Laboratory Exercise #10

Reading

- Read Section 5.1 of Paul Carter's PC Assembly Book

Practice Exercise:

- Execute “array1.asm” and interface it with “array1c.c”.

```
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/linux-ex$ nasm -f elf array1.asm
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/linux-ex$ gcc -m32 -o array1 array1c.c array1.o asm_io.o
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/linux-ex$ ./array1
First 10 elements of array
0       100
1        99
2        98
3        97
4        96
5        95
6        94
7        93
8        92
9        91
Enter index of element to display: 10
Element 10 is 90
Elements 20 through 29 of array
0        80
1        79
2        78
3        77
4        76
5        75
6        74
7        73
8        72
9        71
```

- Analyze the sample codes (array1.asm and array1c.c). Reflective questions:

```
   What does array1.asm do? What does array1c.c do? How does array
being implemented in assembly?
```

Problem #10.

- Write an assembly program that search a number in a given array.
- Create a “searchArray.asm” (which prints the values in an array and identify if the a number is in the array) and then interface it with “array1c.c” . (*See the above example*)

- The output of your program is something like this:

```
Enter array size: 5
Enter value @ array[0]: 2
Enter value @ array[1]: 4
Enter value @ array[2]: 5
Enter value @ array[3]: 6
Enter value @ array[4]: 1
Array contains:
2 4 5 6 1
Enter number to be searched: 2
2 is @ array[0]
```

- A good programming practice is to write comments on important line of codes for readability and documentation.

Note: Take a screen record of your working code and make sure to record a video explaining each line of your code as well as showing the correct output of your code. Use screen recorder application in Ubuntu (https://itsfoss.com/best-linux-screen-recorders/) or Windows (https://atomisystems.com/screencasting/record-screen-windows-10/)

Deadline :_____

Rubric for Programming Exercises				
Program (50 pts)	**Excellent**	**Good**	**Fair**	**Poor**
Program Execution	Program executes correctly with no syntax or runtime errors (9-10)	Program executes with minor (easily fixed) error (4-8)	Program executes with a major (not easily fixed) error (2-3)	Program does not execute (0-1)
Correct Output	Program displays correct output with no errors (9-10)	Output has minor errors (6-8)	Output has multiple errors (3-5)	Output is incorrect (0-2)

Design of Output	Program displays more than expected (7-8)	Program displays minimally expected output (5-6)	Program does not display the required output (3-4)	Output is poorly designed (0-2)
Design of Logic	Program is logically well-designed (9-10)	Program has slight logic errors that do not significantly affect the results (6-8)	Program has significant logic errors (3-5)	Program is incorrect (0-2)
Standards	Program is stylistically well designed (6-7)	Few inappropriate design choices (i.e., poor variable names, improper indentation) (4-5)	Several inappropriate design choices (i.e., poor variable names, improper indentation) (2-3)	Program is poorly written (0-1)
Documentation	Program is well-documented (5)	Missing one required comment (4)	Missing two or more required comments (2-3)	Most or all documentation missing (0-1)