ANSWER
Do Not Print!

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
SEMESTER II AY2017/2018

MIDTERM ASSESSMENT FOR
TIC1002: INTRODUCTION TO COMPUTING AND PROGRAMMING II

8th March 2018 Time Allowed: 1 Hour 15 Minutes

Student Number: | A | | | | | | | | |

INSTRUCTIONS TO CANDIDATES:

1. **Use a pen** to write your student number in the space provided above.
2. This assessment paper consists of **SIX (6) questions.**
3. This assessment paper comprises **SIX(6)** printed pages including this front page.
4. Answer all questions directly in the space given after each question. If necessary, use the back of the page. **You may write in pencil.**
5. Marks allocated to each question are indicated. Total marks for the paper is **40**.
6. This is an open book assessment.

EXAMINER'S USE ONLY		
Questions	**Possible**	**Marks**
Q1	7	
Q2	5	
Q3	5	
Q4	7	
Q5	7	
Q6	9	
Total	**40**	

1. [**7 marks**] Given function **`f()`** below:

```
void f( int input )
{
    int i, d;
    while (input > 0) {
        d = input % 10;
        for (i = 0; i < d; i++){
            printf("%d", d);
        }
        input /= 10;
    }
}
```

Give the output for the following function calls:

`f( 5 );`	55555
`f( 123 );`	333221
`f( 1023 );`	333221

State the complexity of the code, briefly explain how you arrive at the complexity. Be careful to state the meaning of any variables used in the Big-O notation.

O(D) // where D is the number of digits in input

The outer-loop runs D times
The for-loop runs at most 9 times, so we know that the time needed is always lesser than 10D ➔ O(D)

2. [**5 marks**] Implement a function **`print_power_table( B, E )`** such that it prints out **`B`** rows of number, each row contains **`E`** numbers. The value at row B_x and column E_y is $B_x^{E_y}$, where rows and columns both start from 1. For example, **`print_power_table( 3, 4 )`** give the following output:

1	1	1	1
2	4	8	16
3	9	27	81

```
void print_power_table( int B, int E)
{
    int row, col, value;

    for (row = 1; row <= B; row++){
        value = row;
        for (col = 1; col <= E; col++){
            printf("%d ", value);
            value *= row;
        }
        printf("\n");
    }

}
```

3. [**5 marks**] Give a **recursive** implementation for the following function. This function returns the **number of odd digits** in the given number **N**. For example, `count_odd(`***`12345`***`)` returns 3 (the digits "1", "3", "5" are odd), `count_odd(`***`2468`***`)` returns 0 (no odd digits).

```
int count_odd( int N )
{
    if (N == 0)
        return 0;

    return (N % 2) + count_odd( N / 10);

}
```

4. [**7 marks**] Given the following code:

```
#define MAXROW 10
#define MAXCOL 7

void splash(int canvas[MAXROW][MAXCOL],
            int value, int row, int col)
{

    int size = 0, i, j, pi, pj;

    while (value > 0){
        for (i = row-size; i<=row+size; i++){
            for (j = col-size; j <=col+size; j++){
                pi = (i + MAXROW) % MAXROW;
                pj = (j + MAXCOL) %MAXCOL;
                if (canvas[pi][pj] == 0)
                    canvas[pi][pj] = value;
            }
        }
        size++;
        value--;
    }

}
```


If we execute the following code fragment:

```
int canvas[MAXROW][MAXCOL] = {{0}};

splash(canvas, 3, 1, 5);
```

fill in the values for all **non-zero** locations in the **canvas[][]** at the end of execution

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

5. [**7 marks**] Given the **`Frac`** (fraction) structure and function:

```
struct Frac {
   int num, den;   //numerator and denominator
};
```

```
int go(struct Frac Farr[], int N, struct Frac X)
{
   int i;
   for ( i = 0; i < N; i++){
        if ( equal(&Farr[i], &X) ){
             return i;
        }
   }
   return -1;
}
```

Give an implementation of the **`equal()`** function if we are looking for a fraction that matches **exactly** with the target fraction X, e.g. if we look for 2 / 4, then 1 / 2, 4 / 8 are NOT acceptable, only an exact match 2 / 4 is returned.

```
bool equal( struct Fraction* A, struct Fraction* B)
{
     return (A->num == B->num) &&
            (A->den == B->den);
```

Give an implementation of the **`equal()`** function if we are looking for a fraction that matches **in value** with the target fraction X, e.g. if we look for 2 / 4, then 1 / 2, 4 / 8 or similar fractions are all acceptable. If needed, you can assume the function int GCD(int X, int Y) which returns the **greatest common divisor of X and Y** is available.

```
bool equal( struct Fraction* A, struct Fraction* B)
{
    int gcdA = GCD(A->num, A->den);
    int gcdB = GCD(B->num, B->den);
     return (A->num / gcdA == B->num / gcdB) &&
            (A->den / gcdA == B->den / gcdB);
```

6. [**9 marks**] Suppose we have the following two lists:
   a. The citizen list with **N** citizen names.
   b. The criminal list with **M** criminal names.

We know that the citizen list is **much larger** than the criminal list and both lists are **unsorted** initially. Suppose we need to find out which of the **M** criminal is in the citizen list, evaluates the following strategies by using time complexity:

Strategy A: **Linear Search the citizen list for each criminal.**

Cost to search for one criminal : O(__________) O(N)
Total Cost to search for **M** criminals: O(__________)O(M x N)

Strategy B: **Bubble Sort the citizen list then binary search for criminal.**

Bubble Sort the citizen list : O(__________) $O(N^2)$
Cost to search for one criminal : O(__________) O(lg N)
Total Cost to search for **M** criminals: O(__________) O(M lg N)

Suggest a better strategy with what you **have learned in this course so far**. Note that using better sorting algorithms is NOT the expected answer. Briefly describe the strategy and state the time complexity.

Your Strategy: Bubble Sort the criminal list. Then take each citizen and perform binary search in the sorted criminal list.

Analysis:

Bubble Sort the criminal list: $O(M^2)$
Cost for binary search one citizen in the sorted criminal list : O(lg M)
Total Cost for searching through all N citizens: O(N lg M)

As the dominating cost is the sorting, $O(M^2)$ is better than $O(N^2)$ as M << N as stated in question.

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