COSC 320 - Advanced Data Structures and Algorithm Analysis
Lab 7

Dr. Joe Anderson

Due: 24 October 2019

1 Objectives

In this lab you will focus on the following objectives:

1. Practice implementing and studying hash functions
2. Develop familiarity with list-concepts and memory management in `c++`

2 Tasks

1. Put your code in a folder called "Lab-7". This folder will be zipped and turned in at the end.
2. Write a function: `size_t hash(size_t);` that will perform a hash of a given integer argument. Use the "multiplication method" to compute the hash:
   (a) Maintain two internal variables:
      i. $W = 2^w$ where w is the word size of the machine (typically 64). You can hard-code this.
      ii. $M = 2^p$ where p is a prime number less than w (but still reasonably large).
      iii. a is another prime number.
   (b) Given key x, return $\lfloor (ax \mod W)/2^{w-p} \rfloor$. Think about how to do the division and multiplication by powers of 2 quickly in the computer (use the "shift" operator!).
   (c) Why does this hash function make sense? Add an explanation in your README.
3. Write two different functions to hash a `string` argument. Use some creativity!
   (a) Consider what has to happen when the string is excessively long (e.g. a document). You may want to come up with a way to "combine" the hashes of multiple strings, then hash long documents by splitting and combining.
4. Write a function to convert your `size_t` to hexidecimal characters to be displayed in a more efficient way (you can use the `std::hex` object from the `iomanip` library.
5. Write a test program to demonstrate (clearly) the correctness of each of the above methods, displaying strings and their hashes.
6. Include a `Makefile` to build your code.
7. Include a `README` file to document your code, any interesting design choices you made, and **answer the following questions completely and thoroughly**:
   (a) Summarize your approach to the problem, and how your code addresses the abstractions needed.

(b) Explain your `string` conversion functions, and compare how they distribute the hashes of various strings.

(c) How could the code be improved in terms of usability, efficiency, and robustness?

3 Submission

All submitted labs must compile with your provided `Makefile` and run on the COSC Linux environment.

Upload your project files to MyClasses in a single `.zip` file.

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.

4 Bonus

(10 pts) Look up and use the `memcpy` function to create a templated hash function that first converts any arbitrary type to a pure byte string (e.g. `char*`) and then hashes it. This will be a relevant technique soon!