Ngoc N. Tran

CSC-121-01

Professor Turner

**REFLECTIVE ESSAY**

*“As always I am the first in everything.” – Invoker*

This is not the case, sadly. I have only known of this assignment since the day before the exam through Mason, as I did not know there was a description in the Project. I tried looking for it before too, but I went in “Grades” instead of “Assignments,” and as you would have expected by now, there is nothing in the Project page of that section. So, with all my enthusiasm, I finished this Final Project in 2 consecutive all-nighters, with all the ups and downs of a Korean drama.

The main idea of my execution was always stripping all the whitespace for uniformity – it was a requirement of the project to support different whitespace formats, tokenizing numbers and operations using letter-by-letter analysis, and converting the formatted string into a list of lists, which each small list being a mathematics brackets – which would make it very easy to use recursion. I started making a basic binary-tree-like-only machine the first night – it was very easy to just check the length of a list and evaluate if it’s equal to three with the operation being in the middle; then I proceed to convert it into a full-fledged calculator the next night.

Originally, I thought you have to treat your interactive input as a stream of character, so I used the read-word definition in the book to analyze letter-by-letter. After a while, (read-line) comes across me while I was researching Racket functions on Rosetta Code[[1]](#footnote-1), so I converted the function to open a string port rather than reading directly from an input source. Also, originally I replaced the function symbol with the operations directly – I removed this feature later as described in a later paragraph. Evaluation was easy with car, cadr and caddr as mentioned above. Everything took me 15 hours consecutively the first night.

As an overachiever, this was not enough for me. I wanted to make a proper calculator that supports more than just one operation at a time – for example, 1 + 2 + 3 instead of 1 + (2 + 3). This took me the second night. Everything was the same but the evaluation part – but it was not simple as it sounded.

I started the second night with implementing input/output function. This was actually pretty straightforward and did not take much of my time at all. The next three hours were used to look for a good pattern-matching technique. I’ve thought of using 2 accumulators, but the algorithm is not the same for the 6-operation requirement, due to their differences in associativity and order of execution. This proved to be not so effective; and when I get to the inevitable dead-end, and I got insanely depressed and wanted to jump in front of the Crawfordsville - Chicago train.

Surfing through Wikipedia, I found out about the shunting-yard algorithm[[2]](#footnote-2). Thinking that this is the solution, I immediately implemented it, just to realize when it’s too late that converting RPN back to Scheme/Racket’s standard is another huge problem. I then went into a worse depression state. It took me a while to get my cool back and look for a RPN conversion back to Racket. Rosetta Code again came to the rescue – but it gave me more than just a RPN conversion; it gave me the function I have been looking for all the time: match[[3]](#footnote-3). In the hype of finding this gold mine, I commented out all the old evaluate functions, shunting-yard algorithm and RPN conversion, and started re-implementing the new evaluation with match. As usual, new code would always come with bugs and error handling, like match cannot take a procedure as a matching criteria; thus I had to change all the operations to quote (+ to ‘+, remainder to ‘%, etc.) Eventually after a few hours, all worked out just fine. Finishing the code, I went into the CSC exam with head held high and passed it with flying colors.

As of right now, the program has no known bugs. All 6 operations are supported. Users can denote the use of negative number by using brackets and the minus sign – negative 6 can be written as (-6). Errors are handled with try-catch-like structure that prints out the error instead of crashing the machine – which synergizes well with my idea of throwing an error whenever the user enters something that doesn’t make sense. Data protection was not a priority, which explains the definitions of a lot of global variables. I/O are supported – input is for evaluating a batch of equations in a file, and output prints out the calculation into a text file of user’s choice.

With each adventure comes the moral of the story: check the “Assignments” section instead of “Grades.” Wandering around the internet would give you surprisingly useful knowledge. Always move fast and break things[[4]](#footnote-4). People who use Lisp are crazy and dangerous[[5]](#footnote-5). Enjoy object-oriented programming while it lasts.

1. http://rosettacode.org/wiki/User\_input/Text#Racket [↑](#footnote-ref-1)
2. http://en.wikipedia.org/wiki/Shunting-yard\_algorithm [↑](#footnote-ref-2)
3. http://rosettacode.org/wiki/Parsing/RPN\_calculator\_algorithm#Racket [↑](#footnote-ref-3)
4. http://xkcd.com/1428/ [↑](#footnote-ref-4)
5. http://9gag.com/gag/anXEbe0 [↑](#footnote-ref-5)