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

Laurent Mathy - Gaulthier Gain

Submission before Friday April 5

1 General requirements

In this project, you will implement a Word Puzzle Solver in `C`. Consider a $N \times N$ grid of letters (or multi-letters/symbols), your task is to find words hidden among the letters. A word can start at any location, and each succeeding letter of the word must be adjacent to the preceding letter horizontally, vertically, or diagonally.

Note: In the following examples, duplicates have been removed for readability reasons. Moreover, a dictionary file of 82000 words has been used.

There exist two types of grids: simple and complex. In simple grids, one cell equals one character. An example[1] of a simple grid is shown in Figure 1. Using a dictionary of English words (see Section 2), we find that this grid contains the following words: BA, BAN, BANG, BAD, BAUD, BAG, BUN, BUNG, BUD, BUG, BUY, AB, ABU, AN, ...

B A N
D U G
Z J Y

Figure 1: Simple grid (*example1*)

The solver must also be able to handle more complex grids. There exist two types: grids with "obstacle" symbol(s) or/and with multi-letters. A grid which contains "obstacle" symbols (which represent no-character) has a more limited number of words. Indeed, the role of this symbol is to "break" the formation of a word. For example, if we add some obstacle symbols (#) to the previous example like in Figure 2, the following words can no longer be formed: BAN, BANG, BUN, BUNG, BUY, AN, ...

[1] See Appendix for the complete example

B A #
D U G
Z J #

Figure 2: Complex grid with "obstacle" symbol(s) (*example2*)

Complex grids can also have multi-letters. This type of grid may has a smaller set of words too. The following example shows a complex grid with multi-letters.

BI A N
D UN G
Z J Y

Figure 3: Complex grid with multi-letters (*example3*)

In this case, we get the following words: BI BID, AN, AD, NAG, NUN, DAN, DUN, DUNN, DUNG, UN, UNA, GAD, GUN, JUNG.

2 Implementation

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

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

1. the path to a text file (one word per line) that contains a list of English words.
2. the path to a text file that contains the grid row by row (one row is one line of the file where the elements of the row are separated by a single space).

Concerning the output, only the words that have been found by the solver must be displayed on the standard output without any alteration (one per line). An example is provided in Appendix. If your solver finds no word, it should simply display an error message on *stderr* with the keyword `error`.

Example files, `words.txt` (650 words) and `grid_{2x2|3x3|4x4|5x5}.txt`, are provided[2].

[2] `http://www.montefiore.ulg.ac.be/~gain/courses/info0027.php#projects`

3 Bonus Part (Max. 2 points)

For larger grids, you may see a same word may appear multiple times. You can implement a bonus feature that avoids output duplicates. In this case, explain your methodology and your algorithm. Note that a performance study concerning this part is not necessary.

4 Remarks

1. Note that we cannot reuse the same cell several times for the same word.
2. Consider that a grid is **always** a square (2x2, 3x3, 4x4, 5x5, ...). For complex grids with multi-letters, consider that a multi-character represents one cell.
3. A newline is always marked by the backslash and only by that character (not by a \r\n).
4. Assume that the dictionary file contains only ASCII characters (no special symbol, no accented character).
5. The obstacle symbol is always #.
6. Your program must simultaneously handle simple (single character only) and complex grids (which contains the symbol # and/or with multi-characters).

5 Report

You must then realize 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 appreciate an estimation of the time you passed on the assignment, and what the main difficulties that you have encountered were.

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, another criteria such as the structure of your code, the efficiency, the correctness to 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 the Friday April 5, 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`), 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: Detailed output with a large dictionary file (82000 words)

In the following examples, duplicates are shown.

Simple grid output (*example1*)

```
./solver words.txt grid_3x3.txt
```

```
BA
BAN
BANE
BANE
BAD
BADE
BADGE
BE
BEAN
BEAD
BEN
BED
BEE
BEEN
BEG
AB
ABE
ABED
AN
AD
ADEN
AE
AE
NAB
NE
NEB
NED
NEE
NEG
NE
NEE
NEED
DBE
DAB
DAN
DANE
```

DANE
DE
DEB
DEAN
DEN
DEE
EN
ENE
ED
EE
EG
EN
ENE
EE
GENE
GEE
JEAN
JED
JEAN

Complex grid with "obstacle" symbol(s) (*example2*)

```
./solver words.txt grid_3x3.txt
```

BA
BAD
BAUD
BAG
BUD
BUG
AB
ABU
AD
AU
DAB
DAUB
DU
DUB
DUG
GAB
GAD
JUG