Formal grammar specification for this project

Syntax

Basic syntax is NAME = DEFINITION

Aliasing for simpler read: :=

Special keywords NONE empty string, e.g arg list DIGIT 0 through 10 LETTER a through Z

BEGIN DEPRECATED Globbing/patterns uses regex syntax * means 0 or more + one or more. When the normal definition is desirable use * and \+ respectively. Rules are capitalized to distuinguish them from keywords. Groups use brackets. Same rules apply for escaping. END DEPRECATED

```
F:=FUNCTION
S:=STATEMENT
E:=EXPRESSION
T:=TYPE

F = func ID ( ARGS ) RETURN_TYPE S
F = func ID ( ARGS ) S
STRUCT   = struct ID { ARGS }

ID = LETTER
ID = ID LETTER
ID = ID DIGIT

Q := QUALIFIED
Q = ID
Q = ID :: ID

ARGS = NONE
ARGS = ARG
ARGS = ARGS ,
ARGS = ARGS , ARG
ARG  = ID : T

RETURN_TYPE = NONE
RETURN_TYPE = -> T

T = Q
T = T *

S  = S1 ;
S1 = NONE
S1 = BLOCK
```

```
S1 = DECLARE
S1 = CONDITION
S1 = FOR
S1 = WHILE
S1 = RETURN
S1 = break
S1 = continue
S1 = ASSIGN
S1 = E

BLOCK = { BLOCK_INNER }
BLOCK_INNER = NONE
BLOCK_INNER = S
BLOCK_INNER = BLOCK_INNER S

DECLARE = let ARG
DECLARE = let ARG = E

CONDITION = IF
CONDITION = IF ELSE
IF = if ( E ) S
ELSE = else S

FOR = for ( S E S ) S
WHILE = while ( E ) S

RETURN = return
RETURN = return E

ASSIGN = E = E

E = LOGICAL

LOGICAL = COMPARE
LOGICAL = LOGICAL && COMPARE
LOGICAL = LOGICAL || COMPARE

C := COMPARE
AS:=ADDSUB
C = AS
C = C < AS
C = C > AS
C = C <= AS
C = C >= AS
C = C == AS
C = C != AS
```

```
AS = MULDIV
AS = AS + MULDIV
AS = AS - MULDIV

MULDIV = POWER
MULDIV = MULDIV * POWER
MULDIV = MULDIV / POWER

POWER = UNARY
POWER = UNARY ** POWER

U := UNARY
U = + U
U = - U
U = ! U
U = * U
U = & U
U = CAST

P := PRIMARY

CAST = P
CAST = P to TYPE
CAST = P as TYPE
CAST = P [ E ]

P = INT
P = FLOAT
P = STRING
P = Z
P = ( E )
P = sizeof TYPE

Z = QUALIFIED
Z = Z ( PARAMS )
Z = Z -> ID
Z = Z . ID

PARAMS = NONE
PARAMS = E
PARAMS = PARAMS , E

INT = DIGIT
INT = INT DIGIT
```

```
FLOAT = INT . INT
FLOAT = . INT
STRING = I am not going to define a string properly here. You know how it looks. Not interes

TODO: invoke syntax
```