Department of Computer Engineering

CS-315

Project 1

Group:

Aslı Karaman (21901576)

Ege Safi (22001987)

Eren Duran (22003932)

Team : 26

Language Name: JAVA - -

1) BNF Start Variable

- Program begins with statement list

<program> ::=#begin_program_statement <stmt_list> #end_program_statement

2) BNF Statements

- In our language, statement list represents a single or more statements. Statement is either matched, unmatched or declares a function.
- Matched: Number of if and else statements are equal.
- Matched List: Can either represent a single matched or more.
- Unmatched: Unmatched represents a statement that the number of if and else statements are not equal.
- Unmatched List: Can either represent a single unmatched or more.
- Declaration statement declares a variable. Identifier list represents one or more variable identifiers in our language. Assignment statement assigns an expression to an already existing identifier. Return statement returns expression with return function. While statement represents a while loop that has expression within parentheses and statement list in brackets in our language. For statement represents a for-loop that has declaration statement, expression and assignment statement in parnatheses respectively, separated by semicolons with statement list between brackets. Do while statement represents a do while loop in which a statement is written first and then an expression in a while loop.

<stmt_list> ::= <stmt>
| <stmt> <stmt_list>

<stmt> ::= <matched> | <unmatched> | <function_declaration>

<matched> ::= IF LP <expr> RP THEN LEFT_BRACE <matched_list> RIGHT_BRACE ELSE LEFT_BRACE <matched_list> RIGHT_BRACE | <declaration_stmt> |

<assign_stmt> | <for_stmt> | <while_stmt> | <do_while_stmt> | <return_stmt>| <function_call> | <comment> | <connection> | <connection_state> | <switch_state> | <read>

<unmatched> ::= IF LP <expr> RP LEFT_BRACE <stmt_list> RIGHT_BRACE
| IF LP <expr> RP THEN LEFT_BRACE <matched_list> RIGHT_BRACE ELSE LEFT_BRACE <unmatched_list> RIGHT_BRACE

<matched_list> ::= <matched> | <matched> <matched_list>

<unmatched_list> ::= <unmatched> | <unmatched> <unmatched_list>

<declaration_stmt> ::= <type_id> <var_id> SEMICOLON

<assign_stmt> ::= <var_id> ASSIGNMENT_OP <expr> SEMICOLON | <var_id> ASSIGNMENT_OP <function_call>

<return_stmt> ::= RETURN <expr> SEMICOLON

<while_stmt> ::= WHILE LP <expr> RP LEFT_BRACE <stmt_list> RIGHT_BRACE

<for_stmt> ::= FOR LP <declaration_stmt> <expr> SEMICOLON <assign_stmt> RP LEFT_BRACE <stmt_list> RIGHT_BRACE

<do_while_stmt> ::= DO LEFT_BRACE <stmt_list> RIGHT_BRACE WHILE LP <expr> RP SEMICOLON

3) **BNF Components**

- Component consists of either sensor or a timer variable. The sensor consists of the variables listed as temperature humidity, air_pressure, air_quality, light, sound_level, proximity, gps, radio.

- There are 10 switches in total, each represented within the switch list:
- s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_10
- Switch state will be determined by switch_state. It will be a boolean variable.

<component> ::= <sensor>
| TIMER

<sensor> ::= TEMPERATURE
| HUMIDITY
| AIR_PRESSURE
| AIR_QUALITY
| LIGHT
| SOUND_LEVEL
| PROXIMITY
| GPS
| RADIO

<switch_list> ::= SWITCH_1
| SWITCH_2
| SWITCH_3
| SWITCH_4
| SWITCH_5
| SWITCH_6
| SWITCH_7
| SWITCH_8
| SWITCH_9
| SWITCH_10

<switch_state> ::= <switch_list> LP <boolean_variable> RP SEMICOLON

4) Component Functions

- The reader function reads sensor information a its variable identifier. Connection function either sends or receives URL information. The send function takes URL as its parameter and sends an expression. Function of receive takes URL as its parameter and returns an integer that you can assign to variable. Connection state tests the URL connection in terms of connectivity.

<read> ::= <component> LP <var_id> RP SEMICOLON

<connection> ::= <send_int> | <receive_int>

<connection_state> ::= CONNECTION LP <url_prefix> COMMA <boolean_variable> RP SEMICOLON

<send_int> ::= SEND LP <url_prefix> RP <expr> SEMICOLON

<receive_int> ::= RECEIVE LP <url_prefix> RP <var_id> SEMICOLON

5) Expressions

- Expressions consist of digits, variable identifiers and their summations multiplications, divisions and modulo in our language. There are also comparison identifiers (greater than, less than, equal, not equal, greater than or equal, less than or equal). The type identifiers are int, float, string and char.

<expr> ::= <expr> PLUS term | expr OR term | term

<term> ::= <term> STAR <factor> | <term> MINUS <factor> | <term> DIVISION <factor> | <term> MODULO <factor> | <term> AND <factor> | <term> LESS_THAN <factor> | <term> GREATER_THAN <factor> | <term> EQUALITY_OP <factor> | <term> NOT_EQ_OP <factor> | <term> GREATER_EQ <factor> | <term> LESS_EQ <factor> | <factor>

<factor> ::= <var_id> | <int_const> | <boolean_variable> | float | string | char

<string> ::= STRING

<char> ::= CHAR

<float> ::= FLOAT

<type_id> ::= INT| FLO | STR | CH

<var_id> ::= VARIABLE_ID

<int_const> ::= INTEGER

<boolean_variable> ::= TRUE | FALSE

6) URL

- URL represents the prefix of a dots between the consequent digits in our language.

<url_prefix> ::= <int_const> DOT <int_const> DOT <int_const> DOT <int_const>

7) Functions

- Functions are called with its parameter. Parameters are variable identifier(s) to be put into functions. For the declaration of the functions, its visibility, return type, name, parameter as argument list, and statement list are included in our language.

- The argument list represents a single or more arguments and argument itself represents a declaration statement. Functions start with “function” keyword. Visibility of a function in our language is that if a function will either be public, private or protected.

<function_call> ::= <var_id> LP <parameter> RP SEMICOLON

<parameter> ::= <var_id> | <var_id> COMMA <parameter>

<function_declaration> ::= FUNCTION_START <visibility> <type_id> <var_id> LP <arg_list> RP LEFT_BRACE < stmt_list> RIGHT_BRACE

<arg_list> ::= <arg>
| <arg> COMMA <arg_list>

<arg> ::= <declaration_stmt>

<visibility> ::= PUBLIC | PRIVATE | PROTECTED

8) Comment

- Comment functionality can be used for adding comments on the code. Comment statement can either begin with a letter, a digit, a sign and/or more in our language.

<comment> ::= COMMENT

9) Terminals

- The terminals in our language is similar to all other languages.

1. != Not equal operator for logic expressions
2. == equality operator
3. # comment starter
4. + plus operator
5. - minus operator
6. . dot for url
7. ; statement end semicolon
8. % modulo operator
9. >= bigger or equal comparator
10. <= less or equal comparator
11. > bigger than comparator
12. < less than comparator
13. (left parentheses
14.) right parentheses
15. = assignment statement
16. { left curly brace
17. } right curly brace
18. , colon for list items
19. / division operator
20. * multiplication operator

10) Reserve Words

- Our language uses the following reserve words

1. for
2. while
3. do
4. send
5. receive
6. private

7. public
8. protected
9. return
10. if
11. then
12. else
13. true
14. false
15. int
16. float
17. string
18. char
19. function
20. temperature
21. humidity
22. air_pressure
23. air_quality
24. light
25. sound_level
26. proximity
27. gps
28. radio
29. timer
30. s_1
31. s_2
32. s_3
33. s_4
34. s_5
35. s_6
36. s_7
37. s_8
38. s_9

39.s_10

11) Readability

JAVA- - language is pretty similar to most common object oriented programming languages in a lot of ways. The added built in variables for all the sensors in plain english enables the users to use the IOT (Internet of things)sensors with little to no effort. The language defined functions like “send” and “recieve” also allows new users to read what they have typed with great ease. Also, the multiline acceptance of input makes the language a bit harder to read in terms of readability.

12) Writability

The writability is similar to other languages with some improvements like elimination of the double type and wide range of usage nearly for all the operators. However, our language is still requires data types in order to work flawlessly in that aspect it is worse than the fast prototyping language Python but it is pretty simillar to the other languages.

13) Reliability

We have followed the correct recursive definitions for arithmetic operators and assignment operators. Also we have added the “matched” and “unmatched” statement differentiation in order to avoid confusion and ambiguity. However, we have not completely eliminated ambiguity in any way because some ambiguous parts allow the developer to read our code more easily and write codes with little to no effort.