Ultimo elemento lista
```
last([X], L).
last([ _ | Xc], L) :-
      last(Xc, L).
```

Penultimo elemento lista
```
pen([P, _ ], P).
pen([ _, X | Xs],P) :-
      pen(P, [X|Xs]).
```

k-esimo elem lista
```
elementAt(X, [X | _ ], 0).
elementAt(X, [ _ | C ], Index) :-
      NewIndex is Index - 1,
      Index > 0,
      elementAt(X, C, NewIndex).
```

Inserisci alla posizione k
```
insertAt(N, 0, Xs, [N | Xs]) :- !.
insertAt(N, K, [X | Xs], [X | Rs]) :-
      KN is K-1,
      insertAt(N, KN, Xs, Rs).
```

Inserisci ordinato
```
sortedInsert(N, [], [N]) :- !.
sortedInsert(N, [X | Xs], [N, X | Xs]) :-
    N =< X, !.
sortedInsert(N, [X | Xs], [X | Rs]) :-
    N > X,
    sortedInsert(N, Xs, Rs).
```

Elimina il k-esimo elemento
```
deleteAt(0, [_ | Xs], Xs) :- !.
deleteAt(K, [X | Xs], [X | RP]) :-
    KN is K-1,
    deleteAt(KN, Xs, RP).
```

Lunghezza lista
```
lunghezza(0, []).
lunghezza(Len, [ _ | C]) :-
      lunghezza(NewLen, C),
      Len is NewLen+1.
```

Copia lista
```
copy([], [])
copy( [X | Rs], [X | Xs]) :-
      copy(Rs, Xs)
```

Inverti lista
```
reverse(L, LR) :-
      reverse(L, LR, []).
reverse([],LR,LR) :- !.
reverse([X | L], LR,LA):-
    reverse(L, LR, [X | LA]).
```

Lista palindroma
```
palindrome(L) :-
    reverse(L, Reversed),
    palindrome(L, Reversed).

palindrome([], []).

palindrome([X | Xs], [X | Xs]) :-
    palindrome(Xs, Xs).
```

Max lista
```
max([],0) :- !.
max([X | Xs], X) :-
   max(Xs, M),
   X > M,
   !.
max([X | Xs], M) :-
   max(Xs, M),
   X =< M.
```

max di 3 liste
```
max3(X1,X2,X3,M) :-
    max(X1,M1),
    max(X2,M2),
    max(X3,M3),
    max([M1,M2,M3], M).
```

Lista appiattita (esempio: [a, [b, [c, d], e]] → [a, b, c, d, e])
```
flatten([],[]) :- !.

flatten([X | Xs], Z) :-
   is_list(X),
   flatten(X, FlatX),
   append(FlatX, Y, Z),
   flatten(Xs, Y),
   !.

flatten([X | Xs],[X | Ys]) :-
    flatten(Xs,Ys),
    !.
```

Lista compressa

```
compress([], []) :- !.
compress([X], [X]) :- !.
compress([A, B | Xs], [A, Zs]) :-
      A \= S,
      !,
       compress([B | Xs], Zs).
compress([A, B | Xs], Zs) :-
      A = B,
      !,
      compress([A | Xs], Zs).
```

Sostituisci elem lista con altro

```
sostituisci(_, _, [ ], [ ]) :-
         !.
sostituisci(A, B, [X | Xs], [B | Ys]) :-
         A = X,
         !,
         sostituisci(A, B, Xs, Ys).
sostituisci(A, B, [X | Xs], [X | Ys]) :-
         A \= X,
         !,
         sostituisci(A, B, Xs, Ys).
```

Lista divisa in subliste per elemento
(esmepio: pack([a, a, b, c, c, c],[[a, a],[b],[c, c, c]]))

```
pack(L, [R | Rs]) :-
      pack(L, R, Rs, [R]).

pack([], _P, [Acc], Acc) :- !.

pack([X2 | Xs], X2, Y, [X2 | Acc]) :-
      pack(Xs, X2, Y, Acc),
      !.

pack([X2 | Xs], _PrecX, Z, Acc) :-
      append(Y, Z),
      pack(Xs, X2, Y, Acc),
      !.
```

Duplica elementi lista

```
duplica([],[]).
duplica([X | Xs], [X, X|Ys]) :-
      duplica(Xs, Ys).
```

Lista n-Plicata per carattere
(esempio: nPlicate([1,2],2,[1,1,2,2]))

```
nPlicate([],_,[]) :- !.

nPlicate([X | Xs], N, R) :-
      nPlicateSingle(X, N, Z),
      nPlicate(Xs, N, Z1),
      append(Z,Z1,R).

nPlicateSingle(_, 0, []) :- !.

nPlicateSingle(X, N, [X | Ys]) :-
     D is N - 1,
     nPlicateSingle(X, D, Ys).
```

Togli ogni ogni n-esimo elemento lista es. se dico 3 ogni 3 elem ne toglie uno

```
drop(L1,N,L2) :-
      drop(L1,N,L2,N).
drop([],_,[],_).
drop([_|Xs],N,Ys,1) :-
      drop(Xs,N,Ys,N).
drop([X|Xs],N,[X|Ys],K) :-
      K > 1,
      K1 is K - 1,
      drop(Xs,N,Ys,K1).
```

Split lista in 2 parti, la grandezza della prima metà è data

```
split(L,0,[],L).
split([X|Xs],N,[X|Ys],Zs) :-
      N > 0,
      N1 is N - 1,
      split(Xs,N1,Ys,Zs).
```

greaterThanN(N, L, R)

```
greaterThan(_N, [], []) :- !.

greaterThan(N, [X | Xs], [X | Ns]) :-
      X > N,
      greaterThan(N, Xs, Ns), !.

greaterThan(N, [X | Xs], Ns) :-
      X =< N,
      greaterThan(N, Xs, Ns).
```