NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

CS1010E — PROGRAMMING METHODOLOGY
(Semester 1 AY2014/2015)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **TWENTY-THREE (23)** questions and comprises **FIFTEEN(15)** printed pages, including this page.
2. Answer **ALL** questions.
3. Answer Section A (Questions 1 to 20) by shading the letter corresponding to the most appropriate answer on the OCR form provided.
4. Answer Section B (Questions 21 to 23) within the space provided in this booklet. You may use pen or pencil to write your answers.
5. This is an **OPEN BOOK** assessment. The maximum mark is **80**.
6. Calculators are allowed, but not electronic dictionaries, laptops, tablets, or other computing devices.
7. Do not look at the questions until you are told to do so.
8. Please write your **Student number** below. Do not write your name.

This portion is for examiner's use only.

Question	Marks	Remarks
Q21	/14	
Q22	/14	
Q23	/12	
Total	/40	

SECTION B (3 Questions : 40 Marks)
Write your answers in the space provided.

21. **[14 marks]** Given an integer $n \geq 0$, individual digits of n can be rearranged to obtain a smallest value. E.g., smallest value of 241114 is 111244; smallest value of 1010 is 11.

    (a) **[4 marks]** Write a function `fillTable` that fills a frequency table `freq` such that `freq[d]` is the number of occurrences of digit `d` in `n`. Assume that all elements of `freq` have been initialized to zeros before passing to the function.

    ANSWER:

```
void fillTable(int freq[], int n)
{
```

    (b) **[5 marks]** Write a function `getMin` that returns the smallest value upon rearrangement of the digits of `n`. Use the `fillTable` function defined in question 21a.

    ANSWER:

```
int getMin(int n)
{
```

(c) [**5 marks**] Write a function `getMinMax` that computes the smallest value `n_min`, as well as the largest value `n_max`, upon rearrangement of the digits of `n`. Use the `fillTable` function defined in question 21a and construct another **recursive function** to compute the values. You are **not allowed** to use repetition constructs.

ANSWER:

```
void getMinMax(int n, int *n_min, int *n_max)
{
```

22. **[14 marks]** This question is motivated from the game **2048** which has taken the world by storm! Assume that you are given a one-dimensional array of n elements with each element being empty, i.e. 0, or a number of the form 2^k with $k > 0$. An example is given below.

0	2	0	2	0	0	4	0	8

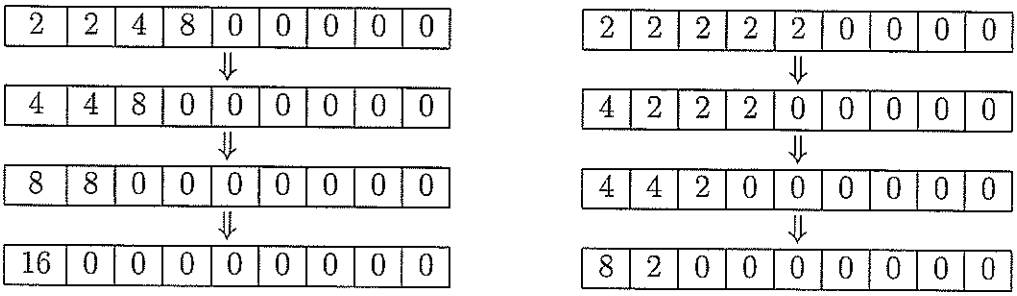
(a) **[6 marks]** Write a function `collapse` that takes an n-element array x, and collapses all the non-zero values to the left. Using the preceding array as an example, the resulting array becomes

2	2	4	8	0	0	0	0	0

ANSWER:

```
void collapse(int x[], int n)
{
```

(b) [**8 marks**] Write a function `merge` that takes an n-element collapsed array x and merges adjacent tiles towards the left. We illustrate with two examples.



Specifically, adjacent pairs of non-zero values are merged if they are the same. The merge repeats until no more adjacent pairs of non-zero values can be found.

ANSWER:

```
void merge(int x[], int n)
{
```

23. **[12 marks]** In the game of *tic-tac-toe*, a player who is playing o first tries to identify an attacking line (row, column or diagonal) that comprises two o's and a blank, so as to win the game. If an attack is not possible, the player then tries to identify a defensive line comprising two x's and blocks the opponent from winning. Clearly, checking lines (either attacking or defensive) requires inspection of all the rows, columns and diagonals of the *tic-tac-toe* board which is cumbersome to program. However, there is an alternative representation of the board that makes use of the magic square. The following shows a *tic-tac-toe* board and magic square side by side.

o	x	
	x	
o		o

8	1	6
3	5	7
4	9	2

Notice that the o's are situated corresponding to tiles 2, 4 and 8 of the magic square, while the x's are situated corresponding to tiles 1 and 5. We can thus represent the *tic-tac-toe* board on a one-dimensional array of size 10 (with element 0 unused).

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	x	o	.	o	x	.	.	o	.

To find out if player o can attack, one just needs to look at all possible pairs of indices $([i],[j])$ with value o (in this case $([2],[4])$, $([2],[8])$ and $([4],[8])$) and from each of these pairs, find out whether the element at index $[k]$ such that $i+j+k=15$ is empty (represented by the dot .). For the pair $([2],[4])$, the element indexed at $k=15-2-4=9$ is empty, so the player places an o there and wins. Likewise, we can determine the defensive lines and place an o to block when necessary. A particular move for player o is described below with the corresponding C program fragment.

if can attack
then attack and win
else
  if need to defend
  then defend
  else place o in any corner

```
bool win;
...
if (attack(board))
   win = true;
else
   if (!defend(board))
      freeCorner(board);
```

You are to define the following functions.

```
/* Returns true if an attack was executed, false otherwise. */
bool attack(char board[]);

/* Returns true if a block was executed, false otherwise. */
bool defend(char board[]);

/* Makes a move on any free corner. Assume at least one free corner. */
void freeCorner(char board[]);
```

Assume that the player plays o. As the functions `attack` and `defend` are almost the same, you may want to consider writing a helper function to modularize your program.

ANSWER:

— END OF PAPER —