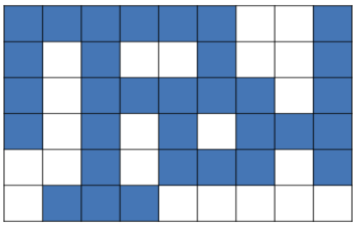
# Logic Programming - Crossword Puzzles

Step 1: Write a Prolog program step1.pl to solve the following specific crossword puzzle.



You are given these words to choose from:

PROLOG, PERL, ONLINE, WEB, GNU, NFS, SQL, MAC, EMACS,
XML, LINUX, JAVA, GOOGLE, PYTHON, PARSER, COOP, LOOP,
FORK, KERNEL, API, MOUSE, FIFO, PIPE

You cannot use predicates *assert*, *asserta*, *assertz* nor those predicates explicitly defined for I/O, like *read*, *write*, *print*. Among predicates for list processing, you can use *append*, *length*, *member*. All other predicates you need for list processing must be defined by yourself. Your query predicate should be named *crossword*.

Step 2: Generalize your solution in Step 1 to solve any crossword puzzles.

There are two input files for each puzzle:

- a plain text file words.txt containing a list of given words, one word per line;
- a plain text file binpattern.txt which contains a 0-1 matrix describing the crossword grid pattern.

The matrix for the grid pattern of the above puzzle is this:

111111001
101001001
101111101
101010111
001011101
011100000

You can assume that the matrix does not exceed the shape of 100*100.

Write a C program generator.c. Using the above two input files for a puzzle, this C program should generate a Prolog program like the one you have in Step 1.

When running your generator.c, the result should be written into a file called step2.pl. The search for the answer to the puzzle must be implemented in step2.pl, not in your C program.

In step2.pl, you cannot use predicates *assert*, *asserta*, *assertz*, *read*. Among predicates for list processing, you can use *append*, *length*, *member*. All other predicates you need for list processing must be defined by yourself. The predicates *open* and *write* can only be used in one statement (either a fact or a rule) to write the query result into a file.

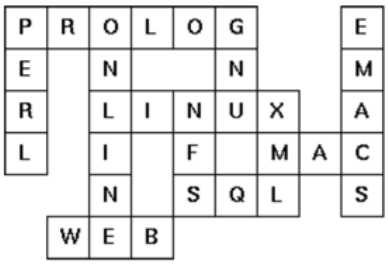
When running step2.pl, the query result should be written into a file called selectedwords.txt. This selectedwords.txt should contain the list of selected words. You can add some additional information into this file in regard of the locations of these words.

Write a C program display.c. It reads from selectedwords.txt. It may also read from words.txt and binpattern.txt. The output of this C program is the grid pattern with words inserted into it.

Illustration of the output:

```
-------------------------        -----
| P |  R  | O | L | O | G |      | E |
-------------------------        -----
| E |     | N |       | N |      | M |
-----     -----------------      -----
| R |     | L |  I  | N | U | X |    | A |
-----     -------------------------------
| L |     |  I |     | F |     | M | A | C |
-----     -----     -----------------------
          | N |     | S | Q  | L |      | S |
      -------------------------      -----
      | W | E | B |
      -------------
```

The above is to simulate the following sample answer:




Your readme file should contain the query in Step 1, and the explanations of its arguments. It should also include the explanations on how to understand your selectedwords.txt.

You should also provide two additional crossword puzzles.

- The number of the words in a puzzle should be greater than 5.
- The number of the given words should be more than double of the number of words in the puzzle.
- Provide the text files words1.txt and binpattern1.txt for your first example, and text files word2.txt and binpattern2.txt for your second example.
- Provide a PDF file colorpatterns.pdf containing the display of the grid patterns of the two puzzles. You can use any software tool to produce those grid patterns. Use different colors to display the patterns nicely.

Make sure that all executions can be done using the servers of the School of Computer Science.

Submission: one zipped file called YourLastName_YourFirstName containing

- step1.pl
- generator.c
- display.c
- readme.txt
- words1.txt
- binpattern1.txt
- words2.txt
- binpattern2.txt
- colorpattern.pdf