NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
SEMESTER II AY2018/2019

MIDTERM ASSESSMENT FOR

TIC1002: INTRODUCTION TO COMPUTING AND PROGRAMMING II

7th March 2019 Time Allowed: 1 Hour 15 Minutes

INSTRUCTIONS TO CANDIDATES:

1. This assessment paper consists of **FOUR (4) questions.**

2. This assessment paper comprises **FIVE (5)** printed pages including this front page.

3. Answer all questions in the **ANSWER SHEET** given. The ANSWER SHEET comprises **TWO (2)** printed pages.

4. Submit only the **ANSWER SHEET** at the end of assessment.

5. Marks allocated to each question are indicated. Total marks for the paper is **40**.

6. This is an open book assessment. No electronics (including calculator).

1. [**12 marks**] Given function **t()** below:

```
int t( int input, int base )
{
    int weight, round, i, result = 0;

    for (round = 0; input != 0; input /=10){
        weight = 1;
        for (i = 0; i < round; i++){
            weight *= base;
        }
        result += weight * (input % 10);
        round++;
    }

    return result;
}
```

a. [3 marks x 3] Give the output for the following function calls:

`t( 10101, 2 )`	`t( 321, 4 )`	`t( 56789, 10 )`

b. [3 marks] State the complexity of the code using the Big-O notation. Be careful to state the meaning of any variables used in the Big-O notation.

2. **[8 marks]** Sudoku is a popular mathematical puzzle. Player tries to fill a 9 x 9 grid with digits 1 to 9 and satisfy the rules that the **digits in a row, column and square must be unique**. For example, row 5 and column 3 shows how such numbers can be placed. Note that the "square" restriction refers to the **nine** 3 x 3 squares (shown with the thicker border below). In this example, the center 3 x 3 square satisfy the "unique digits" restriction.

	0	1	2	3	4	5	6	7	8
0		4	7	**2**				3	
1	9			**3**		6		7	
2	5		6	**8**					
3				**6**	**9**	**8**			
4				**7**	**3**	**4**			
5	**3**	**6**	**9**	**1**	**2**	**5**	**7**	**4**	**8**
6			4	**9**		2	3		
7	2	9		**5**					
8				**4**	8				7

Suppose we use a 9 x 9 two-dimensional integer array to store a sudoko puzzle:

```
int sudoku[9][9];
```

Each location is called a **cell** for ease of reference, e.g. cell at row 5, column 2 is a "9". For simplicity, we assume that **0** means the cell is not filled (i.e. no digit) at the moment.

Give the following two functions in C++:

a. **[3 marks]** The function `filledCellInRow(int sudoku [][9], int row)` returns the number of filled cell in the row, e.g. row 5 has **9** filled cells, row 8 has **3** filled cells etc. Refer to answer sheet for the complete function header with parameters.

b. **[5 marks]** The function `filledCellInSquare(int sudoku [][9], int row, int col)` returns the number of filled cell in the 3 x 3 square where the user indicated cell (i.e. the cell at **[row][col]**) resides. e.g. if user indicated cell is at row 5 column 5, there are **9** filled cells (in the center 3 x 3 square), for user indicated cell at row 0, column 0, there are **5** filled cells (in the top left 3 x 3 square).

3. **[9 marks]** This question reuse the Sudoku setting from Q2, but is otherwise independent.

   Ms.Kudosu is implementing a "smart Sudoku" program. She intends to keep track of the **possible digits** that can be used in any cell. For example, cell at row 0, column 0 **cannot** be {2, 3, 4, 7} (already used in same row), **cannot be** {2, 3, 5, 9} (already used in the same column) and **cannot be** {4, 5, 6, 7, 9} (already use in the same 3 x 3 square). So, combining these info, the cell at row 0, column 0 cannot take the digit {2, 3, 4, 5, 6, 7, 9}

   Her design is as follows: Each cell contains a digit (0 to 9, 0 means not filled) and a **boolean array** to indicate which digit (1 to 9) can be used (**true =** can use, **false** = cannot use).

   a. **[3 marks]** Give the C++ structure declaration for **one single cell**.

   b. **[2 marks]** Give the C++ declaration for one Sudoku puzzle: (there is no need to fill in the values).

   c. **[4 marks]** Give the C++ implementation for the following function. This function update **all other cells** in the same column so that the digit ***D*** can no longer be used for those cells. Note that the first parameter is the same type of declaration of **part (b) above:**

```
void update_column( … Sudoku… , int col, int D);
```

4. **[11 marks]** Read the following scenario carefully and answer the sub-questions. **Note: the sub questions below are independent from each other.**

   Uncle Soo is processing the PE score for the TIC1002 class. There are 500 students in the class, each with a unique student number. The PE score ranges from 0 to 20. Hence, each record contains ***{Student Name, Student Number, PE Score}.***

   a. **[2 marks]** Suppose the records are **already sorted** by PE Score. Uncle Soo want to sort the records now by Student Number while "maintaining relative order of items with similar sorting keys". Can **selection sort** be used **in this scenario? Briefly explain.**

   b. **[2 marks]** Suppose the records are printed together with the source code submitted by students (~5 pages per student) and **sorted by Student Number**. However, Uncle Soo's naughty son pulled out a few (< 5) records randomly from the whole stack. If Uncle Soo want to put back those records in Student Number order, should he use **bubble sort** or **insertion sort** for the paper records? You need to base your answer on this scenario and justify your answer briefly.

   c. **[3 marks]** Suppose the records are **unsorted** (i.e. in some random order) and Uncle Soo wants to use **counting sort**. Give the **array size** for the following for this scenario:

int score[***<size>***]; //Array for storing the PE scores
int freq[***<size>***]; //Array for storing the frequency of the PE scores
int cfreq[***<size>***]; //Array for storing the cumulative frequency of the PE scores

   d. **[4 marks]** In the lecture, the **counting frequency of score** for counting sort is coded as:

```
    //get frequency;
    for (i = 0; i < N; i++){
         idx = score[i];
         freq[idx]++;
         //freq[ score[i] ]++;
    }
```

   Give a **recursive implementation** for the loop's logic. The recursive function header is given below as a hint:

```
void getFrequency( int score[],  int freq[], int nScore);
```

**~~~ End of Paper ~~~**