This is CS50.

Lab 4

How can we make our lab more engaging?

Please ask me if you have any questions any time any day! :)

Concept Deep Dive

Week 4 Concepts:

- Pointers
- Memory Allocation (malloc)
- File I/O
- Hexadecimals

Which one is the most confusing?

PASSING DATA IN C

Let’s think about how data is passed in C. What will the following code print?

```c
#include <stdio.h>

int main(void) {
   int x = 7;
   int y = x;
   x = 2;

   printf("%i\n", y);
}
```

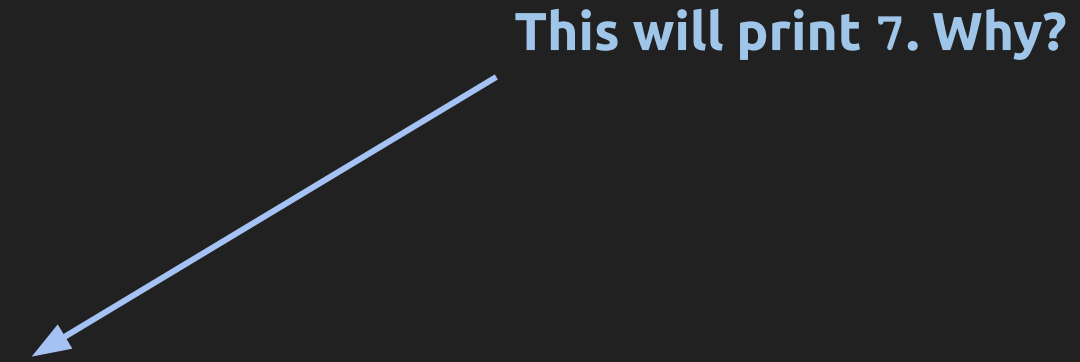
PASSING DATA IN C

Let’s think about how data is passed in C. What will the following code print?

```c
#include <stdio.h>

int main(void) {
   int x = 7;
   int y = x;
   x = 2;

   printf("%i\n", y);
}
```



PASSING DATA IN C

Check out this diagram which represents our computer *memory*:

The number 153 represents our location in memory. We're looking at two "slots" of contiguous memory right now.

PASSING DATA IN C

Check out this diagram which represents our computer *memory*:

We can imagine each memory slot to be 4 bytes each.

The number 153 represents our location in memory. We're looking at two "slots" of contiguous memory right now.

PASSING DATA IN C

```
int x = 7;
```

The program looks for a memory slot big enough to hold an integer, finds slot #153 is free, and then assigns `x` to it.

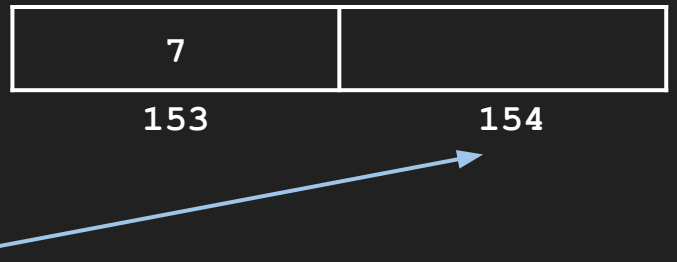
PASSING DATA IN C

```
int x = 7;
```

The program looks for a memory slot big enough to hold an integer, finds slot #153 is free, and then assigns `x` to it.

PASSING DATA IN C

```
int y = x;
```



The program looks for another memory slot big enough to hold an integer. It finds slot #154 and reserves it for `y`.

PASSING DATA IN C

```
int y = x;
```

It then goes to the slot in x and finds what that value is.

PASSING DATA IN C

```
int y = x;
```

It then goes to the slot in x and finds what that value is.

PASSING DATA IN C

```
int y = x;
```

Memory location #154 gets set equal to the value of `x`.

PASSING DATA IN C

```
x = 2;
```

The assignment operator has been passed the value of 2 to update what is stored in `x`'s memory location.

PASSING DATA IN C

```
x = 2;
```

The memory location for `x` is found and updated.

Any questions for me?

Pointers

WHAT IS A POINTER?

To learn to pass by reference, we must first learn about **pointers**.

A pointer is just the address to a location in memory

```
int x = 7;
```

How would you describe x's location in memory?

WHAT IS A POINTER?

To learn to pass by reference, we must first learn about **pointers**.

A pointer is just the address to a location in memory

```
int x = 7;
```

How would you describe x's location in memory? → X is at memory location **153**

WHAT IS A POINTER?

How can we represent the location of x as a pointer?

```
int x = 7;
```

WHAT IS A POINTER?

How can we represent the location of x as a pointer?

```
int x = 7;
int *x_pointer;
```

WHAT IS A POINTER?

How can we represent the location of x as a pointer?

```
int x = 7;
int *x_pointer;
```

An **int *** (int pointer) is of a different type than an **int**

Now… what do we store in `*x_pointer` ?

WHAT IS A POINTER?

How can we represent the location of x as a pointer?

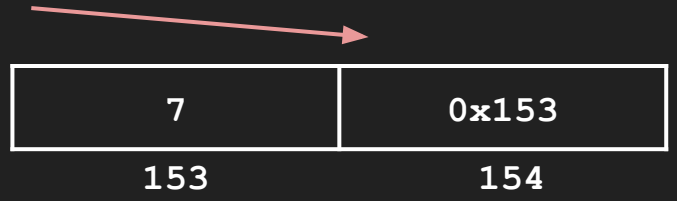
```
int x = 7;
int *x_pointer = &x;
```

& (ampersand) is the equivalent of saying “get the address of…”

WHAT IS A POINTER?

How can we represent the location of x as a pointer?

```
int x = 7;
int *x_pointer = &x;
```

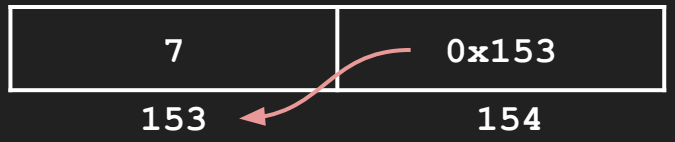


This line of code creates a new variable, x_pointer, to store the address of the variable x.

WHAT IS A POINTER?

So… why is it called a pointer?

```
int x = 7;
int *x_pointer = &x;
```

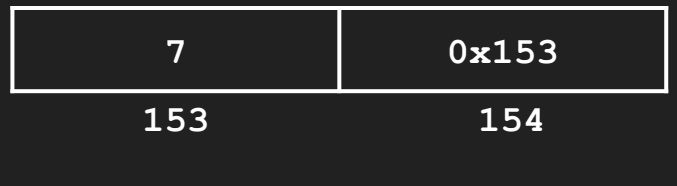


Because it “points” to an address in memory! In this case, x_pointer “points” to the address of variable x.

WHAT IS A POINTER?

What if we want to access the value held in the address that x_pointer “points” to?

```
int x = 7;
int *x_pointer = &x;
int y = *x_pointer;
```

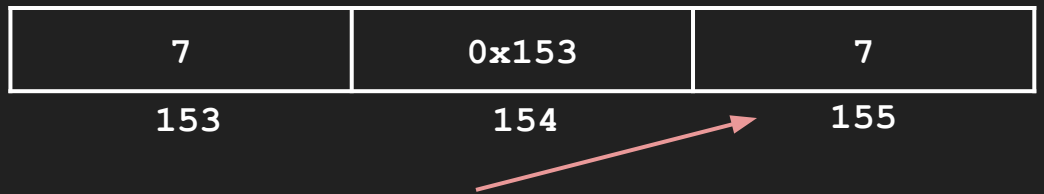


* is also the **dereference operator**, and it is used to “get” the value held at the address of a pointer. Think of ***** as saying “go to the value held at address…”

WHAT IS A POINTER?

What if we want to access the value held in the address that x_pointer “points” to?

```
int x = 7;
int *x_pointer = &x;
int y = *x_pointer;
```



Now we have stored the value that x_pointer “points” to in a new variable **y**, which is represented in memory location 0x155

POINTER REVIEW

`type *x_pointer` → creates a pointer that can store an address

POINTER REVIEW

`type *x_pointer` → creates a pointer that can store an address

`&x` → & Gets the address of a variable, **x**, in memory

POINTER REVIEW

`type *x_pointer` → creates a pointer that can store an address

`&x` → & Gets the address of a variable, **x**, in memory

```
x_pointer = &x;
```

↳ Sets the pointer **x_pointer** equal to the address, or location, of **x**

POINTER REVIEW

`type *x_pointer` → creates a pointer that can store an address

`&x` → & Gets the address of a variable, **x**, in memory

```
x_pointer = &x;
```

↳ Sets the pointer **x_pointer** equal to the address, or location, of **x**

`x_pointer` → x_pointer currently stores the location of x

POINTER REVIEW

`type *x_pointer` → creates a pointer that can store an address

`&x` → & Gets the address of a variable, **x**, in memory

```
x_pointer = &x;
```

↳ Sets the pointer **x_pointer** equal to the address, or location, of **x**

`x_pointer` → x_pointer currently stores the location of x

`*x_pointer` → Using * on a pointer **dereferences** it, getting the **value** stored at that location

POINTER PRACTICE

What will the following code print?

```c
#include <stdio.h>

int main(void) {
   int x = 7;
   int *y = &x;
   x = 2;

   printf("%i\n", *y);
}
```

POINTER PRACTICE

What will the following code print?

```c
#include <stdio.h>

int main(void) {
   int x = 7;
   int *y = &x;
   x = 2;

   printf("%i\n", *y);
}
```

2

This function uses pass by reference - y is a pointer to the address of x in memory. When we modify x, we modify the value that *p represents.

POINTER PRACTICE

What is the output of this function?

```c
# include <stdio.h>
void fun(int x)
{
    x = 30;
}

int main()
{
  int y = 20;
  fun(y);
  printf("%d", y);
  return 0;
}
```

POINTER PRACTICE

What is the output of this function?

```
# include <stdio.h>
void fun(int x)
{
    x = 30;
}

int main()
{
  int y = 20;
  fun(y);
  printf("%d", y);
  return 0;
}
```

20

The function **fun** uses pass by value, not pass by reference. As a result, a copy of y is passed into **fun**, preventing the actual value held at y from changing.

POINTER PRACTICE

What about this one?

```c
# include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}

int main()
{
  int y = 20;
  fun(&y);
  printf("%d", y);

  return 0;
}
```

POINTER PRACTICE

What about this one?

```c
# include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}

int main()
{
  int y = 20;
  fun(&y);
  printf("%d", y);

  return 0;
}
```

30

This time **fun** takes in a pointer, which means it is pass by reference. The **location** (pointer) to y is passed into **fun**, allowing the underlying value of y to change.

PRACTICE PROBLEM #1

Write a function on that uses pointers to swap two integers:

```
void swap(int* a, int* b)
{
    //TODO
}
```

PRACTICE PROBLEM #1 - SOLUTION

```
void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Any questions for me?

Memory Allocation

MEMORY ALLOCATION

You can utilize `malloc()` to allocate memory:

```c
#include <stdlib.h>

int main(void) {
   int *ptr = malloc(sizeof(int));
}
```

MEMORY ALLOCATION

You can utilize `malloc()` to allocate memory from the heap:

```c
#include <stdlib.h>

int main(void) {
    int *ptr = malloc(sizeof(int));
}
```

`malloc()` takes in the number of bytes you want to allocate and returns a memory address. We can use `sizeof()` to determine the size of a data type in C.

MEMORY ALLOCATION

When you’re done with that memory, you need to free it:

```c
#include <stdlib.h>

int main(void) {
    int *ptr = malloc(sizeof(int));
    free(ptr);
}
```

MEMORY ALLOCATION

When you're done with that memory, you need to free it:

```c
#include <stdlib.h>

int main(void) {
    int *ptr = malloc(sizeof(int));
    free(ptr);
}
```

`free()` takes a pointer to a memory location in the heap and frees it for you.

SOME ERRORS TO WATCH OUT FOR

- **Segmentation Fault -** You've tried to access an "illegal" area of memory or write to a read-only part of memory

SOME ERRORS TO WATCH OUT FOR

- **Segmentation Fault -** You've tried to access an "illegal" area of memory or write to a read-only part of memory
- **Stack Overflow -** Your functions have used up all the space available in the stack so they "overflow" out of it

SOME ERRORS TO WATCH OUT FOR

- **Segmentation Fault -** You've tried to access an "illegal" area of memory or write to a read-only part of memory
- **Stack Overflow -** Your functions have used up all the space available in the stack so they "overflow" out of it
- **Memory Leak -** You forget to free dynamically allocated memory, so you have less of it available as your program runs causing performance/memory issues

Any questions for me?

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");
```

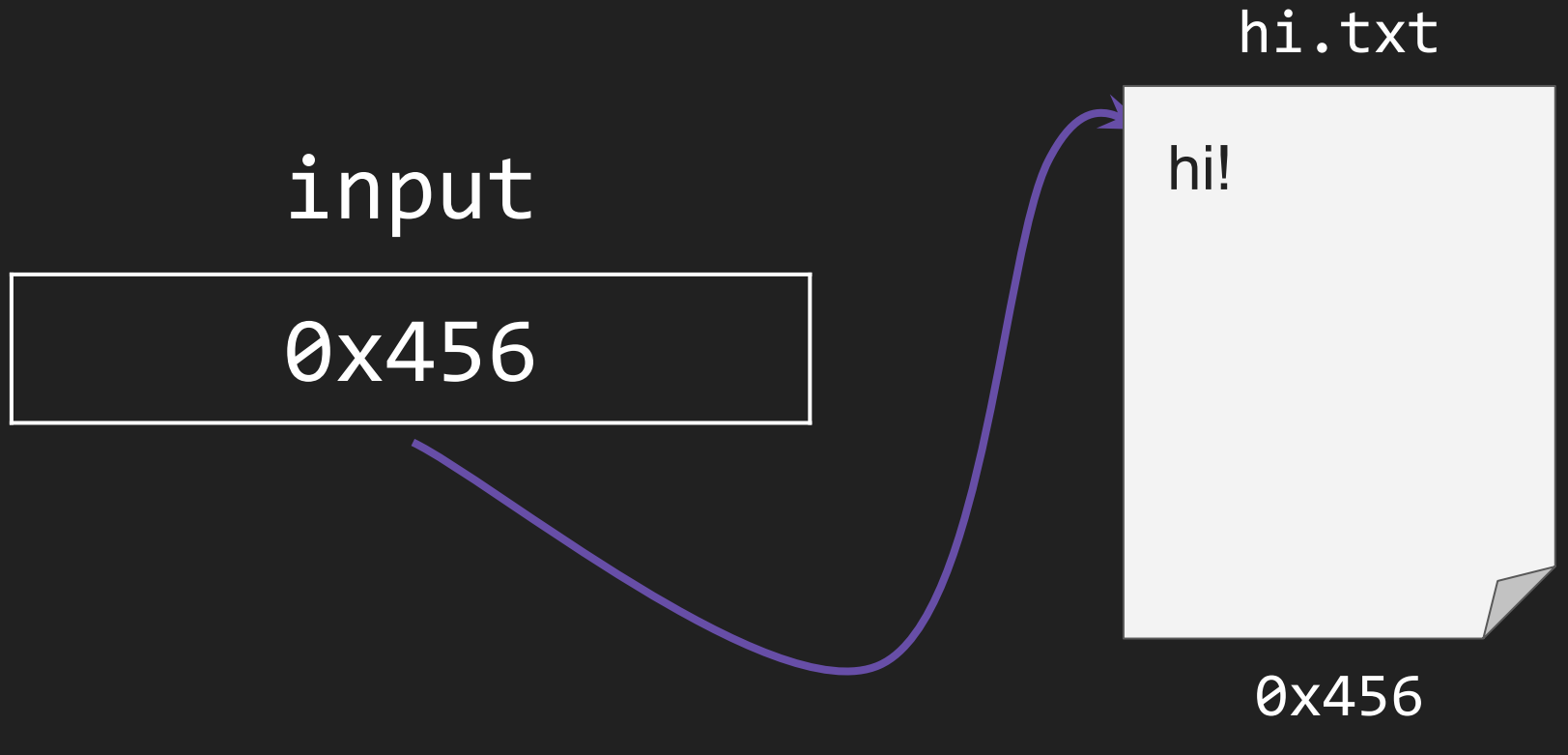
type

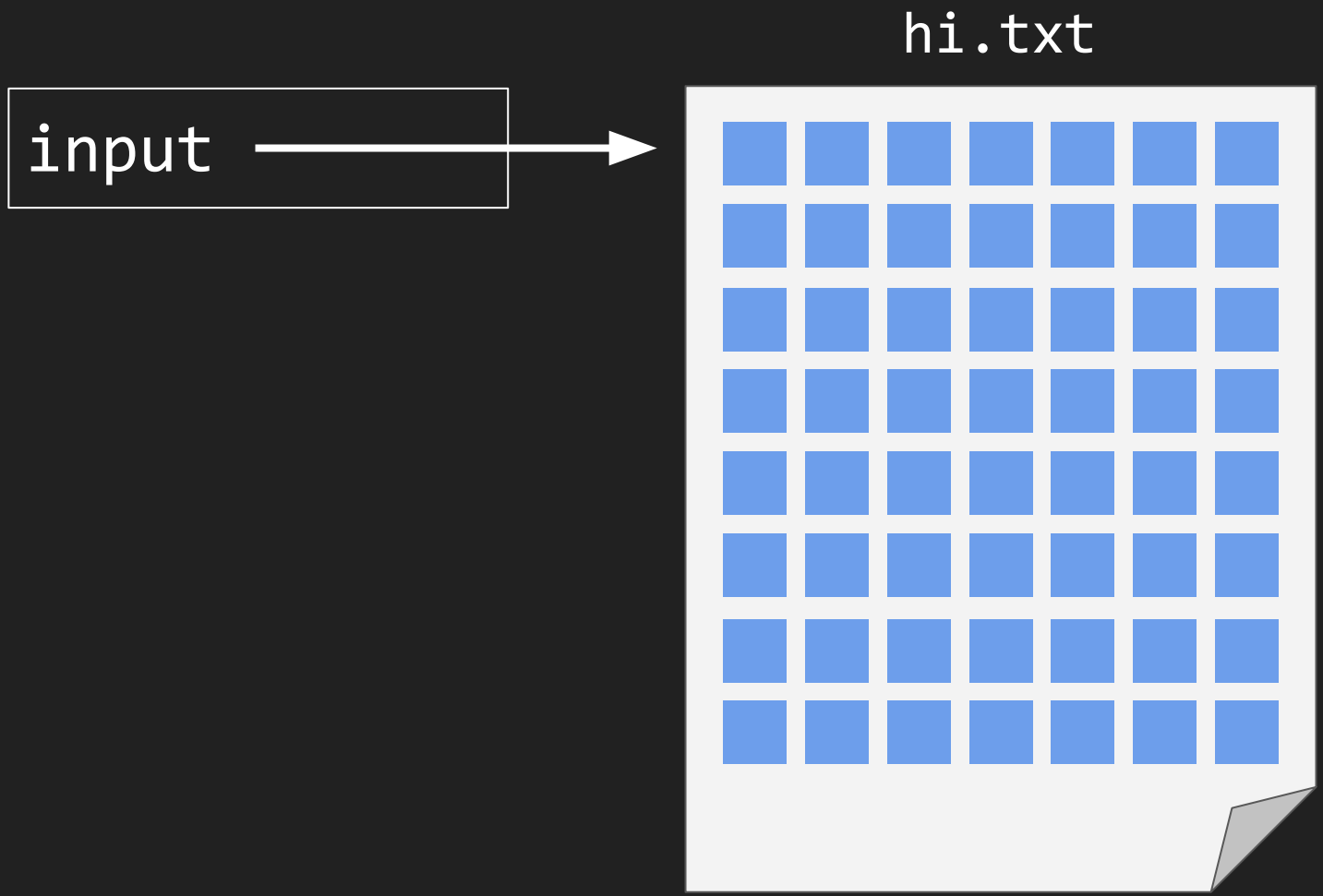
input

0x456

hi.txt

0x456


input
0x456
hi.txt
hi!
0x456


hi.txt
input

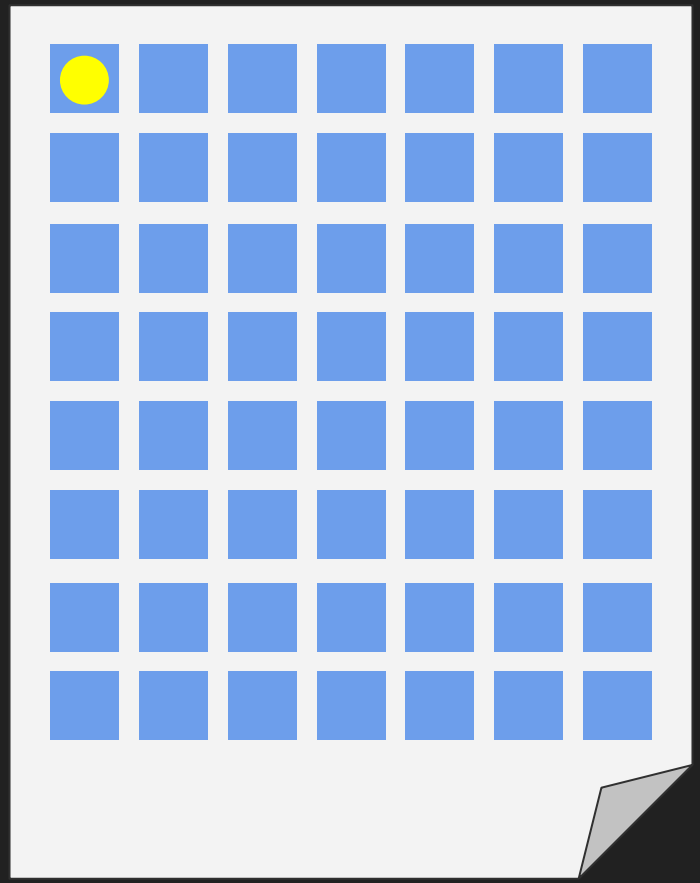
```
fread(buffer, 1, 4, input);
```

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

Location to read from

hi.txt

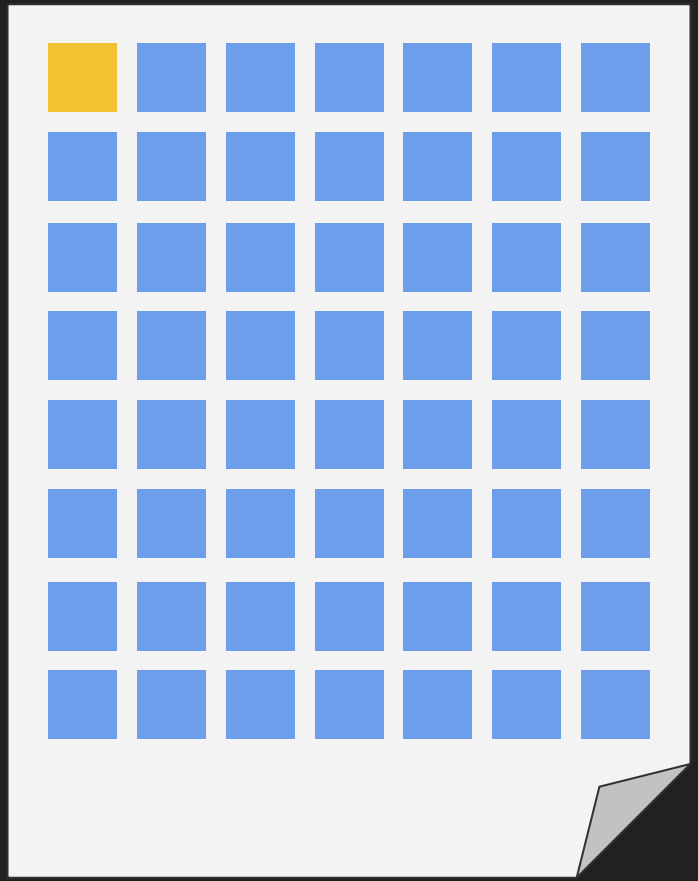
input



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

Size of blocks to read (in bytes)

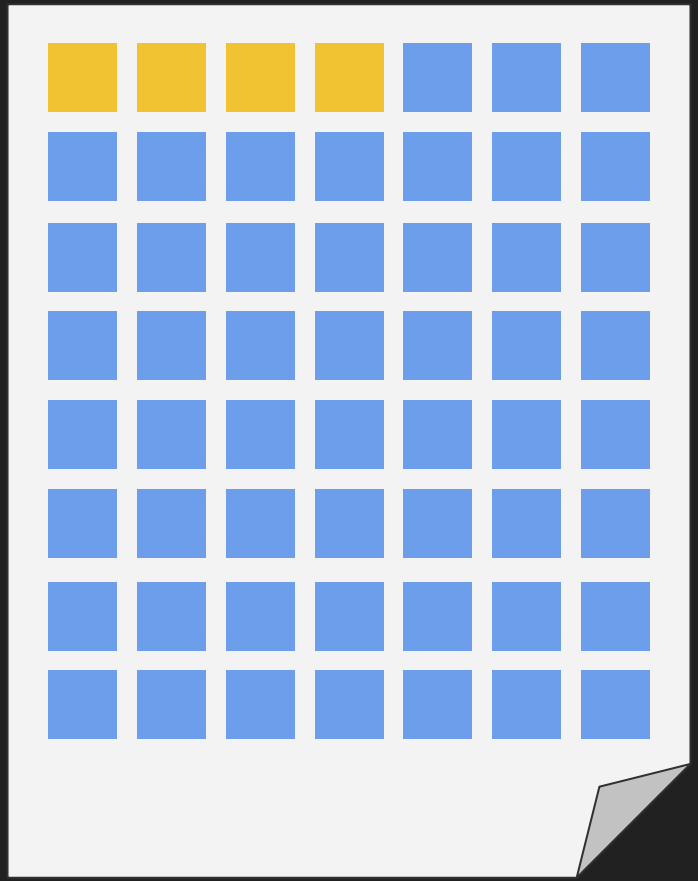
hi.txt



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

How many blocks to read

hi.txt

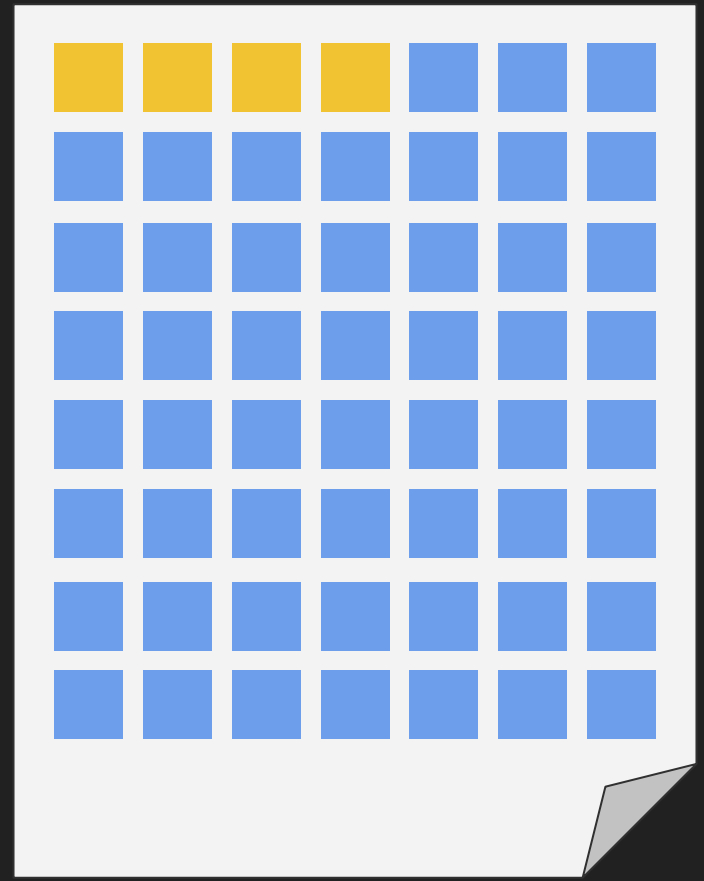


```
fread(buffer, 1, 4, input);
```

Location to store blocks

file_pointer

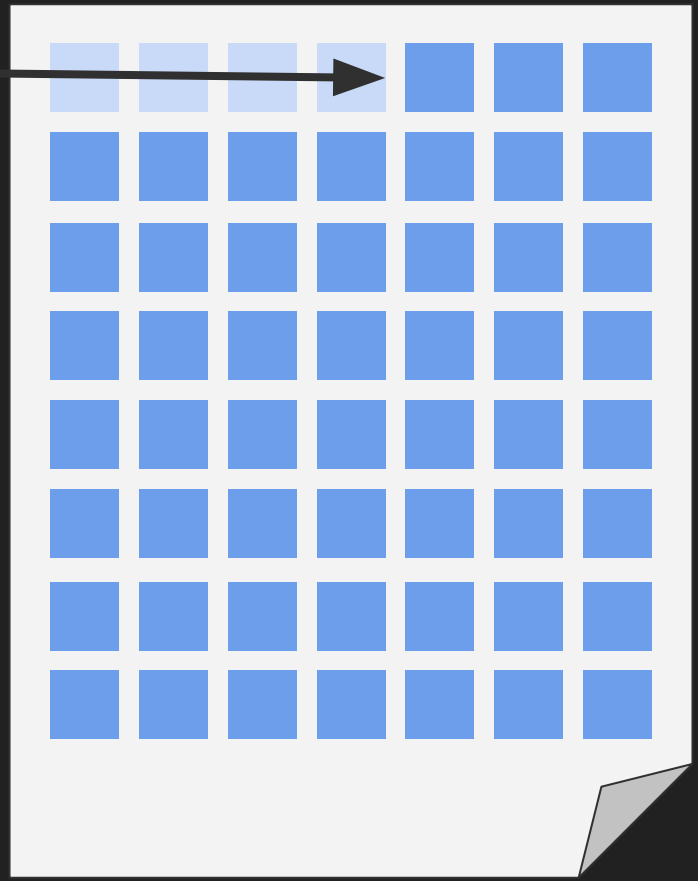
buffer



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

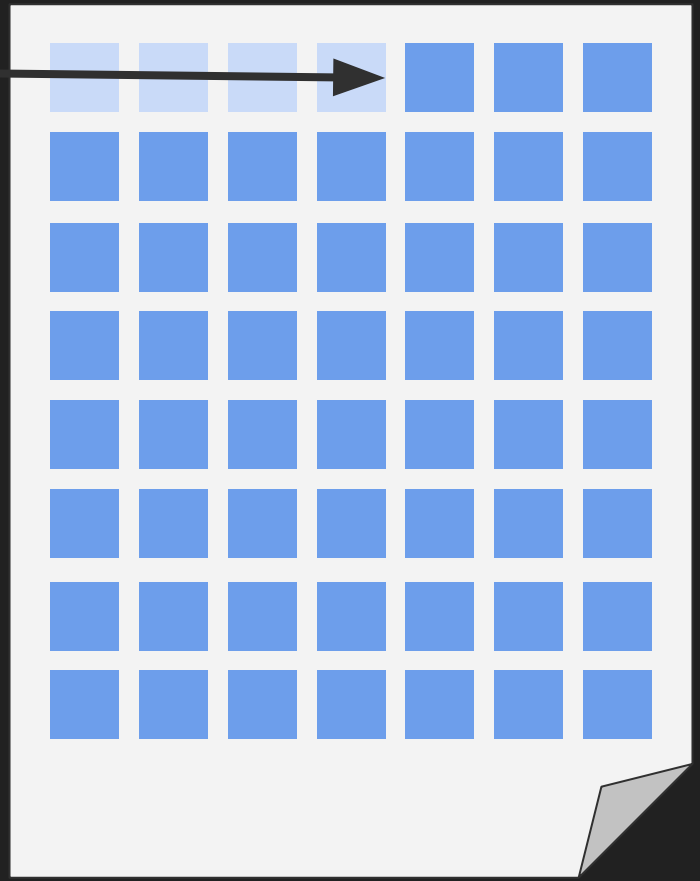
file_pointer

buffer



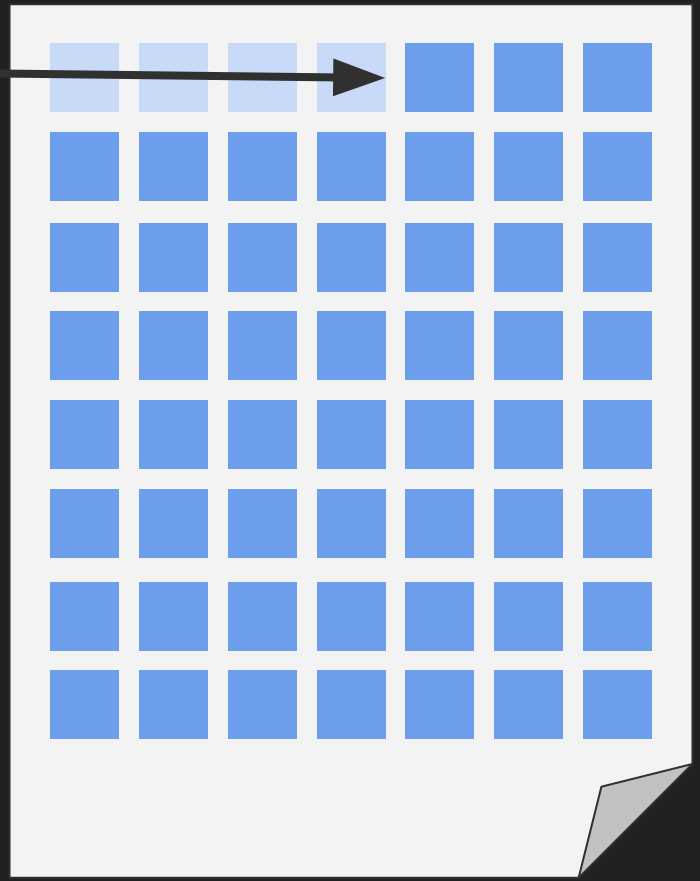
file_pointer

buffer[0]



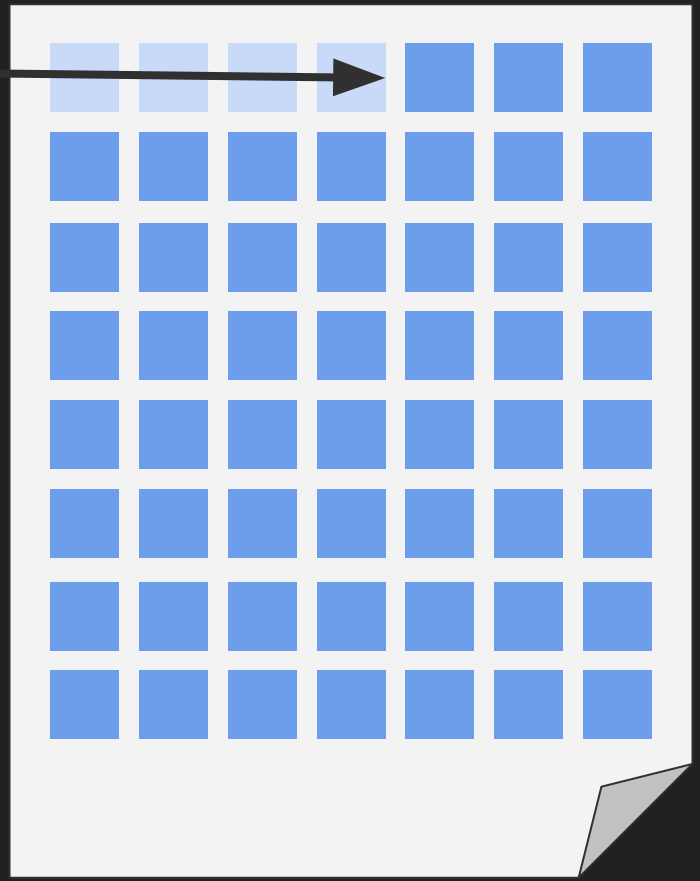
file_pointer

buffer[1]



file_pointer

buffer[2]

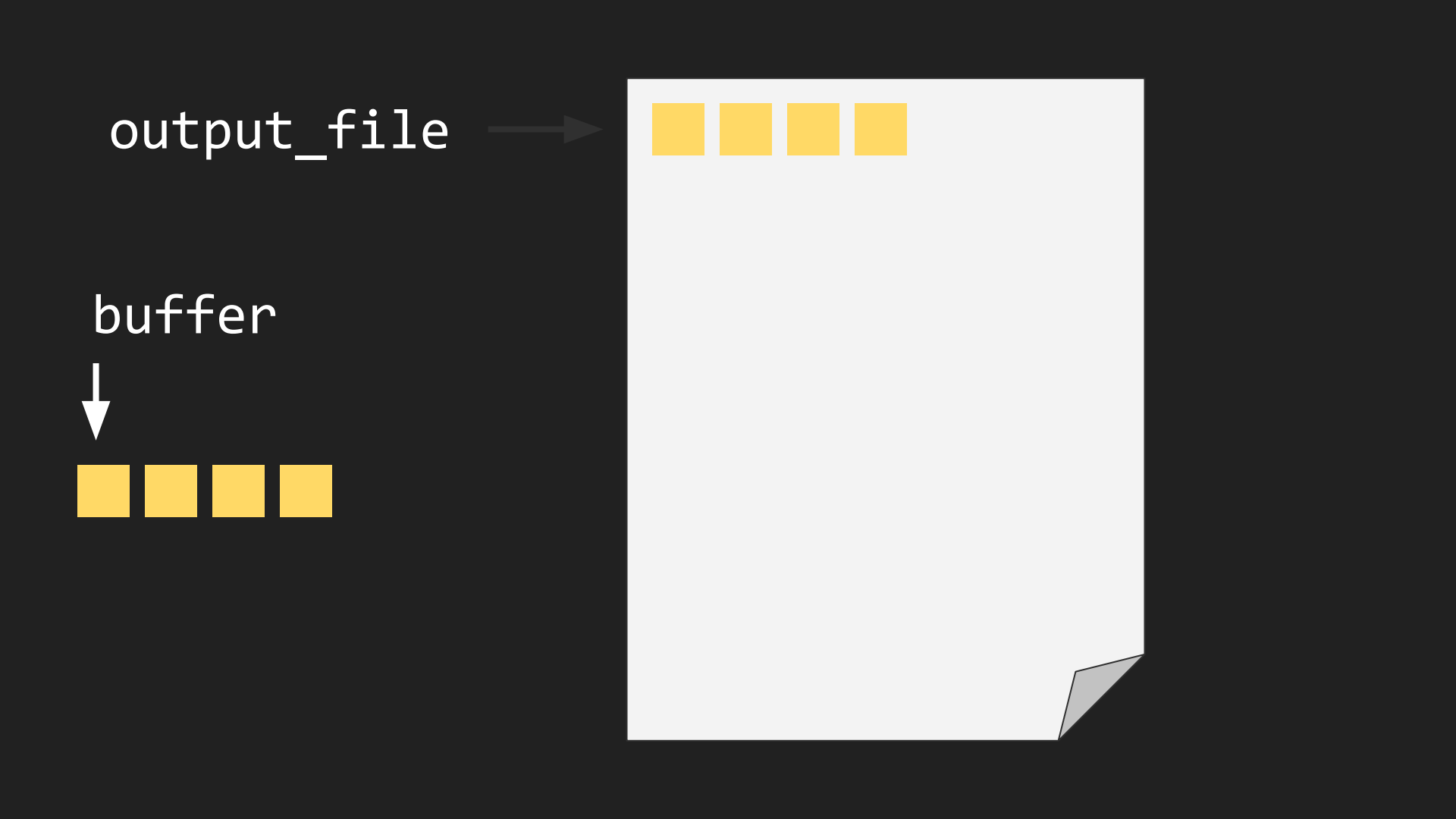


```
fread(buffer, 1, 4, input);
```

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

output_file

buffer


output_file
buffer

FILE I/O - Quick Reference

- `fopen()` - creates a file reference
- `fread()` - reads some amount of data from a file
- `fwrite()` - writes some amount of data to a file
- `fgets()` - reads a single string from a file (typically, a line)
- `fputs()` - writes a single string to a file (typically, a line)
- `fgetc()` - reads a single character from a file
- `fputc()` - writes a single character from a file
- `fseek()` - like rewind and fast forward on YouTube, to navigate around a file
- `ftell()` - like the timer on YouTube, tells you where you are in a file (how many bytes in)
- `fclose()` - closes a file reference, used once done working with the file

Any questions for me?

Lab

Task:

- Copy header from input file to output file
- Read samples from input file and write updated data to output file

Tips:

- You'll likely want to create an array of bytes to store the data from the WAV file header that you'll read from the input file. Using the `uint8_t` type to represent a byte, you can create an array of `n` bytes for your header with syntax like

```
uint8_t header[n];
```

- replacing `n` with the number of bytes. You can then use header as an argument to `fread` or `fwrite` to read into or write from the header.

Tips:

- You'll likely want to create a "buffer" in which to store audio samples that you read from the WAV file. Using the `int16_t type` to store an audio sample, you can create a buffer variable with syntax like

```
int16_t buffer;
```

- You can then use `&buffer` as an argument to `fread` or `fwrite` to read into or write from the buffer. (Recall that the & operator is used to get the address of the variable.)
-

Any questions for me?

Let’s do lab!

```
// TODO: Copy header from input file to output file
uint8_t header[HEADER_SIZE];
fread(header, HEADER_SIZE, 1, input);
fwrite(header, HEADER_SIZE, 1, output);
```

```c
// TODO: Read samples from input file and write updated data to output file
int16_t buffer;
while (fread(&buffer, sizeof(int16_t), 1, input))
{
    // Update volume
    buffer *= factor;
    fwrite(&buffer, sizeof(int16_t), 1, output);
}
```

Any questions for me?

Problem Set Tips

Problem Set: Filter (Less)

Prompt walkthrough, watch Bryan's Video

TIPS:

- **Minimal iterations through the image;** at most once for grayscale and sepia, at most twice for blur and edges, and at most one-half (up to width / 2) for reflect
- **Minimal casting, rounding, square rooting;** no more casts or rounds than are strictly necessary. You should only need a single (float) to do the math needed in this assignment
- **Clean handling of edge cases;** blur handles its edge cases by changing the bounds for the innermost two for loops, NOT by only running the contents of those two loops based on a condition (note that edges does have to use an inner condition)

Problem Set: Recover

Prompt walkthrough, watch Bryan’s Video

TIPS:

- Does not use any magic numbers
- Uses a bitwise operator (likely the & operator) to check the first four bits of the fourth header byte
- When opening a new file, uses an if statement to check if the file pointer being written to is NULL
- Only uses fwrite once-an-iteration within the jpeg discovery/creation loop (minimal repeated code between the case where header is found versus not found)
- Uses the ++ operator inside of the sprintf call to increment the file count
- Error handles within the while loop to ensure that fopen and fwrite calls work properly

Tutorials, OHs, More 1-1 too!! :)

Feedback form:

tinyurl.com/zad-feedback

Thank you!

See you next week!