member(Prvek, Seznam) :- Prvek je prvkem Seznamu
select(Prvek, Seznam, CoZbude) :- CoZbude vznikne ze Seznamu vypuštěním jednoho výskytu Prvku
delete(Seznam, Prvek, CoZbude) :- CoZbude vznikne ze Seznamu vypuštěním všech výskytů Prvku
last(Seznam, Prvek) :- Prvek je posledním prvkem Seznamu
append(Xs, Ys, Zs) :- seznam Zs je zřetězením seznamů Xs a Ys
reverse(Xs, Ys) :- seznam Ys vznikne otočením seznamu X
between(X, Y, Z) :- Z je celé číslo splňující X<=Z<=Y
length(Xs, N) :- N je délka seznamu Xs
sort(Xs, Ys) :- Ys vznikne setříděním seznamu Xs a odstraněním duplicit
msort(Xs, Ys) :- Ys vznikne setříděním seznamu Xs (případné duplicity ponechány)

atom(X) :- X je atom
atomic(X) :- X je konstanta
number(X) :- X je číslo
var(X) :- X je volná proměnná
nonvar(X) :- X není volná proměnná
ground(X) :- X je základní term (bez volných proměnných)
compound(X) :- X je složený term
name(Konstanta, Retezec) :- Retezec je seznam znaků Konstanty
compare(Relace, Term1, Term2) :- Relace udává uspořádání termů Term1 a Term2
functor(Term, F, A) :- Term má hlavní funktor F a aritu A
arg(N, Term, A) :- A je N-tým argumentem Termu

maplist(Cil, Xs) :- uspěje, když Cil lze úspěšně aplikovat na každý prvek seznamu Xs
maplist(Cil, Xs, Ys) :- jako maplist/2, jen Cil se aplikuje na odpovídající dvojice prvků ze seznamů Xs a Ys
exclude(Cil Xs, Ys) :- uspěje, když Ys je seznam obsahující každý prvek X seznamu Xs pro nějž call(Cil, X) selže
call(Cil) :- zavolá Cil
call(Cil, X) :- zavolá Cil s argumentem X
call(Cil,X,Y):- zavolá Cil s argumenty X a Y

once(Cíl) :- vrátí první řešení, které splní Cíl
forall(Podminka, Cil) :- uspěje, pokud Cil lze splnit pro všechny hodnoty proměnných, pro než lze splnit Podminku

Můžete samozřejmě používat operátory jako
\+, = , \= , =.. , is , atd.

Naopak se prosím vyhněte
~~assert, retract, bagof, setof, findall~~

```
div, mod :: Integral a => a -> a -> a
(+), (*), (-) :: Num a => a -> a -> a
(/) :: Fractional a => a -> a -> a
(^) :: (Num a, Integral b) => a -> b -> a
max, min :: Ord a => a -> a -> a

(<), (<=), (>), (>=) :: Ord a => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
even, odd :: Integral a => a -> Bool

all :: (a -> Bool) -> [a] -> Bool
all odd [1,3..100] = True

any :: (a -> Bool) -> [a] -> Bool
any even [1..100] = True

fst :: (a,b) -> a

snd :: (a,b) -> b

sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24
maximum, minimum :: (Ord a) => [a] -> a
maximum [3,1,4,2] = 4
minimum [3,1,4,2] = 1

and, or :: [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True
reverse :: [a] -> [a]
reverse "abcde" = "edcba"

concat :: [[a]] -> [a]
concat ["ab","cd","e"] = "abcde"
(!!) :: [a] -> Int -> a
[9,7,5] !! 1 = 7
head :: [a] -> a
head "abcde" = 'a'
init :: [a] -> [a]
init "abcde" = "abcd"
null :: [a] → Bool
null [] = True

(++) :: [a] -> [a] -> [a]
"abc" ++ "de" = "abcde"
length :: [a] -> Int
length [9,7,5] = 3
tail :: [a] -> [a]
tail "abcde" = "bcde"
last :: [a] -> a
last "abcde" = 'e'
id :: a -> a
sort :: (Ord a) => [a] -> [a]

takeWhile :: (a->Bool) -> [a] -> [a]
takeWhile (<4) [1..6] = [1,2,3]
dropWhile :: (a->Bool) -> [a] -> [a]
dropWhile (<4) [1..6] = [4, 5,6]
zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3] ['a','b'] = [(1,'a'),(2,'b')]
zip With :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith (+) [1,2,3] [10,20] = [11,22]

take :: Int -> [a] -> [a]
take 3 "abcde" = "abc"
drop :: Int -> [a] -> [a]
drop 3 "abcde" = "de"
splitAt :: Int -> [a] -> ([a], [a])
splitAt 3 "abcde" = ("abc","de")
elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "abcde" = True

map :: (a -> b) -> [a] -> [b]
foldr :: (a -> b -> b) -> b -> [a] -> b
scanr :: (a -> b -> b) -> b -> [a] -> [b]

filter :: (a -> Bool) -> [a] -> [a]
foldl :: (a -> b -> a) -> a -> [b] -> a
scanl :: (b -> a -> b) -> b -> [a] -> [b]

(.) :: (b -> c) -> (a -> b) -> a -> c
odd = not . even

repeat :: a -> [a]
iterate :: (a -> a) -> a -> [a]

fromIntegral:: (Integral a, Num b) => a -> b
```