Introduction to Perl for Programmers

Final Project

Project overview

At the beginning of this class, you completed a multiple-choice entrance exam.
That exam was prepared and assessed semi-automatically: using Perl programs to generate the questions (with responses in random order), and also to assess the answers you provided.

This project requires you to construct and test similar software, and then improve it in various ways: by adding fuzzy matching to allow for transcription errors, by flagging unsatisfactory results for further scrutiny, and by highlighting possible instances of academic misconduct.

The project is expected to take you approximately 40 hours to research, design, implement, test, and document.

Project description

Main task (part 1a)
Randomization of questions

- Using the sample examination master files you have been provided with, write a Perl program that generates a random exam file from a master file, by processing each master file as follows:
    - remove the "correct answer" indicator (`[X]`) from any answer,
    - randomize the order of the answers for each question,
    - print each answer with an empty check-box (`[ ]`) in front of it.
- Your program should take a single argument: the name of the examination master file to be processed.
- Your program should write the post-processed contents of the examination master file to another file whose name begins with a timestamp in the form `YYYYMMDD-HHMMSS-<`*`FILENAME`*`>`, where `<`*`FILENAME`*`>` is the name of the original master file.
- For example, if the original master file was `IntroPerlEntryExam.txt`, then your program should write the modified contents to something like: `20220904-132602-IntroPerlEntryExam.txt`

Main Task (part 1b)
Scoring of student responses

- Write a Perl program that takes on the command-line the filename of an exam master file plus a list of filenames of completed exam files (such as the sample files you have been provided with). Your program should then compare each check-marked answer in each completed exam file against the correct answer from the appropriate examination master file, and compute how many answers were correct.
- Your program should accept two or more arguments: the first argument should be the name of the file containing the correct answers (*i.e.* the examination master file); the remaining arguments should be the names of each student's answer file that is to be scored.
- Your program must allow only one check-marked answer per question. If two or more answers have been check-marked for a single question, the score for that question is zero. Likewise, if no answer has been check-marked, the score is zero.
- If one answer has been check-marked, the score is 1 if the check-marked answer is the correct answer indicated in the examination master file. Otherwise, the score is zero.
- Your program should print out a list of the names of each student's answer file, with the total number of correct answers found in that file and the total number of questions for which any answer was provided. For example:

```
> score_exams.pl  correct_answers.txt  exam1/student_*.txt

exam1/student_000001.txt..........15/20
exam1/student_000042.txt..........18/20
exam1/student_Beeblebrox.txt........3/07
exam1/student_extra.txt...........11/12
```

- Your program should report any instance where a question from the examination master file is not present in any student's answer file (because this may indicate that the answer file has been garbled in transmission and that manual checking is therefore required).
- Your program should also report any case where any of the possible answers for the question was not present in the student's answer file (because this omission might make it impossible for them to give the correct answer, or easier to give the correct answer, and in either case that must be investigated further).
- So, for example, your program might report something like:

```
exam1/student_000001.txt:
    Missing question: What is the purpose of the 'use strict' pragma?

exam1/student_000042.txt:
    Missing answer: A hash stores exactly one random value
    Missing answer: A hash stores multiple ordered values
    Missing answer: A hash stores multiple unordered values
```

Extension (part 2)
Inexact matching of questions and answers

- To successfully complete Part 1b of the main task, it is sufficient to match questions and answers exactly (*i.e.* character-for-character), and to score any question or answer from the examination file as zero, if that question or answer does not appear in precisely the same form in the answer file.
- In this extension, your task is to allow for slight changes in the text of either the question or the answers…but still match them accurately.
- Modify your solution to Part 1b of the main task to allow for trivial differences in layout or casing, by "normalizing" each question and each answer before they are compared.
- Normalization consists of:
    - converting the text to lower-case;
    - removing any "stop words" from the text;
    - removing any sequence of whitespace characters at the start and/or the end of the text;
    - replacing any remaining sequence of whitespace characters within the text with a single space character.
- For example, a question such as:
  `"  What is the   airspeed of a fully laden African swallow?  "`
  with extra spaces and a newline in it, would be normalized to:
  `"what airspeed fully laden african swallow?"`
- Now further modify your solution so that normalized questions and answers in the student's file are compared against questions and answers in the master file using their "edit-distances" (*i.e.* their Levenshtein distance or Damerau–Levenshtein distance). You may either write your own implementation of this test, or use a module from CPAN.
- Your edit-distance test must accept a match between two normalized strings only if their edit-distance is no more than 10% the length of the normalized original string (*i.e.* 10% of the length of the normalized string from the examination master file).
- Your modifier program should report each case in which it accepts an inexact match between questions or answers. For example:

```
exam1/student_000001.txt:
    Missing question:  what is the purpose of the 'use strict' pragma?
    Used this instead: what's the purpose of the 'use strict' pragma?

exam1/student_000042.txt:
    Missing answer:    it disallows undeclared variables and barewords
    Used this instead: disallows undeclared variables or bearwords
```

Extension (part 3)

Analyzing cohort performance and identifying below-expectation results

- As your program scores each exam, you should accumulate statistics on the performance of all the students. Specifically: the number of questions answered correctly and the number of questions answered in total (either correctly or incorrectly).
- For each statistic, accumulate the average, minimum, and maximum values, and how many students achieve minimal and maximal scores.
- You may determine these statistics either by writing you own code, or by using a module from CPAN.
- Modify your program to compute these statistics as it processes each answer file, and then print them out once all the files have been processed. For example:

```
Average number of questions answered..... 18
                            Minimum.....  7   (1 student)
                            Maximum..... 20   (12 students)

Average number of correct answers........ 16
                          Minimum........  3   (1 student)
                          Maximum........ 20   (3 students)
```

- Then modify your program again, to detect and report all answer files whose results are significantly below expectation (and which must therefore be examined manually). You must decide yourself what "significantly" and "expectation" mean here. Be sure to document your chosen criteria.
- For example, you might choose to report all students who answer less than half the questions correctly, or who answer less than half the questions in any way, or you may choose to report all students whose scores are in the bottom 25% of results, or any students who correctly answer fewer than half of the questions they answer, or you may choose to report all students who are more than one standard deviation below the mean score for the group, or you may select some other criterion of your own devising. You may also choose to report combinations of two or more of these criteria.
- For each answer file you report, you must print out the name of the file, the student's score, and the reason it was reported.
- For example:

```
Results below expectation:
    exam1/student_Beeblebrox.txt........3/07  (score < 50%)
    exam1/student_extra.txt.............11/12  (bottom 25% of cohort)
    exam1/student_00042.txt.............13/20  (score > 1σ below mean)
```

Extension (part 4)
Identifying possible academic misconduct

- In many circumstances any collaboration between students to collectively answer exam questions, or any copying of one student's answers by another student, will be considered a serious academic infraction.
- Modify your program to detect and report when two or more answer files provide exactly the same set of answers (either right or wrong) to all questions…and therefore should be investigated for possible collusion.
- Of course, not all similar answer sets indicate cheating. If two or more answer files answer every single question correctly, that probably does not indicate any collusion *(they are simply good students)*. If two students answer the same 18 out of 20 questions correctly, that is also unlikely to be misconduct *(there were probably just two difficult questions)*. But if two or more students answer the same 11 out of 20 questions correctly, and also give identical wrong answers to the remaining 9 questions, that is much more likely to be evidence of a problem *(because the probability of two students randomly choosing the same 9 wrong answers in a 5-answer multiple-choice exam is just under 0.0004%)*.
- Devise a way of measuring how likely it is that two sets of independent answers would be exactly the same. Now modify your program to compute this metric for each pair of answer files, and to output a list of pairs of answer files that seem improbably similar, along with the computed value of your metric for each pair.
- For example:

```
Similar patterns of answers:

      exam1/student_000007.txt
  and exam1/student_000019.txt...........probability: 0.57

      exam1/student_000007.txt
  and exam1/student_000123.txt...........probability: 0.73

      exam1/student_000007.txt
  and exam1/student_002109.txt...........probability: 0.93

      exam1/student_000081.txt
  and exam1/student_006012.txt...........probability: 0.44
```

- Be sure to document your chosen metric and your rationale for choosing it.

Project assessment criteria

This project contributes 80% of your final result for the class,
and will be assessed as follows:

- Project documentation 20%
- Correct core functionality (solves main task parts 1a and 1b) 20%
- Code efficiency, clarity, and maintainability (including commenting) 10%
- Extension (part 2): Fuzzy matching of questions and answers 10%
- Extension (part 3): Statistics and below-expectation results 10%
- Extension (part 4): Identifying possible misconduct 10%

80%

Note that your project documentation must explicitly state which parts of the assignment your code is solving (*i.e. "Only Part 1"*, or *"Parts 1 and 3"*, or *"All Four Parts"*, or some other combination).

If your code solves one or more of the extensions, it is only necessary to submit a single solution for part 1b, where that single solution includes ***all*** the extension behaviours you have implemented.