CS 241L - Data Organization

Spring 2022

Programming Assignment 3
Total points: 100
Due date: September 20, 2022 at midnight

This assignment will allow you to practice using the bitwise Boolean operators in C. Each section that follows will guide you through the steps to complete the assignment.

1 Extract files from tarball:

The necessary files are given to you as a compressed tarball. In order to extract the files from this tarball use the command: `tar -xvf programming_assignment_3.tar.gz`.
This will create a folder with name `'Programming Assignment 3'` in your current directory. The contents of this folder are the following:

- `Programming Assignment-3_F22.pdf`
- `hash_handout.c`
- `hash.out`
- `visualization.jar`
- `gettysburgOriginal.txt`
- `gettysburgAltered.txt`

2 Assignment:

Create a directory for this assignment as follows:

```
$ cd cs241
$ mkdir prog03
$ chmod 700 prog03
$ cd prog03
```

2.1 The Hashing technique

Hashing is a technique to radically compress data into a single integer. A hash code for a file is equivalent to the fingerprint of a human being. A fingerprint does not give you much information about the entire person, but fingerprints can be used to distinguish between two different people. In the same way, a file (or any source of data) can be associated with a 32-bit **hash code**. A good hash code should be a pseudo-random number computed from the given data. To create a hash code, we basically want to "mash" all the bits of the data into a single int.

Use the `visualization.jar` file provided in the tarball file, to see how this hashing process works on String data. This is a java program created in another university (credits provided separately in canvas) to visualize several algorithms covered in this class. So keep it handy; you will be executing several algorithms from it.

Instructions to use the visualization for this assignment:

- Open the jar file by double-clicking on it or by using the command `java -jar visualization.jar`
- Click on **Algorithms** along the top
- Click on **Hashing** in the drop-down menu
- Click on the **Hash Strings** button on the right
- Click on the **Pause** button in the lower left
- Type the string "alice" (without the quotes) in the TextField to the left of the Insert button
- Click on **Insert** button

You can now watch the computation of the hash code step-by-step by repeatedly clicking on the **Step** button in the lower left. Each of the five characters of "alice" are used, in turn, to contribute to

the final value of the hash code. The process repeatedly adds the 8 bits of the ASCII code of the next character into the running total. Remember, a **char** is really just a small number. The running total is then shifted 4 bits to the left. The 4 bits that are shifted-out of the running total are then XORed into the running total at bit positions 18-21 (counting bit positions from the right, starting with position 0). This shift-and-XOR is **not** performed for the last character of the input string. The hash value of "alice" is 6,827,925.

Notice that the input string "alice" is very short. But this same process can be used on a file containing billions of characters/bytes (such as a video file).

2.2 Your work on compressing text

Now that you have seen a visualization of the hashing code technique, consider the file `gettysburgOriginal.txt`. You can hash (aka compress) this very large (OK, not so very large) input into a single integer using the command [1]:

```
$ ./hash.out ‘‘$(cat gettysburgOriginal.txt)’’
```

The process of hashing is deeply connected to the concepts of bitcoin, blockchain, and digital signatures. If you change the input data in any way, even if only in a small way, then the resulting hash code will be completely different. Compare the hash codes of `gettysburgOriginal.txt` and `gettysburgAltered.txt`. The main purpose of blockchain is to be able to create signed contracts that can't be forged or repudiated or altered after the fact.

Your work then is to modify the given source code file `hash_handout.c` to carry out this hash computation. You should only need to write a small amount of code. Place your code in the section indicated in the file `hash_handout.c` Do not modify the source code in any other way. If at any time when you are running your program, and you are "stuck" (the program has entered an infinite loop, or is unresponsive) you can "bail out" and halt the execution of the program using ^C (i.e., hold the control and c keys simultaneously). For testing purposes, compile your file with the command:

```
$ gcc -std=c99 -pedantic -Wall hash_handout.c
```

You should be sure to include **YOUR NAME** in a comment at the top of your source code file, and make sure your code follows the code standards for this course (check out the standards in canvas in the `cs241_codingStandards2020.pdf` file). Rename your modified `hash_handout.c` file to a name that uses your last name and the initial of your first name, like this: `<lastName_initialFirstName>_hash_handout.c` Submit this renamed file for grading to Canvas in the place of this assignment.

[1] The hash.out program is provided in the tarball of this programming assignment and has been tested using the cs machines from Lab B146. You can use it for testing purpose. It accepts a single argument as input so any string with white space should use quotes.

3 What to submit:

Submit a tarball to Canvas with name `<Your-Student_ID>_3.tar.gz` containing the following files:

`<lastName_initialFirstName>_hash_handout.c`

A file `hash_gettysburgOriginal.txt` containing the hash value of the content of the file `gettysburgOriginal.txt`

A file `hash_gettysburgAltered.txt` containing the hash value of the content of the file `gettysburgAltered.txt`

The command for this is:
`tar -czvf <Your-Student_ID>_3.tar.gz <lastName_initialFirstName>_hash_handout.c hash_gettysburgOriginal.txt hash_gettysburgAltered.txt`
the command should be written in a single line, we added a new line for readability.

4 Grading Rubric:

If any of your C programs do not compile with the `-std=c99 -pedantic -Wall` options without errors or warnings the points given for the assignment will be zero. Otherwise the following rubric will be used:

+ 10 pt: Your C files follow the class coding standards.

+ 20 pt: The hash value of the contents of the file `gettysburgOriginal.txt` is correct.

+ 20 pt: The hash value of the contents of the file `gettysburgAltered.txt` is correct.

+ 50 pt: Your program correctly computes the hash value of a file only available to the grader.