getdelim() returns the number of characters read, including the delimiter character, but not including the terminating null byte

https://linux.die.net/man/3/getdelim

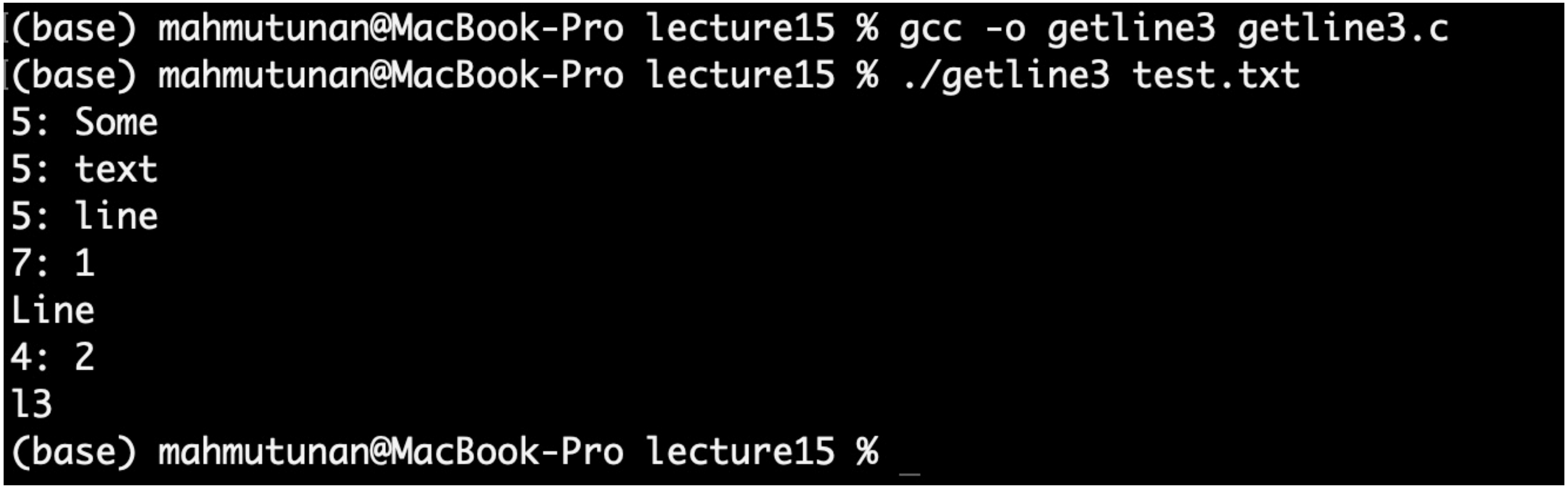
getline3.c



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

int main(int argc, char** argv) {
    char *line=NULL;
    FILE *fp;
    size_t maxlen=0;
    ssize_t n;

    if ((fp = fopen(argv[1], "r")) == NULL) {
        printf("Error opening file %s\n", argv[1]);
        exit(-1);
    }
    while ( (n = getdelim(&line, &maxlen, ' ', fp)) > 0)
        printf("%ld: %s\n", n, line);

    fclose(fp);
    return 0;
}
```



```
Some text line 1
Line 2
l3
```



```
(base) mahmutunan@MacBook-Pro lecture15 % gcc -o getline3 getline3.c
(base) mahmutunan@MacBook-Pro lecture15 % ./getline3 test.txt
5: Some
5: text
5: line
7: 1
Line
4: 2
l3
(base) mahmutunan@MacBook-Pro lecture15 % _
```

gets()

- `char *gets(char *str)`

- The C library function char *gets(char *str) reads a line from stdin and stores it into the string pointed to by str.
- It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

- It reads string from standard input and prints the entered string, but it suffers from Buffer Overflow as gets() doesn't do any array bound testing.
- gets() keeps on reading until it sees a newline character.
- To avoid Buffer Overflow, fgets() should be used instead of gets() as fgets() makes sure that not more than MAX_LIMIT characters are read.

https://www.geeksforgeeks.org/gets-is-risky-to-use/

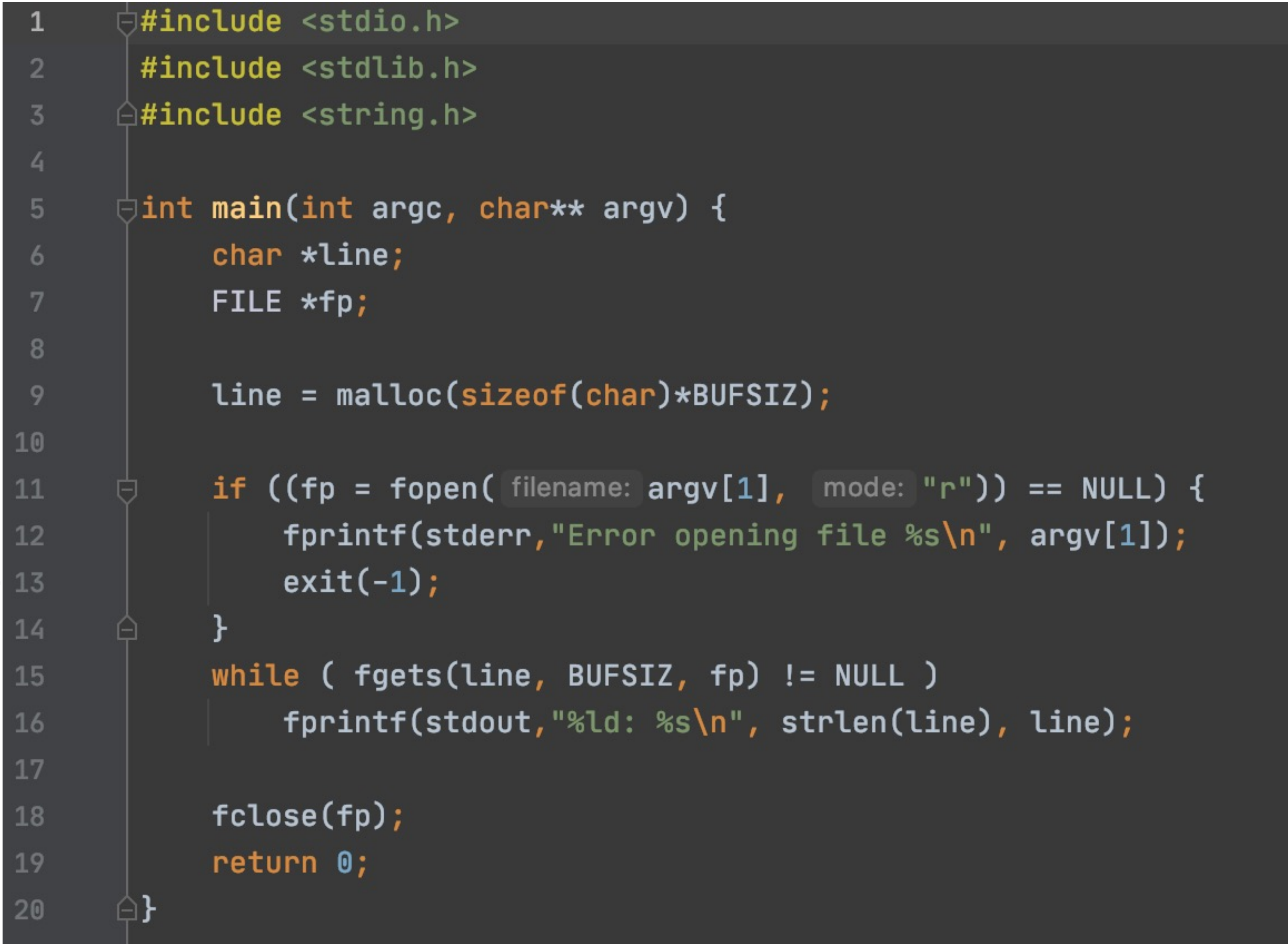
fgets()

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

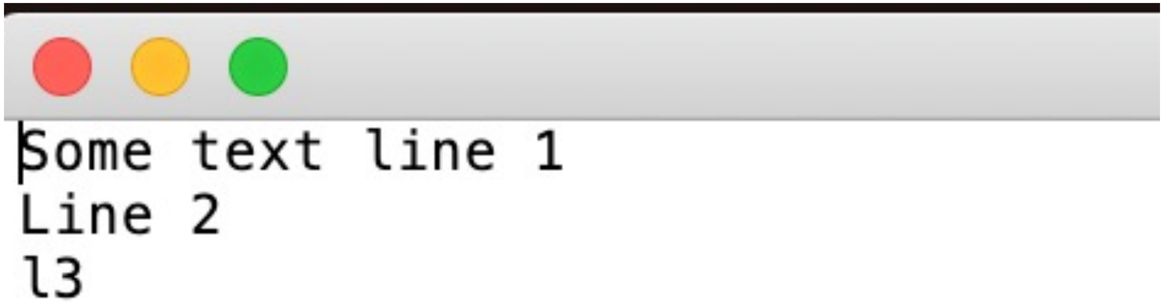
- **str** – This is the pointer to an array of chars where the string read is stored.
- **n** – This is the maximum number of characters to be read (including the final null-character). Usually, the length of the array passed as str is used.
- **stream** – This is the pointer to a FILE object that identifies the stream where characters are read from.
- On success, the function returns the same str parameter. If the End-of-File is encountered and no characters have been read, the contents of str remain unchanged and a null pointer is returned

https://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm#:~:text=The%20C%20library%20function%20char,is%20reached%2C%20whichever%20comes%20first.

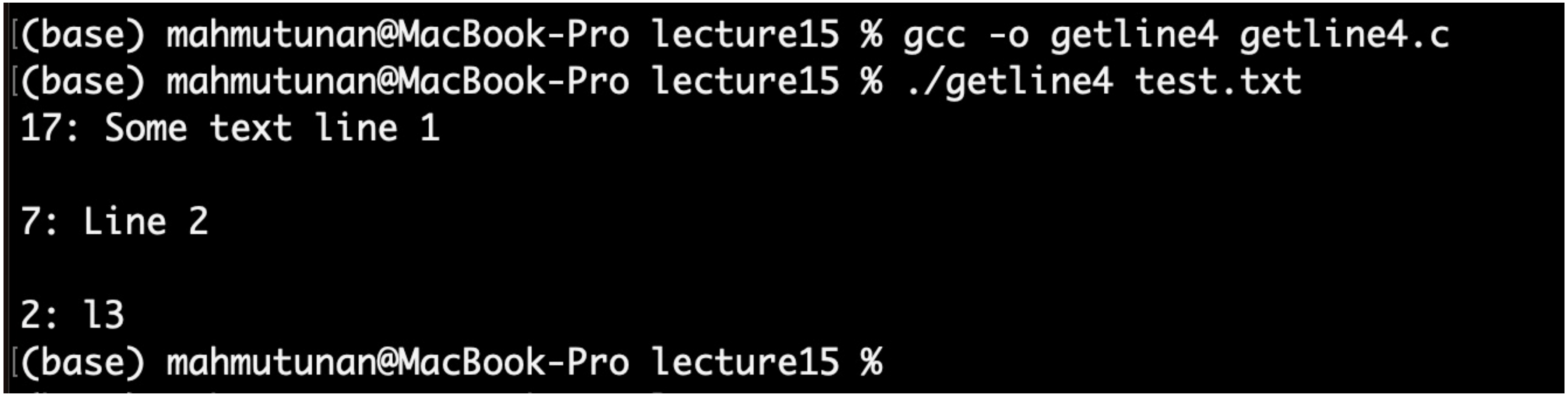
getline4.c



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

int main(int argc, char** argv) {
    char *line;
    FILE *fp;

    line = malloc(sizeof(char)*BUFSIZ);

    if ((fp = fopen( filename: argv[1],  mode: "r")) == NULL) {
        fprintf(stderr,"Error opening file %s\n", argv[1]);
        exit(-1);
    }
    while ( fgets(line, BUFSIZ, fp) != NULL )
        fprintf(stdout,"%ld: %s\n", strlen(line), line);

    fclose(fp);
    return 0;
}
```



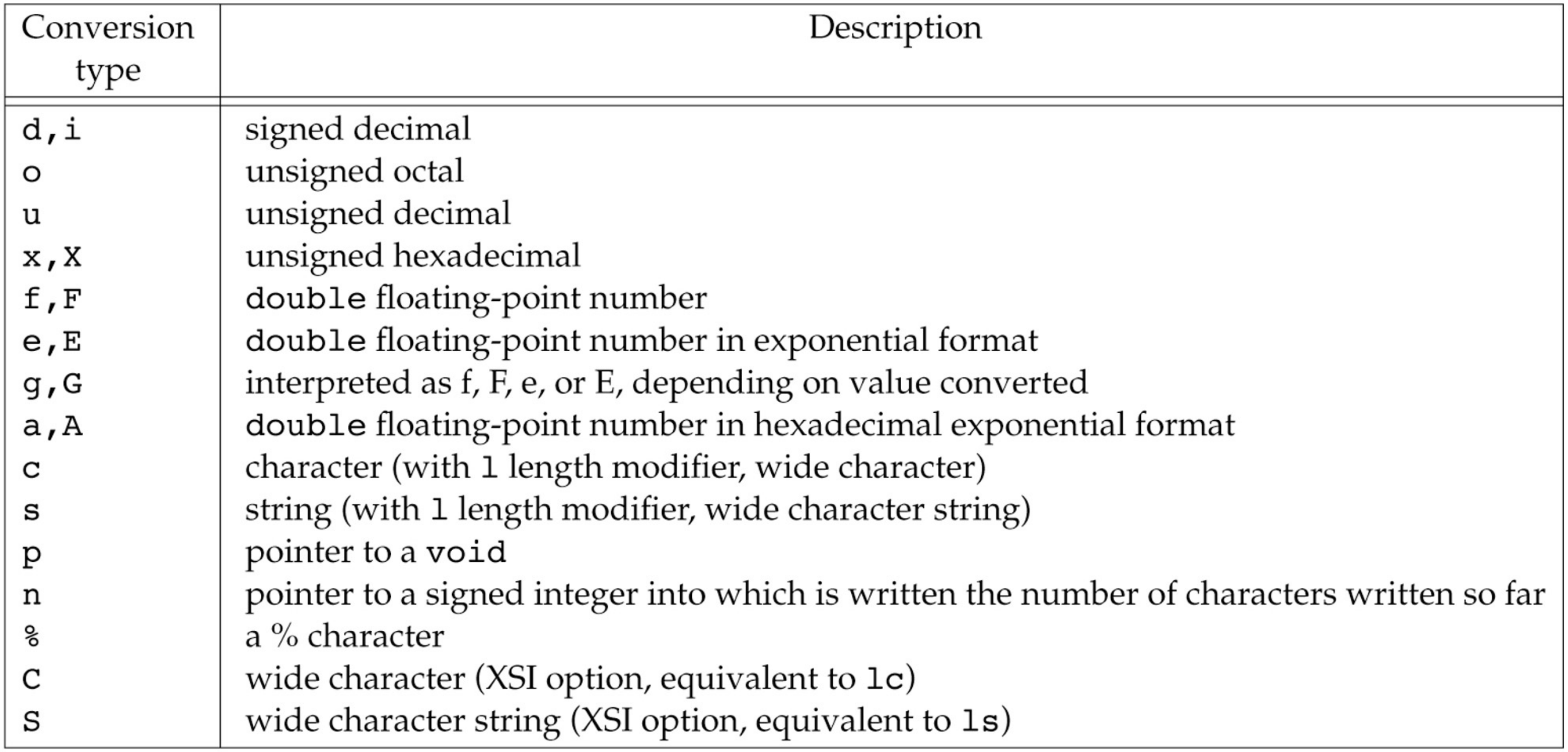
```
Some text line 1
Line 2
l3
```



```
[(base) mahmutunan@MacBook-Pro lecture15 % gcc -o getline4 getline4.c
[(base) mahmutunan@MacBook-Pro lecture15 % ./getline4 test.txt
17: Some text line 1

7: Line 2

2: l3
[(base) mahmutunan@MacBook-Pro lecture15 %
```

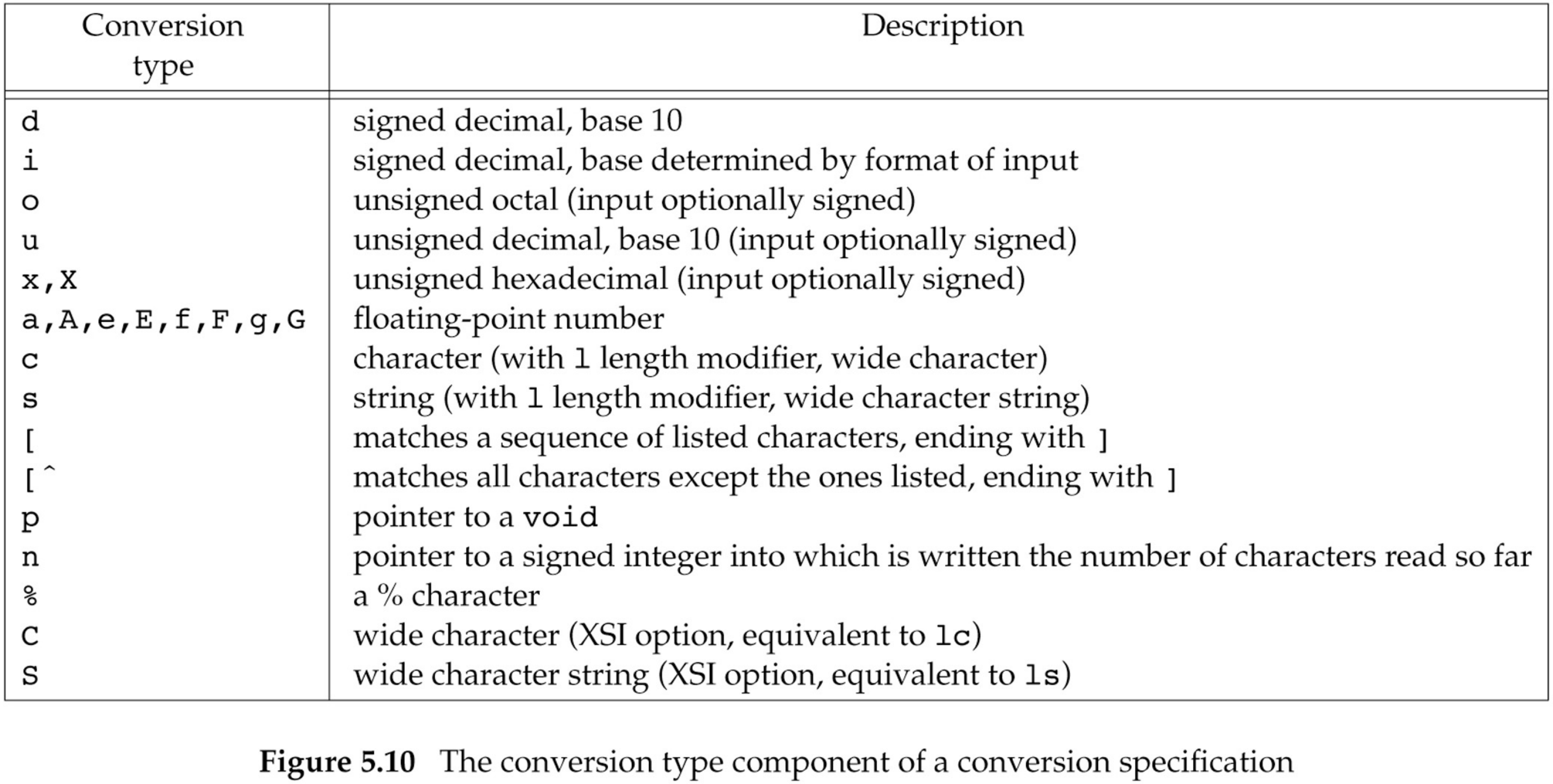
fscanf()

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



Conversion type	Description
`d,i`	signed decimal
`o`	unsigned octal
`u`	unsigned decimal
`x,X`	unsigned hexadecimal
`f,F`	`double` floating-point number
`e,E`	`double` floating-point number in exponential format
`g,G`	interpreted as f, F, e, or E, depending on value converted
`a,A`	`double` floating-point number in hexadecimal exponential format
`c`	character (with `l` length modifier, wide character)
`s`	string (with `l` length modifier, wide character string)
`p`	pointer to a `void`
`n`	pointer to a signed integer into which is written the number of characters written so far
`%`	a % character
`C`	wide character (XSI option, equivalent to `lc`)
`S`	wide character string (XSI option, equivalent to `ls`)

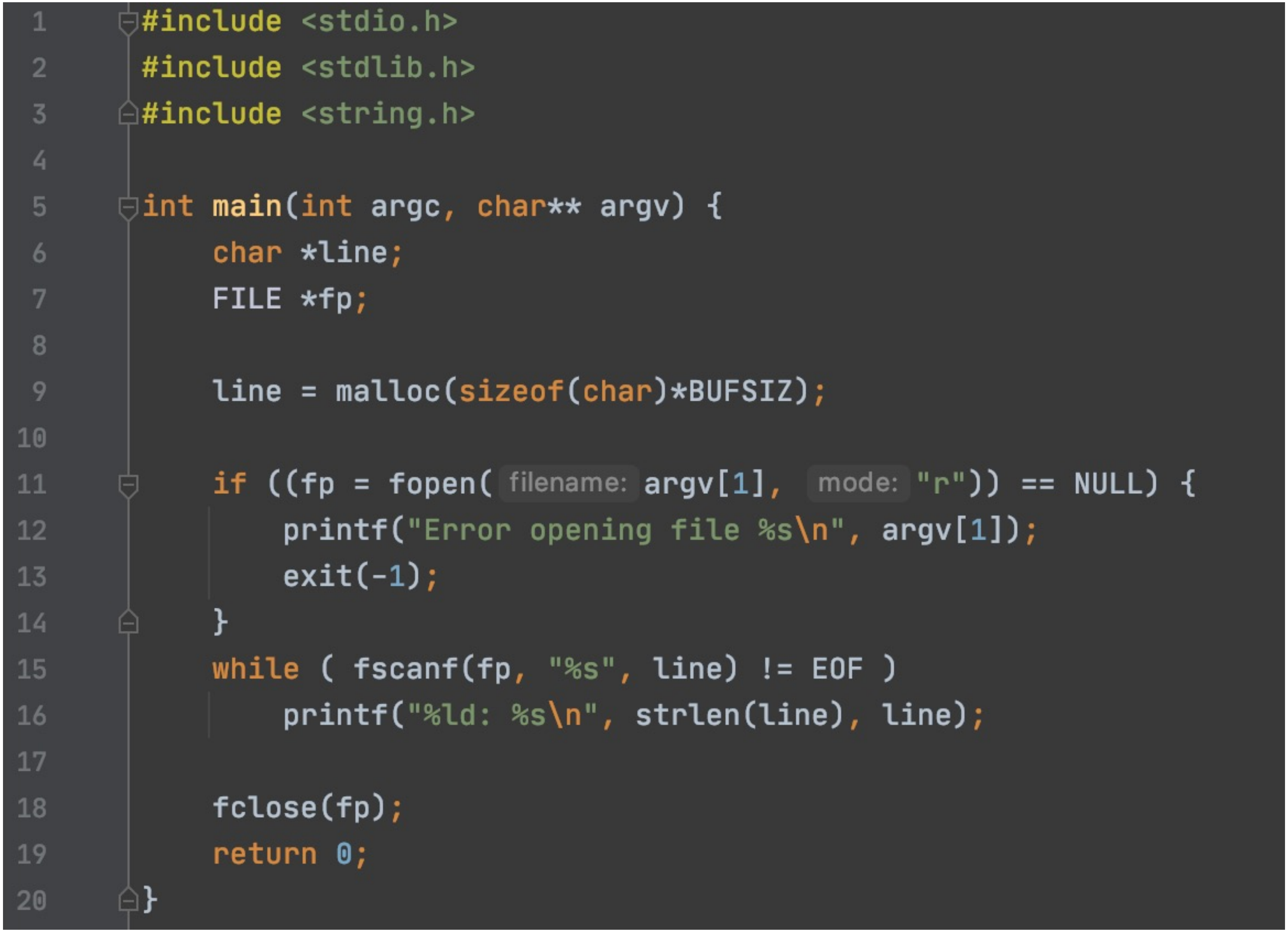
Figure 5.9 The conversion type component of a conversion specification



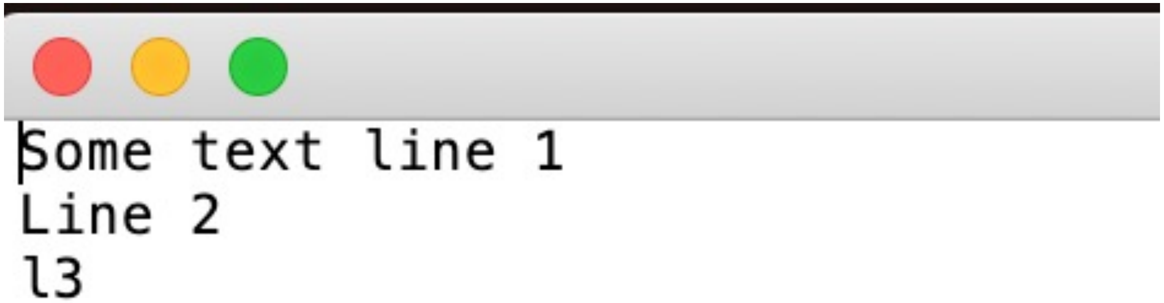
Conversion type	Description
`d`	signed decimal, base 10
`i`	signed decimal, base determined by format of input
`o`	unsigned octal (input optionally signed)
`u`	unsigned decimal, base 10 (input optionally signed)
`x,X`	unsigned hexadecimal (input optionally signed)
`a,A,e,E,f,F,g,G`	floating-point number
`c`	character (with `l` length modifier, wide character)
`s`	string (with `l` length modifier, wide character string)
`[`	matches a sequence of listed characters, ending with `]`
`[^`	matches all characters except the ones listed, ending with `]`
`p`	pointer to a `void`
`n`	pointer to a signed integer into which is written the number of characters read so far
`%`	a % character
`C`	wide character (XSI option, equivalent to `lc`)
`S`	wide character string (XSI option, equivalent to `ls`)

Figure 5.10 The conversion type component of a conversion specification

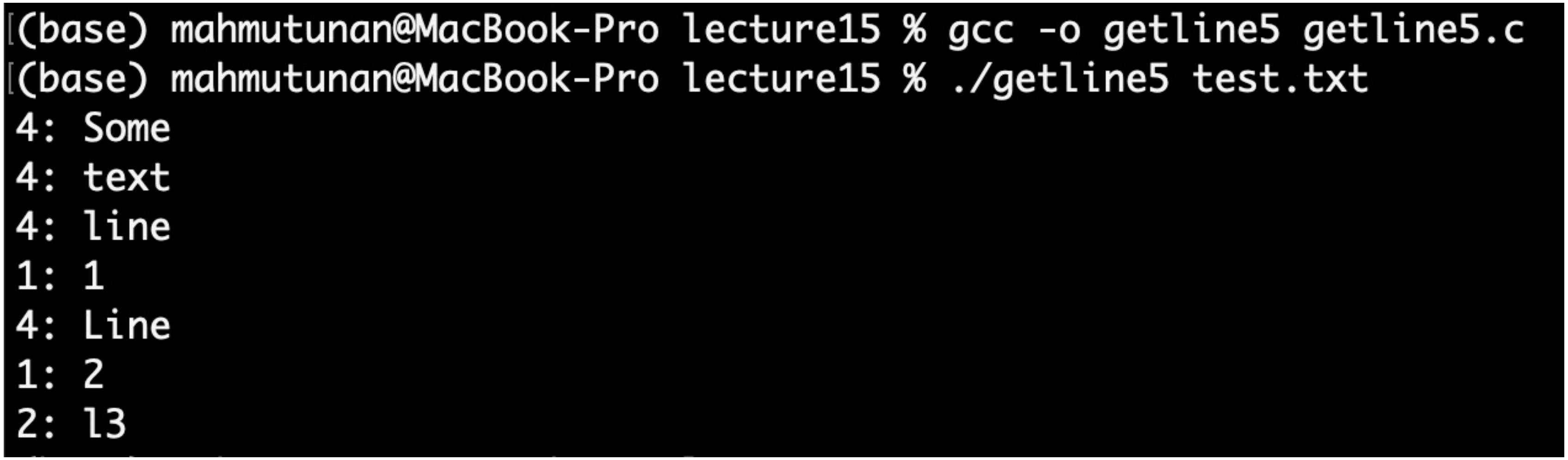
getline5.c



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

int main(int argc, char** argv) {
    char *line;
    FILE *fp;

    line = malloc(sizeof(char)*BUFSIZ);

    if ((fp = fopen( filename: argv[1],  mode: "r")) == NULL) {
        printf("Error opening file %s\n", argv[1]);
        exit(-1);
    }
    while ( fscanf(fp, "%s", line) != EOF )
        printf("%ld: %s\n", strlen(line), line);

    fclose(fp);
    return 0;
}
```



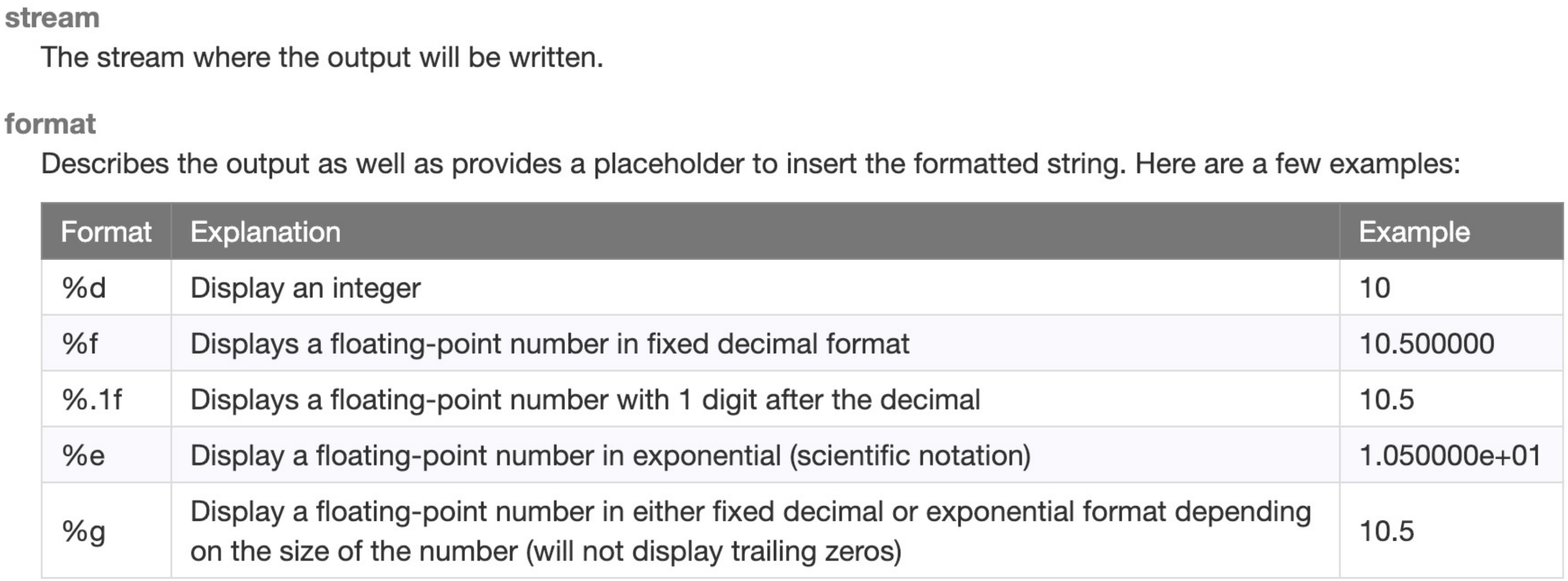
```
Some text line 1
Line 2
l3
```



```
[(base) mahmutunan@MacBook-Pro lecture15 % gcc -o getline5 getline5.c
[(base) mahmutunan@MacBook-Pro lecture15 % ./getline5 test.txt
4: Some
4: text
4: line
1: 1
4: Line
1: 2
2: l3
```

fprintf()

```
int fprintf(FILE *stream, const char *format, ...)
```



stream

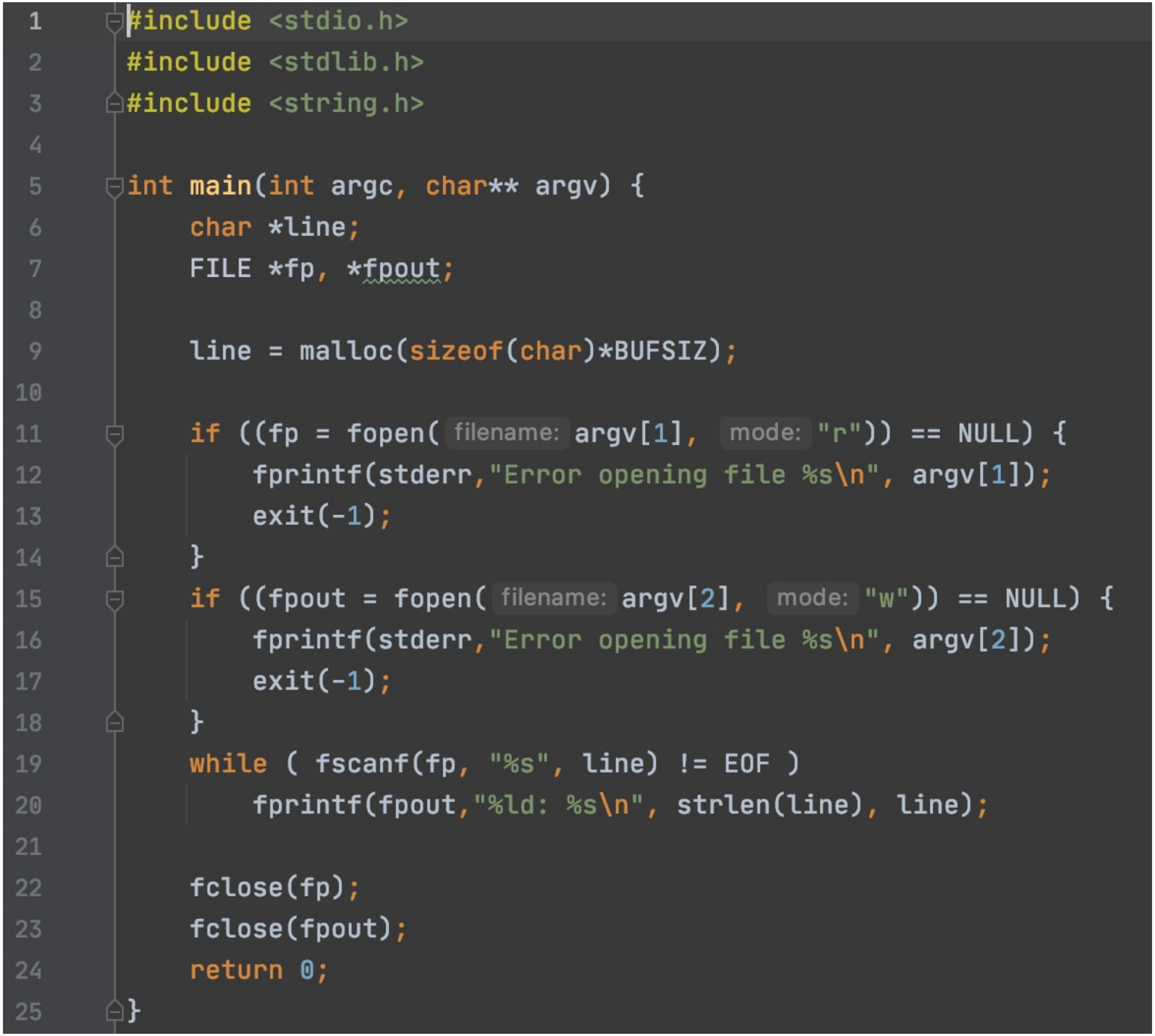
The stream where the output will be written.

format

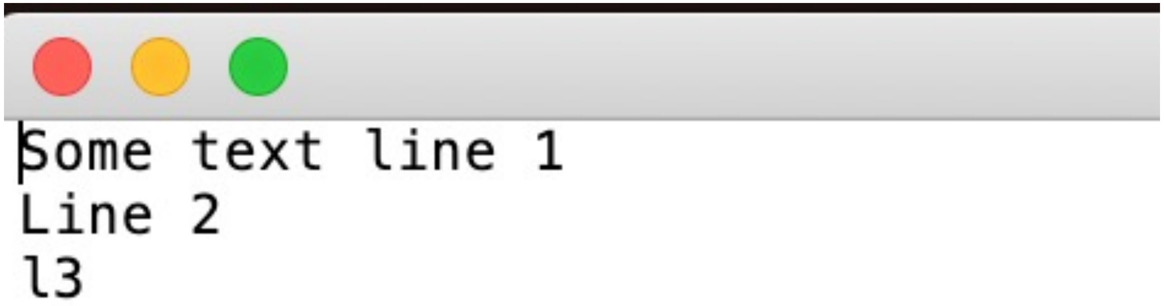
Describes the output as well as provides a placeholder to insert the formatted string. Here are a few examples:

Format	Explanation	Example
%d	Display an integer	10
%f	Displays a floating-point number in fixed decimal format	10.500000
%.1f	Displays a floating-point number with 1 digit after the decimal	10.5
%e	Display a floating-point number in exponential (scientific notation)	1.050000e+01
%g	Display a floating-point number in either fixed decimal or exponential format depending on the size of the number (will not display trailing zeros)	10.5

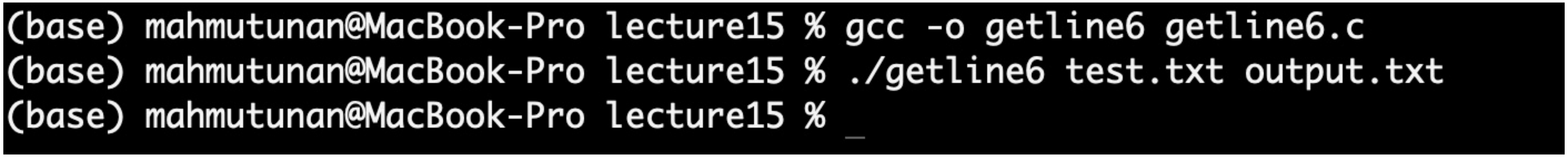
getline6.c



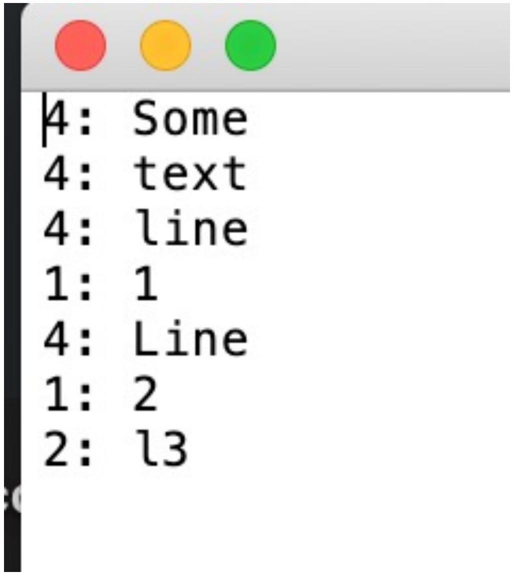
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char** argv) {
    char *line;
    FILE *fp, *fpout;

    line = malloc(sizeof(char)*BUFSIZ);

    if ((fp = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr,"Error opening file %s\n", argv[1]);
        exit(-1);
    }
    if ((fpout = fopen(argv[2], "w")) == NULL) {
        fprintf(stderr,"Error opening file %s\n", argv[2]);
        exit(-1);
    }
    while ( fscanf(fp, "%s", line) != EOF )
        fprintf(fpout,"%ld: %s\n", strlen(line), line);

    fclose(fp);
    fclose(fpout);
    return 0;
}
```



```
Some text line 1
Line 2
l3
```



```
(base) mahmutunan@MacBook-Pro lecture15 % gcc -o getline6 getline6.c
(base) mahmutunan@MacBook-Pro lecture15 % ./getline6 test.txt output.txt
(base) mahmutunan@MacBook-Pro lecture15 % _
```



```
4: Some
4: text
4: line
1: 1
4: Line
1: 2
2: l3
```

Go Extra mile

- You can use the corresponding man page to find out more about each of the functions used above.
- You can also extend the above examples to use *putc, puts, putchar,*
  fputc, and *fputs* functions to write the output.