NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
SEMESTER II AY2019/2020

MIDTERM ASSESSMENT FOR
TIC1002: INTRODUCTION TO COMPUTING AND PROGRAMMING II

5th March 2020 Time Allowed: 1 Hour 15 Minutes

INSTRUCTIONS TO CANDIDATES:

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

2. This assessment paper comprises **FOUR (4)** 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 ***f()*** below:

```
int f( int x, int y)
{
    int d = x - y, r = 0;

    if (d != 0)
        r = d / abs(d);
    return r;
}    //abs( v ) returns the absolute value of 'v'
```

a. [**2 marks x 3**] Give the output for the following function calls:

f(123, 51)	f(15, 321)	f(777, 777)

b. [**4 marks**] Suppose we want to check whether an array of values is ***unimodal***, where the values (from left to right) keep increasing until a certain point (i.e. the max), then keep decreasing until the end **OR** the values keep decreasing until the **min**, then keep increasing until the end. Some examples and counter-examples:

Unimodal	NOT unimodal

{1, 3, 5, 9, 2, -5} //↑ then ↓
{15, 12, 7, -2, -7, -1, 3} //↓ then ↑
	{2, 3, 4, 5, 6} //only increasing
	{3, 2, 1, 0, -1} //only decreasing
	{3, 6, 2, 1, 5, 7} //↑ then ↓ then ↑

Alexei wrote the following **near complete** function. **Give short code (1-4 lines) to complete the function at the indicated location.**

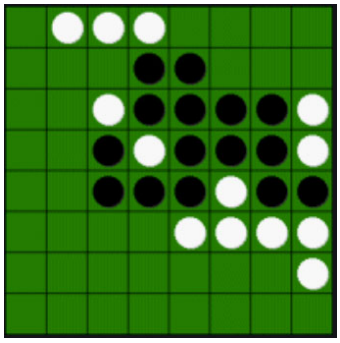
```
bool isUnimodal(int A[], int size)
{
    int i, c, d = 0;

    if (size < 2)
        return false;   //size 2 array cannot be unimodal

    d = f(A[0], A[1]);
    for (i = 1; i < size-1; i++){
        if (d != f( A[i], A[i+1]) ){
            c++;
            d = f( A[i], A[i+1]);
        }
    }
     //PART b. Your Code Needed
}
```

c. [**2 marks**] Briefly describe the purpose of the variable ***c*** in the function above.

2. **[12 marks]** ***Reversi*** is a popular board game. It has a board with 8 x 8 squares. Each square can only be ***Empty*** or containing a ***White*** or ***Black*** piece. An example board is given below:



Suppose we use a C++ **structure** to represent the state of a **single square**:

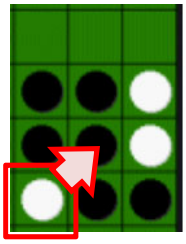
```
struct Square {
    bool isEmpty;
    int color;    //1 = White, 2 = Black
};
```

Answer the following using C++:

a. **[2 marks]** Declare the **board** as 2D array of the above structure.

b. **[3 marks]** Give a function **`void initReversi( <The Board> )`**, which initialize the entire reverse board to be empty. The parameter ***<The Board>*** is the same as your part (a) answer.

c. **[5 marks]** Give a function **`void flipRightUp( <The Board>, int row, int col, int newColor )`**, which changes the **color of the piece(s) starting from location (row, col), continuing <u>rightward + upward</u> (i.e. diagonal)** until you encounter a piece with same color as **`newColor`** **OR** you reach empty location **OR** you reach the border of the board.



To simplify your attempt, you can assume the following helper functions:

- isValid(R, C) ➔ return true if row R and col C is a valid location, false otherwise

d. **[2 marks]** Give the complexity of (c), expressing in terms of **R** (number of rows) and **C** (number of columns).

3. **[4 marks]** Give a **recursive implementation** of the `bool isEven( num )` function, which return true if `num` is Even number, false otherwise. You can assume `num` is a **non-negative integer**, i.e. `num >= 0`. Note: **You are restricted to ONLY add / subtract and equality check (==) mathematical operations.** Other operations e.g. multiply, divide, modulo (remainder), etc **cannot be used**.

4. **[12 marks]** Each of the following part describe a problem scenario and outline **two alternative solutions.** Use ☑ and ☒ to indicate whether the solution **works OR fails to solve the problem and meet stated requirements.** Use **one sentence to briefly explain why the solution(s) failed**. There is **no need** to explain if the solution is working.

   a. **[4 marks]**

Problem: An unsorted list with {***student number***, ***tutorial group number***}. We want to produce a list where tutorial groups are shown in order and student in each tutorial group are ordered by student number.
Solution One: Bubble sort the list by tutorial group number **then** selection sort the list by student number.
Solution Two: Selection sort the list by student number **then** bubble sort the list by tutorial group number.

   b. **[4 marks]**

Problem: An unsorted list of 10,000 { ***NRIC***, ***Yearly Salary***} data randomly selected from Singapore tax payer. We want to find the **top 5 highest Salary.**
Solution One: Binary search the list, then remove the top earner with the highest Salary; repeat for 4 more times.
Solution Two: Counting sort the list, then get the last 5 entries in the sorted list (the data is sorted in ascending order).

   c. **[4 marks]**

Problem: A list of unsorted 4D toto numbers (i.e. 0000 to 9999) bought by residents in a HDB block. We want to produce a list of **unique 4D toto number** by removing the duplicates. The order **does not matter**, but we need to finish processing in **better than $O(N^2)$ complexity.**
Solution One: Insertion sort the list **then** go through the list to pick up first occurrence of each 4D number.
Solution Two: Count frequency using counting sort idea **then** print any number with non-zero frequency.

~~~ End of Paper ~~~