

Tutorial 07 - 03.12.2019

Group 06 - Moritz Makowski

Pointers

Today's Agenda

- Pointers
- **Exercise 7.1: Pointers**
- Functions and Pointers
- Arrays and Pointers
- **Exercise 7.2: Arrays and Pointers**
- Structs and Pointers
- **Exercise 7.3: Practice**

What are Pointers?

Every time we create a variable, its value has to be stored somewhere in memory.

Each slot in memory has a specific address.

A Pointer is simple an adress of a slot in memory.

See `example_7_1_pointer_intro.c` on Github

Getting the Address of a Variable

This is also called "Referencing".

Every Pointer also "has a type" which is equal to type of its corresponding variable.

```
int *pointer_a; // points to a memory slot which stores an integer
```

The star marks that `pointer_a` - which you can name however you want - is a pointer.

You can get the actual address of a variable by using `&`:

```
int my_var = 55;  
int *my_pointer = &my_var;
```

Getting the Value Belonging to a Pointer (Address)

This is also called "**Dereferencing**".

You can get the corresponding value by using `*`:

```
int my_var_2 = *my_pointer; // Now stores 55
```

```
printf("\nThe adress of my_var is %p", my_pointer); // prints 0x7ff...  
printf("\nThe value of my_var is %d", my_var);      // prints 55  
printf("\nThe value of my_var is %d", *my_pointer); // prints 55
```

Exercise 7.1: Pointers

- (a) Given `char c = 'K';`, what kind of type would I need to hold `&c`?
- (b) Given `float *ff`, what kind of data does `ff` hold?
- (c) Given `char *c`, what exactly is `&c`?
- (d) What, if anything, is wrong with the following code?

```
int main(void) {  
    char *some_value;  
    char my_value = '!';  
  
    some_value = my_value;  
  
    return 0;  
}
```

Revision: Passing Data to Functions

What will be printed out?

```
void add_10(int value) {  
    value += 10;  
}  
  
int main() {  
    int my_value = 10;  
    add_10(my_value);  
    printf("My value is %d\n", my_value);  
  
    return 0;  
}
```

See `example_7_2_passing_by_value.c` on GitHub.

What happens inside `add_10` ?

```
void add_10(int value) {  
    // A local copy of "value" gets created  
    value += 10;  
    // The local copy now stores 20  
}
```

To get this local variable "out of" the scope of `add_10` we'd have to include a return statement and a "re-assignment" inside `main` .

Passing Data by Reference

You can also pass a pointer to a variable. Any copy of that pointer still points to the same slot in memory. And we don't modify the pointer itself but the memory space it points to.

```
void add_10(int *value) {  
    *value += 10;  
}  
  
int main() {  
    int my_value = 10;  
    add_10(&my_value);  
    printf("My value is %d\n", my_value); // prints 20  
  
    return 0;  
}
```

See `example_7_3_passing_by_reference.c` on *GitHub*.

Arrays and Pointers

The array variable is actually stored as a pointer.

```
int main() {  
    int my_array[10] = {0,1,4,9};  
    printf("\nAddress of the first element = %p", &(my_array[0]));  
    printf("\nAddress of the first element = %p", my_array);  
  
    printf("\n\nAddress of the second element = %p", &(my_array[1]));  
    printf("\nAddress of the second element = %p", my_array + 1);  
    return 0;  
}
```

See `example_7_4_array.c` on *GitHub*.

See `example_7_5_array_index_access.c` on GitHub.

See `example_7_6_array_pointer_access.c` on GitHub.

Exercise 7.2: Arrays and Pointers

```
char string_buffer [1000];  
char *my_char_ptr = string_buffer;
```

- (a) What kind of data would `*my char ptr` be?
- (b) What about `*string buffer`?
- (c) Let's say you want to store `&(string buffer[2])` somewhere. What type of variable do you need to store this?

(d) What, if anything, is wrong with the following code?

```
char old_buffer[1000];  
char *my_char_ptr = &old_buffer;  
  
char string_buffer[1000];  
string_buffer = my_char_ptr;
```

(e) If you pass an array to a function and you change elements of the array in the function, what do you think happens when you leave the function?

Structs and Pointers

There is a shortcut in accessing element inside a struct, that is present as a pointer.

```
struct Point {  
    float x,  
    float y,  
    float z  
};  
  
struct Point *my_struct;
```

Access by dereferencing the struct first

```
*(my_struct).x = 1.2;
```

Clean version:

```
my_struct->x = 1.2;
```

Exercise 7.3: Practice

See You Next Week!

All **code examples** and **exercise solutions** (available right after my tutorial) on **GitHub**.

<https://github.com/dostuffthatmatters/Engineering-Informatics-1-MSE-WS1920>.



When you copy a snippet from
StackOverflow and it doesn't work

