Bilkent University
CS 315 Project 1

Chain^2

Fall 2022-2023

Group members:

Gülin Çetinus	21902115	section 01
Gizem Gökçe Işık	21803541	section 01
Alper Bozkurt	21802766	section 03

Table of Contents

1. BNF description of the language

```
<program> ::= <stmt_list>

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

<stmt> ::= <assign_stmt>
          | <declare_stmt>
          | <if_stmt>
          | <loops>
          | <comment_stmt>
          | <return_stmt>

<assign_stmt> ::= <variable> ASSIGN_OP <expression> SEMICOLON

<declare_stmt> ::= <type> <variable> SEMICOLON
                  |<type> <variable> ASSIGN_OP <expression>

<variable> ::= IDENTIFIER

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

<matched> ::= IF LB <logic_expression> RB <if_stmt>
              | <stmt>

<unmatched> ::= IF LB <logic_expression> RB <if_stmt>
                | IF LB <logic_expression> RB <matched>
                  ELSE <unmatched>

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

<while_loop> ::= WHILE LB <logic_expr> RB BLOCK_BEGINS
                 <stmt> BLOCK_ENDS

<for_loop> ::= FOR LB <declare_stmt> SEMICOLON
               <logic_expr> SEMICOLON <assign_stmt> RB
               BLOCK_BEGINS <stmt> BLOCK_ENDS
```

```
<logic_expression> ::= <variable> <comparison_op> <variable>
                        | <logic_expression> <connection_op>
                         <logic_expression>
                        | <boolean_stmt>

<boolean_stmt> ::= BOOL

<comparison_op> ::=  GREAT | LESS | EQUAL | LESS_EQ | GREAT_EQ

<connection_op> ::= OR | AND



<expression> ::= <expression> PLUS_OP <term> SEMICOLON
                 | <expression> MINUS_OP <term> SEMICOLON
                 | <term> SEMICOLON

<term> ::=  <term> MULTIPLY <digit_list>
           |<term> DIVIDE <digit_list>
           |<term>

<comment_stmt> ::= #<STRING>#

<function> ::= <return_type> <function_name> LB
               <paramet> RB BLOCK_BEGINS <stmt>
               <return_stmt> BLOCK_ENDS
            |  <return_type> <function_name> LB
               <paramet> RB BLOCK_BEGINS <stmt>
               <empty> BLOCK_ENDS
            |  <return_type> <function_name> LB
               <empty> RB BLOCK_BEGINS <stmt>
               <return_stmt> BLOCK_ENDS
            |  <return_type> <function_name> LB
               <empty> RB BLOCK_BEGINS <stmt>
               <empty> BLOCK_ENDS

<return_type> ::= <type>

<return_stmt> ::= RETURN <empty> | RETURN <variable>

<type> ::= VOID_TYPE | INT_TYPE | CHAR_TYPE | DOUBLE_TYPE
         | STRING_TYPE | BOOL_TYPE | LONG_TYPE | TIMER

<digit_list> ::= <digit>
```

```
                | <digit_list><digit>

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

<string> ::= <letter>
           | <string><letter>

<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
     |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

<symbols> ::= $ | ! | % | [ | ] | { | } | * | ? | " | ' | < |
              > | ~ | + | - | , | . | : | ; | _ | \ | / | & |
              = | (|) | ^ | #

<func_call> ::= <function_name> LB <paramet> RB
               |<function_name> LB <empty> RB

<function_name> ::= <variable>

<paramet> ::= <type> <variable>
               | <type> <variable> COMMA <param>
<empty> ::=

<read_sensor> ::= <sensor_temperature> | <sensor_humidity>
               | <sensor_pressure> | <sensor_air_quality>
               | <sensor_light> | <sensor_sound>
               | <sensor_timestamp>

<sensor_temperature> ::= SENSOR_TEMP

<sensor_humidity> ::= SENSOR_HUMID

<sensor_pressure> ::= SENSOR_PRESSURE

<sensor_air_quality> ::= SENSOR_AIR_QUAL

<sensor_light> ::= SENSOR_LIGHT

<sensor_sound> ::= SENSOR_SOUND_QUAL

<switch> ::= <boolean_expression>
```

```
<sensor_timestamp> ::= SENSOR_TIMESTAMP

<connect_internet> ::= connectInternet()

<disconnect_internet> ::= disconnectInternet()

<read_url> ::= READ_URL

<alphanumeric_string> ::= <letter> | <digit>
          |<alphanumeric_string><letter>
          |<alphanumeric_string><digit>

<url> ::= INTEGER.INTEGER.INTEGER.INTEGER
```

2. Explanation of the language construct

`<program>`
This is the starting non-terminal. This non-terminal includes "`<stmt_list>`".

`<stmt_list>`
This non-terminal forms the whole language. It is a recursive non-terminal. It consists of one <stmt> non-terminal or <stmt> <stmt_list> which makes that non-terminal recursive.

`<stmt>`
This non-terminal base non-terminal that includes all statements such as <assignment_stmt>, <declare_stmt>, <if_stmt>, <for_stmt>, <while_stmt>, <comment_stmt>.

`<assign_stmt>`
This non-terminal assigns the right hand side's value to the variable name which is on the left hand side. For example in Java when we created int count = 5; we are assigning 5 to count variable count.

`<declare_stmt>`
In the declaration statement we have two options. First option is to declare the variable without assignment operator(for example: int count). Second option is to declare the variable and also assign it a value(for example: int count = 5).

`<variable>`
Variable statements in our program is for identifiers which are constructed to name data types like integers, string etc. in our program.

<if_stmt>
In the if statement we distinguished matched and unmatched statements. Our motivation was to avoid ambiguity. The duty of this if statement is to check whether the condition is true or false, similar to other programming languages.

<matched>
This non-terminal matched statement is to check the left and right braces in the unconditional statements(if, else). If braces matched that means the if count and else count is equal.

<unmatched>
This non-terminal unmatched statement also checks the left and right braces in the unconditional statements. If braces are unmatched that means the if counts and else counts are not equal.

<loops>
This loop statement consists of while and for loops which has the same functionality compared to other programming languages.

<while_loop>
This while loop statement is similar to while loop from the other languages. It executes a logical expression and executes the statements inside the while loop until the logical expression is false.

<for_loop>
This for statement is also similar to for loops from other languages. It executes the declare statement(int i = 0), logical expression(i < 5), and also assignment operator(i = i+1) which directs to arithmetic expressions.

<empty>
Empty statement creates empty source code which makes more options.

<expression>
Expression statement in our program provides us to do some arithmetic operations such as addition, subtraction and directs to the term statement also.

<comparison_op>
This statement is used to compare two statements with comparison operators which are <, >, =, =<, >=.

<connection_op>
This statement is to identify “or” and “and” logical expressions. Since we connect two statements while using “or” and “and” statements’ name is connection_op.

<term>
Term statement is for multiply and divide operations. Term and expression statements are different for processing priority while doing mathematical expressions.

<comment_stmt>
Comment statement in our program is simply #.....#. Compare to C++(in C++ "//" used for writing comments), Chain^2 uses "#" sign to define comments.

<function> <function_name>
Function statement in our program works the same compared to Java. Chain^2 has also return type, function name, parameter declaration, function body and return variables. We have function names which are identifiers and aim to differentiate functions from each other.

<return_stmt>
Return mechanism is for returning from functions. While returning we use a special keyword "return" like Java, C++ etc. If return type is void, we can return without a type. Otherwise, we can say return and specify the return type afterwards.

<return_type>
Return types can be void, int, char, double, string, bool, long and also timer.

<type>
Types in Chain^2 are also the same as other programming languages. For example void, int, char, double, bool, short, long_int can be easily understandable for beginner programmers. Our motivation is to increase readability and also since most of the programming languages use the same data types, it makes Chain^2 more familiar to programmers.

<digit> <digit_list> <letter> <symbols> <string>
Digits, letters, symbols, strings are the same as other programming languages and also all of the characters, numbers and symbols can be found in ASCII.

<func_call>
Function call mechanism is similar to other languages. Function style also checks for left and right braces. Parameters of functions are optional that is why we create <empty> statements.

<paramet>
Parameters have three jobs. Firstly, it takes type(for example int) and variable(for example count). Second duty of parameter's is doing the same thing from the first one recursively.

<read_sensor>
Read sensor is also necessary to specify six sensors to read temperature, air humidity, air pressure, air quality, light, sound quality. Additionally, we added sensor timestamp to control timestamp.

`<read_temperature> <read_air_humidity> <read_air_pressure> <read_air_quality> <read_light> <read_sound_quality> <read_url>`

Read data functions are our reserved keyword function names such as the main method in Java. Since Chain^2's job is to build an IoT device with our functions, this functions' job is reading data from each sensor. This device should have several sensors for reading temperature, humidity, air pressure, air quality, light and sound level and also URL.

`<sensor_temperature> <sensor_humidity> <sensor_pressure> <sensor_air_quality> <sensor_light> <sensor_sound> <sensor_timestamp>`

Sensor functions are for controlling the necessary features with on/off control signals.

`<switch>`

Switches are connected to actuators, and they are set to on or off to control actuators. For on/off specification we used bool.

`<read_timestamp>`

Read timestamp method read timer from timestamp.

`<connect_internet> <disconnect_internet>`

Internet connection functions make our device connect to the internet and also disconnect from the internet. Since Chain^2's function names look like natural languages, this convenience makes our language more readable.

3. Definition of non-trivial tokens

Identifiers: Identifiers are constructed to name the variables, methods and class names. While constructing Chain^2 grammar we used simple rules for constructing identifiers. As an example in Java names of identifiers such as i, j, count should start with character, then adding numbers and symbols such as _ can be added later. We followed the same rule for increasing readability for programmers and non-programmers.

Comments: In Chain^2 we used two hashtag signs to define comments which can be shown as (#commentcomment#). Between two hashtag signs users can write strings and create comment statements. We used hashtag signs to avoid confusions compare to other symbols such as /, * etc.

Literals: In Chain^2 literals are similar to Java language. For example, boolean literals in our language are true and false which is similar to Java.

Reserved Words: Keywords we used are similar to Java. As an example, if, else, for, while, int, char, double, boolean, void, string, return, true, false are also used in Java as keywords. Different from Java, we used readTemperature(), readAirHumidity(), readAirPressure(), readAirQuality(), readLight(), readSoundQuality() as reserved words which are function names. While constructing these functions and deciding their names, our motivation was to be readable and comprehensible to non-programmers and beginner programmers. As an example when someone sees the readTemperature() function, he/she can understand that readTemperature() function's duty is to read the temperature and return it.

4. Language Criteria

Readability: Readability makes programming language grammar to be more understandable. In Chain^2 our motivation was to have more clear function names, data types etc. for programmers. The way we designed it is, using clear and simple names for functions. Also, since our function names are similar to natural language form, our reserved functions are also readable.

Writability: Using minimal number of instructions, functions and data types makes Chain^2 more writable for programmers. That is why while using Chain^2 doing hard jobs becomes easier with minimal number of instructions, functions etc.

Reliability: Since our programming language's job scope is smaller than Java, C++'s scope, we don't have that many error checking mechanisms. However, we have some mechanisms for getting correct outputs. As an example, we constructed <term> statements for processing priority while doing mathematical expressions.