CSE340 Spring 2020 HOMEWORK 3
Due by 11:59 PM on Wednesday March 18 2020

PLEASE READ THE FOLLOWING CAREFULLY

1. Your answers can be only be typed.
2. On Gradescope, you should submit the answers to separate question separately.
3. For each question, read carefully the required format for the answer. The required format will make it easier for you to answer and for the graders to grade. Answers that are not according to the required format will not be graded

Problem 1 (Reducible Expression). The goal of this problem is to identify the redexes, if any, in the given expressions. To identify a redex, you should highlight the (λx. t) part of the redex in yellow and the t' part of the redex in blue. Here are two examples.

Example 1. (λx. x x) (λx. x x) (λx. x x) (λx. x x)

Answer. (λx. x x) (λx. x x) (λx. x x) (λx. x x)

Example 2. ((λx. x x) (λx. x x)) ((λx. x x) (λx. x x))

Answer. ((λx. x x) (λx. x x)) ((λx. x x) (λx. x x))
((λx. x x) (λx. x x)) ((λx. x x) (λx. x x))

Note that you are only asked to identify the redexes and you are not asked to do the reduction.

For each of the following, identify the redexes, if any:

1. x (λx. x x) (λx. x x) x

   (((x (λx. x x)) (λx. x x)) x) – None

2. (λx. (λx. x) x) (λx. x) x

   ((λx. (λx. x) x) (λx. x)) x

   ((λx. (λx. x) x) (λx. x)) x

3. (λx. (λx. x) x) λx. ((λx. x) x)

   (λx. (λx. x) x) (λx. ((λx. x) x))

   (λx. (λx. x) x) (λx. ((λx. x) x))

   (λx. (λx. x) x) (λx. ((λx. x) x))

4. (λx. x (λx. x) x) λx. x (λx. x) x

   (λx. (x (λx. x) x)) λx. (x (λx. x) x)

5. x (λx. x (λx. x) x) (λx. x (λx. x) x) x

   (((x (λx. x (λx. x) x)) (λx. x (λx. x) x)) x) - None

6. λx. λx. x λx. λx. x x x

(λx. (λx. x (λx. (λx. x (x x)))))) – None

7. ((λx. λy. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x)

((λx. λy. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x)

((λx. λy. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x)

8. (λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

(λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

(λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

(λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

(λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

9. x (λx. (λy. x) λx. x x) (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

(((x (λx. (λy. x) λx. x x)) (λx. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x))

(((x (λx. (λy. x) λx. x x)) (λx. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x))

(((x (λx. (λy. x) λx. x x)) (λx. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x))

10. (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

(λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

(λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

(λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

Problem 2 (Normal Order). The goal of this problem is to identify the unique redex, if any, that will be reduced first in normal order evaluation strategy for the given expressions. To identify a redex, you should highlight the (λx. t) part of the redex in yellow and the t' part of the redex in blue. Here is an example.

Example. (λx. x x) (λx. x x) (λx. x x) (λx. x x)

Answer. (λx. x x) (λx. x x) (λx. x x) (λx. x x)

Note that you are only asked to identify the redexes and you are not asked to do the reduction.

1. x (λx. x x) (λx. x x) x

   ((x (λx. x x)) (λx. x x)) x) – None
2. (λx. (λx. x) x) (λx. x) x

   (((λx. (λx. x) x) (λx. x)) x)
3. (λx. (λx. x) x) λx. (λx. x) x

   (λx. (λx. x) x) (λx. (λx. x) x)
4. (λx. x (λx. x) x) λx. x (λx. x) x

   (((λx. x (λx. x) x) λx. x (λx. x)) x)
5. x (λx. x (λx. x) x) (λx. x (λx. x) x) x

   (((x (λx. x (λx. x) x)) (λx. x (λx. x) x)) x) - None
6. λx. (λx. x) (λx. (λx. x) x x)

   ((λx. (λx. x)) (λx. (λx. x) x x))
7. ((λx. λy. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x)

   ((λx. (λy. (λy. x) λx. x x))) ((λx. (λy. x)) λx. x x)
8. (λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

   (λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))
9. x (λx. (λy. x) λx. x x) (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

   (((x (λx. (λy. x) λx. x x)) (λx. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x))
10. (λ1x. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

    (λ1x. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

Problem 3 (Call by Value). The goal of this problem is to identify the redexes, if any, that can be reduced first in call by value evaluation strategy for the given expressions. To identify a redex, you should highlight the (λx. t) part of the redex in yellow and the t' part of the redex in blue. Here is an example:

Example. (λx. x x) (λx. x x) (λx. x x) (λx. x x)

Answer. (λx. x x) (λx. x x) (λx. x x) (λx. x x)

Note that you are only asked to identify the redexes and you are not asked to do the reduction.

You are only asked to identify the redex and you are not asked to do the reduction.

1. (λx. x x) λx. x x

   (λx. x x) (λx. x x)
2. (λx. (λx. x) x) x

   ((λx. (λx. x) x) x)
3. ((λx. (λx. x) x) λx. (λx. x) x) ((λx. x) x)

   ((λx. (λx. x) x) (λx. (λx. x) x) ((λx. x) x)
4. ((λx. (λx. x) x) x (λx. x) x) ((λx. x) x)

   (((λx. (λx. x) x) x) (λx. x) x)) ((λx. x) x))

   (((λx. (λx. x) x) x) (λx. x) x)) ((λx. x) x))
5. x (λx. x (λx. x) x) (λx. x (λx. x) x) x

   ((x (λx. (x (λx. x) x))) (λx. (x (λx. x) x)) x) - None
6. λx. λx. x λx. ((λx. x) x) x

   (λx. (λx. x (λx. ((λx. x) x) x))) - None
7. ((λx. λy. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x)

   ((λx. (λy. ((λy. x) λx. x x)))) ((λx. (λy. x)) λx. x x)
8. (λx. (λx. (λx. x) x) x) ((λx. x) (λx. x) (λx. x))

   ((λx. (λx. (λx. x) x) x) (((λx. x) (λx. x)) (λx. x)))
9. x (λx. (λy. x) λx. x x) (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

   (((x (λx. (λy. x) λx. x x)) (λx. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x))
10. (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

    ((λx. (λy. x) λx. x x)) ((λx. (λy. x)) λx. x x))

Question 4 (Beta reduction and alpha renaming). For each of the following expressions, give the resulting redex after the highlighted redex is reduced. **If** there is need for renaming, you should break you answer into two parts. In the first part you do the renaming and in the second part you do the beta reduction. In all cases, you should highlight by using **boldface** the whole part of the result that corresponds to the redex after it is reduced. Also, if alpha renaming is needed, you should highlight the whole redex with the par that is renamed.

Example 1. ((λx. x x) y) ((λx. x x) x)
Answer. **(λx. y y)** ((λx. x x) x) after beta reduction

Example 2. ((λx. λy. x x) y) ((λx. x x) x)
Answer. **((λx. λw. x x) y)** ((λx. x x) x) after alpha renaming
(λw. y y) ((λx. x x) x) after beta reduction

Note that you are asked to do only one beta reduction of the specified redex and no multiple beta reductions.

For each of the following expressions, give the resulting expression after the highlighted redex is reduced. You should only do renaming if renaming is needed. You should follow the format and requirements outlined above in your answers.

1. (λx. x x) (λx. x)

   (λx. x) (λx. x) beta reduction

2. (λx. x x) (λx. x y)

   (λx. x y) (λx. x y) beta reduction

3. (λx. x (λx. x) (λy. x) x) (λx. x x)

   (λx. x x) (λx. x) (λy. **(λx. x x)**) **(λx. x x)** beta reduction

4. (λx. x (λx. x) (λy. x) x) (λx. x y)

   (λx. x (λx. x) **(λl. x)** x) (λx. x y) alpha remaining

   (**(λx. x y)** (λx. x) (λl. **(λx. x y)**) **(λx. x y)**)) beta reduction

5. (λz. x (λx. x) (λy. x) x) (λx. y z) - none

6. (λx. x (λx. x) (λx. λy. x) x) (λx. y z)

   (λx. x (λx. x) **(λx. λw. x**) x) (λx. y z) alpha remaining

   (**(λx. y z)** ((λx. x) ((λx. λw. x) **(λx. y z)**))) Beta reduction

7. λy. (λx. x (λx. x) (λx. λy. x) x) (λx. y z) - None

8. (λx. (λy. λz. x) x) (λx. (λy. y) z)

   (λx. **(λp. λz. x**) x) (λx. (λy. y) z) alpha remaining

   ((λp. λz. x) **(λx. (λy. y) z)**) beta reduction

9. (λx. x (λx. x) x (λx. x) x) y z

(**y** (λx. x) **y** (λx. x) **y**) y z beta reduction

10. (λx. (λy. (λz. λx. x) x) (λx. x y z)

(λx. **(λk. (λz. λx. x) x)** (λx. x y z)) alpha remaining

(λk. (λz. λx. x) (λx. x y z)) beta reduction

Question 3 (Binary Search Trees (BST) with Lambda Calculus). We define a lambda calculus representation of binary search trees. In general, a tree can be either empty or it is not empty. If the tree is not empty, then the node contains an element and two subtrees: the left subtree and the right subtree. As I discussed in class with lists, we need to have a way to determine if a tree is empty, so that we do not attempt to access the fields of an empty tree. This is similar to C++, in which you do not access the fields of a node before checking that the pointer to the node is not the null pointer.

At a high level, a BST is either empty or it is not empty, in which case it is of the form

a LT RT

where a is the element and LT and RT are the left and the right subtrees respectively. In general, either LT alone, or RT alone or LT and RT both can be empty or both LT and RT can be not empty.

To represent this data structure in Lambda calculus, we will use pairs (the old homework on lists should be very educational and I suggest you read it with its solution).

empty tree	= pair fls fls
tree with one element and two empty subtrees	= pair tru (pair a (pair (pair fls fls) (pair fls fls))

Let is consider the pieces

tru	indicates that the tree is not empty
a	is the element. If a tree T is not empty, the fst (snd T) is the root element of the tree
(pair fls fls)	is the left subtree. In this example, it is empty. In general, if a tree is not empty, then fst (snd (snd T) is the left subtree.
(pair fls fls)	is the right subtree. In this example, it is empty. In general, if a tree is not empty, then snd (snd (snd T) is the right subtree.

Of course, having such cumbersome expression can result if hard to read code! So, the best thing is to introduce functions that hide the complexity. Here are two that I propose and you can add more in your solution

LT = λT. fst (snd (snd T)
RT = λT. snd (snd (snd T)

You should be c+areful that these two functions expect that the tree is not empty.

For all questions, you are asked to write functions. You should understand that to mean write a lambda expression. Finally, in all the question, there should be no attempt to balance the BST.

1. (**Empty Tree**) Write a function `isempty` that returns a boolean value to indicate if a BST is empty or not. The function should return tru if the tree is empty and fls if the tree is not empty.
2. (**Root Element**) Write a function `root` that returns the root of a BST. Since a tree can be empty, we need a way to indicate that. Your function should return a pair. If the first element of the pair is tru, then the second element of the pair is the root of the tree. If the first element of the pair is fls, this means that the tree is empty and has no root.

3. (**Insert element**) Write a recursive function that inserts an element into a tree. This function always succeeds if its argument is a BST.
4. (**Sum of elements**) Write a recursive functions that takes as argument a tree whose elements are integers and returns the sum of the elements of the tree. If the tree is empty, the function should return 0
5. (**Merging two tree**) Write a recursive function that takes two trees T1 and T2 and returns a tree T obtained by appending inserting all elements of T2 into T1.
6. **(Tree Height**) Write a recursive function to calculate the height of a tree. If a tree is empty, its height is 0. If a tree has only one element, its height is also 0.
7. **(Deepest Element)** This is probably the hardest question. Write a recursive function that takes a tree as argument and return an element that is the farthest away from the root (its distance to the root is equal to the tree height). Since the tree can be empty, the function should return a pair consisting of a Boolean value and a tree element (see discussion with question 2 above). In general, there might be more than one element that are deepest elements. Your function should only return one such element.

```
1. isempty =  λT.(fst T) fls tru
2. root = λT. (isempty T)
            (pair fls fls)
            (pair tru(fst(snd T)))
3. g  = λInsertE. λT.λa. (isempty T)
           ( (pair tru (pair a (pair fls fls ) (pair fls fls))))
           ((gteq a (root T)) (InsertE (RT T) a) (InsertE (LT T) a))
   insert = fix g
4. g = λadd. λT.(isempty T)
           (0)
           (plus (root T) (plus (add (LT T))(add (RT T))))
    sum = fix g
5. g = λmergeT. λT1. λT2.(isempty T2)
           (T1)
           ((insert T1 (root T2))(merge((mergeT T1 (LT T2)))((mergeT T1 (RT T2)))))
    merge = fix g
6.  max = λa. λb(gteq a b)
           (a)
           (b)
     g =  λH.λT.(and (isempty (LT T)) (isempty (RT T)))
           (0)
           (suc max(H((LT T)) H((RT T))))
    height = fix g
7.  g = λdeep.λT.(equal 0 (height T))
           (isEmpty T)
                   (pair fls fls)
                   ((pair tru(pair (root T)(pair fls fls)(pair fls fls))))
           (deep (LT T))(deep(RT T))
    deepest =  fix g
```