Week 10, Lab 10	Weight: 1%	Due: End of your stream's week 10 lab session (via sync)

Pre-lab Preparation:

- **Week 1, 2, 3, 4, 5, 6, 7, 8, 9 Lectures, Week 1, 2, 3, 4, 5, 6, 7, 8, 9 Labs**
- **Week 10, Lecture 024**

Lab Activities:

Remember to **sync** to obtain the lab starting code.

As you work through the lab exercises, if you encounter programming terminology that you do not understand:

- Refer to the Programming 1 lecture materials.
- Ask a Lab TA for clarification on what the term means.
- After attempting an activity, if you are stuck for an unreasonable amount of time, seek help from a Lab TA! Do not wait too long to seek help!

Exercise 1: Declaring and using a Struct

What is a **`struct`***?*

A structure can be used to group different types of data elements together into something meaningful.

Create a structure call Cat, which will store details about a cat. All cats have an age, weight, tail length, and name. Choose appropriate types for these members.

What type should **`age`** *be and why?*

What type should **`weight`** *be and why?*

What type should **`tail_length`** *be and why?*

What type should **`name`** *be and why?*

Declare the **`Cat`** stucture in the file l**`ab10ex01.c`**.

In the **`main`** function, create a cat.

Read the following paragraph, and given the details within about a particular cat, assign to the members the appropriate values.

> "My pet cat, Whiskers, is seven years old. He is orange, and has a fifteen centimeter long tail. I feed Whiskers twice daily, and he also chases wildlife. Currently he is eight kilograms."

Once setup, use **`printf()`** to access the members and send the details of the cat to the console. Output should take the following form:

```
A cat...
Name:        ?
Age:         ?
Weight:      ?
Tail Length: ?
```

Exercise 2: Structures, pass by value vs pass by reference

Add the following structure to **lab10ex02.c**:

```
struct Thing
{
    int a;
    short b;
    char c;
    double d;
    long e;
    float f;
};
```

Declare a **struct** variable of type **Thing** in the **main**, print out its address. Assign test values to each of the members in **Thing**. Then print out the addresses *and* contents of each member in **Thing**.

Add a function to **lab10ex02.c** declared as follows:

```
void pass_by_value(struct Thing input)
{
    // Insert code here...
}
```

Inside the **pass_by_value** function print out the address of the **input** structure. Print out the addresses and contents of each member.

Call **pass_by_value** from **main**, passing in the stucture locally declared in the **main**.

Are the addresses printed from **main** *and* **pass_by_value** *the same, or different, and why?*

Next, add the following function to **lab10ex02.c** declared as follows:

```
void pass_by_reference(struct Thing* p_input)
{
    // Insert code here...
}
```

Inside the **`pass_by_reference`** function print out the address of the **`p_input`** structure (not the address of the address!). Print out the addresses and contents of each member in the structure.

Call **`pass_by_reference`** from **`main`**, passing in the stucture locally declared in the **`main`**.

Are the addresses printed from **`main`** *and* **`pass_by_reference`** *the same, or different, and why?*

Next, in both **`pass_by_value`**, and **`pass_by_reference`**, increment the new of the member **`b`**.

After the calls to **`pass_by_value`** and **`pass_by_reference`** in the **`main`** function, print out the addresses and contents of the **`Thing`** structure's members.

After **`main`** *calls* **`pass_by_value`** *is the value held in* **b** *changed by* **`pass_by_value`***?*

After **`main`** *calls* **`pass_by_reference`** *is the value held in* **b** *changed by* **`pass_by_reference`***?*

Have a lab TA review your completed exercises 1 and 2 for this lab session. See the end of this document for the review questions

Exercise 3: Creating a structure on the heap

In **lab10ex03.c**, write a function called **card_factory**, which takes no parameters and returns a pointer to a **Playing_Card** structure.

The Playing_Card structure is to be declared as follows:

```
struct Playing_Card
{
    int rank;
    int suit;
};
```

Inside the factory function, allocate a new **Playing_Card** struct on the heap. Populate the **rank** member with a random value of between 1 and 13 inclusive. Populate the **suit** member with a random value of between 0 and 3 inclusive. Return this new **Playing_Card** structure to the caller.

Why must the structure be allocated on the heap, rather than the stack from of the **factory** *function?*

From the main function call **card_factory** five times. Store the result of each call in an array of **Playing_Card** pointers.

Write another method called **print_playing_card** which takes in a pointer to a **Playing_Card**, and prints as follows:

```
Printing playing card:
Rank: Queen
Suit: Clubs
```

Where suits are represented as follows: 0 = diamonds, 1 = hearts, 2 = clubs, 3 = spades.

And the ranks as follows: 1 = Ace, 11 = Jack, 12 = Queen, 13 = King, 2 to 10 are the values 2 to 10.

Ensure before **`main()`** exits, that it cleans up the memory allocations by freeing the appropriate heap allocations.

What happens if heap allocations are not freed?

Exercise 4: Using structures

In **`lab10ex04.c`**, declare a struct called **`Point3d`** which represents a point in three-dimensions. 3d points are three-tuples, with a real numbers for the x, y, and z values. Choose appropriate types and names for these members.

Next, allocate two **`Point3d`** structures inside the main function.

Assign them the test values of $p_1 = < 1, 2, 3 >$ and $p_2 = < 4, 5, 6 >$. To do this you will need to access each member using the dot operator, and assign the appropraite value to the appopriate member.

Write a function called **`distance3d`**, which takes as input two **`Point3d`** structures. Inside this function, calculate the distance between the two points. Return the distance.

Distance between two points in 3d is similar to distance between two points in 2d. Pythagoras' Theorem can be used to find this distance. You will need to use the **`math.h`** library to access the square root function. See the **`man sqrtf`** page.

$$\text{Distance} = \text{sqrt}((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2)$$

Where: $p_1 = < x_1, y_1, z_1 >$

$p_2 = < x_2, y_2, z_2 >$

Exercise 5: The sizeof a structure

Examine the structure below:

```
struct Person
{
    char first_initial;
    char last_initial;
    int age;
    int weight;
    char sex;
    int height;
    char blood_type;
    int shoe_type;
};
```

How many bytes does the **`Person`** *struct take up in memory?*

In **`lab10ex05.c`**, declare the student struct as above. Use **`sizeof`** to confirm its size.

How many bytes does the **`sizeof`** *return?*

Next, create a struct called **`OptimisedPerson`**, which has the same members, but ordered such that it creates a smaller memory footprint. Declare this in **`lab10ex05.c`**, and check its size using sizeof.

How many bytes does the **`OptimisedPerson`** *struct take up in memory?*

Week 10, Lab 10 Submission:

Run the **sync** command to submit your completed lab work.

Shutdown your Raspberry Pi by pressing **ALT-CTRL-DEL**. Power-down and pack up your Raspberry Pi kit.

Marking Criteria:

Have you completed each of the following? Have you submitted your code from lab?

Marking Criteria:	Week 10 Lab 10 Weight 1%	Yes	No
Ex 1:	Code written correctly?		
	Correctly named file with right contents?		
	Questions answered correctly on handout?		
Ex 2:	Correct output from program?		
	Questions answered correctly on handout?		
Ex 3:	Dynamic memory functions correctly used to allocate and free memory for card		
	Reliably generates and prints out cards at random		
Ex 4:	Correct result generated by correct algorithm		
Ex 5:	Person optimised.		

Next activity: Homework 8 and Final Week 10 Lecture