CSCI203 – Week 6 lab exercise 3

Hashing

For this exercise, you are to implement a simple hash table.

What your program should do:

1. Read data from file including a sequence of integer values.
   As usual, your program will prompt for the name of an input file and the read and process the data contained in this file. A sample file, **Ex3.txt**, has been provided for test purposes.
2. Process the integer value (k) from file one-by-one, and calculate its hash value as

   h(k)= k mod 100,

   Then construct a hash chain, where we add all $k_1, k_2, \cdots, k_l$ with the **same hash value, i.e.,** h$(k_1) = h(k_2) = \cdots = h(k_l)$, into a chain according to its order in file.

   For this exercise, you may use dynamic data for chains greater than one in length, but the **majority of the dictionary** should be an array of hash nodes. In other words, you should use array to implement the hash chain.

3. The outputs of your program are:
   1. The number of empty entries in the hash table.
   2. The length of the longest chain.

Note that, you should only include or import the following libraries in the beginning of your program. In other words, only input and output streams are allowed.

1. If you use C++

```
#include<iostream>
#include<fstream>
using namespace std;
```

2. If you use Java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
```

3. If you use Python

```
import sys
import numpy
```

Experiment Guide

In this lab, you are to implement a hashing table with chaining. The pseudocode outline for the program is given below:

```
Begin main
    display a prompt for the file name
    read in the file name
    try to open the file
    if the file fails to open
        print an error message on the screen and exit
    fi
    do
        read in an integer from the file
        if the file read fails
            terminate (break) the loop
        fi
        store the value into the table
    od
    close the file
    Report the statistics for the hash table
       1. The empty slots in the hash array
       2. The longest chain has a length of
End main
```

1. In your program, you will define a structure or class for the hash node, which is used to store the hash table nodes.

   hashNode {
     int value;
     boolean empty;
     hashNode next;
   }

   The bool variable, empty, is used to track whether there is data stored in the current node. If empty is false when we attempt to store a value in our hash table we will need to use overflow chains to enable us to store the extra data. The hashNode variable, next, is used to point to the next hashNode of the current node in the same chain.

2. Then, you need declare an array of hashnode as well as some global variables, including
   hash_size = 100
   emptyCount = hash_size=100 // used to count the empty items in the hash table
   longest = 0 // used to count the longest length of all chains in the table

3. The main functions should be implementd in your program is
   - **TableInsert (int incoming)**

For each value read from the file, we use this function to find the appropriate storage location for that. Specifically, the pseudocode of this function is

```
TableInsert (int incoming)
{
Calculate the hash of the input value by using equation hash = incoming % hash_size;

Add the input value to hash table, first check if hashTable[hash] is empty
        1.  If yes,
            create a new hashNode and add it to hashTable[hash]
            emptyCount –– ;
        2.  Else
            hashTable[hash].next =chainInsert (incoming, hashTable[hash].next, 1)
            // which means that, there already exists a chain in hashtable[hash], we
            need to add a new node to the chain.
            Then, we first call chainInsert() to insert a new node to the chain, and
            associate the chain with the hash table.
  Stop
 }
```

- **ChainInsert (int value, hashNode current, int chain-length)**
  This function is used to insert the incoming value to a position of hashTable, where there already exists a chain of value. The implementation of the function involves following the chain down until the current node is empty. At this point we create a new node and store the incoming value into it. The three inputs to this function is the incoming inserted value, the position of hashTable to be inserted and chain length.

```
   ChainInsert(int value, hashNode current, int chain-length)
{

   Check if current is null
    If yes,
        create a new hashNode for incoming;
        set the longest as longest++ if chain-length==longest
    else
        current.next = call ChainInert(incoming, current.next, ++chainLength);

    return the new hashNode created
  }
```