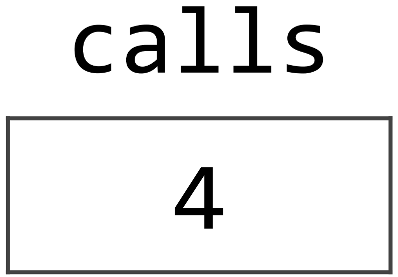
This is CS50

- 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

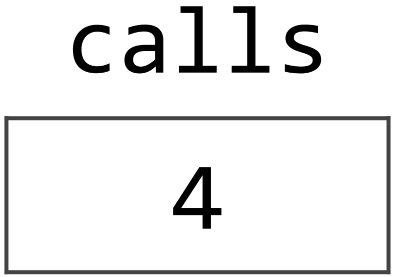
Variables

```
int calls = 4;
```



Variables

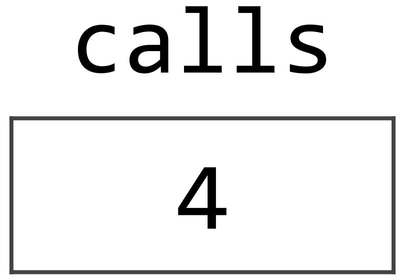
$$\texttt{int}\ \underbrace{\texttt{calls}}_{\text{name}}\ \texttt{= 4;}$$



Variables

```
int calls = 4;
```

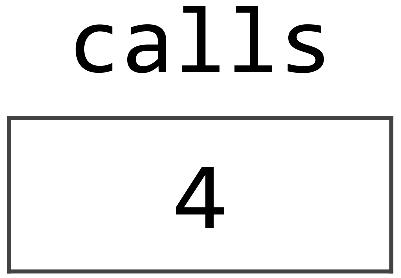
type



Variables

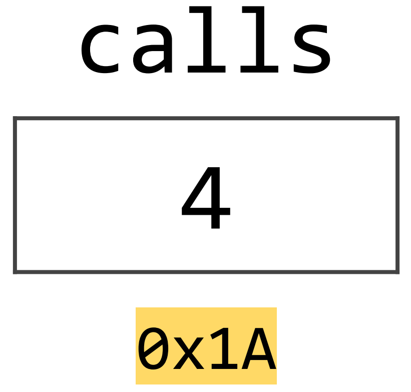
```
int calls = 4;
```

value



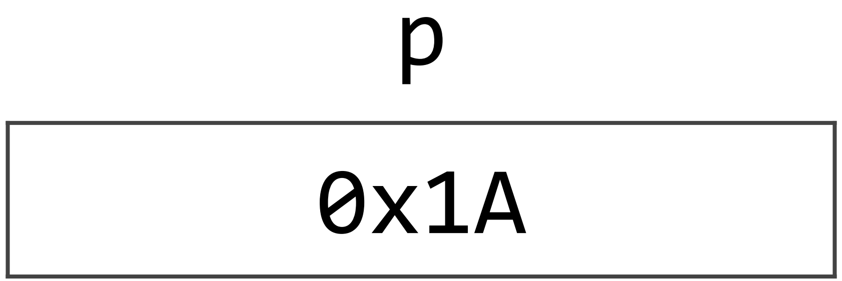
Variables

```
int calls = 4;
```



Pointers

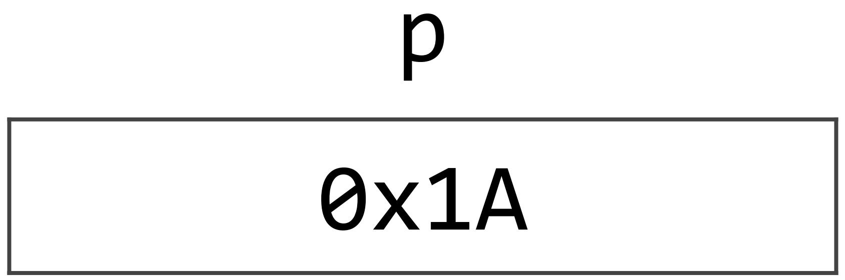
```
int *p = 0x1A;
```



Pointers

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

name



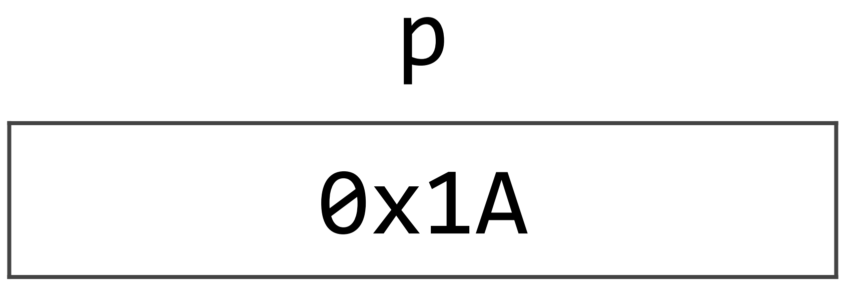
Pointers

$$\underbrace{\texttt{int *}}_{\text{type}}\texttt{p = 0x1A;}$$

$$\overset{\texttt{p}}{\boxed{\texttt{0x1A}}}$$

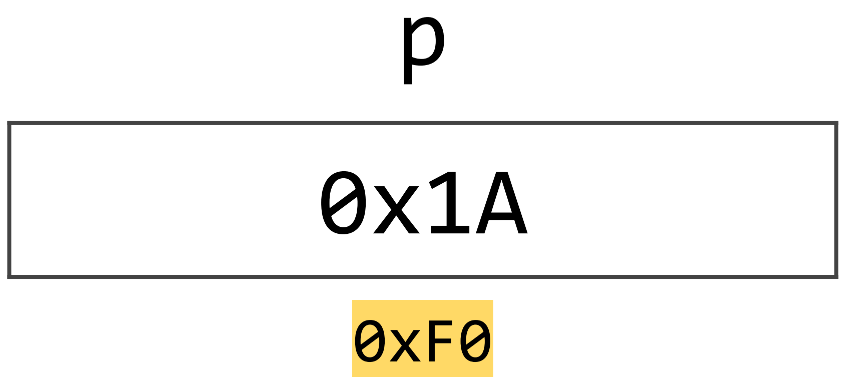
Pointers

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



Pointers

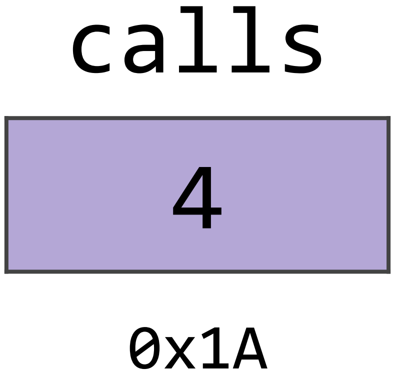
```
int *p = 0x1A;
```



Pointer Syntax

```
calls;
```

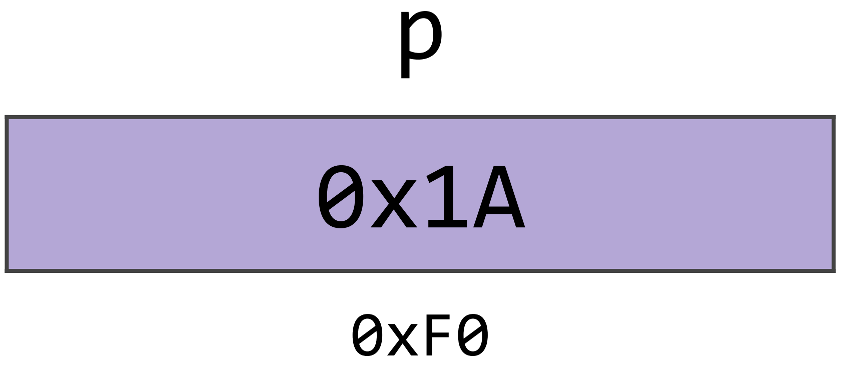
"value of"



Pointer Syntax

```
p;
```

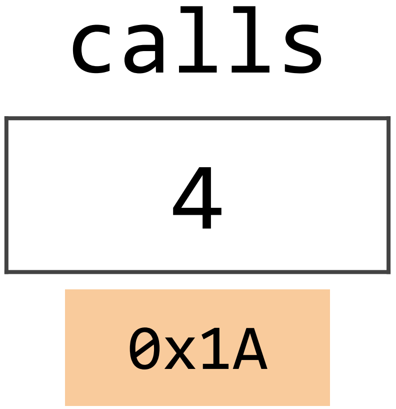
"value of"



Pointer Syntax

```
&calls;
```

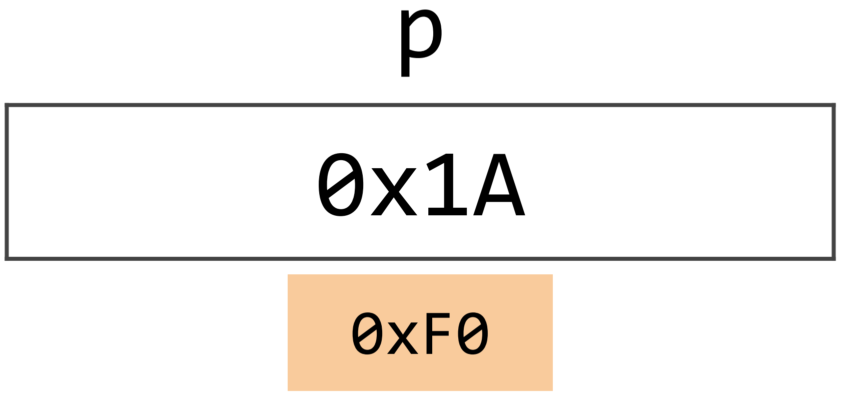
"address of"



Pointer Syntax

```
&p;
```

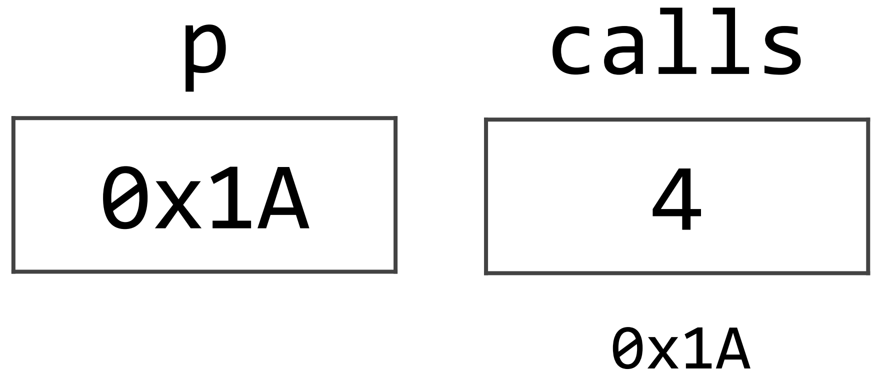
"address of"



Pointer Syntax

```
*p;
```

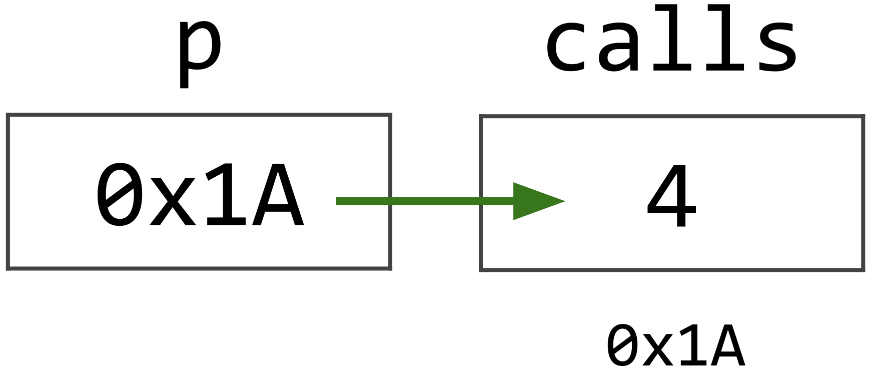
"**go to** the value at address stored in p"



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.

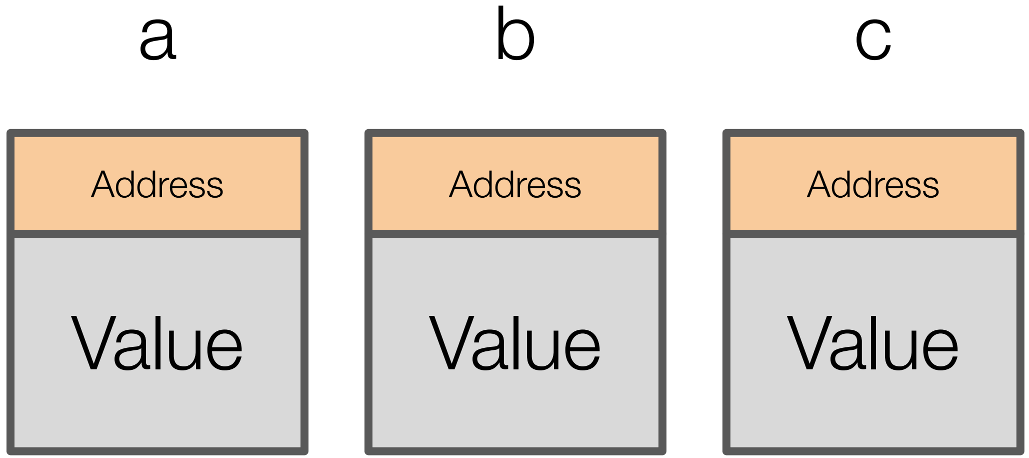
`&x` takes **`x`** and gets its address.

Pointer Prediction Exercise

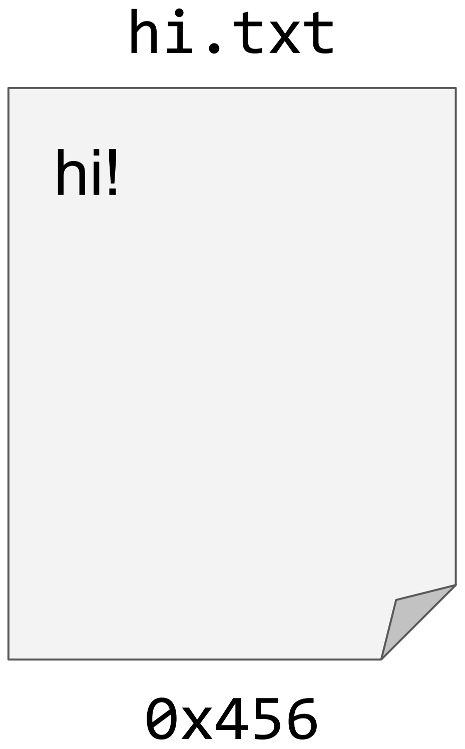
Visualize the code on the left, step by step. How do the values of the variables evolve? It's okay to use made-up addresses.

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

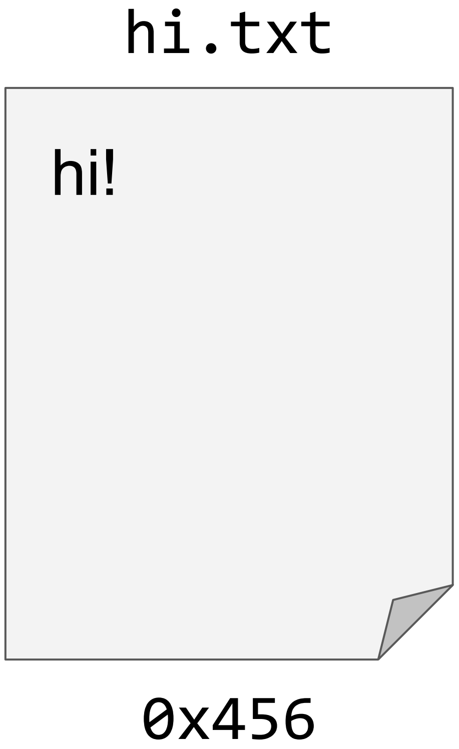
```c
int a = 28;
int b = 50;
int *c = &a;

*c = 14;
c = &b;
*c = 25;
```



File I/O


hi.txt
hi!
0x456

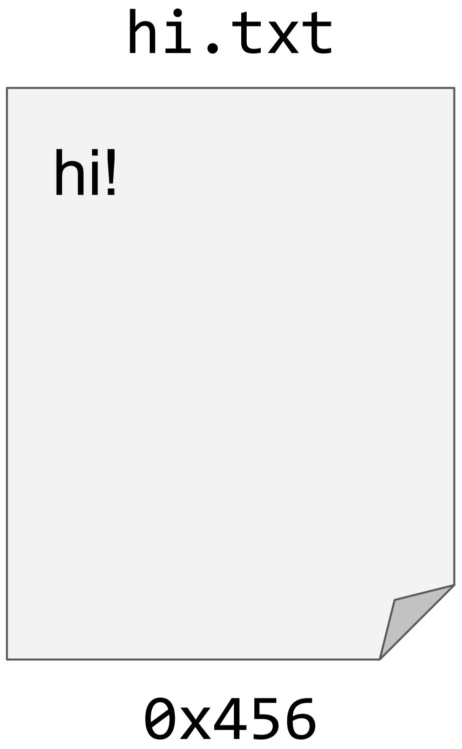
```
FILE *input = fopen("hi.txt", "r");
```



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

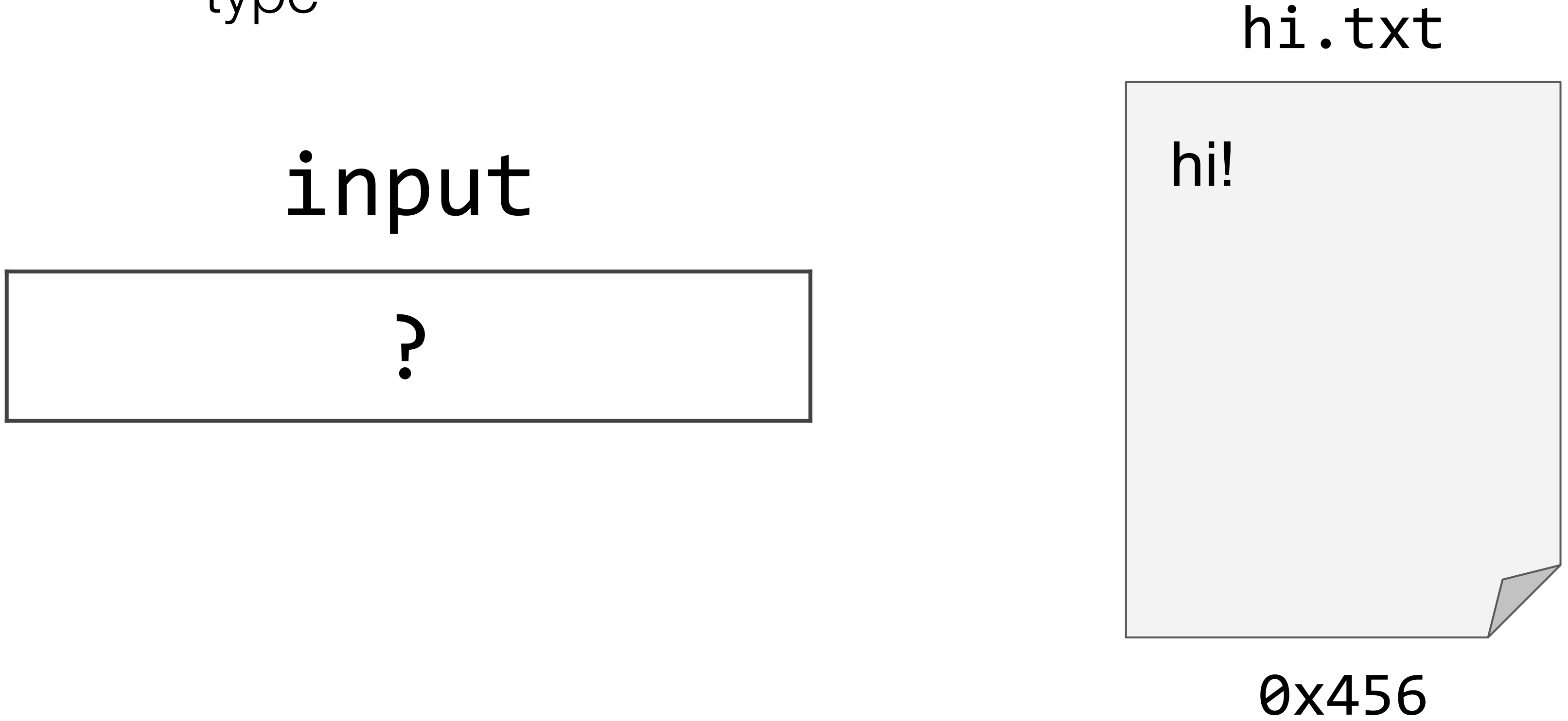
name

input



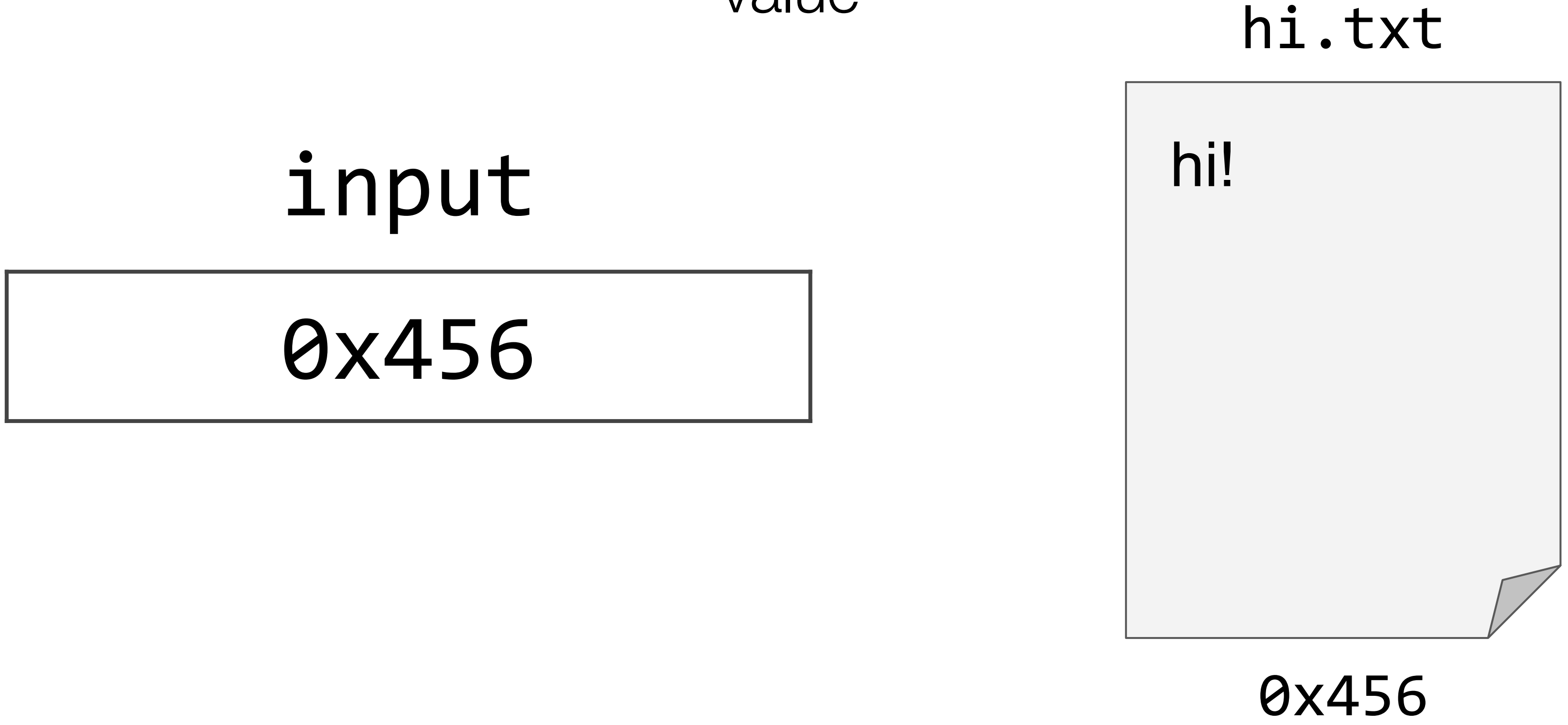
```
FILE *input = fopen("hi.txt", "r");
```

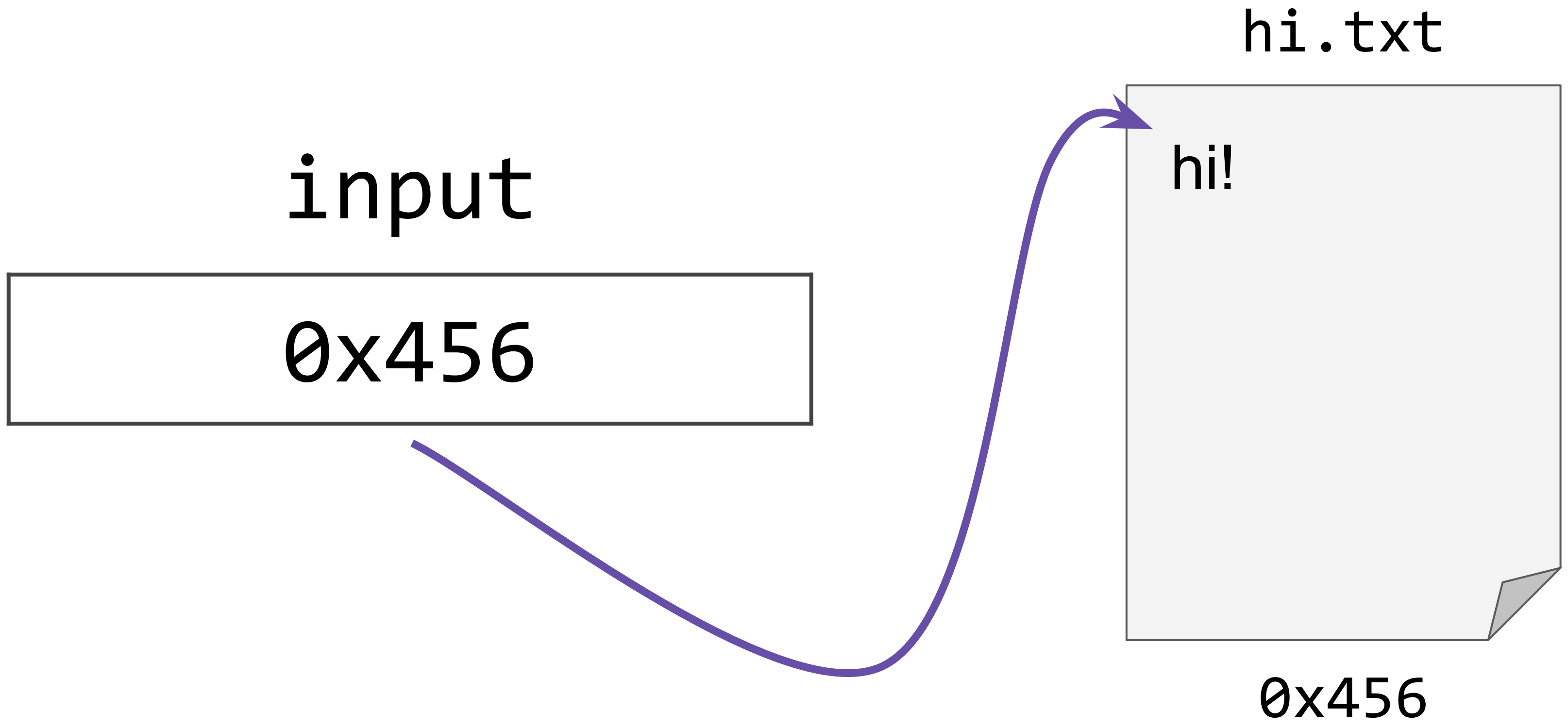
type

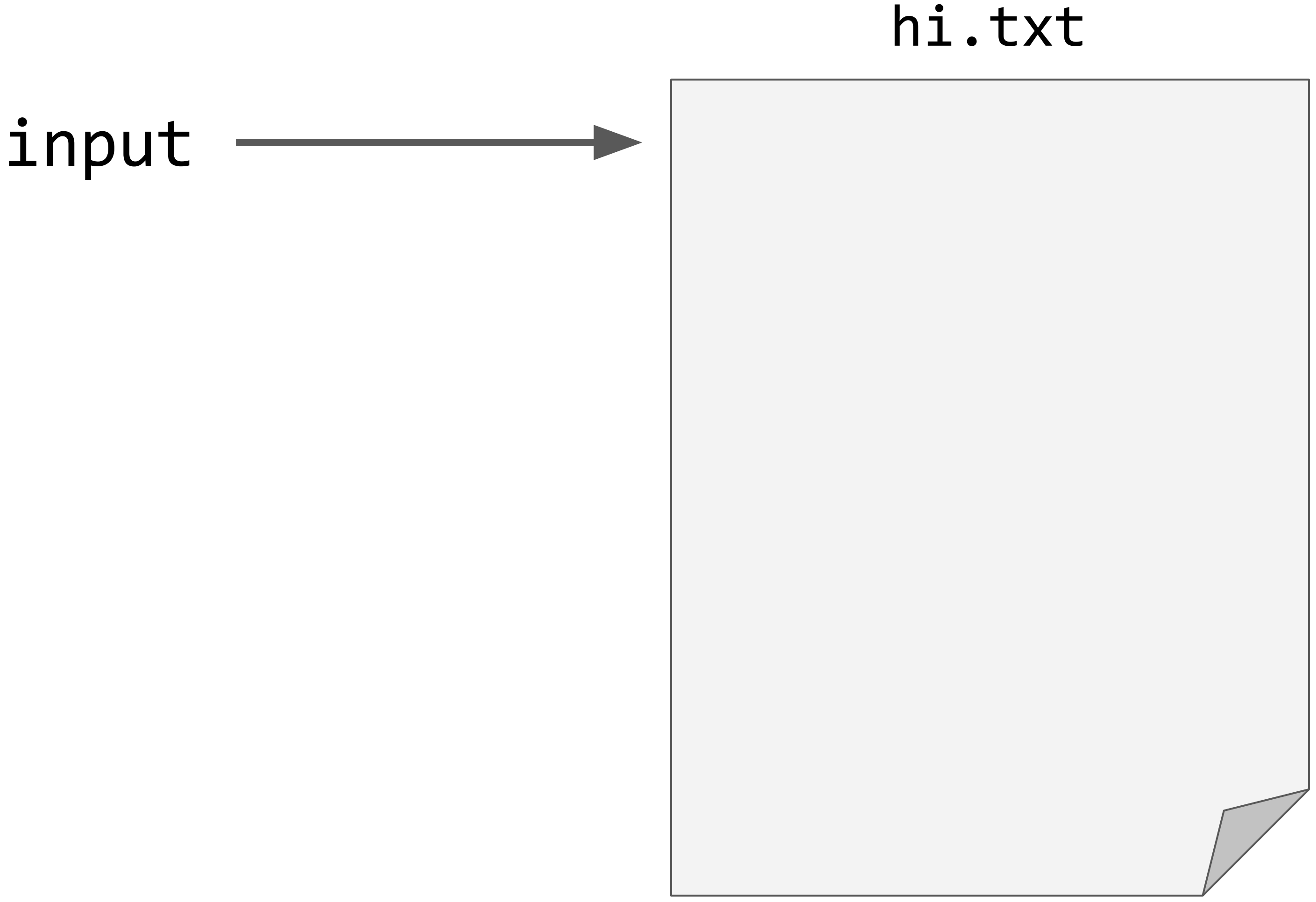


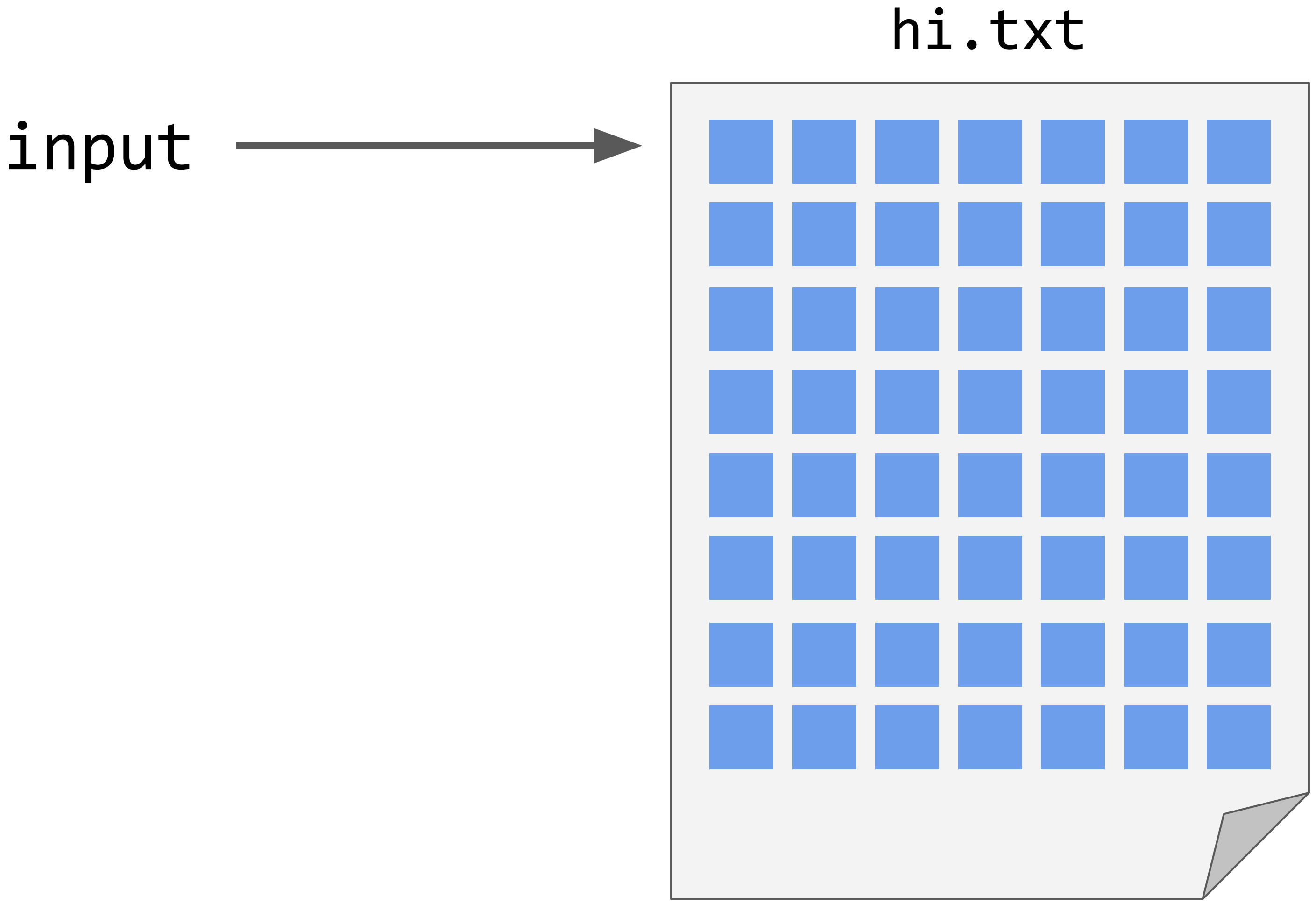
```
FILE *input = fopen("hi.txt", "r");
```

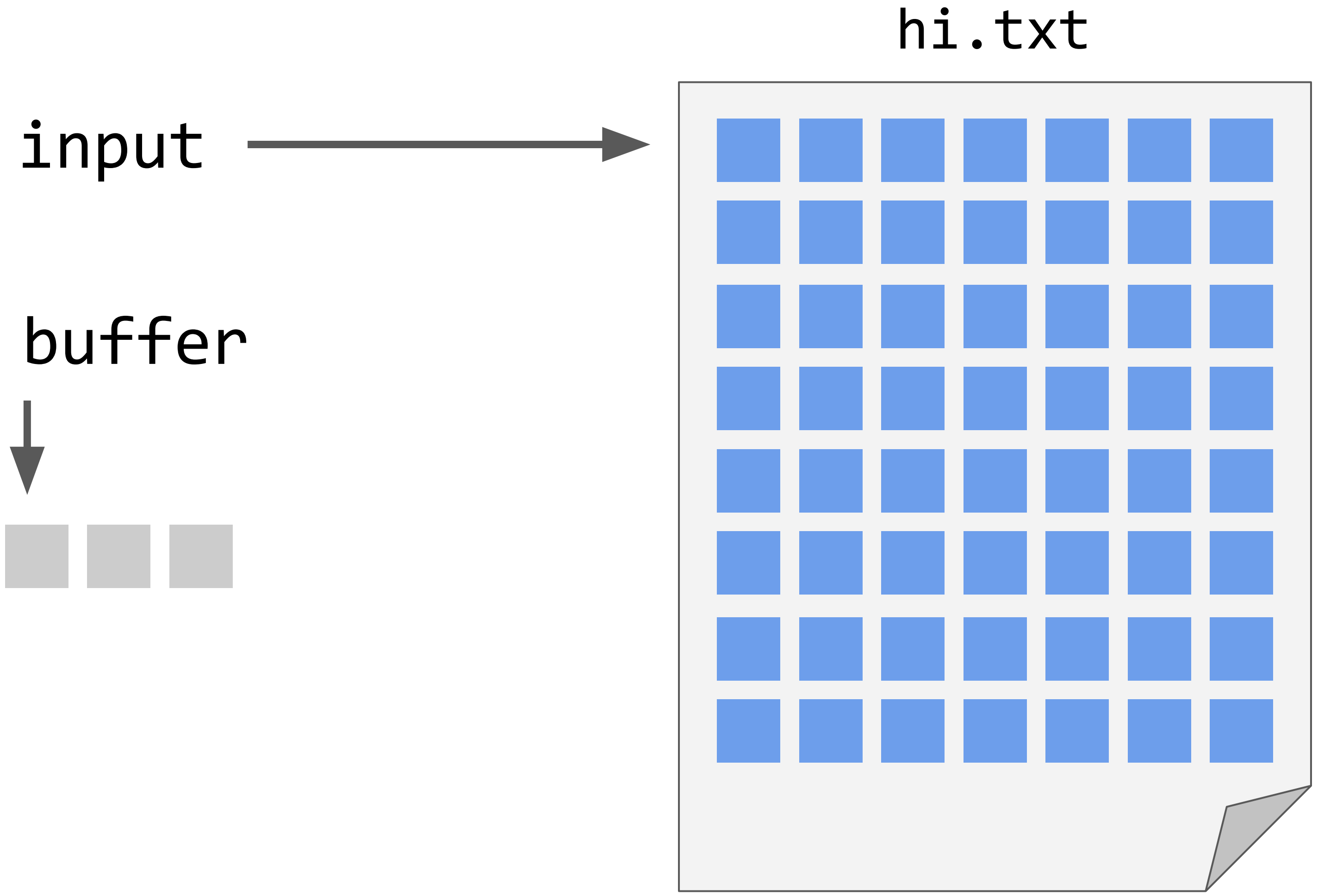
value




input
0x456
hi.txt
hi!
0x456


hi.txt
input


input
hi.txt


hi.txt
input
buffer

```
fread(buffer, 1, 3, input);
fread(buffer, 1, 3, input);
```

$\uparrow$

Location to read from

```
fread(buffer, 1, 3, input);
```

Size of blocks to read (in bytes)

```
fread(buffer, 1, 3, input);
```

How many blocks to read

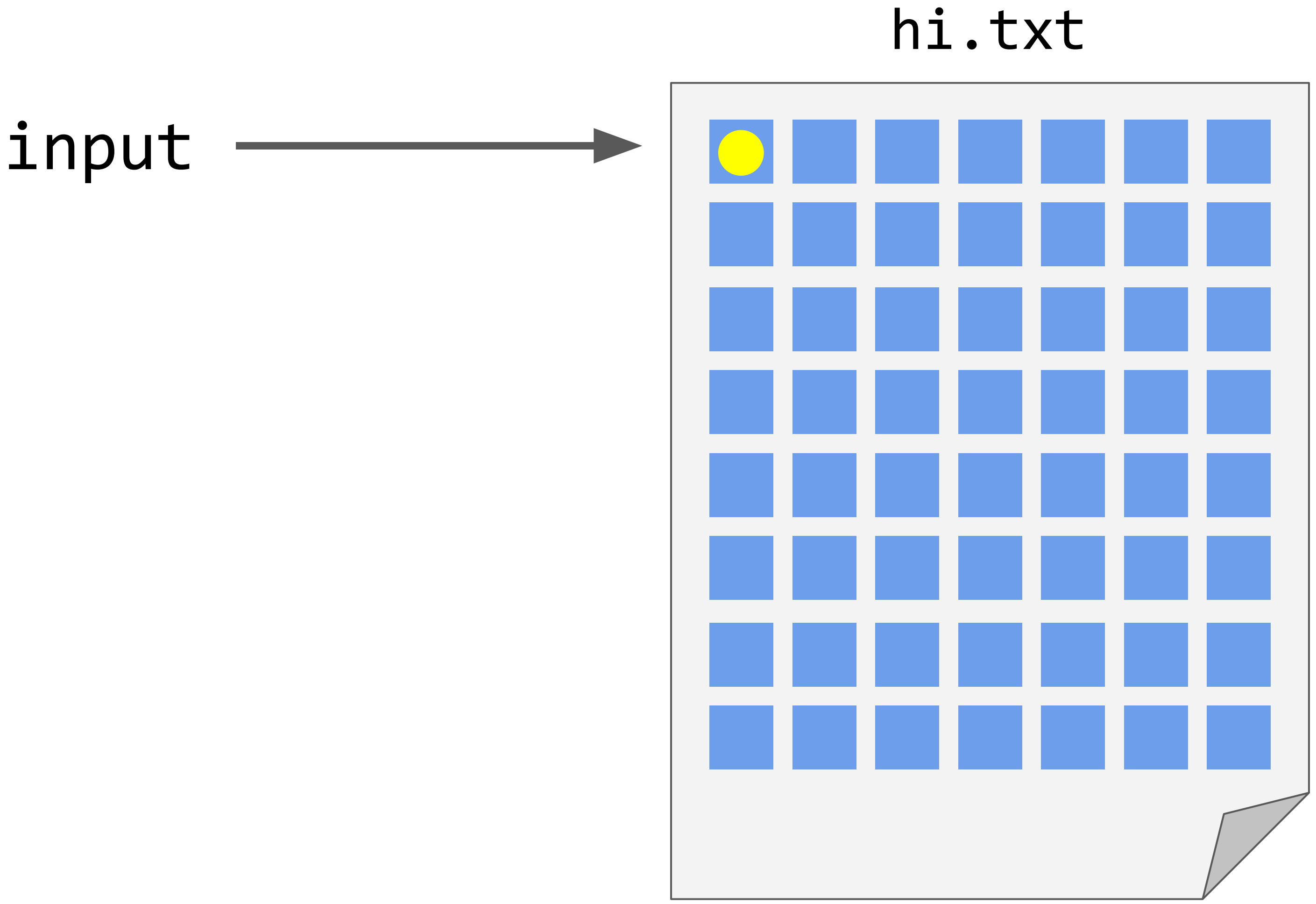
```
fread(buffer, 1, 3, input);
```

$\uparrow$

Location to store blocks

```
fread(buffer, 1, 3, input);
```

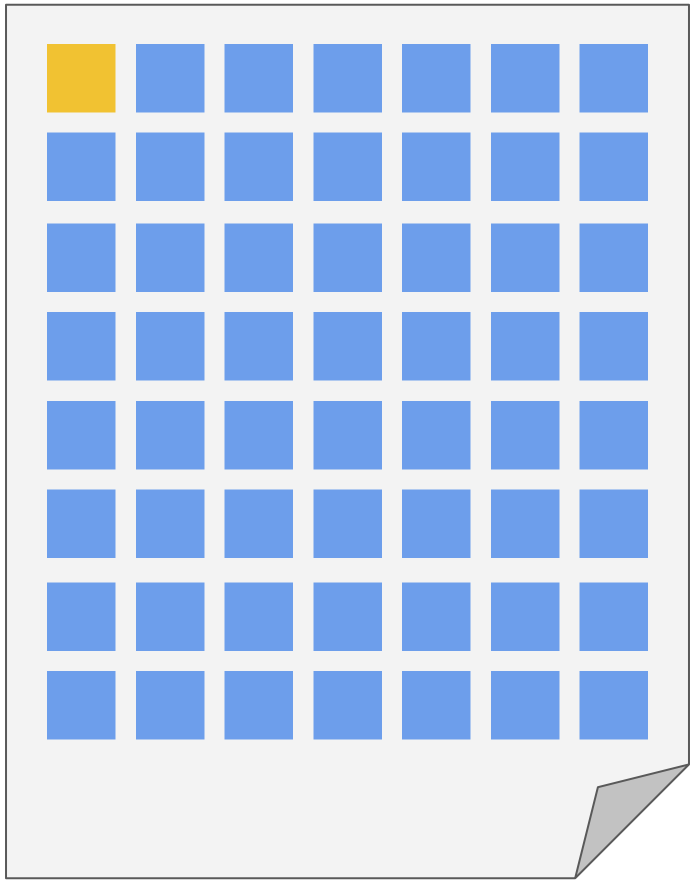
Location to read from


hi.txt
input

```
fread(buffer, 1, 3, input);
```

Size of blocks to read (in bytes)

hi.txt

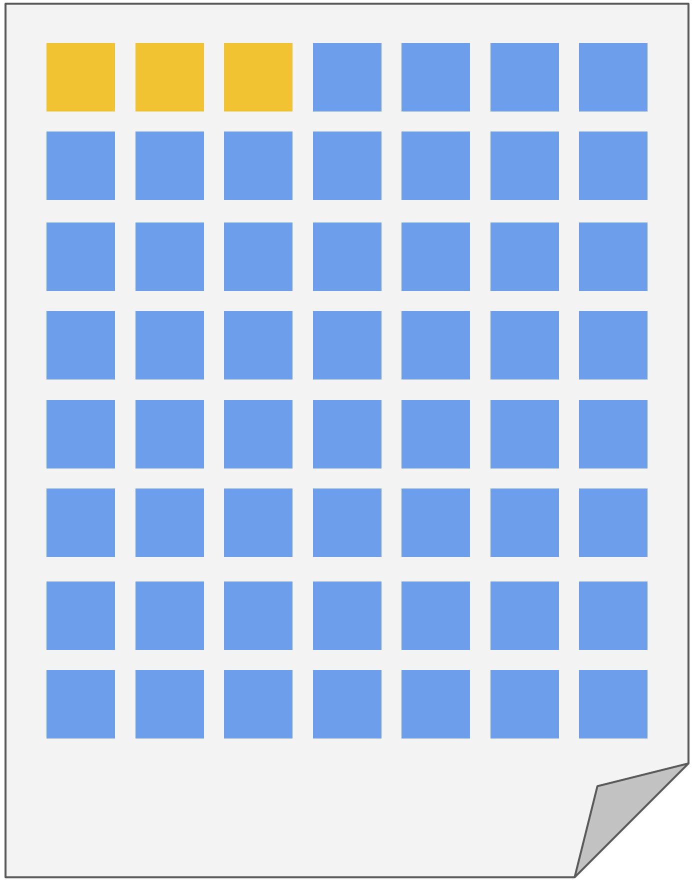


```
fread(buffer, 1, 3, input);
```

$\uparrow$

How many blocks to read

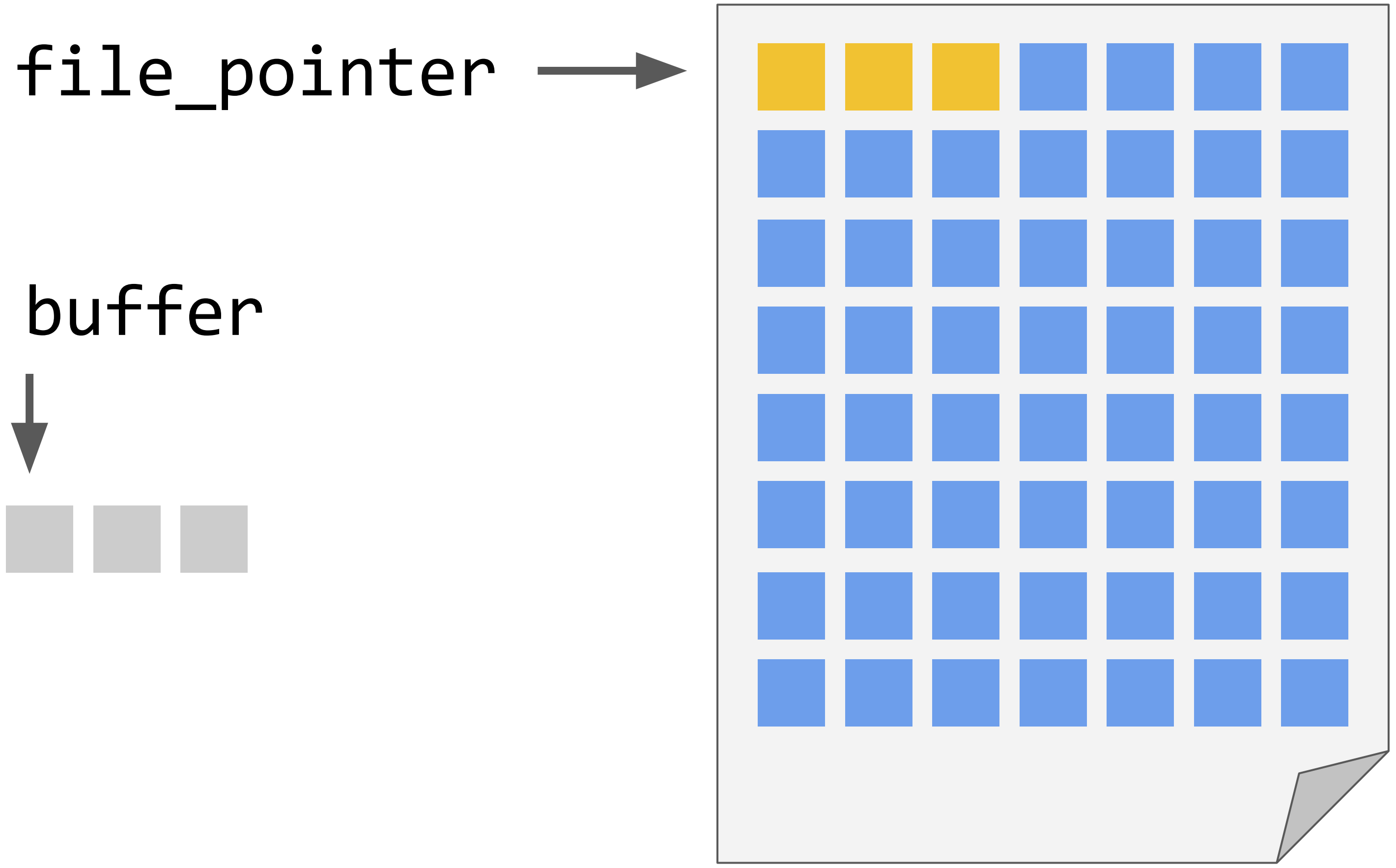
hi.txt

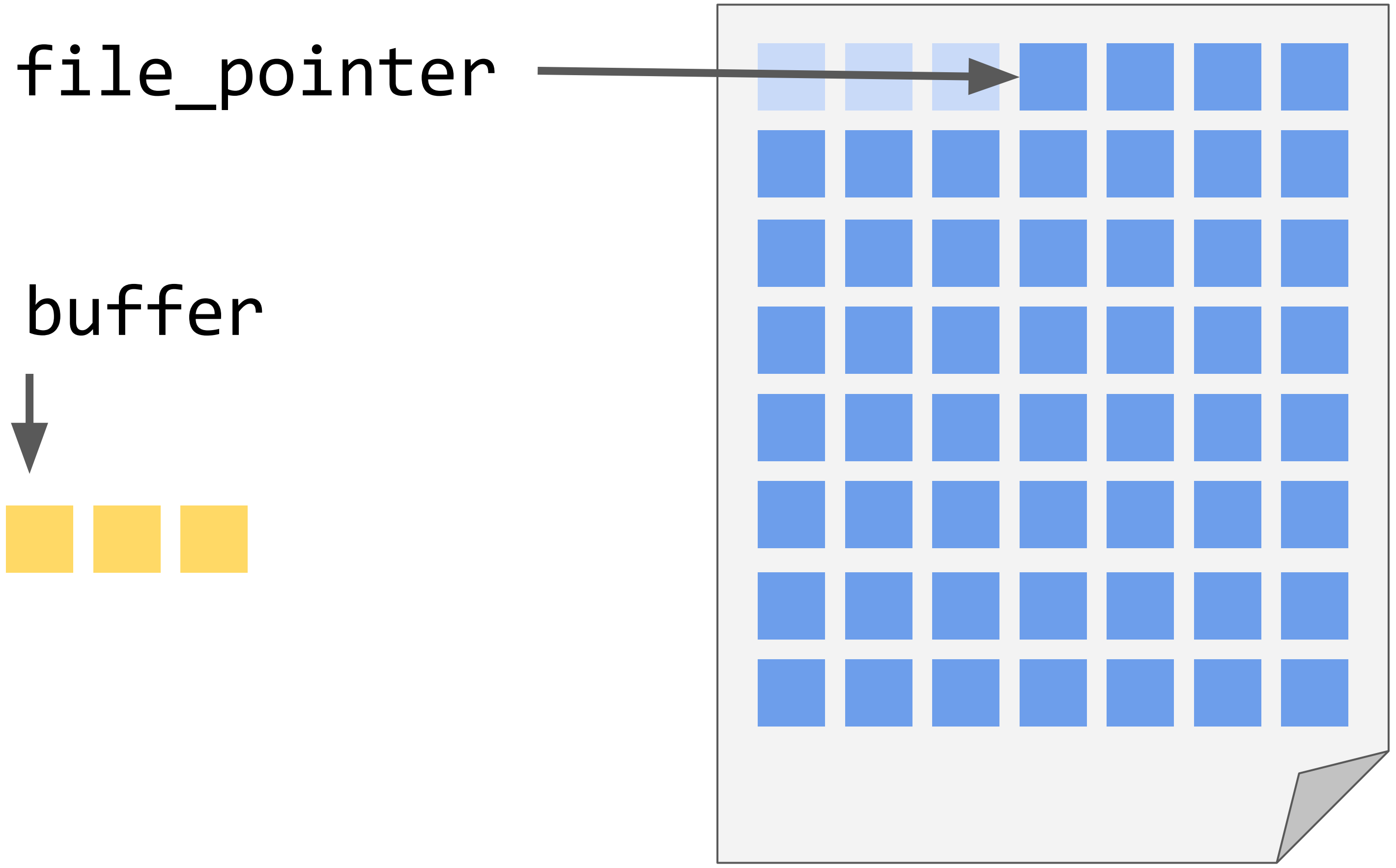


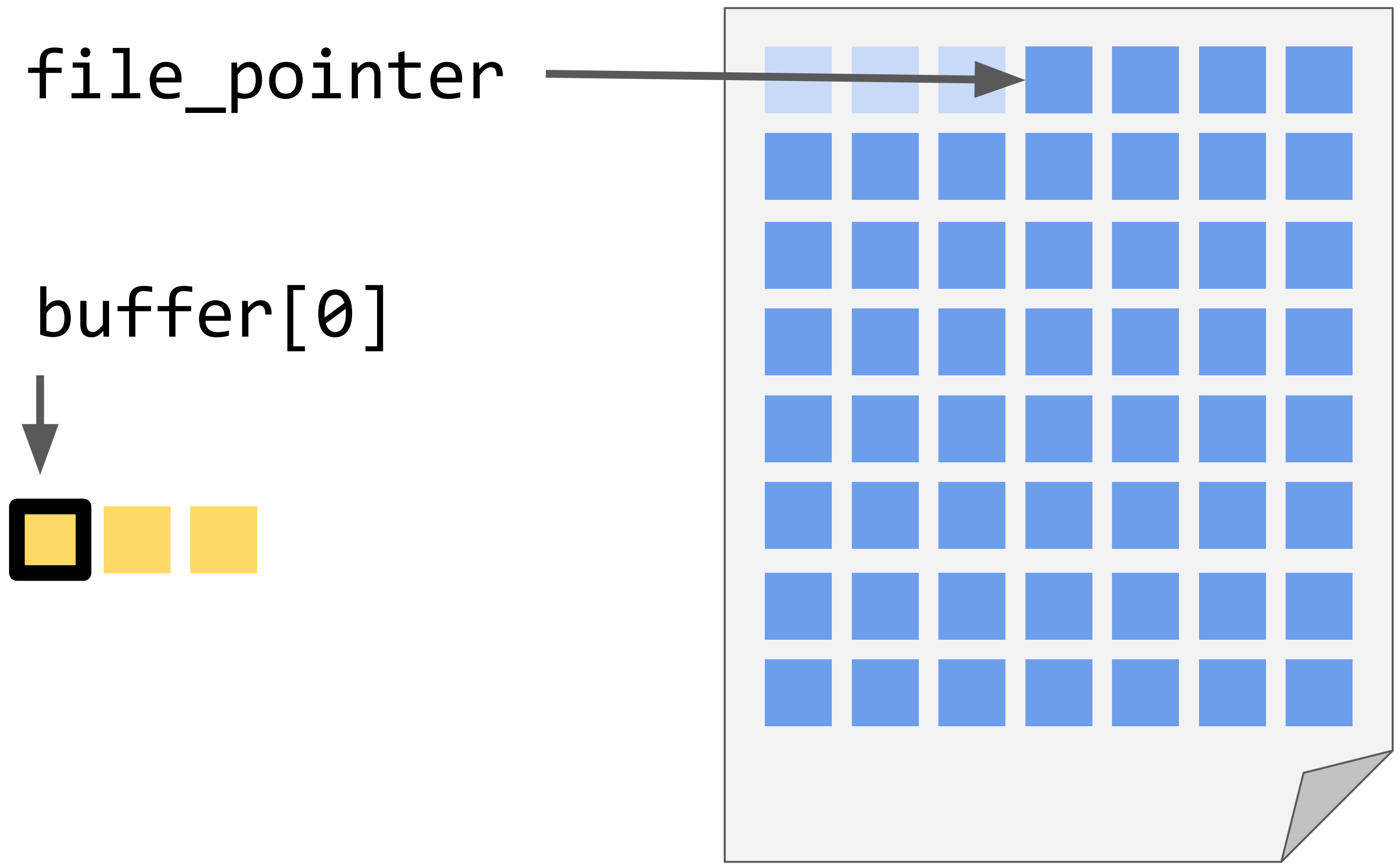
```
fread(buffer, 1, 3, input);
```

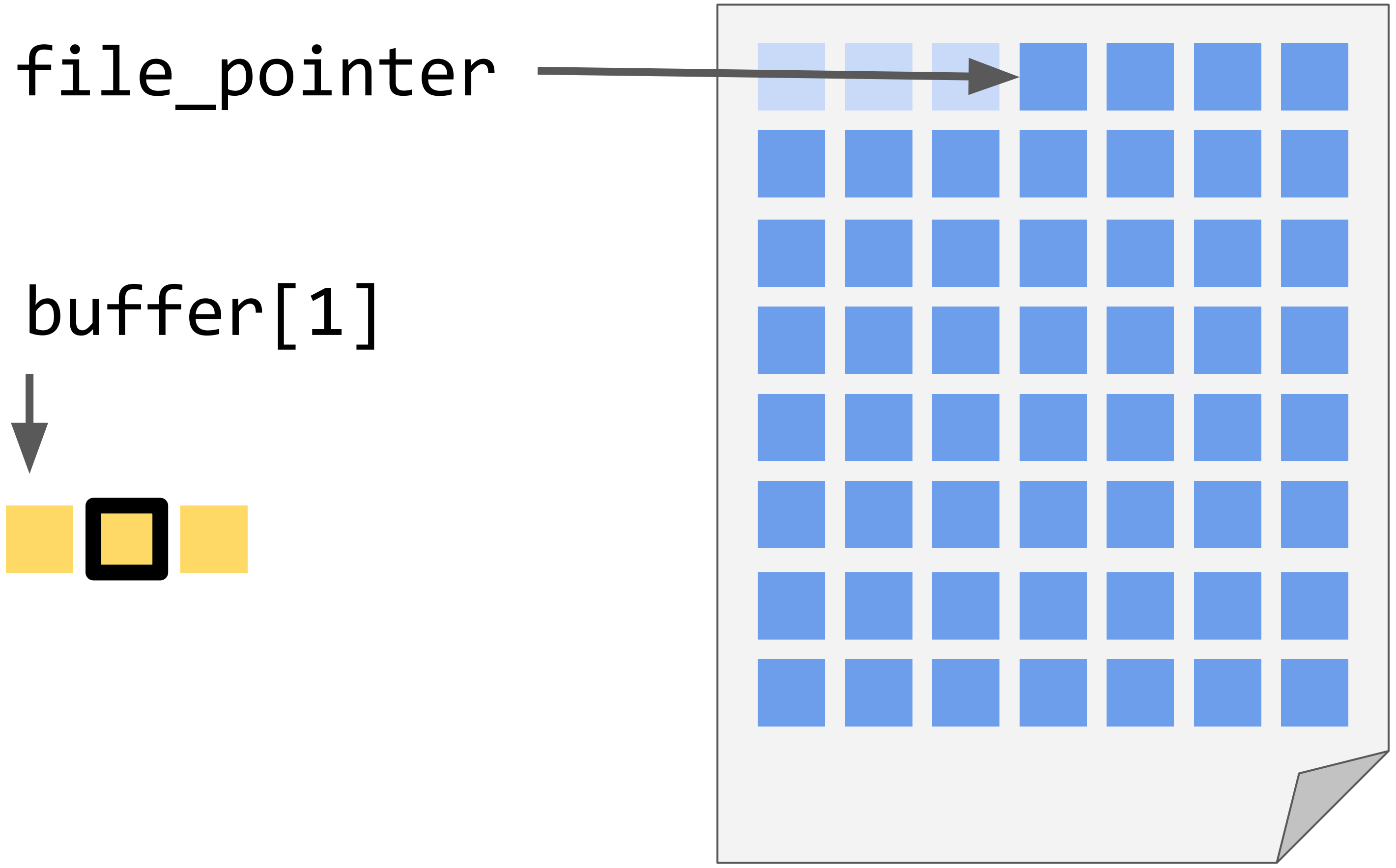
$\uparrow$

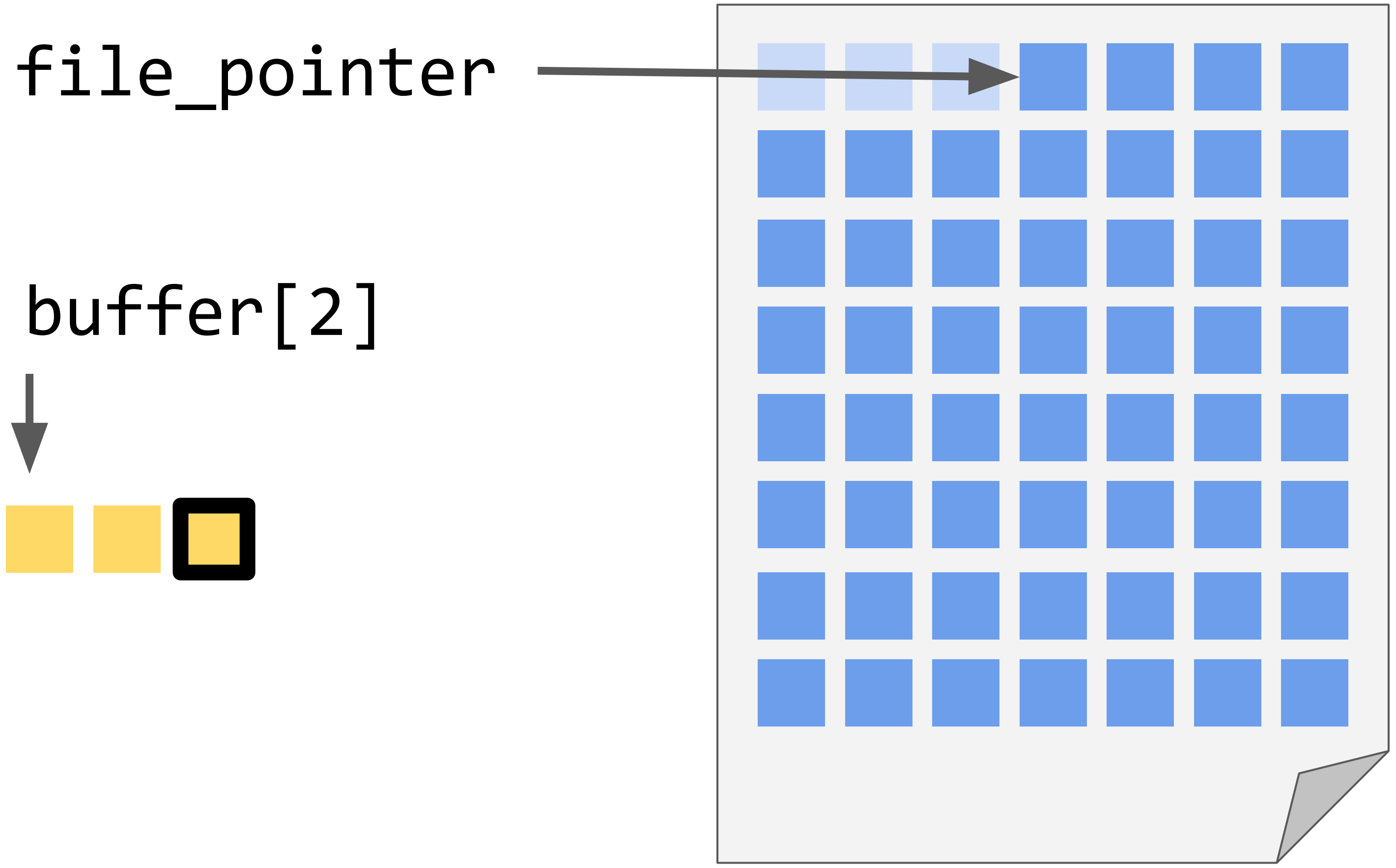
Location to store blocks


file_pointer
buffer


file_pointer
buffer

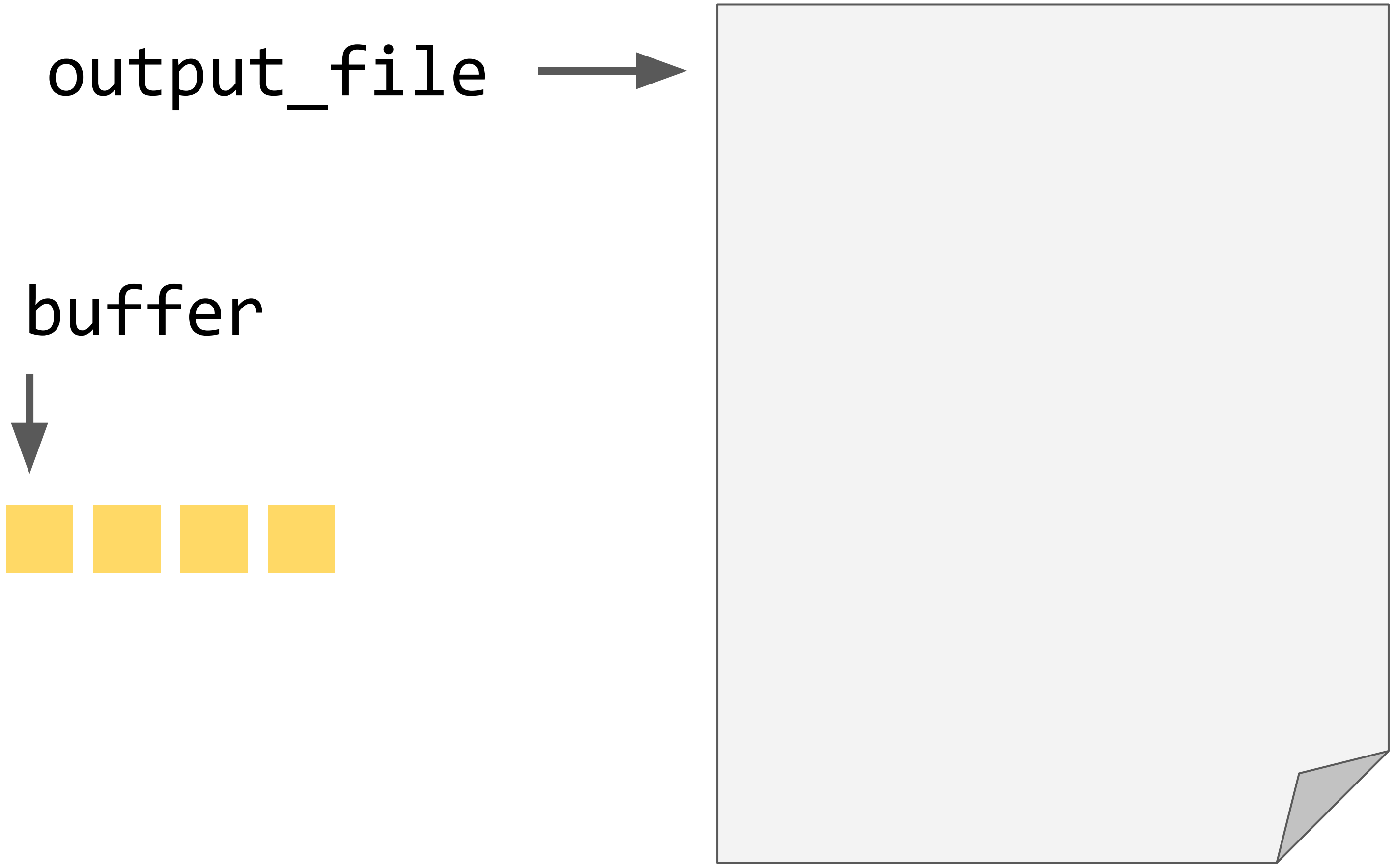

file_pointer
buffer[0]

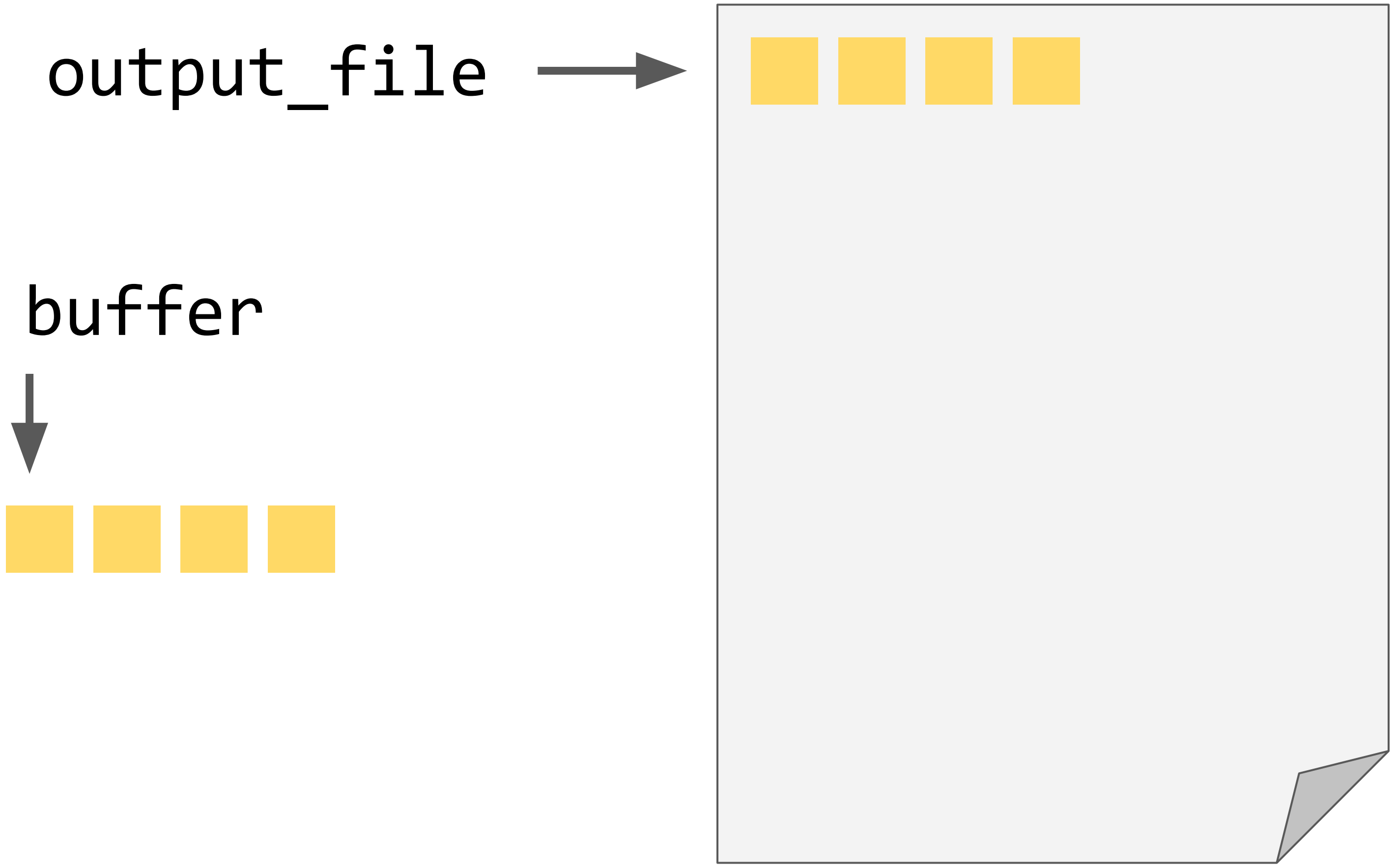

file_pointer
buffer[1]


file_pointer
buffer[2]

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

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


output_file
buffer


output_file
buffer

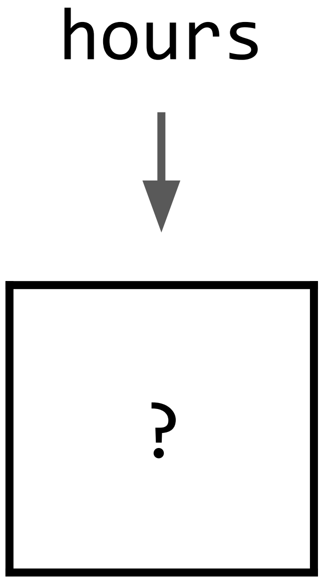
File Reading Exercise

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, corresponding to these integers:

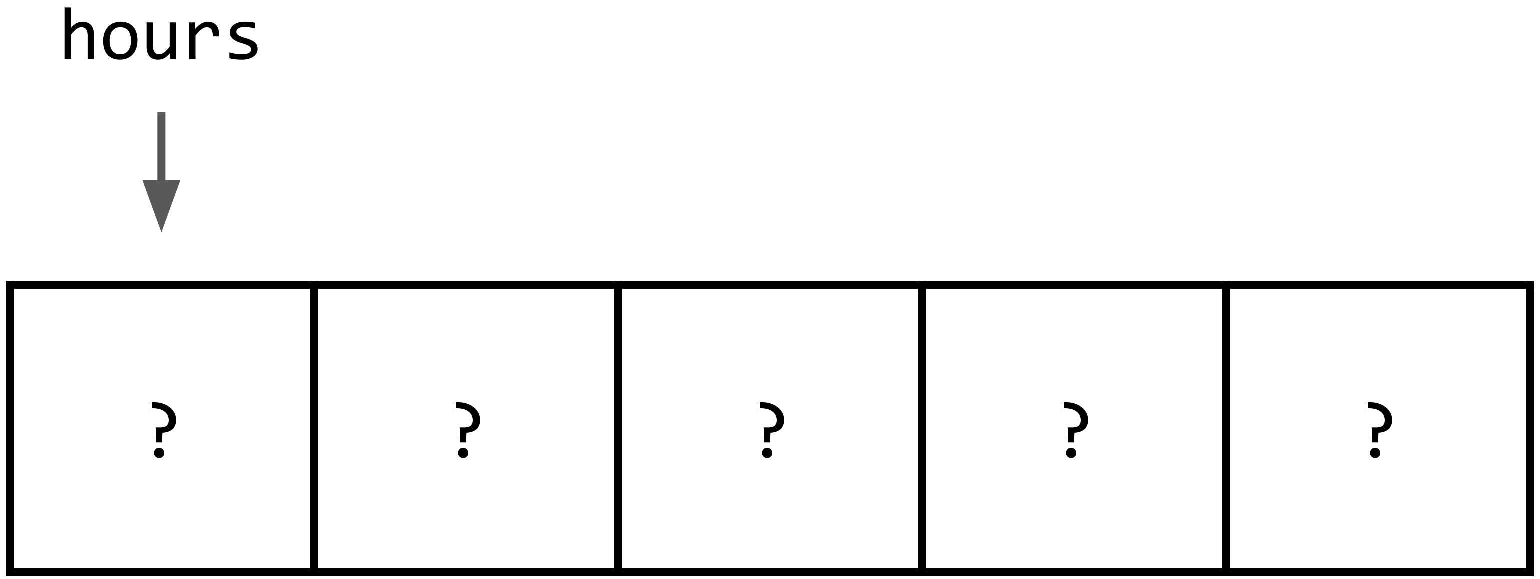
```
37 80 68 70
```

Dynamic Memory

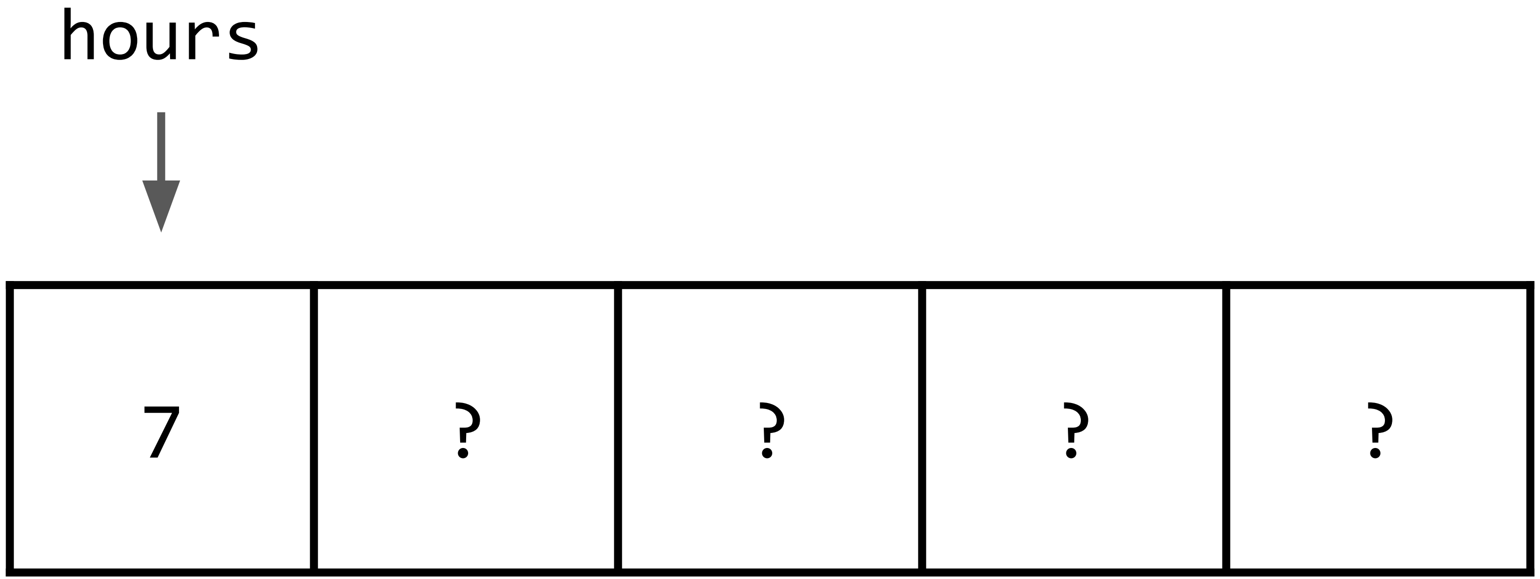
```
int *hours = malloc(sizeof(int));
```



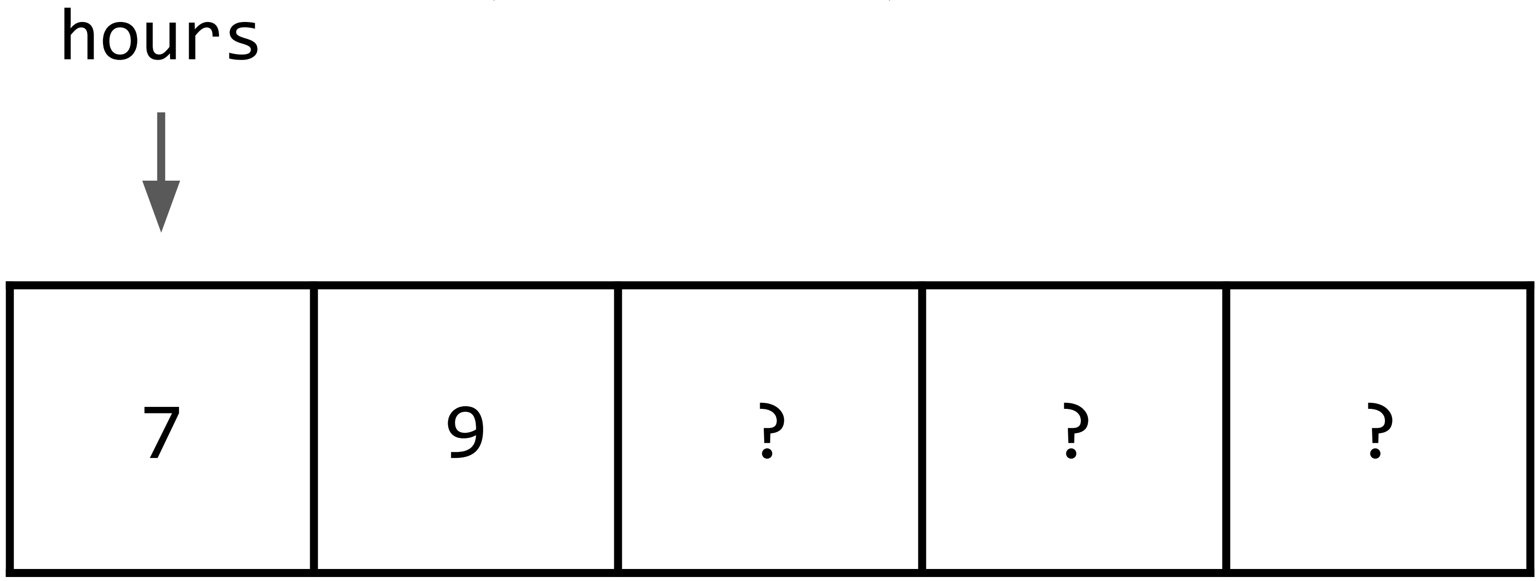
```c
int *hours = malloc(sizeof(int) * 5);
```



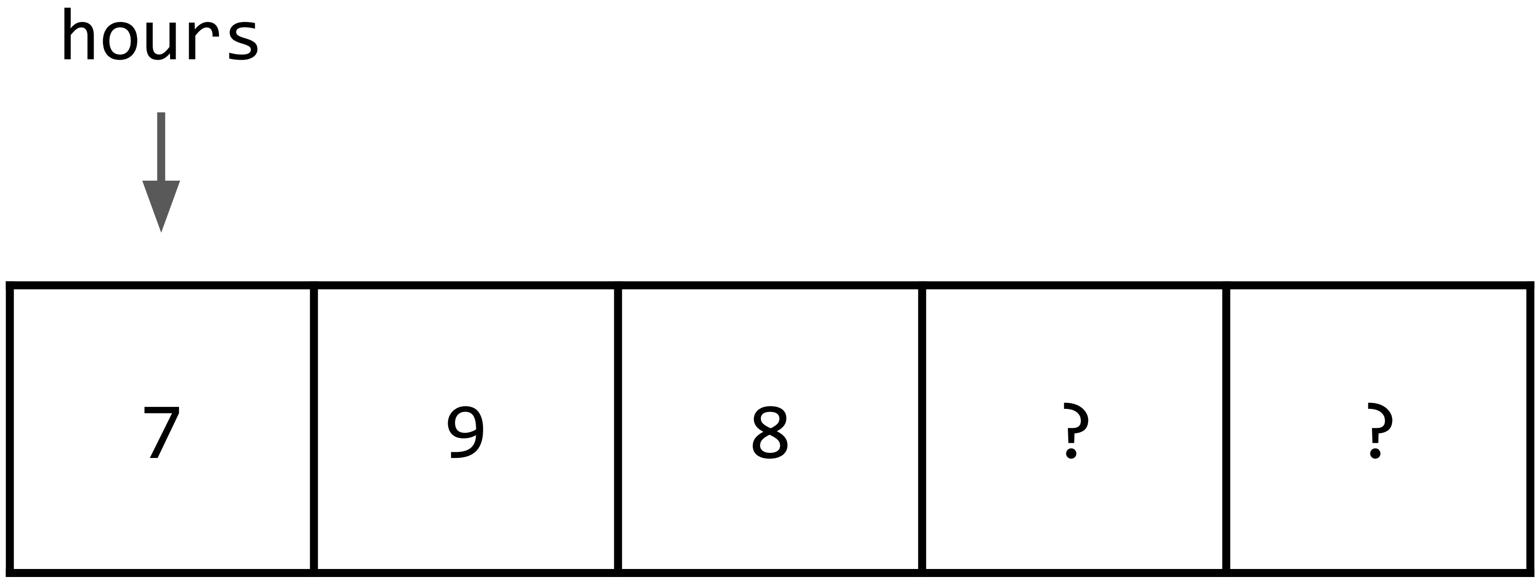
```
*hours = 7;
```



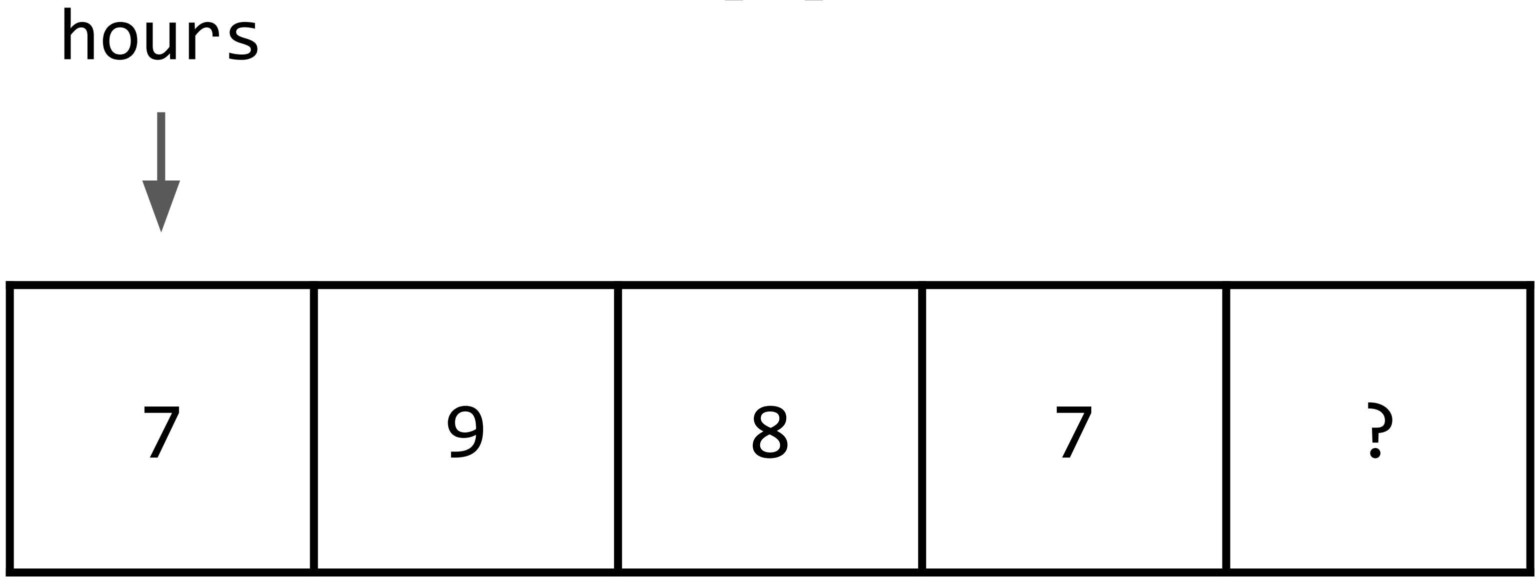
```
*hours = 7;
*(hours + 1) = 9;
```

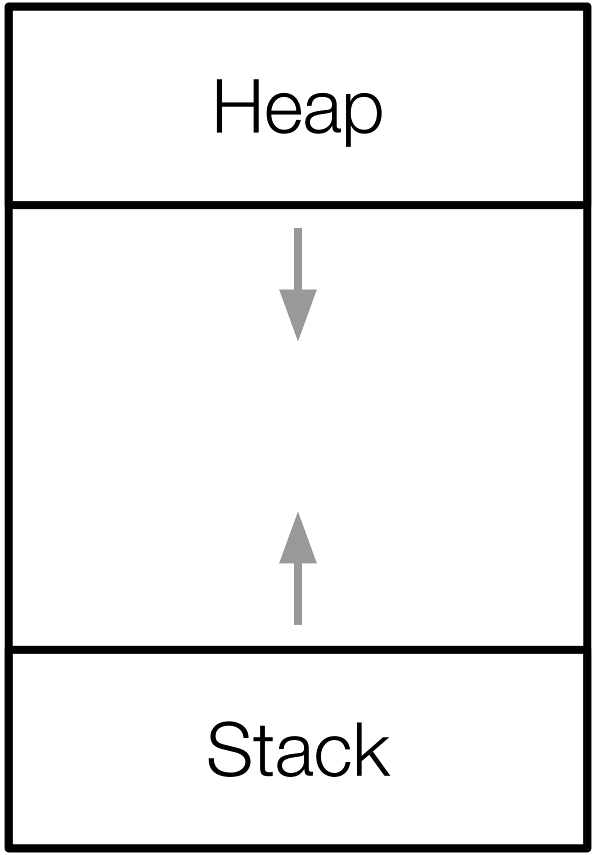


```
hours[2] = 8;
```



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




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
```

This was CS50