NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
SEMESTER II AY2008/2009

CS1102C: DATA STRUCTURES AND ALGORITHMS

Mid Term Test — Time Allowed: 1.5 Hours

MATRICULATION NUMBER:

H	T	0	4	0	8	3	1	R

INSTRUCTIONS TO CANDIDATES:

1. Write your matriculation number in the space provided above. Shade your matriculation number on the OCR form. Remember to sign on the form.

2. This examination paper consists of **FOUR (4) questions. Question 1 consists of 12 MCQ questions.**

3. This examination paper comprises **FOURTEEN (14)** printed pages including this front page.

4. Answer the **MCQ** questions by shading the **OCR** form and answer all of the other questions directly in the space given after each question. If necessary, use the back of the page.

5. Marks allocated to each question are indicated. Total marks for the paper is **100**.

6. This is a closed book examination and you can write in pencil.

EXAMINER'S USE ONLY				
Questions	**Possible**	**Marks**	**Grader**	**Check**
MCQ 1- 12	60			
Question 2	10			
Question 3	10			
Question 4	20			
Total	100			

MCQ (60 Marks)

1. Suppose we are trying to *simulate* the execution of a program P. This program contains a number of functions and each function contains a number of **program statements**. A function can also call another function during execution. An example program is shown below:

```
Function F( )
   statement 1
   statement 2
   statement 3
   Call Function G()
   statement 4
   statement 5
```

```
Function G()
      statement 1
      statement 2
      statement 3
```

Which of the following is the **best way** to represent the execution of Program P in general?

a. Use a queue to store all statements of a function. Use a stack of queues to represent the program execution.
b. Use a stack to store all statements of a function. Use a queue of stacks to represent the program execution.
c. Use a queue to stores all statements of a function. Use a queue of queues to represent the program execution.
d. Use a stack to store all statements of a function. Use a stack of stacks to represent the program execution.
e. None of the above.

2. Given the following template function:

```
template <typename T>
T minimum( const T& left, const T& right)
{
      return (left < right) ? left : right;
}
```

Suppose we are interested to find out the smaller value between 1.23 and 1.25, which of the following statement can accomplish this task **correctly**? You can assume the **smaller** is a double variable in the following code fragments.

```
i.   smaller = minimum<int> ( 1.23, 1.25 );
ii.  smaller = minimum( 1.23, 1.25 );
iii. smaller = minimum<double>( 1.23, 1.25 );
```

a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i) and (iii) only.
e. (i), (ii) and (iii).

3. Given two classes as follows:

```
class A
{
      private:
            int A_i, A_j;
      public:
            A( int m ) { A_i = m;   A_j = m; }
};

class B: public A
{
      private:
            int B_i;
      public:
            //Constructor for class B
};
```

Which of the following implementations for class B's constructor is syntactically correct?

a. `B( int k ) { A_i = k; A_j = k; B_i = k; }`
b. `B( int k ) { B_i = k; }` A does not have default constructor
c. `B( ): A( 123 ) { B_i = 456; }`
d. `B( int k ): A( k ) { A_i = k; A_j = k; B_i = k; }`
e. None of the above.

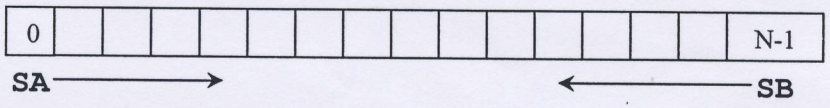
4. Which of the following statement(s) regarding List ADT is/are TRUE?

i. The public operations for the List ADT remain unchanged under different implementations ✓ - Unless the implementation is buggy, this statement is true
ii. The retrieve() operation is always successful ✗, nope, possible error: index out of bound
iii. The insert() operation can only perform insertion at head of the list ✗, can be anywhere! This is list, not stack

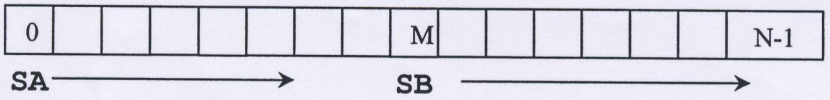
a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i) and (iii) only.
e. (i), (ii) and (iii).

5. Suppose we are forced to implement TWO stacks (**SA** and **SB**) using only a single array of size **N**. Below are three possible implementations:

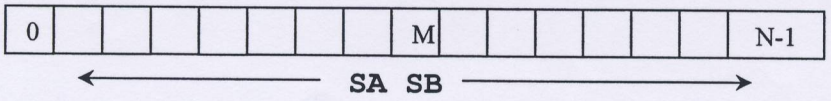
   i. **SA** uses **0** as bottom, stack top grows toward higher index.
   SB uses **N-1** as bottom, stack top grows toward lower index.



   ii. Take the middle point **M = N / 2,**
   SA uses **0** as bottom, stack top grows toward **M-1**.
   SB uses **M** as bottom, stack top grows toward **N-1**.



   iii. Take the middle point **M = N / 2,**
   SA uses **M-1** as bottom, stack top grows toward lower index
   SB uses **M** as bottom, stack top grows toward higher index



   Which of the following statement is TRUE?

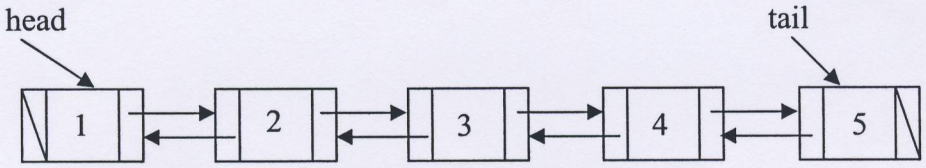
   a. In (iii), only **SA** or **SB** can be full at any time, but not both. X
   b. In (ii), **SB** cannot be empty. X
   c. In (i), **SA** is full only when **SB** is also full and vice versa. ✓
   d. In (ii), **SA** has double capacity compared to **SB**. X
   e. None of the above.

6. Which of the following statement regarding linked list variation is TRUE? All comparisons are done with respect to Singly Linked List with head pointer.

   a. Using dummy head node increase the time efficiency of traversing the linked list. X
   b. Double linked list increase the time efficiency of traversing the linked list. X
   c. Circular singly linked list with head pointer increase the time efficiency of insertion. X
   d. Double linked list will not benefit from having a dummy head node. X
   e. None of the above.

7. Given the following doubly linked list with the normal DListNode definition.

```
struct DListNode {
    int element;
    DListNode *prev;
    DListNode *next;
};
```



What is printed if the following statements are executed?

```
DListNode *front = head; *back = tail;
int temp;
while (front != tail) {
    temp = front->element;
    front->element = back->element;
    back->element = temp;
}
back = tail;
while (back != null) {
    cout << back->element;
    back = back->prev;
}
```

endless loop

a. 12345
b. 54321
c. 12321
d. 54345
e. None of the above

8. Which of the following statement(s) is/are TRUE regarding abstract data type (ADT)?

i. an ADT can have more than one implementation ✓
ii. an ADT can be used in more than one application ✓
iii. an ADT can have more than one specification ? ← debatable

a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i) and (iii) only.
e. (i), (ii) and (iii).

9. If **str1** = "**abcabccaabaaa**" and **str2** = "**abc**", what is the output of the following segment of codes?

```
int count = 0;
for (int j = 0; j < str1.length(); j++) {
    if (str1[j] == str2[0]) {
       bool match = true;
       int k = 1;
       while (match && k < str2.length())
          if (str1[j+k] == str2[k])
             k++;
          else
             match = false;
       if (match) count++;
    }
}
cout << count;
```

a. 0
b. 2
c. 3
d. 4
e. There is an error.

10. Let **intStack** be as stack of integer values and **opStack** be a character stack of arithmetic operators, where each operator is a single character. A function **doOperation()** performs the following steps:

I Pops two values from **intStack**.
II Pops an operator from **opStack**.
III Performs the arithmetic operation.
IV Pushes the result onto **intStack**.

Assume that the integer values **5**, **8**, **3**, and **2** are pushed onto **intStack** in that order (2 pushed last), and '*****', '**-**', and '**+**' are pushed onto **opStack** in that order ('**+**' pushed last).

If the **doOperation()** function is invoked three times. The top of **intStack** contains the result of evaluating which expression?

a. ((2 * 3) - 8) + 5
b. ((2 + 3) - 5) * 8
c. ((2 + 3) - 8) * 5
d. ((5 * 8) - 3) + 2
e. ((5 + 8) - 3) * 2

11. Assume we have the following declarations

```
int sum = 0;
int value;
queue<int> q;    // A queue of integers
```

If a few integer values are stored in q, which of the following code segments will correctly sum the elements of q and leave q unchanged?

i.
```
queue<int> temp;
while (!q.isEmpty()) {
   value = q.dequeue();
   sum += value;
   temp.enqueue(value);
}
q = temp;
```
✓

ii.
```
while (!q.isEmpty()) {
   value = q.dequeue();
   sum += value;
   q.enqueue(value);
}
```
endless loop

iii.
```
queue<int> temp = q;
while (!temp.isEmpty()) {
   value = temp.dequeue();
   sum += value;
}
```
✓

a. (i) and (ii) only
b. (ii) and (iii) only
c. (iii) only
d. (i) and (iii) only
e. (i), (ii) and (iii)

12. Given the following two classes

```
class A
{
   private:
         int A_i;
   protected:
        int A_j;
   public:
         int A_k;
};

class B : public A
{
   private:
         int B_i;
   protected:
        int B_j;
   public:
         int B_k;
};
```

which of the following statement(s) is/are TRUE?

i. An object of class B contains 6 attributes
ii. A public method of class B can access 4 attributes of the calling object directly
iii. A private method of class B can access 4 attributes of the calling object directly

a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i) and (iii) only.
e. (i), (ii) and (iii).

Short Question (40 marks)

Question 2 (10 marks)

Suppose we have a very simple game that is played with two types of color ball (red / blue) and a special "magic" ball. The game is played as follows:

- Color Balls (either red or blue) are stored as they occur
- When a magic ball appears:
  - If the previous two color balls are of the same color, they are merged (i.e. become one ball) and change color (i.e. red→blue or blue→red)
  - If the previous two color balls are of different colors, the magic ball has no effect
  - If there are less than two balls, the magic ball has no effect ✓
  - The magic ball will not be stored ✓

Part I) Suggest a data structure to simulate the game efficiently. Choose the best data structure from what we have learned so far in the course. (3 marks)

Stack

Part II) Use pseudo-code (or C++ code) to describe how to simulate the game correctly. (7 marks)

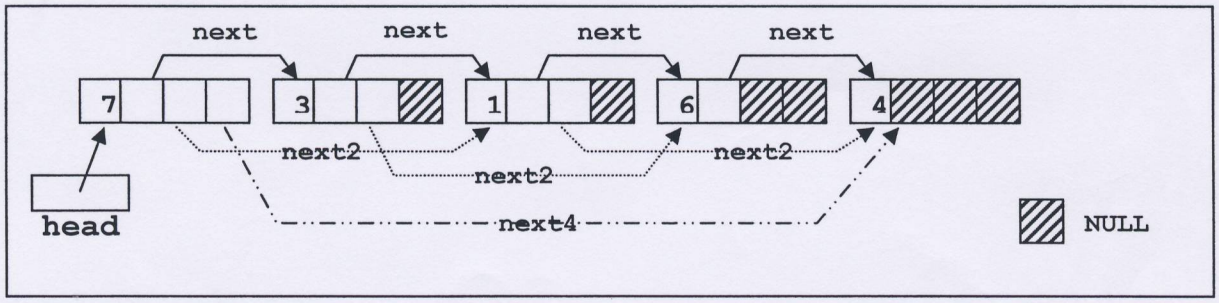
```
stack <char> S;
// Let 'R'→red, 'B'→blue, 'M'→ Magic Ball
read in the character c that represent the next Ball
loop until user say stop
if (c== 'R' || c == 'B')
    S.push(c);
else {// if (c == 'M')
    if (S.size()>=2) {
        char c1 = S.top(); S.pop();
        char c2 = S.top(); S.pop();
        if (c1 == c2)
            S.push (c1 == 'R' ? 'B' : 'R');
        else {
            S.push(c2); // restore them.
            S.push(c1);
        }
    }
}
```

Question 3 (10 marks)

Normally, a singly linked list node points to its immediate successor node (the next node) only. Suppose we add **two more pointers** to every list node to point to the second and fourth successor node as follows:

```
struct NewListNode
{
    int item;
    NewListNode* next;
    NewListNode* next2;     //the 2nd successor node
    NewListNode* next4;     //the 4th successor node
};
```

A graphical representation for a 5 nodes linked list using the NewListNode:



You can see that NULL is used when there is no valid successor for a particular pointer, e.g. the 3rd node (with item "1") does not have a 4th successor, the last node (with item "4") does not have the next node, the 2nd successor node nor the 4th successor node.

Part I) Use pseudo-code (or C++ code) to devise a traversal function that uses the **least number** of hops to arrive at a particular node. You can assume the target node is specified using index, with first node having index 1. (6 marks)

```
NewListNode*   goToNode( NewListNode* list, int index )

//Purpose: Use the least number of hops to arrive at the node
//         at position index
//Return: return the pointer to the target node,
//         or NULL if the index is out of range
//Assumption: index is >= 1
//              the list is setup properly
{

//Use C++ or pseudo code to describe your algorithm here

//Note that space given is not an indication of the
// expected length of answer

        int curIndex = 1;
        NewListNode* curPtr = list;
        while (1) {
            if (curPtr == null) // out of range.
                return null;
            else if (curIndex == index) // found target node.
                return curPtr;
            else if (curPtr->next4 != null) { // can jump 4
                curPtr = curPtr->next4;
                curIndex += 4;
            }
            else if (curPtr->next2 != null) { // can jump 2
                curPtr = curPtr->next2;
                curIndex += 2;
            }
            else { // last choice, move by 1 node
                curPtr = curPtr->next;
                curIndex ++;
            }
        }

}
```

Part II) Devise a function to perform insertion for this new type of linked list. You can make use of the function in Part I in your solution. Use either C++ or pseudo-code. (4 marks)

```
NewListNode*   insert( NewListNode* list, int index, int value )

//Purpose: Insert value at position index
//Return: The updated linked list
//Assumption: Index is >= 1 and within range
//            list is setup properly

{

//Use C++ or pseudo code to describe your algorithm here

//Note that space given is not an indication of the
// expected length of answer
```

```
NewListNode* prev4 = goToNode(list, index-4);
NewListNode* prev2 = goToNode(list, index-2);
NewListNode* prevN = goToNode(list, index-1);
NewListNode* nextN = goToNode(list, index);
NewListNode* next2 = goToNode(list, index+1);
NewListNode* next4 = goToNode(list, index+3);
                                              } off by 1

NewListNode* newNode = new NewListNode(value, nextN, next2, next4);
if(prev4 != null)
   prev4->next4 = newNode;
if(prev2 != null)
   prev2->next2 = newNode;
if(prevN != null)
   prevN->next = newNode;
```


```
}
```

Question 4 (20 marks)

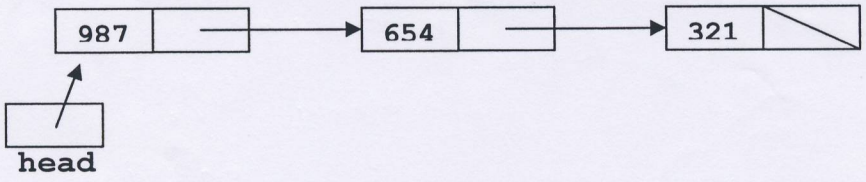
Given the following structure:

```
struct ListNode
{
    int item;
    ListNode* next;
};
```

And the following **recursive function**:

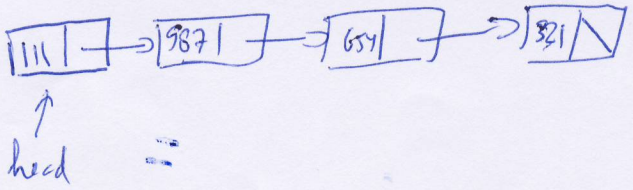
```
ListNode* mystery(ListNode* nodePtr, int m, int value)
{
    ListNode* newPtr;

    if ( m == 0 ) {
        newPtr = new ListNode;
        newPtr->item = value;
        newPtr->next = nodePtr;
        return newPtr;
    }

    nodePtr->next = mystery( nodePtr->next, m-1, value );
    return nodePtr;
}
```

Suppose we built the following linked list using the **ListNode** structure above, with **head** pointing to the first node:



Part I) Draw the resultant linked list after the following statement: (5 marks)

```
head = mystery( head, 0, 111 );
```



Part II) Draw the resultant linked list after the following statement (note: using the original linked list) **(5 marks)**

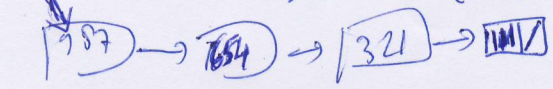
```
head = mystery( head, 3, 111 );
```

987 ->next = mystery (654 , 2, 111);

= 654->next = mystery (321, 1, 111)

= 321->next = (null, 0, 111)

head



Part III) Briefly describe the **purpose** of the `mystery()` function. **(5 marks)**

insert newnode at position n index from given node ptr

Part IV) Suppose we want to remove a node from the linked list given its value. Give a **recursive implementation** using C++ below: (5 marks)

```
ListNode* removeByValue( ListNode* list, int target)
//purpose: Remove the node with item == target from list
//return: Returns the list with node removed.
//    If the node cannot be found, the original list is returned
//    If multiple nodes have the same value, only the first
//        of such nodes is removed
{
    if ( list == null )  //end of list
        return null;
    else if ( list->value == target )  // found
        return list->next;  // skip this list node, so it is detached from the original list.
                            // can delete it if needed
    else // general case
        list->next = removeByValue (list->next, target); // recursive check.
        return list;
    }
}
```

~~~~End of Paper ~~~~