ENGG1340 Programming Technologies / COMP2113 Computer Programming II

Assignment 1

Deadline: 1 March (Saturday), 2025 23:59

General Instructions

Submit your assignment via VPL on Moodle. Ensure that your program can execute, and generate the required outputs in VPL. Work incompatible with the VPL may not be marked.

For shell scripts (Problem 1 and 2), they must starts with the header `#!/bin/bash`, and will be executed using the Bash shell on our standard environment.

As a developer, ensure that your code works flawlessly in the intended environment, not just your own. While you may develop your work in your own environment, always test your program in our standard environment before submission.

Evaluation

For tasks requiring **user input**, utilize the **standard input**. Likewise, your program should **output**/**print** through the **standard output**. Strict adherence to the sample output format is required, or your answer may be marked incorrect.

Your code will be automatically graded for technical correctness. Essentially, we use test cases to evaluate your solution, failure to pass any of the test cases may result in zero marks. Partial credits are generally not given for incomplete solutions as it may be challenging to objectively assess incomplete program logic. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Additional test case will be used during grading. Scoring full marks on VPL does not ensure full marks in the assignment. Sample test cases may or may not encompass all boundary cases. Designing proper test cases to verify your program's accuracy is part of the assessment.

Academic dishonesty

Your code will be cross-checked with other submissions and online sources for logical duplication. Note that providing your work to others, aiding others in copying, or copying from others will be considered plagiarism, and will be dealt with as per departmental policy. Please refer to the course information notes for more details.

Use of generative AI tools, like ChatGPT, is not permitted for all assignment.

Getting help

You are not alone! If you are stuck, post your query on the course forum. This assignment should be educational and rewarding, not frustrating. We are here to help, but we can only do so if you reach out.

Please avoid spoilers on the discussion forum. Do not post any code directly related to the assignments. You are, however, encouraged to discuss general concepts on the forums.

Submission

Deadlines are strictly enforced. Resubmission beyond the submission period will not be accepted.

Late Policy:

- If you submit within 2 days after the deadline, 30% deduction.
- If you submit within 3-5 days after the deadline, 50% deduction.
- After that, no mark.

Problem 1: Count Substring Matches

Write a shell script that takes two **command line arguments** `substring` and `file`. It will count the words that contains `substring` in `file` and produce the result.

Input:

- The shell script does not read input from user. However, it expects two **command line arguments** `substring` and `file`.

Output:

- The script should list all words found, with the number of occurrences of that word in `file`. Refer to the sample outputs for the exact format.
- The words should be listed in descending order of the number of occurrences. For words with the same number of occurrences, they should be listed in ascending order of their ASCII values.
- The script should output nothing when there are fewer than two command line arguments specified or when the `file` does not exist.

Assumptions:

- The command line argument `substring` contains alphabets only. There will be no digits, symbols, or whitespace characters in `substring`.
- `file`, if exists, is a plain text file and is readable by all user.
- The locale settings of the shell can affect the result of sorting. The shell script will be executed using Locale "C". If you are testing in your own Linux environment, please execute command `export LC_ALL=C.UTF-8` to change the locale settings accordingly.

Requirements:

- For this question, a word is bounded by spaces or symbols, or by line boundaries (i.e., start of a line or end of a line). For example, the string `Gutenberg(TM)'s` should be treated as three words `Gutenberg`, `TM`, and `s`.
- Substring matching should be case insensitive. E.g., searching for `tale` should find `TALE` and `tale`.
- On the other hand, when counting the number of occurrences of a word, it should be done in a case-sensitive manner. E.g., `TALE` and `tale` should be counted separately.

Notes:

- A file `ebook.txt` is provided for testing. A different file may be used when grading your work.
- Study the man page of `grep` and `sort` to learn about possible options to use for this task.
- There is no need to follow the exact amount of leading spaces shown in the sample outputs. Leading spaces will be ignored in evaluation. If you are testing in your own environment, you can use flag `-Bw` of command `diff` for comparison.

Sample Test Cases

1_1

Command: `./1.sh tale ebook.txt`

Output:

```
      3 TALE
      2 Tale
```

1_2

Command: `./1.sh time ebook.txt`

Output:

```
     30 time
     10 times
      3 Sometimes
      1 lifetime
      1 oftentimes
      1 sentiment
      1 sometimes
```

1_3

Command: `./1.sh jerry ebook.txt`

Output:

```
     14 Jerry
```

1_4

Command: `./1.sh pokemon ebook.txt`

Output: *(it's empty)*

Problem 2: Credit card number validation

Write a Shell Script for validating credit card numbers using the Luhn algorithm.

The steps to validate a credit number using the Luhn algorithm are as follows:

1. Starting from the rightmost digit (that is the check digit), double the value of every second digit.
2. If the doubled value is a two-digit number, sum the digits of that number together to form a single digit.
3. Add all the 16 digits together.
4. If the final sum is divisible by 10, then the credit card is valid. If it is not divisible by 10, the number is invalid or fake.

For example, consider the credit card number `4512 3456 7890 1234`. Applying the Luhn algorithm:

- Double every second digit, starting from the right: 4, 6, 2, 2, 0, 18, 8, 14, 6, 10, 4, 6, 2, 2, 5, 8.
- Sum all the resulting digits: 4 + 6 + 2 + 2 + 0 + 9 + 8 + 5 + 6 + 1 + 4 + 6 + 2 + 2 + 5 + 8 = 70.
- Since 70 is divisible by 10, the credit card number is valid.

Input:

- The shell script reads one credit card number from user.

Output:

- The script should output a message reporting the validity of the credit card number. Refer to the sample outputs for the exact format.

Assumptions:

- You can assume that the input is always a 16-digit number, and each digit is in the range $[0, 9]$. There is no need to consider invalid inputs.

Sample Test Cases (Inputs are shown in blue)

2_1

```
Enter the number for checking:
4512345678901234
The number 4512345678901234 is valid.
```

2_2

```
Enter the number for checking:
4512345678901235
The number 4512345678901235 is invalid.
```

2_3

```
Enter the number for checking:
1234567890123456
The number 1234567890123456 is invalid.
```

2_4

```
Enter the number for checking:
1234567890123452
The number 1234567890123452 is valid.
```