2019 Data Structure Midterm Exam

Note: 答案僅供參考

1. Label each of the following true or false:
   a. An abstract class may not contain any concrete methods (i.e. methods with actual code).
   b. An interface may not contain any concrete methods.
   c. A concrete class may extend more than one abstract class.
   d. A concrete class may implement more than one interface.
   e. If C is a concrete class extending abstract class A and f is an abstract method of A, then C must contain a concrete definition of f.
   f. If C is a concrete class implementing interface I and f is an abstract method of I, then C must contain a concrete definition of f.

2. (15 pts) Describe the below terms briefly.
   a. Data structure (hint: What are its functions? What are its purposes?)
   b. Algorithm (hint: what are the five criteria?)
   c. Characteristics of an Algorithm
   d. System Life Cycle (hint: what are the major steps?)
   e. ADT
   f. Big-O
   g. Theta-O
   h. Performance Analysis
   i. Performance Measure
   j. Space Complexity
   k. Time Complexity
   l. Array
   m. Stack
   n. Queue
   o. FIFO
   p.

   Ans:
   a. DS is the ways to represent data , or Simply, it is ways to represent data.
   More formal definitions:
   - A data structure is a specialized format for organizing and storing data.
   - Any data structure is designed to organize data to suit a specific purpose

so that it can be accessed and worked with in appropriate ways.

- In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.
- General data structure types include the array, the file, the record, the table, the tree, and so on.

b. Algorithm: An algorithm is a finite set of instructions that, if followed, accomplishes a particular task.

c. Unambiguous , Input, …

d. System Life Cycle includes the following steps: R (Requirements), A (Analysis), D (Design), R (Refinement), C (Coding), and V (Verification)

e.

f.

g.

h.

i.

j.

k. Time complexity is the computational complexity that describes the amount of time it takes to run an algorithm.

Or

Time complexity is the time to run an algorithm. It is (**Time complexity)** is commonly **estimated** by counting the number of elementary operations performed by the **algorithm**

l.

m.

n. Queue is an abstract data structure used to hold a collection of data elements. It follows First-In-First-Out methodology, i.e., the data item stored first will be taken out first.

Or

Queue is an abstract data structure that has two ends: One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). The data elements of a queue are in and out in a First-In-First-Out order.

o.

3. (5pts) Please order the following computing times:

$$1, \log n,\ n!,\ n^2,\ n\log n,\ 2^n, n/2^{n/2}$$

Ans:

$1 < \log n < n\log n < n^2 < 2^n < n/2^{n/2} < n!$

4. (5 pts) Please explain two disadvantages of implementing queues by using

linear array.

Ans: (Here lists two but you can give more)

1. When queue is full, an insertion or push results in an array resizing, i.e. allocating a new array and copy the values to the new array and the resizing is costly.

2. After a sequence of insertions and deletions, items will drift towards the end of the array – Even though there are empty spaces in front of the queue array, insertion (push) operation cannot be performed on the queue, since rear = capacity – 1. Shift array elements to the left after each deletion may achieve better space efficiency. However, shifting is not efficient and dominates the cost of the implementation.

5. (16 pts) Given the function cool() as below. Describe the running time of the following pseudocode in Big-Oh notation in terms of the variable n.

```
int cool( int k ) {
    int cost ;
    for (int i = 0 ; i < k ; ++ i )
        cost = cost + ( i * k ) ;
    return cost ;
}
```

a. *ans* = cool(3*n*);

b. *ans* = cool(*n*)+ cool(2*n*);

c.
```
int sum;
for ( int i = 0; i<n; ++i ) {
    if ( n < 1000)
        sum ++;
    else
        sum += cool(n);
}
```

d.
```
for (int j = 1 ; j < n ; j = j + 1) {
    val = 0;
        for (int i = 0 ; i < j ; ++ i ) {
            val = val + i * j;
            for (int k = 0 ; k < n ; ++ k ) {
                val ++;
            }
            }
        }
```

e. .

```
for (int i = 0; i < n * 1000; ++i) {
    sum = (sum * sum)/(n * i);
    for (int j = 0; j < i; ++j) {
      sum += j * i;
    }
 }
```

f.

```
for (int i = 0; i < n + 100; ++i) {
   for (int j = 0; j < i * n ; ++j){
     sum = sum + j;
   }
   for (int k = 0; k < n + n + n; ++k){
     c[k] = c[k] + sum;
   }
}
```

Ans:

a. $O(n)$

b. $O(n)$

c. $O(n^2)$

d. $O(n^3)$

e ?

f?

6. Analyze the time complexity of the following function. Your don't need to actually solve this relation, just write down the Relation function T(n).

```
int mystery(int n) {

   int answer;
   if (n > 0) {
      answer =(mystery(n-2)+3*mystery(n/2) + 5);
      return answer;
   }
   else
     return 1;
}
```

Ans: Write down the complete recurrence relation, T(n), for the running time of mystery(n). Be sure you include a base case T(0).

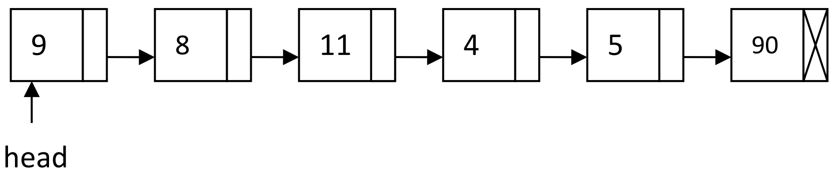
$$T(n) = T(n\text{-}2) + T(n/2) + C, \qquad for\ N > 0$$
$$T(n) = 1, \qquad for\ N = 0$$

7. (5 pts) A is a $10 \times 5 \times 2$ 3-dimension array of short int and α is the address of $A[0][0][0]$ 100, i.e., $\alpha = 100$. Using row major order, what is the memory address of $A[2][3][1]$?

Ans: .

$$100 + 2 \times (2 \times 5 \times 2 + 3 \times 2 + 1) = 154$$

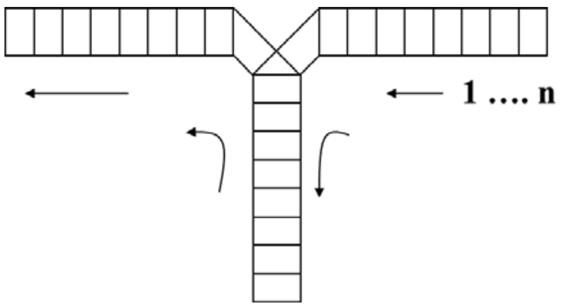
8. (10 pts) Consider the following linked list, show the output from the following program segment:



```
void odd (Node* head){
        Node *curr;
        int i;
        if (head== NULL)
                return;
        cur = head;
        printf ("%d    ", cur->data );
        i = (cur->data % 2) + 2;
        for(int j=0; j <= i; j++)
                if ( cur != NULL){
                        printf ("%d    ", cur->data );
                        cur = cur->next;
                }
        return;
 }
```

Ans: 9 9 8 11 4

9. (10 pts) Consider a railroad switching network as below. Railroad cars numbered 1, 2, 3 ..., *n* are at the right. Each car is brought into the stack **from the right segment** and **moved out to the left horizontal segment at any time**. For instance, if *n* = 3, we could move 1 in, move 2 in, move 3 in and then take the cars out producing the new order 3, 2, 1.



Now, for *n* = 4, is the permutation, 4312, possible? What about 4321? What about

Ans: No, Yes

1234	2134	2314	2341
1243	2143	2413	2431
1324	3124	3214	3241
1342	3142	3412	3421
1423	4123	4213	4231
1432	4132	4312	4321

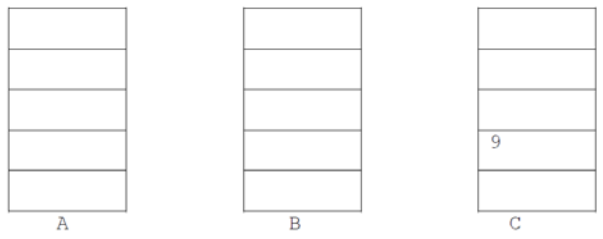
10. (10 pts) Given the Postfix() algorithm as below, illustrate the infix to postfix converting process by taking $A + \big((2 + B) * C - D\big)$ as example. (Fill the below table)

Ans:

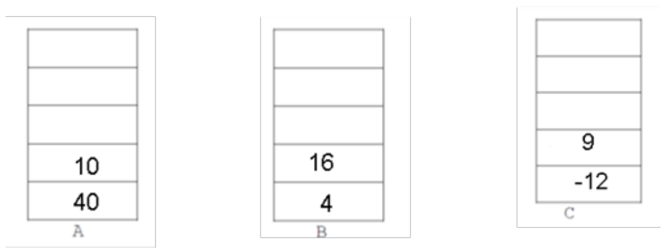
Token	Stack	output
A		A
+	+	
(	+(	
(	+((	
2	+((	A2
+	+((+	A2
B	+((+	A2B
)	+(	A2B+
*	+(*	A2B+
C	+(*	A2B+C
-	+(-	A2B+C*
D	+(-	A2B+c*D
)	+	A2B+C*D-
		A2B+C*D-+

11. (10 pts) Evaluate the following postfix expression, showing the values in the stack at each indicated point in the Postfix string (points A, B, and C).

$$8 \quad 5 \quad * \quad 4 \quad 6 \quad + \quad \overset{A}{\downarrow} \quad / \quad 2 \quad 8 \quad * \quad \overset{B}{\downarrow} \quad - \quad 4 \quad 5 \quad + \quad \overset{C}{\downarrow} \quad *$$



Ans:



12. Use the stack to translate infix to postfix expressions, please write out the processing sequence including input token, stack, and output.

A / (B – C) + D * (E – A) * C / F

Input Token	Stack	Output
/	+ /	ABC-/DEA-*C*
F	+ /	ABC-/DEA-*C*F
None	+	ABC-/DEA-*C*F/
None	Empty	ABC-/DEA-*C*F/+

13. (10 pts) Consider the following postfix expression:

1 8 + 7 / 5 6 - *

a. List the sequence of stack operations (push and pop) and arithmetic operations that would be used to evaluate the expression.

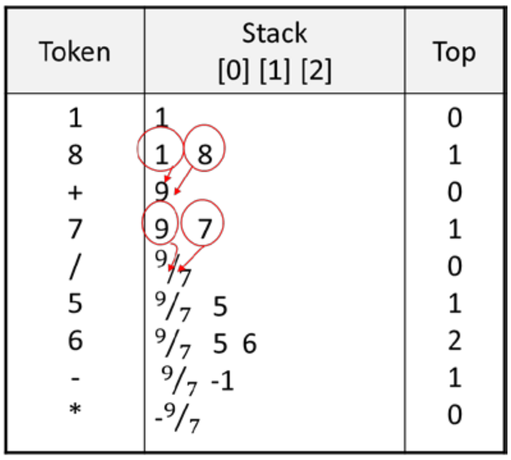
b. What does the stack contain after evaluating the expression?

```
void Eval(運算式 e)
{// 計算運算式 e。假設在 e 裡的最後一個符號是‘#’（一個符號可以是
 // 運算元、運算子、或是‘#’ ）。函式 NextToken 是用來從 e 中擷取
 // 下一個符號。這個函式用了一個堆疊 stack
    Stack<Token>stack; // 初始化 stack
    for (Token x = NextToken(e); x!= ‘#’; x=NextToken(e))
        if (x 是運算元) stak.Push(x) // 推入至 stack
        else {// 運算子
            為運算子 x 從 stack 中彈出正確數量的運算元 ；
            執行運算子 x 的運算並且將結果（如果有的話）推入到堆疊中 ；
        }
}
```

Ans:

a.

expression e = 1 8 + 7 / 5 6 – *



Token	Stack [0] [1] [2]	Top
1	1	0
8	1 8	1
+	9	0
7	9 7	1
/	$^{9}/_{7}$	0
5	$^{9}/_{7}$ 5	1
6	$^{9}/_{7}$ 5 6	2
-	$^{9}/_{7}$ -1	1
*	$-^{9}/_{7}$	0

b. $-^{9}/_{7}$

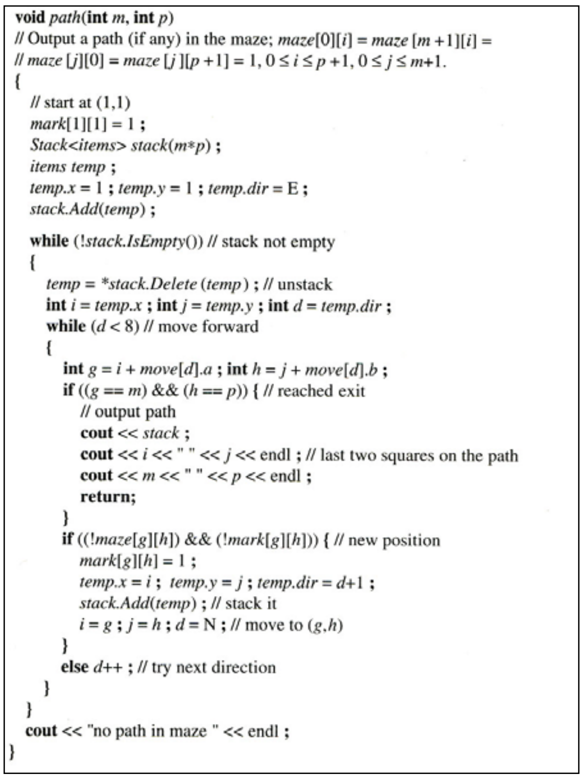
14. (10%) Complete the insertFront() function for a circular list.

```
template <class T>
void CircularList <T>::InsertFront(const T& e)
{// Insert the element eat the “front” of a circular
// List *this, where lastpoints to the last node in the list
    ChainNode <T> *newNode = new ChainNode <T>(e);
    if (last) { //nonempty list

    }
    else { //empty list
        last = newNode;
        newNode → link = newNode;
    }
}
```

Ans:

15. (15%) Given the following program for finding a path in a maze. Analyze its time complexity, and find the path according the given maze from (1, 1) to (10, 12).

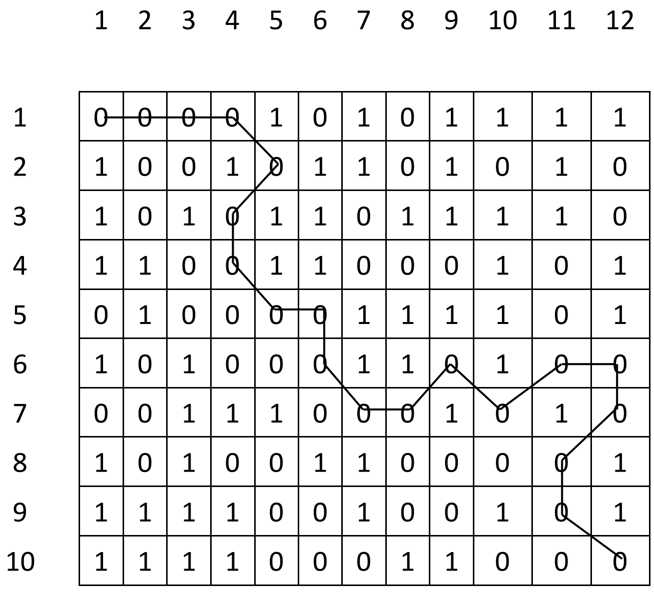


```
void path(int m, int p)
// Output a path (if any) in the maze; maze[0][i] = maze [m+1][i] =
// maze [j][0] = maze [j][p+1] = 1, 0 ≤ i ≤ p+1, 0 ≤ j ≤ m+1.
{
  // start at (1,1)
  mark[1][1] = 1 ;
  Stack<items> stack(m*p) ;
  items temp ;
  temp.x = 1 ; temp.y = 1 ; temp.dir = E ;
  stack.Add(temp) ;

  while (!stack.IsEmpty()) // stack not empty
  {
    temp = *stack.Delete (temp) ; // unstack
    int i = temp.x ; int j = temp.y ; int d = temp.dir ;
    while (d < 8) // move forward
    {
      int g = i + move[d].a ; int h = j + move[d].b ;
      if ((g == m) && (h == p)) { // reached exit
        // output path
        cout << stack ;
        cout << i << " " << j << endl ; // last two squares on the path
        cout << m << " " << p << endl ;
        return;
      }
      if ((!maze[g][h]) && (!mark[g][h])) { // new position
        mark[g][h] = 1 ;
        temp.x = i ; temp.y = j ; temp.dir = d+1 ;
        stack.Add(temp) ; // stack it
        i = g ; j = h ; d = N ; // move to (g,h)
      }
      else d++ ; // try next direction
    }
  }
  cout << "no path in maze " << endl ;
}
```

Ans: a. Time complexity:

The inner while-loop executes at most eight times for each direction. Each iteration takes *O*(1) time. Therefore, the inner loop totally takes O(1) time. The outer while loop executes until the stack is empty. Consider a worst cast where there are walls around the exit and all the rest are zeros in the maze. Therefore, at most *mp*-4 positions can be marked, and the computing time is O(*mp*).



	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	1	0	1	0	1	1	1	1
2	1	0	0	1	0	1	1	0	1	0	1	0
3	1	0	1	0	1	1	0	1	1	1	1	0
4	1	1	0	0	1	1	0	0	0	1	0	1
5	0	1	0	0	0	0	1	1	1	1	0	1
6	1	0	1	0	0	0	1	1	0	1	0	0
7	0	0	1	1	1	0	0	0	1	0	1	0
8	1	0	1	0	0	1	1	0	0	0	0	1
9	1	1	1	1	0	0	1	0	0	1	0	1
10	1	1	1	1	0	0	0	1	1	0	0	0

6. 請回答下列關於 Queues 的問題。

1) 利用 Linear Array 實作 Queues，可能面臨哪二種比較糟的情況？請說明之。

2) 利用 Circular Array可以避免 Linear Array實作Queues所產生的缺點，但亦會面臨一些問題，請說明之並提出解決方法

Ans:

Ans:

1) a) When queue is full, an insertion or push results in an array resizing, i.e., allocate a new array and copy the values to the new array.

b) After a sequence of insertions and deletions, items will drift towards the end of the array – Even though there are empty spaces in front of the queue array, insertion (push) operation cannot be performed on the queue, since $rear = capacity - 1$. Shift array elements to the left after each deletion may achieve better space efficiency. However, shifting is not efficient and dominates the cost of the implementation.

2) In order to distinguish whether a circular queue is full or empty, one space is left when queue is full。

7. The below shows an algorithm to find equivalence classes. If the input file “equiv.in” is as follows:

12 (0, 4) (3, 1) (6, 10) (8, 9) (7, 4) (6, 8) (3, 5) (2, 11) (11, 0) (1, 6)

Please list the lists after pairs have been input. (Show the results when the program has finished the line 34)

```
1   enum Boolean {FALSE, TRUE}
2   class ListNode{
3   friend void equivalence();
4   private:
5       int data;
6       ListNode   *link;
7       ListNode(int);
8   }

9   typedef ListNode   *ListNodePtr;
10  ListNode::ListNode(int d)
11  {
12      data = d;   link = 0;
13  }

14  void equivalence()
15  {
16      ifstream inFile(“equiv.in”, ios::in);
17      if (!infile){
18          cerr << “Cannot open input file” << endl;
19          return;
20      }

21      int i, j, n;
22      inFile >> n;
23      ListNodePtr *seq = new ListNodePtr [n];
24      Boolean *out = new Boolean [n];
25      for (i = 0; i < n; i++) {
26          seq[i] = 0;
27          out[i] = FALSE;
28      }
29      inFile >> i >> j;
```

```
//Phase 1: input equivalence paris
while (inFile.good()) {
    ListNode *x = new ListNode(j); x→link = seq[i]; seq[i] = x;
    ListNode *y = new ListNode(i); y→link = seq[j]; seq[j] = y;
    inFile >> i >> j;
}
// Phase 2: output equivalence paris
.........
```

16. (10 pts) For two programs, A and B, that solve the same problem, the time complexity of program A is $O(n \log n)$ while that of program B is $O(n^2)$. Does this mean that program A must run faster than B? Why?

Ans: No. Because the execution time of a program depends on the complexity of an algorithm, the performance of the machine, **the compiler**, and etc.

Asymptotic 式的演算法分析技巧 ，雖可提供我們評判演算法的優劣，但是在實際運用上必須進一步分析。比方說：演算法 *A* 需要 $10^3 nlogn$ 的步驟、演算法 *B* 需要 n^2 的步驟。分析的結果演算法 *A* 是 $O(nlogn)$ ，而演算法 *B* 是 $O(n^2)$，但這並不是說演算法 *A* 一定比演算法 *B* 好 — 當 *n* 不大於 10^3 的時候，演算法 *B* 就比演算法 *A* 有效率，因此在這個情形下，我們應該選擇演算法 *B* 而不是演算法 *A*。　換句話說，在選擇演算法時，除了考慮其 order 外，也必須考慮所處理問題的大小（即 *n* ）。

6. For each of the functions $f(N)$ given below, indicate the tightest bound possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one of a) – h)):

a) $f(N) = N \cdot (N^2 \log N + N^2)$

b) $f(N) = \log_{16} (2^N)$

c) $f(N) = (N/4) \log (N/4) + N/4$

d) $f(N) = (2N + 2N)^3$

e) $f(N) = N^{1/2} + \log N$

f) $f(N) = \text{N log } (100^3)$

Ans: a) O(N^3logN) b) O(N) c) O(NlogN) d) O(N^3) e) ? f) ?

7. (10 pts) Write the postfix form of the following expressions.

a. $-A + B - C + D$

b. $A * -B + C$

c. $(A + B) * D + E / (F + A * D) + C$

d. $A \; \&\& \; B \; || \; C \; || \; ! \; (E > F)$ // using the precedence of C++

Ans: u represents the unary minus. u is a operator that takes only one operand.

a. A u B + C – D+

b. A B u * C +

c. ?

d. ?

17. (10 pts) For a doubly linked list with a head node, please complete the insert and delete function. Note p is a single node and x is a doubly linked list.

```
class DblList;
class DblListNode {
friend class DblList;
private:
    int data;
    DblListNode *down, *right;
    };
class DblList {
public:

   void Delete(DblListNode *x);

   void Insert(DblListNode *p, DblListNode *x) ;
private:
    DblListNode *first;
};
void DblList :: Insert(DblListNode *p, DblListNode *x)
{ // inset node p to the right of node x

}
void DblList :: Delete(DblListNode *x)
{
    if (x = = first) throw "Deletion of header node not permited";

}
```

a.

```
void DblList :: Insert(DblListNode *p, DblListNode *x)
{ // inset node p to the right of node x
     p→left = x; p→right = x→right;
     x→right→left = p; x→left = p;
}
```

b.

```
void DblList :: Delete(DblListNode *x)
{
    if (x = = first) throw "Deletion of header node not permited";
    else {
     x→left→right = x→right;
     x→right→left = x→left;
     delete x;
}
```

18. Please give some reasons about the situations not to reuse a Class

Ans:

1. Reusing a class to design another may results in a less efficient class

2. Operations that are complex and specialized, not likely to be implemented in a reusable class

19. (10 pts) Write an program that takes two strings x, y and return either $-1, 0$

or $+1$ if $x < y, x = y$, or $x > y$ respectively.

```
int compare(Node *list1, Node *list2)
{
   // Traverse both lists. Stop when either end of a linked
   // list is reached or current characters don't match
   while (list1 && list2 && list1->c == list2->c)
   {
      list1 = list1->next;
      list2 = list2->next;
   }

   //   If both lists are not empty, compare mismatching
   //   characters
   if (list1 && list2)
      return (list1->c > list2->c)? 1: -1;

   // If either of the two lists has reached end
   if (list1 && !list2) return 1;
   if (list2 && !list1) return -1;

   // If none of the above conditions is true, both
   // lists have reached end
   return 0;
}
```