```
Question 1
----------
I ran the occurs check for the set of files okc*.txt. The time finding for each such case is as follows

okc1.txt
----------
time ./CS561A2.exe okcheck-samples/okc1.txt /CS561A2.exe okcheck-samples
Failure
LHS: <(10, y)
RHS: <(x, +(b, y))
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.005s
user    0m0.000s
sys     0m0.000s

time ./CS561A2.exe okcheck-samples/okc1.txt -noOccurCheck
LHS: <(10, y)
RHS: <(x, +(b, y))
Substitution: {x/10, y/+(b, y)}

real    0m0.004s
user    0m0.000s
sys     0m0.000s


okc2.txt
---------
time ./CS561A2.exe okcheck-samples/okc2.txt
Failure
LHS: [a, x]
RHS: x
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.004s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc2.txt -noOccurCheck
LHS: [a, x]
RHS: x
Substitution: {x/L(a, x)}

real    0m0.004s
user    0m0.000s
sys     0m0.000s

okc3.txt
---------
time ./CS561A2.exe okcheck-samples/okc3.txt -noOccurCheck
LHS: +(x, a)
RHS: +(<(y, [x, y, a, 10, b]))
Substitution: {x/<(y, [x, y, a, 10, b])}

real    0m0.004s
user    0m0.000s
sys     0m0.000s

time ./CS561A2.exe okcheck-samples/okc3.txt

Failure
LHS: +(x, a)
RHS: +(<(y, [x, y, a, 10, b]))

Substitution: NO SUBSTITUTION POSSIBLE
real    0m0.004s
user    0m0.000s
sys     0m0.000s

okc4.txt
--------
time ./CS561A2.exe okcheck-samples/okc4.txt
```

```
Failure
LHS: favorite([+(1, b), [f, 6, h], i, 9, k])
RHS: favorite([+(a, max(3, e, b))], avg([5, 6, 7], 8, j, 0), [A, B, C])
Substitution: NO SUBSTITUTION POSSIBLE
real    0m0.004s
user    0m0.000s
sys     0m0.000s

time ./CS561A2.exe okcheck-samples/okc4.txt -noOccurCheck
LHS: favorite([+(1, b), [f, 6, h], i, 9, k])
RHS: favorite([+(a, max(3, e, b))], avg([5, 6, 7], 8, j, 0), [A, B, C])
Substitution: {a/1, b/max(3, e, b)}

real    0m0.004s
user    0m0.000s
sys     0m0.000s

okc5.txt
---------
 time ./CS561A2.exe okcheck-samples/okc5.txt -noOccurCheck
LHS: favorite([[[[[[6, x]]]]]])
RHS: favorite([x])
Substitution: {x/L2([[[[6, x]]]])}

real    0m0.009s
user    0m0.000s
sys     0m0.000s

time ./CS561A2.exe okcheck-samples/okc5.txt
Failure
LHS: favorite([[[[[[6, x]]]]]])
RHS: favorite([x])
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.004s
user    0m0.000s
sys     0m0.000s

okc6.txt
---------
 time ./CS561A2.exe okcheck-samples/okc6.txt
Failure
LHS: favorite([[[[[[6, x]]]]]])
RHS: favorite([x])
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.005s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc6.txt -noOccurCheck
LHS: favorite([[[[[[6, x]]]]]])
RHS: favorite([x])
Substitution: {x/L2([[[[6, x]]]])}

real    0m0.004s
user    0m0.000s
sys     0m0.004s

okc7.txt
---------
 time ./CS561A2.exe okcheck-samples/okc7.txt -noOccurCheck
LHS: [[[1], [2], [[+(x)]]]]
RHS: [[[z], [y], [x]]]
Substitution: {x/L6(+(x)), y/2, z/1}

real    0m0.004s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc7.txt
Failure
```

```
LHS: [[[1], [2], [[+(x)]]]]
RHS: [[[z], [y], [x]]]
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.004s
user    0m0.000s
sys     0m0.000s

okc8.txt
----------
 time ./CS561A2.exe okcheck-samples/okc8.txt
Failure
LHS: [[[1], [2], [[1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s,
t, u, v, w, x, y, z]]]]
RHS: [[[z], [y], [x]]]
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.004s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc8.txt -noOccurCheck
LHS: [[[1], [2], [[1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s,
t, u, v, w, x, y, z]]]]
RHS: [[[z], [y], [x]]]
Substitution: {x/L6(1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r,
s, t, u, v, w, x, y, z), y/2, z/1}

real    0m0.004s
user    0m0.000s
sys     0m0.000s

okc9.txt
---------
 time ./CS561A2.exe okcheck-samples/okc9.txt -noOccurCheck
LHS: +(x, y)
RHS: x
Substitution: {x/+(x, y)}

real    0m0.003s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc9.txt
Failure
LHS: +(x, y)
RHS: x
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.003s
user    0m0.000s
sys     0m0.000s

okc10.txt
----------
 time ./CS561A2.exe okcheck-samples/okc10.txt
Failure
LHS: +(a, +(y, +(b, [c, x])))
RHS: x
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.004s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc10.txt -noOccurCheck
LHS: +(a, +(y, +(b, [c, x])))
RHS: x
Substitution: {x/+(a, +(y, +(b, [c, x])))}

real    0m0.004s
user    0m0.000s
```

```
sys     0m0.000s

okc11.txt
----------
 time ./CS561A2.exe okcheck-samples/okc11.txt -noOccurCheck
LHS: +(a, -(1, 2))
RHS: +(-(a, b), y)
Substitution: {a/-(a, b), y/-(1, 2)}

real    0m0.004s
user    0m0.000s
sys     0m0.000s

 time ./CS561A2.exe okcheck-samples/okc11.txt
Failure
LHS: +(a, -(1, 2))
RHS: +(-(a, b), y)
Substitution: NO SUBSTITUTION POSSIBLE

real    0m0.004s
user    0m0.000s
sys     0m0.000s


Occurs in general they say is an expensive operation. This is because for every term we check in the
expression of the term
is contained in that expression. For every deeply nested expressions this can be expensive. although
slightly expensive
from my experiments i found out that on an average even with the occur check on the running time is
pretty fast.

With really complicated expression the lag due to occurs check comes into play.


Question 2(a)
----------
An algorithm for the unifier can work like this:

unifier(expression1 , expression2):
 expression_list1 = partition expression1 into sub expressions consisting of constants , variables ,
lists and compound statements
 expression_list2 = partition expression2 into sub expressions consisting of constants , variables ,
lists and compound statements

 if length(expression_list1) == length(expression_list2):
    discard every constant expression in expression_list2 and expression_list1 that occur at the same
index

    add every variable expression in expression_list2 and expression_list1 that occur at the same index
    to the variable map. add these variables to the equivalence map also. discard these varibles then
from the
    expression lists

    for every variable expression in expression_list2 or expression_list1 that maps to a constant
expression
    and occurs at the same index, add the variable , constant mapping to the value map
    for every value added , refer the variables that are equal to the variable in the equivalence map
and assign values for
    those variables also.
    discard all such variables and the constant from the expression list

    for every list expression in the expression lists that occurs at the same index, create new
expressions 1 and 2 consisting
    of children in the list , discard these two list and then unify these 2 new expression using the
unify method

    for every compound expression in the expression lists that occurs at the same index, create new
expressions 1 and 2 consisting
    of children in the compound , discard these two compound expressions and then unify these 2 new
expression
    using the unify method
```

a unificaton algorithm like this keeps descreasing the elements to be evaluated and always terminates.

Question 2(b)

Why, in general, is a more general unifier preferred over a less general one?

A more general unifies is always preferred as every other unification algorithm can be obtained by altering the
most general unification algorithm. This is because for every pair of unifiable expressions there is always a single most
general unifier.

A MGU always returns if not all possibilites a proper set of mappings for each variable

Question 3

For testing the code , i built the code step by step. Starting with the scanner code to make sure tokenization works properly.
once the tokeization worked i created classes for lists, variablesm, constant. I tested them. Then i added support for
compound statements. I then created a tree chaines with all these children and checked its consistency. I then implemented
the unify and unify var algorithm. I checked with many test cases starting with the most simple one to many complex types.
As and when if something failed i ran it again and tested all cases till i got the expected output.