Assignment 1

Advance Programming

Name: Kazi Sohrab Uddin Titu, Group Number: 58

Introduction:

This assignment is implementing several functions and try-catch in the interpreter in Haskell. We will see implementation of core functionality like using lambda function, apply expressions, try-catch, error handling and testing. The solution seems almost correct for most of the cases, even in recursive functions. I have tried to ensure coverage of different cases. Though there are some edge cases that could be refined further but I tried to cover them up. To run test cases, run cabal test in the terminal.

Task 1: Lambda Expressions

Functional: Yes.
Failures: I haven't got any failures for basic cases.
Notes: The implementation of Lambda expressions correctly evaluates in test cases. I have applied test case for Lambda base case and another one is Lambda on Apply expression.

Task 2: Apply Expressions

Functional: Yes.
Failures: I was facing issue with parenthesis earlier while implemented a test Lambda within Apply expression. But I have resolved it by removing extra parenthesis on printApply.
Notes: The solution can evaluate valid expressions, but an invalid function like (Apply (CstInt 5) (CstInt 2)) correctly throws an error.

Task 3: Try Catch Expressions

Functional: Yes.
Failures: I haven't got any failures for basic cases.
Notes: I have tried for both exp1 and exp2 and it succeeds on showing second one if first one fails, show first one if it evaluated successfully. I have also mentioned the error message if both exp1 and exp2 fails.

Question 1: What is your observable evaluation order for Apply, what difference does it make, and which of your test(s) demonstrate this?

My observable evaluation order for Apply is "arguments are evaluated first then the function is applied". The difference actually affects performance and error possibility. This is demonstrated by the test

```
testCase "Lambda & Apply" $
  eval envEmpty (Apply
                 (Let "x" (CstInt 2)
                   (Lambda "y" (Add (Var "x") (Var "y"))))
                   (CstInt 3))
  @?= Right (ValInt 5)
```

This test's showing that the lambda expression and the argument CstInt 3 are evaluated before applying the function.

Question 2: Would it be a correct implementation of eval for TryCatch to first call eval on both subexpressions? Why or why not? If not, under which assumptions might it be a correct implementation?

No, it would not be because TryCatch's main aim is to return exp2 when exp1 fails if exp2 also fails it gives an error. If it does immediately, it would be irrelevant for any potential error in exp1. This would not provide the purpose of TryCatch.

lazy evaluation might be a correct implementation of evaluating first call on both subexpressions. Lazy evaluation means they will be evaluated only if expressions' result is required immediately. But in the current evaluation model, we want exp2 to be evaluated only if exp1 fails.

Question 3: Is it possible for your implementation of eval to go into an infinite loop for some inputs?

Yes possible. Y combinator of handout's code or recursive expression might be the reason for an infinite loop. Without proper base case or exit condition, eval can evaluate recursively.