CS 50 Homework 3:
hashtable.c

We all remember in Homework 0 we used a hashtable to implement string lookups and checks in our **`philspel`** program. Now it is time to convert a slightly simplified version of the hashtable code[1] into RISC-V assembly, implementing the functions **`createHashTable`**, **`insertData`**, and **`findData`**.

We provide three files for you: **`main.s`**, **`hashtable.s`**, and **`utils.s`**. You can load all three files at the same time into **`vrv`**., and they all contain suitable symbol references so that other files can access the public APIs within each file. You can modify **`main.s`** and **`utils.s`** if you desire (and indeed probably should add more test cases into **`main.s`**), but you will only turn in your **`hashtable.s`** file.

This assignment is due on May 13th at 2000 PDT (10:00 pm Davis time). You must work on this assignment individually.

Expected Homework Outcomes

1. Understand the process of converting C code into RISC-V assembly language programs

Hints

You are strongly encouraged to build your own copy of the RISC-V simulator. Instructions are available here:
https://docs.google.com/document/d/1gibQ88EylTDgLlui_FIv1qaE8c6ilb2NHsmJ_bTZ-Fc/edit

The **`utils.s`** functions include **`malloc`**, **`strcmp`**, **`printstr`**, and **`printhex`**.

The autograder version of these functions will also implement **`frathouse`** functionality: deliberately corrupting all caller saved registers before they return, corrupting any function which inadvertently relied on caller-saved registers.

The autograder will also use a **`campground`** function in its testing. This function accepts a pointer to a function in **`a7`**, allowing it to work with any function that accepts 7 or fewer arguments (as it doesn't manipulate **`a0-a6`**). It first saves all the callee-saved registers, puts known content into these registers, then it calls the function. When the function finishes it verifies that the callee saved registers are all properly restored before returning. Between these two operations the autograder can ensure that your code follows the RISC-V calling convention.

The autograder will have further functionality checks than the trivial version in **`main.s`**, including using different hash functions, a different **`main.s`**, a version of malloc that doesn't return zeroed data, and numerous campground and frathouse checks. The autograder is expected to be live by Tuesday.

[1] This version excludes the automatic resizing and the freeing of memory functionalities.