Managing Memory with Pointers

Zachary Bennett
SOFTWARE ENGINEER
@z_bennett_ github.com/zbennett10

Pointer ins and outs

Why pointers

Advanced techniques

Managing memory

Dispel impossibility

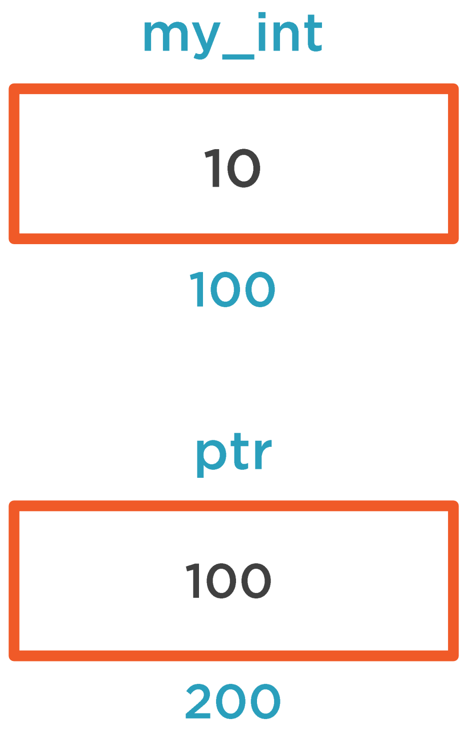
What Are Pointers

A pointer is a variable that contains a memory address.

Pointers Contain Memory Addresses

```
int my_int = 10;

int *ptr = &my_int;
```



```
main() {
  int number1, number2 = 10;
  int *pointer1;
  int* pointer2;

  pointer1 = &my_number;

  pointer2 = (int*)malloc(sizeof(int));

  number1 = *pointer1;
  *pointer2 = 50;
}
```

◀ Declaring pointers

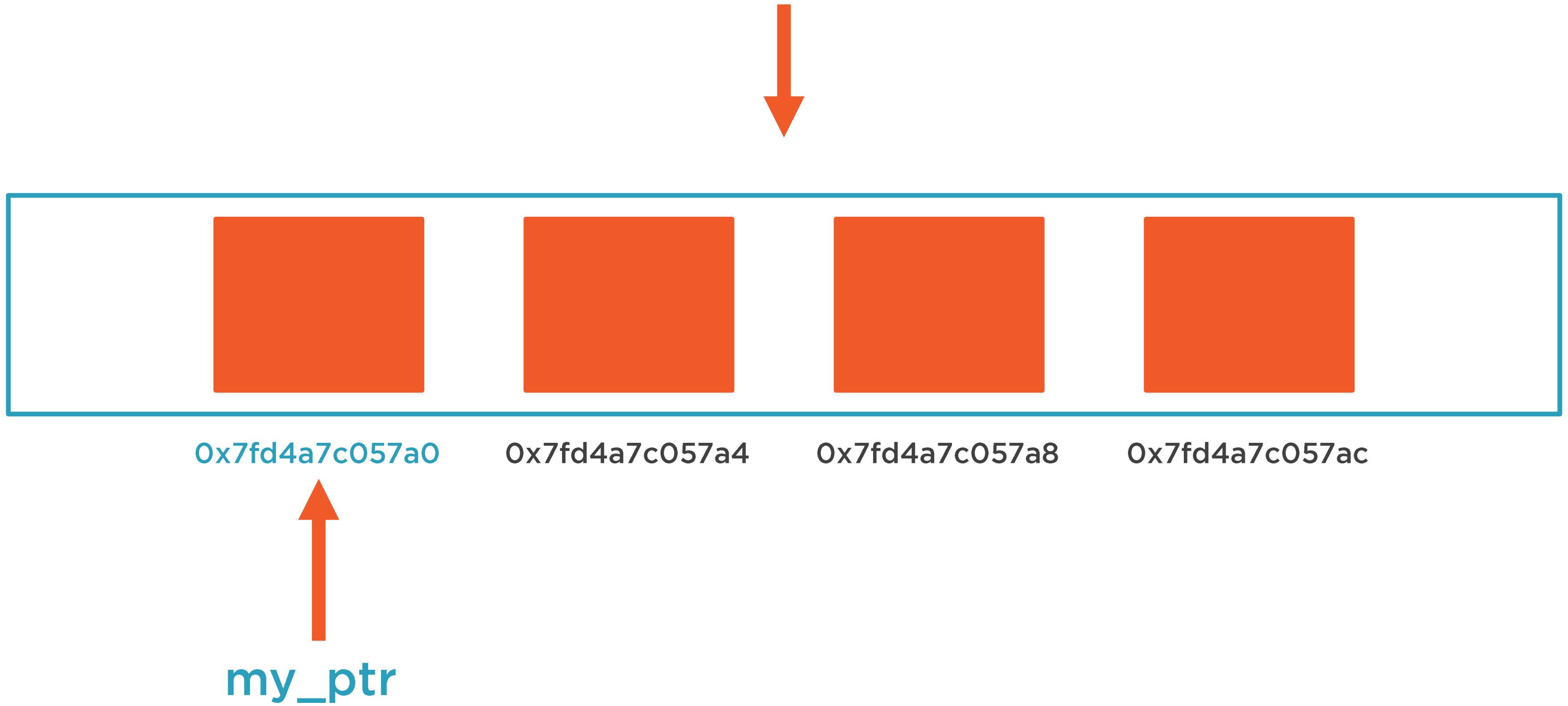
◀ Initializing a pointer using the ”&” operator

◀ Initializing a pointer using malloc

◀ Dereferencing a pointer

Pointers and Dynamic Memory Management

```
int *my_ptr = (int*)malloc(4 * sizeof(int));
```



Demo

Declaring a pointer

Initializing a pointer using the “&” operator

Initializing a pointer to dynamic memory using “malloc”

Dereferencing pointers

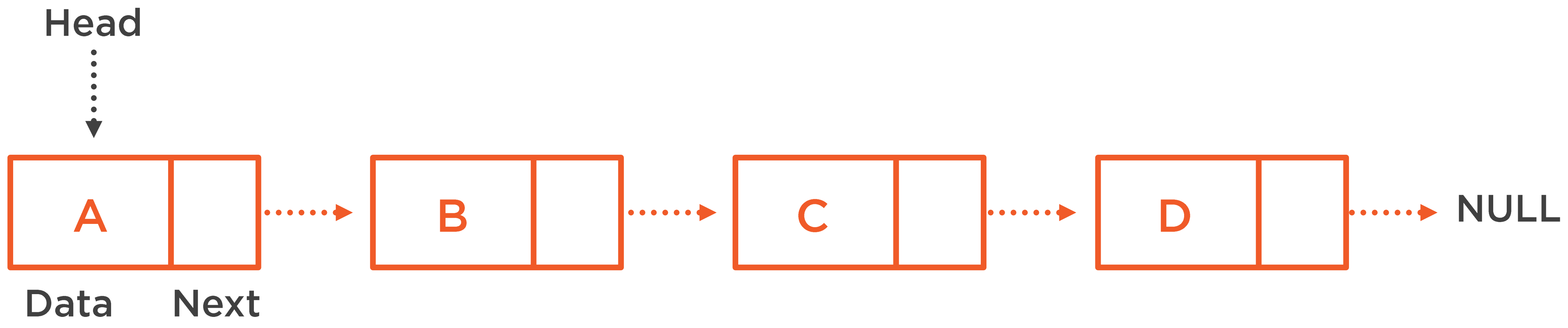
Why Pointers?

Enable the implementation of many data structures

Allows for passing values by reference

Dynamic memory management

Pointers and Data Structures



Pass by Value

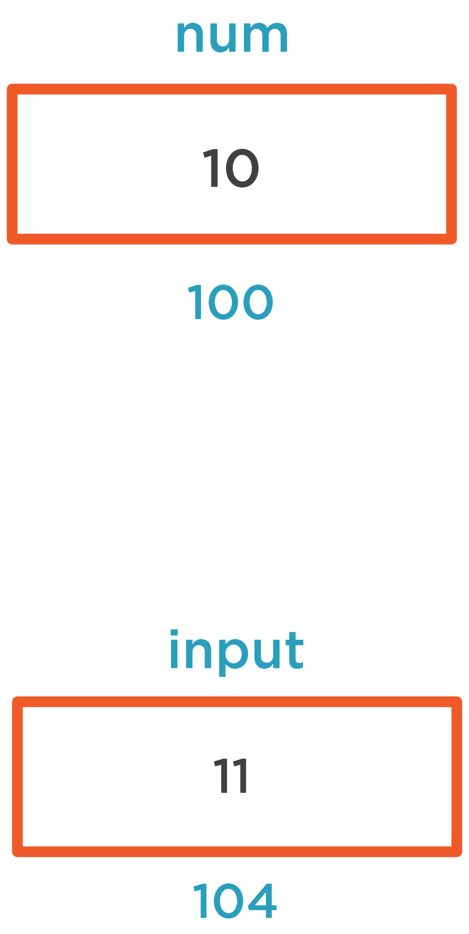
```
main {
   ...

   int num = 10;

   add_one(num);


   ...
}
```

```
void add_one(int input) {
  input + 1;
}
```



Pass by Reference

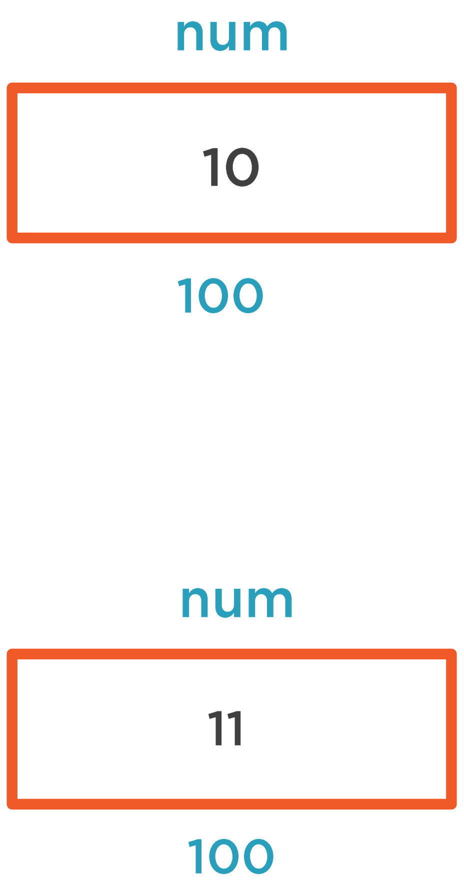
```
main {
   ...

   int num = 10;

   add_one(&num);

   ...
}
```

```
void add_one(int *input) {
  *input = *input + 1;
}
```

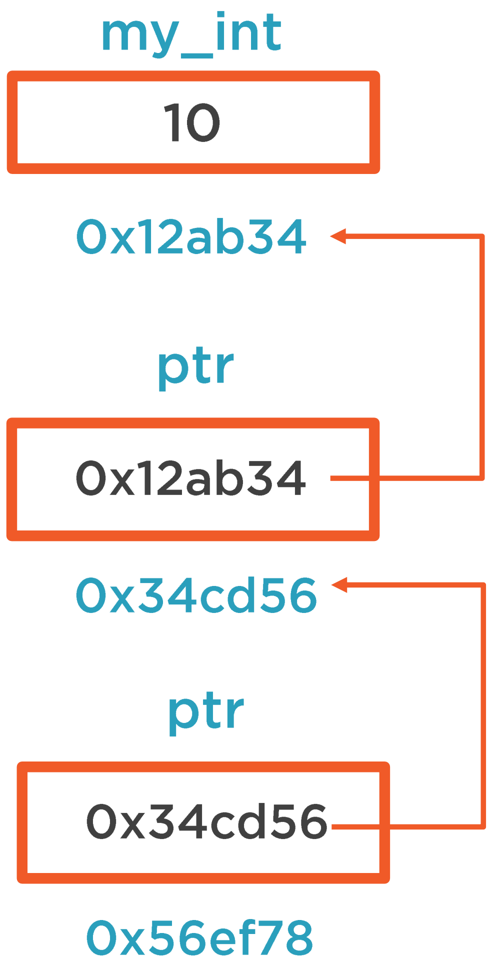


Pointers to Pointers

```
int my_int = 10;

int *ptr = &my_int;

int **dbl_ptr = &ptr;
```



Why Pointers to Pointers?

Multidimensional Arrays

Arrays of character strings

Passing pointers by reference

Demo

Printing memory addresses by value/reference

Passing by reference using pointers

- Attempt to alter a pointer using pass-by-value
- Step through function execution
- Alter pointer using pass-by-reference

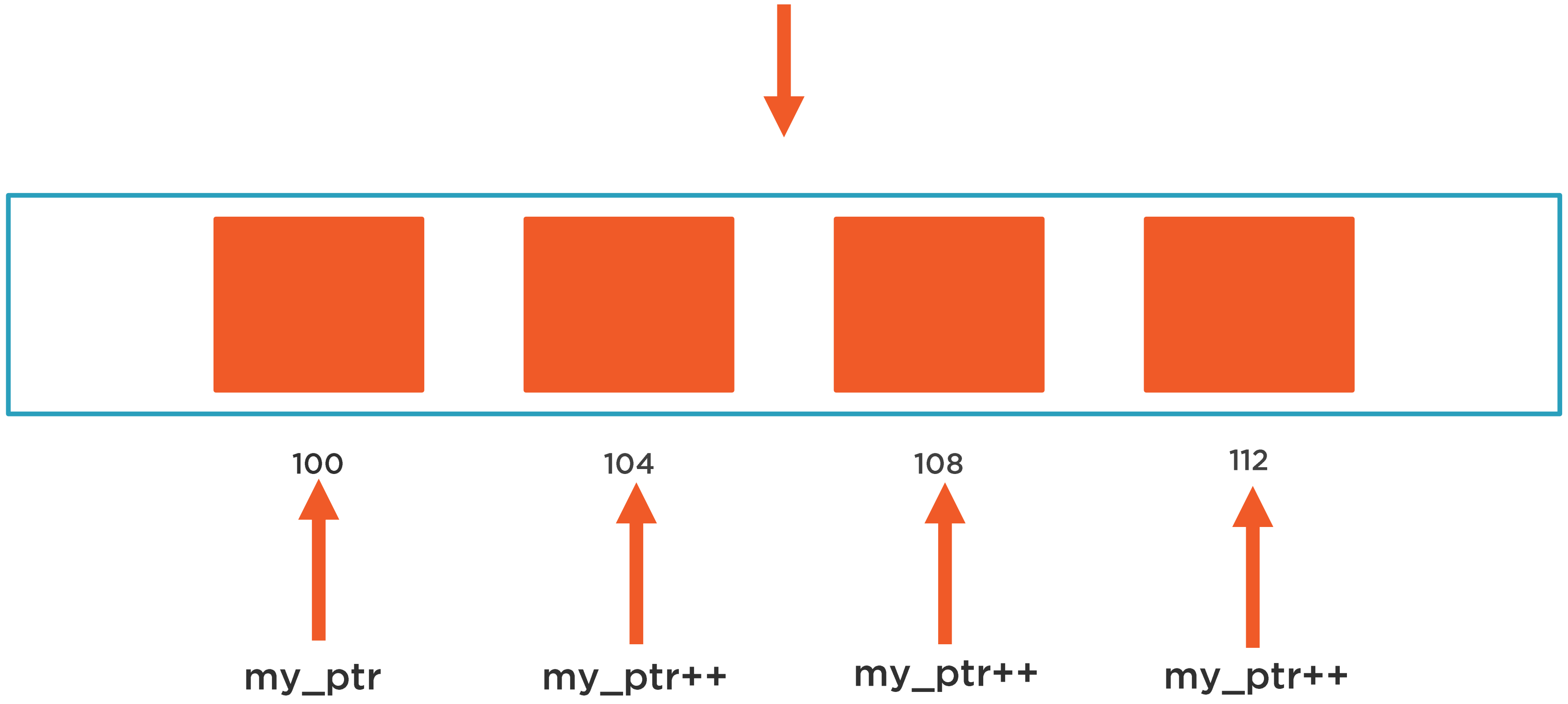
Don’t pointers contain memory addresses?

Aren’t memory addresses just numbers?

Pointer Arithmetic is simply memory address manipulation

Pointer Arithmetic

```
int *my_ptr = (int*)malloc(4 * sizeof(int));
```



When to Use Pointer Arithmetic

Calculating byte offsets

Comparing memory addresses

Rare cases where code is cleaner as a result

Dangers of Pointer Arithmetic

Readability

Pointer arithmetic usually makes code hard to read

Overflow

You can more easily write outside of memory bounds

Complexity

Operator precedence rules can be tough to understand

Demo

Incrementing pointers

Attach values to individual addresses

Dereference pointers

Danger of using pointer arithmetic

Function Pointer

A specific type of pointer that contains the beginning address of a function.

Why Function Pointers?

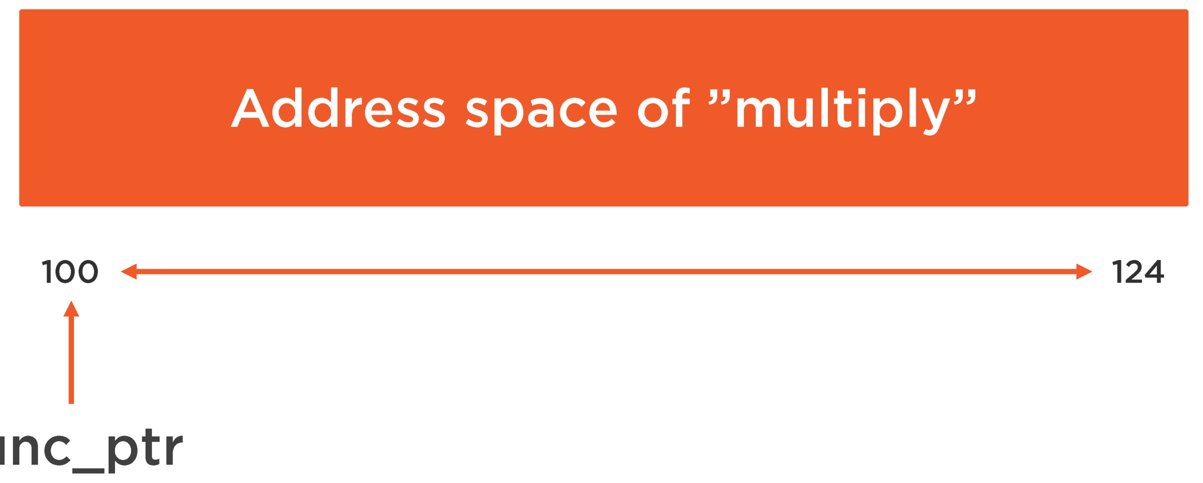
Dynamic
Determine which function to call at runtime

Callbacks
Handle events and compose asynchronous code

Abstraction
Core premise of comprising APIs in C

```
int (*func_ptr)(int, int);


int multiply(int a, int b) {
    return a * b;
}
func_ptr = &multiply;
```



Demo

Function pointer definition

Function pointer initialization

Usage with structures

Determining function calls at runtime

WIRED BRAIN
COFFEE

Wired Brain Coffee

Finding and fixing memory a leak

Team has identified a memory leak

Problem originates from within a function pointer

Demo

Identify a memory leak in a C program

Fix a memory leak that is created within Wired Brain Coffee’s code

While refactoring, you will see

- Pointer Arithmetic
- Function Pointers
- Pass-by-reference

See non-trivial examples of malloc’d pointers

Overview/ Summary

Pointers contain memory addresses

Why Pointers

- Data Structures
- Pass-by-reference
- Dynamic memory management

Pointer Arithmetic

Function Pointers

Manipulating Dynamically Allocated Memory