CS 61A Summer 2009 Week 5A Lab
Monday 7/20 Afternoon

1. In lecture, you have seen the program `double-up!`. You can also find the program in ~`cs61a/lib/double-up.scm` and a tutorial on the main course website. Write a procedure called `divorce!` that is the inverse of `double-up!`, that is, it takes a list of pairs and turn it into a flat list. For example:

```
> (define a-list '((barack . michelle) (john . cindy) (sarah . todd)))
> (divorce! a-list)
 okay
> a-list
 (barack michelle john cindy sarah todd)
```

`divorce!` should return `okay` and preserve original pointer.

2. Write a vector procedure `(double-n-copy vec)` that takes a vector, makes a new vector that is twice as long, and copy every element of the input vector into the first half of the newly created vector, fill the second half of the newly created vector with `nil`, and returns the new vector. For example:

```
> (double-n-copy '#(thunderbolts and lightning very very frightening))
 #(thunderbolts and lightning very very frightening () () () () () ())
```

3. For the rest of this lab, you will build a table using vectors. Recall from the project that a Table is an ADT that stores key-value pairs of information. A table supports the following constructor and selectors:

1. `(make-table)` creates a new table
2. `(put key val table)` that takes a table and puts the key-value pair in
3. `(get key table)` returns the value associated with the key or `#f` if key is not in the table.

We are going to make our table as **a pair of two vectors**: one vector that stores all the keys and one that stores all the values such that the corresponding value would be stored in the same index as the key. For example, if `strength, 20` was a key-value pair in our table, then `strength` and 20 will be stored in different vectors but they will be at the same index. For convenience, we will disallow `nil` as a key.

Implement `make-table` to create a pair of vectors of length 5. They should be initialized with `nil`.

4. Now, implement `put` so that it searches through the vector to see if any position in the key-vector is `nil`. If so, then it adds the key-value pair into the table. **If the key-vector in the table is completely full, put should create a new key-vector that is twice as long as the old one, copy over all the original key-value pairs, and then insert the new key-pair pair.**

5. Implement `get` so that it returns the value associated with the key or `#f` if the key is not found.

Note: You may notice that our table is very inefficient. `put` and `get` will both take linear time to perform; a terrible speed if the table is large. Can you think of a way to store some extra information in the `table` ADT so that if the table is not full, `put` will take constant time? In CS61B, you will learn an important data structure called **Hash Table** where `get` will also take constant time.