[50] Write a MIPS program called BuildEff.s that is similar to the C program you wrote in Homework #1. However, instead of reading in a file, your assembly program will read in lines of input from the console. Each line will be read in as its own input (using spim's syscall support for reading in inputs of different formats). The input is a series of building stats, where each building entry is 3 input lines long. The first line is a name to identify the building (a string with no spaces), the second line is the square footage of the building (an int, which you will need to typecast as a float later), and the third line is the annual amount of electricity used per year (in KwH – Kilowatt hours) (a float). After the last building in the list, the last line of the file is the string "DONE". For example:

```
HudsonAnnex
30000
522000.0
FitzpatrickCIEMAS
332178
4686414.2
DONE
```

Your program should prompt the user each expected input. For example, if you're expecting the user to input a building name, print to console something like "Building name: ". (remember the colon!)

Your program should output a number of lines equal to the number of buildings and each line is the building's name and energy efficiency in kWh per square foot per year. The lines should be sorted in descending order of kWh per square foot per year (least efficient building first). Buildings with equal efficiency should be sorted alphabetically. For example:

```
HudsonAnnex 17.4
FitzCIEMAS 14.10814
FancyHall 12.6
MessyHall 12.6
```

You may assume that pizza names will be fewer than 63 characters.

IMPORTANT: There is no constraint on the number of pizzas, so you may not just allocate space for, say, 10 building records; you must accommodate an arbitrary number of buildings. You must allocate space on the heap for this data. **Code that does not accommodate an arbitrary number of buildings will be penalized (-75% penalty)!** Furthermore, you may NOT first input all names and data into the program to first find out how many buildings there are and *then* do a single dynamic allocation of heap space. Similarly, you may not ask the user at the start how many buildings will be typed. Instead, you must dynamically allocate memory on-the-fly as you receive names. To perform dynamic allocation in MIPS assembly, I recommend looking here.

Note: You must follow calling conventions in this program. See "Calling convention rules" above.

Performance requirement: Automated GradeScope testing will take a while (~5-15 minutes) due to the slowness of the simulator and the size of the instructor BuildEff tests. Your BuildEff will need to be able to process 5000 buildings in 20 minutes in the GradeScope environment, else it will time out. This means that grossly inefficient solutions may not receive full credit (i.e., it might be too slow to copy every name