# Project Language Specification (Attributed Grammar)

## 1 Syntax

(1) `<prog>` → `<stmt_list>`
(2) `<stmt_list>` → $\epsilon$
(3) | `<stmt>` ";" `<stmt_list>`
(4) `<stmt>` → `<print>`
(5) | `<input>`
(6) | `<assign>`
(7) | `<if>`
(8) | `<while>`
(9) `<print>` → "print" `<p-arg>`
(10) `<p-arg>` → **STRING**
(11) | `<expr>`
(12) `<input>` → "get" **ID**
(13) `<assign>` → **ID** "=" `<expr>`
(14) `<if>` → "if" `<expr>` "then" `<stmt_list>` "else" `<stmt_list>` "end"
(15) `<while>` → "while" `<expr>` "do" `<stmt_list>` "end"
(16) `<expr>` → `<n_expr>` `<b_expr>`
(17) `<b_expr>` → $\epsilon$
(18) | "and" `<n_expr>`
(19) | "or" `<n_expr>`
(20) `<n_expr>` → `<term>` `<t_expr>`
(21) `<t_expr>` → $\epsilon$
(22) | "+" `<n_expr>`
(23) | "-" `<n_expr>`
(24) `<term>` → `<factor>` `<f_expr>`
(25) `<f_expr>` → $\epsilon$
(26) | "*" `<term>`
(27) | "/" `<term>`
(28) | "%" `<term>`
(29) `<factor>` → `<value>` `<v_expr>`
(30) `<v_expr>` → $\epsilon$
(31) | ">" `<value>`
(32) | ">=" `<value>`
(33) | "<" `<value>`

(34) | "<=" `<value>`
(35) | "==" `<value>`
(36) | "!=" `<value>`
(37) `<value>` → "(" `<expr>` ")"
(38) | "not" `<value>`
(39) | "-" `<value>`
(40) | **ID**
(41) | **INT**

## 1.1 Tokens

This subsection describes the token used in the above grammar. Provided for each token is a regex and a description. The regex is for those that know regular expressions and prefer it as a description. The description says the same thing in English. Preprocessing describes how the lexeme is transformed before passing it to the parser.

**STRING**

**As a regex:** `"([^"]|\")*"`

**Description:** A quotation mark followed by zero or more characters, where quotation marks must be preceded by a backslash, followed by another quotation mark.

**Preprocessing:** The first and last quotation marks are removed. Scanning from left to right, "\\" is replaced with "\", "\t" is replaced with a tab, "\n" is replaced with a newline, "\"" is replaced with """, and any "\" that is followed by anything else is removed.

**ID**

**As regex:** [_a-zA-Z][_a-zA-Z0-9]*

**Description:** A letter or underscore followed by a combination of zero or more letters, underscores or digits.

**INT**

**As Regex:** (+|-)?[0-9]+

**Description:** an optional "+" or "-" followed by one or more digits.

# 2 Static Semantics

*A variable must be defined before it is used*

1 `<stmt_list>`.*ids* = {}

3 <stmt_list> [1].*ids* = <stmt> .*id* ∪ <stmt_list> [0].*ids*
<stmt> .*ids* = <stmt_list> [0].*ids*

4 <print> .*ids* = <stmt> .*ids* <stmt> .*ids* = {}

5 <stmt> .*id* = { <input> .*id*}

6 <stmt> .*id* = { <assign> .*id*}
<assign> .*ids* = <stmt> .*ids*

7 <if> .*ids* = <stmt> .*ids* <stmt> .*ids* = {}

8 <while> .*ids* = <stmt> .*ids* <stmt> .*ids* = {}

9 <p-arg> .*ids* = <print> .*ids*

11 <expr> .*ids* = <p-arg> .*ids*

12 <input> .*id* = **ID** .*id*

13 <assign> .*id* = **ID** .*id*

16 <n_expr> .*ids* = <expr>
<b_expr> .*ids* = <expr>

18 <n_expr> .*ids* = <b_expr>

19 <n_expr> .*ids* = <b_expr>

20 <term> .*ids* = <b_expr>
<t_expr> .*ids* = <b_expr>

22 <n_expr> .*ids* = <t_expr> .*ids*

23 <n_expr> .*ids* = <t_expr> .*ids*

24 <factor> .*ids* = <term> .*ids*
<f_expr> .*ids* = <term> .*ids*

26 <term> .*ids* = <f_expr> .*ids*

27 <term> .*ids* = <f_expr> .*ids*

28 <term> .*ids* = <f_expr> .*ids*

29 <value> .*ids* = <factor> .*ids*
<v_expr> .*ids* = <factor> .*ids*

31 <value> .*ids* = <v_expr> .*ids*

32 <value> .*ids* = <v_expr> .*ids*

33 <value> .*ids* = <v_expr> .*ids*

34 <value>.*ids* = <v_expr>.*ids*

35 <value>.*ids* = <v_expr>.*ids*

36 <value>.*ids* = <v_expr>.*ids*

37 <expr>.*ids* = <value>.*ids*

38 <value>[1].*ids* = <value>[0].*ids*

39 <value>[1].*ids* = <value>[0].*ids*

40 Predicate: **ID**.*id* ∈ <value>.*ids*