NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 2 (PE2) for Semester 1, AY2017/8
CS1010 Programming Methodology

4 November 2017 Time Allowed: 2 hours

INSTRUCTION TO CANDIDATES

1. Please leave your student card on your desk throughout the PE.
2. You are only allowed to read this cover page and the last page before the start of the PE. Do **not** flip the pages to read the questions inside until you are told to do so.
3. This paper consists of **2** tasks on 10 pages.
4. This is an open-book exam. You may bring in any printed material, but **not** electronic devices, including but not limited to laptop, thumb-drive, electronic dictionary and calculator. You are to switch off/silence your mobile phone and keep it out of view.
5. You may turn to the last page (page 10) to read some advice.
6. You will be logged into a special Windows account at the beginning of the PE. Do not log off until the end of the PE. Do not use your own NUSNET account.
7. A plab account slip will be issued to you at the beginning of the PE.
8. You are to write your program in the given **plab account**. The host name is **pe10** (not sunfire!). No activity should be done outside this plab account.
9. You are not allowed to type your programs in the first **fifteen** (**15**) minutes of the PE.
10. You **do not** need to submit your programs to CodeCrunch. We will retrieve your programs and submit them to CodeCrunch after the PE.
11. Skeleton programs and some test data files are already residing in your plab account. Please leave your 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, and do **not** name your programs differently from the expected one, or we will not be able to find them!
12. **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.
13. Please read carefully and follow all instructions in the question. If in doubt, please ask. Raise your hand and the invigilator will attend to you.
14. Any form of communication with other students or the use of unauthorised materials is considered cheating and you are liable to disciplinary action.
15. Please save your programs regularly during the PE.
16. When you are told to stop, please do so **immediately**, or you will be penalised.
17. At the end of the PE, please **log out from your plab account**.
18. Please check and take your belongings (especially student card) before you leave.
19. We will make arrangement for you to retrieve your programs after we have finished grading. Grading may take a week or more.

ALL THE BEST!

Exercise 1: S'pore Car Registration Number [50 marks]

Problem Statement

A typical Singaporean car registration number comes in the following format:

S	X_1	X_2	N_1	N_2	N_3	N_4	X_3

- The first letter is fixed as 'S', and is followed by two other letters $\mathbf{X_1}$ and $\mathbf{X_2}$.
- $\mathbf{X_1}$ **cannot be a vowel**. (Reason: the letters 'O' and 'I' look like 0 and 1 respectively, while the other vowels may cause it to look like proper words, e.g. SAY, SEE, SUN).
- If $\mathbf{X_1}$ is the letter 'H', then the vehicle is a taxi. If the letter is 'Z', then it is a rental car. If it is 'G', then it is a goods vehicle. If it is not any of these three letters, then the vehicle is a private car.
- $\mathbf{N_i}$ (where $1 \le i \le 4$) is a digit that ranges from 0 to 9. However, leading zeros are **not** printed out on the actual car plate (e.g. 0012 → **12**, 0100 → **100**).
- $\mathbf{X_3}$ is a letter that acts as a checksum letter.

Examples of **valid** registration numbers: "SHA123D", "SDU388Z" and "SCG6100H".

Note: The actual car registration number system is more complicated than what is described here. We are adopting a simplified version of it for this question.

The checksum letter $\mathbf{X_3}$ is used to check the validity of a registration number and it is generated with the following steps:

1. Convert the letters $\mathbf{X_1}$ and $\mathbf{X_2}$ into numbers (the corresponding number is the position of the letters of the alphabet, i.e. A=1, B=2, …, Z=26). Let's call them $\mathbf{NX_1}$ and $\mathbf{NX_2}$ respectively.
2. $\mathbf{NX_1}$, $\mathbf{NX_2}$, $\mathbf{N_1}$, $\mathbf{N_2}$, $\mathbf{N_3}$, and $\mathbf{N_4}$, are then multiplied by six fixed numbers 9, 4, 5, 4, 3, and 2 respectively.
3. The six products are then added up, and the sum is divided by 19. The remainder corresponds to one of the 19 letters used below:

Checksum	**A**	**Z**	**Y**	**X**	**U**	**T**	**S**	**R**	**P**	**M**	**L**
Remainder	0	1	2	3	4	5	6	7	8	9	10

Checksum	**K**	**J**	**H**	**G**	**E**	**D**	**C**	**B**
Remainder	11	12	13	14	15	16	17	18

Take note that the letters 'F', 'I', 'N', 'O', 'Q', 'V', and 'W', are **never used** as checksum letters in the tables provided.

In the case of "SDU388", the sum is (**4** × 9) + (**21** × 4) + (**0** × 5) + (**3** × 4) + (**8** × 3) + (**8** × 2) = 172. Then, 172 ÷ 19 gives a remainder of 1, which corresponds to the checksum letter 'Z' found in the table.

In this exercise, you are to write a program to check the type and validity of a car registration number. First, your program should read in a string that contains a car registration number and check whether the registration number belongs to a private car, taxi, rental car, goods vehicle, or is an invalid car type.

Next, your program should proceed to check whether the input registration number is valid or invalid. It does this by checking if the checksum letter of the input registration number is one of the non-existent ones. If so, there is no need to proceed any further. Otherwise, it will generate the checksum letter and confirm the validity of the registration number.

Write on the skeleton file **carnum.c** given to you. The reading in of the registration number has already been provided. You are not allowed to change it. You **must** include the following functions with the following parameters in your program:

- **`int isValidChecksum(char regNo[])`**

  A function that takes in the registration number string, and returns 1 if the registration number is valid, or 0 otherwise.

- **`char genChecksum(char regNo[])`**

  A function that takes in the registration number string, and returns the checksum letter. It is used inside **`isValidChecksum()`**.

You can assume that the input string is a registration number in which the first character is 'S', the next 2 characters are uppercase letters, the next 1 to 4 characters are digits, and the last character is an uppercase letter. You may define additional functions as needed. Check sample runs for input and output format.

Sample Runs

Five sample runs are shown below with user input highlighted in **bold**.

```
Enter car reg no: SHA123X
Car is a taxi
Invalid registration number
```

```
Enter car reg no: SDU388Z
Car is a private car
Valid registration number
```

```
Enter car reg no: SDU388Q
Car is a private car
Invalid registration number
```

```
Enter car reg no: SCG6100H
Car is a private car
Valid registration number
```

```
Enter car reg no: SEA12B
Invalid car type
```

Exercise 2: Traversing a Topographical Map [50 marks]

Problem Statement

A topographical (or contour) map is one that shows the elevations of an area of surface or terrain, indicating the hills and valleys, steepness or gentleness of slopes. Points with the same elevation are joined using a contour line. Figure 1 gives an example of a topographical map, showing several contour lines, and represents a hill that peaks at 280m. A contour line is one that joins points on a map with the same elevation. A topographical map is useful when one wants to navigate over a terrain. Usually you would want to avoid high areas (peaks) and low areas (valleys).

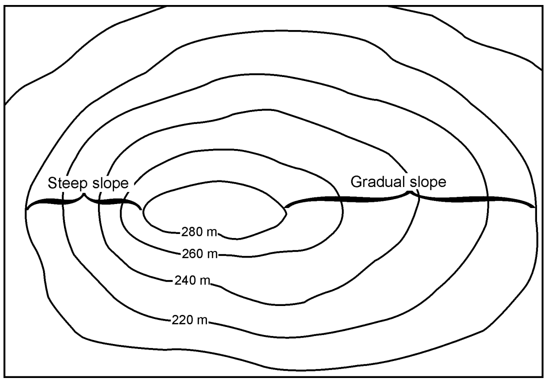

Figure 1: A topographical map showing several contour lines

A topographical map can be represented by a 2D array, where the elements of the array represent the elevation at those points in the terrain. Figure 2 shows an example 2D array (6 rows and 8 columns) representing a particular terrain. In the figure, an example of a contour line of elevation 39 is shown in *italics*. A peak is defined as an element for which all eight neighbours are lower than that element. A valley is defined as an element for which all eight neighbours are higher than that element. In the figure, peaks are shown in bold and valleys are underlined. Note that no element on the perimeter may be considered a peak or valley, because they have fewer than eight neighbours.

25	58	53	23	21	34	21	50
32	45	43	40	**41**	32	30	27
34	40	*39*	*39*	*39*	28	30	35
40	*39*	42	**48**	*39*	34	29	32
39	*39*	*39*	*39*	*39*	**49**	27	30
31	31	31	32	32	33	44	35

Figure 2: 2D array representation of a topographical map

In this exercise, you are to read in a 2D array that represents a topographical map of a terrain, count the number of peaks and valleys and show their locations. You may assume the array has a maximum of 12 rows and 12 columns.

For the example above, the following 2D array (**peaksandvalleys[6][8]**) shows the locations of the peaks (represented by 1) and valleys (represented by 2) as shown in figure 3.

0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0
0	0	0	0	0	**2**	0	0
0	0	0	**1**	0	0	0	0
0	0	0	0	0	**1**	**2**	0
0	0	0	0	0	0	0	0

Figure 3: peaks and valleys map

If you were planning to hike across this terrain, you would be constrained by the differences in elevation between neighbouring entries. A person can traverse between 2 adjacent locations if their elevations differ by no more than two. Adjacency refers to just the four basic compass directions, i.e. North, South, East, and West. Therefore, a point on the map is considered reachable if it is traversable from a starting point of **map[0][0]** (where **map[row][col]** is the topographical map) through any valid sequence of adjacent entries. In this exercise, you are required to compute all of the reachable locations from a starting point of **map[0][0]**. The output will be another 2D array **hike[row][col]** with values of 0s and 1s (1 means reachable, 0 unreachable).

For example, suppose the map is represented by **map[10][10]** as shown in figure 4.

```
50 51 54 58 60 60 60 63 68 71
48 52 51 59 60 60 63 63 69 70
44 48 52 55 58 61 64 64 66 69
44 46 53 52 57 60 60 61 65 68
42 45 50 54 59 61 63 63 66 70
38 42 46 56 56 63 64 61 64 62
36 40 44 50 58 60 66 65 62 61
34 32 40 49 56 62 67 66 65 60
30 36 40 47 50 64 64 63 62 60
50 50 50 50 50 50 50 50 50 50
```

Figure 4: 2D representation of map

Both East and South are traversable from **map[0][0]** so reachable entries would be **map[0][1]** and **map[1][0]**. The reachable points for the map are shown in bold in figure 5.

50	**51**	54	58	60	60	60	**63**	68	71
48	**52**	**51**	59	60	60	**63**	**63**	69	70
44	48	**52**	55	58	61	**64**	**64**	**66**	69
44	46	**53**	**52**	57	60	60	61	**65**	68
42	45	50	**54**	59	61	63	63	**66**	70
38	42	46	**56**	**56**	63	64	61	**64**	**62**
36	40	44	50	**58**	**60**	66	65	**62**	**61**
34	32	40	49	**56**	**62**	67	66	65	**60**
30	36	40	47	50	**64**	**64**	**63**	**62**	**60**
50	50	50	50	50	50	50	50	50	50

Figure 5: Reachable points shown in bold

And the resulting array **hike[10][10]** is as shown in figure 6:

1	**1**	0	0	0	0	0	**1**	0	0
1	**1**	**1**	0	0	0	**1**	**1**	0	0
0	0	**1**	0	0	0	**1**	**1**	**1**	0
0	0	**1**	**1**	0	0	0	0	**1**	0
0	0	0	**1**	0	0	0	0	**1**	0
0	0	0	**1**	**1**	0	0	0	**1**	**1**
0	0	0	0	**1**	**1**	0	0	**1**	**1**
0	0	0	0	**1**	**1**	0	0	0	**1**
0	0	0	0	0	**1**	**1**	**1**	**1**	**1**
0	0	0	0	0	0	0	0	0	0

Figure 6: Hiking map

Write on the skeleton file **contour.c** given to you. Functions to read and print the 2D array have already been provided. You must include the following functions in your program:

-
  ```
  void getPeaksValleys(int map[][MAX_COL],
                       int peaksandvalleys[][MAX_COL],
                       int row, int col,
                       int *nopeaks, int *novalleys)
  ```

  A function that takes in the 2D map array and counts the number of peaks and valleys, and also produces the peaks and valleys map.

-
  ```
  void getHikeTrail(int map[][MAX_COL], int hike[][MAX_COL],
                    int row, int col)
  ```

  A function that takes in the 2D map array and produces the required hiking map.

You may define additional functions as needed. Check sample runs for input and output format.

Sample Runs

Three sample runs are shown below with user input highlighted in **bold**.

```
Enter no of rows and cols: 6 8
Enter the data:
25 58 53 23 21 34 21 50
32 45 43 40 41 32 30 27
34 40 39 39 39 28 30 35
40 39 42 48 39 34 29 32
39 39 39 39 39 49 27 30
31 31 31 32 32 33 44 35
No of peaks: 3
No of valleys: 2
Peaks and Valleys map
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 2 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 2 0
0 0 0 0 0 0 0 0
Hiking map
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
Enter no of rows and cols: 10 10
Enter the data:
50 51 54 58 60 60 60 63 68 71
48 52 51 59 60 60 63 63 69 70
44 48 52 55 58 61 64 64 66 69
44 46 53 52 57 60 60 61 65 68
42 45 50 54 59 61 63 63 66 70
38 42 46 56 56 63 64 61 64 62
36 40 44 50 58 60 66 65 62 61
34 32 40 49 56 62 67 66 65 60
30 36 40 47 50 64 64 63 62 60
50 50 50 50 50 50 50 50 50 50
No of peaks: 1
No of valleys: 1
Peaks and Valleys map
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Hiking map
1 1 0 0 0 0 0 1 0 0
1 1 1 0 0 0 1 1 0 0
0 0 1 0 0 0 1 1 1 0
0 0 1 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 1 0 0 0 1 1
0 0 0 0 1 1 0 0 1 1
0 0 0 0 1 1 0 0 0 1
0 0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
```

```
Enter no of rows and cols: 2 4
Enter the data:
20 20 21 21
21 21 22 22
No of peaks: 0
No of valleys: 0
Peaks and Valleys map
0 0 0 0
0 0 0 0
Hiking map
1 1 1 1
1 1 1 1
```

This page is intentionally left blank.

CS1010 AY2017/8 Semester 1
Practical Exam 2 (PE2)

Advice – Please read!

- You are advised to spend the first 15 minutes thinking and designing your algorithms, instead of writing the programs right away.
- If you write a function, you must have a function prototype, and you must put the function definition after the main() function.
- You may write additional function(s) not mentioned in the question, if you think it is necessary.
- Any variable you use must be declared in some function. You are **not** allowed to use global variables (variables that are declared outside all the functions).
- You may assume that all inputs are valid, that is, you do not need to perform input validity check.
- Manage your time well! Do not spend excessive time on any exercise.
- Be careful in naming your executable code. Do **not** overwrite your source code with your executable code, especially if you are using the –o option in gcc!
- The rough marking scheme for both exercises is given below.

Rough Marking Scheme for each Exercise [50 marks]

1. Style: 5 marks
   - Are your name, student number, plab-id, DG and description filled at the top of the program?
   - Is there a description written at the top of every function (apart from the main() function)?
   - Is pre-condition for a function included wherever appropriate?
   - Are there proper indentation and naming of variables?
   - Are variables unnecessarily initialized, or not initialized when they are supposed to be?
   - Are there appropriate comments wherever necessary?
2. Design: 5 marks
   - Is the program modular?
   - Are function prototypes present?
   - Are the functions correctly defined and called?
   - Is function cohesion abided by?
   - Is algorithm not unnecessarily complicated?
3. Correctness: 40 marks for Task 1, 40 marks for Task 2. **Zero mark if cannot be compiled**.
4. Deductions (not restricted to the following):
   - Compiler issues warning with –Wall: 5 marks deduction
   - Use of global variables: 10 marks deduction
   - Use of goto statements: 10 marks deduction

--- END OF PAPER ---