1. What’s wrong with this definition:

**Arrays arrays = new Arrays();**

**Int[ ]array =new int[5];**

1. Write and test this method:

void reverse(int[] a)

// reverses the elements of a[]

**Void reverse(int[]a){**

**int start =0;**

**int end=a.length-1;**

**while (start<end) {**

**int temp= a[start]=a[end];**

**a[end]=temp;**

**stert++;**

**end--;**

**}**

**}**

1. If linked lists are so much better than arrays, why are arrays used at all?

**الوصول العشوائي: يمكن الوصول إلى عنصر في المصفوفة مباشرةً باستخدام مؤشر \***

**، بينما في القوائم المرتبطة يتطلب الوصول إلى عنصر معين الانتقال عبر العناصر السابقة. (index)**

**الحجم الثابت: يمكن تعيين حجم المصفوفة مسبقًا وإجراء العمليات عليها بشكل فعال، بينما في القوائم المرتبطة \***

**يمكن أن يكون الحجم متغيرًا ويتطلب تخصيص وإعادة تخصيص الذاكرة بشكل ديناميكي.**

**البساطة: المصفوفات لديها بنية بسيطة وسهلة الفهم، بينما القوائم المرتبطة تتطلب فهمًا أكثر تعقيدًا للعمل معه \***

1. **Mark the following statements as true or false.** 
   1. In a linked list, the order of the elements is determined by the order in which the nodes were created to store the elements.

**صحيح**

* 1. In a linked list, memory allocated for the nodes is sequential.

**غير صحيح**

* 1. A single linked list can be traversed in either direction.

**صحيح**

* 1. In a linked list, nodes are always inserted either at the beginning or the end because a linked link is not a random access data structure.

**غير صحيح**

* 1. The head pointer of a linked list cannot be used to traverse the list.

**غير صحيح**

**Consider the linked list shown in Figure. Assume that the nodes are in the usual Element-Next form. Use this list to answer Exercises 5 through 8. If necessary, declare additional variables. (Assume that list, p, s, A, and B are references of type Node.)**



Linked list for Exercises 2–7

1. What is the output of each of the following java statements?
   1. System.out.println( list.getElement());

**list**

* 1. System.out.println( A. getElement());

**A**

* 1. System.out.println( B.getNext().getElement());

**B**

* 1. System.out.println( list.getNext().getNext().getElement());

**list**

1. What is the value of each of the following relational expressions?
   1. list. getElement() >= 18

**ومقارنة 18list تعتمد على العنصر المخزن في العقد المشار اليه بواسطة**

* 1. list.getNext() == A

**وغير صحيح Aهو نفس العقد المشار اليه بواسطة list صحيح اذاكان العقد التالي المشار اليه بواسطة**

**في حالة غير ذلك**

* 1. A.getNext().getElement() == 16

**وغير صحيح في حالة غير ذلك16=A صحيح اذاكان العنصر المخزن في العقد التالي للعقد المشار اليه بواسطة**

* 1. B.getNext() == (NULL)

**وغير صحيح في حالة غير ذلك NULLهو B صحيح اذاكان العقد التالي المشار اليه بواسطة**

* 1. list. getElement() == 18

**مقارنته ب 18list تعتمد على العنصر المخزن في العقد المشار اليه بواسطة**

1. Write java Fragment code to do the following:
   * + 1. Make A point to the node containing element 23. **SetE lement (23);**
       2. Make list point to the node containing 16. **list. SetE lement (16);**
       3. Make B point to the last node in the list **B = list .get Next().get Next();**
       4. Make list point to an empty list. **List =null;**
       5. Set the value of the node containing 25 to 35.

**Node temp = list;**

**while (temp != null) {**

**if (temp.getElement() == 25) {**

**temp.setElement(35);**

**break;**

**}**

**temp = temp.getNext();**

**}**

* + - 1. Create and insert the node with element 10 after the node pointed by A.

**Node newNode = new Node(10);**

**newNode.setNext(A.getNext());**

**A.setNext(newNode);**

* + - 1. Delete the node with element 23. Also, deallocate the memory occupied by this node.

**Node prev = null;**

**Node current = list;**

**while (current != null) {**

**if (current.getElement() == 23) {**

**if (prev == null) {**

**list = current.getNext();**

**} else {**

**prev.setNext(current.getNext());**

**}**

**current = null; // تحرير الذاكرة**

**break;**

**}**

**prev = current;**

**current = current.getNext();**

**}**

1. What is the output of the following java code?

**p = list;**

**while (p != NULL){**

**System.out.println( p.getElement());**

**p = p.getNext(); p = list;**

**while (p != NULL) {**

**System.out.println(p.getElement());**

**p = p.getNext();**

**}**

1. Show what is produced by the following java code. Assume the node is in the usual **getElement()-getNext()** form with the info of type int. (**list** and **p** are pointers of type **node<E>()**.)
   * + 1. list = new node<E>();

list.setElement(10);

p = new node<E>();

p. setElement(13);

p.setNext(null);

list.setNext(p);

p = new node<E>(18, list.getNext());

list.setNext(p);

System.out.println(list.getElement());

System.out.println(p.getElement());

p = p.getNext();

System.out.println(p.getElement());

**list = new Node();**

**list.setElement(10);**

**p = new Node();**

**p.setElement(13);**

**p.setNext(null);**

**list.setNext(p);**

**p = new Node(18, list.getNext());**

**list.setNext(p);**

**System.out.println(list. getElement());**

**System.out.println(p.getElement());**

**الإخراج 10 و18**

* + - 1. list = new node<E>();

list.setElement(20);

p = new node<E>();

p. setElement(28);

p.setNext(NULL);

list. setNext(p);

p = new node<E>();

p.setElement(30);

p.setNext(list);

list = p;

p = new node<E>();

p.setElement(42);

p.setNext(list.getNext());

list.setNext(p);

p = List;

while (p != NULL)

{

System.out.println( p.getElement());

p = p.getNext(); }



1. **Consider the following java statements. (The class SingleLinkedList is as defined in the lectures).**

SingleLinkedList<int> list;

list.addFirst(15);

list.addLast(28);

list.addFirst(30);

list.addFirst(2);

list.addLast(45);

list.addFirst(38);

list.addLast(25);

list.removeNode(30);

list.addFirst(18);

list.removeNode(28);

list.removeNode(12);

list.print();

What is the output of this program segment?

`list` يتم إنشاء قائمة فارغة -

يتم إضافة العناصر التالية إلى القائمة وفقًا للترتيب: 15، 28، 30، 2، 45، 38، 25 -

يتم إزالة العنصر 30 من القائمة -

يتم إضافة العنصر 18 إلى بداية القائمة -

يتم إزالة العنصر 28 من القائمة -

يتم إزالة العنصر 12 من القائمة إلا إذا كان موجودًا فعليًا في القائمة -

يتم طباعة القائمة -

1. For the following doubly linked list figure, show by java code how to insert value (info) 20 between values 15 & 24?



**لإدخال القيمة 20 بين القيمتين 15 و 24 في القائمة المزدوجة المتصلة، يمكن استخدام الخوارزمية التالية**

**قم بإنشاء عنصر جديد بقيمة 20-**

**ابحث عن العنصر الذي يحمل القيمة 15 واحتفظ بالمؤشر عليه-**

**احتفظ بالمؤشر على العنصر التالي بعد العنصر الذي يحمل القيمة 15-**

- قم بتحديث الروابط لإدراج العنصر الجديد بين العنصرين المحفوظين في الخطوتين السابقتين

- قم بتحديث الروابط للإشارة إلى العنصر الجديد بدلاً من العنصر الذي كان بعد العنصر الذي يحمل القيمة15

1. Write and test this method for **SingleLinkedList class** :

**Public int sum(Node<int> list)**

// returns: the sum of the integers in the specified list;

For example, if list is {25, 45, 65, 85}, then sum(list) will return 220.

**public int sum(Node<Integer> list) {**

**int sum = 0;**

**Node<Integer> current = list;**

**while (current != null) {**

**sum += current.getData();**

**current = current.getNext();**

**}**

**return sum;**

**}**

1. Write and test this method for **DoublyLinkedList class**:

**Public E removeLast(Node<E> list)**

// precondition: the specified list has at least two nodes;

// postcondition: the last node in the list has been deleted;

For example, if list is {22, 44, 66, 88}, then removeLast(list) will change it to {22, 44, 66}.

**public E removeLast(Node<E> list) {**

**Node<E> current = list;**

**while (current.getNext() != null && current.getNext().getNext() != null) {**

**current = current.getNext();**

**}**

**E removedData = current.getNext().getData();**

**current.setNext(null);**

**return removedData;**

**}**

1. Write and test this method for **SingleLinkedList class**:

**Public void append(Node<E> list1, Node<E> list2)**

// precondition: list1 has at least one node;

// postcondition: list1 has list2 appended to it;

For example, if list1 is {22, 33, 44, 55} and list2 is {66, 77, 88, 99}, then append(list1, list2) will change list1 to {22, 33, 44, 55, 44, 55, 66, 77, 88}. Note that no new nodes are created by this method.

**public void append(Node<E> list1,**

**Node<E> list2) {**

**Node<E> current = list1;**

**while (current.getNext() != null) {**

**current = current.getNext();**

**}**

**current.setNext(list2);**

**}**

1. Write and test this method for **SingleLinkedList class**:

**Public Node<E> concat(Node<E> list1, Node<E> list2)**

// returns: a new list that contains a copy of list1, followed by a copy of list2;

For example, if list1 is {22, 33, 44, 55} and list2 is {66, 77, 88, 99}, then concat(list1, list2) will return the new list {22, 33, 44, 55, 44, 55, 66, 77, 88}. Note that the three lists should be completely independent of each other. Changing one list should have no effect upon the others.

**public Node<E> concat(Node<E> list1, Node<E> list2) {**

**Node<E> newList = cloneList(list1);**

**Node<E> current = newList;**

**while (current.getNext() != null) {**

**current = current.getNext();**

**}**

**current.setNext(cloneList(list2));**

**return newList;**

**}**

**private Node<E> cloneList(Node<E> list) {**

**if (list == null) {**

**return null;**

**}**

**Node<E> newList = new Node<>(list.getData());**

**Node<E> current = newList;**

**Node<E> originalCurrent = list.getNext();**

**while (originalCurrent != null) {**

**Node<E> newNode = new Node<>(originalCurrent.getData());**

**current.setNext(newNode);**

**current = current.getNext();**

**originalCurrent = originalCurrent.getNext();**

**}**

**return newList;**

**}**

1. Write and test this method for **DoublyLinkedList class**:

**Public void swap(Node<E> list, int i, int j)**

// swaps the ith element with the jth element;

For example, if list is {22, 33, 44, 55, 66, 77, 88, 99}, then swap(list, 2, 5) will change list to {22, 33, 77, 55, 66, 44, 88, 99}.

**public void swap(Node<E> list, int i, int j) {**

**if (i == j) {**

**return;**

**}**

**Node<E> nodeI = getNodeAtIndex(list, i);**

**Node<E> nodeJ = getNodeAtIndex(list, j);**

**if (nodeI == null || nodeJ == null) {**

**return;**

**}**

**E tempData = nodeI.getData();**

**nodeI.setData(nodeJ.getData());**

**nodeJ.setData(tempData);**

**}**

**private Node<E> getNodeAtIndex(Node<E>**

**list, int index) {**

**Node<E> current = list;**

**int count = 0;**

**while (current != null && count < index) {**

**current = current.getNext();**

**count++;**

**}**

**return current;**

**}**

1. Describe in detail(without java code) an algorithm for reversing a singly linked list *L* using only a constant amount of additional space.

**.** واسمهم `null قم بتهيئة ثلاثة مؤشرات بالقيمة الأولية

`next `previous ‘current `

في بداية القائمة `current ابدأ ب

حتى تصل الى نهاية القائمة\*

`next احتفظ به في `current احفظ العنصر الذي يأتي بعد

previous الى current. `next غير الارتباط للعقدة الحالية باتجاه العقدة السابقة عن طريق تعيين

إلى العقدة التالية في القائمة `previous` و `current تحرك

current إلى العقدة التي تأتي بعد next تحرك

عندما تصل إلى نهاية القائمة، غير الارتباط للعقدة الأخيرة باتجاه العقدة السابقة

هو الرأس الجديد للقائمة المعكوسة **`previous**` الآن، يكون

1. Implement the equals( ) method for the DoublyLinkedList class.

**@Override**

**public boolean equals(Object obj) {**

**if (this == obj) {**

**return true;**

**}**

**if (obj == null || getClass() != obj.getClass()) {**

**return false;**

**}**

**DoublyLinkedList<E> other = (DoublyLinkedList<E>) obj;**

**if (size != other.size) {**

**return false;**

**}**

**Node<E> currentThis = head;**

**Node<E> currentOther = other.head;**

**while (currentThis != null) {**

**if (!currentThis.getData().equals(currentOther.getData())) {**

**return false;**

**}**

**currentThis = currentThis.getNext();**

**currentOther = currentOther.getNext();**

**}**

**return true;**

**}**

1. Implement the rotate() methode in CircularLinkedList class.

**public void rotate() {**

**if (head != null && head.getNext() != null) {**

**Node<E> last = head;**

**while (last.getNext() != head)**

1. Implement the addFirst() method in CircularLinkedList class.

**public void addFirst(E data) {**

**Node<E> newNode = new Node<>(data);**

**if (head == null) {**

**newNode.setNext(newNode);**

**head = newNode;**

**} else {**

**Node<E> last = head;**

**while (last.getNext() != head) {**

**last = last.getNext();**

**}**

**newNode.setNext(head);**

**head = newNode;**

**last.setNext(head);**

**}**

**size++;**

**}**