CS-315 Project 2 - Section 2

Fall 2022

Team 19

Programming Language: LanguageOfThings

Team Members

Emirhan Büyükkonuklu 22003049

Emre Tarakçı 21902607

Barış Yıldırım 22003175

1)Complete BNF Description of Language

Statements:

```
<program> ::= <stmts>

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

<stmt> ::=  <if_stmt>

       | <non_if_stmt>

<non_if_stmt> ::= <assign_stmt>

       | <iteration_stmt>

       | <function_def_stmt>

       | <function_call_stmt>

       | <return_stmt>

       | <var_declaration_stmt>

       | <output_stmt>

       | <input_stmt>

<var_declaration> ::= <pure_type> <identifier>

      | <sensor_type> <sensor_identifier> <assign_op> <string>

      | <switch_type> <switch_identifier> <assign_op <string>

<var_declaration_stmt> ::= <var_declaration>

<left_hand_side> ::= <var_declaration> | <identifier>


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

        | <left_hand_side> <assign_op> <int>

        | <left_hand_side> <assign_op> <bool>

        | <left_hand_side> <assign_op> <string>

<output_stmt> ::= <print_identifier> <left_parenthesis> <expression>
<right_parenthesis>

            | <print_identifier> <left_parenthesis> <identifier>
<right_parenthesis>

                  | <print_identifier> <left_parenthesis> <constant>
<right_parenthesis>
```

<input_stmt> ::= <underscore> <scan_identifier> <left_parenthesis>
<identifier> <right_parenthesis>

| <underscore> <scan_identifier> <left_parenthesis> <constant>
<right_parenthesis>

Identifiers:

<sentence> ::= <symbol> | <sentence> <symbol>

<word> ::= <letter> | <word> <letter>

<identifier_beginning_symbol> ::= <minus> | <under_score> | <letter>

<identifier_symbol> ::= <identifier_beginning_symbol> | <digit>

<pure_identifier> ::= <identifier_beginning_symbol>

| <identifier_symbol> <identifier>

<sensor_identifier> ::= <dollar_sign> <pure_identifier>

<switch_identifier> ::= <hashtag> <pure_identifier>

<identifier> ::= <pure_identifier> | <sensor_identifier> |
<switch_identifier>

<number> ::= <non_zero_digit> | <number> <digit>

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

Variables:

<pure_type> ::= <int_type>

| <string_type>

| <bool_type>

| <double_type>

<type> ::= <pure_type>

| <sensor_type>

| <switch_type>

<constant> ::= <string> | <int> | <double> | <bool_identifier>

<string> ::= <quote> <sentence> <quote> | <single_quote> <sentence>
<single_quote>

<int> ::= <sign> <number> | <number>

<double> ::= <sign> <number> <dot> <digits> | <number> <dot> <digits>

Arithmetic Operators:

<arithmetic_expression> ::= <arithmetic_expression> <plus> <term>

| <arithmetic_expression> <minus> <term>

| <term>

<term> ::= <term> <asterisk> <factor>

| <term> <slash> <factor>

| <term> <modulo> <factor>

| <factor>

<factor> ::= <left_parenthesis> <arithmetic_expression> <right_parenthesis>

| <number>

| <identifier>

| <function_call_stmt>

Expressions:

<expression> ::= <arithmetic_expression>

| <comparison_expression>

| <bool_expression>

<for_expression> ::= <var_declaration_stmt> <bool_expression> <assign_stmt>

<bool> ::= <left_parenthesis> | <bool_expression> | <right_parenthesis>

| <bool_identifier>

<bool_identifier> ::= <bool_true> | <bool_false>

<bool_operator> ::= <and>

| <or>

| <equals_op>

| <not_equal_op>

<bool_expression> ::= <bool_expression> <or> <and_expression>

| <and_expression>

<and_expression> ::= <and_expression> <and> <bool>

|<bool>

<comparison_operator> ::= <equals_op>

| <greater_than_op>

| <greater_than_or_equals_op>

| <less_than_op>

| <less_than_or_equals_op>

| <not_equal_op>

<comparison_expression> ::= <comparison_expression> <comparison_operator>
<arithmetic_expression>

| <arithmetic_expression>

| <right_parenthesis> <comparison_expression> <left_parenthesis>

Function Calls:

<var_declaration_list> ::= <var_declaration>

| <var_declaration_list> <comma> <var_declaration>

<function_def_input> ::= <left_parenthesis> <right_parenthesis>

| <left_parenthesis> <var_declaration_list> <right_parenthesis>

<function_def_stmt> ::= <type> <underscore> <identifier> <function_def_input>
<left_brace> <stmts> <right_brace>

<identifier_list> ::= <identifier> | <identifier_list> <comma> <identifier>

<function_call_stmt> ::= <underscore> <identifier> <left_parenthesis>
<right_parenthesis>

| <identifier> <left_parenthesis> <identifier_list>
<right_parenthesis>

<return_stmt> ::= <return_identifier> <identifier> | <return_identifier>
<constant>

Conditionals:

<if_stmt> ::= <matched> | <unmatched>

<matched> ::= <if_identifier> <left_parenthesis> <bool_expression>
<right_parenthesis>

<matched> <else_identifier> <matched>

| <left_brace> <stmts> <right_brace>

<unmatched> ::= <if_identifier> <left_parenthesis> <bool_expression>
<right_parenthesis>

| <if_identifier> <left_parenthesis> <bool_expression>
<right_parenthesis>

<matched> <else_identifier> <unmatched>

Loops:

<iteration_stmt> ::= <while_stmt> | <for_stmt>

<while_stmt> ::= <while_identifier> <left_parenthesis> <bool_expression>
<right_parenthesis> <left_brace> <stmts> <right_brace>

<for_stmt> ::= <for_identifier> <left_parenthesis> <for_expression>
<right_parenthesis> <left_brace> <stmts> <right_brace>

Primitive Functions:

<read_sensor> ::= <underscore> <read_sensor_identifier> <left_parenthesis>
<sensor_identifier> <right_parenthesis>

<post_data> ::= <underscore> <post_data_identifier> <left_parenthesis>
<string> <right_parenthesis>

<get_data> ::= <underscore> <get_data_identifier> <left_parenthesis> <string>
<right_parenthesis>

<connect> ::= <underscore> <connect_identifier> <left_paranthesis> <string>
<right_parenthesis>

<set_switch_on> ::= <underscore> <set_switch_on_identifier>
<left_parenthesis> <switch_identifier> <right_parenthesis>

<set_switch_off> ::= <underscore> <set_switch_off_identifier>
<left_parenthesis> <switch_identifier> <right_parenthesis>

<toggle_switch> ::= <underscore> <toggle_switch_identifier>
<left_parenthesis> <switch_identifier> <right_parenthesis>

Symbol and Terminals:

<symbol> ::= <digit>

| <letter>

| <left_parenthesis>

| <right_parenthesis>

| <left_square>

| <right_square>

| <left_brace>

| <right_brace>

| <exclamation>

| <quote>

```
                | <single_quote>

                | <hashtag>

                | <dot>

                | <equals_op>

                | <plus>

                | <minus>

                | <slash>

                | <backslash>

                | <asterisk>

                | <question_mark>

                | <under_score>

                | <semi_colon>

                | <colon>

                | <comma>

                | <and>

                | <less_than_op>

                | <greater_than_op>

                | <modulo>

<left_parenthesis> ::= (

<right_parenthesis> ::= )

<left_square> ::= [

<right_square> ::= ]

<relational_operators> ::= <less_than_op> | <greater_than_op> |
<less_than_or_equals_op> | <greater_than_or_equals_op> | <equals_op> |
<not_equal_op> | <and> | <or>

<bool_true> ::= true

<bool_false> ::= false

<print_identifier> ::= print

<scan_identifier> ::= scan

<while_identifier> ::= while

<for_identifier> ::= for
```

<if_identifier> ::= if

<else_identifier> ::= else

<read_sensor_identifier> ::= read

<post_data_identifier> ::= post

<get_data_identifier> ::= get

<connect_identifier> ::= connect

<set_switch_on_identifier> ::= setSwitchOn

<set_switch_off_identifier> ::= setSwitchOff

<toggle_switch_identifier> ::= toggleSwitch

<return_identifier> ::= return

<double_type> ::= double

<int_type> ::= int

<string_type> ::= string

<bool_type> ::= bool

<sensor_type> ::= sensor

<switch_type> ::= switch

<less_than_op> ::= <

<greater_than_op> ::= >

<less_than_or_equals_op> ::= <=

<greater_than_or_equals_op> ::= >=

<left_brace> ::= {

<right_brace> ::= }

<exclamation> ::= !

<and> ::= &&

<or> ::= | |

<quote> ::= "

<single_quote> ::= '

<hashtag> ::= #

<dot> ::= .

<equals_op> ::= ==

```
<not_equal_op> ::= !=

<assign_op> ::= =

<plus> ::= +

<minus> ::= -

<slash> ::= /

<backslash> ::= \

<asterisk> ::= *

<question_mark> ::= ?

<under_score> ::= _

<semi_colon> ::= ;

<colon> ::= :

<comma> ::= ,

<letter> ::= [a-zA-Z]

<lowercase> ::= [a-z]

<uppercase> ::= [A-Z]

<digit> ::= [0-9]

<non_zero_digit> ::= [1-9]

<sign> ::= <plus> | <minus>

<dollar_sign> ::= $

<tilde> ::= ~

<modulo> ::= %

<or_op> ::= ||

<and_op> ::= &&
```

2)Explanations of Language Constructs

2.1)Literals

<program> : First non-terminal of the language. It consists of <stmts> which other non-terminals are directly connected to this non-terminal.

<stmts> : This non-terminal is defined as a collection of <stmt> recursively.

<stmt> : Branches to 9 different <stmt> types.

<non_if_stmt> : This <stmt> type encapsulates all of the statements other than if else statements. Branches to 8 different <stmt> types.

<assign_stms> : Represents assignment operations of data types and identifiers.

<iteration_stmt> : Represents loops operations. Branches to 2 different statements. <while_stmt> and <for_stmt>.

<function_def_stmt> : Contains the necessary elements to define a function.

<function_call_stmt> : Contains the necessary elements to call a function.

<return_stmt> : Represents the return statements in the language.

<var_declaration_stmt> : Represents variable declarations of sensor, switch, int, double, string and bool.

<output_stmt> : Represents output statement. Accepts a reserved keyword called print.

<input_stmt> : Represents input statement. Accepts a reserved keyword called scan.

<while_stmt> : Represents while loops.

<for_stmt> : Represents for loops.

<function_def_input> : Defines what goes inside function inbetween parentheses. Either function contains no parameters or a parameter list which contains one or more parameter declarations.

<var_declaration_list> : Used to define function_def_input, var_declaration_list consists of 1 or more var_declarations.

<pure_type> : Consists only of bool, string, double and int.

<type> : Contains all of the data types in LanguageOfThings language. Sensor, switch, int, double, string and bool.

<comment> : This non-terminal is used to declare comments in the language.

<sentence> : Defined recursively, consists of collection of words. Defines string.

<word> : Defined recursively, consists of collection of letters.

<identifier> : This non-terminal represents identifiers such as sensor identifiers, sensor identifiers and some other identifiers which works similar with Java. This non-terminal defines how variable definitions should work.

<number> : Defined recursively, consists of concateneted digits. Cannot start with 0.

<digits> : Defined recursively, consists of concateneted digits. Can start with 0.

<constant> : This non-terminal represents variable constants int the language which are integer, double, string and boolean.

<expression> : Branches to 3 different expression non-terminals. <arithmetic_expression>, <comparison_expression> and <bool_expression>.

<for_expression> : Implicates the necessary statements which are written inside for loop.

<bool_expression> : This non-terminal is used to express boolean either in single or multiple ways. And with and_expression we can precede and over or, like we did in factor and term.

<comparison_operator> : Contains 6 operator terminals like greater_than.

<term> : This non-terminal helps <arithmetic_expression> non-terminal to recursively function and add multiple expression to a single <arithmetic_expression>.

<factor> : This non-terminal is used to resolve the issue of predency between multiply, division and addition, subtraction.

<if_stmt> : This is the basic if-else function of the language. It includes <matched> and <unmatched> non-terminal that clarifies the definition if-else statements.

<matched> : This non-terminal removes ambiguous situation that could be caused by if conditional statements in the language. It enables just matched if-else statements to be written after an if statement.

<unmatched> : Similar to <matched> non-terminal, this non-terminal also removes ambiguous situation. It defines what an unmatched if-else statement is.

<symbol> : Contains 28 terminals. Collection of symbols is <sentence> which is used in the definition of string

<read_sensor**>** : Definition of primitive function readSensor. Gets the contents of the sensor. Contains reserved keyword readSensor.

<post_data> : Definition of primitive function postData. Posts data to cloud. Contains reserved keyword postData.

<get_data> : Definition of primitive function getData. IoT device fetches the data from server. Contains reserved keyword getData.

<connect> : Definition of primitive function connect. Connects IoT device to server. Contains reserved keyword connect.

<set_switch_on> : Definition of primitive function setSwitchOn.Changes the switch content to on. Contains reserved keyword setSwitchOn.

<set_switch_off> : Definition of primitive function setSwitchOff. Changes the switch content to off. Contains reserved keyword setSwitchOff.

<toggle_switch> : Definition of primitive function toggleSwitch. Changes the switch content to off or on depending on its previous state. Contains reserved keyword toggleSwitch.

2.2)Terminals

(left parenthesis is used in method definitions and in for, while, if statements

) right parenthesis is used in method definitions and in for, while, if statements

< less than operator. It is used as a character and as a relational operator. e.g. var1 < var2 creates a relation between var1 and var2 which can be used as a condition of a conditional statement

> greater than operator. It is used as a character and as a relational operator. e.g. var1 > var2 creates a relation between var1 and var2 which can be used as a condition of a conditional statement.

>= greater than or equal operator. It is used as a character and as a relational operator. e.g. var1>= var2 creates a relation between var1 and var2 which can be used as a condition of a conditional statement

<= lessthan or equal operator. It is used as a character and as a relational operator. e.g. var1>= var2 creates a relation between var1 and var2 which can be used as a condition of a conditional statement

{ left brace is used in blocks

} right brace is used in blocks

&& is used to show and operation

| | is used to show or operation

" double quote is used to define strings

' single quote is used to define strings

+ plus sign is used in addition

- minus sign is used in subtraction

/ slash is used in divide operation

* asterisk is used in multiplication operation

, comma is used to split parameters

$ dollar sign is used to define sensors

% modulo is used in modulo operation

; semi colon is used to end a statement

! exclamation mark is used as a not operator

== is equal operator. It is used as a character and as a relational operator. e.g. var1 == var2 creates a relation between var1 and var2 which can be used as a condition of a conditional statement.

!= not equal operator. It is used as a character and as a relational operator. e.g. var1 != var2 creates a relation between var1 and var2 which can be used as a condition of a conditional statement

= equality sign is used as assign operator.

[left square

] right square

hashtag

. dot

\ backslash

: colon

~ tilde

_ under score

? question mark

3) Description of Nontrivial Tokens

3.1) Comments: LanguageOfThings provides only one type of comment unlike popular programming languages like Java and C++. There exists only single line comments and comment lines start with double slash //. We thought that multiple line comments were redundant and confuse programmers of LanguageOfThings, since they are relatively new to programming world, so there exists only one comment style. If they were to write comments

that are too long to fit into one line they can use another double slash and continue the comment on next line.

3.2) Identifiers: LanguageOfThings provides easily readable and writable identifier set. For primitive data types identifiers can start with - _ or any letter. For sensors, identifiers has to start with $ and for switches #. None of the identifiers can start with numbers. Otherwise 1+2 can be perceived as identifier but in reality looks more like arithmetic expression. We did not wanted to constrain the users by eliminating too much characters from naming an identifier. Only the ones that can potentially cause contradiction.

4) Reserved Words

For such a constricted language there exists a lot of reserved word which potentially negatively affects writability. Although all of the reserved words is reserved for a reason and cannot be avoidable.

for is used to to introduce loops.

while is used to introduce loops.

return is used in return statements of functions.

if introduces conditional statements. Conditional statements used to create logical flow control.

else if is used in conditional statements. It follows an if or another else if statement and its block.

else is used in conditional statements. It follows an if statement and its block. int is a data type which represents whole numbers.

int is a data type representing integer numbers.

double is a data type which represents floating point numbers.

string is a data type. It represents a sequence of characters.

bool is the representation of boolean literals.

sensor is the representation of sensor literals.

switch is the representation of switch literals.

print is method name for output statement.

scan is method name for input statement.

read is method name for primitive function read.

post is method name for primitive function post.

get is method name for primitive function get.

connect is the method for primitive function connect.

setSwitchOn is method name for primitive function setSwitchOn.

setSwitchOff is method name for primitive function setSwitchOff.

toggleSwitch is method name for primitive function toggleSwitch.

5) Evaluation

5.1) Readability

In order to maximize the readability, there are some conventions used in other famous programming languages such as using ";" at the end of each statement or using "//" to comment out any line which are used in LanguageOfThings as well. Also declaring variables is relatively easy compared to other languages since only identifiers are required for variable declaration. Notions used in arithmetic expressions are not different than casual mathematical expressions. We used "$" and "#" symbols to differentiate sensors and switch from other identifiers. That allows the programmer to easily distinguish if they are working with a sensor, a switch or another type of variable. We also defined booleans with @ since booleans in this language are vital.

5.2) Writability

Complicated variable declerations, arithmetic expressions and sensor monitoring are avoided in order to maximize writability of the language. Creating arithmetic expressions in LanguageOfThings have nothing different than creating arithmetic expressions in mathematics. Also utilizing sensors and reading values of them is really simple since there is only one method that takes sensor type as parameter and directly returns the value of sensor.

5.3) Reliability

In LanguageOfThings increased utilization of primitive variable types and primitive function brings out the reliability feature with it. In LanguageOfThings in order to remove ambiguity, matched and unmatched statements used for conditional statements. Identifiers of primitive variable types secures wrong identification and initialization of variables that increases the reliability. It has also been tested that there are no other BNF trees that could constructed other than the original one.