# COINS333 Applied Systems

Total 50 points.

You should **NOT** write the project using C++ and then translate. The goal is to be “Perlish” in your approach meaning you should use constructs and concepts specific to Perl. Code should be fairly easy to read and commented where not easy to read. Avoid using $\_ too often. Use hashes and patterns (regular expressions), ask questions if you don’t understand the spec. For each problem write a file named problem-*i*.pl. Submit a zip file hw8.zip containing all the pl files.

1. Write patterns to facilitate parsing of a subset of MIPS including only the mnemonics: add, addi, addiu, sub, lw, sw, and xori. Remember there are many ways of using some of these instructions. Note that patterns can contain variables. This means that you could create string variables as sub-patterns then combine them. This would allow you to reuse your own code and make patterns more readable. You should declare an array named @tests at the top of you file with test cases in it. E.g.:

@tests = ( ‘add $t1, $t2, $t3’, ‘addi $s1, $a0, 15’, ‘lw $t1, 4($t3)’, ‘lw $t3, LABEL’, lw $s1, ($sp)’…. );

Your code should do a loop through the @tests array and for each entry print a message like:

For test add $t1 $t2 $t3 the instruction is valid. Mnemonic is add with 3 parameters $t1, $t2, and $t3.

Your message should indicate if the instruction is valid or invalid. For valid instructions your message(s) should state what the mnemonic was, how many parameters, and what the parameters were. For invalid instructions your code should indicate what was wrong with it using as specific a message as possible. Possible errors would be: invalid mnemonic, wrong number of parameters, bad register name (like $a5 or $x1), wrong type of value for parameter 1 (for example, expected a register name and got an immediate), etc.

Read #2 before you code this.

1. Describe in a text, pdf, or MS Word file named problem-2 how you would extend your work in #1 to cover all of the MIPS instructions we did in class. Obviously, there will be data added to cover the new instructions. Describe how much there will be, how you will reuse the patterns and other code you wrote for #1. You need to come up with a solution that doesn’t involve the execution code (i.e. not the data, but the ifs and loops) scaling up linearly. In other words, your solution MUST NOT be an if for every mnemonic. Give pseudo-code or Perl code as needed to clarify or as examples. Be complete. The idea is that someone with reasonable knowledge of MIPS syntax and Perl should be able to take your #1 solution and #2 description and make a working parser.

I will give extra credit if your answer is particularly well thought out, complete, and represents a good algorithm.

1. Open and read a file named “num2.txt” (you will need to create the file for testing). Each line is of the form:

name city count

which represents the household (family) name, their city of origin, and the number of people in that household. For example:

Underwood Charleston 5

Underwood WarnerRobins 2

Smith NorthCharleston 3

Anderson Charleston 2

Jones NorthCharleston 8

Names and cities will be made ONLY of alphabetic characters. Charleston and charleston should be considered different cities (easier that way).

Your code should print out the total number of people in each city sorted alphabetically listed in the file and the total household per city. You do NOT know the cities in advance. Example output for the above would look like:

Charleston: 7 people in 2 households

NorthCharleston: 11 people in 2 households

WarnerRobins: 2 people in 1 household

For simplicity, read the entire file into an array and then iterate through the array exactly **ONE** time. If you can’t get through reading the array only ONE time, talk to me.

Bonus if you line up columns appropriately using printf and get the plurality of household correct.

1. Consider that you have an array holding student grades (for HW 1, 2, and 3) with the first line being column labels defined as follows:

@grade\_book= (

"User, First, Last, HW1, HW2, HW3, HW4, PROJECT1, PROJECT2, PROJECT3,",

"gunderwood,Gary,Underwood,90,88,70,,60,,92,",

"fworthy,Fred,Worthy,80,90,70,,80,,81,",

"vsessions,Valerie,Sessions,75,85,75,,90,,92,",

"ylin,Joseph,Lin,70,80,90,,95,,99,",

"jroberts,James,Roberts,75,81,95,,88,,78,",

);

You have another array with grades for students who submitted PROJECT2 consisting of username and grade pairs separated by a comma. It is defined as follows:

@project2=(

"vsessions,0",

"fworthy,82",

"gunderwood,100",

"jroberts,80",

"sperrine,30", #yes this is on purpose

);

For simplicity of your testing put these two array assignments in your file.

Then write code that does the following:

1. updates @grade\_book with the PROJECT2 grades found in @project2 by placing the score in the correct location in the lines. Print the updated grade book to the screen.
2. If there are any students listed in @grade\_book (e.g. ylin) that didn’t submit PROJECT2, then print the following message to the screen:

The following students did not submit PROJECT2: ylin

If multiple students didn’t submit, separate their usernames with spaces.

1. If there are any students who submitted PROJECT2 but are NOT in @grade\_book (e.g. sperrine), then print the following message to the screen (again separate usernames with spaces):

The following students were not found in the grade book: sperrine

For full credit, your code should only iterate through @grade\_book and @project2 exactly 1 time each. As a hint, iterate through @project2 completely setting up some useful data structure before using @grade\_book. So, for full credit your code should be **LINEAR** in the size of the two arrays. I may change the arrays during testing so don’t hardcode any positions or sizes.