CS 61B Homework 6
Due 3pm Wednesday, March 15, 2000

This homework will teach you about hash tables, hash codes, and hash functions. This is an individual assignment; you may not share code with other students.

Copy the Homework 6 directory by doing the following, starting from your home directory. Don't forget the "-r" switch in the cp command.

```
    mkdir hw6
    cd hw6
    cp -r $master/hw/hw6/* .
```

Part I (6 points)

Implement a class called HashTableChained, a hash table with chaining. HashTableChained implements an interface called Dictionary. Dictionary defines the methods you need to implement, as well as an object named "NO_SUCH_KEY" (described in Chapter 7 of Goodrich and Tamassia). Both files appear in the "dict" package.

The methods you will implement are a subset of those listed on page 248 of Goodrich and Tamassia (no Enumerations are used in this assignment), plus a makeEmpty() method which removes every item from a hash table. There are also two HashTableChained constructors. One lets aplications specify an estimate of the number of items that will be stored in the hash table; the other uses a default size. Both constructors should create a hash table that uses a prime number of buckets. (Several methods for identifying prime numbers were discussed early in the semester.) In the first constructor, shoot for a load factor between 0.5 and 1. Descriptions of all the methods may be found in Dictionary.java and HashTableChained.java.

Do not change Dictionary.java. Do not change any prototypes in HashTableChained.java, or throw any checked exceptions. Most of your solution should appear in HashTableChained.java, but other classes are permitted. You will probably want to use linked list code (of your choice); if it is in a package, make sure the package is called "list", or you won't be able to submit it.

Look up the hashCode method in the java.lang.Object API. Assume that the objects used as keys to your hash table have a hashCode() method that returns a "good" hash code between Integer.MIN_VALUE and Integer.MAX_VALUE (that is, between -2147483648 and 2147483647). Your hash table should use a hash function, as described in Section 7.6.1 of Goodrich and Tamassia, to map each key's hash code to a bucket of the table. Your hash function should be computed by the hashFunction() helper method in HashTableChained.java (so we can test it). insertItem(), findElement(), and remove() should all use this hashFunction() method.

The methods findElement() and remove() should return (and in the latter case, remove) an item whose key is equals() to the parameter "key". Reference equality is NOT required for a match.

Part II (4 points)

It is often useful to hash data structures other than strings or integers. For example, game tree search can sometimes be sped by saving game boards and their evaluation functions, so that if the same game board can be reached by several different sequences of moves, it will only have to be evaluated once. For this application each game board is a key, and the value returned by the evaluation function is the element with which the key is associated. If we search the same game board again, we can look up its evaluation function in the dictionary, so we won't have to calculate it twice.

The class SimpleBoard represents an 8x8 checkerboard. Each position has one of three values: 0, 1, or 2. Your job is to fill in two missing methods: equals() and hashCode(). The equals() operation should be true whenever the boards have the same pieces in the same locations. The hashCode() function should satisfy the specifications described in the java.lang.Object API. In particular, if two SimpleBoards are equals(), they have the same hash code.

You will be graded on how "good" your hash code and hash function are. By "good" we mean that, regardless of the table size, the hash code and hash function evenly distribute SimpleBoards throughout the hash table. Your solution will be graded in part on how well it distributes a set of randomly constructed boards. Hence, the sum of all the cells is not a good hash code, because it does not change if cells are swapped. The product of all cells is even worse, because it's usually zero. What's better? One possibility: think of the cells as digits of a base-3 number.

Do not change any prototypes in SimpleBoard.java, or throw any checked exceptions. The file Homework6Test.java is provided to help you test your HashTableChained and your SimpleBoard together. Note that Homework6Test.java does NOT test all the methods of HashTableChained; you should write additional tests of your own.

Submitting your solution

Change (cd) to your hw6 directory, which should contain SimpleBoard.java and the dict directory (and possibly a list directory). The dict directory should contain HashTableChained.java and any related class files. You're not allowed to change Dictionary.java, so you can't submit it; nor may you submit Homework6Test.java (though you can change it as much as you like). Include your name, login, and lab section in either GRADER or HashTableChained.java. From your hw6 directory, type "submit hw6".