King Saud University	College of Computer and Information Sciences Department of Computer Science
Data Structures CSC 212	**Midterm Exam - Spring 2022**
Date: 09/03/2022	Duration: 90 minutes

Question 1 .... 15 points

Choose the most appropriate answer.

1. Which of the following is true about linked list implementation of stack?
   Ⓐ In push operation, if new nodes are inserted at the beginning of the linked list, then in pop operation, nodes must be removed from end. Ⓑ In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning. Ⓒ Both of the above Ⓓ None of the above Ⓔ None

2. Which of the following is not an application of stack?
   Ⓐ Reversing a string Ⓑ Implementation of recursion Ⓒ Job scheduling Ⓓ Evaluation of postfix expression Ⓔ None

3. The queue data structure is to be realized by using a stack. The number of stacks needed would be:
   Ⓐ 1 Ⓑ 3 Ⓒ 2 Ⓓ it is not possible Ⓔ None

4. What is the value of the postfix expression 6 3 2 4 + - *?
   Ⓐ 18 Ⓑ -18 Ⓒ 15 Ⓓ Invalid expression Ⓔ None

   6 3 2 4 +
   6 3 6 -
   6 3-6=-3x6=-18

5. The expression (5-3+2)*(4-2) when converted to postfix is
   Ⓐ 5 3 - 2 + 4 2 - * Ⓑ 5 3 2 + - 4 2 - * Ⓒ 5 3 - 4 2 - 2 + * Ⓓ 5 3 - + 2 4 2 - * Ⓔ None

   2 2 2 4 8

Question 2 .... 25 points

Implement a static boolean method `isExponential` user of the Queue ADT that processes a non-empty queue containing positive integers. The method should return `true` if the sequence of integers in the queue has an exponential growth. The queue is unchanged at the end. (Hint: in order for the method to return `true`, each number in the queue should be greater than or equal to the **squared value** of the number that precedes it).

The method signature is as follows: `public static boolean isExponential(Queue<Integer> q)`.

Example 1. *Given the following queue q1 = (1,3,9,100), the method should return* `true`*. On the other hand, the method should return* `false` *if we give it the following queue q2 = (1,2,4,10), because* $10 < 4 \times 4$.

1. Line 1:
   Ⓐ `int i = q.length();`
   Ⓑ `int i = 1;`
   Ⓒ `int i = q.length()- 1;`
   Ⓓ `int i = 0;`
   Ⓔ None

2. Line 2:
   Ⓐ `int v1 = q.serve()+ q.serve();`

(B) int v1 = q.serve();

(C) int v1 = q.serve()* 2;

(D) int v1 = q.length();

(E) None

3. Line 3:

(A) q.serve();

(B) q.enqueue(i);

(C) q.enqueue(v1 / 2);

(D) q.enqueue(v1);

(E) None

4. Line 4:

(A) while (i++ < q.length()){

(B) while (i++ < q.length()- 1){

(C) while (++i < q.length()- 1){

(D) while (i-- < q.length()){

(E) None

5. Line 5:

(A) int v2 = q.length();

(B) int v2 = q.serve();

(C) int v2 = v1;

(D) int v2 = q.serve()* q.serve();

(E) None

6. Line 6:

(A) q.enqueue(v1 + v2);

(B) q.enqueue(i);

(C) q.enqueue(q.serve());

(D) q.enqueue(v2);

(E) None

7. Line 7:

(A) if (2 * v1 > v2)

(B) if (v1 * v1 > v2)

(C) if (q.length()== 0)

(D) if (v1 * v2 < v2)

(E) None

8. Line 8:

(A) q.enqueue(v2);

(B) return v1;

(C) return null;

(D) return false;

(E) None

9. Line 9:

(A) v1 = v2;}

(B) v1 = v1 + v2;}

(C) v2++;}

(D) v1++;}

(E) None

10. Line 10:

(A) return v2;

(B) return q;

(C) return false;

(D) return true;

(E) None

Question 3 .... 25 points

Write the method **public static <T> DoubleLinkedList<T> concat(DoubleLinkedList<T> l1, DoubleLinkedList<T> l2, int k)** that receives two non-empty double linked lists and a positive integer k. The method creates and returns a list containing the concatenation of the first k elements from list l1 and the last k elements from list l2. However, the order of the elements taken from l1 should be from left to right whereas the elements taken from list l2 should be from right to left. Assume that k is strictly less than the length of each of l1 and l2.

Example 2. *If $l1 = A \leftrightarrow B \leftrightarrow C \leftrightarrow D$, $l2 = E \leftrightarrow F \leftrightarrow G \leftrightarrow H$ and $k = 3$, then the method should return $l3 = A \leftrightarrow B \leftrightarrow C \leftrightarrow H \leftrightarrow G \leftrightarrow F$*

```
public static <T> DoubleLinkedList<T> concat(DoubleLinkedList<T> l1, DoubleLinkedList<T>
    l2, int k){
  DoubleLinkedList<T> l3 = new DoubleLinkedList<T>();
  ...
```

1. Line 2:

(A) while (!l1.last()){

(B) current=head;

(C) l1.findFirst();

(D) l2.findFirst();

(E) None

2. Line 3:

   (A) `while (!l1.last()){`
   (B) `for (int i = 0; i < k; i++){`
   (C) `l1.findFirst();`
   (D) `l2.findFirst();`
   (E) None

3. Line 4:

   (A) `l3.insert(l1.retrieve());`
   (B) `l3.insert(l2.retrieve());`
   (C) `l1.current = l1.findNext();`
   (D) `l2.current = l2.current.next;`
   (E) None

4. Line 5:

   (A) `l2.findNext();}`
   (B) `l1.findNext();}`
   (C) `l1.findLast();`
   (D) `current = current.next;}`
   (E) None

5. Line 6:

   (A) `l2.findPrevious();`
   (B) `l2.findFirst();`
   (C) `while (!l1.last()){`
   (D) `while (!l2.last()){` ✓
   (E) None

6. Line 7:

   (A) `current = current.next`
   (B) `l1.findNext();}`
   (C) `l2.findNext();}`
   (D) `Tmp = new Node<T>();`
   (E) None

7. Line 8:

   (A) `for (int i = 0; i < k; i++){`
   (B) `while (!l2.last()){`
   (C) `l1.findFirst();`
   (D) `l2.findFirst();`
   (E) None

8. Line 9:

   (A) `l3.insert(l1.retrieve());`
   (B) `l3.insert(l2.retrieve());`
   (C) `l1.current = l1.findNext();`
   (D) `l2.current = l2.current.next;`
   (E) None

9. Line 10:

   (A) `l2.findNext();}`
   (B) `l2.findPrevious();}`
   (C) `l1.findPrevious();}`
   (D) `current = current.next;}`
   (E) None

10. Line 11:

   (A) `return l1;`
   (B) `return l2;`
   (C) `return l1+l2;`
   (D) `return l3;`
   (E) None

Question 4 .... 35 points

We want to implement the interface `Pol` below, which represents a polynomial:

```
public interface Pol {
  // Return the highest degree in the polynomial
  int getHighestDeg();
  // Return the coefficient of the degree d
  double getCoef(int d);
  // Set the coefficient of the degree d to c.
  void set(int d, double c);
  // Remove the degree d from the polynomial if it exists.
  void remove(int d);
}
```

For this, we write the class `LinkedPol`, a linked implementation of the interface `Pol`. Each node in this representation contains the degree of x, its coefficient and a pointer to the next term in the polynomial.

3. *The polynomial $2x^3 - 4x + 5$ is represented as $(3,2) \to (1,-4) \to (0,5)$, whereas $x^5 - 1$ is represented as ,−1). The empty list represents the polynomial 0.*

the list is sorted in decreasing order of the degree and each degree appears once.

e 4. *The output of the following program:*

```
 static void main(String[] args) {
p = new LinkedPol();
:t(4, 0);
et(0, -1);
et(2, 3);
et(1, 4);
set(5, 2);
remove(2);
 p now contains: (5,2)->(4,0)->(1,4)->(0,-1)
r (int d = p.getHighestDeg(); d >= 0; d--) {
System.out.print(d + "␣:␣" + p.getCoef(d) + ",␣");
```

4, 0

```
: 2.0, 4 : 0.0, 3 : 0.0, 2 : 0.0, 1 : 4.0, 0 : -1.0,
```

mplete the class LinkedPol below.

```
lass Node {
  public int deg;
  public double coef;
  public Node next;
  public Node(int deg, double coef) {
    this.deg = deg;
    this.coef = coef;
    next = null;
  }
}
public class LinkedPol implements Pol {
  private Node head;
  public LinkedPol() {
    head = null;
  }
  public int getHighestDeg() {
    if (head == null)
      return 0;
    else
      return head.deg;
  }
```

Method getCoef.

```
1 public double getCoef(int d) {
2   Node p = ...
3   while (...)
4     ...
5   if (...)
6     return 0;
7   else
8     ...
9 }
```

1. Line 2:

(A) Node p = current;
(B) Node p = null;
(C) Node p = head.next;
(D) Node p = head; ✓
(E) None

2. Line 3:

(A) while (p != null && p.deg > d) ✗
(B) while (p != null && p.deg < d)
(C) while (p.deg != d) ✗

(D) while (p != null)

(E) None

3. Line 4:

(A) p = p.next;

(B) p.next = p;

(C) head = head.next;

(D) return p.deg;

(E) None

4. Line 5:

(A) if (p.deg > d)

(B) if (p == null || p.deg != d)

(C) if (p == null)

(D) if (p.deg != d)

(E) None

5. Line 8:

(A) return p.next.coef;

(B) return p.deg;

(C) return p.coef;

(D) return p;

(E) None

Method set.

```
1  public void set(int d, double c) {
2    if (...) {
3      Node tmp = new Node(d, c);
4      ...
5      ...
6    } else {
7      Node p = head;
8      Node q = null;
9      while (...) {
10       ...
11       ...
12     }
13     if (...) {
14       ...
15     } else {
16       Node tmp = new Node(d, c);
17       ...
18       ...
19     }
20   }
21 }
```

1. Line 2:

(A) if (d == head.deg){

(B) if (head == null){

(C) if (d < head.deg){

(D) if (d > head.deg){

(E) None

2. Line 4:

(A) head.next = tmp;

(B) head = tmp;

(C) head.deg = tmp.d;

(D) tmp.next = head;

(E) None

3. Line 5:

(A) tmp = head;

(B) head.coef = tmp.coef;

(C) tmp.next = head;

(D) head = tmp;

(E) None

4. Line 9:

(A) while (p != null && p.deg > d){

(B) while (p.deg > d){

40

(C) while (p.deg < d){

(D) while (p != null){

(E) None

5. Line 10:

(A) p = q;

(B) q = p;

(C) p = p.next;

(D) q = p.next;

(E) None

6. Line 11:

(A) p = q;

(B) p = p.next;

(C) q = p;

(D) q = q.next;

(E) None

? 7. Line 13:

(A) if (p != null && p.deg == d){

(B) if (q != null && q.deg > d){

(C) if (p != null){

(D) if (p.deg == d){

(E) None

? 8. Line 14:

(A) p = new Node(d, c);

(B) p.deg = d;

(C) q.coef = c;

(D) p.coef = c; ✓

(E) None

Line 17:

(A) tmp.next = q;

(B) tmp.next = p; ✓

(C) tmp.next = p.next;

(D) p.next = tmp;

(E) None

10. Line 18:

(A) p.next = q;

(B) tmp.next = p;

(C) p.next = tmp;

(D) q.next = tmp;

(E) None

ethod remove.

```
public void remove(int d) {
  if (...) {
    ...
  } else {
    Node p = head.next;
    Node q = head;
    while (...) {
      ...
      ...
    }
    if (...) {
      ...
    }
  }
}
```

1. Line 2:

(A) if (head != null){

(B) if (d == head.deg){

(C) if (head != null && d == head.deg){ ✓

(D) if (head != null && d > head.deg){

(E) None

2. Line 3:

(A) head = null; ✗

(B) head = head.next; ✓

(C) head.next = null;

(D) head.next = head;

(E) None

3. Line 7:

(A) while (p.deg != d){ ✗

(B) while (p != null && p.deg < d){ ✗

(C) while (p != null){

(D) while (p.deg == d){

(E) None ✓

4. Line 8:

(A) p = p.next;

(B) q.next = p;

(C) q = p.next;

(D) q = p; ✓

(E) None

5. Line 9:

(A) p.next = q;

(B) q = p;

(C) p = p.next; ✓

(D) p = q.next;

(E) None

6. Line 11:

(A) if (q != null && p.deg == d){

(B) if (p != null){

(C) if (p.deg == d){

(D) if (p != null && p.deg == d){ ✓

(E) None

7. Line 12:

(A) q = p.next;

(B) p.next = q.next;

(C) q.next = p.next; ✓

(D) p.next = q;

(E) None