	King Saud University	College of Computer and Information Sciences Department of Computer Science
	Data Structures CSC 212	**Midterm Exam - Fall 2021**
	Date: 20/10/2021	Duration: 90 minutes

Guidelines

•No calculators or any other electronic devices are allowed in this exam.

Student ID:	Name:
Section:	Instructor:

1	2	3	4	Total

Question 1 ..... 15 points

The operation / below indicates **integer division**.

1. Trace the execution of the following postfix expression: `9 3 1 5 - * 8 4 3 / 1 + - + < 25 5 % 0 > ||`. Show the content of the data structure(s) after parsing each operation. Stacks are shown from bottom to top.

(a) After - :
- (A) 9 3 1 5
- (B) 9 3
- (C) **9 3 -4**
- (D) 9 3 4
- (E) None

(b) After * :
- (A) 9 3 5
- (B) **9 -12**
- (C) 9 3
- (D) 9 12
- (E) None

(c) After / :
- (A) 9 3 5 8 0
- (B) 9 3 8 4
- (C) 9 12 8 0
- (D) **9 -12 8 1**
- (E) None

(d) After + :
- (A) **9 -12 8 2**
- (B) 9 3 5 8
- (C) 9 3 12
- (D) 9 12 8
- (E) None

(e) After - :
- (A) 9 -12 -6
- (B) 9 3 3
- (C) 9 -9
- (D) **9 -12 6**
- (E) None

(f) After + :
- (A) **9 -6**
- (B) 9 6
- (C) 0
- (D) 9 -18
- (E) None

(g) After < :
- (A) 9 T
- (B) 9 F
- (C) T
- (D) **F**
- (E) None

(h) After % :
- (A) 9 T 5
- (B) **F 0**
- (C) T 0
- (D) F 5
- (E) None

(i) After > :
- (A) 9 T F
- (B) T 0 T
- (C) F 0 F
- (D) **F F**
- (E) None

(j) After || :
- (A) T F
- (B) T
- (C) 0
- (D) **F**
- (E) None

2. Trace the execution of the following infix expression: `4+(9-5)/3*(2-(3+5))`. Draw the content of the data structure(s) after parsing each operation.

(a) After + :

(A) 4 , + (
(B) 4 , (
(C) 4 , +
(D) 4 4 , +
(E) None

(b) After - :

(A) 4 9 , +
(B) 13 , (-
(C) 13 , -
(D) 4 9 , + (-
(E) None

(c) After / :

(A) 4 4 , + /
(B) 4 9 5 , (-) /
(C) 4 , +
(D) 4 9 5 , + (- /
(E) None

(d) After * :

(A) 4 0 , + *
(B) 4 1 , + *
(C) 4 , +
(D) 4 4 3 , + * /
(E) None

(e) After - :

(A) 4 0 2 , + * (-
(B) 4 3 , + (/ -
(C) 4 1 2 , + * (-
(D) 4 4 3 , + * /
(E) None

(f) After + :

(A) 4 0 2 3 , + * (- (+
(B) 4 3 2 3 , + (/ -) +
(C) 4 4 3 , + * /
(D) 4 1 2 3 , + * (- (+
(E) None

(g) After `$` :

(A) -2 , $
(B) 2 , $
(C) 2 ,
(D) -2 ,
(E) None

Question 2 ..... 25 points

Given a queue and an integer `k`, write the method `reverseK` which reverses the order of the first `k` elements of `q` leaving the other elements in the same order. If `k` is invalid, then `q` is unchanged.

Example 1. *If `q` = (1, 2, 3, 4, 5, 6, 7, 8, 9), and `k` = 5, then after calling `reverseK(q, k)`, `q` = (5, 4, 3, 2, 1, 6, 7, 8, 9).*

Please complete the method below:

```
1  public static<T> void reverseK(Queue<T> q,
       int k) {
2    if (...)
3      return;
4    if (...)
5      return;
6    Stack<T> s = new LinkedStack<T>();
7    for (...)
8      ...
9    while (...)
10     ...
11   for (...)
12     ...
13 }
```

1. Line 2:

(A) `if (k <= q.length())`
(B) `if (q.length()== 0 || k > q.length())`
(C) `if (q.length()== 0 && k > q.length())`
(D) `if (k < q.length())`
(E) None

2. Line 4:

(A) `if (k < 0)`
(B) `if (k == 0)`
(C) `if (k == -1)`
(D) `if (k <= 0)`

(E) None

3. Line 7:

(A) `for (int i = 0; i < q.length()- k; i++){`

(B) `for (int i = 0; i < k; i++)){`

(C) `for (int i = 0; i < q.length(); i++){`

(D) `for (int i = 0; i < q.length()+ k; i++){`

(E) None

4. Line 8:

(A) `q.enqueue(s.pop());`

(B) `q.enqueue(q.serve());`

(C) `q.enqueue();`

(D) `s.push(q.serve());`

(E) None

5. Line 98:

(A) `while (q.length()> 0)`

(B) `while (q.length()< k)`

(C) `while (q.length()< k)`

(D) `while (!s.empty())`

(E) None

6. Line 10:

(A) `s.push(q.serve());`

(B) `s.push(s.pop());`

(C) `q.enqueue(s.pop());`

(D) `s.push();`

(E) None

7. Line 11:

(A) `for (int i = k; i < q.length()+ k; i++){`

(B) `for (int i = 0; i < q.length()- k; i++){`

(C) `for (int i = 0; i < q.length(); i++){`

(D) `for (int i = 0; i < q.length()+ k; i++){`

(E) None

8. Line 12:

(A) `q.enqueue(q.serve());`

(B) `s.push();`

(C) `s.push(q.enqueue());`

(D) `q.enqueue(s.pop());`

(E) None

Question 3 ..... 25 points

Write the method `checkListEndsSymmetry` that receives a non-empty double linked list and a positive integer `k`. The method checks if the double linked list has identical `k` elements going forward from the first element and backwards from the last one. The method returns **true** if they are identical, and **false** otherwise. Assume that `k` is strictly less than the length of the list.

Example 2. *If* $dl = A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow B \leftrightarrow A$ *and* $k = 2$*, then the method should return* **true**. *If* $k = 3$*, it should return* **false**, *since* C *does not equal* D.

The method signature is: **public static** `<T>` **boolean** `checkListEndsSymmetry(DoubleLinkedList<T> dl,` **int** `k).`

```
public static<T> boolean checkListEndsSymmetry(DoubleLinkedList<T> dl, int k){
  T[] l = (T[]) new Object[k];
  dl.findFirst();
  ...
```

1. Line 3:

(A) `while (!dl.last()){`

(B) `for (int i = 0; k < i; i++){`

(C) `for (int i = 0; i < k; i++){`

(D) `for(int i = 0; i <= k; i++){`

(E) None

2. Line 4:

   (A) `l[i] = dl.retrieve()`
   (B) `dl.findNext();`
   (C) `dl.remove();`
   (D) `l[i] = dl.current.data);`
   (E) None

3. Line 5:

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

4. Line 6:

   (A) **while**`(dl.current!=null){`
   (B) **while**`(!dl.last()){`
   (C) `dl.findLast();`
   (D) **while**`(dl.current!=null){`
   (E) None

5. Line 7:

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

6. Line 8:

   (A) **while** `(!dl.last()){`
   (B) **for**`(int i = k; i >= 0; i--){`
   (C) **for** `(int i = 0; i < k; i++){`
   (D) **if** `(!dl.last())`
   (E) None

7. Line 9:

   (A) **if** `(l[i].equals(dl.retrieve()))`
   (B) **if** `(!l[i].equals(dl.retrieve()))`
   (C) **return false;**
   (D) **if** `(l[i]!=dl.retrieve())`
   (E) None

8. Line 10:

   (A) **return true;**
   (B) **return** `!l[i].equals(dl.retrieve());`
   (C) **break;**
   (D) **return false;**
   (E) None

9. Line 11:

   (A) `dl.findPrevious();}`
   (B) `dl.findNext();}`
   (C) **return false;}**
   (D) **return true;}**
   (E) None

10. Line 12:

    (A) **return** `l[k-1].equals(dl.retrieve());`
    (B) **return** `l[1].equals(dl.retrieve());`
    (C) **return** `!l[k-1].equals(dl.retrieve());`
    (D) **return true;**
    (E) **return false;**

Question 4 . . . . . 35 points

We want to write a linked implementation of the data structure `PairList` which stores unique pairs `(a, b)` of elements of the same type:

```
public interface PairList<T> {
  // Return true if the list is empty, false otherwise.
  boolean empty();
  // Return true if the list is full, false otherwise.
  boolean full();
  // Print the list
  void print();
  // Insert the pair (a, b) if does not already exit and return true. If (a,b) already
      exists, the list is not changed and the method returns false.
  boolean insert(T a, T b);
  // Return true if the pair (a, b) exists, false otherwise.
  boolean exists(T a, T b);
  // Return a list containing all the second elements of pairs starting with a stored
      according to insertion order.
  List<T> getSecond(T a);
  // Remove all pairs starting with a.
  void remove(T a);
}
```

Example 3. *The output of the following program:*

```
public static void main(String[] args) {
  PairList<Integer> pl = new LinkedPairList<Integer>();
  pl.insert(1, 2);
  pl.insert(1, 2);
  pl.insert(3, 3);
  pl.insert(1, 4);
  pl.insert(3, 5);
  pl.insert(2, 1);
  pl.insert(1, 7);
  print(pl.getSecond(3));
  System.out.println("----------------");
  pl.remove(1);
  pl.print();
}
```

is:

```
3
5
----------------
3 3
3 5
2 1
```

Complete the class `LinkedPairList` below.

```
class PNode<T> {
  public T a, b;
  public PNode<T> next;
  public PNode(T a, T b) {
    this.a = a;
    this.b = b;
    next = null;
  }
}
public class LinkedPairList<T> implements PairList<T> {
  private PNode<T> head, tail;
  public LinkedPairList() {
```

```
    tail = head = null;
  }
  public boolean empty() {
    return head == null;
  }
  public boolean full() {
    return false;
  }
  public void print() {
    PNode<T> p = head;
    while (p != null) {
      System.out.println(p.a + "\t" + p.b);
      p = p.next;
    }
  }
  public boolean exists(T a, T b) {
    PNode<T> p = head;
    while (p != null) {
      if (a.equals(p.a) && b.equals(p.b))
        return true;
      p = p.next;
    }
    return false;
  }
```

Method `insert`.

```
1  public boolean insert(T a, T b) {
2    if (...)
3      return ...;
4    PNode<T> tmp = new PNode<T>(a, b);
5    if (head == null)
6      ...
7    else {
8      ...
9      ...
10   }
11   ...
12 }
```

1. Line 2:
   - (A) **if** `(exists(a, b))`
   - (B) **if** `(a.equals(head.a)&& b.equals(head.b))`
   - (C) **if** `(!exists(a, b))`
   - (D) **if** `(a.equals(head.a))`
   - (E) None
2. Line 3:
   - (A) **return false;**
   - (B) **return** `a.equals(head.a);`
   - (C) **return** `b.equals(head.b);`
   - (D) **return true;**
   - (E) None
3. Line 6:
   - (A) `head = tmp;`
   - (B) `tail = tmp;`
   - (C) `tail = head = tmp;`
   - (D) `tail = head =` **null**`;`
   - (E) None
4. Line 8:
   - (A) `head.next = tmp;`
   - (B) `tmp.next = head;`
   - (C) `tail.next = tmp;`
   - (D) `tmp.next = tail;`
   - (E) None
5. Line 9:
   - (A) `tmp = tail;`
   - (B) `head = tmp;`
   - (C) `tail = tmp;`
   - (D) `tmp = head;`
   - (E) None
6. Line 11:
   - (A) **return true;**
   - (B) **return** `head == null;`

(C) `return false;`

(D) `return tail == null;`

(E) None

Method `getSecond`.

```
1  public List<T> getSecond(T a) {
2    List<T> l = new LinkedList<T>();
3    PNode<T> p = ...
4    while (...) {
5      if (...)
6        ...
7      ...
8    }
9    ...
10 }
```

1. Line 3:

   (A) `PNode<T> p = tail;`

   (B) `PNode<T> p = null;`

   (C) `PNode<T> p = head.next;`

   (D) `PNode<T> p = tail.next;`

   (E) **None**

2. Line 4:

   (A) `while (!l.last()){`

   (B) `while (p != tail){`

   (C) `while (p.next != null){`

   (D) `while (p != null){`

   (E) None

3. Line 5:

   (A) `if (a.equals(p))`

   (B) `if (a.equals(p.b))`

   (C) `if (a.equals(p.a))`

   (D) `if (a.equals(p.next.b))`

   (E) None

4. Line 6:

   (A) `l.insert(p.a);`

   (B) `l.insert(p.next.b);`

   (C) `l.insert(p);`

   (D) `l.insert(p.b);`

   (E) None

5. Line 7:

   (A) `p++;`

   (B) `p = head;`

   (C) `p = head;`

   (D) `l.findNext();`

   (E) **None**

6. Line 9:

   (A) `return l.head;`

   (B) `return l.retrieve();`

   (C) `return l;`

   (D) `return new LinkedList<T>();`

   (E) None

Method `remove`.

```
1  public void remove(T a) {
2    while (...)
3      ...
4    if (...)
5      ...
6    else {
7      PNode<T> q = head;
8      PNode<T> p = ...
9      while (...) {
10       while (...)
11         ...
12       q.next = p;
13       ...
14       ...
15       if (...)
16         ...
17     }
18   }
19 }
```

1. Line 2:

   (A) `while (a.equals(head.a)&& head != tail)`

   (B) `while (head != null)`

(C) `while (head != null && a.equals(head.a))`

(D) `while (a.equals(head.a))`

(E) None

2. Line 3:

(A) `p.a = p.b = null;`

(B) `tail = tail.next;`

(C) `head = null;`

(D) `head = head.next;`

(E) None

3. Line 4:

(A) `if (head.next == null)`

(B) `if (head == tail)`

(C) `if (tail == null)`

(D) `if (head == null)`

(E) None

4. Line 5:

(A) `tail = head;`

(B) `head = null;`

(C) `tail = null;`

(D) `tail = head.next;`

(E) None

5. Line 8:

(A) `PNode<T> p = tail;`

(B) `PNode<T> p = null;`

(C) `PNode<T> p = head.next;`

(D) `PNode<T> p = tail.next;`

(E) None

6. Line 9:

(A) `while (head != null){`

(B) `while (p != null){`

(C) `while (p != tail){`

(D) `while (q != null){`

(E) None

7. Line 10:

(A) `while (p != null)`

(B) `while (p != null && a.equals(p.a))`

(C) `while (a.equals(p.a))`

(D) `while (q != null && a.equals(p.a))`

(E) None

8. Line 11:

(A) `p = p.next;`

(B) `q.next = p.next;`

(C) `q = q.next;`

(D) `p.next = q.next;`

(E) None

9. Line 13:

(A) `head = q;`

(B) `tail = p;`

(C) `tail = q;`

(D) `head = p;`

(E) None

10. Line 14:

(A) `q = p;`

(B) `q = tail;`

(C) `p = head;`

(D) `p = q;`

(E) None

11. Line 15:

(A) `if (head == null)`

(B) `if (p != null)`

(C) `if (q != null)`

(D) `if (p == tail)`

(E) None

12. Line 16:

(A) `q.next = p.next;`

(B) `q = q.next;`

(C) `p = p.next;`

(D) `tail.next = null;`

(E) None