OL Academy

ITCS214
Data Structures

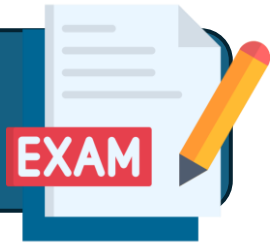
T. Ahmed Karim

Midterm Exam Revision

Exam 1

First Semester, 2024/2025

QUESTION #1:



Output questions

A Array-Based Lists

Show the output of the following:

```
public static void main(String[] args) {
ArrayList<Integer> list1 = new ArrayList<>();
ArrayList<Integer> list2 = new ArrayList<>();
list1.add(5);  list1.add(4); list1.add(11);
list1.add(7);  list1.add(6);  list1.add(3,10);
list1.set(2, list1.remove(list1.size()-2));
list1.add(list1.indexOf(4));
System.out.print("list1: ");
for (int i = 0; i < list1.size(); i++)
    System.out.print(list1.get(i) + " ");
for (int i = 0; i < list1.size(); i++) {
   int num = list1.get(i);
   if (num % 2 == 0)
      list2.add(num * num);
   else
      list2.add(num);
}
System.out.print("\nlist2: ");
for (int i = 0; i < list2.size(); i++) {
   System.out.print(list2.get(i) + " ");
 }
}
```

Output

B Single Linked List

Assume you are given the following **CompareList** method, which is included in class **SingleTest.** Show the output of the main method after calling **CompareList** method.

```
public class SingleTest <E> {
public static<E> boolean CompareList (SingleLinkedList<E> list1, E
item1) {
    if (list1.size()==0)
        return false;
   E maxnum = list1.get(0);
   int x = 1;
   while (x < list1.size()) {
      E item2 =list1.get(x);
      if(((Comparable)item2).compareTo((Comparable) maxnum)>= 0 )
         maxnum = item2 ;
     x++;
  }
   if (((Comparable)item1).compareTo((Comparable) maxnum) > 0 )
      list1.add(list1.size(),item1);
   else {
      int indexMax =list1.indexOf(maxnum);
      list1.remove(indexMax);
   }
 return true;
}
```

```
public static void main(String[] args) {
SingleLinkedList <Integer> SLL = new
SingleLinkedList<>();
SLL.add(14);
SLL.add(10);
SLL.add(25);
SLL.add(5);
SLL.add(9);
CompareList( SLL, 20);
System.out.print("\n List A: ");
for(int i=0; i< SLL.size(); i++)
   System.out.print(SLL.get(i)+" ");
CompareList( SLL, 30);
System.out.print("\n List B: ");
for(int i=0; i< SLL.size(); i++)
   System.out.print(SLL.get(i)+" "); }
}
```

Output

C Double Linked List

Show the output of the following:

```
KWLinkedList<Integer> mylist = new KWLinkedList<>();
mylist.addFirst(10);
mylist.addFirst(12);
mylist.addLast(5);
mylist.add(2, 7);
mylist.add(4);
ListIterator<Integer> iter1 = mylist.listIterator(1);
while (iter1.hasNext())
    System.out.print(iter1.next()+" ");
System.out.println();
iter1.add(9);
iter1.add(3);
iter1.previous();
while (iter1.hasPrevious())
    System.out.print(iter1.previous()+" ");
System.out.println();
iter1.remove();
while (iter1.hasNext())
    System.out.print(iter1.next()+" ");
System.out.println();
iter1.set(15);
System.out.print(iter1.hasNext()+" ");
System.out.print(iter1.previousIndex() +" ");
iter1 = mylist.listIterator(3);
while (iter1.hasNext())
    System.out.print(iter1.next()+" ");
```

Output

QUESTION #2:

Single Linked List

Write a method called **copySpecial** to be included in the SingleLinkedList class. The method has two parameters, **S** of type SingleLinkedList, which is initially empty, the method copies nodes from "this" list to S list if the value of the node is greater than item. You are not allowed to call any of the SingleLinkedList class methods (you can create nodes). The method head is:

public void copySpecial (SinleLinkedList<E> S, E item)

Before method call: item: 7	After method call:
head	head
this: [4 20 25 26 -3 10]	this: [4 20 25 26 -3 10]
	head
S:	S: [20 25 26 10]

QUESTION #3:  Double Linked List

Write a method called **containsRepeatedItem** to be included in an application class called **KWLinkedListApplication** that accepts **list1** of type KWLinkedList<Integer>. The method returns true if list1 contains a repeated element; it returns false, otherwise.

The method head is
public static boolean containsRepeatedItem (KWLinkedList <Integer> list1)

Note: You must use the ListIterator(s) and its methods only.

Example:
If list1: [13 14 13 15 16 17] then the method returns **true** because 13 is repeated.
If list1: [13 14 19 16 17] then the method returns **false** as there is no repeated item.

QUESTION #4:  Stacks

Evaluate the following postfix expression using stacks showing the stack operation one by one.

45 3 3 / * 5 + 10 / -8 -

b) Write a method called **copyHalf** to be included in an application class called StackApplication. The method has three parameters **S1, S2** and **S3** of type ArrayStack. The method copies the first half of the elements to **S2** and copies the second half of the elements to **S3 (as shown in the example given below),** if S1 has **even number** of elements only. Initially, S1 was not empty but S2 and S3 are empty.
Assume that the class ArrayStack is available for use. Use common stack operations only such as push, pop, peek, isEmpty, and copy constructor. Do not use any other data structure in your method.

The method header is given below:
public static<E> void copyHalf(ArrayStack<E> S1, ArrayStack<E> S2, ArrayStack<E> S3)

Example1:
Before method call:
top
S1: 23 29 22 17 16 7 11 9
S2: empty
S3: empty

After method call:
S1: empty
top
S2: 17 22 29 23
S3: 9 11 7 16

Example2:
Before method call:
top
S1: 23 29 22 17 16 7 11 9 12
S2: empty
S3: empty

After method call:
top
S1: 23 29 22 17 16 7 11 9 12
S2: empty
S3: empty

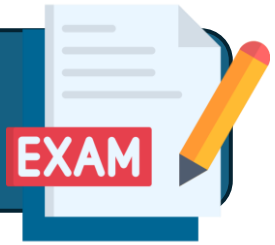
OL Academy

ITCS214
Data Structures

Exam2

Midterm Exam Revision

QUESTION #1:



Output questions

A Array-Based Lists

Show the output of the following:

```
KWArrayList <Integer> list1 = new KWArrayList<Integer>();
list1.add(1);
list1.add(0);
list1.add(1,33);
int index = list1.indexOf(22);
list1.add(index+1,52);
for(int i=0; i<list1.size();i++)
    System.out.print(list1.get(i) + " ");
list1.set(1,70);
 System.out.println();
 for(int i=0; i<list1.size();i++)
     System.out.print(list1.get(i) + " ");
```

Output

B Single Linked List

Assume you are given the following **midterm** method, which is included in the **SingleLinkedList** class.

```
public E midterm ()
{
   if (head == null)
       return null;
    E value = head.data;
    Node current = head.next;
    while (current != null)
    {
       if (((Comparable) value).compareTo ((Comparable) current.data) >0)
       {
          value = (E) current.data;
       }
       current = current.next;
    }
return value;
}
```

Output

```
public class testMidterm {
    public static void main(String[] args)
    {
list.add(15);
list.add(75);
list.add(42);
list.set(1, 34);
list.add(12);
list.add(2, 13);
list.set(3,list.remove(2));
list.add(1, 26);
for (int i = 0; i < list.size(); i++)
   System.out.print(list.get(i) + " ");
list.set(1, list.midterm());
list.add(19);
list.add(1, 37);
System.out.println();
for (int i = 0; i < list.size(); i++)
   System.out.print(list.get(i) + " ");
 }
}
```

C Double Linked List

Assume you are given the following midtermDLL method, which is included in the KWLinkedList class

```
public void midtermDLL (E item)
{
   ListIterator<E> iter = listIterator();
   while (iter.hasNext())
   {
      if (((Comparable)iter.next()).compareTo (item)<0){
         iter.remove();
      }
   }
   while(iter.hasPrevious()){
      System.out.print(iter.previous() + " ");
   }
}
```

Output

```
class TestMidtermDLL{
   public static void main(String[] args) {
      KWLinkedList<Integer> list1 = new
KWLinkedList<Integer>();
      list1.add(15);
      list1.add(10);
      list1.add(13);
      list1.add(45);
      list1.midtermDLL(20);
      System.out.println();
      list1.addFirst(35);
      list1.add(21);
      list1.add(14);
      list1.addLast(52);
      list1.add(34);
      list1.midtermDLL(15);
   }
}
```

QUESTION #2:

Array-Based Lists

Write a method called **removeDuplicates** to be included in an application class called **ArrayListEx** that accepts one parameter **list** of type **KWArrayList**. The method should remove the duplicates of values in list, Return the number of duplicates removed.

Note: Write this method by calling methods of the class **KWArrayList**.

Method head:
public static int removeDuplicates (KWArrayList<E> list)

Before method call:	After method call:
list: [7 12 10 20 14 7 5 10]	list: [7 12 10 20 14 5] Return: 2

QUESTION #3: 

Single Linked List

Write a method called **setAll** to be considered within the SingleLinkedList class. The method has two parameters of type E, **oldEntry** and **newEntry**. The method replaces every oldEntry in the list with newEntry and returns number of replacements. You are not allowed to call any of the SingleLinkedList class methods. The method head is:

public int setAll (E oldEntry, E newEntry)

Before method call:	After method call:
head this: [3 5 3 6 8 5 5 4]	setAll (5,9); this: [3 9 3 6 8 9 9 4] and returns 3.

QUESTION #4:  Double Linked List

Write a method called **removeRepeatedItem** to be included in an application class called **KWLinkedListApplication** that accepts **list** of type KWLinkedList<Integer> and an item of type integer. The method removes all occurrences (and keep only one copy) of item in list, and returns true if removal is done and false if no removal (no item or only one element equal to item in the list. You must use the ListIterator(s) and its methods only.

The method head is

public static boolean removeRepeatedItem (KWLinkedList <Integer> list, int item)

Before method call: head list: [13 14 13 15 16 17 13]	After method call: **removeRepeatedItem** (list,13); list: [13 14 15 16 17]

QUESTION #5:  Stacks

Write a method named **halfReverse** to be implemented within the **ArrayStack** class. This method has no parameters and is intended to swap the two halves of the current stack (this stack).
It returns true if the stack is **not empty** and contains **an even number of elements**. Otherwise, it returns false. **You are not allowed to use any other array or call any of the ArrayStack class methods.** The method head is:

public Boolean halfReverse ( )

Before method call:	After method call:
top this : 5 7 8 9 1 12 40 6	top this : 1 12 40 6 5 7 8 9