CMPSC 111
Introduction to Computer Science I
Spring 2006
Bob Roos

Home page: http://cs.allegheny.edu/~rroos/cs111s2006/

Lab 9
Due Friday, 14 April, at the beginning of class

Purpose: Arrays

Details:

Listen to the Lecture
I’ll say a few things about using arrays for data storage and lookup.

Write an Insertion Method for Strings
Suppose we have an array of Strings named list, together with two int variables named size and lastIndex. Variable size contains the size of the array, and we require that all positions in the array from 0 through lastIndex contain values, while the values from lastIndex +1 are unoccupied. Initially, before anything has been saved in the array, lastIndex = -1. Once the first value has been saved in the array, lastIndex will equal 0, and so on. The array is full when lastIndex = size -1.

The strings in the list array are to be sorted in alphabetical order; however, the strings are not necessarily entered in alphabetical order. For instance,

INITIAL CONTENTS: lastIndex = -1
+-----------------------------------------------+...
|                                               |
+-----------------------------------------------+...
0 1 2 3 4 5

insert "cat":
  lastIndex = 0
+-----------------------------------------------+...
|   cat  |   |   |   |   |   |   |   ...     |
+-----------------------------------------------+...
0 1 2 3 4 5

insert "apple";
  lastIndex = 1
+-----------------------------------------------+...
|   apple |   cat |   |   |   |   |   |   ...     |
+-----------------------------------------------+...
0 1 2 3 4 5

insert "eagle";
  lastIndex = 2

Handed out on 4 April 2006
How do we do this? We have to search through the array to locate where the new string should go; then we have to shift everything forward one place from that position to the end to make room for the new string. Since the array is always in sorted order, we can do this using the “compareTo” method of the String class. Let’s suppose we are inserting a String stored in variable word. If word is greater (alphabetically) than list[0], we advance one place in the list and try again. If word is greater than list[1], we advance one place and try again. We keep doing this for positions i = 0, 1, ... until either word is less than list[i] or until i > lastIndex; we now know that i is the location where we must put word. At this point we need to shift everything forward one place in the array. We can do this by counting backwards from lastIndex down to i, moving each element to the next position in the array. Thus, list[lastIndex+1] is assigned the value of list[lastIndex], then list[lastIndex] is assigned the value of list[lastIndex-1], and so on until list[i+1] is assigned the value of list[i]. Now we can save the value word in list[i].

Here’s a summary (indentation is significant):

```
To insert word into list;
  i = 0
  while i <= lastIndex and word > list[i]
    i = i+1
  j = lastIndex
  while j >= i
    list[j+1] = list[j]
    j = j-1
  list[i] = word
  lastIndex = lastIndex+1
```

Write a Java program that inputs words, one by one, from the user and saves them in alphabetical order in a list. Use a special input word to terminate the process. Then print out the list, showing that it is in sorted order.

Then ask the user to enter a sequence of words and, for each word, state whether or not it belongs to the list you just created. (You will have to search for it in the list.)

Run your program with a minimum of 20 words of input and a minimum of 10 words to search for. Be sure to search for the first word in the list, the last word in the list, several words in the
middle of the list, a word that comes before the first word in the list, a word that comes after the last word in the list, and several other words that do not belong to the list.

Hand in a copy of the program and a printout of your sample run.

Checklist:

- All requirements met as described above
- Javadoc-style header comments
- Your name and the date printed as the first thing
- Correctly indented code; no wrapped lines
- Hard copy (printed using \texttt{a2ps}) of program
- Hard copy (printed using \texttt{a2ps}) of a sample run