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

Define the set of non-terminal symbols to be

$$N = \{expr, add, mult, neg, exp, fact, term, s, dec, int, at, digit, prev\}.$$

Define the set of terminal symbols to be

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

Define the production rules, P, as the following:

1. $expr \to s\ add\ s\ lf \mid s\ =\ s\ at\ s\ lf \mid s\ =\ s\ digit\ at\ s\ lf \mid s\ lf \mid s\ q\ s\ lf$
2. $add \to add\ s\ +\ s\ mult \mid add\ s\ -\ s\ mult \mid mult$
3. $mult \to mult\ s\ *\ s\ neg \mid mult\ s\ /\ s\ neg \mid neg$
4. $neg \to -\ s\ neg \mid exp$
5. $exp \to fact\ s\ \hat{}\ s\ neg \mid fact$
6. $fact \to fact! \mid term$
7. $term \to dec \mid at \mid |s\ add\ s| \mid (s\ add\ s)$
8. $s \to space\ s \mid \epsilon$
9. $dec \to int \mid int.int$
10. $int \to digit \mid digit\ int$
11. $at \to @ \mid @prev$
12. $digit \to prev \mid 0 \mid 9$
13. $prev \to 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8$

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 *expr*. Define the unambiguous, context-free grammar to be

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

Let $\mathcal{L}(G)$ be the language generated from G. Let $@ = @1$, and $@prev$ represent the $prev^{th}$ most-recent result. lf is the Unicode scalar value U+000A, *space* is the Unicode scalar value U+0020, 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 results as well as adds the unary operators $||$, $-$, and ! as well as the binary operator $\hat{}$ to mean absolute value, negation, factorial, and exponentiation respectively. Note that this means for $mult/exp$, exp does not evaluate to 0. Similarly, $term\hat{}exp$ 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. ! is only defined for non-negative integers. @*prev* is only defined iff at least *prev* number of previous expressions have been evaluated. From the above grammar, we see the operator precedence in descending order is the following:

1. (), ||
2. !
3. ˆ
4. − (the unary negation operator)
5. ∗, /
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

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

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 $n = 0$.

For the empty expression and the exit (i.e., q) and "recall" statements (i.e., statements that have =), the previous results are left in tact; all other expressions push the evaluated result to be the next previous result. Recall statements are used purely to display a previous value with the option to round to *digit* number of fractional digits using normal rounding rules. For example,

$$4$$

$$@$$

$$4 + @2$$

returns 4, stores 4 as the previous result, returns 4, pushes 4 to be the second-most previous result, pushes 4 to be the previous result, returns 8, pushes 4 to be the third-most previous result, pushes 4 to be the second-most previous result, and pushes 8 to be the most previous result. In contrast,

$$4$$
$$= @$$
$$4 + @2$$

returns 4, stores 4 as the previous result, returns 4, and fails since the last line is not part of our language L since there is no second-previous result.