CS 3500 – Programming Languages & Translators
Homework Assignment #8

- This assignment is **due by 11:59 p.m. on Friday, December 7th.**
- This assignment will be worth **4%** of your course grade.
- You are to work on this assignment **by yourself**.

Basic Instructions

For this assignment you are to write an **"automated grader script" in bash**. Basically, your script is to take a zip file containing student submissions of **Prolog** programs and execute each program for each input case (which is specified in a given set of input files), comparing the student's output to the respective (given) expected output, and finally creating an output file that has each student's score for the "entire assignment."

Consider the following example that describes more specifically what your script should do.

You will be given a file named **submissions.zip** that contains a Prolog file for each student submission. Each file in the zip file is named with the student's last name and first initial. For this example, assume submissions.zip just contains 3 files: simpsonh.pl, and duckd.pl, and leopoldj.pl.

You will be given a file named **sampleInput.zip** that contains a text file for each input case that we want to test a student's program on. For this example, assume sampleInput.zip just contains 2 files: **input1.txt** and **input2.txt**. These files will contain Prolog code that we want to call to test a student's program. For example, suppose the students were supposed to write a relation named *myReverse* that takes two (presumably list) parameters, and makes the second list the reverse of the first list. You may assume that the program is written so that it can be called from the command line with arguments that specify the elements of the list that we want to reverse. If we were testing a student's program, **input1.txt** might contain ***foo bar baz***, in which case the output should be ***[baz,bar,foo]***. As another example, **input2.txt** might just contain ***foo***, in which case the output should be ***[foo]***.

You also will be given a file named **expectedOutput.zip** that contains a text file for each input file that is in sampleInput.zip; these files are named the same as the sample input files, except they have **.out** on the end of the name. For this example, our expectedOutput.zip would contain 2 files: **input1.txt.out** (which would contain ***[baz,bar,foo]***, the result of reversing ***foo bar baz***) and **input2.txt.out** (which would contain ***[foo],*** the result of reversing ***foo***).

The first thing that your script should do is **unzip the sampleInput.zip file** into a directory (here we'll call that directory **sampleInput**), **unzip the expectedOutput.zip file** into a directory (here we'll call that directory **expectedOutput**), and **unzip**

submissions.zip into a directory (here we'll call that directory **submissions**). For documentation on the linux/unix *unzip* command, see:
https://linux.die.net/man/1/unzip

Next, you'll need to 'process' every entry in the **submissions** directory. This requires that, for every input file in your **sampleInput** directory, you invoke **swipl** (i.e., **prolog**) on that submission entry with that input file, compare the output to the respective output file in the **expectedOutput** directory, and keep a total of how many input cases the student got right out of the total number of cases tested. For each student you process, you should output this statistic (as a percentage), as well as the student's last name and first initial to a text file named **grades.txt**. For example, if simpsonh got 1 of the 2 input cases correct (50%), duckd got 0 of the 2 input cases correct (0%), and leopoldj got 2 of the 2 input cases correct (100%), then your grades.txt file should look like the following:

simpsonh, 50
duckd, 0
leopoldj, 100

In order to compare a student's output to the expected output, you may want to write the student's output to a file and then use ***diff*** on the 'actual' and 'expected' files; see http://ss64.com/bash/diff.html

You may assume that no student's submission will contain code that will "crash" swipl, and hence "crash" or "hang" your script ☹

Your script will need to invoke **swipl**, loading the student's Prolog file and telling **swipl** the arguments for the relation it will evaluate. **You may assume that every student's program will have a 'main' relation that will call whatever relation they were supposed to write using the command line arguments and print the result that that relation was supposed to compute**; for example:

```
#!/usr/bin/swipl

:- set_prolog_flag(verbose, silent).
:- initialization main.

main :-
  current_prolog_flag(argv, Argv),
  myReverse(Argv, X),
  print(X),
  halt.
main :-
  halt(1).

myReverse([], []).
myReverse([H | T], X) :- myReverse(T, Y), append(Y, [H], X).
```

If the above Prolog code were in a file named leopoldj.pl, it could be executed on a list containing **foo bar baz** by doing the following from the operating system prompt:

swipl leopoldj.pl foo bar baz

If you wanted to redirect the output of that command to a file, you would add **>** ***outputFilename***.

Sample files (i.e., 'submissions', 'sample inputs', and 'expected outputs') are provided on Canvas. Note that these files were created on Windows and may contain characters like '\r' that Unix/Linux doesn't like; **run dos2unix** on them before you use them. **These are NOT necessarily the files we will use for grading your homework! Furthermore, your script must work for any number of submissions and any number of test input files; do not assume you will get exactly the same number as are posted in the sample files we put on Canvas!** You may assume that the Prolog functions we test your programs on will not expect numeric parameters; *Argv* in *current_prolog_flag(argv, Argv)* treats the parameters as strings.

But wait…there's one more thing you need to do! Believe it or not, sometimes students hard-code into their programs the outputs for specific inputs that they expect the instructor will test their program on. We want your script to try to detect this by doing the following. If the student's submission produces correct output for an input case, your script must examine his/her source code to see if you can find that output (in any context – it doesn't have to be in a *print* statement); ***grep*** would be a good tool for doing this. If this occurs, we want you to output an asterisk ***** next to the student's score in the grades.txt file. This will inform us that we should manually grade that student's program! ☹

Warning: Your bash script **must** execute on one of the campus Linux machines. Various operating systems (e.g., Windows, Mac OS, etc.) do things a bit differently, particularly when it comes to file permissions, directory management, and versions of *sh*. Therefore, you should **test your script on the campus Linux machines before you submit it for grading!!! Excuses such as "but it runs fine on my Mac" will not be accepted** ☹

What to Submit for Grading

Via Canvas you should submit only your *sh* file. Name your *sh* file using **your last name followed by your first initial** (e.g., Bojack Horseman would name his file **horsemanb.sh**). You can submit multiple times before the deadline; only your last submission will be graded.

WARNING: If you fail to follow all of the instructions for submitting this assignment, your homework will NOT be graded!!!

The grading rubric is given below so that you can see how many points each part of this assignment is worth.

Functionality	Points Possible	Mostly or completely incorrect (0% of points possible)	Needs improvement (50% of points possible)	Mostly or completely correct (100% of points possible)
Script unzips submissions.zip into a separate directory	2			
Script unzips sampleInput.zip into a separate directory	2			
Script unzips expectedOutput.zip into a separate directory	2			
Script has a loop to process each submission	8			
Script has a loop to process each input case for each submission	8			
Script correctly calls swipl for each submission for each input case	8			
Script correctly compares each student's output to expected output	3			
Script correctly keeps score for each student	4			
Script correctly computes each student's percentage for 'assignment'	3			
Script correctly creates grades.txt containing 'assignment' percentages	5			
Script correctly detects 'suspicious' programs and marks them in grades.txt	5			

50