Carleton University
Department of Systems and Computer Engineering
SYSC 3101 - Programming Languages - Winter 2023

Lab 5 - Modifying the Calculator Interpreter

References:

Two documents at the Racket website provide plenty of information about the Racket dialect of Scheme:

The Racket Guide, https://docs.racket-lang.org/guide/index.html

The Racket Reference, https://docs.racket-lang.org/reference/index.html

A guide to the DrRacket IDE can be found here:

http://docs.racket-lang.org/drracket/index.html

Racket Coding Conventions

Please adhere to the conventions described in the Lab 1 handout.

Getting Started

Download file lab5calc.rkt from Brightspace. This file contains the interpreter for a 4-function calculator language. (This interpreter is identical to the one in calc.rkt, which was presented in the lectures.)

Launch DrRacket and open lab5calc.rkt.

To run the calculator interpreter, click Run, then type `(calc)` in the Interactions area. This will call the procedure that executes the interpreter's *read-eval-print loop* (REPL). The REPL will display a prompt (`calc:`) and wait for you to type an expression in the input box.

The calculator supports four operations: `+`, `-`, `*` and `/`. Arithmetic expressions are typed using the same syntax as Racket; for example, to calculate 1 + 2 + 3, enter this expression:

```
(+ 1 2 3)
```

Read procedures `calc-eval` and `calc-apply`. Make sure you understand how `calc-eval` evaluates expressions that have nested expressions; for example `(+ 2 (* 3 4) 5 6)`. Make sure you understand how `calc-apply` handles expressions with 0 arguments, 1 argument and multiple arguments.

(8 Marks) Exercise 1

Modify `calc-apply` to provide an **`sqrt`** operator. This operator expects exactly one argument, and the expression `(sqrt x)` calculates the square root of `x`. Racket provides a procedure that calculates square root (check Section 4.3.2, *Generic Numerics*, in the *Racket Reference*.) The calculator interpreter should call this procedure.

When **`sqrt`** expressions with an incorrect number of arguments are entered, the calculator should call `error` to display this message: `"Calc: sqrt requires exactly 1 arg"`.

Test your modifications. Verify that the calculator correctly evaluates `sqrt` expressions in which the argument is a simple number; for example `(sqrt 4)`, as well as expressions in which the argument is a nested arithmetic expression; for example, `(sqrt (* 3 (- 5 8)))`.

(4 Marks)Exercise 2

Modify `calc-apply` to provide an ** operator. This operator expects exactly two arguments, and the expression `(** a b)` calculates `a` raised to the power of `b`. Racket provides a procedure that calculates powers (check Section 4.2.2, *Generic Numerics*, in the *Racket Reference*.) The calculator interpreter should call this procedure.

When ** expressions with an incorrect number of arguments are entered, the calculator should call `error` to display this message: `"Calc: ** requires exactly 2 args"`.

Test your modifications. Verify that the calculator correctly evaluates ** expressions in which the arguments are simple numbers as well as expressions in which one or both arguments are nested arithmetic expressions.

(8 Marks)Exercise 3

Racket provides a `min` procedure that accepts one or more integers or real numbers and returns the smallest of the numbers. For example,

`(min 1.0 3.0 2.0)` returns `1.0`

and

`(min 1 3 2)` returns `1`.

Modify `calc-apply` to provide a `min` operator. This operator should accept one or more numbers and should call Racket's `min` procedure to calculate the largest one.

When `min` expressions with an incorrect number of arguments are entered, the calculator should call `error` to display this message: `"Calc: min requires 1 or more args"`.

Hint: Suppose you enter this calculator language expression: `(min 1 3 2)`. When `calc-apply` is called, parameter `fn` is bound to `'min` and parameter `args` is bound to the list `'(1 3 2)`. Racket's `min` procedure expects one or more numeric arguments, not a list, so the calculator interpreter can't call Racket's `min` this way:

```
...(min args)...
```

How could you use **`foldr`** to apply Racket's `min` to all the numbers in the list?

Test your modifications. Verify that the calculator correctly evaluates `min` expressions in which the arguments are simple numbers as well as expressions in which the arguments are nested arithmetic expressions; for example, `(min (+ 2 1) (* 3 2) (- 4 2))`

Also, verify that the calculator can evaluate expressions such as this one:

```
(+ 3 (sqrt -4) (** 2 3) (min 4 6 5) (sqrt (* -2 3)))
```