Consider the following code segment:

```
String s1 = "abc";
String s2 = s1;
String s3 = s2;
```

After this code executes, which of the following expressions would evaluate to true?

I. `s1.equals(s3)`
II. `s1 == s2`
III `s1 == s3`

A) I only
B) II only
C) III only
D) I and II only
E) I, II, and III

E) I, II, III

Assume that variable A is an array of Strings and that the variable S is a String. Consider the following code segment:

```
for (int k = 0; k < A.length; k++)
    if (A[k].compareTo(S) < 0)
        return false;
return true;
```

When does this code sement return true?

A) When all of the strings in A come before S in lexicographical order 3
B) When no string in A comes before S in lexicographical order 21
C) When no string in A comes after S in lexicographical order 1
D) When some string in A comes before S in lexicographical order
E) When some string in A comes after S in lexicographical order

B) When no string in A comes before S in lexicographical order

STUDENT OUTLINE

Lesson 30 – Linked-List Algorithms

INTRODUCTION: The linked list in this lesson will have the following special characteristic: the random order of the incoming data will be stored in ordered fashion based on a key field. After creating the linked list, a variety of algorithms will be discussed and solved.

The key topics for this lesson are:

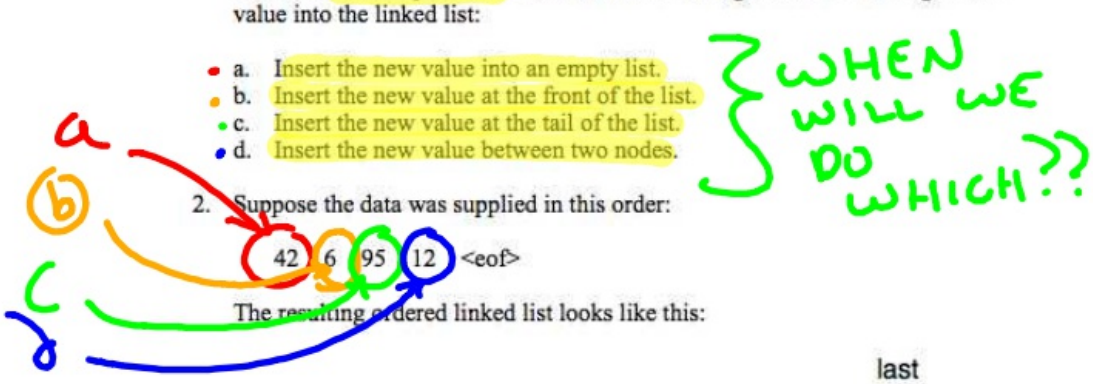
A. Building an Ordered Linked List
B. Linked-List Algorithms
C. Static vs. Dynamic Data Structures

VOCABULARY: INTERNAL POINTER EXTERNAL POINTER

DISCUSSION: A. Building an Ordered Linked List

1. If data is supplied in unordered fashion, building an ordered linked list becomes a harder problem. Consider the following cases for inserting a new value into the linked list:

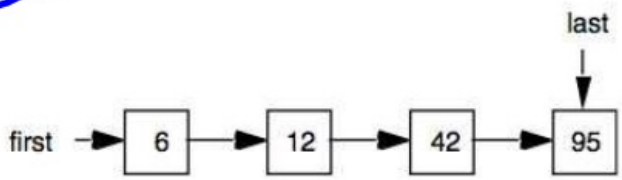
   a. Insert the new value into an empty list.
   b. Insert the new value at the front of the list.
   c. Insert the new value at the tail of the list.
   d. Insert the new value between two nodes.



2. Suppose the data was supplied in this order:

   42 6 95 12 <eof>

   The resulting ordered linked list looks like this:



3. Three of the cases are easy to identify and solve: the empty list case, placing the new value at the front of the list, or adding the value to the end of the list.

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

4. To assist us in our discussion of linked list algorithms, some definitions will be helpful. An internal pointer is one that exists inside a node. An internal pointer joins one node to the next in the linked list. An external pointer is one that points to a node from outside the list. Every linked data structure must have at least one external pointer that allows access to the data structure. Our linked list in the student outline has two external pointers, `first` and `last`.

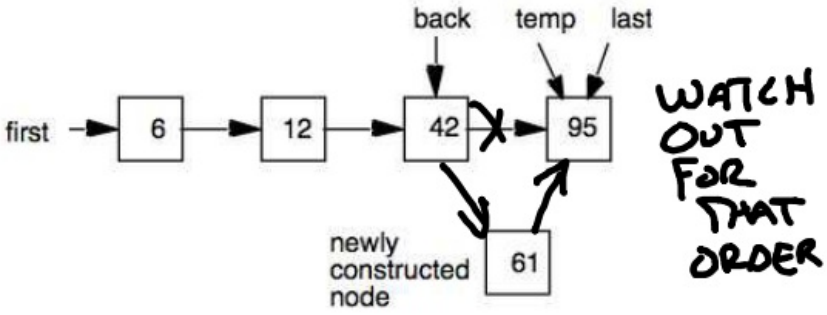
5. Suppose a fifth value, 61, is to be inserted into the list. We can see in the diagram that it will go between the 42 and 95. To help the computer "see" this, we are going to use some helping external pointers, also called auxiliary pointers.

6. It is possible to find the attachment point using just one auxiliary pointer, but we will use two called `temp` and `back`. Using two external pointers makes the hookup easier. Finding the attachment point involves something like this:

WE TRAVERSE THE LIST

```
while we haven't found the attachment point
{
   back = temp;            //move back up to temp
   temp = temp.getNext();  //advance temp one node ahead
}
```

For attaching the value 61, `back` and `temp` will end up pointing to these locations:



B. Linked-List Algorithms

1. Searching an ordered linked list is a sequential search process. A linked list is not a random access data structure. You cannot jump to the middle of a linked list. Only sequential moves are possible. The search function could return a value or a pointer to that ~~cell~~ POSITION

2. Deleting a value involves the following steps:
   a. Locating the value (if it exists) to be deleted.
   b. Rehooking pointers around the node to be deleted.

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

C. Static vs. Dynamic Data Structures

1. An *array* is a somewhat static data structure that has these advantages and disadvantages:

Advantages of an *array*	Disadvantages of an *array*
1. Easy to implement and use. 2. Fast, random access feature.	1. Memory is usually wasted.

2. A linked list (LL) is a dynamic data structure which has these advantages and disadvantages.

Advantages of LL	Disadvantages of LL
1. Memory is allocated when the program is run, therefore the data structure is only as big as it needs to be. 2. Memory is conserved.	1. Each node of the list takes more memory. 2. Processing is slower. 3. The data structure is not random access. 4. Processing must be done in sequential order.

SUMMARY/ REVIEW: We now have two different data structures to implement lists: arrays or linked lists. Each has its appropriate place in a programmer's tool kit. We will discuss variations of singly linked lists in later lessons.

ASSIGNMENT: Lab Exercise L.A.30.1, *OrderedList*

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)