LL - Leahpar`s Language

A ideia: Marcar linhas de uma forma mais visual para que diferentes coisas sejam identificadas facilmente, principalmente quando não estamos utilizando editores de texto que deixem tudo mais legível. Assim, no começo da linha, identificamos o que ocorre em ordem:

@ - Criação de uma váriavel
# - Criação de uma cadeia condicional
$ - Criação de um loop
^ - Criação de uma função

Exemplo:

```
^ def plus(a, b):
        return a+b
@ a = 5
@ b = 5
$ for i in range(0, 5):
        #if(a > 3):
                plus(a+b)
        else:
                a += 1
```

Assim, para procurar a criação de uma variável pode-se procurar somente por @s, etc.

EBNF

<function-definition> ::= ^ {<declaration-specifier>}* <declarator> {<declaration>}* <compound-statement>

<type-specifier> ::= void
| char
| short
| int
| long
| float
| double
| signed
| unsigned
| <struct-or-union-specifier>
| <enum-specifier>
| <typedef-name>

<constant-expression> ::= <conditional-expression>

<equality-expression> ::= <relational-expression>
| <equality-expression> == <relational-expression>
| <equality-expression> != <relational-expression>

```
<additive-expression> ::= <multiplicative-expression>
                        | <additive-expression> + <multiplicative-expression>
                        | <additive-expression> - <multiplicative-expression>

<multiplicative-expression> ::= <cast-expression>
                              | <multiplicative-expression> * <cast-expression>
                              | <multiplicative-expression> / <cast-expression>
                              | <multiplicative-expression> % <cast-expression>

<cast-expression> ::= <unary-expression>
                    | ( <type-name> ) <cast-expression>

<unary-expression> ::= <postfix-expression>
                     | ++ <unary-expression>
                     | -- <unary-expression>
                     | <unary-operator> <cast-expression>
                     | sizeof <unary-expression>
                     | sizeof <type-name>


<primary-expression> ::= @ <identifier>
                       | <constant>
                       | <string>
                       | ( <expression> )

<constant> ::= @ <integer-constant>
             | <character-constant>
             | <floating-constant>
             | <enumeration-constant>

<assignment-operator> ::= =
                        | *=
                        | /=
                        | %=
                        | +=
                        | -=
                        | <<=
                        | >>=
                        | &=
                        | ^=
                        | |=

<unary-operator> ::= &
                   | *
                   | +
```

```
                | -
                | ~
                | !


<selection-statement> ::= # if ( <expression> ) <statement>
                    | if ( <expression> ) <statement> else <statement>
                    | switch ( <expression> ) <statement>

<iteration-statement> ::= $ while ( <expression> ) <statement>
                    | do <statement> while ( <expression> ) ;
                    | for ( {<expression>}? ; {<expression>}? ; {<expression>}? ) <statement>

<jump-statement> ::= goto <identifier> ;
                | continue ;
                | break ;
                | return {<expression>}? ;
```