ENGG1340 Programming Technologies / COMP2113 Computer Programming II

Assignment 1

Deadline: 24 February (Saturday), 2024 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 1.

Total marks: 100 marks

A maximum of 5 marks will be deducted if you fail to follow the submission instructions strictly.

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 3 tasks and a tester would automatically test your submitted program. If your submitted files and program outputs do not conform to the instructions given here, your programs cannot be evaluated, and you will risk losing marks totally.

Sample test cases are provided with each task in this document. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you can design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your assignment submission.

Input and output format

For tasks that require **reading input from user**, your answer should read from the **standard input**. Also, your program should **print** through the **standard output**. If you failed to follow this guide, the tester may not be able to give a score for your program. Additionally, you should strictly follow the sample output format, otherwise, your answer might be considered as wrong.

How to use the sample test cases

Sample test cases in text file formats are made available for you to check against your work. Here's how you may use the sample test cases. Take Problem 3 test case 1 as an example. The sample input and the expected output are given in the files `input3_1.txt` and `output3_1.txt`, respectively. Suppose that you compiled your program to an executable named `3`, do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./3 < input3_1.txt > myoutput.txt
diff -Bw myoutput.txt output3_1.txt
```

Testing against the sample test cases is important to avoid making formatting mistakes. The additional test cases for grading your work will be of the same formats as the sample test cases. Note that for this assignment, the flag `-Bw` will be used when checking the output of your programs. This will ignore leading/trailing spaces and empty lines when comparing outputs.

Coding environment

You must ensure that your program can compile (for C++ programs only), execute and generate the required outputs on our standard environment, i.e., the CS Linux servers (`academy*`).

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

For C++ programs, they must be compilable with the gcc C++11 environment on our standard environment. Make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 -o [executable name] [yourprogram].cpp
```

As a programmer/developer, you should always ensure that your code can work perfectly as expected on a target (e.g., your client's) environment, not only on yours.

While you may develop your work on your own environment, you should always try your program (compile & execute & check results) on our standard environment before submission. If your submitted files cannot be compiled using the compilation method stated above on our standard environment, or if they cannot be executed on our standard environment, your files may not be graded.

Submission

Name your shell scripts/C++ programs as the following table shows and put them together into one directory. Make sure that the folder contains only these source files (`*.cpp`) and no other files. **Compress this directory as a `[uid].zip` file where [uid] is your university number** and check carefully that the correct files have been submitted.

You are advised to prepare the zip file on the standard environment, then transfer the zip file to your own machine through FTP and submit it on Moodle. You should not transfer content of your files through copy-and-paste as the process is error prone. We suggest you to download your submitted file from Moodle, extract them, and check for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect/incomplete files.** Resubmission after the deadline is not allowed.

Filename	Description
`1.sh`	Problem 1
`2.sh`	Problem 2
`3.cpp`	Problem 3

Late submission

If submit within 3 days after the deadline, there will be 50% mark deduction. After that, no mark.

Evaluation

Your code will be auto-graded for technical correctness. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for logical redundancy. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism, and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Use of generative AI tools, like ChatGPT, is not allowed for the assignment.

Getting help

You are not alone! If you find yourself stuck on something, post your question to the course forum. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don't know when or how to help unless you ask.

Discussion forum

Please be careful not to post spoilers. Please don't post any code that is directly related to the assignments. However, you are welcome and encouraged to discuss general ideas on the discussion forums. If you have any questions about this assignment you should post them in the discussion forums.

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.

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.
- As flag `-Bw` will be used when grading your work (refer to **How to use the sample test cases** above), there is no need to follow the exact amount of leading spaces shown in the sample outputs.

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: Dictionary Lookup

Write a shell script that reads one word from the user and find all words in the system dictionary `/usr/share/dict/words` that start with the input word.

Input:

- The shell script reads one word from user.

Output:

- The script should list all words found in the same order as they are listed in the dictionary.
- When listing the words found, the matching part should be highlighted by wrapping it around two asterisks (`*`). Refer to the sample outputs for the exact format.

Assumptions:

- The input will always be one single word that contains alphabets only.

Requirements:

- All matching should be case insensitive.

Notes:

- The system dictionary `/usr/share/dict/words` may contain a different list of words in different systems. Your work will be graded using the dictionary on the academy server.

Sample Test Cases

2_1

Input:

```
compute
```

Output:

```
*compute*
*compute*r
```

2_2

Input:

```
zero
```

Output:

```
*zero*
*zero*axial
*zero*ize
```

2_3

Input:

```
Austri
```

Output:

```
*Austri*an
*Austri*anize
*Austri*c
*austri*um
```

2_4

Input:

```
aaa
```

Output: *(it's empty)*

```

```

Problem 3: Integer Triangles

Write a C++ program which takes one user input **integer** n $(0 \leq n \leq 100,000)$, and find all right-angled triangles with integer side lengths whose perimeter is n.

Input:

- The program reads one integer n from user.

Output:

- Each triangle should be denoted by its three side lengths, a, b, and c, where $a \leq b \leq c$.
- The program outputs all unique triangles, ordered by side lengths a, b, then c.

Note:

- Do not print any input prompt, your program will not pass any test cases if you do so.

Sample Test Cases

3_1

Input:

```
12
```

Output:

```
3 4 5
```

3_2

Input:

```
168
```

Output:

```
21 72 75
24 70 74
42 56 70
```

3_3

Input:

```
1320
```

Output:

```
110 600 610
120 594 606
220 528 572
231 520 569
264 495 561
330 440 550
352 420 548
```

3_4

Input:

```
1
```

Output: *(it's empty)*

```

```