We will define the language, L, of our rational number calculator program.

Define the set of non-terminal symbols to be

$$N = \{prog, expr, empty, quit, eval, store, add, mult, neg, exp,\\ fact, term, rnd, r, abs, recall, par, dec, int, ws, digit, prev, l\}.$$

Define the set of terminal symbols to be

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -, *, /, \hat{}, !, mod, |(,), round, , , @, s,\\ space, rand, tab, lf, cr, q\}.$$

Define the production rules, P, as the following:

1. $prog \to expr \mid expr\ l \mid expr\ l\ prog$
2. $expr \to empty \mid quit \mid eval \mid store$
3. $empty \to ws$
4. $quit \to ws\ q\ ws$
5. $eval \to ws\ add\ ws$
6. $store \to ws\ s\ ws$
7. $add \to add\ ws\ +\ ws\ mult \mid add\ ws\ -\ ws\ mult \mid mult$
8. $mult \to mult\ ws * ws\ neg \mid mult\ ws\ /\ ws\ neg \mid mult\ ws\ mod\ ws\ neg \mid neg$
9. $neg \to -\ ws\ neg \mid exp$
10. $exp \to fact\ ws\ \hat{}\ ws\ neg \mid fact$
11. $fact \to fact! \mid term$
12. $term \to dec \mid par \mid recall \mid abs \mid r \mid rnd$
13. $rnd \to rand(ws) \mid rand(ws\ add\ ws, ws\ add\ ws)$
14. $r \to round(ws\ add\ ws, ws\ digit\ ws)$
15. $abs \to |ws\ add\ ws|$
16. $recall \to @ \mid @prev$
17. $par \to (ws\ add\ ws)$
18. $dec \to int \mid int.int$
19. $int \to digit \mid digit\ int$
20. $ws \to space\ ws \mid tab\ ws \mid \epsilon$
21. $digit \to prev \mid 0 \mid 9$
22. $prev \to 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8$

23. $l \rightarrow lf \mid cr\ lf$

Note that the use of spaces above is purely for visualization purposes (e.g., *digit int* does not actually have a space). Define the start symbol to be *prog*. Define the unambiguous, context-free grammar to be

$$G = (N, \Sigma, P, prog).$$

Let $\mathcal{L}(G)$ be the language generated from G. When @ is not immediately followed by a *prev*, let it mean @1. @*prev* represents the $prev^{th}$ most-recent result that has been stored from a *store* expression. lf is the Unicode scalar value U+000A, *cr* is the Unicode scalar value U+000D, *space* is the Unicode scalar value U+0020, *tab* is the Unicode scalar value U+0009, and ϵ is the empty string. We define $\mathbb{Q} \subset L \subset \mathcal{L}(G)$ with $\mathbb{Q}$ representing the field of rational numbers such that L extends $\mathbb{Q}$ with the ability to recall the previous one to eight *store* results as well as adds the unary operators $||$, $-$, and ! as well as the binary operators ˆ and *mod* to mean absolute value, negation, factorial, exponentiation, and modulo respectively.

Note that this means for *mult*/*exp*, *exp* does not evaluate to 0. Similarly, *term*ˆ*exp* is valid iff *term* evaluates to 1, *term* evaluates to 0 and *exp* evaluates to a non-negative rational number—0^0 is defined to be 1—or *term* evaluates to any other rational number and *exp* evaluates to an integer or $\pm 1/2$. In the event that *exp* is $\pm 1/2$, then *term* must be the square of a rational number. ! is only defined for non-negative integers. @*prev* is only defined iff at least *prev* number of previous *store* expressions have been evaluated.

mod is defined iff the left operand evaluates to an integer and the right operand evaluates to a non-zero integer. This operator returns the minimum non-negative integer, r, that satisfies the equation

$$n\ mod\ m = r = n - q * m$$

for $n, q \in \mathbb{Z}$, $m \in \mathbb{Z}\backslash\{0\}$, and $r \in \mathbb{N}$.

It also adds the function *round* which rounds the passed expression to *digit*-number of fractional digits. The function *rand* when passed no arguments generates a random 64-bit integer. When the function is passed two arguments, it generates a random 64-bit integer inclusively between the two arguments. In the latter case, the second argument must evaluate to a number greater than or equal to the first; and there must be at least one 64-bit integer in that interval.

From the above grammar, we see the expression precedence in descending order is the following:

1. number literals, (), @, $||$, *round*(), *rand*()
2. !
3. ˆ

4. $-$ (the unary negation operator)
5. $*$, $/$, *mod*
6. $+$, $-$

with ˆ being right-associative and the rest of the binary operators being left-associative. Last, for $j \in \mathbb{N}$ and $d_j \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \subset \mathbb{Z}$, we have

$$\begin{aligned} d_0 d_1 \cdots d_n . d_{n+1} \cdots d_{n+i} = (&d_0 * 10^n + d_1 * 10^{n-1} + \cdots + d_n * 10^0 \\ &+ d_{n+1} * 10^{-1} + \cdots + d_{n+i} * 10^{-i}) \end{aligned}$$

where for $k \in \mathbb{N}$

$$10^k = \overbrace{10 * 10 * \cdots * 10}^{k}$$

and for $l \in \mathbb{Z}^-$

$$10^l = \overbrace{1/10 * 1/10 * \cdots * 1/10}^{|l|}.$$

As a consequence of above, we have the following example:

$$1/1.5 = 1/(3/2) = 2/3 \neq 1/6 = 1/3/2.$$

For $n \in \mathbb{N}$ we define the factorial operator as

$$n! = n * (n - 1) * \cdots * 1$$

which of course equals 1 when $n = 0$.

The *empty* expression (i.e., expression consisting of spaces and tabs) returns the result of the previous *eval* expression in decimal form—in the event there is no such previous expression, it returns the empty string. The minimum number of digits will be used; if the value requires an infinite number of digits, then the value will be rounded to 9 fractional digits. The *quit* expression (i.e., expression consisting of spaces, tabs, and exactly one q) causes the program to terminate. The *store* expression (i.e., expression consisting of spaces, tabs, and exactly one s) stores and returns the result from the previous *eval* expression and can be recalled with @. At most 8 results can be stored; after which the oldest result is overwritten. Stored results cannot be unstored.