BILKENT UNIVERSITY

CS 315 - PROGRAMMING LANGUAGES

Project Group: 6

- Atasagun Şanap, 21902435, Section 2
- Melis Atun, 21901865, Section 1
- Öykü Erhan, 21901541, Section 2

Name of our Language: FLYRONE

BNF Description

<program> ::=
<program> <statement_block> | **<statement_block>** | **<statement>**

<statement_block> ::= **<statement_block> <statement>** | **<statement_block>**
<function> | **<statement_block> <loop>** | **<function>** | **<loop>** |**<statement>**

<function> ::= **<function> <statement>** | **<function><loop>** | **<statement>** |
<loop>

<loop> ::= **<for>** | **<while>**
<for> ::= <statement> | **<for> <statement>**
<while> ::= <statement> | **<while> <statement>**

<statement> ::= **<return_statement>** | **<condition_statement>** |
<in_out_statement> | **<comment_statement>** | **<assignment_statement>**

<return_statement> ::= RETURN **<expression>** SC

<condition_statement> ::=
IF LP **<expression>** RP **<statement_block>** |
IF LP **<expression>** RP **<statement_block>** ELSE **<condition_statement>** |
IF LP **<expression>** RP **<statement_block>** ELSE **<statement_block>**

<in_out_statement> ::= **<in_statement>** | **<out_statement>**
<in_statement> ::= INPUT_BOOLEAN <boolean> |
INPUT_DOUBLE **<double>** | INPUT_STRING <**string**> SC

<out_statement> ::= PRINT LP VAR | **<var>** | **<expression>** RP SC

<comment_statement> ::= DS **<comment>** |
STAR_SLASH **<comment>** SLASH_STAR

<assignment_statement> ::= **<double_assignment>** | **<string_assignment>** | **<boolean_assignment>**

<double_assignment> ::= DOUBLE ASSIGNMENT_OP **<double>** SC |
DOUBLE ASSIGNMENT_OP INPUT_DOUBLE SC |
DOUBLE ASSIGNMENT_OP DOUBLE SC |
DOUBLE ASSIGNMENT_OP **<return_statement>**

<string_assignment> ::= STRING ASSIGNMENT_OP **<string>** SC |
STRING ASSIGNMENT_OP INPUT_STRING SC |
STRING ASSIGNMENT_OP STRING SC |
STRING ASSIGNMENT_OP **<return_statement>**

<boolean_assignment> ::= BOOLEAN ASSIGNMENT_OP **<boolean>** SC |
BOOLEAN ASSIGNMENT_OP INPUT_BOOLEAN SC |
BOOLEAN ASSIGNMENT_OP BOOLEAN SC |
BOOLEAN ASSIGNMENT_OP **<return_statement>**

<var> ::= STRING | DOUBLE | BOOLEAN
<general_comparator> ::= LT | LTE | GT | GTE | EE | NE | OR | AND

<expression> ::= **<expression>** * **<var>** | **<var>** * **<var>** |
<expression> <general_comparator> <expression> |
<expression> <general_comparator> <var> |
<var> <general_comparator> <var> |
<expression> + **<var>** | **<var>** + **<var>** | **<var>**

ASSIGNMENT_OP ::= =
DS ::= //
STAR_SLASH ::= */
SLASH_STAR ::= /*
LP ::= (
RP ::=)
SC ::= ;

COLON ::= :
DOT ::= .
COMMA ::= ,

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

<sign> ::= + | -

Reserved Words

IF
ELSE
RETURN
PRINT

Primitive Functions

<builtin_function_identification> ::= get_reading | get_alti | get_temp | go_vert | go_up | go_down | stop_horiz | go_forward | go_backward | stop_turn_left | stop_turn_right | turn_on_nozzle | turn_off_nozzle | connect_WIFI

get_reading
A function to read the reading.

get_alti
A function to read the altitude.

get_temp
A function to read the temperature.

go_vert
A function to go in the vertical direction.

go_up
A function to go in the upper direction.

go_down
A function to go in the downward direction.

stop_horiz
A function to stop horizontally.

go_forward
A function to go forward.

go_backward
A function to go backward.

stop_turn_left
A function to stop and turn the heading left.

stop_turn_right
A function to stop and turn the heading right.

turn_on_nozzle
A function to turn on the nozzle.

turn_off_nozzle
A function to turn off the nozzle.

connect_WIFI
A function to connect to the base computer through a WIFI.

Description of Non-Terminal Literals

<program> is the most general literal of the language. All of the codes that are written fall into this literal. This literal is composed of one or multiple **<statement_block>** or one **<statement>**.

<statement_block> is the second most general literal. This refers to a group of code segments that is a part of the **<program>**. This literal is composed of a combination of a single **<statement>**, a group of **<statement>**, a group of **<function>**, and a group of **<loop>**.

<function> is a literal that is composed of statements or loops and their combinations. Note that this gives the flexibility of using loops in functions as well as in a proper language.

<loop> is another complex literal that consists of **<for>** loop or **<while>** loop.

<for> represents the for loop and is composed of one or more statements.
<while> represents the while loop and is composed of one or more statements.

<statement> statement is the building block of the whole language and is the most important one. It must be one of the five statement types. These are **<return_statement>**, **<condition_statement>**, **<in_out_statement>**, **<comment_statement>** or **<assignment_statement>**.

<return_statement> is as its name suggests a return statement that returns a value.

<comment_statement> is a line that gives a comment and is not supposed to be recognized by the compiler. It must be done with the use of double slash in the beginning of the line or slash star and star slash at the beginning and at the end of the statement.

<in_out_statement> is either an **<in_statement>** or an **<out_statement>**.

<in_statement> either gets a boolean, double or string with their corresponding input words.
<out_statement> prints out an expression with PRINT keyword, with a LP and either a VAR (variable name), **<var>** or an **<expression>** ending with a RP and SC (semicolon)
<condition_statement> is composed of a type of writing. First IF keyword and LP (left parenthesis) needs to be used. Then comes an expression with a RP (right parenthesis). Then, after a **<statement_block>**, either the statement ends or an ELSE part comes in with either another conditional statement or a **<statement_block>**.

<assignment_statement> is either a **<double_assignment>**, a **<string_assignment>** or a **<boolean_assignment>**. All of these assignments are done by a variable (DOUBLE or STRING or BOOLEAN), then an ASSIGNMENT_OP (assignment operator), and then either a value of the same type, an input, another variable or a return statement.

<var> represents a variable and is either a STRING, DOUBLE or BOOLEAN.
<expression> is another very important literal. It expresses a group of values and their relations with each other. It is either a sum or multiple of variables, comparison of expressions or variables or a combination of these.

Evaluation of our Language

Readability: In order to make our programming language readable, we avoided unnecessary details and tried to make it as simple as possible so that the reader can understand every detail clearly. For example we used the enhanced types such as double and string rather than giving place to integers and chars, making the language easier to read by categorizing into a common variable. Furthermore, names of all the literals are chosen in a way that they are understandable by the reader. Also, we made sure that they are resembling the most commonly used programming languages' literals as well, making it easier to adapt to for newcomers.

Writability: We made sure that our language is writable by merging integers into doubles. Also, we do not have a char type because input is taken by string which is way more enhanced than char. This allows our language to be written easily without the concerns of typecasting as much as possible. Furthermore, for and while loop structures are added to our language hence loops are enhanced and detailed, which provides more ability to the writer of the program. In addition, there is a variety of combinations in our language which can be used by the statement block. For instance; functions, loops, and groups of statements or all of their combinations can be used by statement blocks. Therefore, this enhances the flexibility and use of our programming language and it is easier for the writer to accomplish what he/she needs to.

Reliability: There are some recursive definitions of non-terminal literals in our language which prevents ambiguity and enhances the reliability of the language. Furthermore, the simplicity of our language makes it easy-to-use and therefore reliable at the same time.