Programowanie w Prologu

1. Reprezentacja list.

```
- lista pusta []
- lista niepusta   [E1, E2, ..., Ek | L]
```

2. Przykłady prostych predykatów działających na listach

2.1. lista(L) wiw gdy L jest listą

```
lista( [] ).
lista( [ _ | L ] ) :-  lista( L ).
```

2.2. lista_int(L) wiw gdy L jest listą liczb całkowitych

```
lista_int( [] ).
lista_int( [ X | L ] ) :-
          integer( X ), lista_int( L ).
```

2.3. first(L, X) wiw gdy X jest pierwszym elementem listy L
last(L, X) wiw gdy X jest ostatnim elementem listy L

```
first( [ X | _ ], X ).

last( [ X ], X ).
last( [ _ | L ], Y ) :-  last( L, Y ).
```

2.4. prefix(L, LL) wiw gdy L jest prefiksem listy LL

```
prefix1( L, LL ) :-  append( L, _, LL ).

prefix( [], L ).
prefix( [ X | L ], [ X | LL ] ) :-
       prefix( L, LL ).
```

2.5. suffix(L, LL) wiw gdy L jest suffixem (końcówką) listy LL

```
suffix1( L, LL ) :-  append( _, L, LL).

suffix( L, L ).
suffix( L, [ X | LL ] ) :-  suffix( L, LL ).
```

2.6. wypisz(L) - wypisanie wszystkich elementów listy L

```
wypisz( [] ) :-  write( 'Lista jest pusta.' ), nl.
wypisz( [ X | L ] ) :-  write( X ), wypisz_r( L ).

wypisz_r( [] ) :-  nl.
wypisz_r( [ X | L ] ) :-  put( 0', ), write( X ),
                          wypisz_r( L ).
```

2.7. intersect(L1, L2) wiw gdy przecięcie L1 i L2 nie jest puste
(czyli istnieje X t.że X in L1 i X in L2)

```
intersect( L1, L2 ) :-
           member( X, L1 ), member( X, L2 ).
```

% generator elementów L1, test przynależności do L2

2.8. podlista(L1, L2) wiw gdy L1 jest (niekoniecznie spójnym) podciągiem L2

```
podlista( [], L ).
podlista( [ X | L1 ], [ X | L2 ] ) :-
              podlista( L1, L2 ).
podlista(  L1, [ _ | L2 ] ) :-
              podlista( L1, L2 ).
% podlista( [ X | L1 ], [ Y | L2 ] ) :-
%             X \= Y, podlista( [ X | L1 ], L2 ).
```

% zawsze tylko jedna odpowiedź, a nie np. dwie:
% :- podlista([1, 2], [1, 4, 1, 2]).

2.9. podlista(L1, L2) wiw gdy L1 jest spójnym podciągiem L2

```
podlista1( Y, XYZ ) :-
          prefix( XY, XYZ ), suffix( Y, XY ).
podlista2( Y, XYZ ) :-
          append( X, Y, XY ), append( XY, Z, XYZ ).

podlista( L, LL ) :-  prefix( L, LL ).
podlista( L, [ _ | LL ] ) :-  podlista( L, LL ).
```

2.10. Odwracanie list

(a) naiwne (b.kosztowne) odwracanie

```
nrev( [], [] ).
nrev( [ X | L ], Rev ) :-
     nrev( L, RL ), append( RL, [ X ], Rev ).
```

(b) odwracanie z akumulatorem

```
reverse( L, R ) :-  reverse( L, [], R ).

reverse( [], L, L ).
reverse( [ X | L ], A, R ) :-
        reverse( L, [ X | A ], R ).
```

2.11. Palindrom

```
palindrom( L ) :-  reverse( L, L ).

palindrom( L ) :-  palindrom( L, [] ).

palindrom( L, L ).
palindrom( [ _ | L ], L ).
palindrom( [ X | L ], A ) :-
          palindrom( L, [ X | A ] ).
```

2.12. Algorytm QuickSort

(a) wersja prosta (kosztowna — z użyciem **append**)

```
qsort( [], [] ).
qsort( [ X | L ], Sort ) :-
      partition( L, X, A, B ),
      qsort( A, L1 ),
      qsort( B, L2 ),
      append( L1, [ X | L2], Sort ).

partition( [], _, [], []).
partition( [ X | L ], Y, [ X | L1 ], L2 ) :-
          X <= Y,  !,
          partition( L, Y, L1, L2).
partition( [ X | L ], Y, L1, [ X | L2 ] ) :-
          X > Y,
          partition( L, Y, L1, L2).
```

(b) wersja z akumulatorem (bez **append**)

```
qsort( L, S ) :-  qsort( L, [], S ).

qsort( [], S, S ).
qsort( [ X | L ], Akum, Sort ) :-
      partition( L, X, L1, L2 ),
      qsort( L2, Akum, S2 ),
      qsort( L1, [ X | S2 ], Sort ).
```

Uwaga: kolejność rekurencyjnych wywołań **qsort** **dowolna**!

2.13. Rozpoznawanie słów, np. a^{n}b^n, $n \geq 1$ (lub $n \geq 0$)

(a) slowo(S) wiw gdy S jest słowem powyższej postaci

```
slowo( S ) :-  slowo( S, [] ).

slowo( [ a | R ], L ) :-  slowo( R, [ b | L ] ).
slowo( [ b | L ], [ b | L ] ).
% lub slowo( L, L ).
```

(b) slowo(S, Reszta) wiw gdy S = a^{n}b^n $\circ$ Reszta, $n \geq 1$

```
slowo( S, R ) :-  slowo( S, [], R ).

slowo( [ a | S ], A, R ) :-
       slowo( S, [ b | A ], R ).
slowo( [ b | S ], [ b | A ], R ) :-
       append( A, R, S ).
```

Przykładowe zapytanie:

```
?- slowo( [a,a,b,b,b,c], Reszta ).

Reszta = [b,c];
no
?-
```