Assignment 10: Monad Calculator

New Attempt

Due Apr 23 by 11:59pm **Points** 100 **Submitting** a file upload

For this assignment, you are going to modify your assignment 9 in a couple ways. The first will be calculator components that can handle error without causing an exception in an old style, and another with new style (Monad). For this assignment, you must develop two modules. Function names and types in each module are provided and you must make sure that your implementation have the correct types. You are not allowed to change the type of each given function name. You can implement as many helper functions as you like.

Module `BasicAlgExp`

For this module, use the starter file **`BasicAlgExp.hs`** **(https://canvas.pitt.edu/courses/201557/files/12763225?wrap=1)** **(https://canvas.pitt.edu/courses/201557/files/12763225/download?download_frd=1)** . There are the total of five export functions as shown below:

- `checkChars :: [Char] -> [Char]`: This function checks whether an expression in a form of a string contains an invalid character. If there is no invalid characters, it returns the original expression. However, if it contain an invalid character, it returns an error message in the following form `"Error [checkChars]: The character 'g' is not allowed"`. This function also ensures that no operators other than `'+'`, `'-'`, `'*'`, and `'/'` are in the given expression. Support delimiters are the same as in previous assignment which are `'('`, `')'`, `'['`, `']'`, `'{'`, and `'}'`.
- `tokenizer :: [Char] -> [[Char]]`: This function turns an expression in a form of a string into a list of tokens where each token is a string of either a number, an operator, or a delimiter. Note that if the input is an error message in a form of a string, it simply forwards the error message as its output in the form of a list that contains one string. For example `tokenizer "Error [checkChars]..."` should be evaluated to `["Error [checkChars]..."]`
  Otherwise, just tokenize it as usual. For example, `tokenizer "(1 + 23) * 456"` should be evaluated to `["(", "1", "+", "23", ")", "*", "456"]`.
- `checkBalanced :: [[Char]] -> [[Char]]`: This function checks whether a given list of tokens is balanced. If the expression is balance or the input is a list of single error messtion, it simply returns the original input. However, if the expression is unbalance, it will return the output as a list of

single error message in the form of `["Error [checkBalanced]: The expression is unbalanced"]`.

- `infix2Postfix :: [[Char]] -> [[Char]]`: This function turns an infix expression in the form of a list of tokens into a postfix expression in the form of a list of tokens. Note that if the input is a list of single error message, it will simply returns the original input.
- `evaluate :: [[Char]] -> [Char]`: This function evaluate a postfix expression in the form of a list of tokens into a string representation of an integer. Note that if the input is a list of single error message, it will simply returns the original input. If the expression is invalid, it will return the list of error message states whether there are too many operator(s) or too many operand(s). For example, `evaluate ["1", "+"]` should be evaluated to `"Error [evaluate]: Too many operator(s)"`.

The main calculator program is given in **`BasicCalculator.hs`** **(https://canvas.pitt.edu/courses/201557/files/12763230?wrap=1)** **(https://canvas.pitt.edu/courses/201557/files/12763230/download?download_frd=1)** . Note that you are not allowed to modify this program. It contains the function named calculate as shown below:

```
calculate :: [Char] -> [Char]
calculate = evaluate . infix2Postfix . checkBalanced . tokenizer . checkChars
```

The main program use the above calculate function which is a compositions among functions from `BasicAlgExp` module. This is why you have to implement the above five functions with very strict given types.

Module `EitherAlgExp`

For this module, use the starter file **`EitherAlgExp.hs`** **(https://canvas.pitt.edu/courses/201557/files/12763229?wrap=1)** **(https://canvas.pitt.edu/courses/201557/files/12763229/download?download_frd=1)** . There are the total of five export functions as shown below:

- `eitherCheckChars :: [Char] -> Either [Char] [Char]`: This function checks whether an expression in a form of a string contains an invalid character. If there is no invalid characters, it returns the original expression packed in `Right`. However, if it contain an invalid character, it returns an error message in the following form `Left"Error [checkChars]: The character 'g' is not allowed"`. This function also ensures that no operators other than `'+'`, `'-'`, `'*'`, and `'/'` are in the given expression. Support delimiters are the same as in previous assignment which are `'('`, `')'`, `'['`, `']'`, `'{'`, and `'}'`.
- `eitherTokenizer :: [Char] -> Either [Char] [[Char]]`: This function turns an expression in a form of a string into a list of tokens (packed in `Right`) where each token is a string of either a number, an operator, or a delimiter. Note that if the input is an error message in a form of a string, it simply packs the error message in `Left` as its output. For example