University of Bahrain

College of Information Technology

Department of Computer Science

First Semester, 2017-2018

**ITCS214 / ITCS215 (Data Structures)**

#### Test II

Date: 07/12/2017 Time: 16:00 - 17:15

**STUDENT NAME** (Uppercase characters)

|  |
| --- |
|  |

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

**STUDENT ID #**

|  |
| --- |
|  |

**SECTION #**

NOTE: THERE ARE FIVE **(5) 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 **removeWithCondition** to be included in the class **KWLinkedList** that accepts two parameters **item** and **pItem** of type E. The method deletes the node having its data as **item,** if preceded by a node having its data as **pItem**, and returns true. If not found then it returns false.

The method heading is

**public boolean removeWithCondition(E item, E pitem).**

public boolean removeWithCondition(E item, E pitem)

{

if (head == null) return null;

ListIterator<E> iter = listIterator(0);

E a;

while(iter.hasNext())

{

a=iter.next();

if(a.equals(pitem))

{

if(! iter.hasNext())

{

a= iter.next();

if(a.equals(item))

{

iter.remove();

return true;

}

else

{ **/\* we should not skip the last item because it might have pitem \*/**

int b=iter.nextIndex();

iter=ListIterator(b-1);

}

else return false;

}

}

}

}

**Another Solution**

public boolean removeWithCondition(E item, E pitem)

{

if (head == null) return null;

ListIterator<E> iter = listIterator(0);

ListIterator<E> iter1= listIterator(1);

E a,b;

while(iter1.hasNext())

{

a=iter1.next();

b=iter.next();

if((a.equals(item) && b.equals(pitem))

{ iter1.remove(); return true;}

}

return false;

}

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

1. Write a method called **splitStack** in a class called **StackApplication** that receives two objects **st1**and **st2** of type **ArrayStack** as parameters. Assume that initially **st1** in nonempty and **st2** is empty. The method splits the stack **st1** by deleting the alternative elements from the stack **st1** and inserting them in stack **st2**.

Example:

Before method call:

***st1*** : 1 5 10 8 15 25 20 30 40

***st2*** : (empty)

After method call:

***st1*** : 1 10 15 20 40

***st2*** : 5 8 25 30

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

public class StackApplication

{

public static void splitStack(ArrayStack<Integer> st1,

ArrayStack<Integer> st2)

{

public class StackApplication

{

public static void splitStack(ArrayStack<Integer> st1, ArrayStack<Integer> st2)

{

ArrayStack<Integer> st11 = new ArrayStack<Integer>();

ArrayStack<Integer> st2reverse = new ArrayStack<Integer>();

while(! st1.isEmpty())

{

st11.push(st1.pop());

if(! st1.isEmpty())

st2reverse.push(st1.pop());

}

while (! st11.isEmpty())

{

St1.push(st11.pop());

}

while(! st2reverse.isEmpty())

{

st2.push(st2reverse.pop());

}

}

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.

15 3 1 − 5 + 2 \* −

**Question 3 [7 + 7 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. **search:** Searches the element **item** in the queue. If item is found in the queue return true else return false.

Method heading: **public boolean search(E item)**

public boolean search(E item)

{

if(size==0) return false;

Int i=front;

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

{

if( theData[i] == item) return true;

i = (i+1)%capacity;

}

return false;

}

1. **insert:** Inserts the element **item** on the **rear** end of the queue, if **item** is not already in the queue and returns true, else returns false.

Method heading: **public boolean insert(E item).**

Do not call any other method of the class **ArrayQueue** in your methods. However,you can call **search** method of part **(A)** in part **(B)**.

public Boolean insert(E item)

{

If (search(E)) return false;

/\* the part below in red is actually the reallocate method, but those students who did not write it, no mark deduction is considered for them . Actually we can not call it because in the question it is written that you are not allowed to call any method \*/

if (size == capacity)

{

int newcapacity = capacity\*2;

E[] neData = (E[]) new Object[newCapacity];

int j=front;

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

{

theData[i]=theData[j];

j=(j+1)%capacity;

}

theData = newData;

front=0;

rear=size;

capacity=newCapacity;

}

rear = (rear+1)%capacity;

theData[rear] = item;

size++;

return true;

}