Mid-Term Exam of *Data Structures* (Fall 2016)

Department of Information Management and Finance

National Chiao Tung University

PART 1: True or False (Explain the reason if your answer is FALSE.) (10 pts.)

1. (T) Notation big-*O* aims at measuring the upper bound of a function, while Ω focuses on the lower bound.
2. (F) In C, the memory locations allocated for a dynamic 2-D memory are consecutive .
3. (T) $\sum_{i=0}^{n} i^2 = O(n^3)$
4. (T) $n^2+\log n = \Theta(n^2)$
5. (F) $n^3\log n = \Omega(n^4)$
6. (T) $40000 = \Theta(1)$
7. (T) The "undo" (Ctrl-Z) capability of many software applications could be implemented by using a stack.
8. (F) The users' requests in a fair ticket reservation system could be maintained by also using a stack.
9. (F) In C, &data[i][j] is equivalent to *(data[i]+j) and *(*(data+i)+j).
10. (T) Consider a specific problem *P* with a lower bound $\Omega(g(n))$. If there is an algorithm *A* solving *P* in time $O(g(n))$ in the worst case, then *A* is an optimal algorithm for *P*.

PART 2: Answer the following questions:

1. Determine the time complexity of the following pieces of programs using the tightest big-*O* notation.

(a) (4%)
```
for (x=1; x<=n; x++)
    for (y=1; y<=x; y++)
        z++;
```
$O(n^2)$

(b) (3%)
```
for (x=1; x<=n; x*=2)
    for (y=1; y<=x; y++)
        z++;
```
$O(n)$

(c) (3%)
```
for (x=1; x<=n; x++)
    for (y=1; y<=x; y*=2)
        z++;
```
$O(n\log n)$

2. (a) (3 pts.) Design a recursive algorithm to compute the *n*-th term of Fibonacci sequence.

Input: *n*

Output: F_n such that $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$

(b) (2 pts.) Design an iterative (non-recursive) algorithm to compute the *n*-th term of Fibonacci sequence.

(c) (3 pts.) How many additions are required for computing F_9 in (a)?

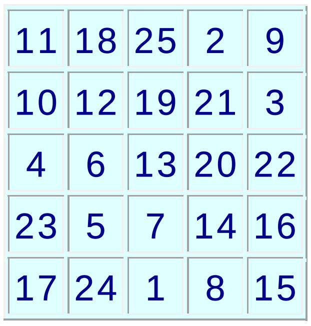
$T(n) = T(n\text{-}1) + T(n\text{-}2) + 1$ for $n \geq 2$ (0, otherwise)

n	0	1	2	3	4	5	6	7	8	9
T(*n*)	0	0	1	2	4	7	12	20	33	54

(d) (2 pts.) How many additions are required for computing F_9 in (b)? **8**

3. (a) (2 pts.) Give the magic square where "1" has been predefined at the center of the bottom row.

		1		



11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Sums of all rows, columns and diagonals are 65

(b) (6 pts.) Design an algorithm that reports (a) as the solution of the magic square. **[講義上有]**

(c) (2 pts) What is the summation value of each row/column/diagonal of an 11×11 magic square?

4. Consider row-major arrays *A* and *B*. Let the starting address be α and the size of each element be *l*. **[講義上有]**

(a) (2 pts) Find the address of $A[i][j]$ in terms of α and l, where A is a $u_1 \times u_2$ 2-D array.

(b) (4 pts) Find the address of $B[i_1][i_2] \dots [i_n]$, where B is a $u_1 \times u_2 \times \dots \times u_n$ n-D array.

(c) (4 pts) How many multiplications (×) and additions (+) are needed to compute the address in (b).

5. (10 pts.) Let S denote a stack. Characters 'A', 'B', 'C', 'D' and 'E' will be pushed into S one by one in this order. Please insert five more pop operations such that the following sequences can be obtained by these five push and five pop operations. Report the ten push/pop operations orderly if a solution exists; otherwise, explain the reason that no solution exists.

(a) A B C D E (b) B D E C A (c) D E C A B (d) E D C B A

(a) push('A'), pop(), push('B'), pop(), push('C'), pop(), push('D'), pop(), push('E'), pop()
(b) push('A'), push('B'), pop(), push('C'), push('D'), pop(), push('E'), pop(), pop(), pop()
(c) Impossible! There is no way to pop 'A' before 'B' when they are both in the stack.
(d) push('A'), push('B'), push('C'), push('D'), push('E'), pop(), pop(), pop(), pop(), pop()

6. Let `A^B` denote A^B in which the priority of ^ is higher than that of `*` and `/`.

(a) `A+(B-C)*D^(E+F/G)+H`
(2 pts.) Give the postfix notation of (a). **`ABC-DEFG/+^*+H+`**

(b) `QRSTUV+W^*X/+*+` **`+Q*R+S/*T^+UVWX`**
(2 pts.) Give the prefix notation of (b).

Suppose that functions "push" and "pop" for strings have already been defined. You may call them directly. Please design algorithms for
(c) (6 pts.) transforming the infix notation into the postfix notation;

```
Input: Infix e
Output: Postfix of e
n = parsing(e, token);
push(“#”);
for (i=0; i<n; i++)
{ s = token[i];
  if (s ∊ Operand) output(s);
  else if (s == ”)”)
          // 將堆疊中第一個 ( 之前的運算子皆pop出並印出之
        while ((x=pop())!=”(”) output(x);
     else
     {  while ( p(s) <= q(Stack[top]) )
        {    x = pop();
             output(x);
13          }
14          push(s);
15       }
16   }
17   while (Stack[top]!=”#”)
18   { x = pop();
19     output(x);
20 }
21 x = pop();
```

(d) (5 pts.) transforming the postfix notation into the prefix notation;

```
Input: Postfix post
Output: Prefix pre
1  n = parsing(post, token);
2  for (i=0; i<n; i++)
3  {  s = token[i];
4     if (s is an operand)
5        push(s);
6     else
7     {  y = pop(); x = pop();
         (or pop out the operand(s) that token[i] needs
          from stack; if any one consider unary operator)
8        S = x + token[i] + y;   // S is a string
9        x = result of expression token[i];
10       push(x);
11    }
12 }
13 // if stack is empty or top>=2 then the input postfix
      is incorrect
14 pre = pop();
```

7. Based on the following declaration:

```
# define Max_QSize 100;
# int Queue[Max_QSize];
int front = -1;
int rear  = -1;   // al
```

(a) (5 pts.) design a function for insert

```
    void insertQ(int ite
```

(b) (5 pts.) design a function for deleti

```
    int deleteQ()
```

```
// even better for insertQ, both are
fine
tmp = (rear+1) % MAX_QSIZE;
if (tmp == rear)
{ queueFull(); exit(1);}
rear = tmp;
queue[rear] = item;
```

```
void insertQ(element item)
{   rear = (rear+1) % MAX-QUEUE_SIZE;
```

```
        if (front == rear)
        { queueFull();  /* 輸出錯誤並離開 */ exit(1); }
        queue[rear] = item;
    }
    int deleteQ()
    {/* remove front element from the queue */
     if (front == rear)
     {  queueEmpty();  /* 回傳錯誤 */ return -1;   }
     front = (front+1) % MAX_QUEUE_SIZE;
     return queue[front];
    }
```

8. (10 pts.) Consider the following program segments.

[A]
```
int data[10][10], i, j;
for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
        data[i][j] = 1000-10*i-j-1;
```

Suppose that we obtain 1681400 when `(int) &data[0][0]` is printed and we have 999 when `data[0][0]` is printed. What values would be printed when the following variables are printed as integers?

(a)	`(data[1]+1)`	**1681448**
(b)	`*(data[0]+1)`	**998**
(c)	`data[1][0]`	**989**
(d)	`&data[1][1]`	**1681448**
(e)	`data[1]`	**1681444**

[B]
```
int ** data, i, j;
 data = new int * [10];
 for (i=0; i<10; i++) data[i] = new int [n];
 for (int i=0; i<m; i++)
     for (j = 0; j < n; j++)
         data[i][j] = 100+10*i+j;
```

Suppose that we obtain 3906000 when `(int) &data[0][0]` is printed,
3906048 `(int) &data[0][0]`
3906096 `(int) &data[1][0]`
100 `data[0][0]`

What the values would be when the following variables are printed as integers?

(f)	`*(data[0]+1)`	**101**
(g)	`data[1][0]`	**110**

(h)	`data[1]`	**3906096**
(i)	`&data[1][1]`	**3906100**
(j)	`*(*(data+1)+1)`	**111**

9. A string S consisting of four characters: '(', ')', '[' and ']' is *parenthese-balanced* if any of the following rules is met:
   (1) S is an empty string.
   (2) If α is parenthese-balanced, then (α) and $[\alpha]$ are parenthese-balanced.
   (3) If α and β are parenthese-balanced, then $\alpha\beta$ is parenthese-balanced.
   (a) (8 pts.) Design an algorithm to determine whether input string S is parenthese-balanced.
   Input: S consisting of members in {'(', ')', '[', ']'}
   Output: 1 if S is parenthese-balanced; 0 otherwise
   (b) (2 pts.) What is the time complexity of your algorithm? ***O*(*n*)**
   // Below is my code. Still, they score (to some degree) as long as they catch the point.

```
int   n = exp.Length();
char stack[256], top = -1;
if (n == 0) return true;
else
{   for (i=1; i<=n; i++)
    {  if (exp[i] == '(' || exp[i] == '[')
          stack[++top] = exp[i];
       else if (exp[i] == ')')
       {     if (stack[top] == '(') top--;
             else return false;
       }
       else if (exp[i] == ']')
       {     if (stack[top] == '[') top--;
             else return false;
       }
       else  return false;
    }
    if (top >= 0) return false;
}
return true;
```

10. In a classic chessboard, Queen is able to move any number of squares vertically, horizontally or diagonally. In the n-Queen problem, you need to place n chess Queens on an $n \times n$ chessboard so that no two queens threaten each other. That is, no two or more Queens would be placed at any single column, any single row and any single diagonal.

(a) (8 pts.) Design an algorithm to determine whether a given chessboard with exactly *n* chess queens already placed is a solution to the *n*-Queen problem or not.

Input: *n* and $n \times n$ chessboard *A* in which $A[i][j] = 1$ if a Queen is placed at board location (i, j); 0 otherwise

Output: 1 if *A* is a solution to the *n*-Queen problem; 0 otherwise

(b) (2 pts.) What is the time complexity of your algorithm? ***O*(n^2)**

// Below is my code. Still, they score (to some degree) as long as they catch the point.

```
 for (i=0; i<size && Sol; i++) // size is the same as n
 {  Queen[i] = -1;  // initial storage for Q's positions
    for (j=0; j<size && Sol; j++)
    { if (Chess[i][j])
          if (Queen[i] >= 0)  // same row detection
          {  Memo1->Lines->Add("Row "+IntToStr(i)+" has
         more than one Q.");
             Sol = false;
          }
          else Queen[i] = j;   // Record this Q's position
     }
     if (Queen[i]<0)
     {  Memo1->Lines->Add("Row "+IntToStr(i)+" has no
         Q.");
        Sol = false;
      }
 }
for (i=0; i<size-1 && Sol; i++)
    for (j=i+1; j<size && Sol; j++)
        if (Queen[i] == Queen[j]) // same col detection
        {   Memo1->Lines->Add("Col
        "+IntToStr(Queen[i])+" has more than one Q.");
            Sol = false;
        }
for (i=0; i<size-1 && Sol; i++)
    for (j=i+1; j<size && Sol; j++) // check diagonals
        if (abs(Queen[i]-Queen[j]) == abs(i-j))
        {   Memo1->Lines->Add("Diagonal
        ["+IntToStr(i)+","+IntToStr(Queen[i])+"]
        ["+IntToStr(j)+","+IntToStr(Queen[j])+"] has
        more than one Q.");
```

```
                Sol = false;
            }

    if (Sol)
        Memo1->Lines->Add("A solution.");
    else
        Memo1->Lines->Add("Not a solution.");
```

各 **row, col, diagonal** 計算 **1** 的個數，再判斷解與否；也是一種解法。