Santa Monica College
CS 20A: C++ Data Structures
Assignment 11: Comparing Hash Functions

In this assignment, you will test three hash functions provided by the instructor and then create one of your own.

Please download the starter files for this assignment from the Files tab. **Do not alter the existing function definitions or driver code in any way**. We **will be using** Standard Template Library classes for this assignment.

1. Comparing Hash Functions

A good hash function distributes keys uniformly across its output range. The simplest way to compare hashing algorithms is to compare the number of collisions resulting when hashing a set of keys.

For each hash function, compute the number of collisions occurring when calculating hash codes for each word in the dictionary provided by the instructor. Print out a list showing each colliding hash-key and the words that collided using the print function provided by the instructor.

```
// http://www.cse.yorku.ca/~oz/hash.html

// First reported by Dan Berstein in the news group comp.lang.c.
unsigned long djb2(unsigned char *str)
{
        unsigned long hash = 5381;
        int c;

        while (c = *str++)
                hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

        return hash;
}

// Created for sdbm (a public-domain reimplementation of ndbm) database library.
// It was found to do well in scrambling bits causing better distribution of
// the keys and fewer splits.
unsigned long sdbm(unsigned char *str)
{
        unsigned long hash = 0;
        int c;

        while (c = *str++)
                hash = c + (hash << 6) + (hash << 16) - hash;

        return hash;
}

// First appeared in K&R (1st edition).  Terrible hashing algorithm because it
// produces many collisions.
unsigned long loselose(unsigned char *str)
{
        unsigned long hash = 0;
        int c;

        while (c = *str++)
                hash += c;

        return hash;
}
```

Here is some code that opens a dictionary file, reads each word, calculates a hash key, and then closes the file.

```
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <map>
#include <vector>

using namespace std;

unsigned long djb2(unsigned char *str);
unsigned long sdbm(unsigned char *str);
unsigned long loselose(unsigned char *str);

//*** Sample custom functions ***

void printCollisions(string name, map<unsigned long, vector<string> > & keys);

int main()
{
        const int MAX_CHARS_PER_LINE = 512;
        unsigned long hkey = 0;

        map<unsigned long, vector<string> > djb2Keys;
        map<unsigned long, vector<string> > sdbmKeys;
        map<unsigned long, vector<string> > loseloseKeys;

        ifstream fin;
        char buf[MAX_CHARS_PER_LINE];

        fin.open("dictionary.txt");

        if (!fin.good())
        {
                cerr << "Error Opening File" << endl;
                return 1;
        }

        while (!fin.eof())
        {
                fin.getline(buf, MAX_CHARS_PER_LINE);
                string word = (char *)buf;

                hkey = djb2((unsigned char *)buf);
                djb2Keys[hkey].push_back(word);

                hkey = sdbm((unsigned char *)buf);
                sdbmKeys[hkey].push_back(word);

                hkey = loselose((unsigned char *)buf);
                loseloseKeys[hkey].push_back(word);

                //*** Test your functions here. ***
        }

        fin.close();

        printCollisions("djb2", djb2Keys);
        printCollisions("sdbm", sdbmKeys);
        printCollisions("loselose", loseloseKeys);

        system("pause");
        return 0;
}
```

You may use the following print function if you like.

```
void printCollisions(string name, map<unsigned long, vector<string> > & keys)
{
        int collisions = 0;

        cout << name << " Collision List:" << endl << endl;
        for (map<unsigned long, vector<string>>::iterator it = keys.begin();
                        it != keys.end(); ++it) {
                if (it->second.size() > 1)
                {
                        collisions += (it->second.size() - 1);

                        cout << "Hash key " << it->first << " occurs "
                                        << it->second.size() << " times." << endl;

                        for (vector<string>::iterator itr = it->second.begin();
                                        itr != it->second.end(); ++itr) {
                                cout << *itr << " ";
                        }

                        cout << endl << endl;
                }
        }

        cout << "Total " << name << " collisions = " << collisions << endl << endl;
}
```

2. Create and test three original hash functions.

Your hash function must have the following signature; you may assign your own function name:

```
unsigned long hf(unsigned char *str);
```

Print out the collisions for your hash function using the code provided by the instructor.

Sample output:

```
djb2 - Collisions = 59

5863248: bi d'
5863413: gi i'
5863611: mi o'
121541278: joyful synaphea
153103337: broadened kilohm
193486210: ais c's
193487299: bis d's
193488388: cis e's
193489477: dis f's
193492729: gid i'd
193492738: gim i'm
193492744: gis i's
193493833: his j's
193498189: lis n's
193499278: mis o's
193500367: nis p's
193502545: pis r's
193505812: sis u's
193506901: tis v's
193509079: vis x's
193510168: wis y's
193511257: xis z's
```

```
226706708: mensis menu's
253411110: arris art's
253986102: baris bat's
256495158: delis den's
256861062: doris dot's
266348430: loris lot's
267026877: manis map's
267031233: maris mat's
267168447: melis men's
268702848: nobis nod's
270582462: palis pan's
270869958: pilis pin's
271301202: pulis pun's
272953215: rakis ram's
272956482: ranis rap's
874715923: heliotropes neurospora
919705060: animate animate
1006249872: appling bedaggle
```