

**NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING**

Practical Exam 2 for Semester 2, AY2017/18
CS1010E — Programming Methodology

31 March 2018

Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of 2 exercises on 6 pages.
2. This is an open-book assessment. You may bring in any printed material and a calculator, but **not** other electronic devices, including but not limited to laptops and thumb-drives.
3. In line with the university rules, any form of communication with other students, or the use of unauthorised materials is considered cheating and you are liable to the disciplinary action.
4. Please switch off/silence your mobile phone and keep it out of view.
5. Please place your student card on the desk in front of you throughout the PE.
6. Your computer has been logged in with a special account. Do not log off or try to log in with your own NUSNET account.
7. A **plab account slip** will be issued to you at the beginning of the PE. You must write your programs in the given plab account. The host name is **pe10** (not sunfire!) No activity should be done outside this plab account.
8. Skeleton programs are already planted in your plab account. Please leave the programs in the home directory, and use the same program names as specified in the paper. Do not create subdirectory to put your programs there or we will not be able to find them!
9. You **do not** need to submit your programs to CodeCrunch. We will retrieve your programs and submit them to CodeCrunch after the PE. **Only your source codes (.c programs)** from your plab account will be collected after the PE. Hence, how you name your executable files is not important.
10. The first 10 minutes of the PE is reserved for you to read questions and design algorithms. You are not allowed to type your programs in this period of time.
11. You are **not** allowed to use syntax out of the scope of this PE (e.g. string functions or global variables). If in doubt, please check with the invigilator.
12. At the end of the PE, the invigilators will log you out of your computer remotely after issuing a warning.
13. Grading of the PE will take about 2 weeks. The result will be known to you in week 13.

ALL THE BEST!

Please DO NOT upload questions and solutions onto the Internet.

Exercise 1: Rearranging Array Elements

[35 marks]

Write a program **arrangeElements.c** to re-arrange a given integer array as follows:

1. All negative elements (if any) are moved to the front of the array;
2. All non-negative elements (if any) are moved to the back of the array;
3. Relative order between negative elements, as well as relative order between non-negative elements, must be preserved.

For example,

Array [1, -2, 3, -4] will be rearranged as [-2, -4, 1, 3].

Write in the skeleton program **arrangeElements.c** that has been loaded into your working directory. In particular, you are required to complete the function called **arrangeArr (int array[], int size)** that takes in an array and its **size** as parameters, and performs the re-arrangement. You may assume that **0 < size < 11**.

You **are not allowed to change any other part** of the skeleton program, except to provide necessary comments.

5 marks will be deducted from “Design” if additional array (or function) is used to implement your solution. However, if you find this requirement too challenging, you may create and use additional array in your program (which is easier). Though you will lose 5 marks from “Design”, you may still gain up to 20 marks from “Correctness”.

Two sample runs of the program are shown below with the user’s input shown in **bold**.

Sample run #1

```
Enter the array size: 6
Enter 6 integers into the array: 1 -2 3 -4 5 -6
Initial array: [ 1, -2, 3, -4, 5, -6 ].

After re-arranging, the array is:
[ -2, -4, -6, 1, 3, 5 ].
```

Sample run #2

```
Enter the array size: 5
Enter 5 integers into the array: 1 0 -3 1 -6
Initial array: [ 1, 0, -3, 1, -6 ].

After re-arranging, the array is:
[ -3, -6, 1, 0, 1 ].
```

Exercise 2: Fill Table

[65 marks]

Write a program **table.c** to perform the following:

- Read the number of rows and columns, and create a two-dimensional integer array with that number of rows and columns.
- Fill the array with values as specified in the **fill_table()** function below.
- Read the row numbers and column numbers of two array elements, and sum up all the values in the rectangular portion of the array cornered by these two elements.

You may assume that:

- The maximum size of the array is 10 rows and 8 columns, and the minimum size is 2 rows and 2 columns.
- All user inputs are integers.

Write in the skeleton program **table.c** that has been loaded into your working directory. In particular, you are required to complete the following three functions.

- A function **fill_table(int table[MAX_ROW][MAX_COL], int num_rows, int num_cols)** that fills **table** with values in the following way: the first row and the first column are all 1s; every other element is the sum of the element on top of it and the element to the left. An example is shown below.

Note that **MAX_ROW** and **MAX_COL** are two constants representing the declared size of the **table**; **num_rows** and **num_cols** are the actual size of the **table** in each sample run.

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	1	1	1	1	1	1
[1]	1	2	3	4	5	6
[2]	1	3	6	10	15	21
[3]	1	4	10	20	35	56
[4]	1	5	15	35	70	126

- A function **read_two_elements(int num_rows, int num_cols, int *ele1_r_p, int *ele1_c_p, int *ele2_r_p, int *ele2_c_p)** that reads the row numbers and column numbers of two array elements and passes them back to the caller function through the 4 pointer parameters (pointer **ele1_r_p** will pass back the row number of the first element **ele1**; pointer **ele2_c_p** will pass back the column number of the second element **ele2**, etc.).

Note that **ele1** and **ele2** must be two different elements of the array and their row numbers and columns numbers must be within the boundary of the array (e.g.

cannot be negative integers). If the user inputs are invalid, the function will prompt the user to type again. Maximum 3 attempts are allowed. The function will return 1 if the user enters valid inputs within 3 attempts, or 0 otherwise. You may refer to the sample runs #2 and #3 for examples.

- A function `compute_square_sum(int table[MAX_ROW][MAX_COL], int num_rows, int num_cols, int ele1_r, int ele1_c, int ele2_r, int ele2_c)` that returns the sum of values in the rectangular portion cornered by `ele1` and `ele2`. Parameters `ele1_r` and `ele1_c` represent the row number and column number of the first element `ele1`, etc.

Take note of the following programming restrictions:

- You must **NOT** change the function headers given in the skeleton program.
- You are **NOT** allowed to change the given `main()` function.
- You may write additional functions as necessary.

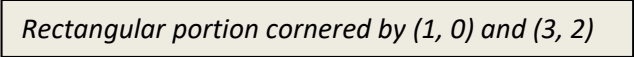
Four sample runs of the program are shown below with the user's input shown in **bold**.

Sample run #1

```

Enter the number of rows and columns of the table: 4 6
The table is shown below:
1      1      1      1      1      1
1      2      3      4      5      6
1      3      6      10     15     21
1      4      10     20     35     56
Enter row and column numbers of the first element: 1 0
Enter row and column numbers of the second element: 3 2
Sum = 31

```



Rectangular portion cornered by (1, 0) and (3, 2)

Sample run #2

```

Enter the number of rows and columns of the table: 5 5
The table is shown below:
1      1      1      1      1
1      2      3      4      5
1      3      6      10     15
1      4      10     20     35
1      5      15     35     70
Enter row and column numbers of the first element: -1 0
Enter row and column numbers of the second element: 9 9
Enter row and column numbers of the first element: 4 5
Enter row and column numbers of the second element: 0 0
Enter row and column numbers of the first element: 4 4
Enter row and column numbers of the second element: 4 4
Wrong inputs!

```

Sample run #3

```
Enter the number of rows and columns of the table: 7 3
The table is shown below:
1      1      1
1      2      3
1      3      6
1      4      10
1      5      15
1      6      21
1      7      28
Enter row and column numbers of the first element: 0 2
Enter row and column numbers of the second element: 0 -1
Enter row and column numbers of the first element: 0 2
Enter row and column numbers of the second element: 7 3
Enter row and column numbers of the first element: 0 2
Enter row and column numbers of the second element: 4 2
Sum = 35
```

Sample run #4

```
Enter the number of rows and columns of the table: 4 3
The table is shown below:
1      1      1
1      2      3
1      3      6
1      4      10
Enter row and column numbers of the first element: 3 2
Enter row and column numbers of the second element: 2 1
Sum = 23
```

Example Marking Scheme for an Exercise

1. Style:
 - Write student name, number, plab userID, tutorial group and program description in the program header.
 - Write a short and meaningful function description for every function (except main).
 - Apply consistent indentation and good naming of variables.
 - Write appropriate comments wherever necessary.
2. Design:
 - Correct definition and use of functions
 - Presence of function prototypes (if additional functions are used)
 - Is the algorithm simple or unnecessarily complicated?
3. Correctness:
 - Graders will manually go through your program and award marks function by function.
4. Additional penalties:
 - Program is not compilable: severely penalized
 - Break programming restrictions: very severely penalized

=== END OF PAPER ===