Prolog Problems

Status	Not started

▼ P1

1. a. Write a predicate to determine the lowest common multiple of a list formed from integer numbers.

```
% flow(i, i, o)
gcd(A, 0, A).
gcd(A, B, Res):-
    B\=0,
    R is A mod B,
    gcd(B, R, Res).

% flow (i, i, o)
lcml([], L, L).
lcml([H|T], C, Res):-
    gcd(H, C, R),
    L is (H*C)//R,
    lcml(T, L, Res).

main(List, Res):-
    lcml(List, 1, Res).
```

   b. Write a predicate to add a value *v* after *1-st*, *2-nd*, *4-th*, *8-th*, … element in a list.

```
% flow (i, i, o, o, o)
addVal([], _, _, _, []).
addVal([H|T], Val, Pos, Pos, [H, Val|Res]):-
    % if the position is valid, add the Value
    NewPos is Pos+1,
    NextPos is Pos*2,
    addVal(T, Val, NewPos, NextPos, Res).
addVal([H|T], Val, Pos, NextPos, [H|Res]):-
    NewPos is Pos+1,
    addVal(T, Val, NewPos, NextPos, Res).

main(List, Val, Res):-
    addVal(List, Val, 1, 1, Res).
```

2. a. Write a predicate to remove all occurrences of a certain atom from a list.

```
% flow (i, i, o)
rem([], _, []).
rem([Val|T], Val, Res):-
    rem(T, Val, Res).
rem([H|T], Val, [H|Res]):-
    H =\= Val,
    rem(T, Val, Res).
```

   b. Define a predicate to produce a list of pairs (atom n) from an initial list of atoms. In this initial list atom has
   n occurrences.
   Eg.: numberatom([1, 2, 1, 2, 1, 3, 1], X) ⇒ X = [[1, 4], [2, 2], [3, 1]].

```
incFreq([], Val, [[Val, 1]]).
incFreq([[Val, F]|T], Val, [[Val, NewF]|T]):-
    NewF is F+1.
incFreq([[H, F]|T], Val,[[H, F]|Res]):-
    incFreq(T, Val, Res).

numberatom([], F, F).
numberatom([H|T], ListF, Res):-
    incFreq(ListF, H, NewListF),
    numberatom(T, NewListF, Res).

main(List, Res):-
    numberatom(List, [], Res).
```

3. a. Define a predicate to remove from a list all repetitive elements.
   Eg.: l=[1,2,1,4,1,3,4] ⇒ l=[2,3])

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).

% flow (i, o, o)
removeRep([], _ ,[]).
removeRep([H|T], Removed, Res):-
    isInList(T, H),!,
    removeRep(T, [H|Removed], Res).
removeRep([H|T], Removed, Res):-
    isInList(Removed, H),!,
    removeRep(T, Removed, Res).
removeRep([H|T], Removed, [H|Res]):-
    removeRep(T, Removed, Res).

main(List, Res):-
    removeRep(List, [], Res).
```

b. Remove all occurrence of a maximum value from a list on integer numbers.

```
findMax([], -1, 0).
findMax([], C, M):-
    M is C.
findMax([M|T], CM, F):-
    M > CM,
    findMax(T, M, F).
findMax([_|T], CM, F):-
    findMax(T, CM, F).

removeMax([], _,[]).
removeMax([M|T], M, Res):-
    removeMax(T, M, Res).
removeMax([H|T], M, [H|Res]):-
    H =\= M,
    removeMax(T, M, Res).
```

```
main(List, Res):-
   findMax(List, -1, M),
   removeMax(List, M, Res).
```

4. a. Write a predicate to determine the difference of two sets.

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).



diff([], _, []).
diff([H|T], L, Res):-
    isInList(L, H), !,
    diff(T, L, Res).
diff([H|T], L, [H|Res]):-
    diff(T, L, Res).
```

   b. Write a predicate to add value 1 after every even element from a list.

```
add([], []).
add([H|T],[H, 1|Res]):-
    H mod 2 =:= 0,
    add(T,Res).
add([H|T], [H|Res]):-
    add(T, Res).
```

5. a. Write a predicate to compute the union of two sets.

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).

union([], O, O).
union([H|T], O, Res):-
    isInList(H, O),
    union(T, O, Res).
union([H|T], O, [H|Res]):-
    union(T, O, Res).
```

b. Write a predicate to determine the set of all the pairs of elements in a list.
Eg.: L = [a b c d] ⇒ [[a b] [a c] [a d] [b c] [b d] [c d]].

```
findPairs(_, [], []).
findPairs(H, [X|T], [[H, X]|Res]):-
    findPairs(H, T, Res).

makePairs([],[]).
makePairs([H|T], Res):-
    findPairs(H, T, Pairs),
    makePairs(T, PairsFromTail),
    append(Pairs, PairsFromTail, Res).
```

6. a. Write a predicate to test if a list is a set.

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).



isSet([]):-
    true.
isSet([H|T]):-
    isInList(T, H),!,
    false.
isSet([_|T]):-
    isSet(T).
```

b. Write a predicate to remove the first three occurrences of an element in a list. If the element occurs less than three times, all occurrences will be removed.

```
remove([], _, _, []).
remove([H|T], V, Pos, [H|Res]):-
    H =\= V,
    remove(T, V, Pos, Res).
remove([V|T], V, Pos, Res):-
    Pos < 3,
    NewPos is Pos + 1,
    remove(T, V, NewPos, Res).
remove([V|T], V, 3, [V|Res]):-
    remove(T, V, 3, Res).

main(List, Val, Res):-
    remove(List, Val, 0, Res).
```

7. a. Write a predicate to compute the intersection of two sets.

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).

intersect([], _, []).
intersect(_, [], []).
intersect([H|T], Other, [H|Res]):-
    isInList(Other, H),
    intersect(T, Other, Res).
intersect([_|T], Other, Res):-
    intersect(T, Other, Res).
```

b. Write a predicate to create a list (m, ..., n) of all integer numbers from the interval [m, n].

```
createList(Current, End, []):-
    Current > End.
createList(End, End, [End]).
createList(Current, End, [Current|Res]):-
    NewCurr is Current + 1,
    createList(NewCurr, End, Res).
```

```
main(M, N, Res):-
    createList(M, N, Res).
```

8. a. Write a predicate to determine if a list has even numbers of elements without counting the elements from the list.

```
even([]).
even([_]):-
    false.
even([_,_|T]):-
    even(T).
```

b. Write a predicate to delete first occurrence of the minimum number from a list.

```
minimum([], CM, M):-
    M is CM.
minimum([H|T], CM, M):-
    H < CM,
    NewMin is H,
    minimum(T, NewMin, M).
minimum([_|T], CM, M):-
    minimum(T, CM, M).

removeMin([], _, _, []).
removeMin([M|T], M, 0,Res):-
    removeMin(T, M, 1, Res).
removeMin([M|T], M, 1, [M|Res]):-
    removeMin(T, M, 1,Res).
removeMin([H|T], M, X, [H|Res]):-
    removeMin(T, M, X, Res).

main(List, Res):-
    minimum(List, 100, M),
    removeMin(List, M, 0, Res).
```

9. a. Insert an element on the position n in a list.

```
addE(X, 1, [H|T], [X, H|T]).
addE(X, Y, [H|T], [H|T1]):-
    Y1 is Y-1,
    addE(X, Y1, T, T1).
```

b. Define a predicate to determine the greatest common divisor of all numbers from a list.

```
gcd(A, 0, A).
gcd(A, B, Res):-
    B\=0,
    R is A mod B,
    gcd(B, R, Res).

gcdList([A], A).
gcdList([A, H|T], R):-
    gcd(A, H, X),
    gcdList([X|T], R).
```

10. a. Define a predicate to test if a list of an integer elements has a "valley" aspect (a set has a "valley" aspect if elements decreases up to a certain point, and then increases.

    eg: 10 8 6 9 11 13 – has a “valley” aspect

```
valley(_, [], 1).
valley(E1,[E2|T], 0):-
    E1 > E2,
    valley(E2, T, 0).
valley(E1,[E2|T], 0):-
    E1 < E2,
    valley(E2, T, 1).
valley(E1, [E2|T], 1):-
    E1 < E2,
    valley(E2 ,T, 1).
valley(E1, [E2|_], 1):-
    E1 > E2,
    fail.
```

```
main([H|T]):-
    valley(H, T, 0).
```

b. Calculate the alternate sum of list's elements (l1 - l2 + l3 ...).

```
altSum([], _, Sum, End):-
    End is Sum.
altSum([H|T], Pos, Sum, EndSum):-
    % even
    Pos mod 2 =:= 0,
    NewPos is Pos + 1,
    NewSum is Sum - H,
    altSum(T, NewPos, NewSum, EndSum).
altSum([H|T], Pos, Sum, EndSum):-
    % odd
    Pos mod 2 =\= 0,
    NewPos is Pos + 1,
    NewSum is Sum + H,
    altSum(T, NewPos, NewSum, EndSum).

main(List, S):-
    altSum(List, 0, 0, S).
```

11. a. Write a predicate to substitute an element from a list with another element in the list.

```
substitute(_, _, [],[]).
substitute(E1, E2, [E1|T], [E2|Res]):-
    substitute(E1, E2, T, Res).
substitute(E1, E2, [E2|T], [E1|Res]):-
    substitute(E1, E2, T, Res).
substitute(E1, E2, [H|T], [H|Res]):-
    H =\= E1,
    H =\= E2,
    substitute(E1, E2, T, Res).
```

b. Write a predicate to create the sublist (lm, ..., ln) from the list (l1,..., lk).

```
sublist(_, _, N, [], []).
sublist(Pos, _, N, _, []):-
    Pos > N.
sublist(Pos, M, N, [H|T], [H|Res]):-
    Pos >= M,
    Pos =< N,
    NewPos is Pos + 1,
    sublist(NewPos, M, N, T, Res).
sublist(Pos, M, N, [_|T], Res):-
    Pos < M,
    NewPos is Pos + 1,
    sublist(NewPos, M, N, T, Res).

main(List, M, N, Res):-
    sublist(1, M, N, List, Res).
```

12. a. Write a predicate to substitute in a list a value with all the elements of another list.

```
substitute(_, _, [],[]).
substitute(E1, E2, [E1|T], Res):-
    substitute(E1, E2, T, TailRes),
    append(E2, TailRes, Res).
substitute(E1, E2, [H|T], [H|Res]):-
    H =\= E1,
    substitute(E1, E2, T, Res).

main(List, Elem, List2, Res):-
    substitute(Elem, List2, List, Res).
```

b. Remove the *n*-th element of a list.

```
removeNth([], _, _, []).
removeNth([_|T], Pos, N, Res):-
    Pos =:= N,
    NewPos is Pos + 1,
    removeNth(T, NewPos, N, Res).
removeNth([H|T], Pos, N, [H|Res]):-
    NewPos is Pos + 1,
    removeNth(T, NewPos, N, Res).
```

13. a. Transform a list in a set, in the order of the last occurrences of elements. Eg.: [1,2,3,1,2] is transformed in [3,1,2].

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).

turnSet([], []).
turnSet([H|T],Res):-
    isInList(T, H),
    turnSet(T, Res).
turnSet([H|T], [H|Res]):-
    turnSet(T, Res).
```

b. Define a predicate to determine the greatest common divisor of all numbers in a list.

```
gcd(A, 0, A).
gcd(A, B, Res):-
    B\=0,
    R is A mod B,
    gcd(B, R, Res).

gcdList([A], A).
gcdList([A, H|T], R):-
    gcd(A, H, X),
    gcdList([X|T], R).
```

14. a. Write a predicate to test equality of two sets without using the set difference.

```
equal([],[]).
equal([H|T], Set):-
    member(H, Set),
    delete(Set, H, Reduced),
    equal(T, Reduced).
equal(List1, List2):-
    length(List1, N),
    length(List2, N),
    equal(List1, List2).
```

b. Write a predicate to select the n-th element of a given list.

```
selectN([], _, _,_).
selectN([H|T], N, N, H):-
    NewPos is N + 1,
    selectN(T, NewPos, N, H).
selectN([_|T], Pos, N,Res):-
    NewPos is Pos + 1,
    selectN(T, NewPos, N,Res).
```

15. a. Write a predicate to transform a list in a set, considering the first occurrence.

Eg: [1,2,3,1,2] is transform in [1,2,3].

```
isInList([H|_], H).
isInList([H|T], V):-
    H =\= V,
    isInList(T, V).

listToSet([], Acc, Acc).
listToSet([H|T], Acc, Res):-
    isInList(Acc, H),
    listToSet(T, Acc, Res).
listToSet([H|T], Acc, Res):-
    \+ isInList(Acc, H),
    listToSet(T, [H|Acc], Res).
```

```
main(List, Res):-
    listToSet(List, [], RevList),
    reverse(RevList, Res).
```

b. Write a predicate to decompose a list in a list respecting the following: [list of even numbers list of odd numbers] and also return the number of even numbers and the numbers of odd numbers.

```
separateLists([], [], [], 0, 0).
separateLists([H|T], [H|EvenNrs], OddNrs, EvenLen, OddL
    H mod 2 =:= 0,
    separateLists(T, EvenNrs, OddNrs, NewEvenLen, OddLe
    EvenLen is NewEvenLen + 1.
separateLists([H|T], EvenNrs, [H|OddNrs], EvenLen, OddL
    H mod 2 =:= 1,
    separateLists(T, EvenNrs, OddNrs, EvenLen, NewOddLe
    OddLen is NewOddLen + 1.
```

▼ P2

1. Sort a list with removing the double values. E.g.: [4 2 6 2 3 4] → [2 3 4 6]

```
selectMin([H], H, []).
selectMin([H|T], Min, Res):-
    selectMin(T, Min, Res),
    H =:= Min.
selectMin([H|T], Min, [H|Res]):-
    selectMin(T, Min, Res),
    H > Min.
selectMin([H|T], H, [Min|Res]):-
    selectMin(T, Min, Res),
    H < Min.

selectionSort([],[]).
selectionSort(List, [Min|SortedT]):-
    selectMin(List, Min, Res),
    selectionSort(Res, SortedT).
```

2. Sort a list with keeping double values in resulted list. E.g.: [4 2 6 2 3 4] -→ [2 2 3 4 4 6]

   → basic selection sort

```
selectMin([H], H, []).
selectMin([H|T], Min, [H|Res]):-
    selectMin(T, Min, Res),
    H >= Min.
selectMin([H|T], H, [Min|Res]):-
    selectMin(T, Min, Res),
    H < Min.

selectionSort([],[]).
selectionSort(List, [Min|SortedT]):-
    selectMin(List, Min, Res),
    selectionSort(Res, SortedT).
```

3. Merge two sorted lists with removing the double values.

```
removeDuplicates([],[]).
removeDuplicates([H|T], Res):-
    member(H, T),
    removeDuplicates(T, Res).
removeDuplicates([H|T], [H|Res]):-
    removeDuplicates(T, Res).

mergeList([],[],[]).
mergeList([], List, List).
mergeList(List, [], List).
mergeList([H|T1], [H|T2], [H|Res]):-
    mergeList(T1, T2, Res).
mergeList([H1|T1], [H2|T2], [H1|Res]):-
    H1 < H2,
    mergeList(T1, [H2|T2], Res).
mergeList([H1|T1], [H2|T2], [H2|Res]):-
    H1 > H2,
    mergeList([H1|T1], T2, Res).
```

```
main(List1, List2, Res):-
    mergeList(List1, List2, Temp),
    removeDuplicates(Temp, Res).
```

4. Write a predicate to determine the sum of two numbers written in list representation.

```
numToList(0, []).
numToList(Number, [LastDigit|Res]):-
    LastDigit is Number mod 10,
    NextNumber is Number//10,
    numToList(NextNumber, Res).
numberToList(Number, Res):-
    numToList(Number, ReverseList),
    reverse(ReverseList, Res).

sumList([], [], _, []).
sumList([], List, _, List).
sumList(List, [], _, List).
sumList([H1|T1], [H2|T2], C, [H|Res]):-
    H is (H1 + H2 + C) mod 10,
    NC is (H1+H2)//10,
    sumList(T1, T2, NC, Res).

sumMain(A, B, S):-
    numToList(A, AList),
    numToList(B, BList),
    sumList(AList, BList, 0, SRev),
    reverse(SRev, S).
```

5. Substitute all occurrences of an element of a list with all the elements of another list.
   Eg. subst([1,2,1,3,1,4],1,[10,11],X) produces X=[10,11,2,10,11,3,10,11,4].

```
substitute(_, _, [],[]).
substitute(E1, E2, [E1|T], Res):-
    substitute(E1, E2, T, TailRes),
    append(E2, TailRes, Res).
```

```
substitute(E1, E2, [H|T], [H|Res]):-
    H =\= E1,
    substitute(E1, E2, T, Res).

main(List, Elem, List2, Res):-
    substitute(Elem, List2, List, Res).
```

6. Determine the product of a number represented as digits in a list to a given digit.
   Eg.: [1 9 3 5 9 9] * 2 ⇒ [3 8 7 1 9 8]

```
prod([], _, 0, []).
prod([], _, C, [C]):-
    C > 0.
prod([H|T], P, C, [H1|Res]):-
    Temp is H * P + C,
    H1 is Temp mod 10,
    C1 is Temp // 10,
    prod(T, P, C1, Res).

main(List, Prod, Res):-
    reverse(List, Rev),
    prod(Rev, Prod, 0, ResRev),
    reverse(ResRev, Res).
```

7. Determine the position of the maximal element of a linear list.
   Eg.: maxpos([10,14,12,13,14], L) produces L = [2,5].

```
findMax([], CM, M):-
    M is CM.
findMax([H|T], CM, M):-
    H > CM,
    NewMax is H,
    findMax(T, NewMax, M).
findMax([_|T], CM, M):-
    findMax(T, CM, M).

% (List, CurrentPos, Max, Res)
```

```
findPosMax([], _, _, []).
findPosMax([M|T], Pos, M, [Pos|Res]):-
    NewPos is Pos + 1,
    findPosMax(T, NewPos, M, Res).
findPosMax([_|T], Pos, M, Res):-
    NewPos is Pos + 1,
    findPosMax(T, NewPos, M, Res).

main(List, Res):-
    findMax(List, -1, M),
    findPosMax(List, 1, M, Res).
```

8. Determine the successor of a number represented as digits in a list. Eg.: [1 9 3 5 9 9] → [1 9 3 6 0 0]

```
incList([], _, []).
incList([H|T], C, [H1|Res]):-
    H1 is (H + C) mod 10,
    NC is (H + C) // 10,
    incList(T, NC, Res).

main(List, S):-
    reverse(List, Rev),
    incList(Rev, 1, ResRev),
    reverse(ResRev, S).
```

9. For a list of integer number, write a predicate to add in list after 1-st, 3-rd, 7-th, 15-th element a given value e.

```
addVal([], _, _, _, []).
addVal([H|T], Val, Pos, Pos, [H, Val|Res]):-
    % if the position is valid, add the Value
    NewPos is Pos+1,
    NextPos is Pos*2 + 1,
    addVal(T, Val, NewPos, NextPos, Res).
addVal([H|T], Val, Pos, NextPos, [H|Res]):-
    NewPos is Pos+1,
    addVal(T, Val, NewPos, NextPos, Res).
```

```
main(List, Val, Res):-
    addVal(List, Val, 1, 1, Res).
```

10. For a list of integer numbers, define a predicate to write twice in list every prime number.

```
is_prime(2).
is_prime(N) :-
    N > 2,
    \+ has_divisor(N, 2).

has_divisor(N, Current) :-
    Current * Current =< N,
    N mod Current =:= 0.
has_divisor(N, Current) :-
    Current * Current =< N,
    Next is Current + 1,
    has_divisor(N, Next).

% (List, Res)
duplicatePrime([], []).
duplicatePrime([H|T], [H, H|Res]):-
    is_prime(H),
    duplicatePrime(T, Res).
duplicatePrime([H|T], [H|Res]):-
    duplicatePrime(T, Res).
```

11. Replace all occurrences of an element from a list with another element e.

```
occ([], _, _, []).
occ([E|T], E, R, [R|Res]):-
    occ(T, E, R, Res).
occ([H|T], E, R, [H|Res]):-
    H =\= E,
    occ(T, E, R, Res).
```

12. Define a predicate to add after every element from a list, the divisors of that number.

```
div(E, Curr, []) :-
    Curr > E.
div(E, Curr, [Curr|Res]) :-
    Curr =< E,
    E mod Curr =:= 0,
    Next is Curr + 1,
    div(E, Next, Res).
div(E, Curr, Res) :-
    Curr =< E,
    E mod Curr =\= 0,
    Next is Curr + 1,
    div(E, Next, Res).

addDiv([], []).
addDiv([H|T], [H|Res]):-
    div(H, 1, Div),
    append(Div, Temp, Res),
    addDiv(T, Temp).
```

13. Given a linear numerical list write a predicate to remove all sequences of consecutive values. Eg.: remove([1, 2, 4, 6, 7, 8, 10], L) will produce L=[4, 10].

```
% (List, Res)
remCons([], []).
remCons([X], [X]).
remCons([A, B|T], Res):-
    B =:= A + 1,
    skipCons(A, [B|T], Next),
    remCons(Next, Res).
remCons([A, B|T], [A| Res]):-
    B =\= A+1,
    remCons([B|T], Res).

% (PrevElem, List, Res)
```

```
skipCons(_, [], []).
skipCons(Prev, [Curr|T], Res):-
    Curr =:= Prev + 1,
    skipCons(Curr, T, Res).
skipCons(_, List, List).
```

14. Define a predicate to determine the longest sequences of consecutive even numbers (if exist more maximal sequences one of them).

```
longest_even_sequence([], []).
longest_even_sequence(List, Result) :-
    find_sequences(List, [], Sequences),
    longest_sequences(Sequences, Result).

find_sequences([], [], []).
find_sequences([], Current, [Current]) :-
    Current \= [].
find_sequences([H|T], Current, Sequences) :-
    H mod 2 =:= 0,
    append(Current, [H], NewCurrent),
    find_sequences(T, NewCurrent, Sequences).
find_sequences([H|T], Current, [Current|Sequences]) :-
    H mod 2 =\= 0,
    Current \= [],
    find_sequences(T, [], Sequences).
find_sequences([H|T], [], Sequences) :-
    H mod 2 =\= 0,
    find_sequences(T, [], Sequences).

longest_sequences([], []).
longest_sequences(Sequences, Longest) :-
    findall(Seq, (member(Seq, Sequences), length(Seq, L
    list_to_set(Filtered, Longest).

max_length(Sequences, MaxLen) :-
    maplist(length, Sequences, Lengths),
    max_list(Lengths, MaxLen).
```

15. Define a predicate to determine the predecessor of a number represented as digits in a list. Eg.: [1 9 3 6 0 0] ⇒ [1 9 3 5 9 9]

```
% (list)
% H = 0 -> (H + 10) - 1 -> -1 after
dec([], _, []).
dec([H|T], C, [H1|Res]):-
    H =:= 0,
    H1 is (H + 10 + C) mod 10,
    dec(T, C, Res).
dec([H|T], C, [H1|Res]):-
    H =\= 0,
    H1 is (H + C) mod 10,
    dec(T, 0, Res).



% (List, Res)
main(List, Res):-
    reverse(List, RevList),
    dec(RevList, -1 ,RevRes),
    reverse(RevRes, Res).
```

16. Sa se adauge in lista in fata fiecarui element non-prim divizorii lui: [1, 2, 8] → [1, 2, 8, 1, 2, 4, 8]

```
div(E, Curr, []) :-
    Curr > E.
div(E, Curr, [Curr|Res]) :-
    Curr =< E,
    E mod Curr =:= 0,
    Next is Curr + 1,
    div(E, Next, Res).
div(E, Curr, Res) :-
    Curr =< E,
    E mod Curr =\= 0,
    Next is Curr + 1,
    div(E, Next, Res).
```

```
% add(List, Res)
add([], []).
add([H|T], [H|Res]):-
    div(H, 1, Div),
    length(Div, L),
    L =\= 2,
    append(Div, Temp, Res),
    add(T, Temp).
add([H|T], [H|Res]):-
    add(T, Res).
```