# Assignment 9: Not So Simple Calculator

New Attempt

**Due** Apr 9 by 11:59pm **Points** 100 **Submitting** a file upload

For this assignment, you are going to write a calculator program that allows a user to enter a long algebraic expression. For simplicity, an expression can consist of

- **operands**: multiple digit positive integers (for simplicity)
- **operators**: `+`, `-`, `*`, and `/`
- **delimiters**: `()`, `[]`, and `{}`
- **whitespaces**: spaces only

Here are examples of expressions that a user can enter:

- `1 + 2 * 3 + 4 - 5`
- `(1 + 2) * [3 + 4] - 5`
- `1+2*(3+4)-5`

As we already know, `*` and `/` have higher precedence that `+` and `-`. However, a user can use multiple of `()`, `[]`, or `{}` to overwrite some operator precedence.

## Evaluating an Algebraic Expression

Slides about evaluating an algebraic expression can be found **here (https://canvas.pitt.edu/courses/201557/files/11636340?wrap=1)** **(https://canvas.pitt.edu/courses/201557/files/11636340/download?download_frd=1)** .

## The `AlgExp` Module

As usual, you are going to develop this program using modules. The `AlgExp` module (`AlgExp.hs`) is actually the only module in this assignment. Its main purpose is to evaluate a string representation of an algebraic expression into a number. We develop this as a module so that we can reuse it later (good practice). This module must export only the following functions:

- `isValid :: [Char] -> Bool`
  This function checks whether a given string contains only valid characters. In an algebraic expression (in our case), can only contain characters '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '-', '*', '/', '(', ')', '[', ']', '{', '}', and ' '

- `isBalanced :: [Char] -> Bool`
  This function checks whether a given string representation of an algebraic expression is balanced based on delimiters '(', ')', '[', ']', '{', and '}'. Note that if the given string is not valid (contains characters that are not allowed), isBalanced should return False.
- `infixToPostfix :: [Char] -> [Char]`
  Turn the infix form of an algebraic expression into the postfix form if the expression is valid and balanced. For example:
  - `infixToPostfix "12 + 5 * 3"` should return the string `"12 5 3 * +"`
  - `infixToPostfix "(12 + 5) * 3"` should return the string `"12 5 + 3 *"`
- `evaluate :: (Integral a, Read a) => [Char] -> a`
  If the given string is a valid and balanced algebraic expression, the evaluate function will return the result. For simplicity, you should use the `div` function for division. Note that the `Integral` type class is a type constraint because we use the `div` function. Similarly, we have to use the read function to turn a string representation of an integer into an integer.

To handle the error on this assignment, simply use the `error` function with an error message about the error.

## The main program

The main program (`calculator.hs`) uses the `AlgExp` module to create a simple calculator. It should ask the user to enter an algebraic expression, evaluate the result, and display the result on the console screen. Here is an example of a run:

```
$ ./calculator
Enter an algebraic expression
(12 * 3)/4+[5 * (6 - 8)]
(12 * 3)/4+[5 * (6 - 8)] = -1
Enter an algebraic expression
12 * 3 / 4 + 5 * 6 - 8
12 * 3 / 4 + 5 * 6 - 8 = 31
Enter an algebraic expression
46    - 12 * (4 - 2)
46    - 12 * (4 - 2) = 22
Enter an algebraic expression
q
```

If the user enter `q`, simply exit the program. Note that there is a chance that the user may enter an invalid algebraic expression. For this assignment, the `AlgExp` module use the `error` function to handle errors. Therefore, it will cause our calculator program to cause an exception. Here are examples of some possible errors:

```
$ ./calculator
Enter an algebraic expression
62 ^ 2
calculator: The expression contains an invalid symbol.
CallStack (from HasCallStack):
  error, called at ./AlgExp.hs:...

$ ./calculator
Enter an algebraic expression
(13 - 2)] * 6
calculator: The expression is not balanced.
CallStack (from HasCallStack):
  error, called at ./AlgExp.hs:...$ ./calculator

$ ./calculator
Enter an algebraic expression
12 +
calculator: Invalid Expression: Too many operator(s)
CallStack (from HasCallStack):
  error, called at ./AlgExp.hs:...

$ ./calculator
Enter an algebraic expression
23 34 + 5
calculator: Invalid Expression: Too many operand(s)
CallStack (from HasCallStack):
  error, called at ./AlgExp.hs:...
```

Again, simply let the error cause an exception with appropriate error message as shown above.

# Submission

Zip `AlgExp.hs` and `calculator.hs` into one single file named `assignment9.zip`. Make sure we can compile your program without error. Use undefined for incomplete functions.

**Some Rubric (2)**