CS 101 Spring 2006 Final Exam    Name: ______________________________    Email ID: ________

This exam is open text book but closed-notes, closed-calculator, closed-neighbor, etc. Unlike the midterm exams, you have a full 3 hours to work on this exam. Please sign the honor pledge here:

______________________________________________________________

______________________________________________________________

______________________________________________________________

______________________________________________________________

| | |
|---|---|
| Page 1 | none |
| Page 2 | none |
| Page 3 | none |
| Page 4 | ____ / 15 |
| Page 5 | ____ / 25 |
| Page 6 | ____ / 20 |
| Page 7 | ____ / 25 |
| Page 8 | ____ / 15 |
| **Total** | **____ / 100** |

*Note: When an integer type is required use* `int`*, when a floating-point type is required use* `double`*.*

1. (0 points) What section are you in?

____ CS 101-E

____ CS 101-3 (lab 8-9:15 a.m. Thu)

____ CS 101-4 (lab 9:30-10:45 a.m. Thu)

____ CS 101-5 (lab 11-12:15 a.m. Thu)

____ CS 101-6 (lab 12:30-1:45 p.m. Thu)

____ CS 101-7 (lab 2-3:15 p.m. Thu)

____ CS 101-8 (lab 3:30-4:45 p.m. Thu)

____ CS 101-9 (lab 5-6:15 p.m. Thu)

____ CS 101-10 (lab 6:30-7:45 p.m. Thu)

____ CS 101-11 (lab 8-9:15 p.m. Thu)

XX / XX

The following questions ask you to code up a class to implement an *associative array*. An **associative array** is composed of a collection of keys and a collection of values, where each key is associated with one value. The operation of finding the value associated with a key is called a *lookup* or indexing, and this is the most important operation supported by an associative array. The relationship between a key and its value is sometimes called a mapping or binding. For example, if the value associated with the key "bob" is 7, we say that our array *maps* "bob" to 7. This is similar to how a dictionary operates: the key is the word you are looking up, such as ‘set’; the value is the definitions for that word.

From the perspective of a programmer using an associative array, it can be viewed as a generalization of an array of objects: while a regular array maps integers (the array index) to arbitrarily typed objects (Locations, Strings, etc.), an associative array maps arbitrarily typed objects to arbitrarily typed objects. For this exam, we will be mapping Strings to Strings.

In this series of questions, we will be using the associative array as a phonebook, able to map something like "UVa Computer Science Department" to "434 982 2200". Therefore, you will write Java code for the AssocList class, providing code for the following:

- Any necessary instance variables.
- `AssocList()`: the default constructor.
- `void add(String key, String value)`: Bind a new key to a new value
- `void increaseCapacity()`: Increases the size of the array so it can hold more values
- `void reassign(String key, String newValue)`: Bind an old key to a new value
- `void remove(String key)`: Unbind a key from a value and remove it from the key set
- `String lookup(String key)`: Find the value (if any) that is bound to a key
- `String toString()`: Returns the String representation of this associative array
- `boolean isEmpty()`: Returns if there are no elements inserted into this associative array
- `int size()`: Returns the number of elements inserted into the array

**While there can be many different implementations of an associative array, the way in which you must implement it is by keeping two separate 1-dimensional arrays, called "keys" and "values". The String entry in the first array (the "keys" array) in a particular position is "matched" by a String entry in the second array (the "values" array). If keys[12] = "UVa Computer Science Department", then values[12] = "434 982 2200".**

To emphasize that you are writing a single functional class, we have written this exam up in the form of skeleton code for you to complete. The different code sections equate to exam questions, and are explained further in the comments along with their point values. Your complete answers to these questions should be a functioning Java class. Don’t forget to count your braces, parentheses, etc. We also include the full code for a **`main()`** method to illustrate the use of the AssocList class.

When developing this class, you may assume the other methods are written. In other words, when writing the add() method, you can assume that the size() method is written properly, and thus can use size() in add().

XX / XX

```
// The below main() method below is simply to give you an example of how the
// AssocList class might be used and how it should behave. You do not need
// to modify, add, or otherwise do anything with the below code.  Note that
// the code here tests some of the functionality of the AssocList class, but
// not all.

public static void main (String [] args) {

    // create a new AssocList
    AssocList phonebook = new AssocList ();

    // Add entries for Papa Johns, Dominos, and Pizza Hut
    phonebook.add("papa johns", "434-296-7272");
    phonebook.add("dominos", "434-979-2656");
    phonebook.add("pizza hut", "434-979-5588");

    // print the phone book
    System.out.println(phonebook);

    // Try a search: lookup domino's
    // Should print "Lookup for dominos returned: 434-979-2656"
    String ret = phonebook.lookup("dominos");
    System.out.println("Lookup for dominos returned: " + ret);

    // Oops! Wrong domino's! Get one closer to campus!
    phonebook.reassign("dominos", "434-971-8383");

    // print the phone book again -- should show only one dominos, with
    // updated value
    System.out.println(phonebook);

    // remove one real entry, one bogus entry
    boolean b1 = phonebook.remove("papa johns");
    boolean b2 = phonebook.remove("little caesars");

    // print the phone book one last time -- should show only one
    // dominos, with updated value
    System.out.println(phonebook);

    // Show size and emptiness
    System.out.println ("The array has " + phonebook.size() +
                        " items; isEmpty() returns " +
                        phonebook.isEmpty());
  }

// Output:
//
// AssocList[papa johns=434-296-7272, dominos=434-979-2656, pizza hut=434-979-5588]
// Lookup for dominos returned: 434-979-2656
// AssocList[papa johns=434-296-7272, dominos=434-971-8383, pizza hut=434-979-5588]
// AssocList[dominos=434-971-8383, pizza hut=434-979-5588]
// The array has 2 items; isEmpty() returns false
```

XX / XX

```
public class AssocList {

// Question 2: 10 points
// instance variables: give the code necessary to declare the instance
// variables needed by this class, using the information given in the
// introduction. The instance variables may be initialized here or in the
// constructor below.  You may choose any value you want for the initial
// capacity of the AssocList arrays
```

```
// Question 3: 5 points
// Provide the code for the definition of the default constructor. You may
// initialize your instance variables here or when they are defined above.
```

___ / 15

```
// Question 4: 10 points
// add(): Add the key,value to an empty position in the arrays.  You must
// “expand” the array capacity as needed via the increaseCapacity() method
// (see question 5).  If the key already exists in the AssocList, then the
// old value is replaced by the new value (the one passed in as a parameter)

public void add(String key, String value) {
```

```
// Question 5: 15 points
// increaseCapacity(): This method will double the size of BOTH
// arrays, and copy the values from the old arrays to the new ones.

private void increaseCapacity() {
```

___ / 25

```
// Question 6: 10 points
// reassign(): Change the "value" associated with the "key" to "newValue".
// This method returns whether the reassignment was successful (i.e. if the
// value was not found, then it returns false; if it was found (and thus
// reassigned), it returns true)

public boolean reassign(String key, String newValue) {
```

```
// Question 7: 10 points
// remove(): Remove the "key" and its associated value from the arrays.
// Note that if a value is removed from the middle of the associative array,
// you DO need to “shift” the successive values down one spot.  This method
// should return true or false, depending on whether the value was removed or
// not (i.e. if the value was not found, it will return false).

public void remove(String key) {
```

___ / 20

```
// Question 8: 10 points
// lookup(): Find the value associated with the key

public String lookup(String key) {
```

```
// Question 9: 15 points
// toString(): Return a String with the elements currently stored in the queue.
//
// The String should be of the form, "AssocList[a=val1, b=val2, c=val3]" where
// a,b,c are keys and val1, val2, and val3 are the associated values.

public String toString () {
```

___ / 25

```
// Question 10: 5 points
// isEmpty(): Returns whether the associative array is empty or not.

public boolean isEmpty() {
```

```
// Question 11: 10 points
// size(): Returns the number of elements in the associative array.

public int size() {
```

___ / 15