University of Bahrain

College of Information Technology

Department of Computer Science

Second Semester, 2017-2018

**ITCS214 / ITCS215 / ITCS216 (Data Structures)**

#### Test II

Date: 03/05/2018 Time: 17:00 - 18:15

**STUDENT NAME** (Uppercase characters)

|  |
| --- |
|  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | 0 |  |  |  |  |  |  |

**STUDENT ID #**

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

**COURSE (Select one): ITCS214 ITCS215 ITCS216**

|  |
| --- |
|  |

**SECTION #**

NOTE: THERE ARE SIX **(6) PAGES** IN THIS TEST

ONLY ONE SOLUTION WILL BE CONSIDERED FOR EACH QUESTION

|  |  |  |  |
| --- | --- | --- | --- |
| QUESTION # | MARKS | | COMMENTS |
| 1 | 10 |  |  |
| 2 | 16 |  |  |
| 3 | 14 |  |  |
| TOTAL | 40 |  |  |

**Question 1 [10 Marks] [Double linked list]**

Write a method called **findAll** to be included in class **KWLinkedList** that finds all nodes in “this” list with their **data** values less than **item,** passed as the first parameter and inserts these values in a new linked list, called **list2** passed as the second parameter to the method.

If “this” list is empty or no values less than **item** exists in the list, then the method returns false, otherwise it returns true. Assume that **list2** is initially empty.

Function method:

**public boolean findAll(E item, KWLinkedList<E> list2)**

Example:

**Before method call**

**“this” list**: **10 5** 25 **12**  18 **7** 40 50 **11**

**list2: (empty)**

Item: 15

**After method call**

**“this” list**: **10 5** 25 **12**  18 **7** 40 50 **11**

**list2: 10 5 12 7 11**

**You can use ListIterator and its methods in your method to access elements of “this” list. Do not call any other method of class KWLinkedList in your method.**

**Answer Q1**

public boolean findAll(E item, KWLinkedList<E> list2)

{

if(head == null)

return false;

E item1;

ListIterator<E> itr = listIterator();

while(itr.hasNext())

{

item1 = itr.next();

if(item1.compareTO(item) < 0) // item1 < item

{

Node<E> node = new Node<E>(item1); //create node

if(list2.head == null){//insert in an empty list2

list2.head = node;

list2.tail = node;

(list2.size)++;

}

else{ // insert new node at the end of list2

node.prev = list2.tail;

list2.tail.next = node;

list2.tail = node;

(list2.size)++;

}

}//end if

}// end while

if(list2.size != 0)

return true;

else

return false;

}

**Question 2 [10 + 6 Marks] [Stacks]**

1. Write a method called **compareStacks** in a class called **StackExample** that receives two objects **st1**and **st2** of type **ArrayStack** as parameters. Assume that both stacks **st1** and **st2** have the same number of elements and number of elements is even. If the first half (the upper part) of **st1** is equal to the second half (the lower part) of **st2,** the method returns true. In all other cases the method returns false.

Example:

top

***st1*** : **10 5 15 7** 25 20 30 9

***st2*** : 18 20 25 11 **10 5 15 7**

The method will return **true** in the above case.

Use common stack operations only such as ***push***, ***pop***, ***peek,*** ***isEmpty*** and **copy constructor**.

**Answer Q2(A)**

public class StackExample

{

public static<E> boolean compareStacks(ArrayStack<E> st1,

ArrayStack<E> st2)

{

ArrayStack<E> st3 = new ArrayStack<E>();

ArrayStack<E> st4 = new ArrayStack<E>();

int count = 0; // number of elements in st1 and st2

E item1, item2;

while(!st1.isEmpty())

{

st3.push(st1.pop());

count++;

}

while(!st2.isEmpty())

st4.push(st2.pop());

// push top half of st3 back in st1 as

// they are not used in comparison

for(int i = 0; i < count/2; i++)

st1.push(st3.pop());

boolean equal = true;

// Compare second half of st3 with top half of st4

while(!st3.isEmpty())

{

item1 = st3.peek();

item2 = st4.peek();

if(!item1.equals(item2))

equal = false;

st1.push(st3.pop());

st2.push(st4.pop());

}

// push remaining elements of st4 in st2

for(int i = 0; i < count/2; i++)

st2.push(st4.pop());

return equal;

}

}

1. Consider the following postfix expression. Use stack to evaluate it and show all the push and pop operations by clearly drawing the stack status.

12 24 6 / + 5 3 – 10 \* –

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

Consider the generic class called **ArrayQueue**as discussed in the lectures, having following data fields :

private E [ ] theData; // The data array

private int size; // The current size

private int capacity ; // The current capacity

private int front; // Index of the front of the queue.

private int rear; // Index of the rear of the queue.

Write following methods to be included in this class:

1. **poll:** Deletes the element at **front** of the queue and returns the item deleted. If queue is empty returns null.

Method heading: **public E poll()**

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

1. **trimQueue:** Deletes all duplicate elements of the queue, i.e., if an **item** appears more than once in the queue, then keep the first occurrence of the **item** and delete all remaining occurrences of the **item** from the queue.

Method heading: **public void trimQueue()**

**Hint**: Create an array **newData** of the same capacity as **theData**. Copy elements (in a loop) from **theData** (staring at location **front**) into **newData** (starting at location 0), if the element is not already in **newData**. After that assign **newData** to **theData** and assign appropriate values to front, rear and size.

Example:

Queue before method call: 10 3 5 8 3 3 12 10 15

Queue after method call: 10 3 5 8 12 15

**Answer Q3(A)**

public E poll()

{

if (size == 0)

return null;

E result = theData[front];

front = (front + 1) % capacity;

size--;

return result;

}

[Empty page]

**Answer Q3(B)**

public void trimQueue()

{

E[] newData = (E[]) new Object[capacity];

int j = front; // index in theData

int i = 0,; // index in newData

for(int count = 0; count < size; count++)

{

boolean found = false;

for(int k = 0; k < i; k++)

{

if(newData[k] == theData[j])

found = true;

break;

}

if(!found)

{

newData[i] = theData[j];

j = (j + 1) % capacity;

i++;

}

}

size = i;

front = 0;

rear = size - 1;

theData = newData;

}