|  |  |
| --- | --- |
| **Serial No.** |  |

University of Bahrain

College of Information Technology

Department of Computer Science

Second Semester, 2018-2019

ITCS214 / ITCS215/ ITCS216 (Data Structures)

#### Final Exam

Date: 09/06/2019 Time: 11:30 - 13:30

|  |
| --- |
|  |

**STUDENT NAME**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

**STUDENT ID #**

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

**COURSE Code: ITCS214 ITCS215 ITCS216**

|  |
| --- |
|  |

**SECTION #**

NOTE: THERE ARE ELEVEN**(11) PAGES** IN THIS TEST

ONLY ONE SOLUTION WILL BE CONSIDERED FOR EACH QUESTION

|  |  |  |  |
| --- | --- | --- | --- |
| QUESTION # | MARKS | | COMMENTS |
| 1 | 10 |  |  |
| 2 | 14 |  |  |
| 3 | 14 |  |  |
| 4 | 16 |  |  |
| 5 | 15 |  |  |
| 6 | 11 |  |  |
| TOTAL | 80 |  |  |

**Question 1 [10 Marks]**

Write a method called **isProper** in the class **KWLinkedList** (a class for doubly linked list as discussed in the lectures) that accepts one parameter called **elem** of type **int**. If **elem** exists in the list and the node before **elem** has data less than **elem** and the node after **elem** has data greater than **elem**, return true. In all other cases return false. If the list is empty or does not contain **elem** return false. If the **elem** is in the first node or last node, then also return false.

**Note: Do not use iterators.**

**Example:**

elem: 35

Iist (before method call): 15 75 20 **35** 40 60 45

Method heading:

public boolean isProper( E elem )

{

**Question 2 [6 + 8 Marks]**

1. **[6 Marks]** Consider the generic class called **ArrayStack** as discussed in the lectures, having following data fields :

private E[ ] theData; // array to store stack elements

private int topOfStack = -1; // Index to top of stack, initially empty stack.

private static final int INITIAL\_CAPACITY = 10;

Write a method called **replaceTop** to be included in this class **ArrayStack**. If stack is not empty, then the method replaces the top element of the stack by parameter **item** and returns the old value at the top. If the stack is empty, return null.

Do not call any other method of the class **ArrayStack** in your method.

**Example:**

top

**stack (Before method call)**: 5, 20, 10, 50, 15, 30

**item**: 7

top

**stack (After method call)**: 7, 20, 10, 50, 15, 30

Method will return 5.

**Method Heading**:

public E replaceTop(E item )

1. **[8 Marks]** Write a method called **samePosition** in a class called **StackExt** that receives two parameters **st1** and **st2** of type **ArrayStack** and a third parameter **pos** of type int. The method returns true, if stack **st1** and **st2** contains the same value in position **pos**. Otherwise, the method returns false. Assume that both stacks **st1** and **st2** contains same number of elements.

Assume that the position of the top element is 0 and increases by 1 for each subsequent element. The elements of both stacks **st1** and **st2** must remain in the original relative positions in the stacks at the end of the method.

Use common stack operations only such as **push**, **pop**, **peek**, **isEmpty,** constructor and copy constructor. You can create local objects of type **ArrayStack** in your method.

Example1:

top top

**st1**: 4 5 3 7 8 **st2**: 5 9 4 7 1

If the pos is 3, then both stacks contain 7 in position 3. The method returns true.

If the pos is 1, then **st1** contains 5 and **st2** contains 9 in position 1. The method returns false.

public class StackExt

{

public boolean sameposition(ArrayStack<Integer> st1,

ArrayStack<Integer> st2, int pos)

{

**Question 3 [8 + 6 Marks]**

1. **[8 Marks]** Write a method called **compareQueues** to be included in class **QueueEx**. The method accepts two parameters **q1** and **q2** type **ArrayQueue.** Assume that both queues contain same number of elements. The method will compare the contents of q1 and q2 in the corresponding positions and will return the number of elements which are same in the same position. The elements of both queues **q1** and **q2** must remain in the original relative positions in the queues at the end of the method.

Assume that class **ArrayQueue** is available for use. Use common queue operations only such as **offer**, **poll**, **peek, isEmpty,**constructor and **copy constructor**.

**Example :**

**Before method call**:

front rear

**q1**: 10 15 30 20 25 12

q2: 10 16 18 20 25 9

The method will return 3 as 3 elements are same.

|  |
| --- |
|  |
|  |

public class QueueEx

{

public static int compareQueues(ArrayQueue<Integer> q1,

ArrayQueue<Integer> q2)

{

1. **[6 Marks]** Show the contents of the queue q at three different places, asked in the question, with the help of suitable diagrams (circular array), showing the locations of front and rear clearly.

public class QueueTest

{

public static void main(String [ ] args)

{

ArrayQueue<Character> q = new ArrayQueue<Character>(5);

// initial capacity = 5

q.offer(‘A’);

q.offer(‘B’);

q.offer(‘C’);

q.offer(‘D’);

//Diagram 1

q.poll();

q.poll();

//Diagram 2

q.offer(‘E’);

q.offer(‘F’);

//Diagram 3

}

}

**Question 4 [6 + 10 Marks]**

**(A) [6 Marks]** Write a recursive private method called **countTwoChilds** to be included in class **BinaryTree** as discussed in the lectures. The method counts and returns the number of nodes having two child nodes in the binary tree.

This method is called from a public method **countTwoChildsBT**, given as follows:

public int countTwoChildsBT(){

return countTwoChilds(root);

}

Method heading:

private int countTwoChilds(Node<E> node)

**(B)Consider the following binary search tree:**

1. **[5 Marks]** Find the sequence of nodes, if the binary tree is traversed in **inorder traversal.**
2. **[2 Marks]** Redraw the above binary search tree after inserting the nodes with keys **38** and **52** consecutively, in the original binary search tree.
3. **[3 Marks]** Redraw the above binary search tree after deleting the nodes with keys **35** and **60** consecutively, from the original binary search tree.

**Question 5 [10 + 5 Marks]**

1. **[10 Marks]** For the following graph, find the **adjacency matrix** and **adjacency lists** representation of the graph:

**0**

**1**

**2**

**4**

**3**

1. **[5 Marks]** For the following graph, find the sequence of vertices visited in the graph, if the graph is traversed using **Breadth-FirstTraversal** algorithm. Assume vertex 0 as the starting vertex.

**1**

**0**

**3**

**2**

**4**

**6**

**7**

**5**

**Question 6 [7 + 4 Marks]**

Given the following input keys: 43, 44, 62, 17, 35, 72, 67, 32, and

Hash function h(X) = X mod 11, HTSize = 11.

1. Obtain the resulting hash table when open addressing with **quadratic probing** is used to resolve collisions.
2. Obtain the resulting hash table when **chaining** is used to resolve collisions.