**CMPS 12M**

**Introduction to Data Structures**

# Lab Assignment 7

## Objective

The purpose of this assignment is to learn how to write a good hash function in C.

## Overview

You are given a test program in C that tests for collisions in a hash function. Your task is to implement the hash function and try to get the best score possible. Files for this lab can be found in:

`/afs/cats.ucsc.edu/users/r/nwhitehe/cmps12/lab7/`

## Hash Function

The hash function you write must accept a string of characters as input and return an integer between 0 and 65535. The input string parameter is declared as a pointer to an unsigned char. Strings in C consist of sequential characters stored in memory, with a 0 character terminating the string. For easier math the characters are declared unsigned in the hash function which means each character takes values from 0 to 255.

Your function is tested against a file that contains 65536 random English words. Your hash function is applied to each word individually. The perfect hash function would give each of the 65536 words a different value from 0 to 65535 with no collissions. This function would show output:

```
65536 lines read
65536 bucket(s) with 1 entries

---------------------------
Total collisions: 0
SCORE = 0
```

This perfect hash is probably impossible. The hash you write will almost surely have many collisions. Here is example output for a poor quality hash function:

```
65536 lines read
1 bucket(s) with 6608 entries
1 bucket(s) with 4438 entries
1 bucket(s) with 3252 entries
1 bucket(s) with 3033 entries
1 bucket(s) with 2862 entries
1 bucket(s) with 2223 entries
1 bucket(s) with 2034 entries
1 bucket(s) with 1701 entries
1 bucket(s) with 1307 entries
1 bucket(s) with 1046 entries
1 bucket(s) with 909 entries
1 bucket(s) with 832 entries
1 bucket(s) with 815 entries
1 bucket(s) with 519 entries
1 bucket(s) with 514 entries
1 bucket(s) with 461 entries
1 bucket(s) with 434 entries
1 bucket(s) with 382 entries
1 bucket(s) with 336 entries
1 bucket(s) with 296 entries
1 bucket(s) with 281 entries
1 bucket(s) with 213 entries
1 bucket(s) with 189 entries
1 bucket(s) with 97 entries
1 bucket(s) with 86 entries
1 bucket(s) with 24 entries
1 bucket(s) with 12 entries
65482 bucket(s) with 0 entries

--------------------
Total collisions: 34877
SCORE = 1.09673e+08 (lower is better)
```

The fewer the collisions your function has the lower the score will be. The score is calculated as a sum of squares of number of collisions for each bucket. In other words, if a bucket has *n* entries, it contributes $(n-1)^2$ points to the score. This means it is better to have lots of buckets with a few collisions than a few buckets with lots of collisions.

A good score for this assignment is less than 50000.

# Bit Manipulation

You may want to investigate various ideas for bit manipulation and binary arithmetic for your hash function. For example, to turn an arbitrary integer value into a hash value in the range 0 through 65535 you can do:

```
x % 65536
```

Or equivalently but faster you can use the bitwise AND function:

```
x & 0xffff
```

Hash functions often use operations such as addition, subtraction, multiplication, bitwise AND, bitwise XOR, bit shifts left and right, and modulus.

# Pointer Arithmetic

Your hash function takes a pointer as input. Dereferencing the pointer with * gets the first character in the string. How do you get the remaining characters? There are a few different ways to do it, here are some examples:

```
*key        // dereference, gets first character
key[0]      // this is the same as *key, first character
key[1]      // gets second character
key++;      // increments key pointer, now *key will be second character
*(key + 1)  // dereference pointer with offset, will be second character
*key++      // first time evaluated will be first character
            // second time evaluated will be second character, etc.
```

# What to Turn In

All files you turn in for every assignment and lab should begin with a comment block that includes your name, CruzID, class, date, filename, short description of the file's role in the assignment, and any special instructions related to the file. Also create a file called README. The README file should have the normal comment block, then list all the files being submitted (including itself) along with any special notes to the graders. If you do pair programming, both names and CruzIDs should appear as author and in the README file.

For this lab, submit the following files:

```
README
Makefile
main.c
```

To submit, use the submit command.

```
submit cmps12b-nojw.f14 lab7 ...
```

Your makefile should have a default target that builds an executable called "hash", along with phony targets "clean" that removes compiled object files, "spotless" that cleans up all built files, and "test" which runs the executable with appropriate arguments to get a score.