

Pointers

Help Sessions Check timetable!

Revision sessions reminder

Pointers

.....

.....

.....

.....

.....

.....

.....

Memory

- All data (variables) are stored in **memory**
- You can think of memory as a big grid
- Each segment of this grid has a unique identifier

.....

.....

.....

.....

.....

.....

.....

Visualising memory with addresses

Memory					
32 bits					
0x00: NULL	0x00: 53	0x01: 'a'	0x02: 0.35		
		0x19: 'j'	0x20: 'k'	0x21: 'k'	0x21: 'E'

.....

.....

.....

.....

.....

.....

.....

- We can also access the address
- By storing that address in a variable, we have a **pointer**

[illegible]

To declare a pointer

^ The `*` means don't request the storage to store `<type>`, but requests memory to store a memory address of `<type>`

```
int *pointer  
  
struct student  
*student
```

Visualise pointer declaration

```
// declare a pointer to an integer
int *number; // operating system returns 0x17
```

0x00: NULL	0x00: 53	0x01: 'g'	0x02: 0.35		
0x17: 0x1231		0x19: 'j'	0x20: 'A'	0x21: 'K'	0x21: 'E'

Address of operator &

- What if we want to query what the address of a variable is?
- We can use the address_of operator:

&

Syntax of address of: &

<variable>

Example

```
int number = 2;
&number // the address of number
```

```
int number = 2;
```

```
int *pointer_to_number =  
&number
```

Memory

32 bits					
0x00: NULL	0x00: 53	0x01: 'a'	0x02: 0.35	0x03: 2	
			0x14: 0x03		
0x17: 0x1231		0x18: 'j'	0x20: 'K'	0x21: 'K'	0x21: 'E'

.....

.....

.....

.....

.....

.....

.....

.....

Dereferencing

- Dereferencing is simply accessing the value at the address of a pointer
- It uses the `*` symbol again (which causes confusion)
- `*my_int_pointer` -> will get the integer at the address location

.....

.....

.....

.....

.....

.....

.....

.....

Three components to pointers in code

```
int main(void) {  
    // Declare an integer  
    int my_age = 23;  
  
    // Declare an integer pointer  
    // Assign it the address of my_age  
    int *pointer_to_my_age = &my_age;  
  
    // Print out the address and value  
    at the pointer  
    printf("Pointer is: %p value is:  
    %d\n", pointer_to_my_age,  
    *pointer_to_my_age)  
    return 0;  
}
```

.....

.....

.....

.....

.....

.....

.....

.....

Common mistakes

```
int number;  
int *number_ptr;
```

1. `number_ptr = number;`
2. `*number_ptr = &number;`

Syntax cheat sheet

- Declare a pointer: `int *int_ptr;`
- Address of: `&my_variable;`
- Dereference (Get the value at a pointer): `*int_ptr;`

Demo

Goals:

- ☐ Create a variable
- ☐ Get the address of that variable
- ☐ Create a pointer variable
 - ☐ Use it!

But JAKE, why are they *USEFUL*

- Let's look at an example with pointers and parameters

How can we edit a variable within a function?

Pass by reference*

```
#include <stdio.h>

void change_value(int *x) {
    *x = *x * 2;
}

int main(void) {
    int x = 5;
    change_value(&x);
    printf("%d\n", x);

    return 0;
}
```

- Technically pass-reference-by-value but it's fine!

In the previous example, by passing the memory address, we can change the value *in place* and main will point to the updated value!

pointers and arrays 🍷

```
void double_array_of_ints(int data[], int size) {  
    for (int I = 0; I < size; I++)  
    {  
        data[i] = data[i] * 2;  
    }  
  
int main(void) {  
    int data[5] = {1, 2, 3, 4, 5};  
    double_array_of_ints(data, 5);  
    //is data doubled?  
}
```

^ does data in main contain the doubled values?

How?

Arrays decay to pointers

- Arrays point to the memory location which contains the first element
- As arrays are contiguous, we can then move through the memory sequentially to find the next values
- Very cool!

Feedback

<https://forms.office.com/r/K3PjvWebtD>


