Homework #2

Deadline: Tuesday, 16 September 2014, 12:00 noon

Question 1 (1 point)

Solve Exercise 4.1 from the Lecture Notes (`plink/plonk` expressions).

Question 2 (1 point)

Solve Exercise 4.2 from the Lecture Notes (understanding operators).

Question 3 (2 points)

Imagine you have built a robot that can execute three different commands: turn *right* (by 90 degrees), turn *left* (by 90 degrees), and *move* forward (by 1 metre). Suppose you place your robot on a grid at position (0,0), facing north. Write a Prolog predicate `status/3` that will return the robot's position and orientation after having executed a given list of commands. For example, if your robot first moves forward twice, then turns right, and then moves three more times, then it will be at position (3,2), facing east. Examples:

```
?- status([move, move, right, move, move, move], Position, Orientation).
Position = (3,2)
Orientation = east
Yes

?- status([], Position, Orientation).
Position = (0,0)
Orientation = north
Yes

?-status([left, left, move], Position, Orientation).
Position = (0,-1)
Orientation = south
Yes
```

Start by writing a predicate `execute/5` for executing a single command: it should take the current position, the current orientation, and a single command (one of the atoms `right`, `left`, `move`) as input in the first three argument positions, and return the new position and orientation in the last two argument positions. Note that positions are pairs of the form `(X,Y)`, with `X` and `Y` representing integers, while the orientation has to be one of the four atoms `north`, `south`, `west`, `east`.

Then implement a predicate `status/5` that takes as input the current position, the current orientation, and the list of commands still to be executed, and that returns the final position and final orientation. That is, this predicate is like the predicate `status/3` you are ultimately supposed to implement, except that it also includes the current position and orientation as input. Finally, implement `status/3` as specified above.

Question 4 (2 points)

In 1742, in a letter to the famous mathematician Leonhard Euler, Christian Goldbach conjectured that every even integer greater than 2 can be expressed as the sum of two prime numbers. In his own words (quote taken from Wikipedia, 09/09/2014):

> *"Dass . . . ein jeder numerus par eine summa duorum primorum sey, halte ich für ein ganz gewisses theorema, ungeachtet ich dasselbe nicht demonstriren kann."*

To this date, nobody has been able to prove the truth of this statement for *all* integers, although it has been verified for very many of them with the help of computers.

Write a predicate called `goldbach/2` that, when given an even integer greater than 2 in the first argument position, will return an expression of the form `A + B`, such that both `A` and `B` are prime numbers and their sum is equal to the input number. Examples:

```
?- goldbach(30, Solution).
Solution = 7+23
Yes
```

```
?- goldbach(17420000, Solution).
Solution = 109+17419891
Yes
```

Start by implementing a predicate `prime/1` to test whether a given number is prime. Then think about what you need to do to find two prime numbers that add up to a given number `N`. First you need to choose the first number `A`, which can be any number between 3 and `N`/2 (think about why these are the correct bounds!). Then you need to check whether `A` really is prime. Then you need to compute `B` as the difference of `N` and `A`, and finally you also need to check whether `B` is prime. You may find the built-in predicate `between/3` useful.

Question 5 (2 points)

One of the major news stories involving AI in recent years has been about *IBM Watson*, a computer program that successfully competed in the American television game show *Jeopardy!*, beating the very best human contestants. Another famous television game show is *Countdown* (also known as *Cijfers en Letters*), and the purpose of this exercise is to see whether we can win this one for AI as well. We will focus on the *letters* game of the *Countdown* show. In this game, we are given nine letters of the alphabet (possibly including some repetitions). The goal then is to construct the longest possible word from these letters. Your score is the length of your word (provided it is a valid word of the English language).

Your ultimate task is to write a Prolog predicate `topsolution/3` to play this game. When given a list of nine letters in the first argument position, it should return as good a solution as possible, consisting of a word of the English language that can be constructed from those letters, in the second argument position and the length of that word (i.e., the score) in the third argument position. Example:

```
?- topsolution([g,i,g,c,n,o,a,s,t], Word, Score).
Word = agnostic,
Score = 8
Yes
```

Start by downloading the file `words.pl` from *Blackboard* (available under *Course Information*) and put it in the same directory as your program file. This is a list of a little over 350,000 of the most common words of the English language, from *a* to *zyzzyva*, presented as a sequence of facts, such as "`word(agnostic).`", etc. Include the line "`:- consult(words).`" in your program to make these facts available to you. Then proceed as follows.

First, search the SWI-Prolog reference manual for a built-in predicate for decomposing an atom into a list of characters. Use it to implement a predicate `word_letters/2` for converting a word (i.e., a Prolog atom) into a list of letters. Example:

```
?- word_letters(hello, X).
X = [h, e, l, l, o]
Yes
```

As an aside, note that you can use this predicate to find words with 45 letters:

```
?- word(Word), word_letters(Word, Letters), length(Letters, 45).
Word = pneumonoultramicroscopicsilicovolcanoconiosis,
Letters = [p, n, e, u, m, o, n, o, u|...]
Yes
```

Second, write a predicate `cover/2` that, given two lists, checks whether the second list "covers" the first. That is, it checks whether every item that occurs k times in the first list also occurs at least k times in the second. Examples:

```
?- cover([a,e,i,o], [m,o,n,k,e,y,b,r,a,i,n]).
Yes

?- cover([e,e,l], [h,e,l,l,o]).
No
```

Third, write a predicate `solution/3` that, when given a list of letters as the first argument and a desired score as the third argument, returns a word covered by that list that would provide the given score (i.e., the length of which is equal to the desired score). Example:

```
?- solution([g,i,g,c,n,o,a,s,t], Word, 3).
Word = act
Yes
```

Finally, implement `topsolution/3`. Document three examples for different lists of letters in your submission. One of them should be `[y,c,a,l,b,e,o,s,x]`. This was one of the lists used in the edition of *Countdown* aired in Britain on 18 December 2002, when Julian Fell achieved the best overall score in the history of the show. He found the word *cables*, earning him a score of 6. Can your program beat the champion?

Question 6 (2 points)

Solve Exercise 3.3 from the Lecture Notes (Fibonacci numbers). For full marks, give an implementation that is powerful enough to compute the 42nd Fibonacci number (and give the value of F_{42}). Make sure you use the exact definition given in the Lecture Notes.