Sample Slides for Week 4 (Memory)

Welcome!

Here you'll find sample slides to adopt or adapt when teaching Week 4, on topics such as pointers, hexadecimal, malloc/free, and File I/O.

Some slides contain speaker notes to illustrate why the sample slides take a certain approach to illustrating a concept or leading an exercise. You're welcome to modify these slides as you see fit, though do try to keep some of the same elements of active learning that have been included in these samples.

This is CS50

Think.
Pair.
Share.

- What are **pointers**, and how can we become familiar with their **syntax**?
- How can we **read** and **write** data from a file?
- What is **dynamic memory**, and how should we use it?

Pointers

Variables

```
int calls = 4;
```

Variables

```
int calls = 4;
```

name

calls

Variables

```
int calls = 4;
```

type

calls

Variables

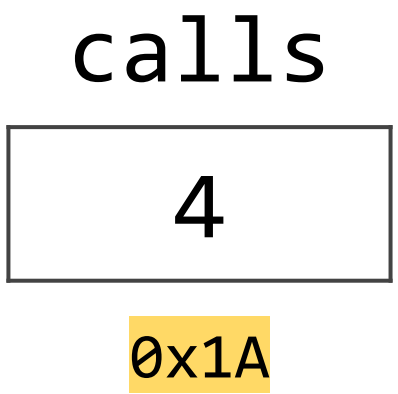
```
int calls = 4;
```

value

calls

Variables

```
int calls = 4;
```



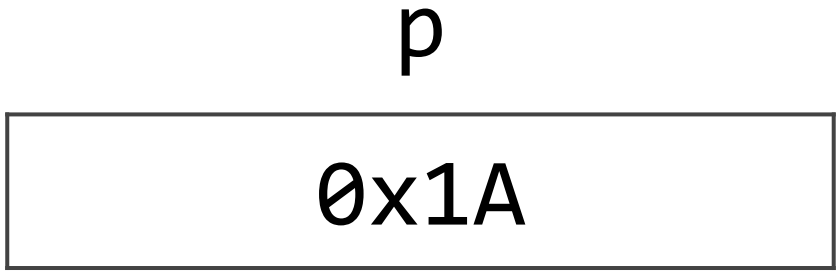
Pointers

```
int *p = 0x1A;
```

Pointers

```
int *p = 0x1A;
```

name



Pointers

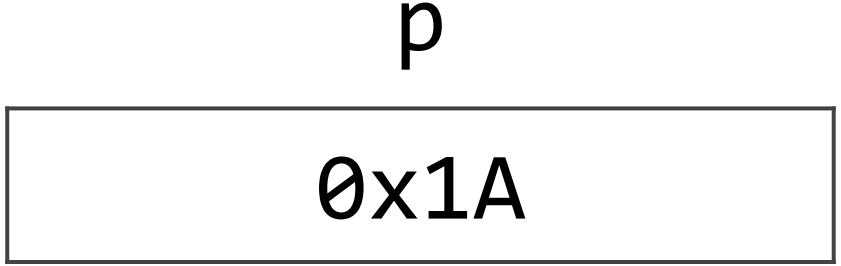
```
int *p = 0x1A;
```

type

Pointers

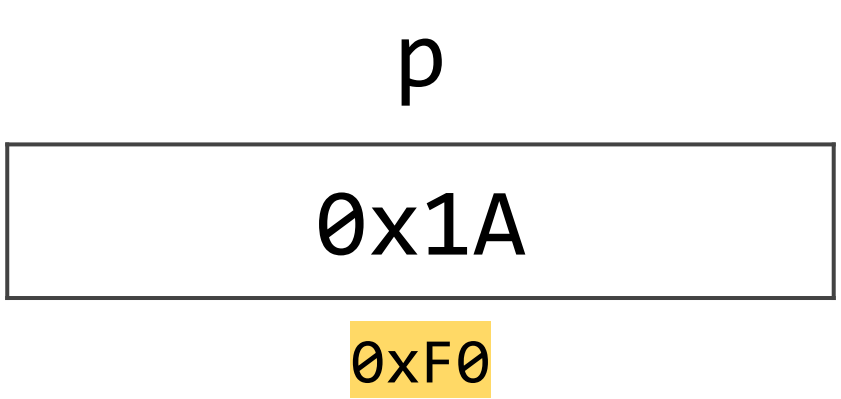
```
int *p = 0x1A;
```

value



Pointers

```
int *p = 0x1A;
```



Pointer Syntax

`calls;`

"value of"

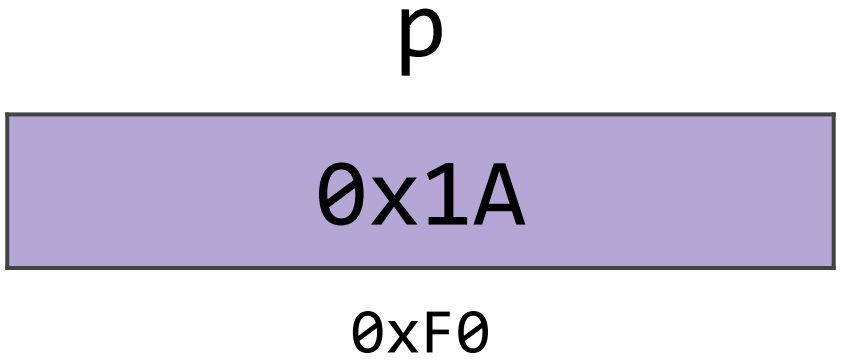
`calls`

`0x1A`

Pointer Syntax

```
p;
```

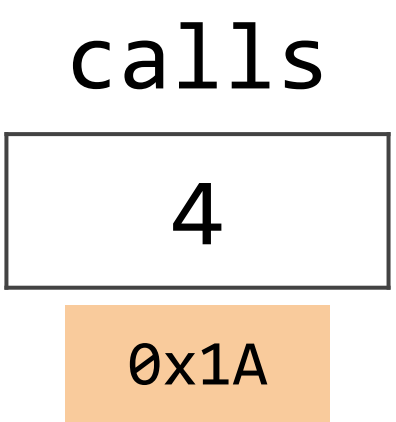
"value of"



Pointer Syntax

`&calls;`

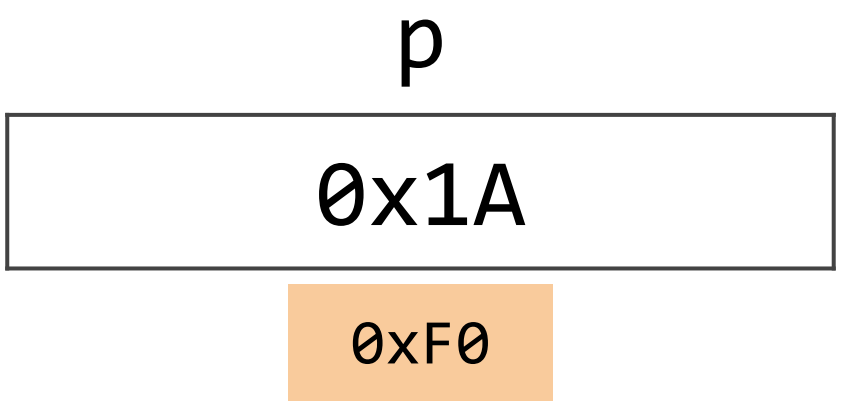
"address of"



Pointer Syntax

`&p;`

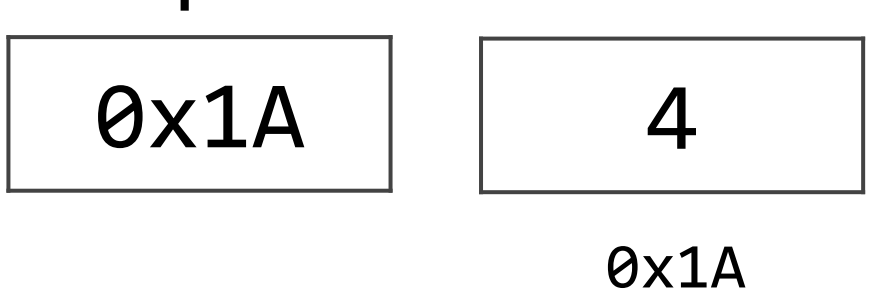
"address of"



Pointer Syntax

```
*p;
```

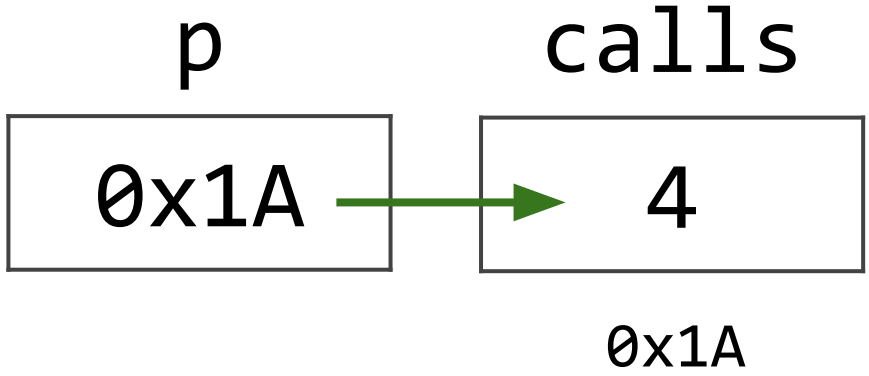
"**go to** the value at address stored in p"



Pointer Syntax

```
*p;
```

"**go to** the value at address stored in p"



`type *` is a pointer that stores the address of a **`type`**.

`*x` takes a pointer **`x`** and goes to the address stored at that pointer.

`&x` takes **`x`** and gets its address.

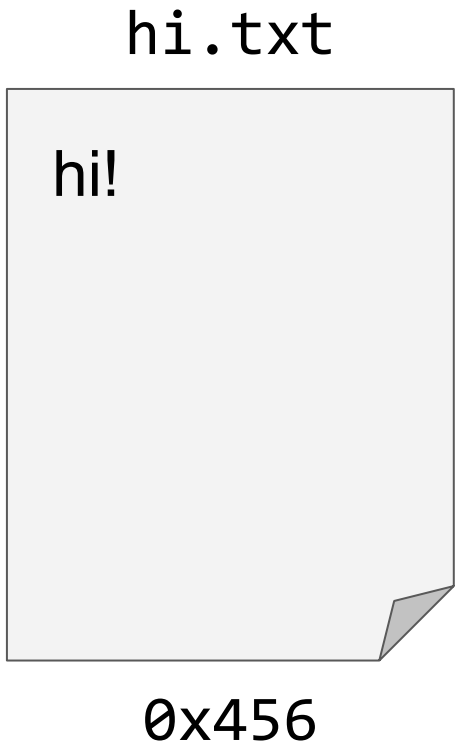
Pointer Prediction Exercise

Go to **[INSERT SHORTENED URL]**.

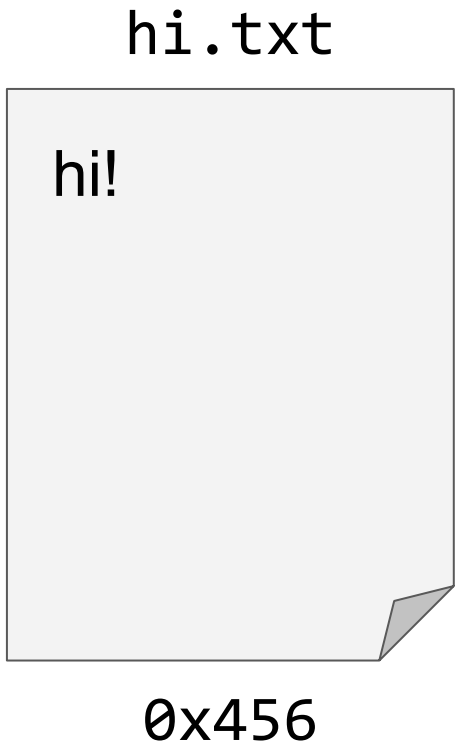
Visualize the code on the left, step by step. How do the values of the variables and pointers evolve? It's okay to use made-up addresses.

What will the final values for each variable or pointer be? Download, compile, and run **pointers.c** in VS Code to find out.

File I/O


hi.txt
hi!
0x456

```
FILE *input = fopen("hi.txt", "r");
```

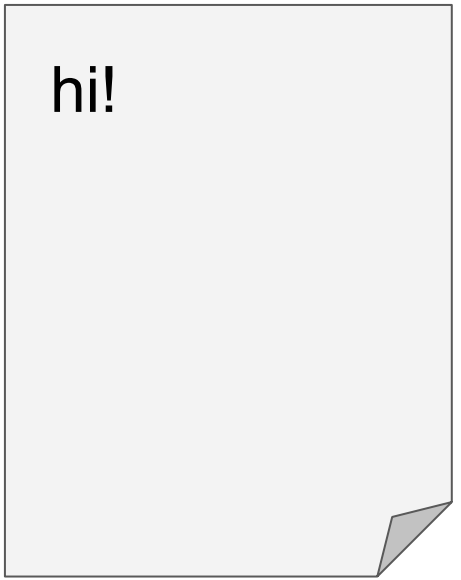


```
FILE *input = fopen("hi.txt", "r");
```

name

input

hi.txt



0x456

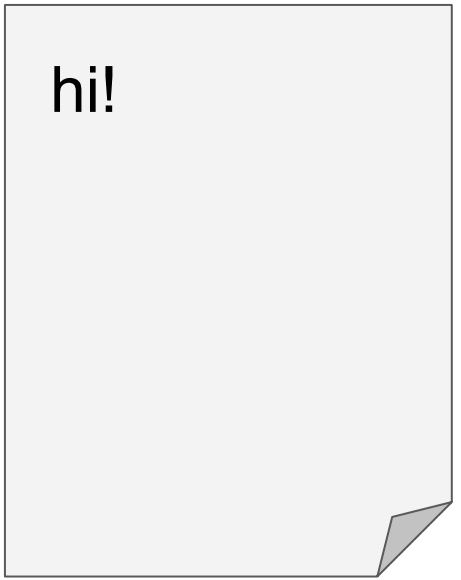
```
FILE *input = fopen("hi.txt", "r");
```

type

input

?

hi.txt



0x456

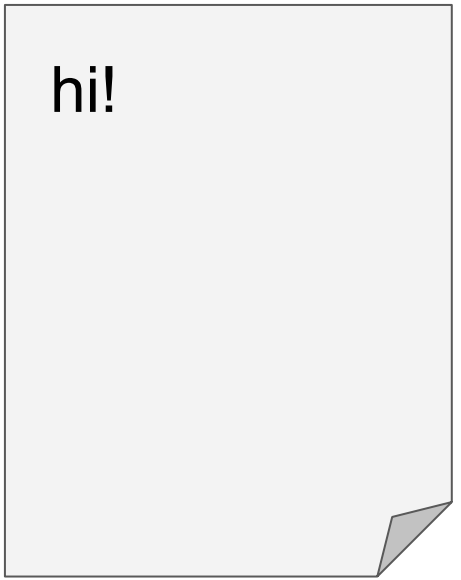
```
FILE *input = fopen("hi.txt", "r");
```

value

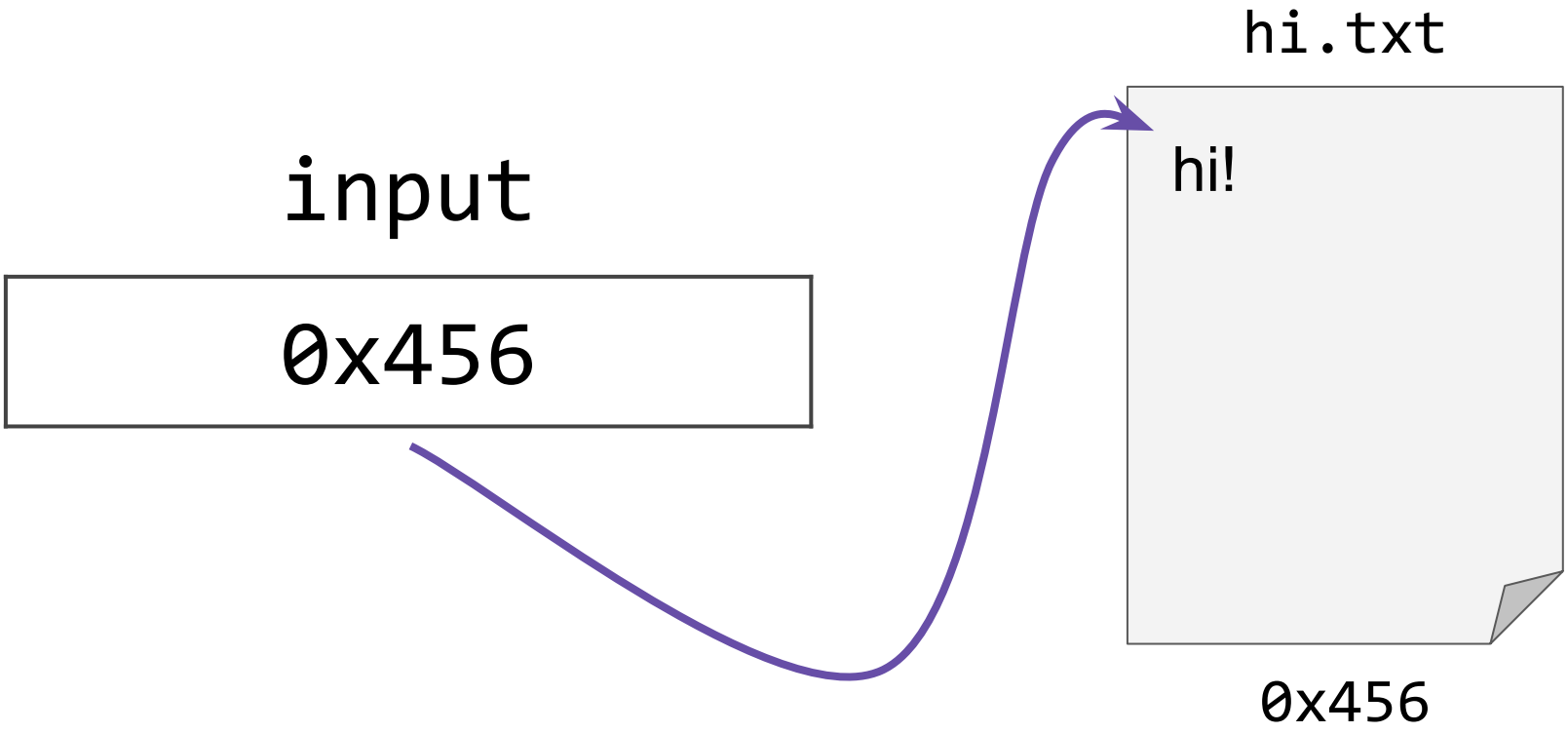
hi.txt

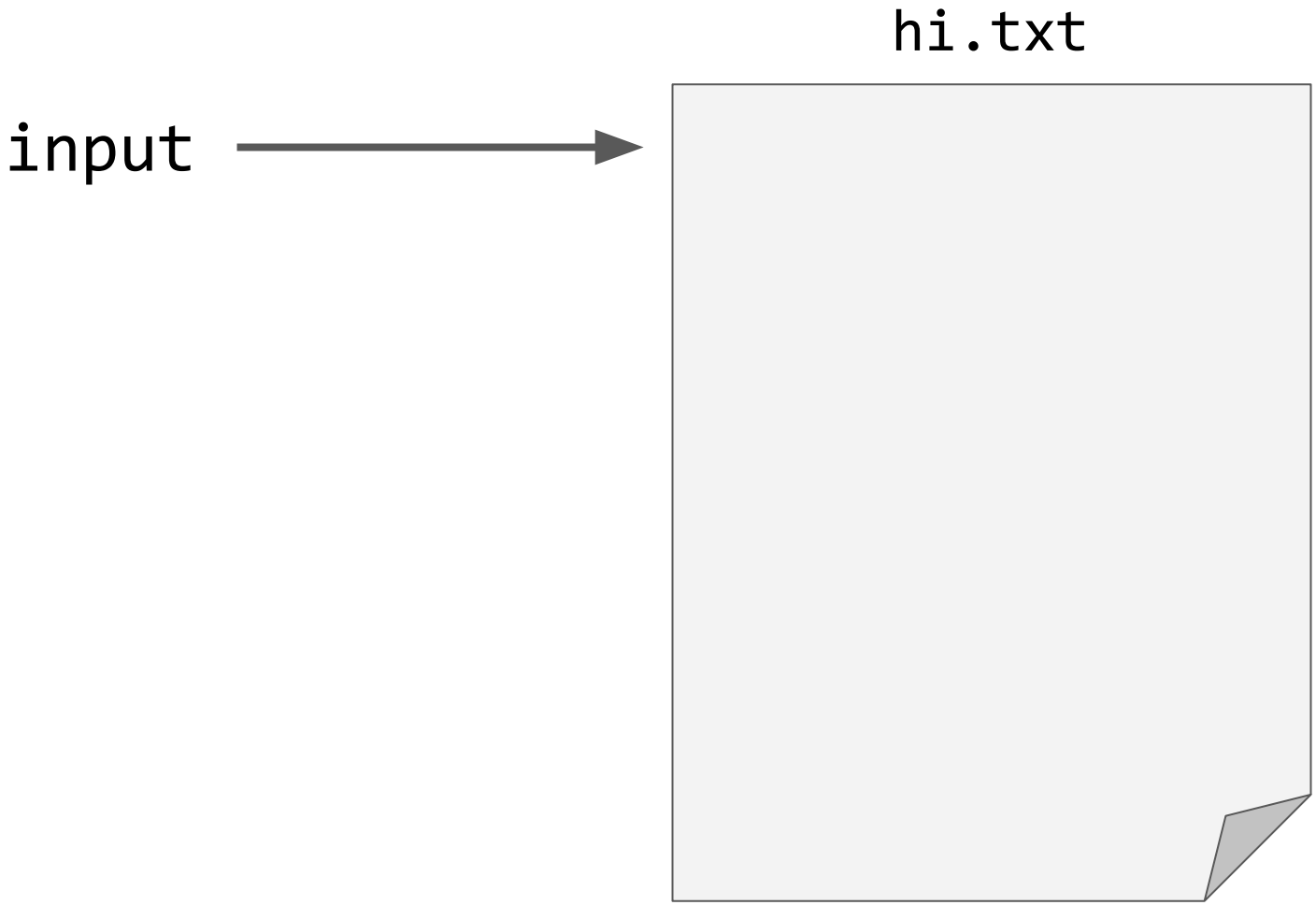
input

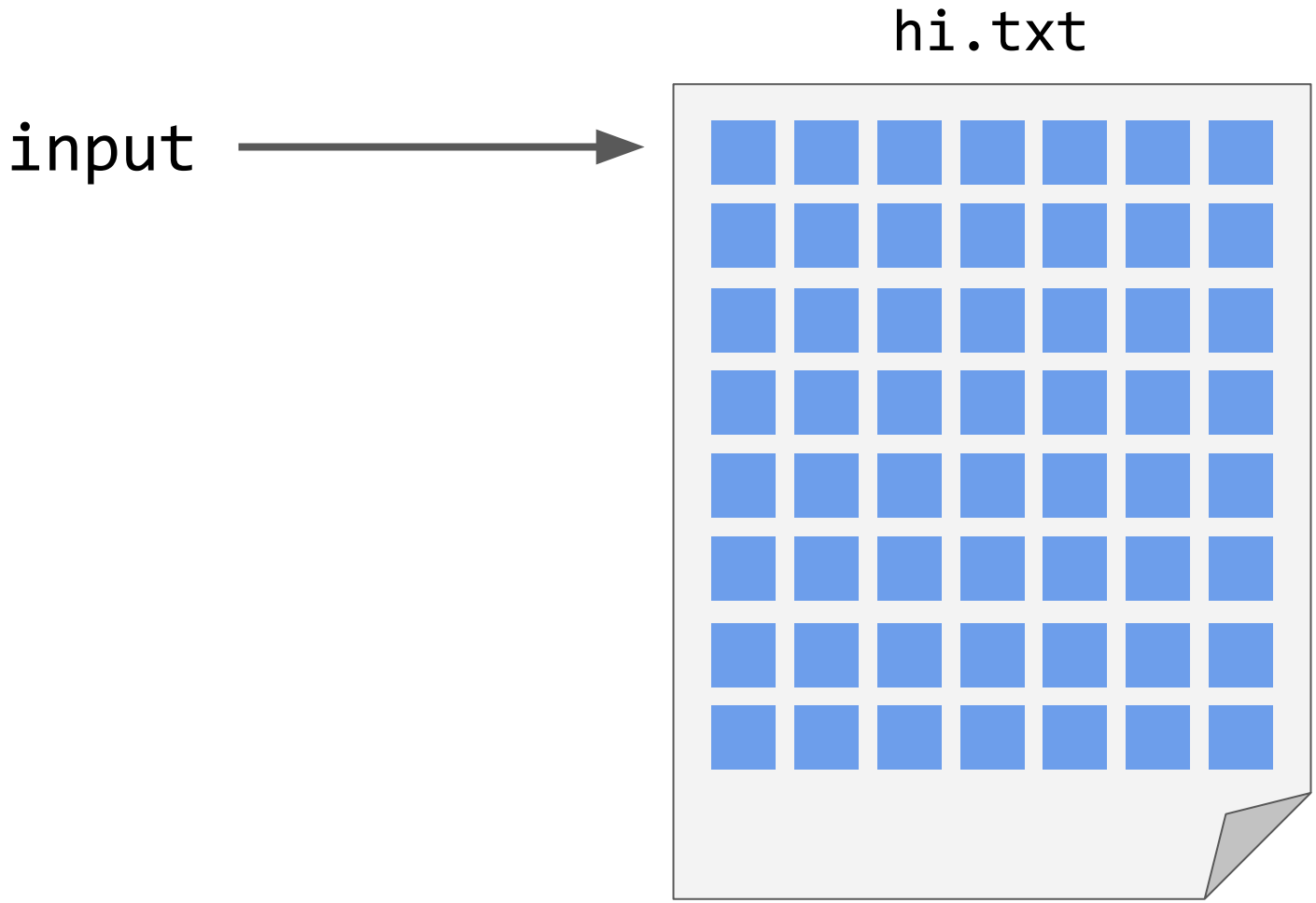
0x456



0x456


hi.txt
input
hi!
0x456
0x456

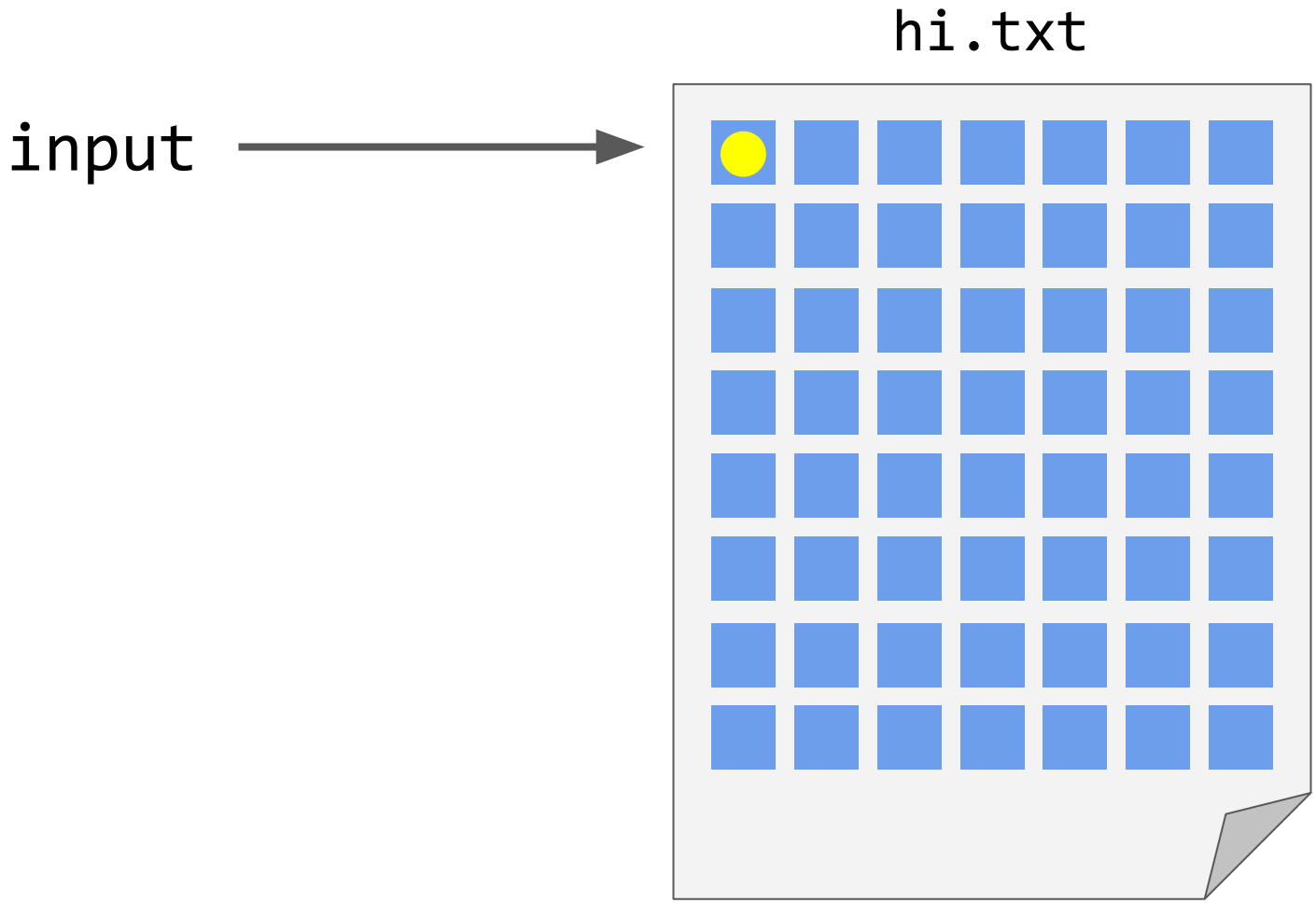

hi.txt
input


hi.txt
input

```
fread(buffer, 1, 4, input);
```

```
fread(buffer, 1, 4, input);
```

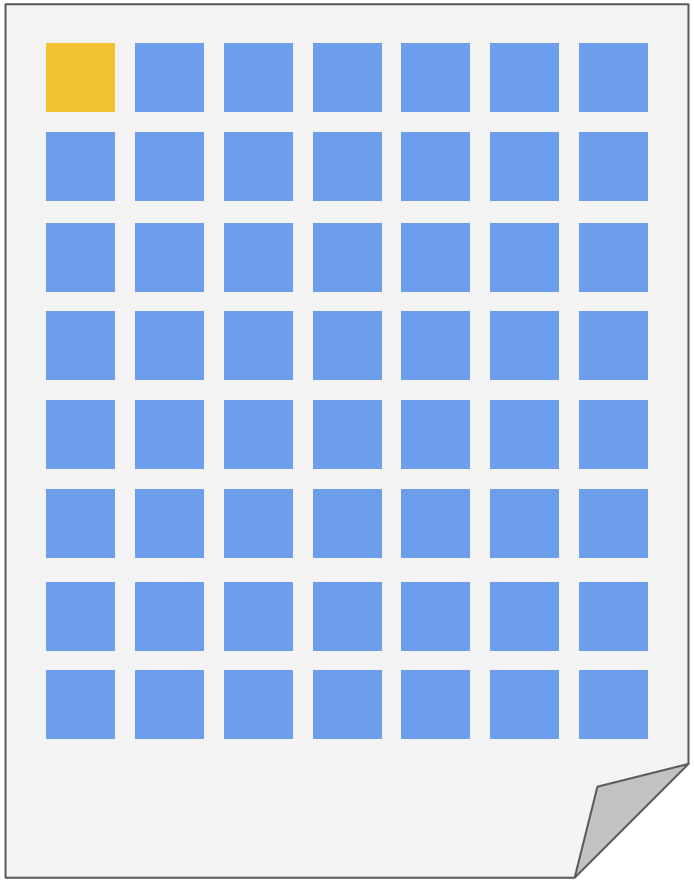
Location to read from


hi.txt
input

```
fread(buffer, 1, 4, input);
```

Size of blocks to read (in bytes)

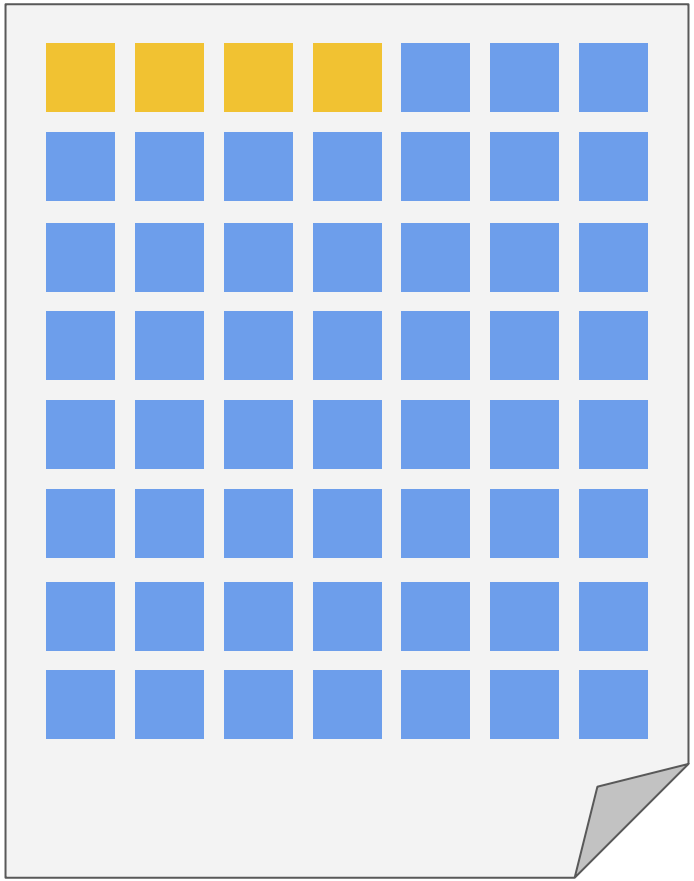
hi.txt



```
fread(buffer, 1, 4, input);
```

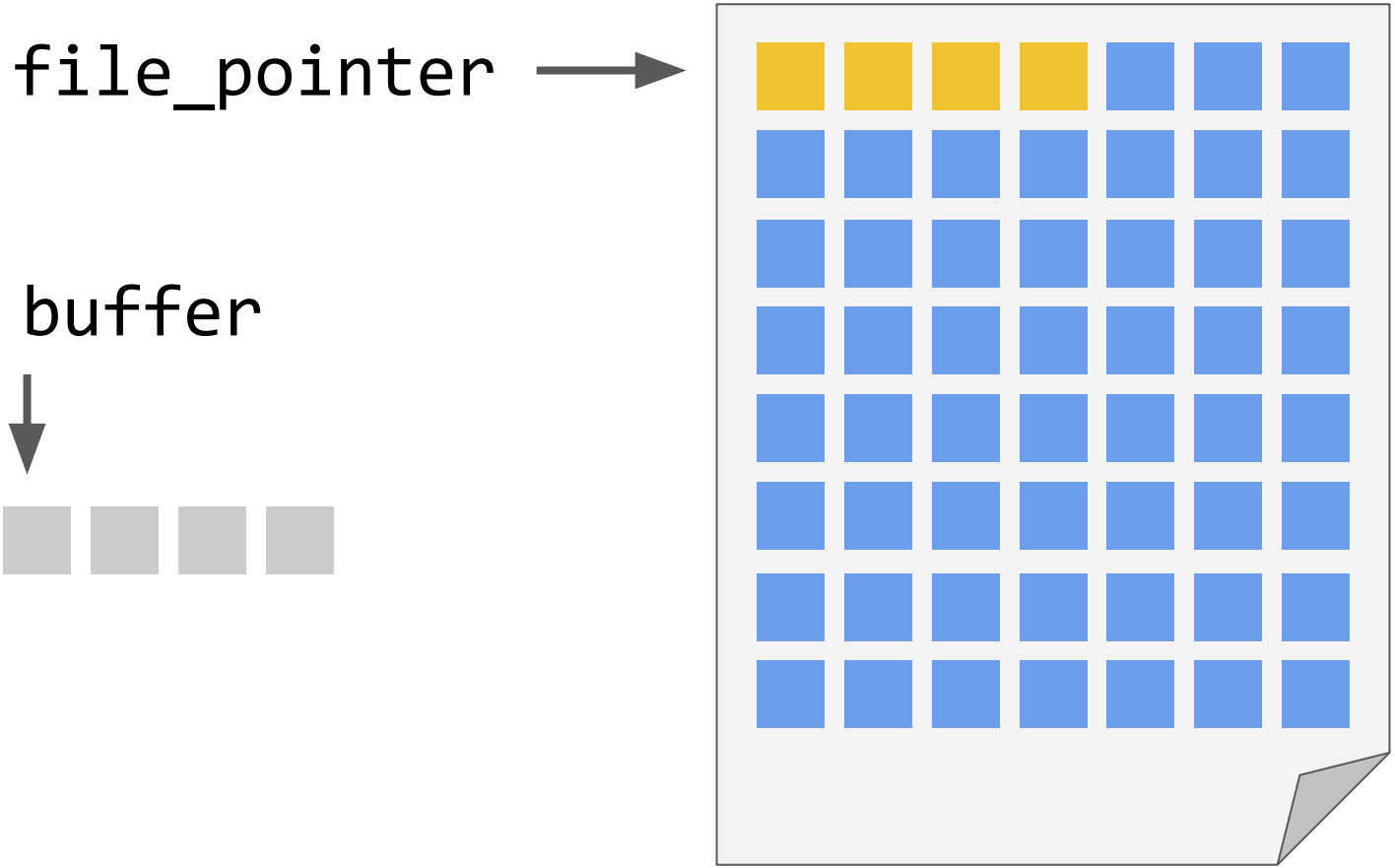
How many blocks to read

hi.txt

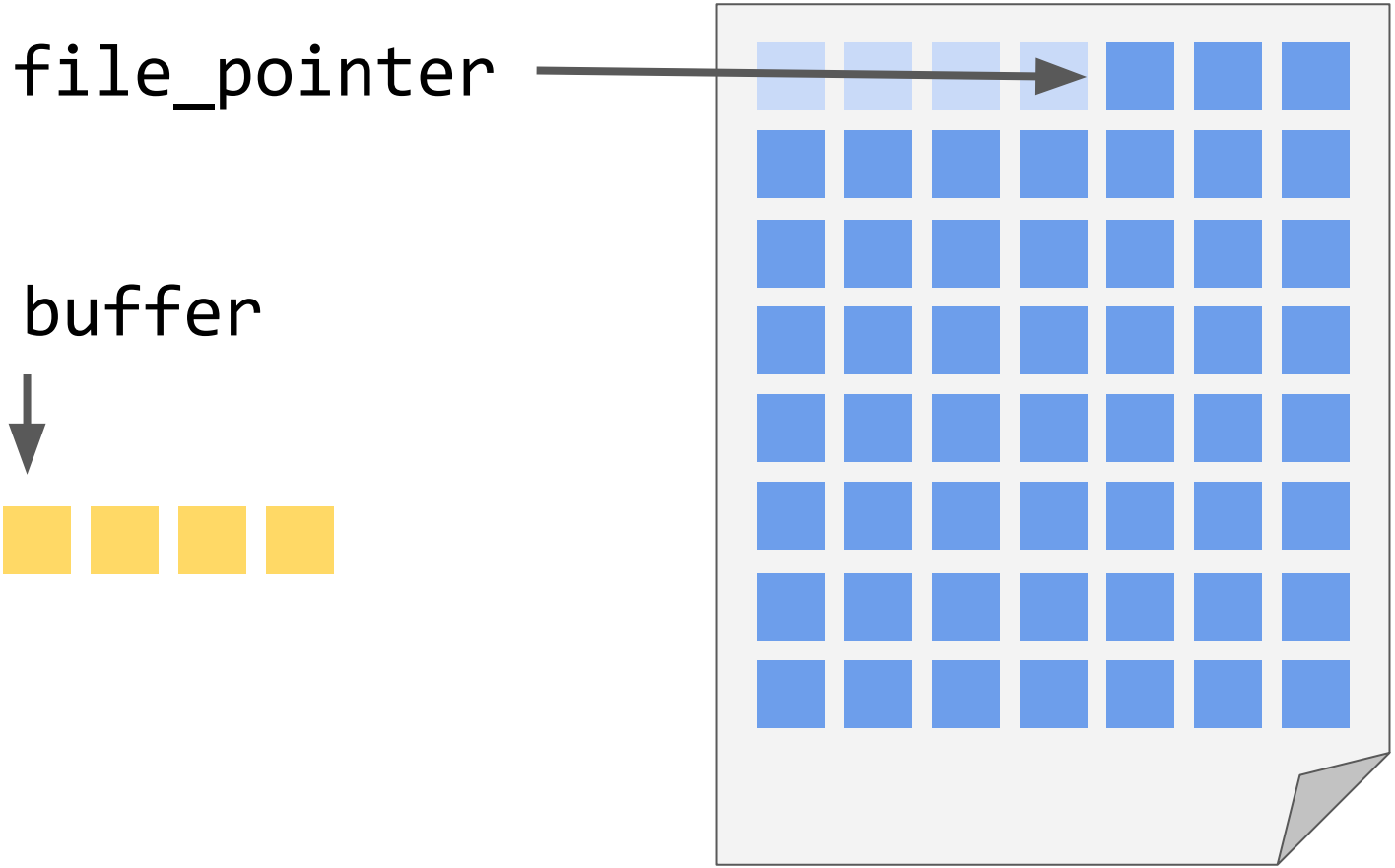


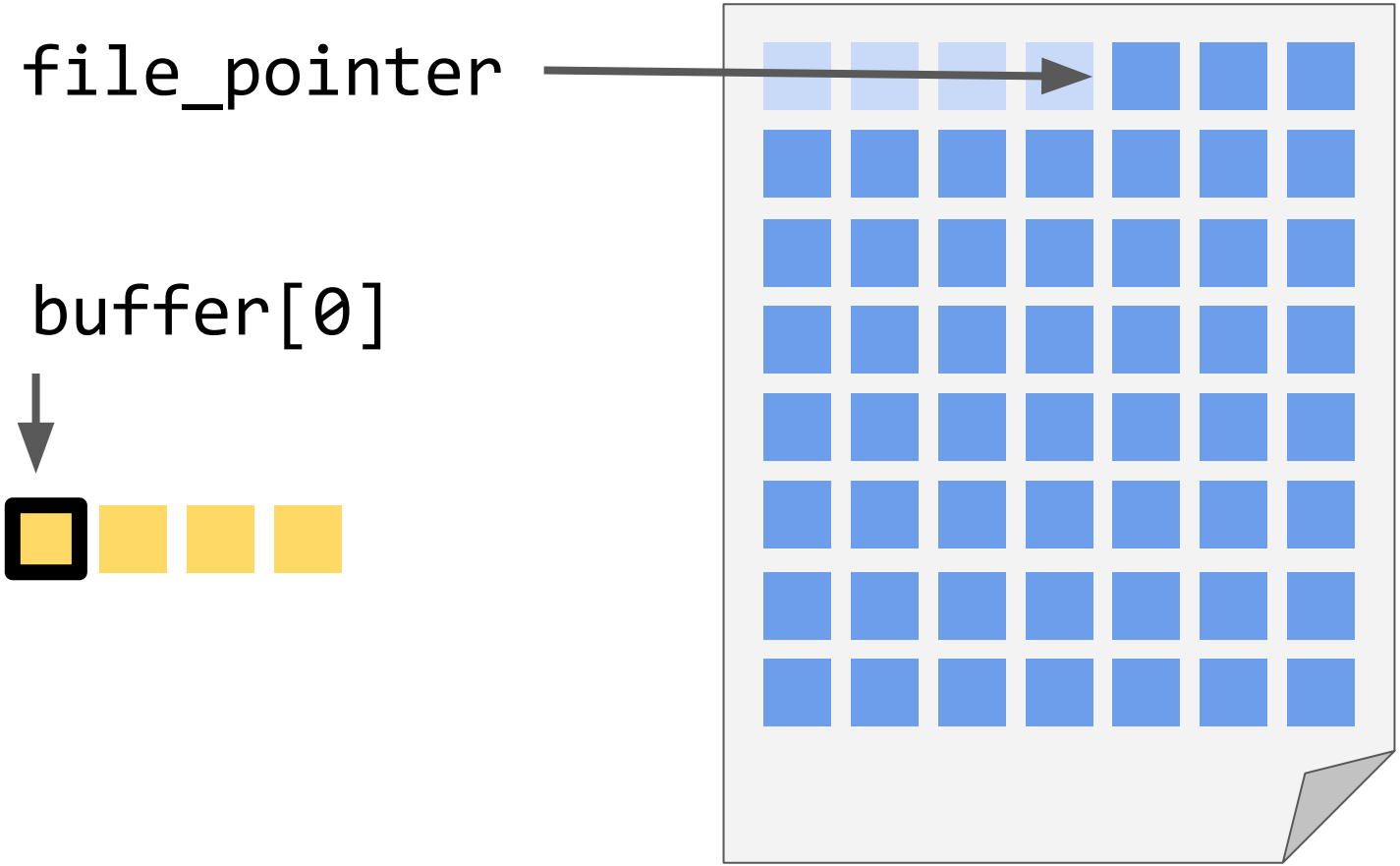
```
fread(buffer, 1, 4, input);
```

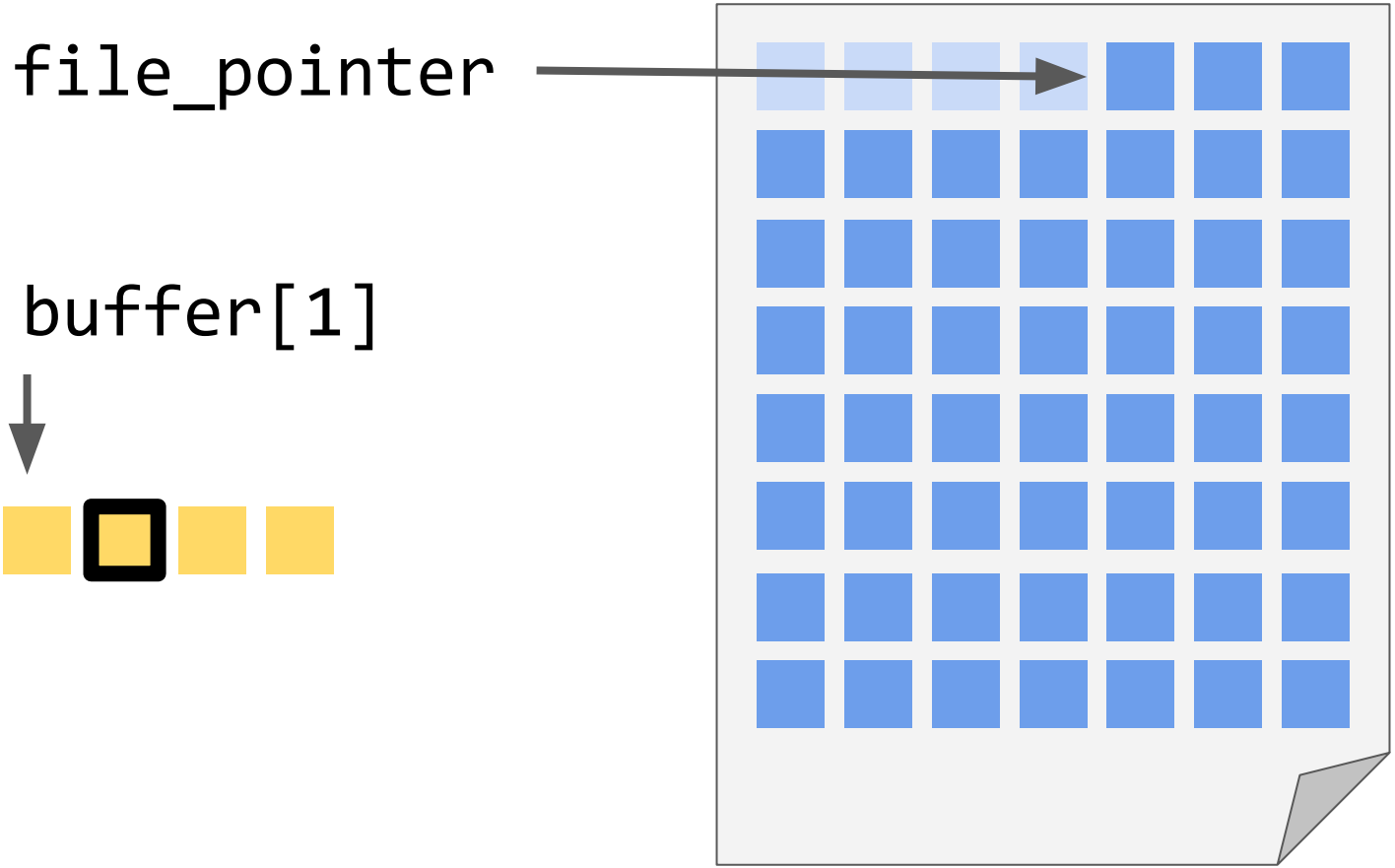
Location to store blocks


file_pointer
buffer

```
fread(buffer, 1, 4, input);
```

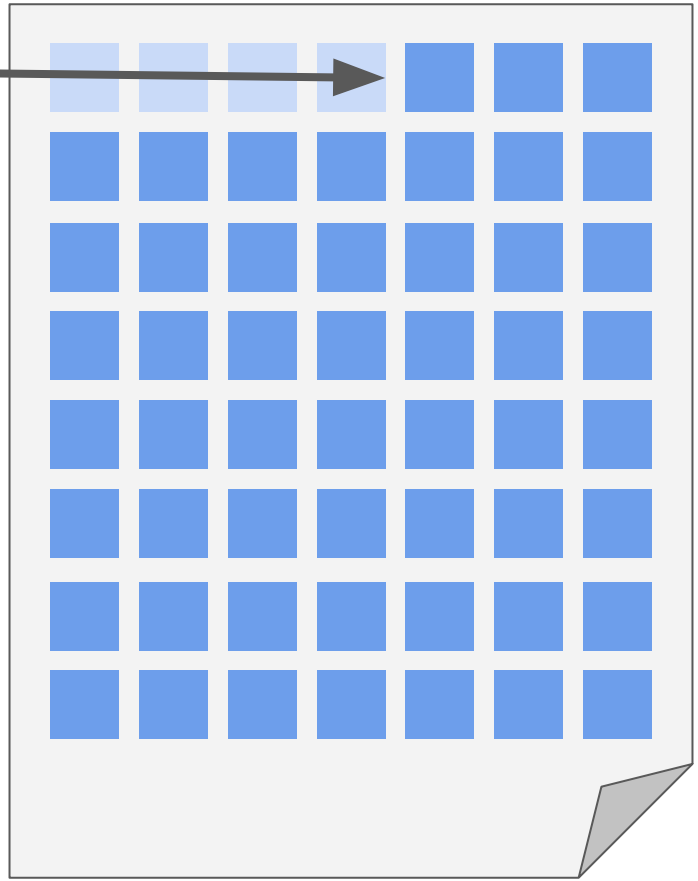

file_pointer
buffer


file_pointer
buffer[0]


file_pointer
buffer[1]

file_pointer

buffer[2]



```
fread(buffer, 1, 4, input);
```

```
fwrite(buffer, 1, 4, output);
```


output_file
buffer


output_file
buffer

File Reading Exercise

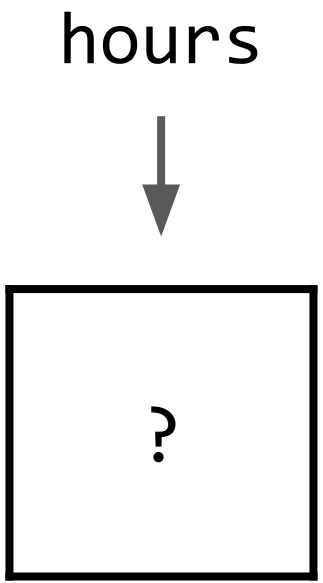
Create a program, **pdf.c**, that checks whether a file, passed in as a command-line argument, is a PDF. All PDFs will begin with a four byte sequence:

```
0x25 0x50 0x44 0x46
```

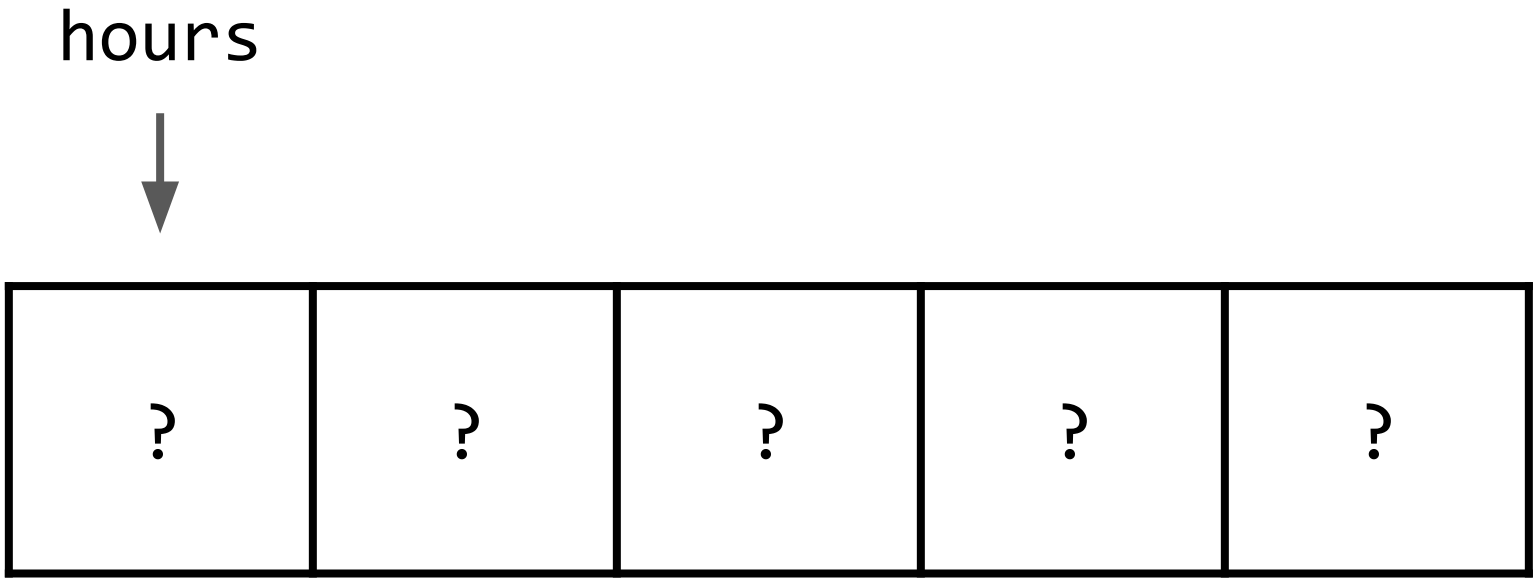
Use the **.pdf** and **.jpg** files in the section resources page to check your work.

Dynamic Memory

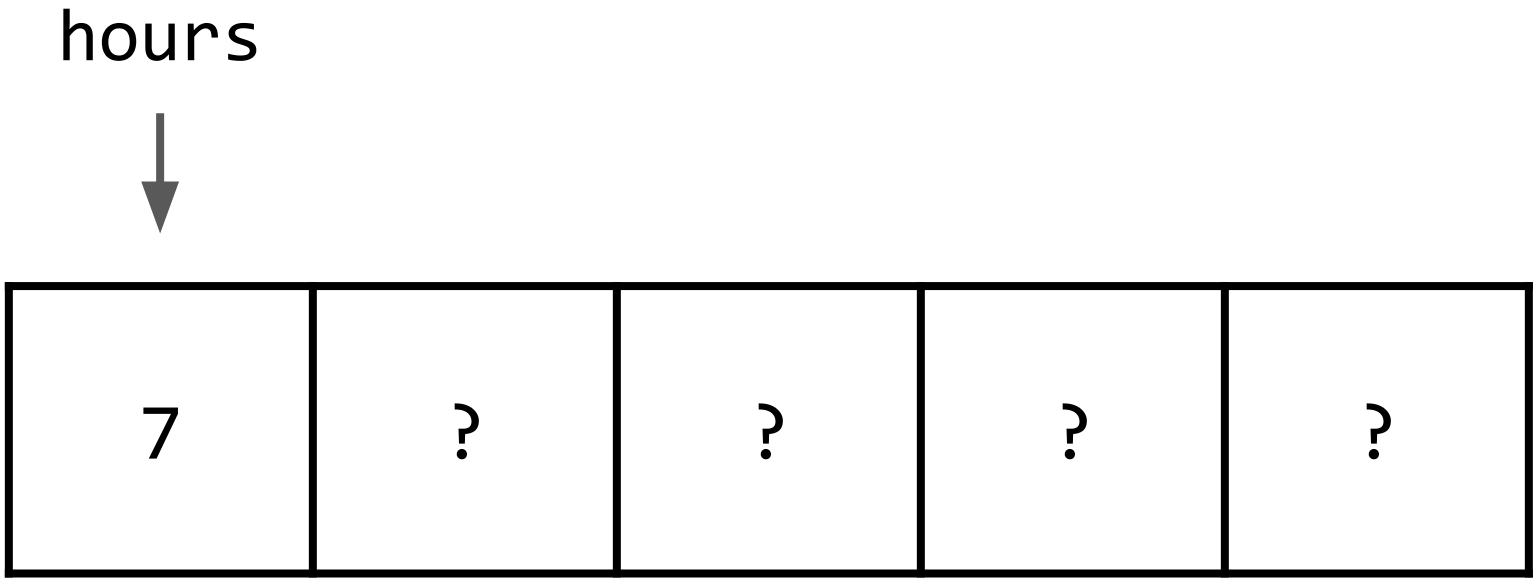
```
int *hours = malloc(sizeof(int));
```



```
int *hours = malloc(sizeof(int) * 5);
```

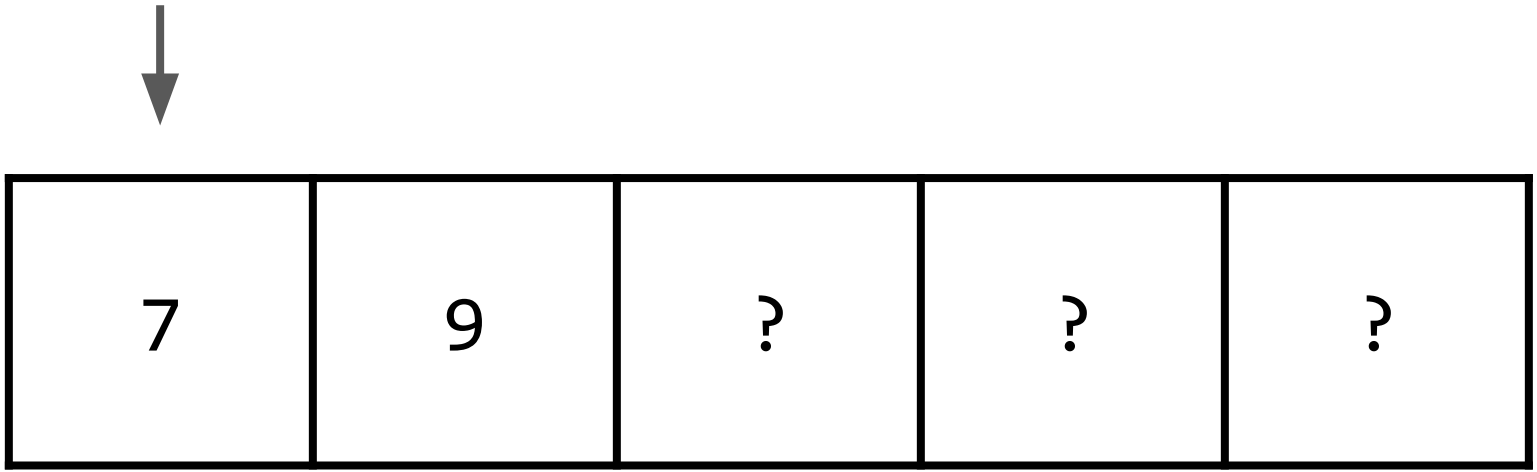


```
*hours = 7;
```

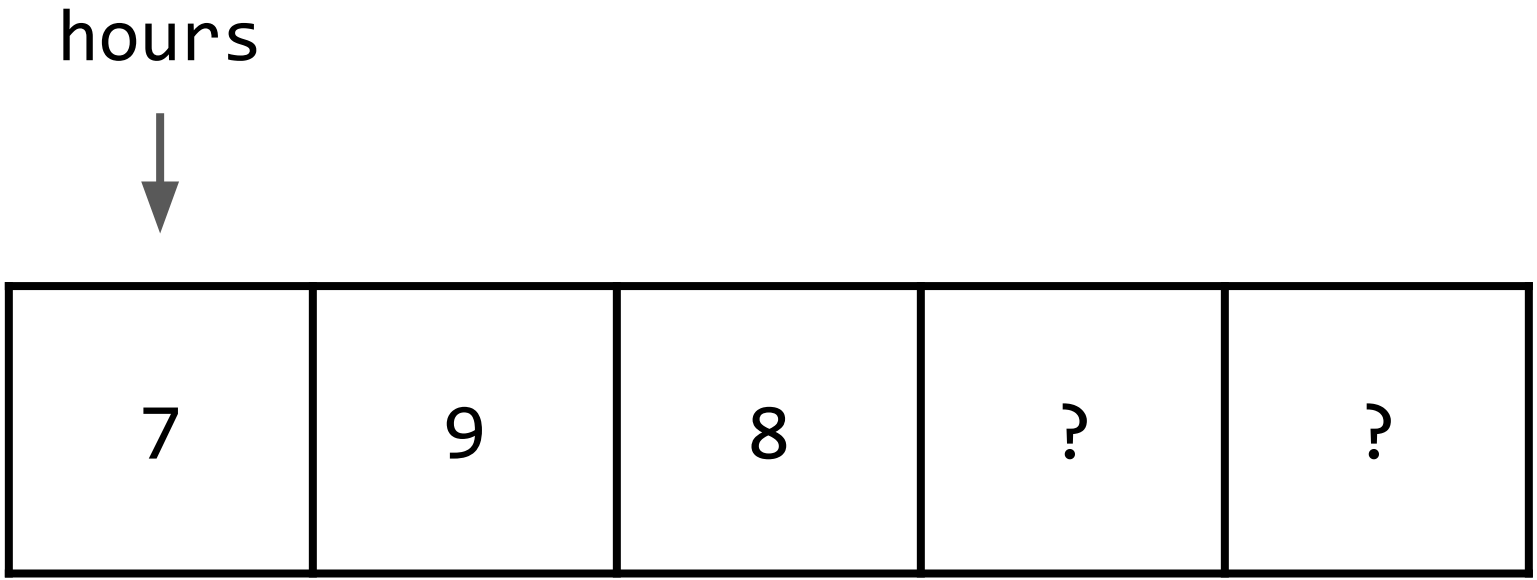


```
*hours = 7;
*(hours + 1) = 9;
```

hours

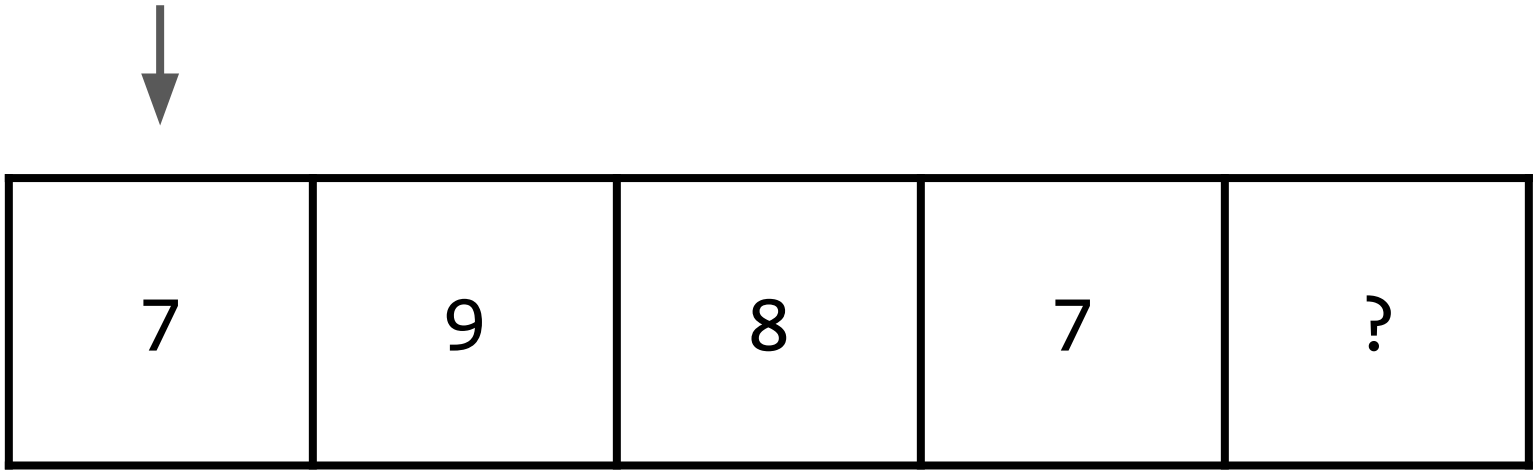


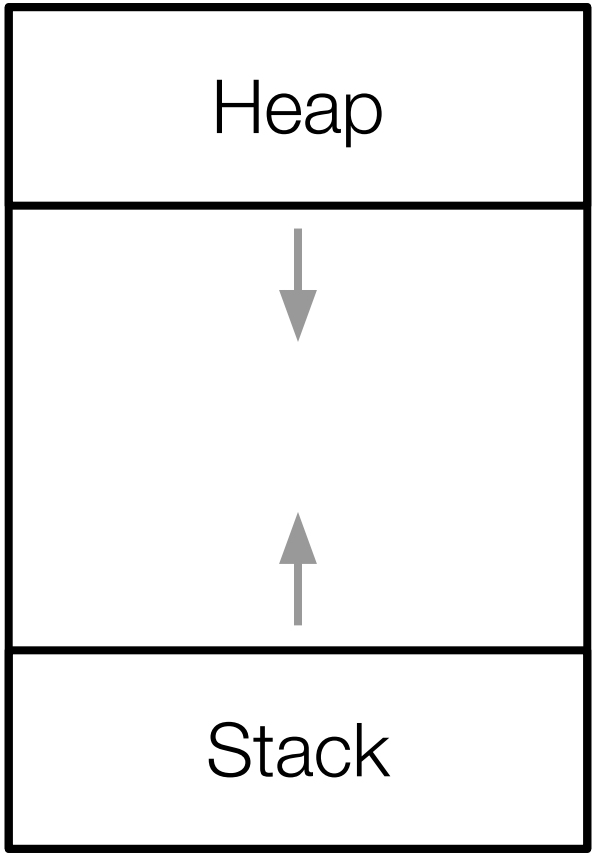
```
hours[2] = 8;
```



```
hours[2] = 8;
hours[3] = 7;
```

hours




Heap
Stack

Common memory errors

Failing to **free** every block of memory which we've **malloc**'d.

Failing to **fclose** every file we've **fopen**ed.

Using more memory than we've allocated.

Debugging Memory Exercise

Debug a program, **create.c**, that creates the file given as input at the command-line. For example,

```
./create test.c
```

will create a file, **test.c**. But our code has three memory errors! Can you find and fix them? Try running the below to check:

```
valgrind ./create test.c
```

Volume

Tutorials

Office Hours

This was CS50