COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 8

Using dynamic arrays and introducing memory!

ON TUESDAY...

LAST LECTURE...

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

THIS LECTURE...

TODAY

- Revisiting scanf() and EOF
- Recap of 2D arrays and going back to the ice-cream hunt question (with diagrams this time!)
- Introducing 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/~CS1511/22T1/LIVE/WEEK04/

ARRAY OF ARRAYS

A RECAP

For example, let's say we declare an array of arrays:

```
int array[3][4];
```

Visually it looks like this and showing how to access each of the grid elements:

ARRAY OF ARRAYS

A RECAP: OUR PROBLEM FROM TUESDAY

`ice_cream_hunt.c`

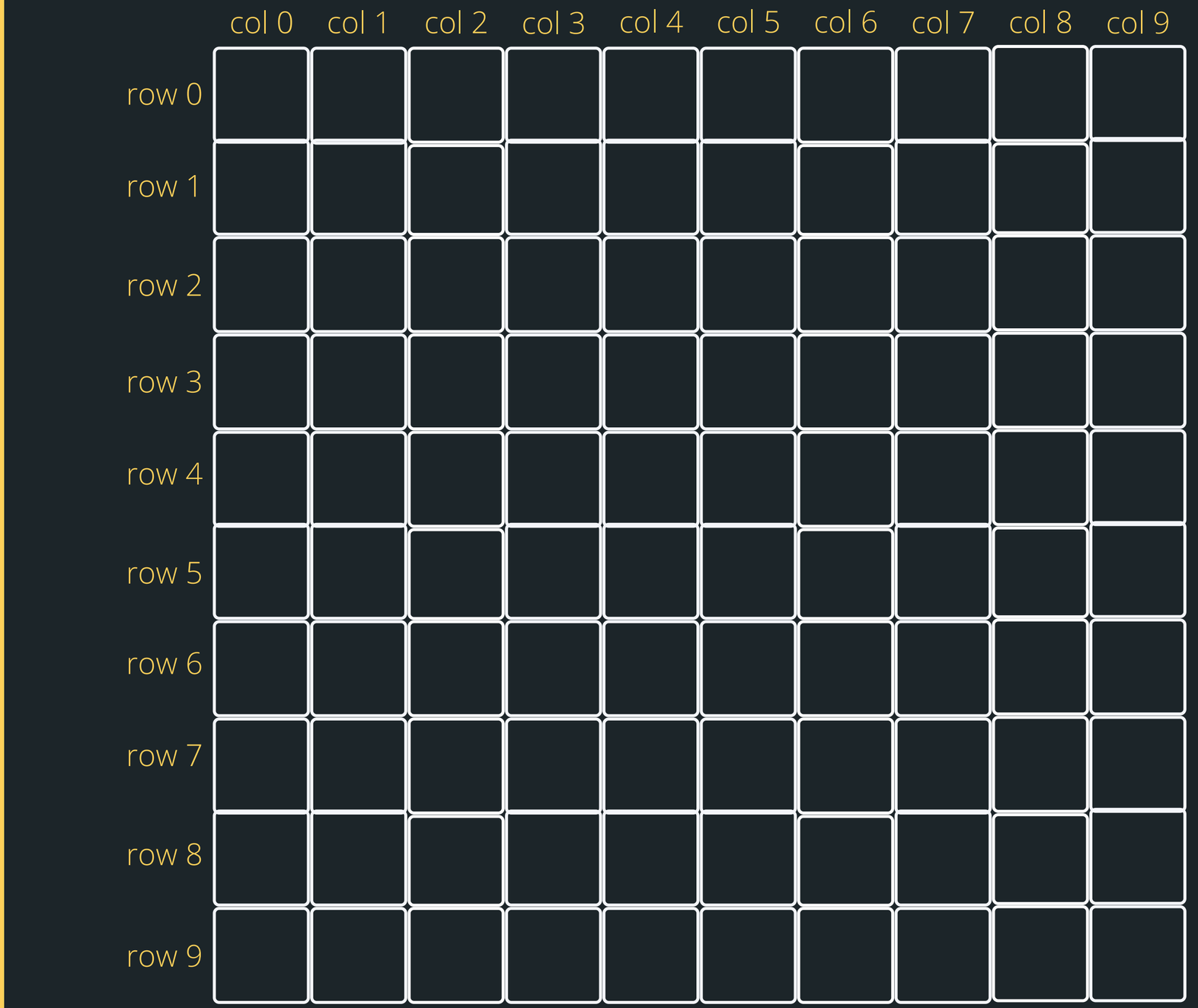
I am on the hunt for ice-cream (what else is new?). I I would like to explore a certain area in Kinsgsford to see if I can find ice-cream, on 10x10 grid. I am able to move around this section of Kingsford (left, right, up and down) and want to explore as many places in this section as possible, so as not to miss an ice-cream opportunity. I always start in the bottom right corner. Once I have explored the section in Kingsford, or I go to the same place more than once - it will be time to go home.

We can make this problem more complex if we have time with either Tom or Tammy deciding to join me on the hunt!

ARRAY OF ARRAYS

LET'S GO BACK TO OUR PROBLEM FROM TUESDAY AND DIAGRAM IT UP

`ice_cream_hunt.c`



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 an asterisk:

```
type_pointing_to *name_of_ variable;
```

  - For example, if your pointer points to an int:

```
int *pointer;
```

WHY DO WE NEED POINTERS?

- Pointers solve two common problems:
  - Remember how I said that when we pass some inputs into a function it actually makes a copy of that variable? Well, pointers kind of allow us to share information easier between sections of code without all that copying
  - Pointers also allow us to play with more complex data structures such as linked lists - coming in Week 7 and will really help with pointers :)

THERE ARE THREE PARTS TO A POINTER

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

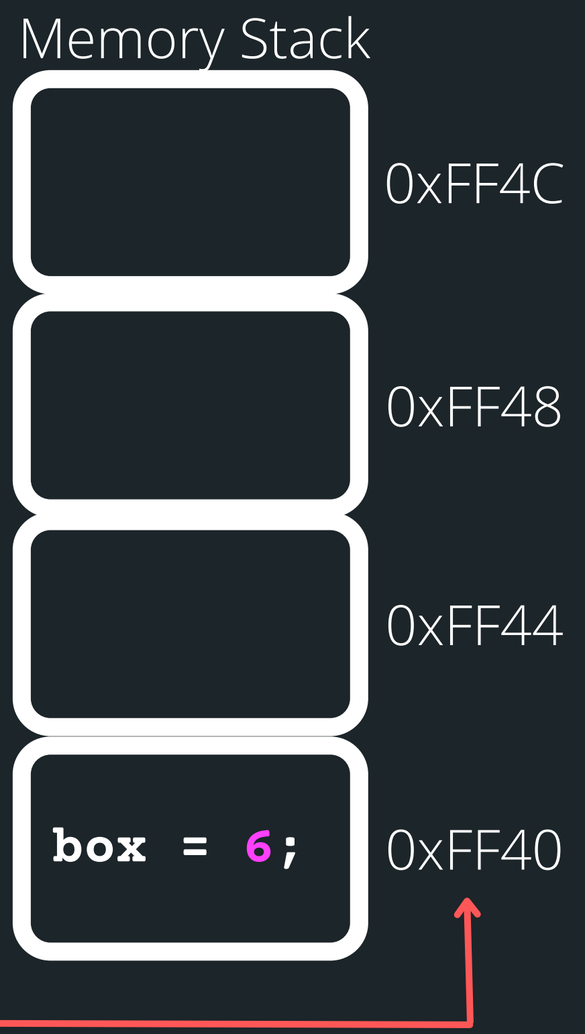
2. Initialise a pointer - assign the address to the variable with &

```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;
}
```

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

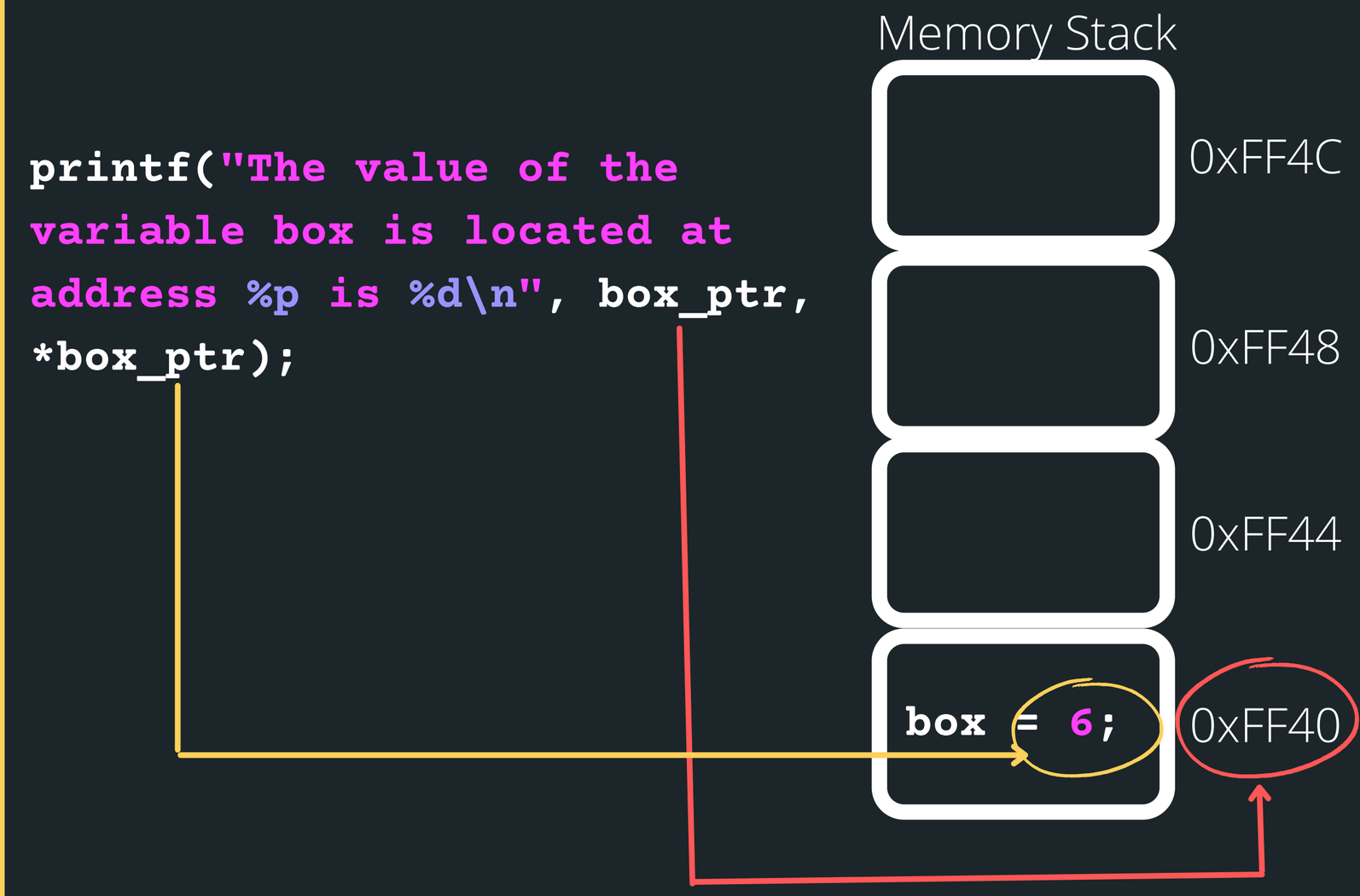
VISUALLY WHAT IS HAPPENING?

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

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



VISUALLY WHAT IS HAPPENING?

```
printf("The value of the variable box is 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 the value 6 to box
int box = 6;

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

// Declare a variable of type double called box
// Assign the value 3.2 to box
double box = 3.2;

// Declare a pointer variable that points to
// a double and assign the address of box to it
double *box_ptr = &box;

// Declare a variable of type char called box
// Assign the value 'c' to box
char box = 'c';

// Declare a pointer variable that points to
// a double and 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 POINTER

- 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 THAN CHAOS...

```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
```

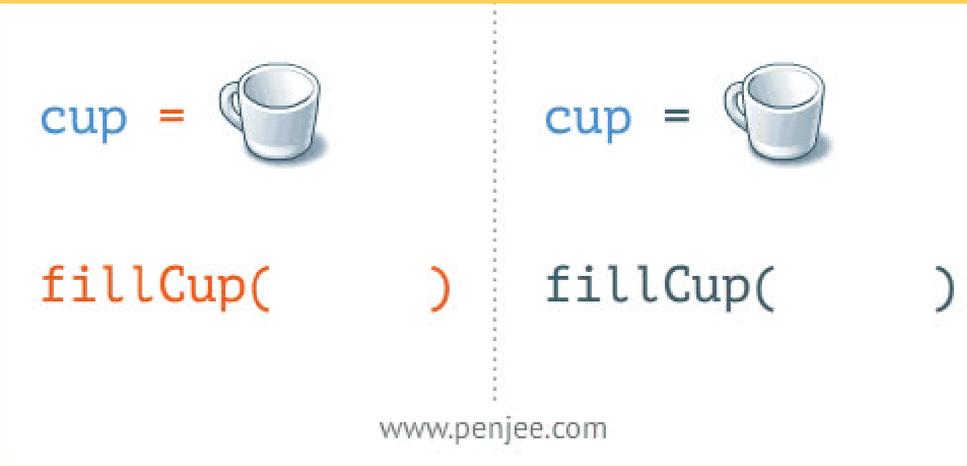
TIME TO CODE AND SEE A POINTER IN ACTION!

`pointers_intro.c`

- Best way to learn about pointers is to start using them

POINTERS AND FUNCTIONS

pointers_function.c



- This is one of the benefits of pointers (yes, you don't have to use them, but it makes your solutions so much easier and more elegant!)
    - If we pass a pointer into a function instead of a normal variable, it can modify at the direct address of our variable (no need to return the result, can modify multiple things if needed etc)...
- 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

POINTERS AND ARRAYS

IS AN ARRAY A POINTER?

`array_pointer.c`

- 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!)
    - You may have heard the term that the array decays to a pointer to the first element of the array by the compiler
- 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

POINTERS AND ARRAYS

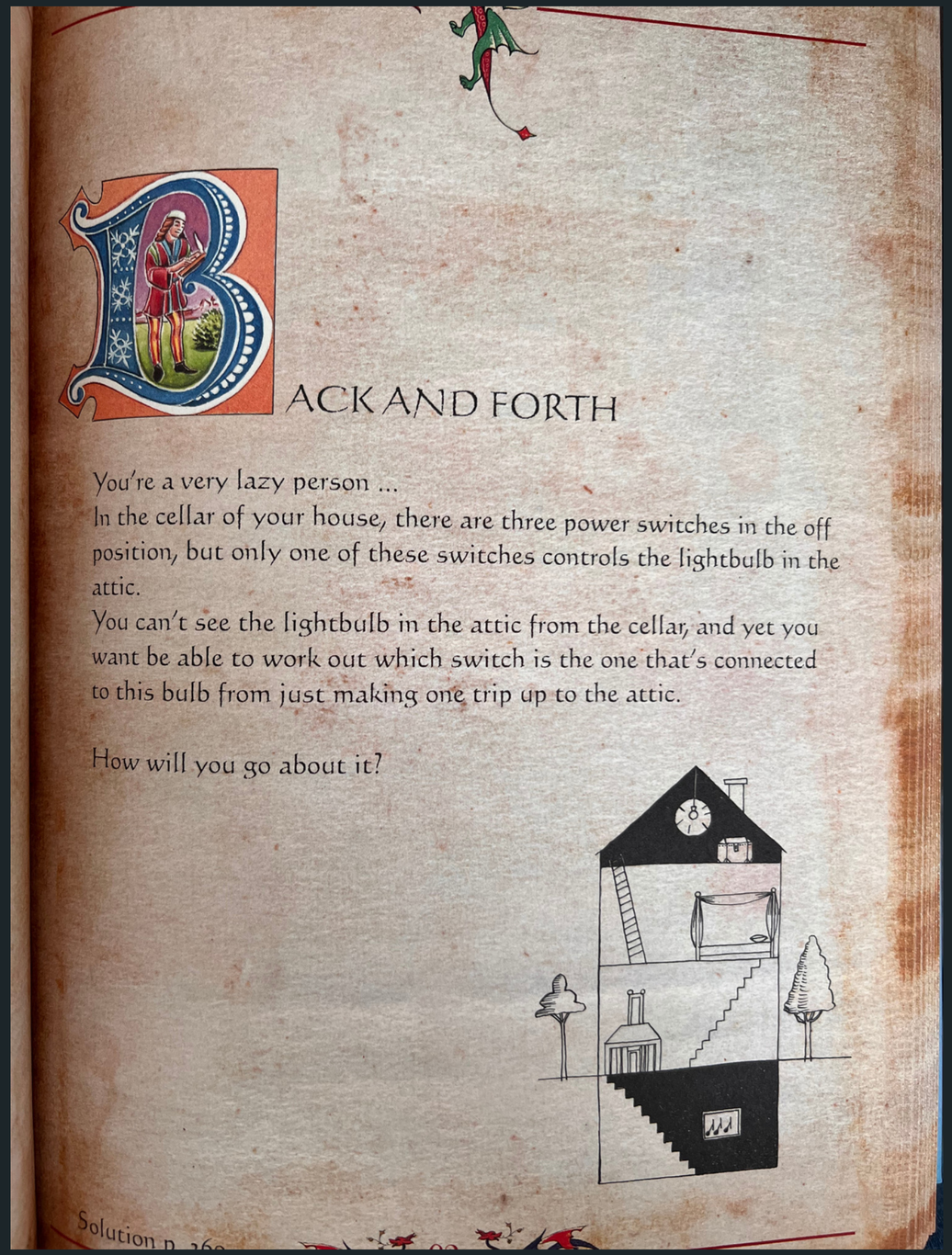
IS AN ARRAY A POINTER?

array_pointer.c

```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
```

BREAK TIME



BACK AND FORTH

You're a very lazy person ...
In the cellar of your house, there are three power switches in the off position, but only one of these switches controls the lightbulb in the attic.
You can't see the lightbulb in the attic from the cellar, and yet you want be able to work out which switch is the one that's connected to this bulb from just making one trip up to the attic.

How will you go about it?

Solution p. 360

CODE CODE CODE

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

`shufflin.c`

- Let’s see and use some 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!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://www.menti.com/sy4jf115wu

WHAT DID WE LEARN TODAY?

2D ARRAY RECAP

ice_cream_hunt.c

INTRO TO POINTERS

pointers_intro.c

pointers_functions.c

array_pointer.c

the_shuffle.c

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@cse.unsw.edu.au