Week 8

- Dynamic memory management
- Malloc, Free and ways to detect memory leaks

MALLOC

FREE

Where do we struggle?

- Constructors (Return a string from a function)
- Dynamic memory (Write some text, where to store?)
- Not being sure when you don’t need something anymore

Repetition:

Stack vs Heap

Stack

Heap

Heap

- You can ask for a memory block with a specific size
- You can free a block when you don’t need it anymore
- Knowing your heap separates bois from MEN

Allocation - malloc

- int *i = malloc(sizeof(int));
- int *i = malloc(10 * sizeof(int)); //Array of 10 ints
- What happens if you don’t have enough memory?

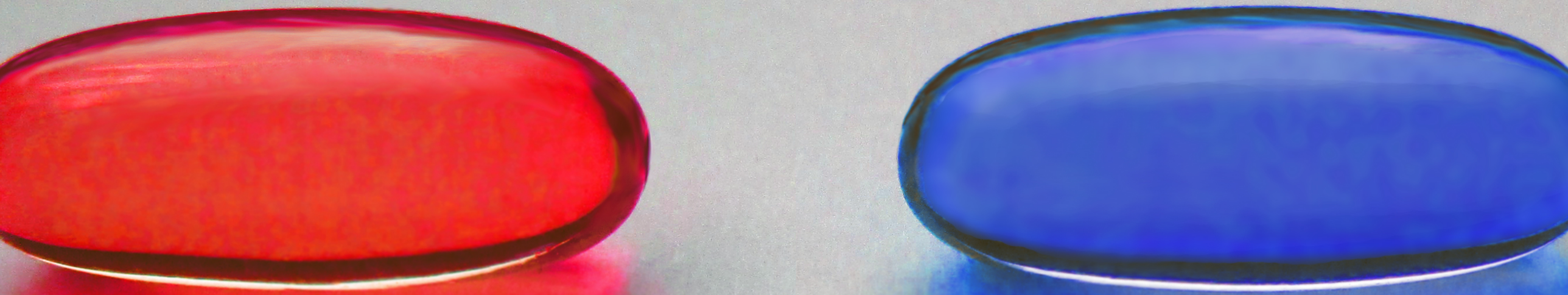
Deallocation - free

- int *i = malloc(sizeof(int));
  free(i); //the memory i occupied is freed

- int *i = malloc(10 * sizeof(int));
  free(i); //The complete array is freed

- int *i = malloc(sizeof(int));
  free(i); //the memory i occupied is freed
  free(i); //Your system will die in a car accident

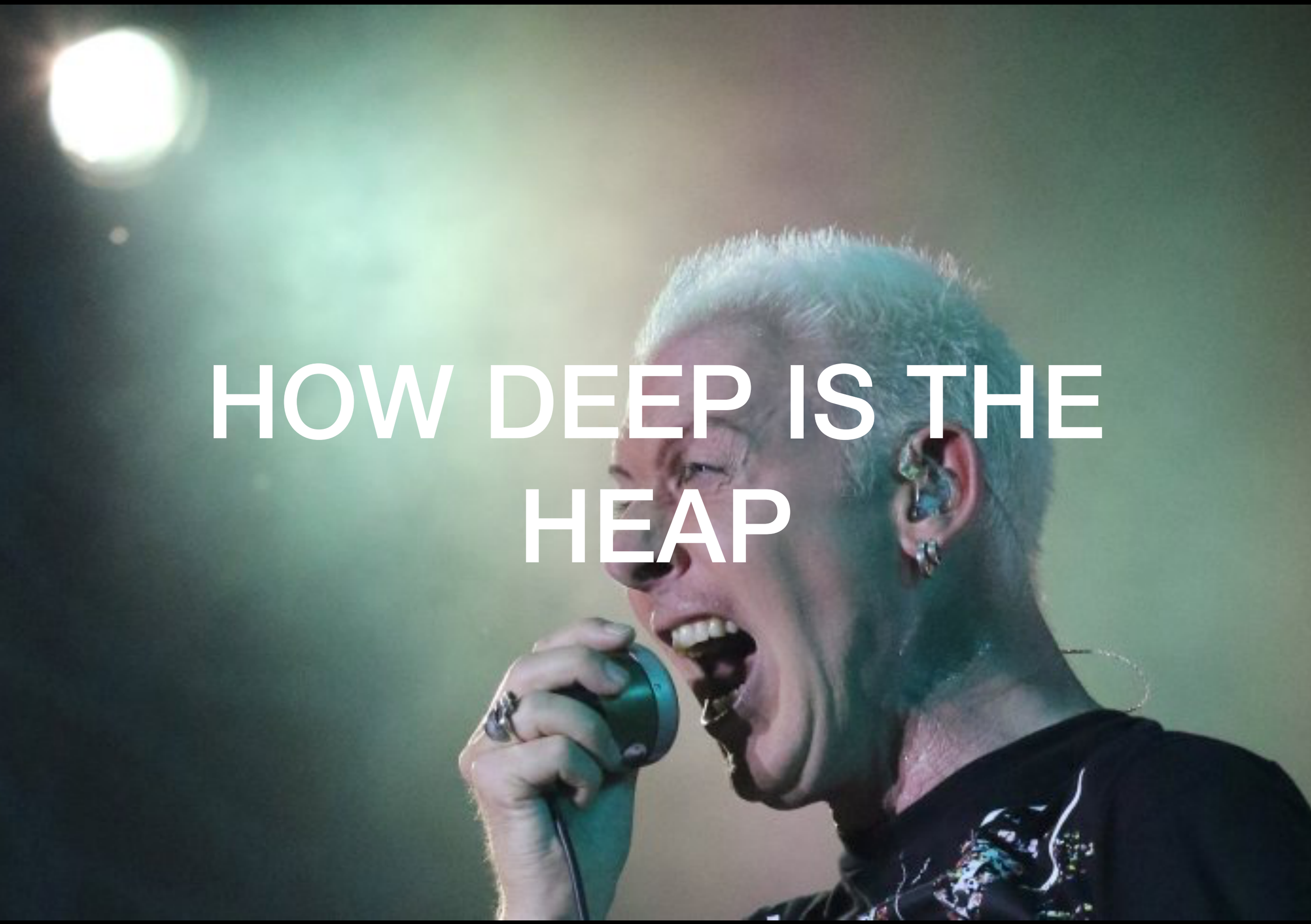
Blue pill:
Story ends, you wake up in your bed and never wonder why all elements are freed when an array is freed



Red pill:
You stay in wonderland and I show you how deep the rabbit hole goes

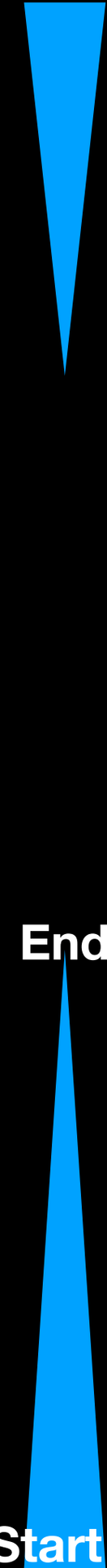
Why is the whole array freed?

- I want you to know this because the answer to this simple question explains dynamic memory management
- How is the whole array allocated in the first place?
- Who is keeping track over what is allocated?
- How is memory freed again?


HOW DEEP IS THE
HEAP

Where does the Heap end?

- The heap has a fixed starting and a fixed end point
- The end point can move up to increase the total heap size
- Some algorithm manages most of the heap for you



How allocate?

- When you allocate memory of size x, the algorithm searches for x free bytes in the heap, marks them as not free and returns the beginning of that area

- There are many algorithms, data structures and optimizations that help. E.g. Firstfit, nextfit, bestfit, having differently sized blocks (64, 512, 1024 bytes etc.)

- If the heap is full, the algorithm increases the heap end automatically (if it didn’t hit the stack)

How Free?

- You give the algorithm a pointer you want to free
- It searches the associated memory block and marks it as not allocated anymore
- Therefore if you free a pointer to an array of 10 int, the whole array is freed

Now you know it all, Mr. Anderson

What if you need more space?

realloc

- realloc(ptr*, newSize)
- Allocates buffer with new size and copies old contents

What happens if you malloc without freeing?

- The memory you allocated wastes space
- Your program will eventually crash
- Even small leaks will eventually add up
- Finding these is crucial to prevent crashes
- Difficult to debug

What happens if you free without mallocing?

- Calling free on some pointer that is not allocated causes the system to crash immediately

- Double free is dreaded because of it’s lethality

That’s a Pickle!

- Forgetting a free causes memory leaks
- Freeing one time too much causes an immediate crash

How can you deal with this?

- Create artificial constructors and destructors where all allocated memory is freed
- Exhaustive testing (in 2 Weeks!)
- Tools like Valgrind (Only available on UNIX)
- Thinking about structure

This sparked and interesting memory for me. I was once working with a customer who was producing on-board software for a missile. In my analysis of the code, I pointed out that they had a number of problems with storage leaks. Imagine my surprise when the customers chief software engineer said "Of course it leaks". He went on to point out that they had calculated the amount of memory the application would leak in the total possible flight time for the missile and then doubled that number. They added this much additional memory to the hardware to "support" the leaks. Since the missile will explode when it hits it's target or at the end of it's flight, the ultimate in garbage collection is performed without programmer intervention.

Exercises

- Write an ArrayList
  The ArrayList should have the methods append, remove, get and size
  Let it run with random numbers and operations for some time and look how much memory it needs after a while

- Russian roulette:
  in 1/1023 free random memory, in all else cases allocate a byte

- char* repeatText(char *text, int length, char repetitions)

More Exercises

- Look up how a LinkedList works and implement it
- Improve your ArrayList to work on void* types