Computer Laboratory 10
CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development
November 12–13, 2019

0. Introduction.

This assignment involves implementing a data structure called an *association list,* a term borrowed from the programming language *Lisp.* It acts something like a Python dictionary, because it associates keys with their corresponding values. However, it uses a linear singly-linked list internally, so it is slower than a dictionary.

1. Theory.

An *association list* is a linear, singly-linked list of nodes. Each node in an association list has three slots, called `key`, `value`, and `next`. The `key` and `value` slots point to objects of specific types. The `next` slot points to the next node in the list (or to `null`). The node associates its `key` object with its `value` object. Association lists have the following operations, all of which involve traversing lists to search for nodes with given `key` objects.

- You can get the `value` object from a node. It's the object that is associated with the node's `key` object.
- You can change the `value` object in a node. After that, the `key` object is associated with a different `value` object.
- You can delete the node that contains a `key`. After that, the `key` object is not associated with a `value` object.
- If you cannot find a node, then you can add a new one that contains a `key` object and a `value` object. After that, the `key` object is associated with the `value` object.

As a result of these operations, you can use an association list something like a dictionary. For example, the `key` objects might be `String`'s that are English words for numbers. The `value` objects might be `Integer`'s that correspond to those words. If you give the association list an English word, then you can get back its corresponding number.

Association lists work by doing linear search. As a result, if an association list has n keys, then each of the operations described above will require $O(n)$ comparisons. Later in this course, we'll discuss more efficient alternatives to association lists. These will need only $O(\log n)$ or even $O(1)$ comparisons.

2. Implementation.

You must write a Java class called `AssociationList` that implements an association list. To simplify grading, your class must use the same names for things that are used here. It must have two class parameters, called `Key` and `Value`, so it looks like this.

```
class AssociationList<Key, Value>
{
  ⋮
}
```

Here `Key` is the type of the association list's `key` objects, and `Value` is the type of the association list's `value` objects.

Within the class `AssociationList`, you must have a `private` class called `Node`. The class `Node` must have three `private` slots: a slot called `key` whose type is `Key`, a slot called `value` whose type is `Value`, and a slot called `next` whose type is `Node`. It must have a constructor that initializes these three slots from its arguments.

Don't try to use the node classes from the lectures, or from other lab assignments. They have the wrong number of slots, and the wrong types of slots. Also, *you are not allowed to use arrays in any way.* If you implement `AssociationList` using one or more arrays, then you will receive zero points for this lab.

Your `AssociationList` class must also have a `private` variable called `head`. It must point to the head `Node` in a linear singly-linked list of `Node`'s. The head node is a dummy node that is always at the front of a linked list; it helps eliminate special cases when deleting nodes. Along with `Node` and `head`, your class must have these methods. All of them (except for `isEqual`) work with `head` somehow.

`public AssociationList()`

> Constructor. Initialize an empty instance of `AssociationList`. Hint: make the `head` node here.

`public void delete(Key key)`

> Search the association list for a `Node` whose `key` slot equals the `key` parameter, according to `isEqual`. Delete that `Node` from the list. If no `Node` has a `key` slot that equals the `key` parameter, then do nothing. Your `delete` method must not use a special case to delete the first `Node` in the list after `head`. Hint: use the ''left-right trick'' discussed in the lectures.

`public Value get(Key key)`

> Search the association list for a `Node` whose `key` slot equals the `key` parameter, according to `isEqual`. Return the `value` slot of that `Node`. If no `Node` has a `key` slot that equals the `key` parameter, then throw an `IllegalArgumentException`. Hints: skip the `head` node. You don't need the ''left-right trick'' here.

`private boolean isEqual(Key leftKey, Key rightKey)`

Test if `leftKey` is equal to `rightKey`. Either or both may be `null`. This method is necessary because you must use `==` when `leftKey` or `rightKey` are `null`, but you must use the `equals` method when both are not `null`. (Recall that `null` has no methods.)

```
public boolean isIn(Key key)
```

Search the association list for a `Node` whose `key` slot equals the `key` parameter according to `isEqual`. Return `true` if you find such a `Node`, and return `false` otherwise. Hints: skip the `head` node. You don't need the ''left-right trick'' here.

```
public void put(Key key, Value value)
```

Search the association list for a `Node` whose `key` slot equals the `key` parameter, according to `isEqual`. Change the `value` slot of that `Node` to be the `value` parameter. If there is no such `Node`, then add a new `Node` immediately after the `head` node. The new `Node`'s `key` slot is the `key` parameter, and its `value` slot is the `value` parameter. Hint: you don't need the ''left-right trick'' here.

The file **`tests10.java`** on Canvas contains Java code that performs a series of tests. Each test calls a method from your class `AssociationList`, and prints what the method returns. Each test is also followed by a comment that tells how many points it is worth, and what must be printed if it works correctly.

3. Deliverables.

Run the tests, then turn in the Java source code for your class `AssociationList`. If you do not know how or where to turn it in, then ask your lab TA. Your work must be submitted by **11:55 PM** on **Wednesday, November 20, 2019.**