SYMBOLIC DIFFERENTIATION

P.35

```
d( X , X , 1 ) .
d( C , = , 0 )   : -     atom( C ) .
d( [ - , U ] , X , [ - , A ] )   : -     d( U , X , A ) .
d( [ U , + , V ] , X , [ A , + , B] )   : -
                    d( U, X , A ) ,   d( V , X , B ) .
d( [ U , - ,  V ] , X , [ A , - , B ] )   : -
                    d( U , X , A ) ,  d( V , X , B ) .
d( [ C , * , X ] , X , C )   : -     atom( C ) .

| ? -   d( x , x , Q ) .
        Q = 1

| ? -   d( a , x , Q ) .
        Q = 0

| ? -   d( [ - , x ] , x , Q ) .
        Q = [ - , 1 ]

| ? -   d( [[ a , * , x ] , + , b ] , x , Q ) .
        Q = [ a , + , 0 ]
```

MORE DIFFERENTIATION

P.36

Rewrite the last rule to be more general.

```
d( [ C , * , U ] , X , [ C , * , A ] )  : -
                    atom( C ) ,
                    C   \= = X ,
                    d( U , X , A ) .

d( [ U , ^ , V ] , X , [ V , * , [U , ^ , [ V , - , 1 ] ] , * , W ] )
          : -       atom( V ) ,
                    V   \= = X ,
                    d( U , X , W ) .


| ? -   d( [ c , * , [ x , ^ , t ] ] , x , Q ) .
        Q = [ c , * , [ t , * , [ x , ^ [ t , - , 1 ] ] , * , 1 ] ]


| ? -   d( [ c , * , [ [ [ a , * , x ] , + , b ] , ^ , t ] ] , x , Q ) .
        Q = [ c, * , [ t , * , [ [ [ a , * , x ] , + , b ] ,
             ^ , [t , - , 1 ] ] , * , [ [ a , * , 1 ] , + , 0 ] ] ]
```

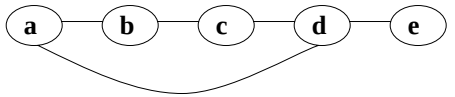
P.37

Last One

```
d( [ U , * , V ] , X , [ [ B , * , U ] , + , [ A , * , V ] ] )   : -
        d( U , X , A ) ,  d( V , X , B ) .

| ? -  d( [ [ [ a , * , x ] , + , b ] , * , [ [ c , * , x ] , + , d ] ], x , Q ) .

Q = [ [ [ [ c , * , 1 ] , + , 0 ] , * , [ [ a , * , x ] , + , b ] ] ,
        + , [ [ [ a , * , 1 ] , + , 0] , * , [ [ c , * , x ] , + , d ] ] ]
```

P.38

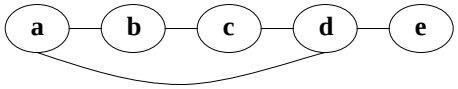
Alan's connected program




```
adjacent( a , b ) .
adjacent( b , c ) .
adjacent( c , d ) .
adjacent( d , e ) .
adjacent( a , d ) .

next_to( X , Y )  : -  adjacent( X , Y ) .
next_to( X , Y )  : -  adjacent( Y , X ) .

connected( A , B )  : -  connect_mark( A , B , [ A ] ) .

connect_mark( A , B , MarkedList )  : -
            next_to( A , B ) ,
            nonmember( B , MarkedList ) ,
            print( [ B  | MarkedList ] ) .

connect_mark( A , B , MarkedList )  : -
            next_to( A , C ) ,
            nonmember( C , MarkedList ) ,
            connect_mark( C , B , [ C | Marked List ] ) .

nonmember( X , L )  : -  not( member ( X , L ) ) .
```

P.39

Trying It




```
connected( e , a ) .
  connect_mark( e , a , [ e ] )
    next_to( e , a )
        adjacent( e , a )                fail
        adjacent( a , e )                fail
    fail
    next_to( e , c )
    ⋮
    next_to( e , d )
    nonmember( d , [ e ] )
    succeed
    connect_mark( d , a , [ d , c ] )
      next_to( d , a )
          adjacent( d , a )              fail
          adjacent( a , d )              succeed
      succeed
      nonmember( a , [ d , e ] )
      succeed
       print( [ a , d , e ] )
    succeed
  succeed
succeed
```

P.40

Inserting an Element (Somewhere) in a List

e **[e1 , e2 , e3 , e4]**

[e , el , e2 , e3 , e4]
[e1 , e , e2 , e3 , e4]
[e1 , e2 , e , e3 , e4]
[e1 , e2 , e3 , e , e4]
[e1 , e2 , e3 , e4 , e]

insert(X , L , [X | L]) .

insert(X , [H | T] , [H | U]) : - insert(X , T , U).

If you only ask for one solution, it will only use the first rule.

```
| ? -  insert( a , [ b , c , d ] , Ans ) .
       Ans = [ a , b , c , d ]
```

For a second answer, it goes on to the next rule.

```
| ? -  insert( a , [ b | [ c , d ] ] , [ b | U ] )
         insert( a , [ c , d ] , U2 )
         insert( a , [ c , d ] , [ a , c , d ] )
       insert( a , [ b | [ c , d ] ] , [ b , a , c , d ] )
```

Permutations of a List , Using Insert

P.41

```
permute( [ ],  [ ] ).
permute( [ H | T ],  L )   :-      permute( T, U ),
                                   insert( H, U, L ).
```

```
|?-  permute ([1, 2, 3], Q).
     permute ([1 | [2, 3]], Q)  :-   permute  ([2, 3], U), insert (1, U, Q).
       permute ([2, 3], U1)
       permute ([2 | [3]], U1)  :-   permute ([3], U2), insert (2, U2, U1)
         permute ([3], U2)
          permute ([3 | NIL], U2)  :-  permute (NIL, U3), insert (3, U3, U2)
                                       permute (NIL, NIL), insert (3, NIL, U2)
                                                           insert (3, NIL, [3])
         permute ([3], [3])
                                       insert (2, [3], U1)
                                       insert (2, [3], [2, 3])
       permute ([2, 3], [2, 3] )
                                       insert (1,[2, 3], Q)
                                       insert (1, [2, 3], [1, 2, 3])
     permute ([1, 2, 3], [1, 2, 3])
       Q = [1, 2, 3 ]  ◄——  first answer!
```

Now if you ask it for another answer it backs up.

P.42

```
                                  redo  insert( 1, [ 2, 3 ], Q )
                                        insert( 1, [ 2, 3 ], [ 2, 1, 3 ] )

permute( [ 1, 2, 3 ], [ 2, 1, 3 ] )
Q = [ 2, 1, 3 ]
                                  redo  insert( 1, [ 2, 3 ], Q )
                                        insert( 1, [ 2, 3 ], [ 2, 3, 1 ] )

permute( [ 1, 2, 3 ], [ 2, 3, 1 ] )
Q = [ 2, 3, 1 ]
```

Now redoing the last insert fails.

```
                        redo   permute( [ 2, 3 ], U1 )
                               permute( [ 2, 3 ], [ 3, 2 ] )

                                     insert( 1, [ 3, 2 ], Q )
                                     insert( 1, [ 3, 2 ], [ 1, 3, 2 ] )

permute( [ 1, 2, 3 ], [ 1, 3, 2 ] )
Q = [ 1, 3, 2 ]
 ⋮
Q = [ 3, 1, 2 ]
 ⋮
Q = [ 3, 2, 1 ]
```

Using Permute to Sort (Inefficiently)

P.43

```
sorted( [     ] ) .
sorted( [ X ] ) .
sorted( [ A, B | R ] )  :-   A = < B ,
                             sorted( [ B | R ] ) .
sort2( L, S )  :-   permute( L, S ),  sorted( S ) .


sort2( [ 2, 3, 1 ],  Ans )
   permute( [ 2, 3, 1 ],  Ans )
   permute( [ 2, 3, 1 ],  [ 2, 3, 1 ] )     sorted( [ 2, 3, 1 ] )
                                             ⋮
                                             fail
   permute( [ 2, 3, 1 ], [ 3, 2, 1 ] )      sorted( [ 3, 1, 2 ] )
                                             ⋮
                                             fail
   permute( [ 2, 3, 1 ], [ 3, 1, 2 ] )      sorted( [ 3, 1, 2 ] )
                                             ⋮
                                             fail
   permute( [ 2, 3, 1 ], [ 1, 2, 3 ] )      sorted( [ 2, 1, 3 ] )
                                             fail
   permute( [ 2, 3, 1 ], [ 1, 2, 3 ] )      sorted( [ 1, 2, 3 ] )
Ans  =  [ 1, 2, 3 ]
```