Matching terms are called Unification

Algorithm (Unification)

```
Unify(x, y, θ)
        x – a variable, constant, term or list
        y – a variable, constant, term, or list
        θ – a substitution so far
if(θ == fail)
        return fail
if (x(θ) == y(θ))
        return θ
else if ( var? (x))
        return unify-var(x, y, θ)
else if ( var? (y))
        return unify-var(x, y, θ)
else if ( term? (x) and term? (y))
        return (unify(args (x), args(y), unify( op(x), op(y), θ))
else if (list? (x) and list? (y)                    )
        return (unify(cdr x, cdr y, unify(car x, car y, θ)))
else
        return fail

unify-var(var, y, θ)
        var – a variable
        y – an expression
        θ – a substitution

if(var |-> val) exists in θ
        return unify(val, x, θ)
else if (x |-> val) exists in θ
        return unify(var, val, θ)
else if (occur-ck? (var, x))
        return fail;
else
        return (cons (var |-> x) θ)
```

Example: if x:= ((z*z) + 35)
so op(x) = +
op takes first outer operation
args(x) = (z*z 35)
args(x) contains 2 terms, the first is the left of the + symbol, the right is the right of the symbol

suppose x is f(var), then var occurs in f(var) and occur-ck of (var, x) is true!
(note: prolog doesn't do occur-ck)

Example

```
num(0)
num(s(x)) :- num(x)
1?- num(s (s (x)))
(returns fail)
```

Example

```
x' = g(h(x, y), z)
y' = g(w, t(v))
```

Initially, θ is ( )
Since x' and y' are both terms, we return unify ((h(x, y) z), (w t(v)), unify(g(a, b), g(a, b)))
notice that a, b are new variable names
the second unify will return θ = ( ) since g(a, b) are the same thing
So now we try to unify two lists (h(x, y) z, (w t(v)))

So we will do unify((z) (t(v)) unify (unify (h(x, y), w, θ) θ is still ( )
now we call unify-var (w, h(x,y), θ)
this statement will return (w |-> h(x, y))
We now need to unify (z) & (t(v)) with respected substitution (w |-> h(x,y'))
When we get the final recursive answer, we will have
((z |-> t(v)) (w |-> h(x,y)))

PROLOG

Prolog prompt looks like
l?-

Write your programs in your favorite text editor then save.
To load them into prolog, can use [myfilename.P]