CS315 PROGRAMMING LANGUAGES PROJECT-1

IsOT PROGRAMMING LANGUAGE

2022-23 Fall

Group:08

Group Members

Mahmut Furkan Gön Section-3 21902582
Ahmet Arda Ceylan Section-3 22003148
Omar Hamdash Section-3 22001398

Complete BNF Description of IsOT Language

<program> ::= **start**<stmts>**end**

<stmts> ::= <stmt> | <stmt><stmts>

<stmt> ::= <matched_stmt> | <unmatched_stmt>

<matched_stmts> ::= <matched_stmt> | <matched_stmt> <matched_stmts>

<unmatched_stmts> ::= <unmatched_stmt> | <unmatched_stmt> <unmatched_stmts>

<unmatched_stmt> ::= **if**<LP><logic_expr><RP><LB><unmatched_stmts><RB>
| **if**<LP><logic_expr><RP><LB><matched_stmts><RB><trial>

<trial> ::= // empty
| else<LB><unmatched_stmts><RB>

<matched_stmt> ::=
if<LP><logic_expr><RP><LB><matched_stmts><RB>**else**<LB><>matched_stmts> <RB>
|<assign_stmt><end_stmt>
| <input_stmt><end_stmt>
| <output_stmt><end_stmt>
| <func_define>
|<func_call><end_stmt>
| <read_from_sensor><end_stmt>
| <time_from_timer><end_stmt>
| <connect_stmt><end_stmt>
| <declare_stmt><end_stmt>
| <break_stmt><end_stmt>
| <continue_stmt><end_stmt>
| <return_stmt> <end_stmt>
| <arithmetic_op><end_stmt>
| <comment>
| <end_stmt>
| <while_stmt>
| <for_stmt>
| <input_from_connection><end_stmt>
| <output_to_connection><end_stmt>
| <connect_obj_creation><end_stmt>
| <get_connect_status><end_stmt>
| <get_connect_protocol><end_stmt>
| <get_connect_URL><end_stmt>
| <turn_switch_on><end_stmt>
| <turn_switch_off><end_stmt>
| <toggle_switch><end_stmt>
| <get_switch_name><end_stmt>

1. Function Definition and Function Call

<func_define> ::= **func**<space><func_var><LP><parameters><RP><LB><stmts><RB>
<parameters> ::= <parameter> | <parameter><comma><parameters>
<parameter> ::= <var_type><space><identifier> | <empty>
<empty> ::= // empty
<func_call> ::= <identifier><LP><variables><RP>
<variables> ::= <identifier> | <identifier><comma><variables>
<var_type>::=**int | float | string | char | bool | status | protocol**

2. Continue-Break, Return, and End Statements

<continue_stmt> ::= **continue**
<break_stmt> ::= **break**
<return_stmt> ::= **return**
<values> ::= <bool> | <string> | <char> | <int> | <float>
<end_stmt> ::= <semicolon>

3. Comment

<comment> ::= <hashtag><sentence><newline>

4. Arithmetic Operations

<arithmetic_op>::= <numeric_value>
| <arithmetic_op><plus_op><numeric_value>
| <arithmetic_op><minus_op><numeric_value>

<numeric_value>::= <num_value> | <numeric_value><mult_op> <num_value> |
<numeric_value><divide_op><num_value>
<num_value> ::= <int> | <float> | <LP><arithmetic_op><RP>

5. Data Types

<int> ::= <digits> | <minus_op><digits>
<digits> ::= <digit> | <digit><digits>
<digit> ::= 0|1|2|3|4|5|6|7|8|9

<float> ::= <digits><dot><digits>
| <minus_op><digits><dot><digits>
| <dot><digits>
| <minus_op><dot><digits>

<letter>::= <uppercase_letter> | <lowercase_letter>
<uppercase_letter>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<lowercase_letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

6. Input Output

<input_stmt> ::= **input**<LP><identifier><RP>

<input_from_connection> ::= <identifier><dot>**input_from_connection**<LP><inbody><RP>

<output_stmt> ::= **print**<LP><identifier><RP>

<output_to_connection> ::= <identifier><dot>**output_to_connection**<LP><identifier><RP>

<read_from_sensor> ::= **read_from_sensor**<LP><identifier><comma><identifier><RP>

<time_from_timer> ::= **time_from_timer**<LP><identifier><RP>

<URL> ::= <string>

7. Creating a Connection Object and All Related Statements

<connect_obj_creation> ::= **Connection_Class**<space><connect_var><assign_op>
new<space>**Connection_Constructor**

<connect_stmt> ::= <identifier><dot>**connect**<LP><URL><RP>

<get_connect_status> ::= <identifierr><dot>**getStatus**<LP><identifier><RP>

<get_connect_protocol> ::= <identifier><dot>**getProtocol**<LP><identifierr><RP>

<get_connect_URL> ::= <identifier><dot>**getURL**<LP><identifier><RP>

<status_value> ::= **connected** | **connecting** | **disconnected** | **host_not_found** |
connection_timeout

<protocol_value> ::= **http** | **https** | **tcp** | **ftp** | **tftp**

8. Conditional Statements: If-Else

<matched_stmt> ::=
if<LP><logic_expr><RP><LB><matched_stmt><RB>**else**<LB><>matched_stmt> <RB>|
<non-if_stmt>

<unmatched_stmt> ::= **if**<LP><logic_expr><RP><LB><stmts><RB>
|**if**<LP><logic_expr><RP><LB> <matched_stmt><RB>**else** <LB><unmatched_stmt><RB>

9. Loops: While and For

<while_stmt>::= **while** <LP> <logic_expr> <RP> <LB><stmts><RB>

<for_stmt>::= **for** <LP> <declare_stmt><end_stmt> <logic_expr>
<end_stmt><assign_stmt><RP> <LB><stmts><RB>

10. Logic Expressions

```
<logic_expr_wout_id>::= <bool_value>
                | <logic_operation>
                | <comparison_operation>
                | <not_op>  <logic_expr>



<logic_expr>::= <logic_operation>
                | <comparison_operation>
                | <not_op>  <logic_expr>
                | <identifier>


<logic_operation>::= <logic_value>
                        |<logic_op><or_op><logic_value>
<logic_value>::= <bool_factor>|
                <logic_value><and_op><bool_factor>
<bool_factor> ::=
                <bool_value>
                | <bool_var>
                | <LP><comparison_operation><RP>
                | <LP><logic_operation><RP>



<comparison_operation>::=   <num_value><comparison_op><num_value>

<comparison_op>::= <less_op> | <greater_op> | <equivalent_op> | <greatEq_op>
| <lessEq_op> | <not_eq_op>
```

11. Operators

<plus_op>::= +
<minus_op>::= -
<mult_op>::= *
<divide_op>::= /
<expo_op>::= ^

<assign_op>::= =

<and_op>::= &
<or_op>::= |

<less_op>::= <
<greater_op>::= >
<equivalent_op>::= ==
<greatEq_op>::= >=
<lessEq_op>::= <=
<not_eq_op>::= !=
<not_op>::= !

12. Declaration Statements

```
<declare_stmt>::= <var_type><space><identifier>
                | <var_type><space><assign_stmt>
```

13. Assignment Statements

```
<assign_stmt>::=    <identifier><assign_op><identifier>
                    | <identifier><assign_op><num_value>
                    | <str_assign>
                    | <char_assign>
                    | <bool_assign>
                    | <status_assign>
                    | <protocol_assign>
```

<str_assign>::= <identifier><assign_op><string>
<char_assign>::= <identifier><assign_op><char>
<bool_assign>::= <identifier><assign_op><logic_expr_wout_id>
<status_assign>::= <identifier><assign_op><status>
<protocol_assign>::= <identifier><assign_op><protocolr>

<numeric_value>::= <num_value> | <numeric_value> <mult_op><num_value>|
<numeric_value> <div_op><num_value>
<num_value>::= <int> | <float> | <LP><arithmetic_operation><RP>
<bool_value>::= **true** | **false** | **yes** | **no** | **on** | **off**

<identifier>::= <char_list>
<char_list>::= <alphanumeric> | <alphanumeric><char_list>

14. Switches

<turn_switch_on> ::= **turn_switch_on**<LP><switch_name><RP>
<turn_switch_off> ::= **turn_switch_off**<LP><switch_name><RP>
<toggle_switch> ::= **toggle_switch**<LP><switch_name><RP>
<get_switch_state> ::= **get_switch_state**<LP><switch_name><comma><identifier><RP>
<switch_name>::= **switch_0**
| switch_1
| **switch_2**
| switch_3
| **switch_4**
| switch_5
| **switch_6**
| switch_7
| **switch_8**
| switch_9

Explanation of the Language Constructs

1. Program

<program> ::= **start**<stmts>**end**

This non-terminal is our whole program. Our program starts with the “start” command and ends with the “end” command. Other statements come between these two comments.

2. Statements

<stmts> ::= <stmt> | <stmt><stmts>

This non-terminal states that all of the statements in the program consist of either a statement or a list of statements that are described recursively.

3. Statement

<stmt> ::= <matched_stmt> | <unmatched_stmt>

This non-terminal says that a statement is either a matched statement or an unmatched statement.

4. Matched Statements

<matched_stmts> ::= <matched_stmt> | <matched_stmt> <matched_stmts>

This non-terminal is defined as a matched statement or a matched statement followed by matched statements

5. Unmatched Statements

<unmatched_stmts> ::= <unmatched_stmt> | <unmatched_stmt> <unmatched_stmts>

This non-terminal is defined as an unmatched statement or an unmatched statement followed by unmatched statements

6. Unmatched Statement

<unmatched_stmt> ::=
if<LP><logic_expr><RP><LB><unmatched_stmts><RB> |
if<LP><logic_expr><RP><LB><matched_stmts><RB><trial>

This non-terminal states that an unmatched statement is either a trial-less if or an if with trial. A trial-less if contains if reserved word followed by logic expression inside the parentheses and unmatched statements between braces follow these.
An if with trial is similar to an trial-less if except it contains matched statements between braces and has trial at the end.

7. Trial

<trial> ::= // empty

| else<LB><unmatched_stmts><RB>

A trial is defined as either nothing or else followed by left parenthesis followed by unmatched statements followed by a right parenthesis. This allows the if statement not to have an else and also allows an else with unmatched statements between braces.

8. Matched Statement

<matched_stmt> ::= **if**<LP><logic_expr><RP><LB><matched_stmts><RB>**else**<LB><>matched_stmts> <RB>

|<assign_stmt><end_stmt>

| <input_stmt><end_stmt>

| <output_stmt><end_stmt>

| <func_define>

|<func_call><end_stmt>

| <read_from_sensor><end_stmt>

| <time_from_timer><end_stmt>

| <connect_stmt><end_stmt>

| <declare_stmt><end_stmt>

| <break_stmt><end_stmt>

| <continue_stmt><end_stmt>

| <return_stmt> <end_stmt>

| <arithmetic_op><end_stmt>

| <comment>

| <end_stmt>

| <while_stmt>

| <for_stmt>

| <input_from_connection><end_stmt>

| <output_to_connection><end_stmt>

| <connect_obj_creation><end_stmt>

| <get_connect_status><end_stmt>

| <get_connect_protocol><end_stmt>

| <get_connect_URL><end_stmt>

| <turn_switch_on><end_stmt>

| <turn_switch_off><end_stmt>

| <toggle_switch><end_stmt>

| <get_switch_name><end_stmt>

This non-terminal states that a matched statement is either one of the following:

- if reserved word, and logic expressions between parentheses. Matched statements between braces follows this. After them, else reserved word comes, and it is followed by matched statements between braces.
- <assign_stmt><end_stmt>
- <input_stmt><end_stmt>
- <output_stmt><end_stmt>
- <func_define>
- <func_call><end_stmt>
- <read_from_sensor><end_stmt>
- <time_from_timer><end_stmt>
- <connect_stmt><end_stmt>
- <declare_stmt><end_stmt>
- <arithmetic_op><end_stmt>
- <break_stmt><end_stmt>
- <continue_stmt><end_stmt>
- <comment>
- <end_stmt>
- <for_stmt>
- <while_stmt>

9. While Statement

<while_stmt>::= **while** <LP> <logic_expr> <RP> <LB><stmts><RB>

This non-terminal states the syntax of the while statement. It contains while reserved word followed by logic expression inside parentheses. Statements inside the braces follow this.

10. For Statement

<for_stmt>::= **for** <LP> <declare_stmt><end_stmt> <logic_expr> <end_stmt><assign_stmt><RP> <LB><stmts><RB>

This non-terminal states for loops' syntax in our language. A for loop contains for reserved word. Then, in the parentheses a declare statement, logic expression, and assignment statement which are separated by end statements follow. Statements inside the braces follow these.

11. Logic Expressions

<logic_expr>::= <logic_operation> | <comparison_operation> | <not_op> <logic_expr> | <identifier>

This non-terminal states the logic expression. logic expression can be logic operation or comparison operation or not operator with logic expression or an identifier.

<logic_expr_wout_id>::= <bool_value> | <logic_operation> | <comparison_operation> | <not_op> <logic_expr>

This non-terminal states the logic expression without identifier.

<logic_operation>::= <logic_value>|<logic_expr><or_op><logic_value>

This non-terminal states that logic operation can be either a logic value or logic expression, or operand and logic value.

<logic_value>::= <bool_factor> | <logic_value><and_op><bool_factor>

This non-terminal states that logic value can be boolean factor or logic value, and operator, boolean factor.

<bool_factor> ::= <bool_value> | <bool_var> | <LP><comparison_operation><RP> | <LP><logic_operation><RP>

This non-terminal states that boolean factor or logic value, and operator, boolean factor.

<comparison_operation>::= <num_value><comparison_op><num_value>

This non-terminal states the comparison operation that is num value followed by comparison operator and num value.

<logic_op>::= <and_op> | <or_op>

This non-terminal states the logic operators. In our language there are two logic operators: and operator and or operator.

<comparison_op>::= <less_op> | <greater_op> | <equivalent_op>
| <greatEq_op> | <lessEq_op> | <not_eq_op>

This non-terminal states what the comparison operators are. In our language, comparison operators are less than, less than or equal, equivalent, greater than, greater than or equal to and not equal operators.

12. End Statement

<end_stmt> ::= <semicolon>

This non-terminal says that an end statement is a semicolon.

13. Comment

<comment> ::= <hashtag><sentence><newline>

This non-terminal states the syntax of the comment. Comments start with a hashtag symbol (#) and end with a newline character. Inside these two, it counted as sentence and isn't implemented as comments

14. Continue Statement

<continue_stmt> ::= **continue**

This terminal states that continue statement is fulfilled by the reserved word continue. It is used to break one iteration in the loops.

15. Break Statement

<break_stmt> ::= **break**

This terminal states that break statement is fulfilled by the reserved word break. It is used in loops to terminate the loop.

16. Data Types

<int> ::= <digits> | <minus_op><digits>

This non-terminal defines that integers consist either of digits or of minus operation followed by digits.

<digits> ::= <digit> | <digit><digits>

This non-terminal statements define digits as either one digit or a list of digits.

<digit> ::= 0|1|2|3|4|5|6|7|8|9

This terminal defines digit as numbers from 0 to 9.

<float> ::= <digits><dot><digits>

| <minus_op><digits><dot><digits>

| <dot><digits>

| <minus_op><dot><digits>

This non-terminal defines floats that they can either be digits (digits defined above) a dot followed by another digits, with an optional minus sign before it, or they can just a dot followed by digits, with an optional minus sign before that.

<letter>::= <uppercase_letter> | <lowercase_letter>

This non-terminal defines that letter is either an uppercase letter or a lowercase letter.

<uppercase_letter>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

This terminal defines uppercase letters that are letter from A to Z.

<lowercase_letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

This terminal defines lowercase letters that are letter from a to z.

17. Arithmetic Operations

<arithmetic_op>::=	<numeric_value>

| <arithmetic_op><plus_op><numeric_value>

| <arithmetic_op><minus_op><space><numeric_value>

This non-terminal states that an arithmetic operation can be a numeric value or add or subtract operations(There must be a space between minus operator and the numeric value in our language. Spaces in other operations are optional). Arithmetic operations designed in a way that respects operation precedence.

<numeric_value>::= <num_value> | <numeric_value><mult_op> <num_value> | <numeric_value><divide_op><num_value>

This non-terminal states that numeric value can be a num_value or division or multiplication operation with the first term being numeric_value and the second being num_value. Numeric values designed in a way that respects operation precedence.

<num_value> ::= <int> | <float> | <LP><arithmetic_op><RP>

This non-terminal states that a num_value can either be an integer value or a float value or an arithmetic operation in parentheses.

18. Function Definition and Function Call

<func_define> ::= func<space><func_var><LP><parameters><RP><LB><stmts><RB>

This non-terminal defines the form of the function definition of the language. Function definition starts with “func” reserved word followed by space and function variable then, function parameters between two parentheses. Finally, followed by function body between braces.

<parameters> ::= <parameter> | <parameter><comma><parameters>

Parameters are defined as parameter or parameter followed by comma and parameters to include multiple amount of parameters.

<parameter> ::= <var_type><space><identifier> | <empty>

This non-terminal is defined as variable type followed by a space followed by an identifier or <empty> since the parameter list can be empty.

<empty> ::= //empty

This non-terminal is defined as nothing

<func_call> ::= <identifier><LP><variables><RP>
This non-terminal is defined as an identifier followed by variables between parentheses.

<variables> ::= <identifier> | <identifier><comma><variables>
This non-terminal is defined as identifier or identifier followed by comma and variables to have multiple variables .

<var_types>::=**int | float | string | char | bool | status | protocol**
This terminal is defined as one of the int, float, string, char, bool, status, protocol key words.

19. Assign Statements

<assign_stmt>::= <identifier><assign_op><identifier>

| <identifier><assign_op><num_value>

| <str_assign>

| <char_assign>

| <bool_assign>

| <status_assign>

| <protocol_assign>

This non-terminal states that there are seven different types of assignment. These are identifier followed by assign operator followed by an identifier or an identifier followed by assign operator followed by a num_value or one of: string, character, boolean, status and protocol assignments.

<str_assign>::= <str_var><assign_op><string>
This non-terminal says that string assignment is that string variable name immediately followed by the assign operator, which is = . After that, there comes a string value.

<char_assign>::= <char_var><assign_op><char>
This non-terminal says that character assignment is that character variable name immediately followed by the assign operator, which is = . After that, there comes a character value.

<bool_assign>::= <bool_var><assign_op><bool_value>

| <bool_var><assign_op><logic_expr>

This non-terminal says that boolean assignment is that boolean variable name immediately followed by the assign operator, which is = . After that, there can be either a boolean value or a logic expression.

<status_assign>::= <status_var>=<status_var>

| <status_var><assign_op><status_value>

This non-terminal says that status assignment is that status variable name immediately followed by the assign operator, which is = . After that, there can be either a status value or another status variable name.

<protocol_assign>::= <protocol_var><assign_op><protocol_var>

| <protocol_var><assign_op><protocol_value>

This non-terminal says that protocol assignment is that protocol variable name immediately followed by the assign operator, which is = . After that, there can be either a protocol value or another protocol variable name.

<num_var>::= <int_var> | <float_var>

This non-terminal states that a numeric variable name is either an integer variable name or a float variable name.

<num_value>::= <int> | <float> | <arithmetic_operation> | <char>

This non-terminal states that a numeric value can be an integer value, a float value, an arithmetic operation or a character.

<bool_value>::= **true** | **false** | **yes** | **no** | **on** | **off**

This terminal states that a boolean can have values true, yes, on, false, no and off. The first three corresponds 1 and the rest corresponds 0.

<var>::= <string_var>

| <char_var>

| <int_var>

| <float_var>

| <bool_var>

| <func_var>

| <status_var>

| <protocol_var>

This non-terminal states that there are six different types of variable names. These are integer, string, character, float, boolean, status and protocol variable names.

<string>::= <string_id><identifier><string_id>

This non-terminal states that a string corresponds to an identifier consists of the same characters between double-quotes.

<char>::= <char_id><alphanumeric><char_id>

This non-terminal states that a character corresponds to an alphanumeric consists of the same characters between single-quotes.

<identifier>::= <char_list>

This non-terminal says that identifier is a charlist.

<char_list>::= <char> | <char><char_list>

This non-terminal states that a charlist can be a character or a list of characters.

<char>::= <letter> | <digit>

This non-terminal states that a character is a letter or a digit.

<char_var>::= <letter> | <letter><identifier>

This non terminal states that a character variable’s name (variable name) is either a letter or a letter followed by an identifier.

<string_var>::= <letter> | <letter><identifier>

This non terminal states that a string variable's name (variable name) is either a letter or a letter followed by an identifier.

<int_var>::= <letter> | <letter><identifier>

This non terminal states that a integer variable's name (variable name) is either a letter or a letter followed by an identifier.

<float_var>::= <letter> | <letter><identifier>

This non terminal states that a float variable's name (variable name) is either a letter or a letter followed by an identifier.

<bool_var>::= <letter> | <letter><identifier>

This non terminal states that a boolean variable's name (variable name) is either a letter or a letter followed by an identifier.

<func_var>::= <letter> | <letter><identifier>

This non terminal states that a function's name (function name) is either a letter or a letter followed by an identifier.

<status_var>::= <letter> | <letter><identifier>

This non terminal states that a status variable's name (variable name) is either a letter or a letter followed by an identifier.

<protocol_var>::= <letter> | <letter><identifier>

This non terminal states that a protocol variable's name (variable name) is either a letter or a letter followed by an identifier.

20. Declaration Statement

<declare_stmt>::= <var_type><space><identifier>

| <var_type><space><assign_stmt>

This non-terminal states that a declare statement can be either a variable type followed by a space followed by an identifier or variable type followed by a space followed by an assign statement

21. Input Output

<input_stmt> ::= **input**<LP><identifier><RP>

This non-terminal is defined as input reserved word followed by identifier between two parentheses. It will be used to get an input from the standard input device(i.e. keyboard). Inputbody is defined as a variable therefore input given will be assigned to the variable automatically and in this way it will be easy to keep track of data type of the input.

<input_from_connection> ::= <identifier><dot>**input_from_connection**<LP><identifier><RP>

This non-terminal is defined as a connection variable followed by a dot, input_from_conection reserved word and inputbody between two parentheses. The first identifier can be any variable and the data received from the connection is stored into this variable. The second identifier is a connection object (more details about it later).

<output_stmt> ::= **print**<LP><identifier><RP>

This non-terminal is defined as a “print” reserved word followed by an identifier between two parentheses. It will be used to transfer information to standard output device(i.e. screen). outbody is defined later.

<output_to_connection> ::= <identifier><dot>**output_to_connection**<LP><identifier><RP>

This non-terminal is defined as output_to_connection followed by left parenthesis, URL, comma, identifier and right parenthesis. This allows sending data to a connection, for example a command line connection can take linux commands using this function. It is called on a connection object.

<read_from_sensor> ::= **read_from_sensor**<LP><identifier><comma><identifier><RP>

This non-terminal is defined as read_from_sensor followed by left parenthesis then the identifier that refers to sensor name then a comma then the identifier refers to input body that we get from the sensor. This allows it to read values from different sensors according to the sensor type. For example if the sensor name provided was a name of a temperature sensor then it will return in input body the temperature, but in this case <inbody> should be a float variable because temp sensor returns float values.

<time_from_timer> ::= **time_from_timer**<LP><identifier><RP>

This non-terminal is defined as: “time_from_timer” reserved keyword followed by an identifier that is an integer variable between parentheses. It takes the timestamp from the timer and puts it into the identifier that is an integer variable <int_var>. The timestamps taken from the timer is the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970, not counting leap seconds.

<URL> ::= <string>

This non-terminal defines URL as string.

22. Creating a Connection Object and All Related Statements

<connect_obj_creation> ::=
Connection_Class<space><connect_var><assign_op>
new<space>**Connection_Constructor**<LP><RP>

This non-terminal is defined as: Connection class name followed by a space followed by the connection variable name followed by the assignment operator followed by the new keyword followed by a space followed by the Connection() constructor. This allows users to initialize connection objects. The connection class has multiple useful functions that can be used to manipulate connections and do some basic operations. IsOT's approach for connection handling is by creating a connection object and data is stored in parameters. The data can be accessed using getter functions.

<connect_stmt> ::= <identifier><dot>**connect**<LP><URL><RP>

This non-terminal is defined as: identifier refers to a connection variable followed by a dot followed by the connect keyword followed by a left parenthesis followed by the URL followed by a right parenthesis. This is one of the Connection class functions that allows to establish a connection by taking a URL as the address.

<get_connect_status> ::=
<identifier><dot>**getStatus**<LP><identifier><RP>

This non-terminal is defined as: an identifier refers to connection variable followed by a dot followed by getStatus followed by a left parenthesis followed by an identifier refers to status variable followed by a right parenthesis. This Connection function puts the connection status into a status variable. status is a data type in IsOT, making it easier and simpler to deal with connections as this an IoT oriented programming language. status type can have specified values predefined by the language and are reserved keywords.

<get_connect_protocol> ::=
<identifier><dot>**getProtocol**<LP><identifier><RP>

This non-terminal is defined as: an identifier refers to connection variable followed by a dot followed by getProtocol followed by a left parenthesis followed by an identifier refers to protocol variable followed by a right parenthesis. Exactly as status data type and variables, protocol variable has predefined values and is useful for connections used for IoT devices.

<get_connect_URL> ::= <identifier><dot>**getURL**<LP><identifier><RP>

This non-terminal is defined as: an identifier refers to connection variable followed by a dot followed by getURL followed by a left parenthesis followed by a identifier refers string variable followed by a right parenthesis. This function puts the URL this connection is connected to to a string variable provided between parenthesis.

<status_value> ::= **connected** | **connecting** | **disconnected** | **host_not_found** | **connection_timeout**

Status value is defined as these reserved values.

<protocol_value> ::= **http** | **https** | **tcp** | **ftp** | **tftp**

Protocol value is defined as these reserved values.

23. Switches

<turn_switch_on> ::= **turn_switch_on**<LP><switch_name><RP>

This non-terminal is defined as turn_switch_on reserved function name followed by a left parenthesis followed by the switch name followed by a right parenthesis. This function is used to turn the switch on whatever its current position is.

<turn_switch_off> ::= **turn_switch_off**<LP><switch_name><RP>

This non-terminal is defined as turn_switch_off reserved function name followed by a left parenthesis followed by the switch name followed by a right parenthesis. This function is used to turn the switch off whatever its current position is.

<toggle_switch> ::= **toggle_switch**<LP><switch_name><RP>

This non-terminal is defined as toggle_switch reserved function name followed by a left parenthesis followed by the switch name followed by a right parenthesis. This function is used to change the switch to the position that is opposite to what it is on when calling the function (if switch is on it becomes off and if off it becomes on after this function call).

<get_switch_state> ::= **get_switch_state**<LP><switch_name><comma><identifier><RP>

This non-terminal is defined as get_switch_name reserved function name followed by a left parenthesis followed by the switch name followed by a comma followed by an identifier which should be boolean type followed by a right parenthesis. This function is used to get the state of the switch (on/off).

<switch_name>::= **switch_0 | switch_1** | **switch_2 | switch_3** | **switch_4 | switch_5** | **switch_6 | switch_7** | **switch_8 | switch_9**

Switch name variable is defined as an the name of one of the 10 switches. The switch names are reserved keywords.

24. Return Statement Explanation

In IsOT, all functions defined by the programmer or reserved functions use the pass by reference mechanism, which means that any value that need to be returned should be passed through a variable in the parameters list. This feature helps unexperienced programmers to use the language without having to deal with return types for the functions and all related return errors. The only use of the **return** reserved keyword is to stop the function's execution in the case of recursive functions for example.

25. Indentation in IsOT

IsOT does not support indentation since it might cause some readability problems for beginner programmers and confusion in the case of wrong indentation. Though we encourage the use of new lines as much as needed to increase readability of the code in the case of if statements or function definitions for example.

Non-Trivial Tokens

start Starts the program in our language.
end Finishes the program in our language.
if Used for the if statements in our language.
else Used for the else statements in our language.
while Used in the while statements in our language.
for Used in the for statements in our language.
break Used for break statements in our language.
return Used for return statements in our language.
continue Used for continue statements in our language.
print Used while printing outputs.
input Used while getting inputs.
input_from_connection Used as function name which is used while getting input.
output_to_connection Used as function name which is used while getting output.
Read_from_sensor Takes information from the sensors
Time_from_timer It takes the timestamp from the timer
int We have int datatype in our language.
bool We have bool datatype in our language.
float We have float datatype in our language.
string We have string datatype in our language.
char We have char datatype in our language.
status We have status datatype in our language.
protocol We have protocol datatype in our language.
func We have func reserved word in our language to define a function.
true It is the boolean value true in our language.

false It ıs the boolean value false in our language.
yes It is the same boolean value as true in our language.
no It is the same boolean value as false in our language.
connection_Class The connection class has multiple useful functions that can be used to manipulate connections and do some basic operations.
connection_Constructor Constructor to initialize the connection class.
connect This reserved word is used for connecting to a URL.
getStatus This reserved word is used to get status of the connection.
getProtocol This reserved word is used to get protocol of the connection.
getURL This reserved word is used to get URL of the connection.
connected , connecting , disconnected , host_not_found , connection_timeout These reserved words are used to specify the status of the connection.
http , https , tcp , ftp , tftp These reserved words are used to specify the protocol of the connection.
on It is the same boolean value as true in our language.
off It is the same boolean value as false in our language.
turn_switch_on It is reserved for turning on the switches.
turn_switch_off It is reserved for turning off the switches.
toggle_switch It is reserved for changing the value of the switch on to off or off to on.
get_switch_state In IsOT, this reserved word is used to get the state info of a switch.
switch0-9 In IsOT, we have ten switches each named by switch and a number up to 10.

Readability, Writability, Reliability

IsOT language adapts common naming conventions that exist in general-purpose programming languages such as Java, C++, Python with enhancements in IoT specific functions, datatypes and other required mechanisms. It has simple and easy-to-use and easy-to-understand syntax. IsOT language has mechanisms that don't exist in common other languages. It has no ambiguity. IsOT doesn't have ambiguous definitions. Programmers can use IsOT since it is comprehensive and it makes complex programs, subprograms simple. Security is an important point in IoT development as people entrust valuable things to robots. Since IsOT makes jobs easier and simpler without any ambiguity, IsOT would make programmers job a lot easier.