ENSF 337: Programming Fundamentals for Software and Computer
Lab-5 – Thursday October 11, 2018

Department of Electrical & Computer Engineering
University of Calgary

M. Moussavi, PhD, PEng.

Objectives:

This lab consists of several exercises, mostly designed helping you to understand the concept of C `struct` type.

Due Date:

In-lab exercises that must be always handed in on paper **by the end of your scheduled lab period**, in the hand-in boxes for your section. The hand-in boxes are on the second floor of the ICT building, in the hallway on the west side of the building.

When you hand in your in-lab exercises, make sure that course name (ENSF 337) and your name and lab section are written in a clear and easy-to-spot way on the front page| it is a waste of time for both you and your TAs to deal with an assignment that doesn't have a name on it. Also make sure that your pages are stapled together securely |pages held together with paperclips or folds of paper tend to fall apart.

Post-lab exercises must be submitted electronically using the D2L Dropbox feature. All of your work should be in a single PDF file that is easy for your TA to read and mark.

Due dates and times for post-lab exercises are: Thursday October 18, before 2:00 PM

Important Notes:

- Some post-lab exercises may ask you to draw a diagram, that most of the students prefer to hand-draw them. In these cases you need to scan your diagram with an appropriate device and insert the scanned picture of your diagram into your PDF file (the post-lab report).

- 20% marks will be deducted from the assignments handed in up to 24 hours after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

Marking scheme:

The total mark for the exercises in this lab is 40 **marks**

- Exercise A (in-lab): 10 marks
- Exercise B (in- lab): 8 marks
- Exercise C: 2 marks
- Exercise D: 10 marks
- Exercise E: 10 marks

Exercise A (10 marks): C `struct` Objects on the Computer Memory

This is an In-lab exercise

Read This First - Structures on the Memory

A structure type in C is a type that specifies the format of a record with one or more members, where each member has a specified name and type. These members are stored on memory in the order that they are declared in the definition of the structure, and the address of the first member is identical to the address of the structure object itself. For example, if we consider the following definition for structure `course`:

```
struct course{
      char code[5];
      int number;
      char year[4];
};
```

And, the following declaration of an instance of `struct course`:

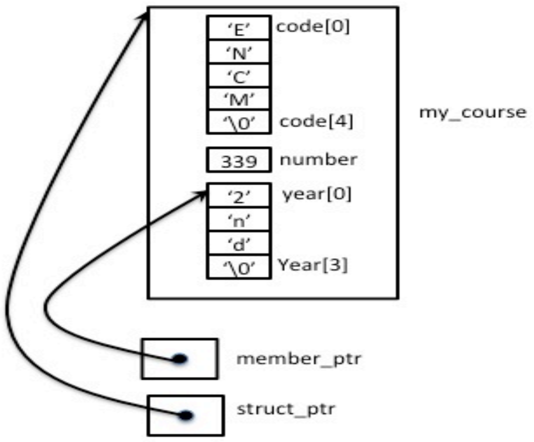
```
struct course my_course = {"ENCM", 339, "2nd"};
```

The address of member `my_course.code` is identical to the address of `my_course`, and the address of member `my_course.number` is greater than the address of the previous member, `my_course.code.` However, the address of the member `my_course.number` will not be necessarily the address of the following byte right after the end of memory space allocated for the member `my_course.code.` It means there might be gaps, or unused bytes between the members. The compiler may align the members of a structure for certain kind of the addresses, such as 32-bit boundaries, to ensure fast access to the members. As a result, the size of an instance of structure such as `course` is not necessarily equal to the sum of the size of the members; it might be greater.

Read This Second – Structures and Pointers

In principle, a pointer to a C structure type is not much different from other types of pointers. They are basically supposed to hold the address of a `struct` object and they are of the same size as other pointers. Please notice, when drawing AR diagrams make sure to be clear whether the arrowhead points to the entire struct instance or to a member of the structure. Please see the following example:

```
      struct course* struct_ptr = & my_course;
      char* member_ptr = mty_course.year;
```



What To Do

Download the file `lab5exA.c` , and `lab5_point.h` from D2L. Read the program carefully, and try to predict the output of the program. **Note: when you compile the program, some compilers may display a warning. For this exercise you can ignore this warning.** Now, run the program to compare the results with your prediction. Then, draw an AR diagram for point **one**. Your diagram doesn't need to show the string constants used within printf functions, on the static storage.

What To Submit:

Submit your AR diagram as part of your in-lab report.

Exercise B (8 marks): Nested Structure

This is an in lab exercise.

What to Do:

Download file `lab5axB.c` and `lab5exB.h` from D2L. In the file `lab5exB.h` there are two structures called Time and Date and a third structure called Timestamp that nests the other two structures. Study the files to understand what the program does. Then draw memory diagrams for point one in the file `lab5exB.c`.

What to Submit:

Submit your AR diagrams as part of you in-lab report.

Exercise C (2 marks): A Simple Macro:

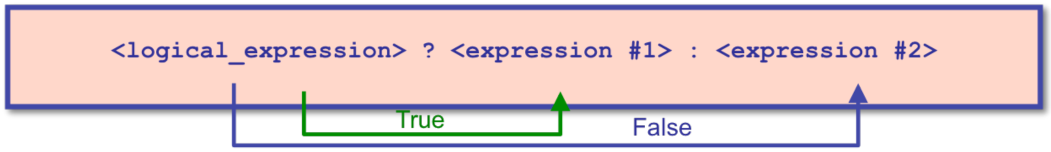
This is a post-lab exercise

Read This First: What is the C Conditional Operator

C is a programming language rich in variety of useful operators. One of the operators that can be very useful in problem solving and in particular when defining macros is called the conditional-operator that consists of two symbols `:` `?` (a question mark) and : (colon). This operator works more or less like a simple if … else statement.

The following figure shows the general format of the conditional operator:

```
–  If logical expression is true the value after question mark will be returned.
   Otherwise the value after the colon will be returned
```



Example: The following example shows how the conditional operator works and how it is comparable with an if-else statement.

If … else version	Conditional operator version

`int x = 8, y = 20;`
`int z;`
`if(x > y)`
`  z = x;`
	`int x = 8, y = 20;`
	`int z;`
	`z = (x > y) ? x : y;`

```
else
  z = y;

print("larger value is: %d", z);
// Prints: larger value is 20
```

```
print("larger value is: %d", z);
// Prints: larger value is 20
```

What To Do:

Download the file `lab5exC.c` from D2L. If you try to compile this file you will get an error because the definition of macro `LARGEST_OF_THREE`, is missing.

Your task in this exercise is to write the macro `LARGEST_OF_THREE`. This macro should return the largest value among its three numeric parameters. Here is an example of calling this macro that should return `0.999`.

```c
#include <stdio.h>
int main(void)
{
    double x = 0.300, y = 0.500, z= 0.999;
    double largest = LARGEST_OF_THREE(x, y, z);
    printf("the largest value is %f", largest);
    return 0;
}
```

What To Submit:

Submit the definition of your macro (the printout or handwritten), as part of your post-lab report (PDF format).

Exercise D (10 marks): Writing Functions that Use C struct

This is a post-lab exercise

What to Do:

Step 1: Download file `lab5exD.c` and `lab5exD.h` from D2L, and change the definition of `struct point` to a three dimensional point, by adding the third coordinate of type `double`, called `z`.

Step 2: change the first two lines in the main function to assign values for z-coordinate of struct instances `alpha` and `beta` to 56.0 and 97.0, respectively.

Step 3: modify the definition of any function, if needed, so that they all work for a three-D point.

Step 5: Complete the missing code in functions `distance, mid_point`, and `swap`.

For your information: The distance, `d`, between two three-D point `a(x1, y1, z1)` and point `b(x2, y2, z2)` can be calculated by:

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

What To Submit:

Submit your source code and your program output as part of your post-lab report (PDF).

Exercise E (10 marks): Using Array of Structures

What to Do:

Download file `lab5exE.c` and `lab5exE.h` from D2L. In this exercise an array of `Point` with 10 elements is created and filled with some sort of random values.

A sample-run of the program shows that the ten elements of `struct_array`, were filled with the following values for `<x, y, z>`. Also, points have labels such as `A9, z9, B7`, and so on.

```
Array of Points contains:
struct_array[0]: A9 <700.00, 840.00, 1050.00>
struct_array[1]: z8 <300.00, 360.00, 450.00>
struct_array[2]: B7 <999.00, 1200.00, 1500.00>
struct_array[3]: y6 <599.00, 719.00, 900.00>
struct_array[4]: C5 <198.00, 239.00, 299.00>
struct_array[5]: x4 <898.00, 1079.00, 1349.00>
struct_array[6]: D3 <497.00, 598.00, 749.00>
struct_array[7]: w2 <97.00, 118.00, 149.00>
struct_array[8]: E1 <796.00, 958.00, 1198.00>
struct_array[9]: v0 <396.00, 477.00, 598.00>
```

Now your job is to write the definition of the functions `search,` and `reverse,` based on the function interfaced comments given in the file `lab5ExF.h`.

What To Submit:

Submit your source code and your program output as part of your post-lab report (PDF).