Homework 3

1. Explore[1] the theory of natural numbers using Prolog. Remember,

   - 0 is a natural number,
   - if x is a natural number, then so is $s(x)$ (its successor), and
   - this is defined in Prolog as a unary predicate `is_nat/1` which is true when its argument is a natural number:

   ```
   is_nat(0).
   is_nat(s(X)):-is_nat(X).
   ```

   With 0 and s, the successor function, one can introduce the sum + of two natural numbers x, y:

   $$0 + y = y,$$
   $$s(x) + y = s(x + y).$$

   In Prolog (where we only have predicates), we introduced the ternary predicate `pluss/3`, which says that the sum of the first two arguments is the third one:

   ```
   pluss(0, Y, Y) :-
                  is_nat(Y).
   pluss(s(X), Y, s(Z)) :-
              pluss(X, Y, Z).
   ```

2. Write a predicate that recognizes palindromes[2].

3. Write a predicate for determining the maximum element from a list of integers.

4. Define the relation `lshift(List1, List2)` so that List2 is List1 "shifted rotationally" by one element to the left. Example:

   ```
   ?- lshift([1,2,3,4,5], L1), shift(L1,L2).

   L1=[2,3,4,5,1]
   L2=[3,4,5,1,2];

   No
   ```

5. Define the relation `rshift(List1, List2)` so that List2 is List1 "shifted rotationally" by one element to the right. Example:

   ```
   ?- rshift([1,2,3,4,5], L1), shift(L1,L2).

   L1=[5,1,2,3,4]
   L2=[4,5,1,2,3];

   No
   ```

[1]Define addition(done by me as an example), multiplication, exponentiation, less, less equal, divides, minus - note that only the binary version makes sense for natural numbers, divides

[2]A palindrome is a word, phrase, number or other sequence of units that has the property of reading the same in either direction (the adjustment of punctuation and spaces between words is generally permitted). [source:wikipedia.org]

6. Write a program for factorial computation using an accumulator.

7. Write a program delete_vowels(String, NoVowelsString) that deletes all vowels from a given string.

8. Write a program modify_string that changes a string by transforming all vowels into uppercase, all consonants in lower case letters and all other characters in 0.

9. Write a program sum_and_squaresum that from a given list of numbers finds the sum of its elements and the sum of their squares. Use accumulators. Example:

```
?- sum_and_squaresum([1,-3,2,0], Sum, SQS).

Sum = 0
SQS = 14 ;

No
```

10. Define a binary relation prefix/2 between lists and all its prefixes. Hint: [], [a] and [a, b] are prefixes of the list [a, b].

11. Define a binary relation suffix/2 between lists and all its suffixes. Hint: [], [b] and [a, b] are suffixes of the list [a, b].

12. Define a binary relation sublist/2 between lists and their sublists.

13. Implement the insert-sort algorithm for integers in Prolog – informally it can be formulated as follows:

    Given a list, remove its first element, sort the rest, and insert the first element in its appropriate place in the sorted list.

14. Implement the selection-sort algorithm for integers in Prolog – informally it can be formulated as follows:

    Given a list, find the minimum of the list, swap this minimum with the first possition, repeat the steps for the remainder of the list.

15. Implement the quick-sort algorithm for integers in Prolog – informally it can be formulated as follows:

    Given a list, split the list into two one part containing elements less than a given element (e.g. the first element in the list) and one part containing elements greater than or equal to this element. Then sort the two lists and append the results.

16. Implement the merge-sort algorithm for integers in Prolog – informally it can be formulated as follows: Given a list, divide the list into two halves. Sort the halves and merge the two sorted lists.

17. Write a predicate twice_as_long(L1,L2) that succeeds if the list L2 is twice as long as the list L1. Do NOT compute the lengths of the lists.

18. Write predicate fib(N,F) that is true if F is the Nth Fibonacci number[3]. Compute fib(5,F), fib(10,F), fib(50,F).

19. Implement Extended Euclidean Algorithm[4] to compute greatest common divisors in integers.

20. Write a predicate without_doubles_1(Xs, Ys) that is true if Ys is the list of the elements appearing in Xs without duplication. The elements in Ys are in the same order as in Xs with the last duplicate values being kept.

    Sample run:

```
?- without_doubles_1([1,2,3,4,5,6,4,4],X).
   X = [1, 2, 3, 5, 6, 4];

   No
```

21. Write a predicate without_doubles_2(Xs, Ys) that is true if Ys is the list of the elements appearing in Xs without duplication. The elements in Ys are in the reversed order of Xs with the first duplicate values being kept.

    Sumple run:

```
?- without_doubles_2([1,2,3,4,5,6,4,4],X).
   X = [6, 5, 4, 3, 2, 1];

   No
```

22. Write a ternary predicate delete_all(Item,List,Result) that is true if result is obtained from list by deleting all occurrences of item. Sample run:

```
?-delete_all(a,[a,b,c,a,d,a],X).
  X=[b,c,d]


?-delete_all(a,[b,c,d],X).
  X=[b,c,d]
```

23. Write a ternary predicate delete_first (Item,List,Result) that is true if result is obtained from list by deleting the first occurence of the element. Sample run:

```
?-delete_all(a,[a,b,c,a,d,a],X).

X=[b,c,a,d,a]


?-delete_all(a,[b,c,d],X).

X=[b,c,d]
```

[3]See http://en.wikipedia.org/wiki/Fibonacci_number

[4]See http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

24. Write a binary predicate count_occurrences(Input,Result) that is true if Result is a list of two-element lists [el, number_of_occurrences_in_input], where 'el' is an element of the list 'input', and 'number_of_occurrences_in_input' is an integer specifying how many times 'el' occurs in 'input'. For each element 'el' in 'input' there should be a corresponding pair [el, number_of_occurrences_in_input] in 'result'. Sample run:

```
?- count_occurrences([a,b,a,a,b,c],X).
  X = [[c, 1], [b, 2], [a, 3]] ;

No
```

25. Write a ternary predicate delete_nth that deletes every N'th element from a list. Sample runs:

```
?- delete_nth([a,b,c,d,e,f],2,L).

L = [a, c, e] ;

No
?- delete_nth([a,b,c,d,e,f],1,L).

L = [] ;

No
?- delete_nth([a,b,c,d,e,f],0,L).

No
?- delete_nth([a,b,c,d,e,f],10,L).

L = [a, b, c, d, e, f] ;

No
```