ITCS 201 – Fundamentals of Programming
Lecture 12: Lab Assignments
(Submit via PC^2)

Q1: Write a program to compute the average of an array using a self-defined function, named `compute_avg`. The program first receives a positive integer `n`, which specifies the size of the array. It then creates an array of size `n`, and asks the user to fill in all values. Next, it calls `compute_avg` to compute the average, which is then printed out with two decimal places.

Sample inputs and outputs:

Case 1:

Input	Output	Expected screen
5 1 4 4 4 9	4.40	5 1 4 4 4 9 4.40

Case 2:

Input	Output	Expected screen
1 7	7.00	1 7 7.00

Q2: Write a program to find the maximum and the minimum values of the input array. **You MUST write your code in the provided section** in the `find_max_min.c` file on the MyCourse website. If you modify the code in other sections, you will receive a **ZERO** score.

Sample inputs and outputs:

Case 1:

Input	Output	Expected screen
6 8 -9 -99 1 33 7	33 -99	6 8 -9 -99 1 33 7 33 -99

Case 2:

Input	Output	Expected screen
4 42 42 42 42	42 42	4 42 42 42 42 42 42

Case 3:

Input	Output	Expected screen
1 -9	-9 -9	1 -9 -9 -9

Q3: Write a program to sort THREE input values. You **MUST** define and use two self-defined functions, named `swap` and `sort_values`, as follows:

```
void swap(int *x, int *y);
void sort_values(int *x, int *y, int *z);
```

Note:

- The `swap` function is used to swap the values of two input variables.
- The `sort_values` function is used to sort the input values.
- In the `sort_values` function, you **MUST** use the `swap` function to sort the input values. Otherwise, you will receive a **ZERO** score.

Sample inputs and outputs:

Case 1:

Input	Output	Expected screen
3 2 1	1 2 3	3 2 1 1 2 3

Case 2:

Input	Output	Expected screen
2 99 8	2 8 99	2 99 8 2 8 99

Case 3:

Input	Output	Expected screen
8 4 4	4 4 8	8 4 4 4 4 8

Q4: Write a program to determine the intersection of two arrays using a self-defined function, named `intersect`. The program first receives a positive integer `n`, which specifies the size of the two arrays. It then creates two arrays of size `n`, and asks the user to fill in all values. Next, it calls the `intersect` function that determines the intersections of the two arrays and stores the results in the `out` array. Finally, the program prints out the intersections.

The function prototype of the `intersect` function **MUST** be as follows:

```
int intersect(int *arr1, int *arr2, int n, int *out);
```

Note:

- The `intersect` function **MUST** return the number of intersections.
- You can assume that the given inputs for each array in the test cases are always unique (i.e., always be a valid set) and sorted in ascending order (i.e., from small to large).
- The output intersections **MUST** be sorted in ascending order.

Sample inputs and outputs:

Case 1:

Input	Output	Expected screen
5 13 17 27 55 100 17 18 19 20 55	17 55	5 13 17 27 55 100 17 18 19 20 55 17 55

Case 2:

Input	Output	Expected screen
3 1 2 3 4 5 6	No intersect	3 1 2 3 4 5 6 No intersect

Case 3:

Input	Output	Expected screen
1 7 7	7	1 7 7 7