	King Saud University	College of Computer and Information Sciences Department of Computer Science
	Data Structures CSC 212	**Midterm Exam - Trimester-1 2022**
	Date: 02/10/2022	Duration: 90 minutes

Question 1 ..... 15 points

Clearly mark one answer for the following:

1. Consider the following code:

```
int x = 0;
for (int i = n-4; i > 5; i--)
    x = x + i;
```

The total number of steps $T(n)$ is:

(A) $2n - 17$ **(B) $2n - 16$** (C) $2n - 15$ (D) $18 - 2n$ (E) None

2. For $f(n) = \frac{n^2 \log(2^n)}{2^{\log(n)}}$, which is the most appropriate? $f(n)$ is:

(A) $O(n^2)$ (B) $O(n^2 \log(n))$ (C) $O(n^3)$ (D) $O(n^3 \log(n))$ (E) None

3. Using stacks to evaluate the expression 7 5 4 * 8 2 3 ^ / 9 * - + which of the following shows the numbers stack (bottom to top) after parsing ^ ?

(A) 7, 5, 4, 8, 2, 3 (B) 7, 20, 8, 2, 3 **(C) 7, 20, 8, 8** (D) 7, 5, 32, 8 (E) None

4. Using stacks to evaluate the expression $1 + (9/3 - (8 - 3\hat{}2) * 4)$ which of the following shows the operators stack (bottom to top) after parsing the first) ?

(A) +, (, -, (, -, ^ (B) +, (, /, - **(C) +, (, -** (D) +, (, / (E) None

5. Consider the following method:

```
public static void m(List<Integer> l, int n) {
  l.findFirst();
  while (!l.last()) {
      Integer x = l.retrieve();
      if (x < n)
          l.remove();
      else {
          l.update(x - n);
          l.findNext();
      }
  }
}
```

Suppose we have a list of integers $l = 8, 4, 2, 5, 3$. Calling $m(l, 5)$ results in l becoming:

(A) $3, 0, 3$ (B) $3, 0$ (C) $-1, -3, 3$ (D) $3, 3$ (E) None

Question 2 ..... 25 points

You are given a stack data structure with push and pop operations, the task is to implement a queue (enqueue and serve operations) using two instances of stack data structure. Please fill in the most appropriate answer to the two methods below.

```
1  public class QueueStack<T>{
2      Stack<T> s1= new LinkedStack();
3      Stack<T> s2= new LinkedStack();
4      public void enqueue(T x)
5      {
6          while (!s1.isEmpty())
7          {
8              ....
9          }
10         ....
11         while (!s2.isEmpty())
12         {
13             ....
14         }
15     }
16     public T serve()
17     {
18         if (....)
19         {
20             ....
21             ....
22         }
23         T x = ...
24         return x;
25     }
26 }
```

1. Line 8:
   - (A) `s2.push(s2.push());`
   - (B) `s2.push(s2.pop());`
   - (C) `s2.push(s1.push());`
   - **(D) `s2.push(s1.pop());`**
   - (E) None
2. Line 10:
   - (A) `s1.push(s2.pop());`
   - (B) `s1.pop();`
   - **(C) `s1.push(x);`**
   - (D) `s2.push(x);`
   - (E) None
3. Line 13:
   - (A) `s2.push(s1.pop());`
   - (B) `s1.push(s1.pop());`
   - **(C) `s1.push(s2.pop());`**
   - (D) `s2.push(s2.pop());`
   - (E) None
4. Line 18:
   - **(A) `if (s1.isEmpty())`**
   - (B) `if (s1.push())`
   - (C) `if (s2.isEmpty())`
   - (D) `if (!s1.isEmpty())`
   - (E) None
5. Line 20:
   - (A) `System.out.println("the number is not found ");`
   - **(B) `System.out.println("Q is Empty");`**
   - (C) `System.out.println("Q is full");`
   - (D) `System.out.println("Q already exist");`
   - (E) None
6. Line 21:
   - (A) **`return 5`**
   - (B) **`return true`**
   - **(C) `return null`**
   - (D) **`return -1`**
   - (E) None
7. Line 23:
   - (A) `T x = s1.push();`
   - (B) `T x = s2.pop();`
   - **(C) `T x = s1.pop();`**
   - (D) `T x = null;`
   - (E) None

Question 3 ..... 25 points

Write the method **public static** `<T> DoubleLinkedList<T> combine(DoubleLinkedList<T> d1, DoubleLinkedList<T> d2)` that receives two non-empty double linked lists. The method should return a new double linked list containing the combination of the elements from list `d1` (in the same order) and the elements from list `d2` (in reversed order).

Example 1. *If* $d1 = A \leftrightarrow B \leftrightarrow C$, $d2 = D \leftrightarrow E \leftrightarrow F$, *then the method should return* $d3 = A \leftrightarrow B \leftrightarrow C \leftrightarrow F \leftrightarrow E \leftrightarrow D$

```
public static <T> DoubleLinkedList<T> combine(DoubleLinkedList<T> d1, DoubleLinkedList<T>
    d2){
  DoubleLinkedList<T> d3 = new DoubleLinkedList<T>();
  ...
```

1. Line 2:
   - (A) **while** `(!d1.last()){`
   - (B) `current=head;`
   - (C) `d1.findFirst();`
   - (D) `d2.findFirst();`
   - (E) None

2. Line 3:
   - (A) **while** `(!d1.last()){`
   - (B) **for** `(int i = 0; i < k; i++){`
   - (C) `d1.findFirst();`
   - (D) `d2.findFirst();`
   - (E) None

3. Line 4:
   - (A) `d3.insert(d2.retrieve());`
   - (B) `d1.current = d1.findNext();`
   - (C) `d3.insert(d1.retrieve());`
   - (D) `d2.current = d2.current.next;`
   - (E) None

4. Line 5:
   - (A) `d2.findNext();}`
   - (B) `d1.findNext();}`
   - (C) `d1.findLast();`
   - (D) `current = current.next;}`
   - (E) None

5. Line 6:
   - (A) `d3.insert(d2.retrieve());`
   - (B) `d1.current = d1.findNext();`
   - (C) `d3.insert(d1.retrieve());`
   - (D) `d2.current = d2.current.next;`
   - (E) None

6. Line 7:
   - (A) `d2.findPrevious();`
   - (B) `d2.findFirst();`
   - (C) **while** `(!d1.last()){`
   - (D) **while** `(!d2.last()){`
   - (E) None

7. Line 8:
   - (A) `current = current.next`
   - (B) `d1.findNext();}`
   - (C) `d2.findNext();}`
   - (D) `Tmp = new Node<T>();`
   - (E) None

8. Line 9:
   - (A) **for** `(int i = 0; i < k; i++){`
   - (B) **while** `(!d2.last()){`
   - (C) **while** `(!d2.first()){`

(D) `d2.findFirst();`

(E) None

9. Line 10:

(A) `d3.insert(d1.retrieve());`

(B) `d3.insert(d2.retrieve());`

(C) `d1.current = d1.findNext();`

(D) `d2.current = d2.current.next;`

(E) None

10. Line 11:

(A) `d2.findNext();}`

(B) `d1.findPrevious();}`

(C) `d2.findPrevious();}`

(D) `current = current.previous;}`

(E) None

11. Line 12:

(A) `d3.insert(d1.retrieve());`

(B) `d3.insert(d2.retrieve());`

(C) `d1.current = d1.findNext();`

(D) `d2.current = d2.current.next;`

(E) None

12. Line 13:

(A) **return** `d1;`

(B) **return** `d2;`

(C) **return** `d1+d2;`

(D) **return** `d3;`

(E) None

Question 4 ..... 35 points

We want to implement the interface `QS` that adds and removes elements from the queue. You will need to implement `enqueue` operation which adds an element if it does not exist. If it is already in the queue, its `count` should be incremented. You will also need to implement a `remove` operation. When removing a given element from the queue, the element's count should be decremented. When an element's count becomes 0, the corresponding node should be removed from the queue unless it is the only remaining node (i.e., the queue must contain at least one element). Complete the class `QueueSet` below.

Example 2. *The queue after adding the following elements:*

```
public static void main(String[] args) {
  QueueSet<String> q = new QueueSet<String>();
  q.enqueue("A");
  q.enqueue("B");
  q.enqueue("C");
  q.enqueue("B");
  q.enqueue("C");
}
```

will look like the following (the numbers indicate the count of each element):

```
(A, 1) --> (B, 2) --> (C, 2)
```

After performing the following `remove(''B'')` operation, the queue should look like the following:

```
(A, 1) --> (B, 1) --> (C, 2)
```

After performing the following `remove(''A'')` operation, the queue should look like the following:

```
(B, 1) --> (C, 2)
```

```
public interface QS<T> {
  public int length();
  public boolean full();
  // Add dat to the queue
  public void enqueue(T dat);
  // Serve dat from the queue
  public T remove(T dat);
}
```

```
public class Node<T> {
  public T data;
  public int count;
  public Node<T> next;
  public Node(T t) {
    data = t;
    next = null;
    count = 1;
  }
}
```

```
public class QueueSet<T> implements QS<T> {
  private Node<T> head, tail;
  private int size;
  public QueueSet() {
    head = tail = null;
    size = 0;
  }
  public int length() {
    return size;
  }
  public boolean full() {
    return false;
  }
}
```

- Method `enqueue`.

```
1  public void enqueue(T dat) {
2    if (...) {
3      ...
4    }
5    else {
6      ...
7      while (...) {
8        if (...) {
9          ...
10         ...
11       }
12       ...
13     }
14     if (...) {
15       ...
16       ...
17     }
18   }
19   size ++;
20 }
```

1. Line 2:
   - (A) `!full()`
   - (B) `head == tail`
   - **(C) `tail == null`**
   - (D) `tail != head`
   - (E) None
2. Line 3:
   - **(A) `head = tail = new Node<T>(dat);`**
   - (B) `tail = new Node<T>(dat);`
   - (C) `head = new Node<T>(dat);`
   - (D) `Node<T> node = new Node<T>(dat);`
   - (E) None
3. Line 6:
   - **(A) `Node<T> tmp = head;`**
   - (B) `Node<T> tmp = head.next.next;`
   - (C) `Node<T> tmp = head.next;`
   - (D) `Node<T> tmp = tail;`
   - (E) None
4. Line 7:
   - (A) `tmp.next == null`
   - **(B) `tmp != null`**
   - (C) `tmp.next != null`
   - (D) `tmp == null`
   - (E) None
5. Line 8:
   - (A) `tmp.data == dat`
   - (B) `tmp.equals(dat)`
   - **(C) `tmp.data.equals(dat)`**
   - (D) `tmp == dat`
   - (E) None
6. Line 9:
   - (A) `tmp.count = 1;`
   - **(B) `tmp.count ++;`**
   - (C) `tmp.count = tail.count;`
   - (D) `tmp.count = head.count;`

(E) None

7. Line 10:

(A) `tmp.data = dat;`

(B) `tmp.data = tmp.next.data;`

(C) **`break;`**

(D) **`return;`**

(E) None

8. Line 12:

(A) **`tmp = tmp.next;`**

(B) `tmp = tmp.next.next;`

(C) `tmp = tail.next;`

(D) `tmp = head.next;`

(E) None

9. Line 14:

(A) **`tmp == null`**

(B) `tmp.next == null`

(C) `tmp.equals(null)`

(D) `tmp != null`

(E) None

10. Line 15:

(A) `head.next = new Node<T>(dat);`

(B) `tmp = new Node<T>(dat);`

(C) `tail = new Node<T>(dat);`

(D) **`tail.next = new Node<T>(dat);`**

(E) None

11. Line 16:

(A) `tail = tmp;`

(B) `head = head.next;`

(C) `tail = head;`

(D) **`tail = tail.next;`**

(E) None

- Method `remove`.

```
1  public T remove(T dat) {
2    if (...)
3      ...
4    Node<T> tmp = ...;
5    Node<T> prev = ...;
6    while (...) {
7      if (...) {
8        ...
9        if (...) {
10         if (...)
11           ...
12         else
13           ...
14       }
15       size--;
16       ...;
17     }
18     ...;
19     ...;
20   }
21   ...;
22 }
```

1. Line 2:

(A) `full()`

(B) `!full()`

(C) `size > 0`

(D) **`size == 1`**

(E) None

2. Line 3:

(A) **`return null;`**

(B) **`return new Node<T>(dat);`**

(C) `Node<T> node = new Node<T>(dat);`

(D) **`return`** `head.next;`

(E) None

3. Line 4:

(A) **`null`**

(B) **`head`**

(C) **`new`** `Node<T>(dat)`

(D) `tail`

(E) None

4. Line 5:

(A) **`null`**

(B) `head.next`

(C) `new Node<T>(dat)`

(D) `tail`

(E) None

5. Line 6:

(A) `tmp == null`

(B) `tmp.next != null`

(C) `prev != null`

(D) `tmp != null`

(E) None

6. Line 7:

(A) `tmp.data == dat`

(B) `prev.data = dat`

(C) `prev.data.equals(dat)`

(D) `tmp.data.equals(dat)`

(E) None

7. Line 8:

(A) `tmp.count ++;`

(B) `tmp.count --;`

(C) `prev.count --;`

(D) `tmp.count = 0;`

(E) None

8. Line 9:

(A) `tmp.count <= 1`

(B) `prev.count == 0`

(C) `tmp.count >= 1`

(D) `tmp.count == 0`

(E) None

9. Line 10:

(A) `tmp == null`

(B) `tmp == head`

(C) `tmp.next == tail`

(D) `tmp == tail`

(E) None

10. Line 11:

(A) `tmp = head;`

(B) `tmp = head.next;`

(C) `tail = tmp;`

(D) `head = head.next;`

(E) None

11. Line 13:

(A) `prev.next = tmp.next.next;`

(B) `prev = tmp.next;`

(C) `tmp = prev.next;`

(D) `prev.next = tmp.next;`

(E) None

12. Line 16:

(A) `return tmp.data;`

(B) `return prev.data;`

(C) `return tmp;`

(D) `return null;`

(E) None

13. Line 18:

(A) `tmp = prev;`

(B) `tmp = prev.next;`

(C) `prev = tmp;`

(D) `prev = tmp.next;`

(E) None

14. Line 19:

(A) `tmp = tmp.next;`

(B) `tmp = prev;`

(C) `prev = null;`

(D) `tmp = prev.next;`

(E) None

15. Line 21:

(A) `return tmp.data;`

(B) `return null;`

(C) `return head.data;`

(D) `return prev.data;`

(E) None