**PR 2 – Logic Programming – First Term 2009/2010**
**Deadline: November 3rd, 2009, 23:55 WIB (uploaded on SCeLE)**
**Filename: `LP_PR2_YourName_YourNPM.pl`**
**Late submission is not allowed.**
**Credit for the problems goes to Walter Nauber, TU Dresden.**
**Note: to trace the execution in Prolog, use `trace` command. See the documentation.**

1. **[10]** Define a predicate `scalar(V1, V2, S)` which is true if `S` is the scalar product of two vectors of integers `V1` and `V2`. Example:

```
?- scalar([3,2,4], [6,1,5], S).
S = 40.
```

2. **[10]** Define a predicate `delduplelems(List, DList)` which is true whenever `DList` is obtained from `List` by deleting all duplicate elements in it, starting from the left. Example:

```
?- delduplelems([5,3,2,2,6,1,2,2,5,6],DList)
DList = [5,3,2,6,1].
```

3. **[10]** Define a predicate `deloneelem(Elem, L, RL)` which is true when `RL` is the list obtained from `L` by deleting the *first* occurrence of `Elem` in it.

```
?- deloneelem(4,[1,0,4,2,4,7],RL)
RL = [1,0,2,4,7].
```

4. **[10]** Define a predicate `convert(E, EL, DL, D)` which converts $E$ into $D$ with the help of conversion table encoded as lists $EL$ and $DL$. $E$ is an element of list $EL$, and $D$ is an element of list $DL$.

```
?- convert(c,[a,b,c,d],[3,7,8,10],D)
D = 8.
```

5. **[10]** Define a predicate `split(L,P,N)` which for a list of numbers $L$, it gives a list of nonnegative numbers of $L$ in $P$ and a list of negative numbers of $L$ in $N$. Give two versions of this predicate: the first version is written without cut, and the second is with cut. Example query:

```
?- split([4,-2,0,1,-7],P,N).
P = [4,0,1],
N = [-2,-7].
```

6. **[20]** Define a predicate `roman(N,R)` which is true when for a given decimal number $N$, $R$ is the string representing $N$ in roman numeral system.

```
?- roman(1999, R)
R = 'MCMXCIX'
```

Note: **Roman numerals** are letters used by the Romans for representation of cardinal numbers:
1 is represented as I, 5 by V, 10 by X, 50 by L, 100 by C, 500 by D and 1000 by M. Other numbers are represented by the shortest sequence of these letters with the required total value: their values are added except when a letter of lower denomination precedes one of higher in which case it is substracted from the total value; e.g.: IV is $4 = 5 - 1$, CD is $400 = 500 - 100$, but VI is $600 = 500 + 100$, etc. A value of a letter is substracted at most once: 8 is VIII not IIX.

Note: In this problem, the string representing a roman number is given as an atom (thus enclosed by single quotes), not a Prolog string (which is enclosed by double quotes).

Hint for implementation:

- The predicate `roman(N,R)` should convert a decimal number $N$ into the appropriate string of roman number $R$.
- For conversion, define an auxiliary predicate `numeral(N,NL,RL,R)` which converts $N$ into the string $R$ using a conversion table which is encoded as lists $NL$ and $RL$.
- Thus, the program should begin with:

```
roman(N,R) :-
  numeral(N,[1000,900,500,400,100,  90, 50, 40,  10,  9,   5,  4,   1],
            ['M','CM','D','CD','C','XC','L','XL','X','IX','V','IV','I'], R).
```

  (a) Add a fact for $N = 0$ which generates the empty string, i.e., `''`

  (b) Add a rule for `numeral` which recursively reduces the conversion table to `[N1 | NL2], [R1 | RL2]` with $N \geq N1$.
  E.g., $N = 25$ leads to $N1 = 10$, $R1 =$`'X'` and the query:
  `numeral(25,[10,9,5,4,1],['X','IX','V','IV','I'],R).`

  (c) Add a rule for `numeral` which repeats the conversion with the remainder $N2 = N - N1$, yielding string $R2$; concatenates strings $R1$ and $R2$ to the solution $R$ using built-in predicate `atom_concat`

7. **[30]** A typical cryptarithmetic puzzle is

```
  a b c
+ b c c
  -----
  c a a
  =====
```

This is the problem to assign decimal digits to letter `a, b, c`, etc., in a way that the above sum holds. Different letters must be assigned to different digits.

A possible solution to this problem is to generate, with the help of backtracking, conversion tables (from letters to digits), and then, converting lists of digits into numbers to check the correctness of the arithmetic addition.

**Hint for the solution:**

Define a predicate `cryptadd(S1,S2,R,D1,D2,D3)` which for given lists $S1$, $S2$, and $R$ of letters, checks if the corresponding arithmetic puzzle for addition of two numbers have a solution. An existing solution is then given as lists $D1$, $D2$, and $D3$ of digits corresponding to $S1$, $S2$, and $R$. Example:

```
?- cryptadd([a,b,c],[b,c,c],[c,a,a],D1,D2,D3)
D1 = [0,4,5],
D2 = [4,5,5],
D3 = [5,0,0].
```

Implementing `cryptadd`:

Define a predicate
`gendigits(L,DigitList,UsedLetters,UsedDigits,NewDL,NewUL,NewUD,D)`
which for a given **list** of letters $L$ produces a **list** of corresponding digits $D$. The idea here is the conversion table represented by the lists `UsedLetters` and `UsedDigits` is built "on the fly". In the conversion process, there are two cases.

- The letter occurs for the first time.
  The letter is then converted by a nondeterministic selection (with backtracking) of a digit from a list of digits `DigitList`. The letter is then put to the list `UsedLetters` and the corresponding digit is then put to the list `UsedDigits`. The digit should also be removed from `DigitList`.
- The letter is already converted before, i.e., it's already in the list `UsedLetters`. Here, the letter is then converted using the conversion table (the lists `UsedLetters` and `UsedDigits`)

The parameters `NewDL`, `NewUL` and `NewUL` represent the new lists for `DigitList`, `UsedLetters` and `UsedDigits` after complete conversion.

(a) Add a fact for predicate `gendigits` if list $L$ is empty.

(b) Add a rule for predicate `gendigits` to handle the case that the first letter from list $L$ is not in list `UsedLetters`. Use system predicate `member` for nondeterministic selection (backtracking) of a digit from list `DigitList`. Use predicate `deloneelem` in problem 3 to remove the selecte digit from `DigitList`. Put the letter and the selected digit into `UsedLetters` and `UsedDigits` respectively. Convert the next letter recursively.

(c) Add a rule for predicate `gendigits` fo the case that the first letter from list $L$ is already in the list `UsedLetters`. Convert this letter using the predicate `convert` in problem 4. Convert the next letter recursively.

(d) Add a rule for `cryptadd` which starts the conversion process for $S1$ using predicate `gendigits` with the list of digits `DigitList = [0,1,2,3,4,5,6,7,8,9]` and empty lists for `UsedLetters` and `UsedDigits`. After converting $S2$ and $R$ with new, updated lists for `DigitList`, `UsedLetters`, `UsedDigits`, translate the digits into numbers and check whether the addition of the cryptarithmetic puzzle holds.