



Lecture 8

Continuing on the memory train to
Pointers



YESTERDAY...

- Talked a bit more about libraries
- Went back to reinforce 1D arrays
- Looked at 2D arrays (which make up a grid and allow us to do some pretty cool stuff)



TODAY...

- Pointers ... (they point)
 - another type of variable that holds an address of a variable

WHERE IS THE CODE?

**LIVE LECTURE CODE CAN BE
FOUND HERE:**

<https://cgi.cse.unsw.edu.au/~cs1511/21T3/live/Week04/>

LET'S WELCOME POINTERS INTO THE MIX

- A pointer is another variable that stores a memory address of a variable
- This is very powerful, as it means you can modify things at the source (this also has certain implications for functions which we will look at in a bit)
- To declare a pointer, you specify what type the pointer points to with a star:

If your pointer points to an int:

```
int *pointer;
```

THERE ARE THREE PARTS TO A POINTER

```
#include <stdio.h>

int main (void) {

    //Declare a variable of type int, called box.
    //Assign value 6 to box
    int box = 6;
    //Declare a pointer variable that points to an int.
    //Assign the address of box to it
    int *box_ptr = &box;

    printf("The value of the variable 'box' located at address %p is %d\n"
        , box_ptr, *box_ptr);

    return 0;
}
```

1. Declare a pointer with a * - this is where you will specify what type the pointer points to

2. Initialise a pointer - assign the address to the variable with &

3. Dereference a pointer -Using a *, go to the address that this pointer variable is assigned and find what is at that

LET'S SEE THIS AS A WHOLE - WHAT HAPPENS?

LET'S SEE IT VISUALLY

```
//Declare a variable of type int, called box.  
//Assign value 6 to box  
int box = 6;
```

0xFF4C

0xFF48

0xFF44

0xFF40

box = 6;



THERE ARE THREE PARTS TO A POINTER

LET'S SEE IT VISUALLY

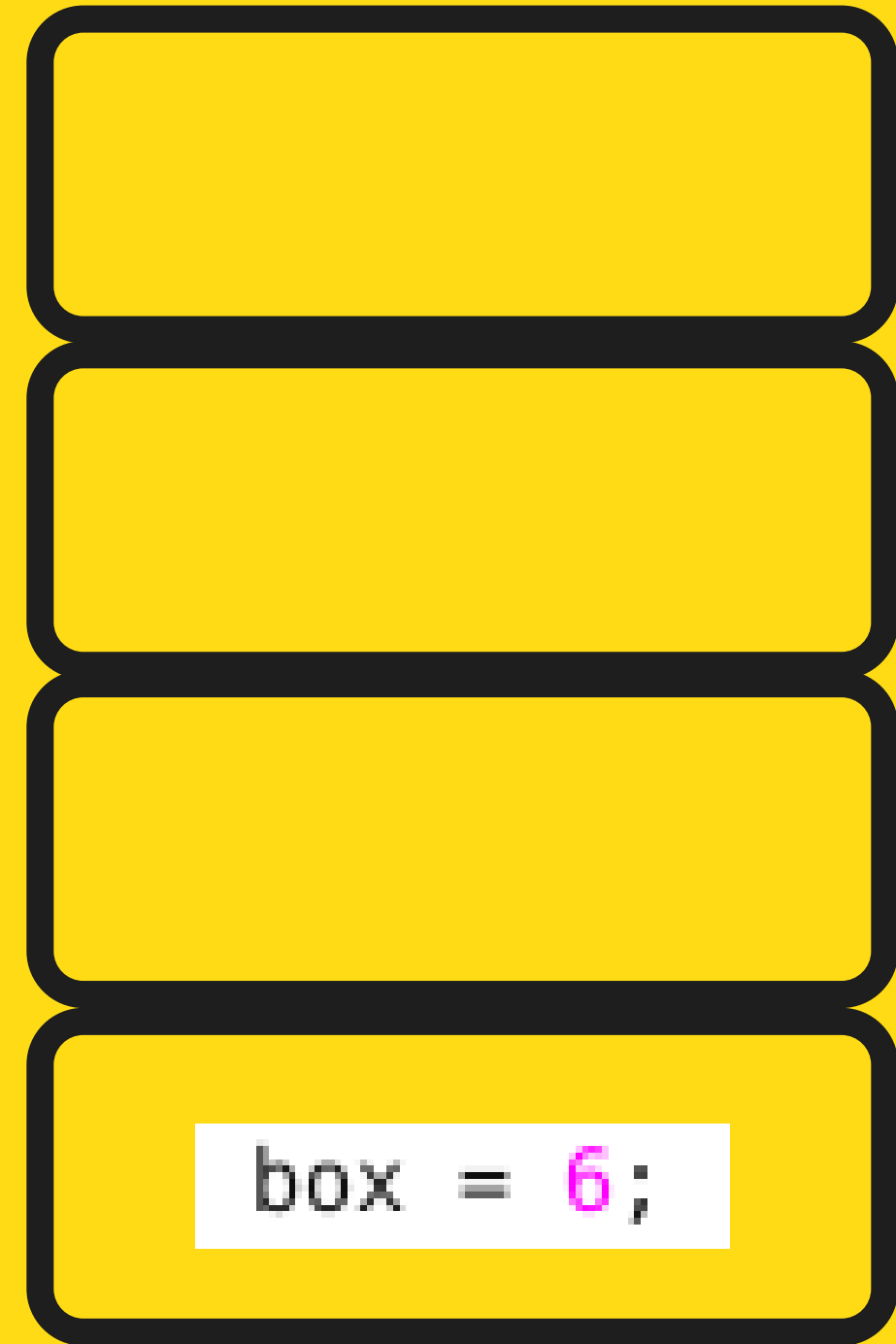
```
//Declare a pointer variable that points to an int.  
//Assign the address of box to it  
int *box_ptr = &box;
```

0xFF4C

0xFF48

0xFF44

0xFF40



`box = 6;`

THERE ARE THREE PARTS TO A POINTER

LET'S SEE IT VISUALLY

```
printf("The value of the variable 'box' located at address %p is %d\n", box_ptr, *box_ptr);
```

0xFF4C

0xFF48

0xFF44

0xFF40

box = 6;



YOU CAN HAVE A POINTER TO DIFFERENT VARIABLES

WHEN YOU DECLARE A POINTER, YOU WILL SPECIFY THE TYPE THAT IT POINTS TO FOLLOWED BY *

```
//Declare a variable of type int, called box.  
//Assign value 6 to box  
int box = 6;  
//Declare a pointer variable that points to an int.  
//Assign the address of box to it  
int *box_ptr = &box;
```

```
//Declare a variable of type double, called box.  
//Assign value 3.2 to box  
double box = 3.2;  
//Declare a pointer variable that points to a double.  
//Assign the address of box to it  
double *box_ptr = &box
```

```
//Declare a variable of type char, called box.  
//Assign value 'a' to box  
char box = 'a';  
//Declare a pointer variable that points to a char.  
//Assign the address of box to it  
char *box_ptr = &box
```

INITIALISING POINTERS WHEN YOU DON'T HAVE ANYTHING TO INITIALISE THEM WITH YET

NULL

- Pointers are just another type of variable, and just like our other variables it should be initialised after it is declared.
- Generally, we will initialise a pointer, by pointing it at a variable
- If we need to initialise a pointer that is not yet pointing to anything, we use: **NULL**
 - This is a special word in a C library which is #define
 - It is basically a value of 0, but for a pointer, we use this keyword **NULL**

WHAT HAPPENS IF YOU FORGET TO EVER GIVE THIS NULL POINTER AN ACTUAL ADDRESS WITH SOMETHING AND THEN TRY AND DEREference A NULL POINTER?

COMPILES, THEN CHAOS, CRASH.

```
#include <stdio.h>

int main (void) {

    //Declare a pointer variable that points to an int.
    //Assign NULL to it as it is not yet pointing to anything
    int *box_ptr = NULL;

    //Try to access the address of NULL.... CRASH
    printf("The value of the variable 'box' located at address %p is %d\n"
        , box_ptr, *box_ptr);

    return 0;
}
```

```
avas605@vx8:~/TestCode/Week04$ gcc -o pointers_intro pointers_intro.c
avas605@vx8:~/TestCode/Week04$ ./pointers_intro

pointers_intro.c:13:16: runtime error - accessing a value via a NULL pointer
gcc explanation: You are using a pointer which is NULL
A common error is accessing *p when p == NULL.

Execution stopped in main() in pointers_intro.c at line 13:

    //Declare a pointer variable that points to an int.
    //Assign NULL to it as it is not yet pointing to anything
    int *box_ptr = NULL;

    //Try to access the address of NULL.... CRASH
    printf("The value of the variable 'box' located at address %p is %d\n"
--> , box_ptr, *box_ptr);

    return 0;
}

Values when execution stopped:
box_ptr = NULL
```

LET'S DO A QUICK CODE DEMO OF IT

CODE, CODE, CODE!

```
011110110110111110111000001111000001000001111100101010000000110000011101010101
1110101110100111010000111111011011101101100111101101110000111101010101000001111
01110111100110111000101111100000101110110101011110000001001011010010100111110101
11110111111110110010010101100101010100111101010100100000001010100011111111010
110011010100111010101110100110100101001011100110011000010111101010001010100101
0111011001011011000000110101111010000100010011011111111101110000111101000011110
11111101001100011001110001000110111000010001111011000000001101101001000111000001
1100110110101111110100100100100101110000110011011100110011000111001100010100110
00101010101000000100110101000100010101101011000000101001010001100011001010010001
001001101011011001010101100110101000110111000000001101001001000100000011110011011
01010101100011011011101000101001111110001000000100010010110000101110001110001100
11110010101001011011110011000011110000000110010100111110101100101100001000001110
1000110101010000000000111111001000100000011110100011110100001010010011100011001
11100101111101110011001100001010110001010110001110101110000111001100110101000011
01000000101101101111100000000011111100011101001001111101110110011110000010101001
10010100101011111100011100101101001100000111011001000111000010111011111010110101
10100000011110101101100110000011111000010100001100001111000110000000000010000101
11010010001101100100011100110111010010000010111110010010011111011110111101100000
011101110000111010000000010010000010000010001001111111011111101101011001000010
1100011000101110100001100111101100010110101000000101011011010011101001000100100
11011001010000101111001000111001011110000010101101101001000110000010101010110010
01000010011010111001011111100110110110011011010011111111000100100101000000011000
00110100100110111011111010000001101011100101110101010100011010100010011001001101
11100111000101000000101001111101100101010001001100001101011110101110100000110111
11011111111010000100010001101011100001110001110101100110101010111001101110100111
01111001101111011100100001100111111010110011000010110111100000011100000010000000
11110111010011101000010111010111011011101000010000101001110011110111100100010011
11100011010000110001011010111101011001110111100010011010000100001100100010110110
0111010101111011101111111010
```

pointers_intro.c

BUT WHAT IS THE POINT OF POINTERS?

FUNCTIONS...

passing a pointer to a variable to a function

[change is made to the original variable, as we now know its address]

passing a variable to a function

[no change to original variable, because change is to the copy]

The diagram is split into two columns by a vertical dotted line. The left column is titled 'pass by reference' in red. It shows a blue 'cup' variable pointing to a blue cup icon. Below it, the function call 'fillCup()' is written in red. The right column is titled 'pass by value' in blue. It shows a blue 'cup' variable pointing to a blue cup icon. Below it, the function call 'fillCup()' is written in blue. At the bottom center, the URL 'www.penjee.com' is written in grey.

- Remember a week ago when I threw some easter eggs at you and told you when we pass something to a function, it makes a copy of it?
- Well this is where pointers come in, if we pass a pointer into a function instead, it can modify at the direct address of our variable...
- So if you pass a normal variable to a function, changing that variable in the function will have no effect on that variable in the main (because you are changing a copy)
- However, if you pass it a pointer, it can make changes directly that will also reflect back in the main function

[GIF source](#)

LET'S DO A QUICK CODE DEMO OF IT

CODE, CODE, CODE!

```
011110110110111110111000001111000001000001111100101010000000110000011101010101
1110101110100111010000111111011011101101100111101101110000111101010101000001111
01110111100110111000101111100000101110110101011110000001001011010010100111110101
1111011111111101100100101011001010101001111010101001000000010101000111111111010
110011010100111010101110100110100101001011100110011000010111101010001010100101
01110110010110110000001101011110100001000100110111111111101110000111101000011110
11111101001100011001110001000110111000010001111011000000001101101001000111000001
1100110110101111110100100100100101110000110011011100110011000111001100010100110
00101010101000000100110101000100010101101011000000101001010001100011001010010001
001001101011011001010101100110101000110111000000001101001001000100000011110011011
01010101100011011011101000101001111110001000000100010010110000101110001110001100
11110010101001011011110011000011110000000110010100111110101100101100001000001110
1000110101010000000000111111001000100000011110100011110100001010010011100011001
11100101111101110011001100001010110001010110001110101110000111001100110101000011
01000000101101101111100000000011111100011101001001111101110110011110000010101001
10010100101011111100011100101101001100000111011001000111000010111011111010110101
10100000011110101101100110000011111000010100001100001111000110000000000010000101
11010010001101100100011100110111010010000010111110010010011111011110111101100000
011101110000111010000000010010000010000010001001111111011111101101011001000010
1100011000101110100001100111101100010110101000000101011011010011101001000100100
11011001010000101111001000111001011110000010101101101001000110000010101010110010
01000010011010111001011111100110110110011011010011111111000100100101000000011000
00110100100110111011111010000001101011100101110101010100011010100010011001001101
11100111000101000000101001111101100101010001001100001101011110101110100000110111
11011111111010000100010001101011100001110001110101100110101010111001101110100111
01111001101111011100100001100111111010110011000010110111100000011100000010000000
11110111010011101000010111010111011011101000010000101001110011110111100100010011
11100011010000110001011010111101011001110111100010011010000100001100100010110110
01110101011110111011111111010
```

pointers_functions.c

POINTERS AND ARRAYS

IS AN ARRAY A POINTER?

```
1 #include <stdio.h>
2
3 int main (void) {
4
5     int array[4] = {0};
6
7     // Loop through the array and print out the address of each of
8     // the elements
9     int i = 0;
10    while (i < 4) {
11        printf("The address of the array[%d] is %p\n", i, &array[i]);
12        i++;
13    }
14    // Now notice that the address of the array is the same as of the
15    // first element in the array. Therefore, an array name is a
16    // constant pointer to the array - which is why we can input a
17    // whole array into a function just by giving the array name as input
18    printf("The address of the array name is %p\n", array);
19    return 0;
20 }
21 }
```

```
avas605@vx2:~/TestCode/Week04$ ./array
The address of the array[0] is 0x7ffe101864a0
The address of the array[1] is 0x7ffe101864a4
The address of the array[2] is 0x7ffe101864a8
The address of the array[3] is 0x7ffe101864ac
The address of the array name is 0x7ffe101864a0
```

- They are not the same
- An array is not a pointer – they are two different things!!
- However, an array name is a constant pointer to the array (the subtle differences!)
 - This means that the name of the array always points to the first element of the array.
 - This means that we can pass an array to a function just by giving it the whole array name only
- For example: array_pointer.c

BREAK TIME (5 MINUTES)

Sasha thinks of a number between 1 and 1,000 inclusive. Your job is ask her questions to discover what that number is. Sasha will always be truthful, to the best of her knowledge, but is only allowed to reply either "Yes", "No" or "I don't know." What is the fewest number of questions you need to ask Sasha in order to guarantee you will discover her number?

PROBLEM TIME

ARRAYS AND POINTERS AND FUNCTIONS - LET'S BRING IT ALL TOGETHER...



Let's see a good use of pointers. Now remember that you can only return one thing back to main and you can't return an array*

The problem is this:

Read in an array of numbers (user will specify how many numbers they plan to read in). Then the first number and the last number in the array will be swapped, and the modified array printed out again.

*So without using pointers, can you have a swapping function that swaps out two things? How would you return both of those things back to the main?

FEEDBACK?

**PLEASE LET ME KNOW ANY
FEEDBACK FROM TODAY'S
LECTURE!**

www.menti.com

Code: 88 99 69 6



WHAT DID WE LEARN TODAY?

POINTERS

pointers_intro.c
pointers_functions.c
array_pointer.c
the_shuffle.c

ANY QUESTIONS?

**DON'T FORGET YOU CAN
ALWAYS EMAIL US ON
CS1511@CSE.UNSW.EDU.AU
FOR ANY ADMIN QUESTIONS**

**PLEASE ASK IN THE FORUM
FOR CONTENT RELATED
QUESTIONS**

