Frogger Language Definition.

**VARIABLES:**

The only variable data types are double and string. Variables are defined at first use and are typed based on usage. If a variable is determined to be double it is initialized to 0, conversely the empty string (‘’) for string types. Variable identifiers are alpha with underscores starting with an alpha character. So, valid variable names (identifiers) follow the regex [a-zA-Z][a-zA-Z\_]\*, so numerical digits are not allowed.

**BUILT-IN COMMANDS:**

retrieve() - obtains a double value from the user and can be used anywhere that a double is allowed.

display() - prints the argument to the user; display only takes an arithmetic expression or a string literal.

random() - generates a new pseudo random number between 0 and 1.

end - terminates execution.

**OPERATORS:**

**Arithmetic operators:** (listed in order of operations) [Left Operand – LO, Right Operand – RO]

exponention (^^) - LO is raised to the power of RO (5^^2 = 25)

rootation (##) - RO is rooted to the power of LO (3##8 = 2)

multiplication (\*\*) - Standard multiplication

division (//) - Standard division

integer division (\\) -

modulus division (%%) -

addition (++) - Standard addition

subtraction (--) - Standard subtraction

assignment (=) - RO is evaluated and assigned to LO as a double

**String operators:** (left associative)

string concatenation (++) - Standard concatenation between two strings

assignment (=) - RO is evaluated and assigned to LO as a string

**Boolean operators:** (All standard Boolean operators)

less than (<)

greater than (>)

equal (==)

less than or equal (<=)

greater than or equal (>=)

not (!)

**STRINGS:**

Strings include only printable characters and the listed escape characters enclosed within single quotes. Escape characters: &t (tab), &n (new line), &’ (single quote), and && (ampersand).

No control characters are allowed.

**KEYWORDS:**

retrieve, end, display, random, if, then, and else.

**COMMENTS:**

Comments are enclosed within tildes (~) and are completely ignored.

**PROCESSING ORDER:**

Each FLOWSTMT is associated with a number (starting at 0 and incremented by 1 until the end of file linearly). Frogger is not a linear language; that is, code is not processed top to bottom. Instead, at the end of each JMPSTMT, control is passed to the FLOWSTMT corresponding to the number obtained by the following process:  
Add up all the printable characters’ ascii codes for the current JMPSTMT (excepting extraneous parens and subsequent spaces within string literals), mod this number by the total number of FLOWSTMTs in the source program. (Note ascii values for comment characters are ignored because comments do not carry over into the CFG. Note also that conditional structures themselves are not included in ascii summation because the JMPSTMTs are structures contained within the conditional structure.)

**CFG**:

1. PROG -> FLOWSTMT FLOWSTMTS
2. FLOWSTMTS -> FLOWSTMT FLOWSTMTS
3. |
4. FLOWSTMT -> IFSTMT
5. | JMPSTMT
6. NESTEDFLOWSTMT -> IFSTMT
7. |JMPSTMT

**Control:**

1. IFSTMT -> if ( BOOLEXP ) then NESTEDFLOWSTMT else NESTEDFLOWSTMT
2. BOOLEXP -> EXPR BOOLOP EXPR
3. | EXPR not BOOLOP EXPR
4. BOOLOP -> lt
5. | gt
6. | eq
7. | lte
8. | gte

**Action Statements:**

1. JMPSTMT -> display ( EXPR );
2. | end ;
3. | id assign EXPR ;

**Expressions:**

1. EXPR -> EXPR ADDOP ADDTERM
2. | ADDTERM
3. ADDTERM -> ADDTERM MULOP MULTERM
4. | MULTERM
5. MULTERM -> MULTERM EXPOP EXPTERM
6. | EXPTERM
7. EXPTERM -> dbl
8. | id
9. | string
10. | ( EXPR )
11. | retrieve ( )
12. | random ( )

**Operators:**

1. ADDOP -> add
2. | sub
3. MULOP -> mul
4. | div
5. | mod
6. | idiv
7. EXPOP -> rt
8. | exp

**OBFUSCATION:**

Frogger offers in-line obfuscation if the programmer should choose to further confuse herself/himself. Including the first line of source code as strictly an even number of tildes(~) followed by a carriage return will trigger the de-obfuscator. Note: 0 is considered an even number so if the first character in the source code is a carriage return, the de-obfuscator will run.

Obfuscator Examples (<\n> denotes the new line character within the source code):   
<\n> ~~<\n> ~~~~<\n> ~~~~~~<\n>  
Non-Obfuscated Examples:  
~<\n> (odd number of tildes) ~a~<\n> (‘a’ is not a tilde character) ~~ <\n> (space is not a tilde)

Obfuscation is as follows:  
Each character within identifiers should be ascii incremented based on the number of identifiers occurring prior in the source code. Similarly for each keyword but based on the number of keywords previously occurring. The de-obfuscator will decrement by these counters. Valid symbols are restricted to alphanumeric and the underscore and are incremented in order of ascii value, so order is 0-9A-Z\_a-z. Then if a variable named x\_Dbl is to be used and 6 identifiers have been used between the start of file and the current location, x\_Dbl should be represented (+7) as 4gKis.