ENSF 337: Programming Fundamentals for Software and Computer
Lab-5, Fall 2022

Department of Electrical & Software Engineering
University of Calgary

M. Moussavi, PhD, PEng.

Objectives:

This lab consists of several exercises, mostly designed helping you to understand the concept of dynamic allocation of memory and C struct type.

Due Dates:

B01	**Wed Oct 19, 3:00 PM**
B02	**Fri Oct 21, 9:00 AM**

Late Due Date:

20% marks will be deducted from the assignments handed in up to 24 hours after each due date. That 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.

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

Marking scheme:
The total mark for the exercises in this lab is 42 **marks**
- Exercise A: 12 marks
- Exercise B: 4 marks
- Exercise C: 8 marks
- Exercise D: 6 marks
- Exercise E: 6 marks
- Exercise F: 6 marks

Exercise A (12 marks) - Different kinds of allocation

Read This First

This exercise is a test of your understanding of the different rules for automatic, static, and dynamic allocation.

What To Do

Download file `allocation.c.` Read the code, then make AR diagrams for point one and for both times point two is reached.

Exercise B (4 marks): Fixing a Defective C Program

Read This First – Common Mistake with Dynamic Allocation of Memory

Developing programs that use dynamic allocation of memory needs special attention and additional care to avoid runtime errors. Although dynamic allocation of memory may provide some flexibility to write programs that manage their required memory spaces more efficiently, but on the other hands it puts more responsibilities on the programmer's shoulder to avoid malicious errors.

Here is the list of some of the common and possible mistakes that programmers can make when allocating memory dynamically in a C program:

1. Losing the track of an allocated memory, which results in memory leak.
2. Allocating memory in a function that doesn't return a reference to the calling function.
3. Loosing track of the begging of the allocated memory.
4. Trying to free a portion of the allocated memory. For example, starting to release memory from third element of a dynamically allocated block to the end of the block.

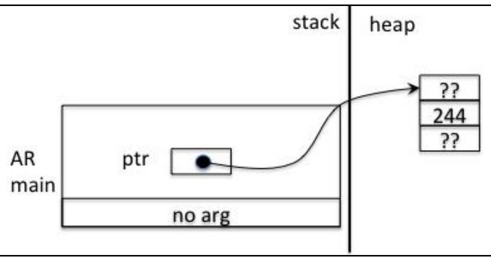
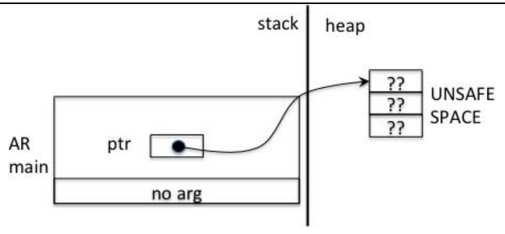
Note: Please notice that even dynamic allocation of memory in a program may allow more efficient use of memory spaces, but on the other hand it will involve addition overhead for extra time for the operations such as searching for available space, etc. This may not be a major concern for many programs, as most of the modern computers have very fast processors hence the overhead time may not be an issue.

Also Important to Notice:

- You can only use C library function **free** to deallocate a block of memory that is allocated by one of the C library functions: malloc, calloc or realloc.
- Pointer that is passed to the function free() MUST be pointing to the first byte of the allocated memory, otherwise it causes "*undefined behavior*".
- Notice that function free does not change the of value the pointer that is passed to it. In other words the pointer that is passed to free, still points to the same (now invalid) location.

For the purpose of AR diagrams in ENSF 337, the de-allocated memory must be labeled with "UNSAFE SPACE" phrase. Please see the following code and diagrams at points one and two:

```
int main(void) {
   int* ptr = malloc (3 * sizeof(int));
   ptr[1] = 244;
   // Point 1
   free(ptr);
   // Point 2
   return 0;
}
```

Point 1	**Point 2**
	

What to Do:

Download file `Lab5exe_B.c` from D2L and study the code to find out how the program works. If you compile and run the program, you will notice that it doesn't do anything useful -- only prints the following messages:

```
Program started...
Program terminated...
```

If you fix the defective parts of the code, which needs some changes to the function `create_array` and some changes to the `main` function, the program output should be as follows:

```
Program started...
100.000000
200.000000
300.000000
400.000000
500.000000
600.000000
700.000000
800.000000
900.000000
1000.000000
Program terminated...
```

Exercise C (8 marks): C `struct` Objects on the Computer Memory

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`:

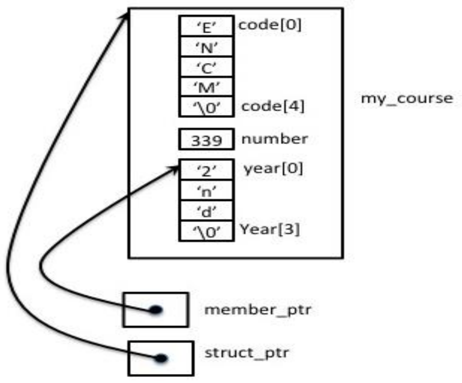
```
struct course my_course = {"ENCM", 339, "2nd"};
```

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 `lab5exC.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 lab report.

Exercise D (6 marks): Nested Structure

Download file `lab5exD.c` and `lab5exD.h` from D2L. In the file `lab5exD.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 `lab5exD.c`.

What to Submit:

Submit your AR diagrams as part of your lab report.

Exercise E (6 marks): Writing Functions that Use C struct

What to Do:

Step 1: Download file `lab5exE.c` and `lab5exE.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 for 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 lab report (PDF).

Exercise F (6 marks): Using Array of Structures

What to Do:

Download file `lab5exF.c` and `lab5exF.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 interface comments given in the file `lab5ExD.h`.

What to Submit:

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