1. What’s wrong with this definition:

**Arrays[] arrays = new Arrays[];**

1. Write and test this method:

void reverse(int[] a)

// reverses the elements of a[]

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

**1. Random Access: Arrays provide constant-time access to elements based on their index. This is an important feature for many algorithms and data structures, as it allows direct access to any element in the array without having to traverse the entire data structure.**

**2. Cache Locality: Arrays offer better cache locality compared to linked lists. This means that when iterating through an array, the elements are stored contiguously in memory, which can lead to better performance due to caching effects. In contrast, linked list nodes are often scattered in memory, leading to more cache misses.**

**3. Simplicity: Arrays are simpler to use and implement. They don't require additional storage for pointers linking elements together, whereas linked lists do.**

**4. Memory Overhead: Linked lists require additional memory allocation for each node. In contrast, arrays have a smaller memory overhead, as they don't require additional storage for pointers or references.**

**5. Predictable Time Complexity: In many cases, arrays have more predictable time complexity for basic operations like access, insertion, and deletion. Linked list operations depend on traversal and pointer manipulation, which can introduce more variability in performance.**

**6. Space Efficiency: In situations where memory is a concern, arrays can offer better space efficiency. While linked lists allow for dynamic resizing, they may use more memory due to the overhead of storing pointers/references for each element.**

**7. Performance Benefits for Small Data Sets: For small data sets, the overhead of linked lists (extra memory allocation, pointer traversal, etc.) may outweigh their benefits, making arrays more practical and efficient.**

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. true
   2. In a linked list, memory allocated for the nodes is sequential.true
   3. A single linked list can be traversed in either direction.false
   4. 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. true
   5. The head pointer of a linked list cannot be used to traverse the list.true

**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());element in the node
   2. System.out.println( A. getElement());content of node A
   3. System.out.println( B.getNext().getElement());get next element
   4. System.out.println( list.getNext().getNext().getElement());get the next to the next element
2. What is the value of each of the following relational expressions?
   1. list. getElement() >= 18
   2. list.getNext() == A make the next element = A
   3. A.getNext().getElement() == 16
   4. B.getNext() == (NULL) make the value null
   5. list. getElement() == 18 make the list hold 18 as value
3. Write java Fragment code to do the following:
   * + 1. Make A point to the node containing element 23.
       2. Make list point to the node containing 16.
       3. Make B point to the last node in the list.
       4. Make list point to an empty list.
       5. Set the value of the node containing 25 to 35.
       6. Create and insert the node with element 10 after the node pointed by A.
       7. Delete the node with element 23. Also, deallocate the memory occupied by this node.
4. What is the output of the following java code?

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());

* + - 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?

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



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.

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}.

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.

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.

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}.

1. Describe in detail(without java code) an algorithm for reversing a singly linked list *L* using only a constant amount of additional space.
2. Implement the equals( ) method for the DoublyLinkedList class.
3. Implement the rotate() methode in CircularLinkedList class.
4. Implement the addFirst() method in CircularLinkedList class.