COMP1511 PROGRAMMING FUNDAMENTALS

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

```c
#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*

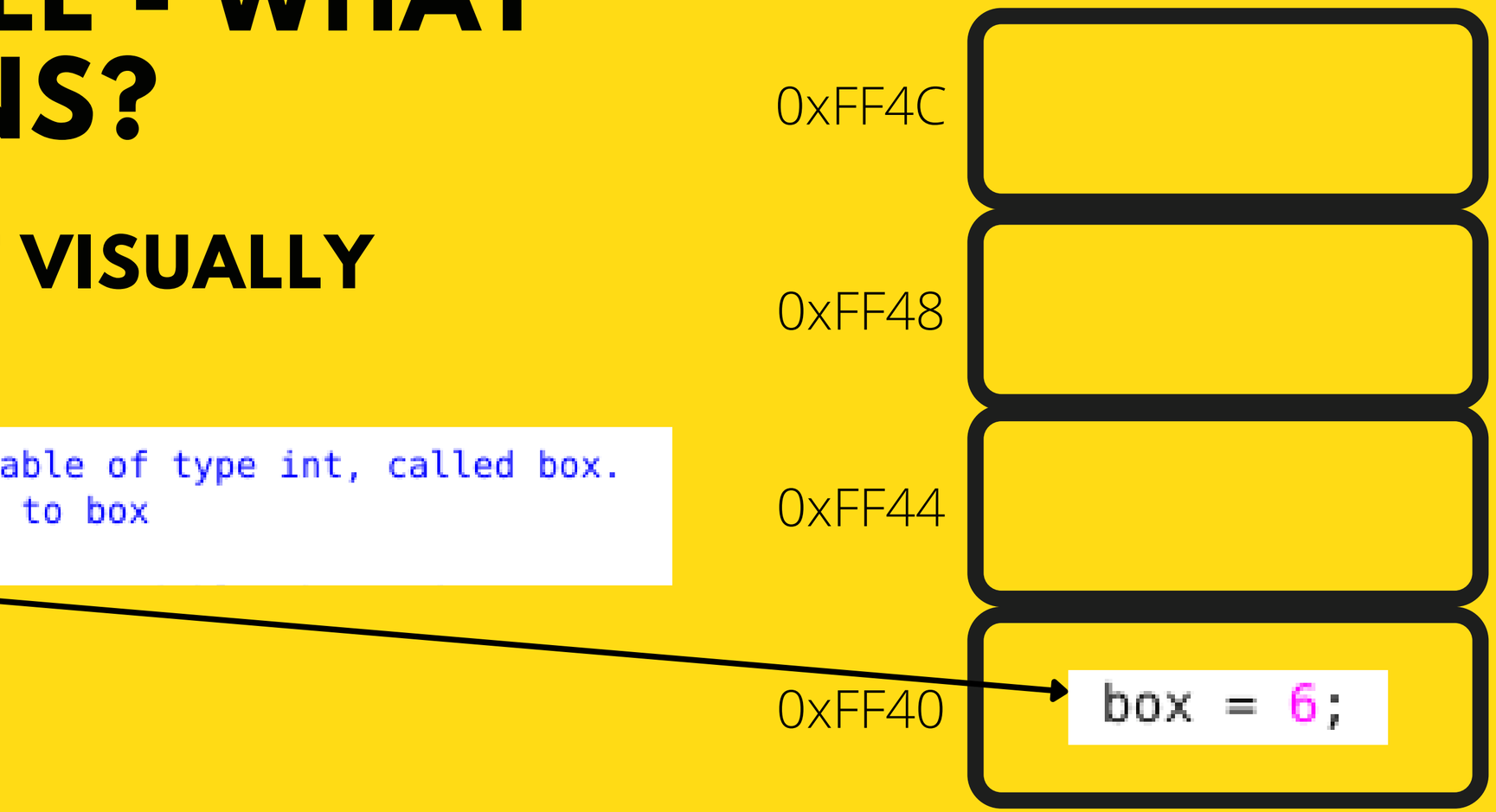
2. Initialise a pointer - assign the address to the variable with &

*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

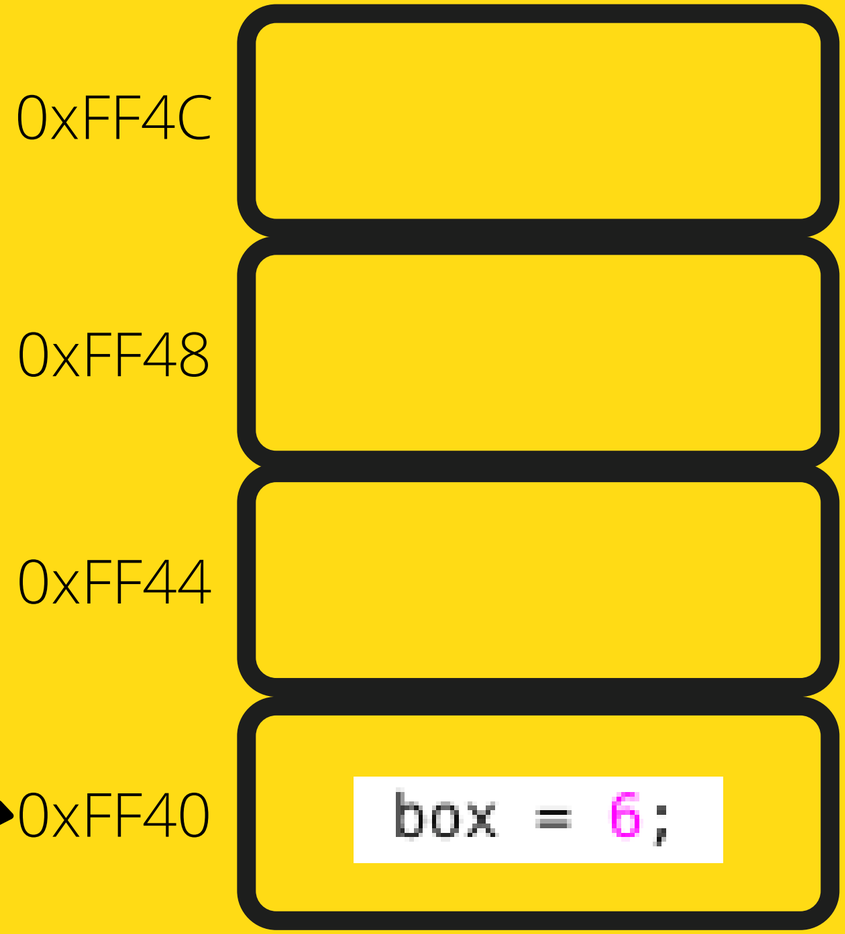
```
//Declare a variable of type int, called box.
//Assign value 6 to box
int box = 6;
```



THERE ARE THREE PARTS TO A POINTER

LET'S SEE IT VISUALLY

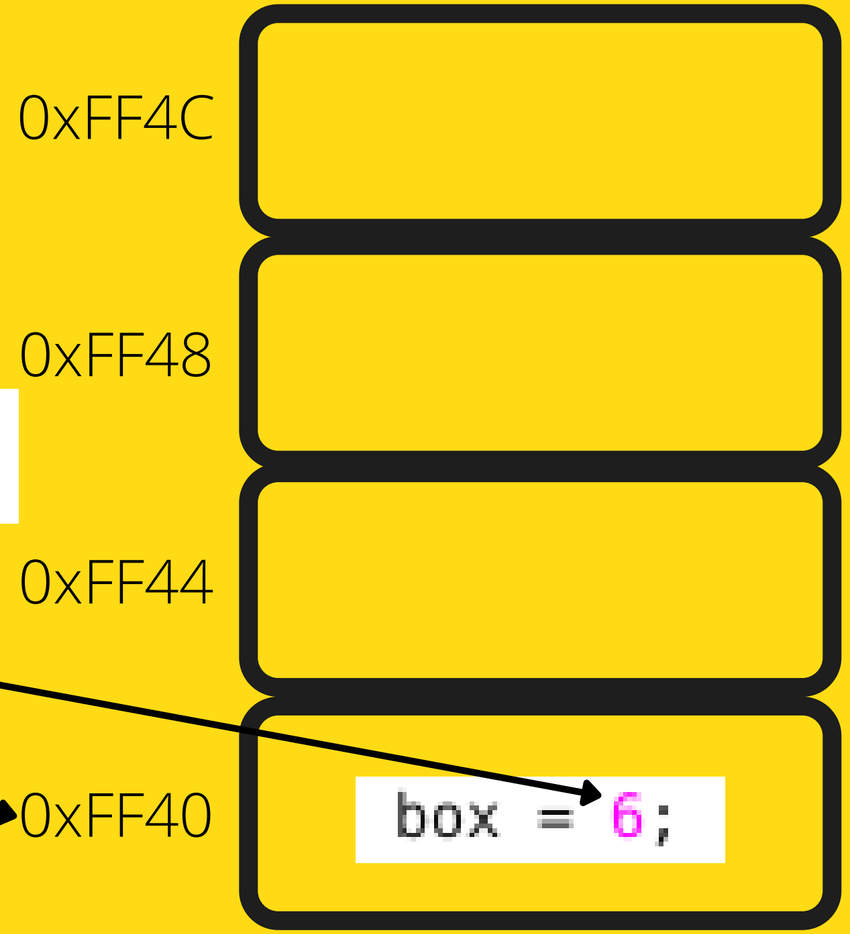
```
//Declare a pointer variable that points to an int.
//Assign the address of box to it
int *box_ptr = &box;
```



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);
```



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 WIHT 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.

```c
#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$ dcc -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

dcc 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!

pointers_intro.c

BUT WHAT IS THE POINT OF POINTERS?

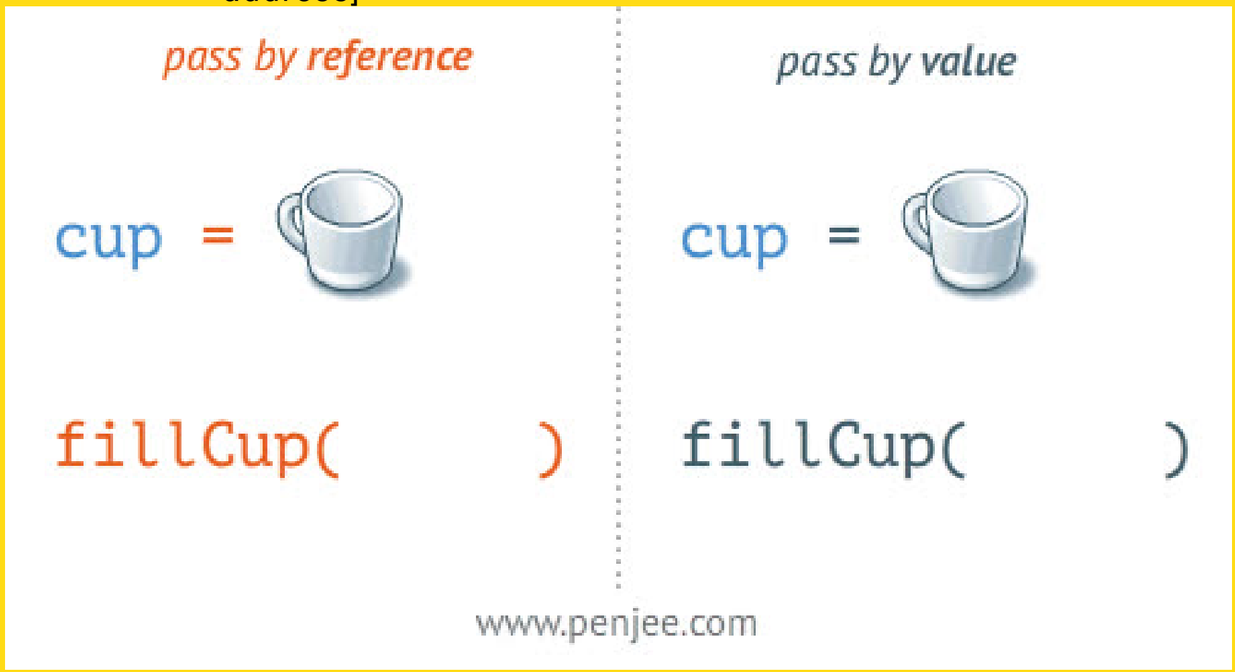
FUNCTIONS...

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]



- 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!

pointers_functions.c

POINTERS AND ARRAYS

IS AN ARRAY A POINTER?

```c
#include <stdio.h>

int main (void) {

    int array[4] = {0};

    // Loop through the array and print out the address of each of
    // the elements
    int i = 0;
    while (i < 4) {
        printf("The address of the array[%d] is %p\n", i, &array[i]);
        i++;
    }
    // Now notice that the address of the array is the same as of the
    // first element in the array. Therefore, an array name is a
    // constant pointer to the array - which is why we can input a
    // whole array into a function just by giving the array name as input
    printf("The address of the array name is %p\n", array);
    return 0;

}
```

```
ayas605@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...

the_shuffler.c

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