INFO0027: Project 1 (20% – Groups of 2)
Scrabble Puzzle Solver

Laurent Mathy - Ben Mariem Sami

Submission before Friday, April 24^{th}

1 General requirements

In this project, you will implement a Scrabble Puzzle Solver in `C`.

Considering a list of words, a list of letters $[a - z]$ and a list of corresponding scores for the letters, your task is to find the set(s) of words from the list of words that can be formed with the letters and that result(s) in the maximum score.

Once a word has been picked from the list of words, it cannot be picked again, unless other copies of the word are still available in the list.

In addition to that, several sets of words could lead to a same maximum score. In that case, your solution must output all the sets that could be chosen.

Several examples illustrating the rules are provided in the Appendix.

2 Implementation

The implementation of your program is up to you, however you must provide a well structured code where each part has a well-defined role (e.g. "library" or "business" code). In other words, it is strictly forbidden to have a single file containing all the code.

The only restrictions concerns the name, the inputs and the outputs of your program. Indeed, your program, named `solver`, must take three command line arguments (in that order):

1. `file1`: the path to a text file that contains a list of words (one word per line).
2. `file2`: the path to a text file that contains a list of letters (one letter per line).

3. `file3`: the path to a text file that contains a list of score for each letter in file2 (one score per line).

As the score of each line must correspond to a letter, the number of lines of the last two text files must be equal. In addition to that, inputs may contain capital letters which should all be converted to lower-case letters.

Concerning the output, only the set(s) of words that has/have the highest score as well as the highest score must be displayed on the standard output using the form illustrated in Appendix.

Example files are provided[1].

3 Bonus Part (Max. 2 points)

As an extension to your solver, you may consider that words of the input list can be included several times in your set(s).

To implement this feature, your solver will have to take an additional argument as input which is the number of times each word can be reused.

An example of behaviour of your bonus feature is illustrated in the Appendix.

4 Remarks

Several remarks:

1. A newline is always marked by the line feed character (\n) and only by that character (not by a \r\n).
2. Assume that the input files contain only ASCII characters (no special symbol, no accented character).

5 Report

You must provide a report of maximum 5 pages. The report should contain a description of your algorithm as well as a performance study, showing the fitness for purpose of your solution to the problem. What to measure, and how, is left to your discretion.

We would also appreciate an estimate of the time you passed on the assignment, and what the main difficulties you encountered were.

[1] `http://www.montefiore.ulg.ac.be/~benmariem/Courses/INFO0027/Project/Project1/ressources.zip`

6 Evaluation and tests

Your program can be tested on the submission platform. A set of automatic tests will allow you to check if your program satisfies the requirements.
Depending on the tests, a **temporary** mark will be attributed to your work. Note that this mark does not represent the final mark. Indeed, other criteria such as the structure of your code, the efficiency, the correctness of other tests and your report will also be considered.

You are however **reminded** that the platform is a **submission** platform, not a test platform.

7 Submission

Projects must be submitted before Friday, April 24^{th}, 11:59pm.

After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of $2^N - 1$ marks (where N is the number of started days after the deadline).

Your submission will include your code (all the required `.c,.h` files and your `Makefile` at the root of the archive), along with a max five-page report (in English) explaining briefly your algorithms and presenting your performance study.

Submissions will be made, as a `.tar.gz` or `.zip` archive, on the submission system.

Your code must compile with `gcc`, on the *ms8*** machines, without error or warning. Failure to compile will result in an awarded mark of 0. Likewise, warnings and poor spatial or time performance when run over large input will systematically result in lost marks.

Bon travail...

Appendix

General Example

Consider the following input:

1. `words.txt`:

   AB AC AD

2. `letters.txt`:

   A B C D A B

3. `scores.txt`:

   5 7 3 2 5 7

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[[ab ac]] : 20

Empty Set

Consider the following input:

1. `words.txt`:

   AB AC AD

2. `letters.txt`:

   B B C D D

3. `scores.txt`:

   5 5 3 2 5

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[ ] : 0

Without Duplicates

Consider the following input:

1. `words.txt`:

   AB AC AD

2. `letters.txt`:

   A B C B A

3. `scores.txt`:

   5 7 3 7 5

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[[ab ac]] : 20

With Duplicates

Consider the following input:

1. `words.txt`:

   AB AB AC AD

2. `letters.txt`:

   A B C B A

3. `scores.txt`:

   5 7 3 7 5

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[[ab ab]] : 24

Same Letter, Different Scores

Consider the following input:

1. `words.txt`:

   AB AB AC AD

2. `letters.txt`:

   A B C B A

3. `scores.txt`:

   5 7 3 2 5

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[[ab ac]] : 20

Several Sets Solution

Consider the following input:

1. `words.txt`:

   AB AC AD

2. `letters.txt`:

   A B C D A

3. `scores.txt`:

   5 7 3 3 5

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[[ab ac] [ab ad]] : 20

Indeed, both sets lead to the same maximum score.

Negative Scores Letters

Consider the following input:

1. `words.txt`:

   AB AC AD

2. `letters.txt`:

   A B C D A B A

3. `scores.txt`:

   5 7 -5 -9 5 7 5

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt
```

Your solver should display the following output:

[[ab] [ab ac]] : 12

Bonus Feature

Consider the following input:

1. `words.txt`:

   AB AC AD

2. `letters.txt`:

   A B C D A B

3. `scores.txt`:

   5 7 3 2 5 7

and the following call to your solver:

```
./solver words.txt letters.txt scores.txt 2
```

Your solver should display the following output:

[[ab ab]] : 24