CS315

2019 - 2020 Fall

Project 2 Report

Group 19

Sec 2

Yusuf Avcı 21702724

Çerağ Oğuztüzün 21704147

Mehmet Bora Kurucu 21703404

PAPER

CONTENTS

BNF OF PAPER

<program> → MAIN LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET
| <empty>

<stmt_list_empty> → <stmt_list> | <empty>

<stmt_list> → <stmt> | <stmt_list> <stmt>

<stmt> → <general_assign> SEMICOLON | <if_stmt> | <define> | <for_loop> | <while_loop> | <func_def> | <func_call> SEMICOLON | <print_stmt> SEMICOLON
| <switch_stmt> SEMICOLON | <connection> SEMICOLON
| <connection_send> SEMICOLON

<define> → <def_multiple> SEMICOLON

<def_multiple> → <def_one> | <def_one> COMMA <def_multiple> | <empty>

<def_one> → <def_string> | <def_int> | <def_double>

<def_string> →STRING_TYPE IDENTIFIER |STRING_TYPE IDENTIFIER ASSIGN_OP IDENTIFIER |STRING_TYPE IDENTIFIER ASSIGN_OP <string> |STRING_TYPE IDENTIFIER ASSIGN_OP <func_call>

<def_int> →INT_TYPE IDENTIFIER |INT_TYPE <int_assign>

<def_double> → DOUBLE_TYPE IDENTIFIER| DOUBLE_TYPE <all_numeric_assign>

<general_assign> → <all_numeric_assign> | IDENTIFIER ASSIGN_OP STRING

<all_numeric_assign> → IDENTIFIER ASSIGN_OP <expr>

<int_assign> → IDENTIFIER ASSIGN_OP <int_expr>

<func_def> → <func_int_def> | <func_double_def> | <func_string_def> | <func_void_def>

<func_string_def> →STRING_TYPE IDENTIFIER LEFT_PARANTHESIS <def_multiple> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RETURN <string> SEMICOLON RIGHT_CURLY_BRACKET
|STRING_TYPE IDENTIFIER LEFT_PARANTHESIS <def_multiple> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RETURN IDENTIFIER SEMICOLON RIGHT_CURLY_BRACKET

<func_int_def> →INT_TYPE IDENTIFIER LEFT_PARANTHESIS <def_multiple> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RETURN <int_expr> SEMICOLON RIGHT_CURLY_BRACKET

<func_double_def> → DOUBLE_TYPE IDENTIFIER LEFT_PARANTHESIS <def_multiple> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RETURN <expr> SEMICOLON RIGHT_CURLY_BRACKET

<func_void_def> → VOID_TYPE IDENTIFIER LEFT_PARANTHESIS <def_multiple> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET

<func_call> → IDENTIFIER LEFT_PARANTHESIS <call_parameters> RIGHT_PARANTHESIS | IDENTIFIER LEFT_PARANTHESIS <empty> RIGHT_PARANTHESIS | <read_data> | <connection_receive> | GET LEFT_PARANTHESIS RIGHT_PARANTHESIS

<if_stmt> → IF LEFT_PARANTHESIS <conditional_stmt> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET
| <if_stmt> ELSE LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET

<for_loop> → FOR LEFT_PARANTHESIS <def_int> SEMICOLON <conditional_stmt> SEMICOLON <general_assign> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET |
FOR LEFT_PARANTHESIS <general_assign> SEMICOLON <conditional_stmt> SEMICOLON <general_assign> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET

<while_loop> → WHILE LEFT_PARANTHESIS <conditional_stmt> RIGHT_PARANTHESIS LEFT_CURLY_BRACKET <stmt_list_empty> RIGHT_CURLY_BRACKET

<print_stmt> → <println> | <print>

<println> → PRINTLN LEFT_PARANTHESIS <call_parameters> RIGHT_PARANTHESIS

<print> → PRINT LEFT_PARANTHESIS <call_parameters> RIGHT_PARANTHESIS

<call_parameters> → <alphanumeric_factor> | <call_parameters> COMMA <alphanumeric_factor>

<read_data> → READ LEFT_PARANTHESIS <sensor> RIGHT_PARANTHESIS | READ LEFT_PARANTHESIS <time_stamp> RIGHT_PARANTHESIS

<switch_stmt> → SWITCH LEFT_SQUARE_BRACKET <int_expr> RIGHT_SQUARE_BRACKET ASSIGN_OP <switch_value>

<switch_value> → ON | OFF

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>

<term> → <term> MULTIPLY <factor> | <term> DIVIDE <factor> | <factor>

<int_expr> → <int_expr> PLUS <int_term> | <int_expr> MINUS <int_term> | <int_term> | LEFT_PARANTHESIS INT_TYPE RIGHT_PARANTHESIS LEFT_PARANTHESIS <expr> RIGHT_PARANTHESIS

<int_term> → <int_term> MULTIPLY <int_factor> | <int_term> DIVIDE <int_factor> | <int_factor>

<alphanumeric_factor> → STRING | <factor>

<factor> → <int_factor> | DOUBLE | MINUS DOUBLE | PLUS DOUBLE

<int_factor> → LEFT_PARANTHESIS <expr> RIGHT_PARANTHESIS | INTEGER | MINUS INTEGER | PLUS INTEGER | IDENTIFIER | <func_call>

<conditional_stmt> → <conditional_stmt> <logic_op> <conditional_basic_number>
| <conditional_stmt> <logic_op> LEFT_PARANTHESIS <conditional_basic_number> RIGHT_PARANTHESIS
| <conditional_stmt> <logic_op> NOT LEFT_PARANTHESIS <conditional_basic_number> RIGHT_PARANTHESIS
| LEFT_PARANTHESIS <conditional_stmt> RIGHT_PARANTHESIS
| <conditional_basic_number>
| NOT LEFT_PARANTHESIS <conditional_stmt> RIGHT_PARANTHESIS
| NOT <conditional_basic_number>

<conditional_basic_number> → <factor> <cond_op> <factor>

<connection> → IDENTIFIER ASSIGN_OP CONNECT LEFT_PARANTHESIS URL RIGHT_PARANTHESIS | IDENTIFIER ASSIGN_OP CONNECT LEFT_PARANTHESIS IDENTIFIER RIGHT_PARANTHESIS

<connection_send> → SEND LEFT_PARANTHESIS <func_call> COMMA IDENTIFIER RIGHT_PARANTHESIS | SEND LEFT_PARANTHESIS INTEGER COMMA IDENTIFIER RIGHT_PARANTHESIS | SEND LEFT_PARANTHESIS IDENTIFIER COMMA IDENTIFIER RIGHT_PARANTHESIS

<connection_receive> → RECEIVE LEFT_PARANTHESIS IDENTIFIER RIGHT_PARANTHESIS

<cond_op> → SMALLER_THAN | BIGGER_THAN | COMPARISON | NOT_EQUAL | SMALLER_THAN_OR_EQ | BIGGER_THAN_OR_EQ

<logic_op> → OR | AND | XOR

<sensor> → TEMPERATURE | HUMIDITY | AIR_PRESSURE | AIR_QUALITY | LIGHT | HEAT | <sound_level>

<sound_level> → SOUND COMMA INTEGER

<time_stamp> → TIME

<empty> →

<string> → STRING | URL

LANGUAGE CONSTRUCTS

<program> used to start the program. It includes main which wraps the statement list which can be used to write all the codes.

<stmt_list_empty> used to indicate that the statement list can also be empty.

<stmt_list> Comprises of statements. Each statement besides loops and if-else ends with a semicolon.

<stmt> There are various statements in this language. Each statement rule is defined here.

<define> This rule simply adds a semicolon to the end of define multiple rules.

<define_multiple> This rule can be used to define one or more variables in the program. Each variable must be separated with a comma. The variables do not have to be the same type.

<def_one> This rule is for defining only one variable of any type.

<def_string> Used when a string variable is wanted to be declared. For example string studentName;. Also, it can be used to assign a value to that variable. For example string studentName = “Çerağ”; That value can be a literal, a function result or another variable.

<def_int> Used when an integer variable is wanted to be declared. For example, int studentID; or int studentAge = 18; It can be used to declare a variable, or declare and assign any value.

<def_double> Used when a double variable is wanted to be declared. For example, double studentID; or double studentGPA = 4.00; It can be used to declare a variable, or declare and assign any value.

<general_assign> Used in assignment to anytype. For example: int x = 5; string y = "StevenWilson";

<all_numeric_assign> Used by <general_assign>, used in assignment of integers. For example: int x = 5; int y = 3*5;

Note that, double x = 4; int y = (int)(5.4); are possible assignments in PAPER.

<int_assign> Used in assignment of anything int. For example int x = 5 - 3; int y = 100; (int)(4 + 4.4);

<func_def> This rule is a statement that is used to define all the functions in our language. It defines 4 ways to write a function. Default parameters are supported by the functions.

<func_string_def> This rule is used to define a function that returns a string. Our function definition is similar to C++. The syntax is like this. string fun(int a, int b = 3) {... return "..";} Variables can take default values in definition. This creates flexibility of not plugging in some parameters while calling the function.

<func_int_def> This rule is used to define a function that returns and integer. Our function definition is similar to C++. The syntax is like this. int fun(int a, int b = 3) {... return 4;} Variables can take default values in definition.

<func_double_def> This rule is used to define a function that returns a double. Our function definition is similar to C++. The syntax is like this. string fun(int a, int b = 3) {... return 4.0;} Variables can take default values in definition.

<func_void_def> This rule is used to define a function that returns no value. The definition is similar to the other functions. Example: void fun1(string a, int b, int c) {println(b + c);}

<func_call> This rule is used to call a function. Calling a function in Paper is similar to calling a function in Java. functionName(parameters) syntax is used. A function can return an integer, double, string or void.

<if_stmt> The if statement that is common in C based languages.

<for_loop> Typical for loop, the statement list inside the brackets is done until the conditional statement becomes false. For example:

```
for( int i = 0; i <10;i++){

  print("hey"),
```

}

<while_loop> Typical while loop, the statement list inside the brackets is done until the conditional statement becomes false. For example:

while(i < 5)

{

int a = 5;

}

<println> Prints the given string into the console and ends the current line.

<print> Prints the given string into the console.

<call_parameters> used inside when a function with parameters is called. Multiple parameters with a variety of types are supported. For example: fnc(5, “hello”);

<read_data> This rule is used to get the data from sensors or the timer. In this language, sensors are numerically limited and predefined. Therefore, the syntax is very basic. To read data from a sensor or timer read(nameOfTheTarget) syntax is used.

<switch_stmt> There are 10 switches that can be set on or off. To do this, the user should write switch[(A number from 0 to 9)] = ON or OFF. A number bigger than 9 is invalid since there are 10 switches.

<switch_value> used to identify ON And OFF states for switches. For example: switch[6] = ON;

<expr> used to define subtraction and summation operations with <term>. This nonterminal helps multiplication and division have higher precedence

<term> used to define multiplication and division operations with <factor>.

<int_expr> Used to indicate integer expression. For example: for example (5 + 4) / 2. It doesn’t support double values.

<int_term> Used to indicate integer terms with multiplication and division for higher presedence. Used in <int_expr>.

<alphanumeric_factor> Used in <call_parameters> to include Strings to factors. For example: double getNumber(string ex,int y = 5,int a)

<factor> indicates any numerical factor. For example (4 - 2) or 5 or -5 or 3.2 or -3.2 or a variable x being a numerical value.

<int_factor> This is used by <factor> to differentiate the integer type of factor.

<conditional_stmt> Recursive behaviour using <conditional_basic_number> as a base. Useful for comparing multiple conditional statements. Returns the result as a boolean value. For example:

((a==b) >= (c <= 5))

(((a==b) >= (3 <= 5)) < ((a==b) >= (c <= 5)))

<conditional_basic_number> Is used to compare the values of identifiers and numbers,is composed of two identifiers or numbers with a conditional operator in the middle. Derives the result as a boolean. For example:

i <= 5

y != C

<connection> This is used to assign a connection connected by the connect(URL) command. For example: firstConnection = connect("https://iotserver.xxx.ooo.com");

<connection_send> This rule is used to send an integer to the connection. An example of its syntax is send(5,connection);. Note that before using this function a connection must be established via connect(URL) command.

<connection_receive> This rule is used to receive an integer from the connection. An example to it's syntax is x = receive(connection). Note that before using this function a connection must be established via connect(URL) command.

<cond_op> used to denote comparison operators to be used in <conditional_stmt>. Conditional operators include smaller than(<), bigger than(>), comparison(==), not equal to(!=), smaller than or equal(<=) and bigger than or equal(>=).

<logic_op> used to denote logic operators such as or, and, not, xor. For example in a while loop:

while((i <= 10 and i == 1.5) or not(i != 7.5))

<sensor> used to denote the sensor in which the program reads the temperature, humidity, air pressure, air quality, light, heat and sound level from. Used in read(<sensor>) utility.

<sound_level> Sound sensor is different from the other sensors. All the other sensors can be used by writing read(sensorName). However, reading the sound level also requires a frequency. Its syntax is like read(sound, 2000); 2000 being frequency. <sound_level> defines this rule.

<time_stamp> used to denote the current timestamp to be read by read(<time_stamp>) utility. Timestamp is the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970, not counting leap seconds.

<empty> This rule represents no code. It is added to the rules make it possible to create empty programs.

<string> used to denote strings or URLs used during the programming.

TERMINALS OF PAPER

println - terminal used to print output with a newline appended
print - terminal used to print output in PAPER
URL - terminal used to denote the URL to be connected to in PAPER.
get - It is used to get any type of value from the user.
send - function name used to send an integer or an integer returning function regarding connection utility in PAPER
receive - function name used to receive a connection in PAPER
connect - function name used to connect to a given URL in PAPER
read - function name terminal used to get timestamp or sensor values in PAPER
time - terminal to indicate time stamp read by sensors in PAPER
temperature - used and get by a sensor in temperature-related work
airPressure - used and get by a sensor in airPressure related work
airQuality - used and get by a sensor in airQuality related work
light - used and get by a sensor in light related work
sound - used and get by sensor in sound-related work
heat used and get by a sensor in heat-related work
humidity used and get by a sensor in humidity-related work
“<” - strictly smaller than symbol in PAPER
“>” - strictly greater than symbol in PAPER
“<=” - smaller than or equal to symbol in PAPER
“>=” - greater than or equal to symbol in PAPER
“==” - equal symbol in PAPER
“!=” - not equal to symbol in PAPER
“=” - assignment operator in PAPER
or - logical or, disjunction symbol in PAPER
xor - logical xor, exclusive disjunction symbol in PAPER
and - logical and, conjunction symbol in PAPER
not - negation symbol in PAPER
“#” - Comment line start and finish symbol in PAPER
“+” - Addition in PAPER
“-” - Subtraction in PAPER

“*” - Multiplication in PAPER
“/” - Division in PAPER
“,” - Comma in PAPER
“(“ left and **“)”** right parentheses of PAPER
“{“ left and **“}”** curly brackets of PAPER
“[“ left and **“]”** right square brackets of PAPER

NONTRIVIAL TOKENS

- ### COMMENTS

Comments of the language is a string between two #’s is identified as a comment. The motivation behind this implementation is based on its writability and readability. That kind of syntax is very common, so it is easily readable and writable for most of the programmers.

- ### IDENTIFIERS

Identifiers can not start with numbers but can be followed by numbers. The reason for choosing such a syntax is simplicity and commonness. It is the same with Java and C syntax, so most programmers will not have any difficulty with using identifiers. There are two types of identifiers, numerical and string. That way is chosen to reduce the work in avoiding changing the type of an identifier to the wrong type.

- ### LITERALS

Literals **int** and **double** are used for numerical literals as in most of the popular languages.**double is** for fractional numbers while **int** is for the others.

The literal **string** is covered with quotation marks to avoid confusion with identifiers.

Example confusion to indicate the usage of quotation marks
int hey = 5;
string example = hey;

- ### RESERVED WORDS

int used in the declaration of a numerical identifier
double used in declaration of a numerical identifier
string used in the declaration of a string identifier
void used to declare void functions.
or the or operator in logical expressions

and the and operator in logical expressions
not the not operator in logical expressions
ON used to indicate the switch is on
OFF used to indicate the switch is off
temperature used and get by a sensor in temperature-related work
airPressure used and get by a sensor in airPressure related work
airQuality used and get by a sensor in air-quality related work
light used and get by a sensor in light related work
sound used and get by a sensor in sound-related work
heat used and get by a sensor in heat-related work
humidity used and get by a sensor in humidity-related work
time used in time-related work to get the time
send used to send an integer
receive used to receive an integer
switch used to indicate a switch
connect used to connect to an URL
if used in if or if-else statements
else used in if-else statements
while used in while statements
for used in for statements
read used to read from the timestamp and sensor
print used to print a string to the console
println used to print a string to the console and to end the line
return used to return a value in a function

,

MOTIVATION OF DESIGN CHOICES in PAPER

- **READABILITY**
  Readability is one of the most important features of a programming language. When the language is readable, the users are more prone to work efficiently because they familiarize themselves with it on a deeper level. PAPER is an IoT programming language but It is very similar in syntax with C and Java programming languages. As those are one of the most fundamental and most know programming languages, our users with little programming knowledge can adapt to PAPER. The usage of primitive data types; string, boolean, integer and double usages are familiar to the users. The reserved words' and methods' namings are done such that their functionalities are very clear to the user. The initializations and declarations of variables are the same as the C programming language. Also, the function calling and defining syntax is the syntax that is most common to all programmers. Also, every code of block

related to an if-else statement is between curly brackets to improve readability.

- **WRITABILITY**
  In order to improve writability, PAPER has two of the most common loop structures: for and while loops. Their syntax is the same with C, where the user declares a variable and iterates given that a condition holds. The print statements are easier than the Java syntax lengthwise, the print statement prints the parameter It is given, and println statement prints the parameter with a new line appended to it.

- **RELIABILITY**
  PAPER has a variety of types regarding the numerical integer and double values, string values and boolean values. Users can rely on these indicators while declaring their variables, as their data will correspond to these types. PAPER supports the precedence rules that are most common globally, in descending order: expressions in the parenthesis, multiplication, and division, subtraction, and addition. The clarity of the code's language is crucial when the programmer wants to minimize the errors. The language of PAPER provides clarity and formalization to the language.