Chakeera Wansoh
6080566

MUIC: Functional and Parallel Programming Exam

Date: Thursday, October 22nd, 2020
Instructor: Rachata Ausavarungnirun

Problem 1 (30 Points):	
Problem 2 (15 Points):	
Problem 3 (35 Points):	
Problem 4 (20 Points):	
Total (100 points):	

Instructions:

1. This exam is designed to take 24 hours, but you have 5 days on it.
2. Submit your work as a zip file on Canvas.
3. For the coding part, I expect everyone to **thoroughly test your code**. Hence, you must **submit the test cases for all the coding questions** along with the code and explain what each test case is used for. **Test cases, along with the explanation of how you test your code, will factor into 10% of all the questions.**
4. If not specified, input and output types are a part of the question. Please use appropriate input and output types that make sense for the purpose of the question.
5. Please clearly comment your code, especially if your code do not work perfectly,
6. Clearly indicate your final answer for each conceptual problem.
7. Our typical restriction for Scala packages are similar to Assignments 1 and 2.
8. **DO NOT CHEAT.** If we catch you cheating in any shape or form, you will be penalized based on **our plagiarism policy**.

Tips:

- **Read everything.** Read all the questions on all pages first and formulate a plan.
- **Be cognizant of time.** The submission site will close at 00.01AM on Saturday.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.

1. Potpourri [30 points]

For this questions, please put your answers here and submit the pdf.

(a) **Compound Types** [5 points]

What is the key difference between a sum type and a product type?

-In the product type, it is a predefined type. We will know exactly what the type of the value is and it can have multiple values.
-Product type is cartesian product
for eg: Tuple.
-In the sum type, the value could be of any type but there is only one single value but can have many types.
-Sum type is disjoint union
Eg: Option

(b) **Class vs. Objects** [5 points]

What is the key difference between a class and an object?

Class: Template for creating objects in program.
Object: The object is an instance of a class.
Difference:
1) A class does not allocate memory space when it is created while object allocates memory space whenever they are created.
2) Class is a logical entity while object is physical entity.
3) Class is declared once while object can be created more than one.
4) Classes can't be manipulated as they are not available in memory while objects can.
5) Class doesn't have any values which are associated with the fields while in object each and every object has its own values, which are associated with the fields.

(c) **Currying** [6 points]

What is the benefit of currying? Please list one example (do not copy the one from our lectures).

-You can stage the function that is break them down to sequences.
-Parts of the execution can run as soon as the values are ready
-Maps well with data flow model which can allow the compiler and the hardware to be faster
-Eliminate data dependency as soon as possible
for eg: currying subtraction

```
def sub_curry(a: Int) (b: Int) = a - b;

  def main(args: Array[String])
  {
     println(sub_curry(15)(5));
  }
```

(d) **CBV vs. CBN** [8 points]

Consider the following code.

```
def leftCBV(x:Int, y:Int) = x
def leftCBN(x:=> Int, y: =>Int) = x
def loop:int = loop

leftCBV(1+1,loop)        // Call 1
leftCBV(loop, 1+1)       // Call 2
leftCBN(1+1,loop)        // Call 3
leftCBN(loop, 1+1)       // Call 4
```

What happen to each of the four calls at the end? Please explain why each of them behave the way you observed using what we learned from the class.

Behavior of CBN and CBV:
CBV or Call By Value is when every function's argument is evaluated once.
CBN or Call By Name is when the function's argument is evaluated only when it is used in the function else it wont be evaluate.

Call 1: CBV evaluates first hence it will not terminate as y is loop.
Call 2: CBV evaluates first hence it will not terminate as x is loop.
Call 3: CBN will only evaluate when the argument is used hence here it return 2 as it calls x argument.
Call 4: Similarly to Call 3, It wont evaluate until used hence here our x argument is a loop so it wont terminates.

(e) **Types** [6 points]

What is the difference between upper/lower bounds and variance?

An upper type bound X <: B declares that type variable X refers to a subtype of type B ie, X extends B
An lower type bound X >: B declares that type parameter X or the abstract type X refer to a super type of type B. Lower type bounds declare a type to be a super type of another type. ie, B extends A

Variance: The ability of type parameters to vary on high-kinded types. It is in a generic type. It explains inheritance correlation of Types that have parameters or arguments within them. These types belongs to the generic classes, which takes a type like a parameter.

2. More Expression Support [15 points]

In this question, we will extend our good old friend Expression from the in-class exercise to make sure they can support more types of expressions.

(a) Extend our expression to support `Sub`, which should behave in a way a subtract works for expression. You also must overload the subtract sign (−) so that they works on `Expr`.

(b) Extend our expression to support `Div`, which should behave in a way a division works for expression. You also must overload the divide sign (/) so that they works on `Expr`. Also, if the denominator is zero, the expression should generate a divide by zero exception.

(c) Extend our expression to support logical negation `LogNeg`, which should behave in a way a logical negate works (flip its binary values). You also must overload the negate sign (~) so that they works on `Expr`.

Please save the file with the name `betterExpr.scala`

3. Binary Search Tree [35 points]

In this question, you are going to implement a binary search tree in Scala.

Below is the definition of a binary search tree:

A binary tree is a tree that consist of up to two childs: left and right subtrees. The left and right subtrees either be a binary tree or a leaf node (which contains no child).

A binary search tree (BST) is a binary tree where every single element on the left subtree is lower or equal to the value of the current node, and the values of every single element on the right subtree is higher than the current node.

An in-order traversal is a method to traverse your tree in a way that it first visits all the left subtree first, follow by the middle (current node), then the right subtree. Performing an in-order traversal on a binary search tree will result in a list that is sorted from lowest to highest.

(a) First, create a class called `BST` that contains all the necessary components you need including any constructors that you want. You must include at least *one* constructor that initialize your tree and the following functions (please feels free to add any helper functions as needed):

(b) Implement an `insertCPS` function that **takes an input value and an input tree**, and insert this value as a new node to your tree while ensuring you preserve the BST's property using continuation passing style. **This function should return a resulting tree after you insert the node.**

(c) Implement an `walkCPS` function that performs an in-order traversal **on the input tree** and **return a list of values** (which should now be sorted from lowest to highest) using continuation passing style.

(d) Implement an `heightCPS` function that **takes an input tree and return the height of the tree**. We define height as the maximum depth of all the branches in your tree.

(e) Implement a function `checkBST` that takes an input tree, and return a boolean value true of the input is a binary search tree and false if the input tree is not a binary search tree. **Note that you do not need to use CPS here!**

(f) Now, you might have realized I have not specify the types of the value stored in your tree. In this part, make sure that your list works for any data types that have total ordering. If you do not want to attempt this question, then, make sure your other questions works with Integer.

(g) Once everything is all done, please write your own test cases that thoroughly test your code **so that it also works in corner cases**. Your test should shows, as best as possible, that your implementation works as intended.

Please save the file with the name `bst.scala`

4. Scope [20 points]

In this question, we will use a pseudocode below that does not exactly compile in Scala to test your understanding of the lexical scope and dynamic scope. Consider the following pseudocode. In this question, let us assume emphval is allowed to remap the variable name (i.e., x, y, z) to the newly defined expression and the value used inside the expression follow the lexical or dynamic scope rules from our lecture.

If the expression does not evaluate to a value, please state why. If the expression break the type rule, please explain why.

Hint: Your answers might be a function (functions are values!).

```
val x = 10                              // Line 1
val y = 20                              // Line 2
def f1(x:Int) = x + y                   // Line 3
def f2(y:Int) = (z:Int) => x - y - z    // Line 4
val x = f1(x) + x                       // Line 5
val x = f2(x) + x                       // Line 6
val y = f1(y) + (f2(y)) (10)            // Line 7, note that 10 is the
                                        //         input to f2(y)
val y = f1(y)                           // Line 8
val z = x(y)                            // Line 9
```

(a) What is the difference between Lexical and Dynamic scope? (5 points)

Lexical scope: The body of a function is evaluated in the environment where the function is defined, not when it is called. When we declare a function using lexical scope, what ever is use in the function will refer to the environment before it. (Like a Global variable - from my understanding :D)

Dynamic scope: It is the inverse of lexical scope. Here Environment is used when the function is called instead of when it is defined.

(b) Assuming we use **Lexical Scope**, what is the value of x at line number 5? Briefly explain why. (2 points)

40

because x is declared to be equal to 10 and y equal to 20 in Line 1 and 2 putting x in the function f1(x) = f1(10) = x+y = 10+20 = 30 Hence f1(x) + x = 30 + 10 = 40.

(c) Assuming we use **Lexical Scope**, what is the value of x at line number 6? Briefly explain why. (2 points)

10-z

because f2 function takes in 2 arguments but here it was given only one argument hence we will have to return z function. In Line 6, the value x is put in the f2 function which is declared as y so its x-y-z = x-40-z = 10 - 40 - z (took the value of x from Line 1) = -30-z
=> x = -30-z+40 = 10-z

not possible to evaluate

(d) Assuming we use **Lexical Scope**, what is the value of y at line number 7? Briefly explain why. (2 points)

20

because for the f1(y) = f1(20) we got y value from Line 2 = 20+20 = 40
for the f2(y)(10) we got the two arguments with the y value as 20 and z value as 10
= x - y - z = x - 20 -10 = x value from Line 1 so 10 - 20 -10 = -20
=> 40 - 20 = 20

(e) Assuming we use **Lexical Scope**, what is the value of y at line number 8? Briefly explain why. (2 points)

40

because now the value of y got updated to 20 = f1(20) = 20 + 20 (y value from Line 2) = 40

(f) Assuming we use **Lexical Scope**, what is the value of z at line number 9? Briefly explain why. (2 points)

-30
because x(y) is a function of x so we will go to Line 6 which is f2(x) +x the value of x here will be the one from Line 5 which is 40 so we get and x value is from Line 1 = 10- 40 -z + x = 10-40-40+40 = -30

(g) Assuming we use ***Dynamic Scope***, what is the value of z at line number 9? Briefly explain why. (5 points)

-40
because the x, y ,z value will be updated when they are used like in Line 5, 6 the value of x got updated twice from 10 to 40 . The value of y also changes to 40 at Line 7 and so on as seen in the picture.

I have attached the steps I have done for this questions below

* Dynamic Scope

Line 1	x = 10
Line 2	y = 20
Line 3	f1(x) = x + y
Line 4	f2(y)(z) => x - y - z
Line 5	x = f1(x) + x => (10+20) + 10 => 40
Line 6	x = f2(x) + x => (40 - 40 - z) + 40 => 40 - z
Line 7	y = f1(y) + f2(y)(10) => (20+20) + ((40-z) - 20 - 10) => 40 + (40 - 30 - 20 - 10) => 40 //
Line 8	y = f1(y) => 40 + 40 => 80
Line 9	z = x(y) => 40 - 80 => -40 //

* Lexical Scope

# 1	x = 10
# 2	y = 20
# 3	f1(x) = x + y
# 4	f2(y)(z) = x - y - z
# 5	x = 30 + 10 = 40
# 6	x = f2(40) + 40 => 10 - 40 - z + 40 => -30 - z + 40 => 10 - z
# 7	y = f1(20) + f2(20)(10) => (20+20) + (10 - 20 - 10) => 20
# 8	y = f1(20) => 20 + 20 = 40
# 9	z = x(40) : f2(40) + 40 => 10 - 40 - 40 + 40 => -30 //