Looping and Processing Files Part 1

Object Oriented Programming - Programming Coursework #4 (12 pts)

Due date : June 04, 2020 5pm

1 Task

The objective of this coursework is to gain a deeper insight into looping and file processing. The task is to implement two programs. The first program (Program1) will read data from a text file ('Input.txt') and generate an output text file ('Output.txt') according to the instructions given in Sec. 2 (4pts). The second program (Program2) will read inputs from a file ('Paragraph.txt') and it will also take inputs from the keyboard, and print an integer according to the instructions in Sec. 2 (8pts).

2 Program instructions and specifications

2.1 Program1

Input file: 'Input.txt' to be provided by the CW assessor. The first line gives an integer value corresponding to the total number of classes (C). From the second line, each line corresponds to a class: Each line starts with an integer representing the total number of students (T) in the respective class followed by real numbers corresponding to students' scores (S) within that class, each separated by a space. Here, C, T, and S are bounded in that $1 \leq C \leq 1000$, $1 \leq T \leq 100$, and $0 \leq S \leq 100$.

Output file: 'Output.txt' to be generated by Program1. Each output line should start with $\#H$ (H: class number starting from 1) and print the required answer after a space. After the class index, Program1 will print out the percentage of students, whose score is lower than the class average for each class. Program1 should print (in 'Output.txt') the results rounded off to six digits after the decimal point.

Sample inputs and outputs.

Program1 'Input.txt' file:

```
5
5 50 50 70 80 100
7 100 95 90 80 70 60 50
3 70 90 80
3 70 90 81
9 100 99 98 97 96 94 93 92 91
```

Program1 'Output.txt' file:

```
#1 40.000000%
#2 42.857143%
#3 33.333333%
#4 33.333333%
#5 44.444444%
```

2.2 Program2

Input file: 'Paragraph.txt' to be provided by the CW assessor. This file has a sequence of words each separated by a space.

User input: to be provided by the CW assessor using the keyboard. When started, Program2 should wait for user inputs. The user will then type a single word, e.g., 'network' (followed by enter key).

Output: to be displayed by Program2 in the monitor. Program2 will count the number of occurrences of the user-provided word in 'Paragraph.txt' and print the result as an integer. If no word in 'Paragraph.txt' matches the input word, the output should be 0.

Sample inputs and outputs.

Program2 input 'Paragraph.txt' file:

```
We investigate filter level sparsity that emerges in convolutional neural networks (CNNs) which employ Batch Normalization and ReLU activation and are trained with adaptive gradient descent techniques and L2 regularization or weight decay We conduct an extensive experimental study casting our initial findings into hypotheses and conclusions about the mechanisms underlying the emergent filter level sparsity This study allows new insight into the performance gap observed between adaptive and nonadaptive gradient descent methods in practice Further analysis of the effect of training strategies and hyperparameters on the sparsity leads to practical suggestions in designing CNN training strategies enabling us to explore the tradeoffs between feature selectivity network capacity and generalization performance Lastly we show that the implicit sparsity can be harnessed for neural network speedup at par or better than explicit sparsification pruning approaches with no modifications to the typical training pipeline required
```

Program2 user input (*Word*):

```
neural
```

Program2 output to be visualized in the monitor:

```
2
```

Note: 'Paragraph.txt' file will consist of only alphabetic and numeric characters, and spaces. A word is defined as a sequence of characters and two words are separated by a space. Program2 should count the occurrences of the input word only when that input finds exact matches at the word level: For instance, when the user-provided word is 'network', the output of Program2 should be 2: There are two occurrences of 'network' and a single case of 'networks'. However, since 'networks' does not match the input 'network' at the word level (even though 'network' is a prefix of 'networks'), it should not be counted as a match. Also, **the upper- and lower-case letters should not be distinguished**: if the input word is 'we', Program2 output should be 3 as there are two cases of 'We' and one case of 'we'.

2.3 Assessment

Program1 will be assessed based on five different instances of 'Input.txt'. 'Input.txt' file will be located at the same directory as the executable file of Program1. When executed, Program1 will generate 'Output.txt' file. Similarly, Program2 will be assessed based on five instances of 'Paragraph.txt' (in the same directory as the executable file of Program2) and user inputs (provided via the keyboard). It will print (i.e., display in the monitor) an integer value.

For this CW, your programs do not have to handle exceptions caused by incorrect input data types. For instance, for Program 2, 'Paragraph.txt' won't contain anything else then alphabetic and numeric characters and spaces.

3 What to hand in for assessment

Please submit the source codes for Program1 and Program2, and a brief report (a single report for both programs) via Black Board. Comments in the codes and the report should be written in English. The report should provide code comments explaining the algorithms. If the submission does not include a report, the maximum possible points will be kept at 40% of the full points. Please format the submission as a single zip file 'StudentID_Name.zip' that includes the files of the source codes and the report,

e.g.,
20201111_KwangInKim.zip
- Program1.cpp
- Program2.cpp
- Report.pdf

The submitted source codes should contain 'Program1.cpp' and 'Program2.cpp' files each including the id and name of the submitting student.

In 'Program1.cpp'

```
//StudentId Name
#include <iostream>
Your code here...
```

3.1 Compilation requirements and instruction

Each submitted code needs to be compilable by the GNU C++ compiler and in the Linux environment. Specifically, they should be **compilable and executable in UNI06 server**. Students can verify the correct compilation and execution of their codes by following the instructions provided in 'howtocompile.pdf' document accompanying the CW1 announcement.