assign3: A Heap of Fun

assign3: A Heap of Fun

Debugging Guide

Course Site -> Handouts -> Debugging Guide

- It’s easy to make mistakes with pointers, arrays, and heap memory!
- Checklist/step-by-step guide for diagnosing your problem and collecting more information

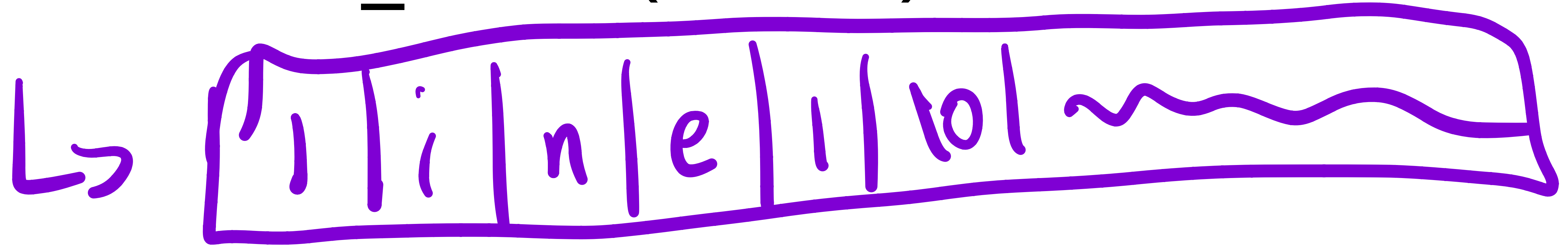
read_line

```
char *read_line(FILE *file_pointer)
```

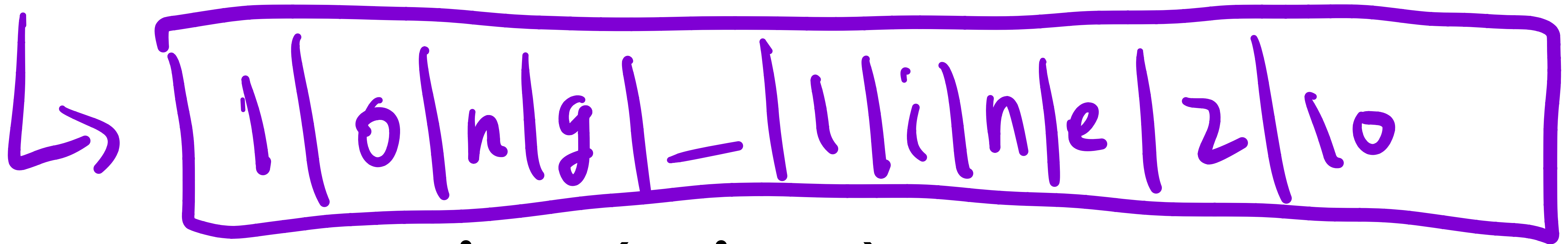
lines.txt

```
line1
long_line2
line3
```

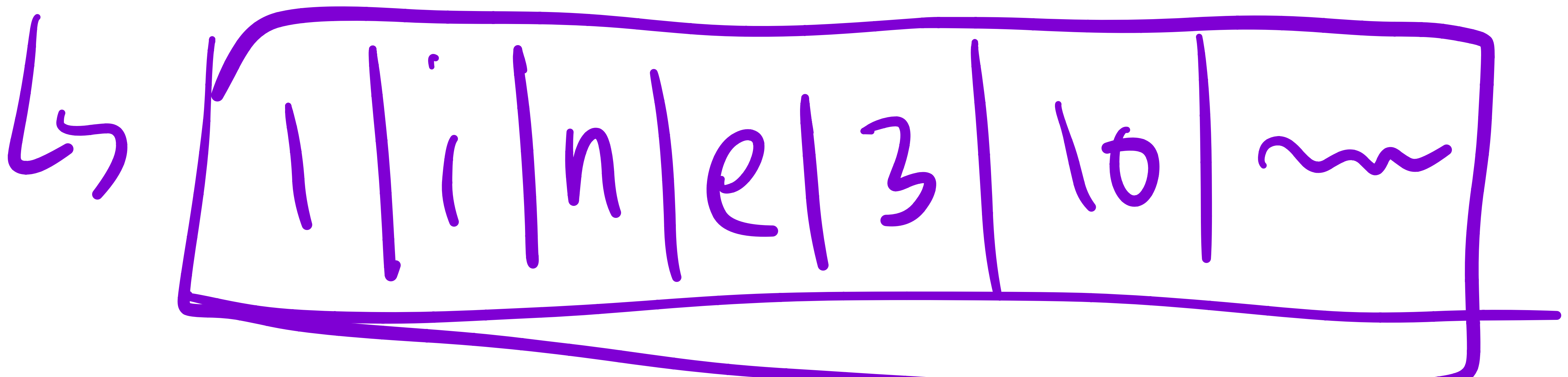
```
1. read_line(file)
```



```
2. read_line(file)
```



```
3. read_line(file)
```



read_line

man fgets

```
char *fgets(char *buf, size_t buflen, FILE *file)
```

- Standard C function for reading lines from a file but trickier to use
- Reads at most *buflen - 1* characters from the next line in the file and copies them into the buffer
    - Includes the newline character if it exists
    - Always null-terminates the buffer

read_line

fgets example

file

lines.txt

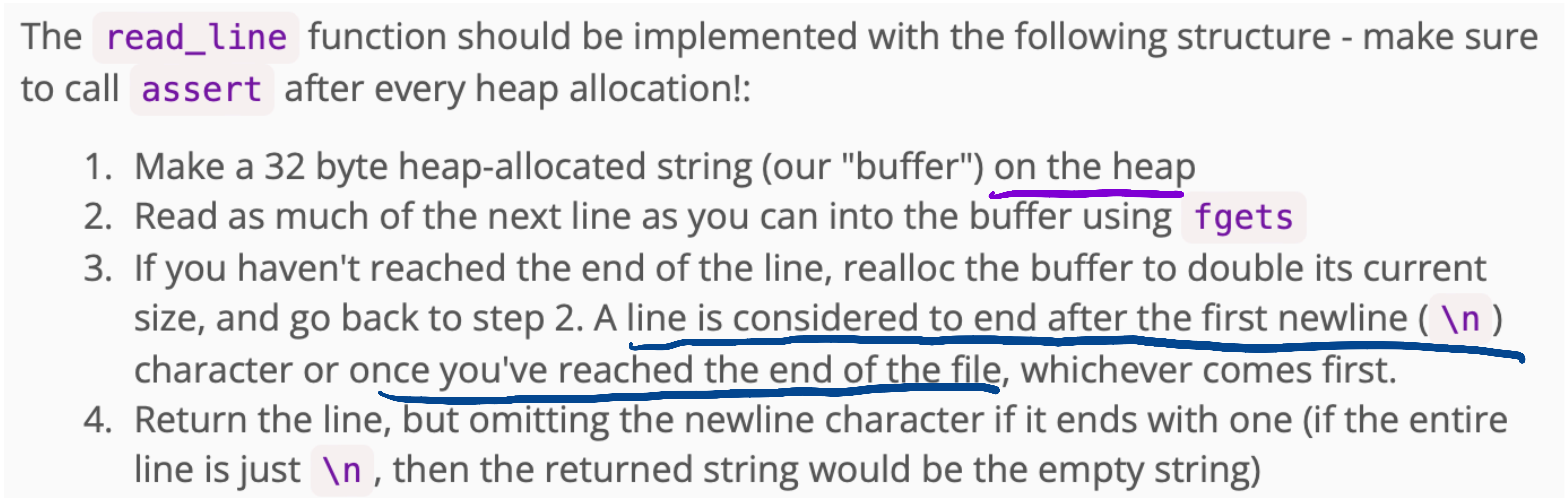
```
line1\n
long_line2\n
line3
```

Buffers (8 bytes each)

l	i	n	e	1	\n	\0	
l	o	n	g	_	l	i	\0
n	e	2	\n	\0			
l	i	n	e	3	\0		

read_line

Implementation structure



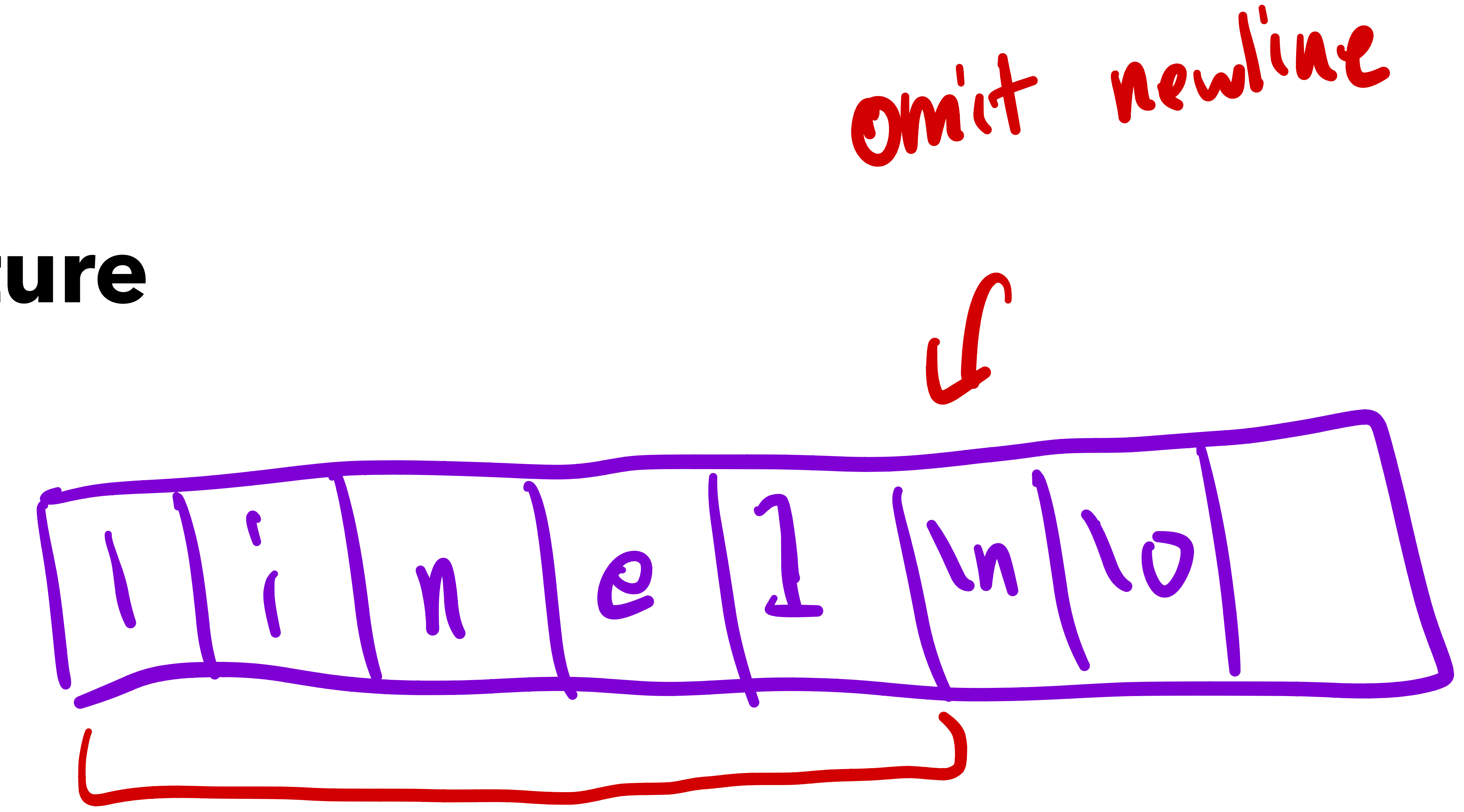
The `read_line` function should be implemented with the following structure - make sure to call `assert` after every heap allocation!:

1. Make a 32 byte heap-allocated string (our "buffer") on the heap
2. Read as much of the next line as you can into the buffer using `fgets`
3. If you haven't reached the end of the line, realloc the buffer to double its current size, and go back to step 2. A line is considered to end after the first newline (`\n`) character or once you've reached the end of the file, whichever comes first.
4. Return the line, but omitting the newline character if it ends with one (if the entire line is just `\n` , then the returned string would be the empty string)

read_line

Implementation structure

```
line1\n
long_line2\n
line3
```

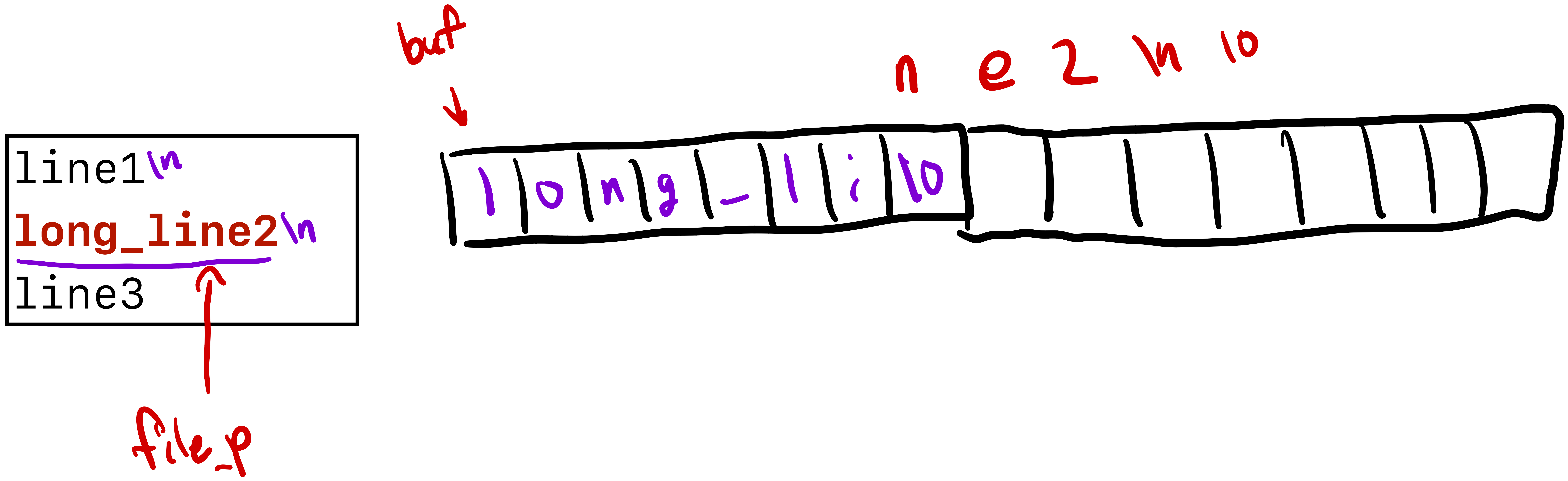


how to know if entire line?

read_line

Implementation structure

continue where line left off



mytail

Implementing the UNIX tail command

`lines.txt`

```
line1
line2
line3
line4
line5
line6
line7
line8
```

```
$ ./mytail -3 lines.txt
line6
line7
line8
```

```
$ ./mytail -5 lines.txt
line4
line5
line6
line7
line8
```

num lines to print

mytail

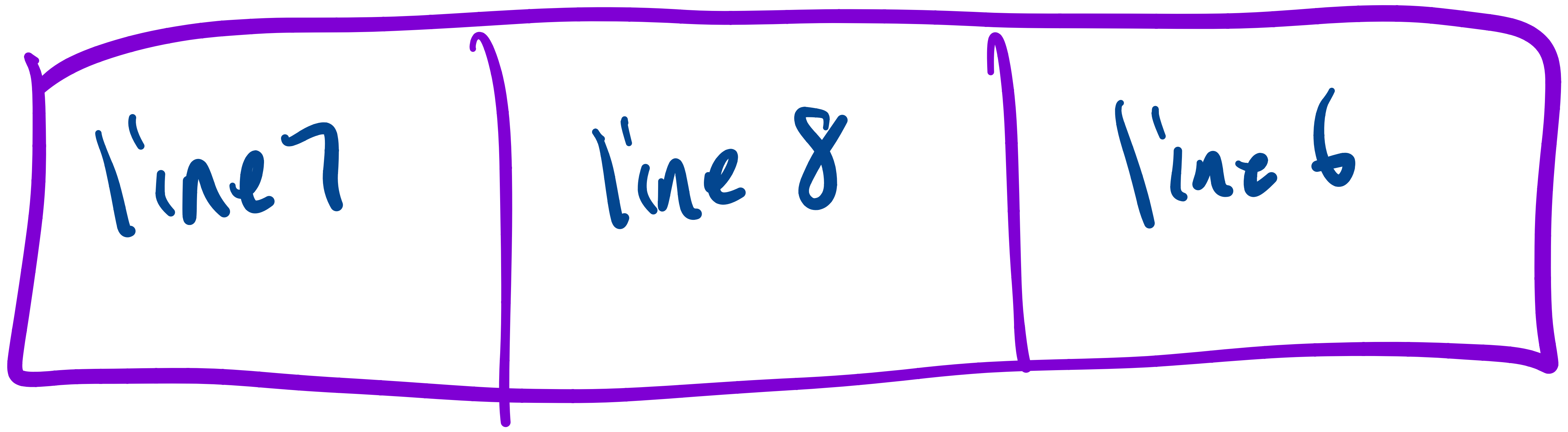
```
void print_last_n(FILE *file_pointer, int n)
```

- Use your `read_line()` function!
- DON'T try to store all the lines!
- Build a circular queue of N lines using a stack array

challenge

how to print lines in correct order?

n=3



myuniq

Implementing the UNIX uniq command

colors.txt

```
red
blue
red
green
green
red
```

```
$ ./myuniq colors.txt
3 red
1 blue
2 green
```

myuniq

`void print_uniq_lines(FILE *file_pointer)`

- Use your `read_line()` function
- **Challenge**: keeping track of unique lines and their frequencies
  - Use a struct to bundle associated data together
  - Make an array of structs to store multiple instances of those data bundles

```
{
  Nick
  107
}
```

```
{
  John
  111
}
```

```
{ Jerry
  109
}
```

```
struct Lecturer {
  char * name;
  int code;
}
```

General Tips

- Debugging guide (Course page -> Handouts -> Debugging guide)
- Working with heap memory
  - Remember to free whatever you've allocated once you're done with it
  - Run with valgrind to catch memory leaks
- Use GDB to examine memory and make sure your buffers/arrays contain what you expect