Froggy language

Basic Math stuff in frog language

# Keywords

* **^#.\*?:** matches any that begins with ‘#’, and indicates a **commented line.**
* **(Frog)**: Matches "Frog," representing **a binary digit 1**.
* **(Toad)**: Matches "Toad," representing **a binary digit 0**.
* **(FROG)**: Matches "FROG," indicating **an assignment operation**.
* **(fwog)**: Matches "fwog," indicating **an output operation**.
* **Froggy[A-Za-z0-9\_]){1,20}froG**: Matches **a variable name pattern**.
  + Ex. froggybrofroG 🡪 the name of the frog variable is bro
  + Ex2~~. froggyFrog~~ 🡪 the name must be distinguishable from any keywords
  + Ex3. FroggyfrogfroG 🡪 the name of the variable is frog
  + Ex4 FroggyfroG 🡪 ERROR
* **(froggY)**: Matches “froggY” indicating **statement end**
* **(Kermit|exit|quit)**: Matches "Kermit," indicating **program exit**.
* **(Wart)**: Matches "Wart," representing **a signed integer type**.
* **(make)**: Matches “make” , representing a **conversion operator**.
* **(Happy)**: Matches "Happy," representing **a integer type**.
* **(Sad)**: Matches "Sad," representing **an float type**.
* **(Hop)**: Matches "Hop," indicating **a division operation**.
* **(Boing)**: Matches "Boing," indicating **a multiplication operation**.
* **(Croak)**: Matches "Croak," indicating **a subtraction operation**.
* **(Ribbit)**: Matches "Ribbit," indicating **an addition operation**.
* **[\s\t\n\r]+**: Matches whitespace characters and skips them with **no action**.

# Context Free Grammar

program: statement\_list

statement\_list: statement | statement statement\_list

statement: expression END\_OF\_STATEMENT | assignment END\_OF\_STATEMENT | output END\_OF\_STATEMENT

output: OUTPUT\_COMMAND expression { output($2); }

assignment: VARIABLE\_NAME ASSIGN expression { set($1, $3); }

expression: term ADD expression { $$ = $1 + $3; }

| term SUBTRACT expression { $$ = $1 - $3; }

| term

| expression CONVERT INT\_TYPE { $$ = (float)((int)$1); }

| expression CONVERT FLOAT\_TYPE { $$ = $1; }

term: factor MULTIPLY term { $$ = $1 \* $3; }

| factor DIVIDE DIVIDE term { if (isInt($1) && isInt($4)) { $$ = (float)(((int)$1) / ((int)$4)); } else { $$ = $1 / $4; } }

| factor DIVIDE term { $$ = $1 / $3; }

| factor

factor: num { $$ = $1; }

| OPEN\_PREN expression CLOSE\_PREN { $$ = $2; }

| VARIABLE\_NAME { $$ = get($1); }

num: binaryNum { $$ = (float)$1; } | Float

Float: binaryNum PERIOD binaryNum { $$ = (float)$1 + to\_decimal($3); }

binaryNum: binaryNum ZERO { $$ = $1 \* 2 + 0; } // Append 0 (shift left 1)

| binaryNum ONE { $$ = $1 \* 2 + 1; } // Append 1 (shift left 1, add 1)

| ZERO { $$ = 0; }

| ONE { $$ = 1; }

----------------------------------------------------------------------------------------------------------

This grammar takes binary numbers, converts them into integers and stores them as floats. Statements can be an assignment or an output or just an expression (which will do nothing really).

Expressions will do simple math calculations. (+,-,/,\* and convert to (int)). Uses flex’s %left to establish operator precedence.

# Facilities – what does the language provide for you (structures)

* Variables –
  + stored in a finite size struct
  + immutable
* Numerical expressions
  + Int
  + float