Haskell you may assume (if allowed in the question)

Basic data types:

Class / Type	Functions
Bool	`&&  ||  not`
Tuple	`(,) fst snd`
List	`[] : head tail ++ !! length take zip`
Num	`+ - * abs`
Integral	`div mod even odd`
Fractional	`/`
Eq	`==  /=`
Ord	`>  >=  <=  <`
Show	`show`
Read	`read`
IO	`getLine getChar putStr do <- return let`

Basic types: Char, Int, Integer, Double

List functions:

List comprehension `[f x | x <- list, cond  x]`

`foldr` $\oplus$ v $[a1, a2, ..an] = a1 \oplus (a2 \oplus (... \oplus (an \oplus v)))$

`foldl` $\oplus$ v $[a1, a2, ..an] = (((v \oplus a1) \oplus a2) \oplus ...) \oplus an$

```
type  String = [Char]
data  Maybe a  =  Nothing | Just a
map :: (a -> b) -> [a] -> [b]
class Functor p where
   fmap :: (a -> b) -> p a -> p b
```

Prolog you may assume (if allowed in the question)

```
% dif(X,Y)  is true if X and Y denote different individuals
% X < Y  is true if expression X is less than Y
% X =< Y  is true if expression X is less than or equal to Y
% X > Y  is true if expression X is greater than Y
% Ordering on pairs (and tuples) is defined by the lexicographic ordering, as though
%     through the clauses:
(A,_) < (B,_) :- A<B.
(A,C) < (A,D) :- C<D.
% (a,b,c) is an abbreviation for (a,(b,c))

% append(A,B,C) is true if C contains the elements of A followed by the elements of B
append([],L,L).
append([H|T],L,[H|R]) :-
    append(T,L,R).

% reverse(L,R) true if R has same elements as L, in reverse order
reverse(L,R) :-
    reverse3(L,[],R).
% reverse3(L,A,R) is true if R consists of the elements of L reversed followed by the elements of A
reverse3([],R,R).
reverse3([H|T],Acc,R) :-
    reverse3(T,[H|Acc],R).
```