CS 260
Programming Assignment 4

This lab is worth a total of 100 points; 10 points are the self-evaluation, 50 points for the basic lab, 30 points for the advanced lab, and 10 points for a solution to the thinking problem.

Base Lab Specification

Create a class StringHash. You may base your code off of the pseudocode in the document in Moodle.

If the array is over 1/2 full, you should create a new larger array and move the old values to the new array using a revised hash method (optimally, you would use a series of prime sizes, but this is not required (11, 23, 43, 89, …)

You will store strings in the array. You should initialize the array with the string "_empty_" and use "_deleted_" for deleted items. You should define constants for these two strings.

Your hash method should convert the strings to an integral index using the algorithm shown below.

This hash table should use open addressing and linear probing

This lab should follow the course coding requirements and be split into multiple files as per your chosen language. There is a driver provided for this lab, you should test each part of your code as it is developed.

The base methods required are:

- constructor(Python init) – create the array for your class and initialize it as needed. The default size should be 11, but you should allow the user to specify an alternate size if desired (if the specified size is less than 11, go with the default size).
- destructor(c++ only) – free up allocated memory
- addItem(value) – add value to the new table in the appropriate spot, no return value
- findItem(value) – return true if value is present in the table, false otherwise • removeItem(value)

  – remove the item from the table if present, do nothing otherwise
- displayTable() – return a string showing the contents of the table, one item per line.

Advanced Lab Specification

Create a class ChainHash that implements separate chaining. You may base your code off of the pseudocode in the document in Moodle.

This class should start with a default size of 7. You do **not** need to double if it fills up.

You will need to define a ChainLink class and use it in your ChainHash class.

You need to make it an array of ChainLinks (C++ pointers) and have it initialized to nullptr/None/null as per the document.

You should use the same basic hash function as used in your StringHash class.

This work by Jim Bailey is licensed under a Creative Commons Attribution 4.0 International License.

The advanced methods are:

- constructor(Python init) – create the array for your class and initialize it as needed. The default size should be 7, but you should allow the user to specify an alternate size if desired (if the specified size is less than 7, go with the default size).
- destructor – required for C++ to free up all allocated memory
- addItem(value) – add value to the new table in the appropriate spot, no return value
- findItem(value) – return true if value is present in the table, false otherwise
- removeItem(value) – remove the item from the table if present, do nothing otherwise
- displayTable() –returns a string with one line per element with its links separated by spaces. Empty cells should be shown as "_empty_". There is example output in the Moodle document on separate chaining.

Thinking problem

How would you implement expanding a hashTable using separate chaining? Provide a method that does so in your class. Your table sizes should be primes and roughly double each time. Since ChainHash starts with a size of 7, example sizes are (13, 29, 59, 127, 257). Modify your ChainHash class to call this method when adding a new item would cause the number of items to be more than twice the size.

Hash Function

The following algorithm is from the Hash Table document. Your program must implement it for the tests to properly evaluate the results.

```
hashFunc(key)

  // initialize index
  hashValue = 0

  // walk through string one char at a time
  for i = 0; i < key.length(); i++

    // multiply current sum
    hashValue *= 128

    // add current character's ascii value
    hashValue += key[i]

    // shrink to fit
    hashValue %= arraySize


  // return the result
```

```
    return hashValue
```