Sabancı University
Faculty of Engineering and Natural Sciences

CS305 Programming Languages

Homework 4

Due: May 9, 2021 @ 11:55 PM

1 Introduction

In this assignment, you will implement a calculator using Scheme. For more detailed information about the homework read the rest of the document.

2 The Task

The assignment is divided into several parts. Each part is connected to another and by combining them, you will obtain a left-associative calculator in Scheme:

1. Write a procedure named "twoOperatorCalculator" which takes only one argument, which is an infix expression consisting of constant numbers, addition and subtraction operators only. The argument is given in the form of a list. The operators and the constant numbers are given as separate items of the argument list. The addition and the subtraction operators are left–associative. twoOperatorCalculator gets such an expression, which is guaranteed to be syntactically correct, and returns the value of the expression. Below is a sample output for this procedure:

```
1 ]=> (twoOperatorCalculator '(1))
Value: 1

1 ]=> (twoOperatorCalculator '(1 + 2))
Value: 3

1 ]=> (twoOperatorCalculator '(1 + -2))
Value: -1

1 ]=> (twoOperatorCalculator '(2 - 1 + 3 + 1))
Value: 5

1 ]=> (twoOperatorCalculator '(1 + 15 - 32/5 + -2))
```

```
Value: 38/5

1 ]=> (twoOperatorCalculator '(32/5))
Value: 32/5
```

Note that `32/5` in the last two examples is considered as a number, not the expression 32 divided by 5. However if we slightly change the syntax and give the last example as

```
1 ]=> (twoOperatorCalculator '(32 / 5))
```

in this case it is the expression which requires us to divide 32 by 5. For such an expression, the behaviour of twoOperatorCalculator is not defined, as it is designed only to handle addition and subtraction. You can consider that twoOperatorCalculator will never be applied to an expression in which there are division or multiplication operators.

Similarly, in `(1 + -2)`, the symbol `-` stands for the negative sign, not the subtraction operator. Whenever we use `-` as a subtraction operator, it is given as a separate item in the list. Consider the following two examples:

```
1 ]=> (twoOperatorCalculator '(1 - 2))
Value: -1

1 ]=> (twoOperatorCalculator '(1 -2))
```

The first example is perfectly correct. However, there is a syntax error in the second example. There are two constant numbers (1 and -2) but there is no operator. Hence the behaviour of twoOperatorCalculator is not defined in this case, it will never be applied to such an argument.

2. Write a procedure named "fourOperatorCalculator" which will calculate left-associative infix multiplication and division operations. Operations are given as a list and the output should be a list which contains infix addition, infix subtraction operations and the results of multiplication and division operations. Below is a sample output for this procedure:

```
1 ]=> (fourOperatorCalculator '(1))
Value: (1)

1 ]=> (fourOperatorCalculator '(2 * 3))
Value: (6)

1 ]=> (fourOperatorCalculator '(5 / 2))
Value: (5/2)
```

```
1 ]=> (fourOperatorCalculator '(5 / 2 * 2)))
Value: (5)

1 ]$=> (fourOperatorCalculator '(1 + 3 * 5 - 4 / 5 * 8 + 2 * -1))
Value: (1 + 15 - 32/5 + -2)
```

We also consider only valid, i.e. syntactically correct arguments for fourOperatorCalculator.

3. Write a procedure named "calculatorNested" which will calculate operations in nested lists. Some operations may have precedence over other operations and these operations are represented using nested lists. This procedure should calculate the operations inside nested lists and return a list which contains infix addition, subtraction, multiplication, division operations and the results of nested lists. Below is a sample output for this procedure:

```
1 ]$=> (calculatorNested '(1 + (1 + 1 * 2) * 5 - 4 / 5 * 8 + (5 - 3) * -1))
Value: 13: (1 + 3 * 5 - 4 / 5 * 8 + 2 * -1)

1 ]=> (calculatorNested  '(1 + 2 * ( 2 - (3 + 5) / 2)))
Value: 24: (1 + 2 * -2)
```

Note that there can be nested lists inside other nested lists. In this case calculatorNested should calculate all nested lists.

4. Write a procedure named "checkOperators" which will make an input check for the calculator. Allowed operations on the calculator are infix addition, infix subtraction, infix multiplication and infix division. These operations are given as elements of a list. The list can also contain nested lists which can have their own calculations. Below is a sample output for this procedure:

```
1 ]=> (checkOperators '(1))
Value: #t

1 ]=> (checkOperators '((((1))))
Value: #t

1 ]=> (checkOperators '(1 + (1 + 1 * -2) * 5 - -4 / 5 * 8 + (5 - 3) * -1))
Value: #t
```

```
1 ]=> (checkOperators '(+ 1 3))
Value: #f

1 ]=> (checkOperators '(1 < 3))
Value: #f

1 ]=> (checkOperators '1)
Value: #f

1 ]=> (checkOperators '())
Value: #f
```

5. Write a procedure named "calculator" which will calculate infix addition, subtraction, multiplication and division operations. Operators should be left-associative. Multiplication and division should have precedence over addition and subtraction. Expressions surrounded by parenthesis should have precedence. Expressions are given as a list. The procedure should first check whether the given list is syntactically correct. If the list is not syntactically correct, then the procedure should return false, otherwise it should calculate the operations and give the output as a result. Below is a sample output for this procedure:

```
1 ]=> (calculator '(1))
Value: 1

1 ]=> (calculator '(1 + (1 + 1 * 2) * 5 - 4 / 5 * 8 + (5 - 3) * -1))
Value: 38/5

1 ]=> (calculator '(1 -2))
Value: #f

1 ]=> (calculator '(1 < 3))
Value: #f

1 ]=> (calculator '1)
Value: #f

1 ]=> (calculator '())
Value: #f
```

3 How to Submit

Submit your Scheme file named as `username-hw4.scm` where `username` is your SUCourse username. We will test your submissions in the following manner. A set of test cases will be created to asses the correctness of your scheme calculator. Each test case will be automatically appended to your file. Then the following command will be executed to generate an output. Then your output will be compared against the desired output.

```
scheme < username-hw4.scm
```

So, make sure that the above command is enough to produce the desired output.

4 Notes

- **Important**: Name your files as you are told and **don't zip them.** [-10 points otherwise]
- **Important: Make sure your procedure name are exactly the same as it is supposed to be!**
- **Important: Since this homework is evaluated automatically make sure your output is exactly as it is supposed to be. (check sections 1 for details).**
- No homework will be accepted if it is not submitted using SUCourse+.
- You may get help from our TA or from your friends. However, **you must implement the homework by yourself**.
- Start working on the homework immediately.
- LATE SUBMISSION POLICY:
  Late submission is allowed subject to the following conditions:
  - Your homework grade will be decided by multiplying what you get from the test cases by a "submission time factor (STF)".
  - If you submit on time (i.e. before the deadline), your STF is 1. So, you don't lose anything.
  - If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
  - We will not accept any homework later than 500 mins after the deadline.
  - SUCourse+'s timestamp will be used for STF computation.
  - If you submit multiple times, the last submission time will be used.