CSC 316
Homework #4: Hash Programming Assignment

This programming assignment involves the use of hashing.
All necessary details are provided below.

Each *key* is a vector of length L whose components are bounded integers:

$$(n_1, n_2, \ldots, n_L),$$

where $0 \leq n_i \leq max_i$, for $i = 1, 2, \ldots, L$.

Your program should request a user (the TA) to supply the name of the file from which the input is to be read. The first line of the input file will contain the value L; line 2 will contain $max_1, max_2, \ldots, max_L$; the keys will follow at a rate of one per line; the last line will contain -911. The same key may be present many times at many different places in the input file.

This assignment requires you to develop a data structure (or structures) from which you can extract any key and the order in which that key was inserted. You must make use of a hash table in this programming assignment. You must develop your own hash function and collision avoidance strategy based on the concepts discussed in class.

Your program should read in the keys from the input file one at a time. For each key read from the input file, you need to check if that key has already been inserted. Every time a new key is found, it must be inserted into your data structure. Only a single copy of each different key must be stored in your hash table. Additionally, you must associate an integer with each different key. This integer should indicate the number of *different* keys that have been found prior to finding this key. Essentially, this integer is an index for the key.

Sample input file

```
4                ===> L (no. of components in a key)
2   4   5   3    ===> Maximum values
1   1   0   3    ===> 1st key (index 0)  This key must be kept.
0   1   0   1    ===> 2nd key (index 1)  This key must be kept.
1   1   0   3    ===> Same as 1st key.         Ignore.
0   1   0   1    ===> Same as 2nd key.         Ignore.
0   1   0   1    ===> Same as 2nd key.         Ignore.
2   4   5   3    ===> 3rd key (index 2)  This key must be kept.
1   1   1   1    ===> 4th key (index 3)  This key must be kept.
0   1   0   1    ===> Same as 2nd key.         Ignore.
0   1   0   1    ===> Same as 2nd key.         Ignore.
-911             ===> End of file
```

The objective of this programming assignment is to develop data structures that perform the required functions in a way that minimizes both computation time and computer memory requirements. To help quantify the success of your data structure, after all the keys have been properly inserted, your program must print out:

1. The total number of bytes used by your data structure(s).
2. The total time (in seconds) taken by your program to insert all the keys.

For testing purposes, a set of 9 data files are compressed into a single file, HashData.tar.gz that may be downloaded using the sftp protocol (or scp/WinSCP, if you prefer) from the class web page:
/afs/eos.ncsu.edu/courses/csc/csc316/lec/001/www/wrap/HashData

When all the keys in the input file have been inserted, your program should provide a user with two functions

1. The user should be provided with a means to input a key (as a sequence of integers). The program should check that the sequence of integers consists of L components, the i^{th} of which lies between 0 and max_i. If your program determines that these conditions are satisfied, it should then find the index of the key and print out the key and its index. If the key is not found, then a message to this effect should be printed. Your program should **not** insert this (user-supplied) key into your data structure. In both cases, the number of probes taken by your program should also be printed.
2. The user should be provided with a means to input an index (a single integer). Your program should verify that this integer lies between zero and one less than the number of keys that were inserted. Your program should then print out the corresponding key.

In the README file that you submit, describe (in words and sentences):

- The hash function you used, explaining why you chose this particular function (or functions) and whether it performed as you thought it would.
- The collision avoidance policy you used, again explaining why you chose this particular policy and whether you found it to perform as expected.

In the README file, you must also include a table containing the time needed to insert all the keys and the maximum number of bytes used by all data structures for each of the 9 data sets provided in the class directory. Also state the the average values (time and memory) for these 9 cases.