CS315 - PROGRAMMING LANGUAGES

PROJECT 1 REPORT

Oct. 14, 2022

Diamond IoT

Team 50:

Dağhan Ünal - 22002182 - Section 2

Eylül Badem - 22003079 - Section 3

Muhammed İkbal Doğan - 21702990 - Section 3

Table of Contents

BNF Design

1. Program Definition

<program>::= <main_function>|<main_function><functions>

<main_function>::= main{<bodies>}

<bodies> ::= <body> | <body> <bodies>

<body>::= <statements>|<comment>

2. Statements

<statements>::=<statement>;|<statement>;<statements>

<statement>::= <declaration> | <init_declaration> | <assignment> | <print> | <empty> |
<function_call> | <loops> | <stmt>

<declaration>::= <type><variable_name>

<init_declaration>::= <type> <assignment>

<assignment> ::= <integer_assign> | <string_assign> | <character_assign> |
<double_assign> | <boolean_assign> | <sensor_assign> |
<connection_assign> | <switch_assign>

<type>::= int | String | char | double | boolean | sensor

<variable>::= integerV | doubleV | characterV | stringV | booleanV

<variable_name>::= identifier

3. Assignments

<sensor_assign> ::= <variable_name> = doubleV

<switch_assign>::= <switches> = booleanV

<connection_assign> ::= <connection> <variable_name> = StringV

<integer_assign> ::= <variable_name> = integerV

<string_assign> ::= <variable_name> = StringV

<double_assign> ::= <variable_name> = doubleV

<character_assign> ::= <variable_name> = characterV

```
<boolean_assign> ::= <variable_name> = booleanV |
                     <variable_name> = <boolean_operations> |
                     <variable_name>  = <comparison>
```

<arithmetic_assign> ::= <variable_name> = <arithmetic_operation>

4. Loops

<loops> ::= <for_loop> | <while_loop>

```
<for_loop> ::= for ( <init_declaration> ; <comparison> ; <assignment>)

           {statements}
```

```
<while_loop> ::= while(<comparison>) {<statements>} | while(booleanV) {<statements>} |
             while(<variable_name>) {<statements>} | while (<boolean_op>)
             {<statements>}
```

5. Conditions

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

```
<matched>::= if (<comparison>) <stmt> | if (<booleanV>) <stmt> | if (<boolean_op>) <stmt>
             | {<statements>}
```

```
<unmatched> ::= if (<comparison>) <matched> else <unmatched> |
                if (<booleanV>) <matched> else <unmatched> |
                if (<boolean_op>) <matched> else <unmatched> |
                if (<comparison>) <matched> else {<statements>} |
                     if (<booleanV>) <matched> else {<statements>} |
                     if (<boolean_op>) <matched> else {<statements>}
```

6. Functions

```
<function_protoypes> ::= <function_prototype><function_prototypes> | <function_prototype>

<function_prototype> ::= <int_function> | <string_function> | <double_function> |
                         <void_function>

<int_function> ::= function int <variable_name> (<parameters>) {<statements>
                         return   integerV ; }

<string_function> ::= function string <variable_name> (<parameters>) {<statements>
                         return   stringV ; }

<double_function> ::= function double <variable_name> (<parameters>) {<statements>
                         return   doubleV ; }

<void_function> ::= function int <variable_name> (<parameters>) {<statements>}


<parameters> :: <parameter>,<parameters> | <parameter> | <empty>

<parameter> ::= <empty> | <variable> | <variable_name>| <type> <variable_name>

<function_call> ::=  reach <varible_name> (<parameters>)

<primitive_functions> ::= CONNECT_INTERNET(String <variable_name>, String
                        <variable_name>) | CONNECT_TO(connection <variable_name>) |
                        SEND_INT(connection <variable_name>, int) |
                        RECEIVE_INT(connection <variable_name>, int <variable_name>) |
                        ELAPSED_TIME(int <variable_name>)| GET_TIME() |
                        READ_TEMPERATURE(sensor <variable_name>) |
                        READ_HUMIDITY(sensor <variable_name>) |
                        READ_AIRPRESSURE(sensor <variable_name>) |
                        READ_AIRQUALITY(sensor <variable_name>) |
                        READ_LIGHT(sensor <variable_name>) |
                        READ_SOUNDLEVEL(sensor <variable_name>, int <variable_name>
```

7. Operations

```
<arithmetic_operation> ::= <term> + <arithmetic_operation> |
                             <term> - <arithmetic_operation> | <term>

<term> ::= <term> * <variable> | <term> / <variable> | <variable>


<boolean_op> ::= <variable_name> <boolean_chars> <variable_name>
```

```
<boolean_chars> ::= && | ||


<comparison> ::= <variable_name> <comparison_chars> <variable> |
                 <variable> <comparison_chars> <variable_name> |
                 <variable_name> <comparison_chars><variable_name>|

                 <variable> <comparison_chars> <variable>

<comparison_chars> ::= > | < | >= | <= | != | ==

<print> ::= print ( stringV ) | print ( <variable_name>)

<endline> ::= \n

<empty> ::= null

<switches> ::= switch1 | switch2 | switch3 | switch4| switch5 | switch6 | switch7 |
               switch8 | switch9 | switch10

<connection> ::= connection

<comments> ::= comment
```

Grammar Explanation

<program>::= <main_function>|<main_function><functions>

A program can either consist of a main function, or other defined functions can come after the main function as well.

<main_function>::= main{<bodies>}

A main function consists of bodies. The word “main” indicates main function and bodies are written in curly brackets.

<bodies> ::= <body> | <body> <bodies>

Bodies can consist of one or more bodies.

<body>::= <statements>|<comment>

A body can either consist of statements or simply be a comment.

<statements>::=<statement>;|<statement>;<statements>

Like bodies, statements can consist of one or more statements. A semicolon is needed to end a statement.

<statement>::= <declaration> | <init_declaration> | <assignment> | <print> | <empty> | <function_call> | <loops> | <stmt>

A single statement can be a variable declaration, an initialization, an assignment operation, a print operation, an empty function, a function call, a loop or a condition. Empty functions are called whenever an empty space (e.g. []) is defined, and simply accepts it as a null value. ‘stmt’ values are used when regulating with if-else conditions.

<declaration>::= <type><variable_name>

A type must be specified when declaring variables. Variable names can only consist of digits, letters and underscore (_).

Ex: int x;

<init_declaration>::= <type> <assignment>

It is possible to both declare and assign a value in a single line.

Ex: int x = 4;

<assignment> ::= <integer_assign> | <string_assign> | <character_assign> | <double_assign> | <boolean_assign> | <sensor_assign> | <switch_assign> |<connection_assign>

It is possible to assign sensors, switches, connections, integers, strings, characters, doubles and booleans.

<type>::= int | String | char | double | boolean | sensor

Variable and function types can be integer, string, character, double or boolean.

<variable>::= integerV | doubleV | characterV | stringV | booleanV

A variable will hold a value for each of the types, value is represented by the last letter V.

<variable_name>::= identifier

A variable name will always start with a letter and can continue with any alphanumeric.

<sensor_assign>::= <variable_name> = doubleV

A sensor can be assigned with a given variable name with a double value.

<switch_assign>::= <switches> = booleanV

A switch can be assigned with a given variable name with a boolean value (can be on or off).

<connection_assign> ::= <connection> <variable_name> = connect (
String <variable_name>)

A connection can be assigned with the connect functionality.

<integer_assign> ::= <variable_name> = integerV

Users can assign an integer to an integer variable created.

Ex: int x = 17;

<string_assign> ::= <variable_name> = StringV

Users can assign a string to a string variable created.

Ex: String wifipass = "12345";

<double_assign> ::= <variable_name> = doubleV

Users can assign a double to a double variable created.

<character_assign> ::= <variable_name> = characterV

Users can assign a char to a char variable created.

<boolean_assign> ::= <variable_name> = booleanV |
<variable_name> = <boolean_operations> |
<variable_name> = <comparison>

Used to assign a boolean value to a boolean variable, to compare a variable name, and to operate boolean operations.

<arithmetic_assign> ::= <variable_name> = <arithmetic_operation>

Used to assign a variable name to an arithmetic operation.

<loops> ::= <for_loop> | <while_loop>

Defined loops are limited with for and while loops.

<for_loop> ::= for (<init_declaration> ; <comparison> ; <assignment>) {statements}

A for loop should be indicated with the key word “for”. A variable should be initialized, a comparison operation should be defined and an assignment operation should be made in order to complete the for loop definition.

Ex:
```
for(int a = 0; a <= x; a = a + 1)

{ y = y +2;

print: y; }
```

<while_loop> ::= while(<comparison>) {<statements>} | while(booleanV) {<statements>} | while(<variable_name>) {<statements>} | while (<boolean_op>) {<statements>}

A while loop should be indicated with the key word “while”. A condition must be specified before the statement executes. Statements should be written between curly brackets.

Ex:
```
while(exampleV < 50 && y != 150)

{ y = y + 1; }
```

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

Starting point for conditional statements.

<matched>::= if (<comparison>) <stmt> | if (<booleanV>) <stmt> | if (<boolean_op>) <stmt> | {<statements>}

Matched part is for continuation of simple if statements.

<unmatched> ::= if (<comparison>) <matched> else <unmatched> | if (<booleanV>) <matched> else <unmatched> | if (<boolean_op>) <matched> else <unmatched> | if (<comparison>) <matched> else {<statements>} | if (<booleanV>) <matched> else {<statements>} | if (<boolean_op>) <matched> else {<statements>}

Unmatched part is for continuation of if / not if (else) statements

<function_prototypes> ::= <function_prototype><function_prototypes> | <function_prototype>

Function prototypes include one or more functions, right hand recursively.

<function_prototype> ::= <int_function> | <string_function> | <double_function> | <void_function>

Function types can be int, string, double and void.

<int_function> ::= function int <variable_name> (<parameters>) {<statements> return integerV ; }

Function that returns an integer type by definition.

<string_function> ::= function string <variable_name> (<parameters>) {<statements> return stringV ; }

Function that returns a string type by definition.

<double_function> ::= function double <variable_name> (<parameters>) {<statements> return doubleV ; }

Function that returns a double type by definition.

<void_function> ::= function int <variable_name> (<parameters>) {<statements>}

Function without a return type.

<parameters> :: <parameter>,<parameters> | <parameter> | <empty>

There can be zero to infinite amount of parameters a function can take.

<parameter> ::= <empty> | <variable> | <variable_name>| <type>

A parameter can be a variable name that consists of a variable, a direct variable, a type or nothing.

<variable_name>

Variable name is a user defined name to hold a given variable type.

<function_call> ::= reach <varible_name> (<parameters>)

The keyword “reach” must be used to call a function.Although this is not a necessary functionality, it is required to see if a function is called through lex tokens.

**<primitive_functions> ::= CONNECT_INTERNET(String <variable_name>,
String <variable_name>) |
CONNECT_TO(connection <variable_name>) |
SEND_INT(connection <variable_name>, int) |
RECEIVE_INT(connection <variable_name>, int
<variable_name>) |
ELAPSED_TIME(int <variable_name>) |
GET_TIME() |
READ_TEMPERATURE(sensor <variable_name>) |
READ_HUMIDITY(sensor <variable_name>) |
READ_AIRPURESSURE(sensor <variable_name>) |
READ_AIRQUALITY(sensor <variable_name>) |
READ_LIGHT(sensor <variable_name>) |**

READ_SOUNDLEVEL(sensor <variable_name>, int <variable_name>)

CONNECT_INTERNET: primitive function to connect to an internet

CONNECT_TO: primitive function to connect to a given connection

SEND_INT: primitive function to send an integer to a given connection

RECEIVE_INT: Primitive function to receive an integer from a given connection.

ELAPSED_TIME: Primitive function to get the time elapsed from January 1, 1970 (UTC).

GET_TIME: Returns the current time.

READ_TEMPERTURE: Reads temperature from the given temperature sensor.

READ_HUMIDITY: Reads humidity from the given humidity sensor.

READ_AIRPRESSURE: Reads air pressure from the given air pressure sensor.

READ_AIRQUALITY: Reads air quality from the given air quality sensor.

READ_LIGHT: Reads light from the given light sensor.

<arithmetic_operation> ::= <term> + <arithmetic_operation> |
<term> - <arithmetic_operation> | <term>

<term> ::= <term> * <variable> | <term> / <variable> | <variable>

Executable arithmetic operations are limited to addition, subtraction, multiplication and division.This specific tree is to create priority for multiplication and division.

<boolean_op> ::= <variable_name> <boolean_chars> <variable_name>

Boolean operation is simply a comparison operation between variables.

<boolean_chars> ::= && | ||

These boolean operations can be used to extend logic operations in comparison statements.

<comparison> ::= <variable_name> <comparison_chars> <variable> | <variable> <comparison_chars> <variable_name> | <variable_name><comparison_chars><variable_name> | <variable> <comparison_chars> <variable>

Comparison can be between declared variables or direct types through comparison chars.

<comparison_chars> ::= > | < | >= | <= | != | ==

These comparison chars can be used to compare variables.

<print> ::= print (stringV) | print (<variable_name>)

Indicates print statement.

Ex: print: "working";

<endline> ::= \n

Indicates new line.

<empty> ::= null

Indicates empty values.

<switches> ::= switch1 | switch2 | switch3 | switch4| switch5 | switch6 | switch7 | switch8 | switch9 | switch10

There are 10 switches that control actuators.

<connection> ::= connection

Connection terminal that defines a connection to a given URL.

1. Terminals

==>	: Terminal for input operation
<==	: Terminal for output operation
>=	: Terminal for greater or equal to operator
<=	: Terminal for lesser or equal to operator
==	: Terminal for equality operator
=	: Terminal for assignment operator
;	: Terminal for ending statement
+	: Terminal for plus operator
-	: Terminal for minus operator
*	: Terminal for multiplying operator
/	: Terminal for division operator
>	: Terminal for greater operator
<	: Terminal for lesser operator
!=	: Terminal for not equals operator
{	: Terminal for left bracket
}	: Terminal for right bracket
(	: Terminal for left parenthesis
)	: Terminal for right parenthesis
[	: Terminal for left square bracket
]	: Terminal for right square bracket
_	: Terminal for underscore
#	: Terminal for hashtag used in comments
.	: Terminal for dot
&&	: Terminal for AND operation
||	: Terminal for OR operation

Nontrivial Tokens

1. Literals

Integers	: Common integer structure is kept to increase writability and readability.
Doubles	: Common double structure is kept to increase writability and readability.
Strings	: Common string structure is kept to increase writability and readability.
Boolean	: Boolean can have two values, true or false.
Characters	: Only letters are accepted as characters (both lowercase and uppercase) to keep the language writable.
Void	: Void is only used in function declarations.

2. Reserved Words

We chose to use reserved words because they would provide convenience in terms of readability and writability. We have kept most of the reserved words similar to common programming languages, except for a few extremes such as connect (connects to the given URL String), sensor (sensor variable), connection (URL variable), reach (for function calls) and c (function).

String	: Token for declaring string values
integer	: Token for declaring integer values
doubles	: Token for declaring double values
boolean	: Token for declaring logical values
characters	: Token for declaring character values
void	: Token for declaring void function type
if	: Token for if statements
else	: Token for else statement of the corresponding if statement
return	: Token for returning a value from function
for	: Token for for statements
while	: Token for while statements
print	: Token for printing values
null	: Token for null value
connect	: Token for connecting to the URL String

reach	: Token for function caller
sensor	: Token for declaring sensors
connection	: Token for declaring connections

3. Identifiers

In our language, we wanted to keep identifiers to common programming languages as similar as possible to increase both writability and readability. We didn’t set a new rule for identifiers, variables can start with lowercase letters as usual and continue as desired.

4. Comments

In our language, we chose to use # at the beginning for both single-line and multi-line comments and # again at the end. We thought it would support readability and writability as it reminds of the concept of notes and supports multi-line comments.

Evaluation

1. Writability

In order to increase the writability, we tried not to stray too far from English and the syntax of commonly used languages such as Java, C++ and Python.

To give examples; semicolons at the end of statements were inspired by Java, using hashtags for commenting was inspired by Python and we tried to keep the common structure as close to these languages as possible.

If we had to make a self-criticism about writability, it would be on the key word “reach”. The word choice could have been much more obvious and easier, but we tried not to give up on originality.

2. Readability

While trying to keep the writability feature high, we tried to make our program so that it can be written by users using basic programming knowledge and English knowledge. As a result, the program is planned to be quite readable given the universality of the two and assuming they are known by all of the users.

3. Reliability

Since our language is only designed for and meant to be used in IoT devices, it does not include features like arrays, lists, and unnecessary loop types. Therefore, our language may not be reliable when used for another purpose, but it is believed to be reliable for IoT devices.