Programming Assignment 3

Fundamentals of Arrays II

(Deadline: Oct 12, 23:59)

If you are found guilty of plagiarism, strictest actions will be taken.

Instructions for the output: The below instructions apply to all the programs (even for the upcoming labs).

While printing the output, there is no new line

1. between any two output lines
2. after the last output line
3. Before the first input line

Refer to Appendix 1 for more details.

If the program fails the sanity check, it should just print **"error",** nothing less or more. Please note that "error" is different from "ERROR" and "Error" in C.

==

==

REGULAR EXERCISES

There are 3 regular exercises. You need to make only one program for all the 3 exercises. Since the exercises differ in the number of input parameters, the first argument to the program should be the exercise number. Refer to the test cases for further clarification.

Sanity checks required:

1. The number of input parameters should be in accordance with the exercise number.
2. You can assume the input parameters to be integers. However, you should verify them for their sign (positive/negative).

==

Exercise 1. Write a program to insert a new element after the second occurrence of the key element (already existing element) in the given 1-D array.

Input format: ./a.out [exercise number] [size of the array] [Elements of the array (space separated)] [New element to be inserted] [The key element]

Output format: Elements in the new array, separated by space

Test Case1: ./a.out 1 5 -15 64 459 64 10 78 64 *(First argument is the exercise number)*

```
-15 64 459 64 78 10
```

Test Case2: ./a.out 1 5 18 200 200 200 18 80 200

18 200 200 80 200 18

Test Case 3: ./a.out 1 4 10 20 21 40 2 10 *(There is no second occurrence of the key element,10)*

error

Test Case 4: ./a.out 1 4 10 20 2140 2 2 *(Incorrect number of total arguments)*

error

==

Exercise 2. Write a program to delete the last occurrence of the specified element from a 1-D array.

Input format: ./a.out [exercise number] [size of the array] [Elements of the array (space separated)] [Element to be deleted]

Output format: Elements in the new array, separated by space

Test Case1: ./a.out 2 5 -15 64 459 2 10 64 *(First argument is the exercise number)*

-15 459 2 10

Test Case2: ./a.out 2 4 200 56 200 34 200

200 56 34

Test Case 3: ./a.out 2 4 10 20 2140 2 6 *(The given element doesn't exist in the input array)*

error

Test Case 4: ./a.out 2 4 10 20 2140 2 20 7 *(Incorrect number of total arguments)*

error

==

Exercise 3. Write a program which displays all the prime numbers in the given array with the following constraint.

***Constraint**:* Only those prime numbers should be displayed whose location is a composite number.

Although you may have several prime numbers in the array, only those prime numbers should be displayed which are stored at non-prime locations. Remember that the first position in an array corresponds to the location/index 0.

Note: *Negative integers, 0 and 1, are neither prime nor composite.*

Input format: ./a.out [exercise number] [size of the array] [Elements of the array (space separated)]

Output format: Elements separated by space

> ***Print # if there is no element which satisfies the given constraints.***

Test Case1:

./a.out 3 9 2 64 7 83 67 7 3 9 -15 *(67 and 3 occur at locations 4 and 6, respectively)*

67 3

Test Case2: ./a.out 3 7 5 7 346 1 2 19 17

2 17

Test Case 3: ./a.out 3 9 10 20 2 2 6 1 0 8 1 *(no number will be displayed)*

#

Test Case 4: ./a.out 3 4 10 20 2140 2 20 7 *(Incorrect number of total arguments)*

error

Note the following points.

1. In the test case 1, -15 is not considered for printing since it is not prime.
2. In the test case 2, 5 and 7 are not printed because their locations are 0 and 1, none of which is a composite number.
3. In the test case 3, even though 0 and 1 occur at composite locations, they are not printed because they are not prime.

==
==

BONUS EXERCISES

There are 2 bonus exercises. Create separate programs for them.

Sanity checks required:

1. You can assume the input parameters to be integers. However, you should verify them for their sign (positive/negative).

Exercise 1: Corresponding to the two courses, Data-Structures and Professional-Ethics; you are given two arrays, one for each course. Each array stores the marks of the students for that subject. Refer to the sample table given below.

Sample Table (For 6 students)

	Array Index (Roll Number)	**Data-Structures Marks**	**Professional-Ethics Marks**
Student A	0	78	45
Student B	1	67	45
Student C	2	12	90
Student D	3	56	90
Student E	4	67	45
Student F	5	80	43

The goal is to prepare a grade-sheet where,

1. The total marks of the students should be printed in the descending order.
2. Alongwith the total marks, also print the individual marks of the two courses.
3. If the total marks for two students is the same, then the student scoring more marks in the Data-Structures course should be ranked higher and therefore his/her marks should be displayed before the marks of the other student.
4. Moreover, if two students get the same marks for both the subjects, the ordering of their marks should be according to their roll numbers. You can assume the students' roll numbers to be the array indices. For e.g. In the above sample table, students B and E have scored the same marks in both subjects. However, as per their roll numbers, the entry of Student B should be placed before the entry of Student E in the generated grade-sheet.

Refer to the test cases for further understanding.

Input format: ./a.out [number of students] [Data-Structures Marks Array] [Professional-Ethics Marks Array]

Output format:

[Roll_Number] [Total_marks] [Data_Structures_Marks] [Professional-Ethics_Marks] **<--***Highest scorer*

[Roll_Number] [Total_marks] [Data_Structures_Marks] [Professional-Ethics_Marks] **<--** *Second Highest scorer*

.

.

.

[Roll_Number] [Total_marks] [Data_Structures_Marks] [Professional-Ethics_Marks] **<--***Lowest scorer*

The test case for the above table is as follows

Test Case 1: ./a.out 6 78 67 12 56 67 80 45 45 90 90 45 43

```
3 146 56 90

5 123 80 43

0 123 78 45

1 112 67 45

4 112 67 45

2 102 12 90
```

Note: Ensure that the elements in one line of the output are separated by space and not some other whitespace character like tab or double spaces etc.

Test Case 2: ./a.out 4 12 20 21 40 10 20 5

```
error
```

Test Case 3: ./a.out 2 -3 4 1 2 *(marks can't be negative)*

```
 error
```

===

Exercise 2: Assume that you are given the variation in the price of a bungalow as an array where the *i*th element of the array denotes the price of the bungalow (in crores) on the *i*th day. As a real estate agent, you are interested in buying and selling this bungalow in order to earn a profit from this transaction (buy and sell).

1. You can do at most one transaction.
2. You cannot sell the bungalow before buying it.
3. If you buy the bungalow, it is mandatory to sell it.
4. As you intend to earn a profit, you will not buy the bungalow if you can't sell it for a higher price.

Input format: ./a.out [number of days] [Price of the bungalow at a given day (separated by spaces)]

Output format: [Buying_price] [Selling_price] [Profit_earned]

Test Case 1: ./a.out 5 4 12 20 21 40

```
4 40 36
```

Test Case 2: ./a.out 5 40 12 20 21 4

12 21 9

Test Case 3: ./a.out 5 40 39 38 21 19 *(no transaction will be done in this case)*

0 0 0

Test Case 4: ./a.out 5 40 -39 38 21 19 *(price can't be negative)*

error

==
==

SUBMISSION INSTRUCTIONS

Create separate folders for the submission of regular problems and the bonus problems. Name the folders as [your_roll_number]_[your_name]_regular and [your_roll_number]_[your_name]_bonus, respectively.

[your_roll_number]_[your_name]_regular: This folder should have one file named [your_roll_number]_[your_name]_p1.c.

[your_roll_number]_[your_name]_bonus: This folder should have two files named [your_roll_number]_[your_name]_p2.c (exercise 1) and [your_roll_number]_[your_name]_p3.c (exercise 2).

Zip both the folders together and name the zip file as [your_roll_number]_[your_name]_lab3. Upload the zip file to the Google classroom. Kindly use your complete roll number as assigned by the institute.

==

==

Appendix 1

Your program should not print any unnecessary new lines (or even tabs and spaces). As an example, assume that your code is printing 2 numbers as output, n (having value 10) and m (having value 40). Then your sequence of printf statements should be
=====================================
```
printf("%d",n);
printf("\n");
printf("%d",m);
```
This will result in an output like
```
10
40
```
=====================================
and not any of the following
1.
```
printf("%d",n);
printf("\n\n");
printf("%d",m);
```
Here, there is an additional newline between two output lines.
This will result in an output like
```
10

40
```
=====================================

2.
```
printf("%d",n);
printf("\n");
printf("%d",m);
printf("\n");
```
Here, there is an additional newline after the last output line.
This will result in an output like
```
10
40

```
=====================================
3.
```
printf("\n");
printf("%d",n);
printf("\n");
printf("%d",m);
```
Here, there is an additional newline before the first output line.
This will result in an output like
```

10
```

40
==