BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 315 PROJECT 1

DESIGN REPORT

Can Kırımca 21802271 Section-2

Can Kırşallıoba 21801768 Section-2

**RUNE**

1. **Program Definition**

<program> ::= <statement\_list>

<statement\_list> ::= <statement>

                    | <statement\_list> <statement>

<statement> ::= <comment>

               | <expression>;

               | <loop>

               | <conditional>

               | <function\_definition>

<comment> ::= <comment\_sign> <form> <comment\_sign>

<form> ::= <identifier> <form>

          | <identifier>

<char> ::= <letter>

         | <digit>

<letter> ::= <lowercase\_letter>

           | <uppercase\_letter>

<lowercase\_letter> ::= a | b | c ... | z

<uppercase\_letter> ::= A | B | C ... | Z

<assignment\_operator> ::= <-         // so that it is different than other languages.

<comment\_sign> ::= "$"

**2) Types and Constants**

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

<int> ::= <int> <digit>

        | <digit>

<integer> ::= <sign> <int>

            | <int>

<fl> ::= <int> <dot> <int>

        | <dot> <int>

<float> ::= <sign> <fl>

          | <fl>

<boolean> ::= TRUE | FALSE

<space> ::=

<left\_curly\_parenthesis> ::= "{"

<right\_curly\_parenthesis> ::= "}"

<left\_parenthesis> ::= "("

<right\_parenthesis> ::= ")"

<greater\_than\_relation> ::= ">" | ">="

<smaller\_than\_relation> ::= "<" | "<=

<plus\_operator> ::= +

<subtraction\_operator> ::= -

<multiplication\_operator> ::= \*

<division\_operator> ::= /

<remainder\_operator> ::= %

<exponentiation\_operator> ::= \*\*

<and\_operator> ::= &&

<or\_operator> ::= ||

<equal\_operator> ::= ==

<not\_equal\_operator> ::= !=

<identifier> ::= <letter>

                | <identifier> <char>

**3) Loop Definition**

<loop> ::= <while>

        | <for>

<while> ::= while <left\_parenthesis> <logical\_expression> <right\_parenthesis> <left\_curly\_parenthesis> <statement\_list> <right\_curly\_parenthesis>

<for> ::= for <left\_parenthesis> <for\_loop\_expression> <right\_parenthesis> <left\_curly\_parenthesis> <statement\_list> <right\_curly\_parenthesis>

<for\_loop\_expression> ::= <expression> ; <boolean>; <expression>

**4) Conditional Definition**

<conditional> ::= <if>

<if> ::= if <left\_parenthesis> <logical\_expression> <right\_parenthesis> <left\_curly\_parenthesis> <statement\_list> <right\_curly\_parenthesis> <else>

        | if <left\_parenthesis> <logical\_expression> <right\_parenthesis> <left\_curly\_parenthesis> <statement\_list> <right\_curly\_parenthesis>

<else> ::= else <left\_curly\_parenthesis> <statement\_list> <right\_curly\_parenthesis>

**5) Expressions**

<int\_expression> ::= <integer>

<float\_expression> ::= <float>

<function\_expression> ::= <identifier> <left\_parenthesis> <argument\_list> <right\_parenthesis>

<logical\_expression> ::= <boolean>

                        | (<integer> | <float> | <identifier>) (<greater\_than\_relation> | <smaller\_than\_relation>) (<integer> | <float> | <identifier>)

                        | <logical\_expression> <and\_operator> <logical\_expression>

                        | <logical\_expression> <or\_operator> <logical\_expression>

                        | <logical\_expression> <equal\_operator> <logical\_expression>

                        | <logical\_expression> <not\_equal\_operator> <logical\_expression>

<function\_call> ::= <function\_expression>;

<assignment> ::= <identifier> <assignment\_operator> <expression>

<expression> ::= <expr> <plus\_operator> <expression2>

                | <expression> <subtraction\_operator> <expression2>

                | <expression2>

<expression2> ::= <expression2> <multiplication\_operator> <expression3>

                | <expression2> <division\_operator> <expression3>

                | <expression3>

<expression3> ::= <expression4> <exponentiation\_operator> <expression3>

                | <expression4>

<expression4> ::= <left\_parenthesis> <expression> <right\_parenthesis>

                | <expr>

                | <identifier>

<expr> ::= <int\_expression>

        | <float\_expression>

        | <function\_expression>

        | <logical\_expression>

**6) Function Definition**

<return\_type> ::= <void>

                | <data\_type>

<function\_definition> ::= <return\_type> <identifier> <left\_parenthesis> <argument\_list> <right\_parenthesis> <left\_curly\_parenthesis> <statement\_list> <right\_curly\_parenthesis>

<argument> ::= <integer>

            | <float>

            | <char>

            | <boolean>

            | <identifier>

            | <function\_expression>

<argument\_list> ::= <argument>, <argument\_list>

                | <argument>

**7) Input Output Definition**

<input\_statement> ::= scan <left\_parenthesis> <expression> <right\_parenthesis>;

<output\_statement> ::= print <left\_parenthesis> <expression> <right\_parenthesis>;

**8) Drone Functions**

<read\_inclination\_function> ::= readInclination <left\_parenthesis> <right\_parenthesis>;

<read\_altitude\_function> ::= readAltitude <left\_parenthesis> <right\_parenthesis>;

<read\_temperature\_function> ::= readTemperature <left\_parenthesis> <right\_parenthesis>;

<read\_acceleration> ::= readAcceleration <left\_parenthesis> <right\_parenthesis>;

<turn\_on\_camera\_function> ::= turnOnCamera <left\_parenthesis> <boolean> <right\_parenthesis>;

<take\_picture\_function> ::= takePictures <left\_parenthesis> <right\_parenthesis>;

<read\_timestamp\_function> ::= readTimestamp <left\_parenthesis> <right\_parenthesis>;

<connect\_to\_computer\_with\_wifi\_function> ::= connectToComputer <left\_parenthesis> <right\_parenthesis>;

**9) Our Creative Functions**

<do\_flip\_function> ::= doFlip <left\_parenthesis> <boolean> <right\_parenthesis>;

<takeoff\_function> ::= takeoff <left\_parenthesis> <right\_parenthesis>;

<land\_function> ::= land <left\_parenthesis> <right\_parenthesis>;

<emergency\_function> ::= emergency <left\_parenthesis> <boolean> <right\_parenthesis>;

<up\_function> ::= up <left\_parenthesis> (<integer> | <float>);

<down\_function> ::= down <left\_parenthesis> (<integer> | <float>) <right\_parenthesis>;

<right\_function> ::= right <left\_parenthesis> (<integer> | <float>) <right\_parenthesis>;

<left\_function> ::= left <left\_parenthesis> (<integer> | <float>) <right\_parenthesis>;

<forward\_function> ::= forward <left\_parenthesis> (<integer> | <float>) <right\_parenthesis>;

<backward\_function> ::= backward <left\_parenthesis> (<integer> | <float>) <right\_parenthesis>;

<rotate\_clockwise\_function> ::= rotateClockwise <left\_parenthesis> <boolean>, <integer> <right\_parenthesis>;

<set\_speed\_function> ::= setSpeed <left\_parenthesis> (<integer> | <float>) <right\_parenthesis>;

<get\_speed\_function> ::= getSpeed <left\_parenthesis> <right\_parenthesis>;

<get\_battery\_function> ::= getBattery <left\_parenthesis> <right\_parenthesis>;

**Explanation of the RUNE Language Constructs**

**TYPES**

In RUNE, there are 4 data types, which are the following: int, float, boolean, and char.

**SYMBOLS**

**Assignment Operator (<-):**

In our language assignment operator is coded as <-. An example of its usage is as follows:

a <- b

In the example above, the value of b is assigned to a.

**Comment ($):**

Comments are handled with $ in RUNE. Comments should be applied as inline comments. For instance;

$ This is an example comment.

**Exponentiation Operator (\*\*):**

Exponentiation operation will be implemented just like in Python language.

An example of its usage would be as follows:

 var <- 5\*\*2;

**PROGRAM DEFINITIONS**

A program in RUNE consists of a statement list. Statement list can be either one statement or it can be multiple statements. Statements include comments, expressions followed by a semicolon, loops, conditionals or it can be a function definition. These expressions will be explained in their respective sections.

**LOOP DEFINITIONS**

We have two loop types in RUNE; while loop and for loop. The syntax of these loops is the same as C group languages.

**CONDITIONAL DEFINITIONS**

In RUNE we have two types of conditional to keep the language simple and friendly for beginners. The two conditionals we have are the if statement and else statement. An else could only follow an if statement, however an if statement can be a standalone block. The syntax of if and else statements are inspired by C group languages.

**EXPRESSIONS**

We have gathered all the expressions in the section of the code. In this section, we have expressions of integer types, float types, logical expressions, functions, and operators. For the associativity and the precedence of the operators, we have followed the pattern that was explained to us by our instructor and in our book. We have separated the operators that have different precedences. The operators that have the lowest precedence are higher in the hierarchy, and the operator that has the highest precedence is at the lowest of the hierarchy, which is the exponentiation operator. In general, when we refer to an expression throughout the file, we mean that it can be an integer expression, a float expression, a logical expression, and a function expression.

**FUNCTION DEFINITIONS**

We were inspired by the C group languages for the function definitions, as well. We have a return type which can be one of the data types we have introduced to the language. However, if the user does not want to return anything in that function, the return type can also be void. For the arguments of the function, we have enabled floats, chars, booleans, identifiers, and also function calls.

**INPUT AND OUTPUT DEFINITIONS**

For the input and output definitions, we have selected the “scan” and “print” keywords.  “Scan” gets the user input from the console and “print” outputs the given expression to the console.

**DRONE FUNCTION DEFINITIONS**

The drone functions which are described in the project document will be implemented as inbuilt functions.

Read Inclination functionality takes the data from the gyroscope and outputs it to the console.

Reading Altitude functionality takes the data from the barometer,  and outputs it to the console.

Reading temperature functionality takes the data from the thermometer and outputs it to the console.

Read Acceleration functionality takes the data from the acceleration sensor and outputs it to the console.

Turning On/Off  camera functionality turns on the camera if the user passes the related boolean expression.

Taking Pictures functionality uses the camera that is on the drone to take pictures.

 Reading the timestamp from the timer outputs the timestamp to the console.

Connecting to the base computer via wifi connects to the computer.

**OUR CREATIVE FUNCTIONS**

Doing flip We looked at the Software Development Kit and decided to have the full functionality of the commercial drone “Tello”. We have a variety of inbuilt functions which includes but is not limited to, taking off, doing flips, moving in the specified directions, and so on. We will explain these functions in great detail down below.

Do flip functionality flips the drone in the specified direction. The user can specify the direction by passing the corresponding boolean value as an argument. The value ‘TRUE’ means front flip and ‘FALSE’ means backflip.

**Take Off** functionality takes off the drone when the function is called.

**Land** functionality lands the drone when the function is called.

**Emergency** functionality stops the drone in case of an emergency if the regarding boolean expression is passed to the function.

**Up** functionality moves the drone in up direction.

**Down** functionality moves the drone in down direction.

**Right** functionality moves the drone in right direction.

**Left** functionality moves the drone in left direction.

**Forward** functionality moves the drone in forward direction.

**Backward** functionality moves the drone in backward direction.

**Rotate Clockwise** functionality rotates the drone by looking at the passed boolean value. If the user passes TRUE then the drone rotates clockwise, if the user passes FALSE the drone rotates counterclockwise.

**Set Speed** functionality sets the drone’s speed to the value passed as argument.

**Get Speed** functionality gets the drone’s speed.

**Get Battery** functionality gets the current value of the battery.

**EVALUATION OF OUR LANGUAGE IN TERMS OF READABILITY, WRITABILITY, AND RELIABILITY**

**Readability**

Our language is designed to be easy to use for users ranging from beginners to experts. The syntax is very close to C group languages. We chose this approach because the C group languages are around for a very long time and people have grown accustomed to the syntax of them. In terms of orthogonality, we have paid close attention not to disregard the issue. We did not assign an operator to multiple operations, hence making RUNE appropriate to orthogonality. When it comes to data types, the types that we created are to be assigned to their respective identifiers to avoid confusion.

**Writability**

Our language has an adequate number of data types and functionalities so that it does not confuse the writer of the code. Since we have a small number of primitive constructs and consistent sets of rules for combining them we have enabled orthogonality for the language. For expressivity of the language, we created different and easy forms of writing the same expression such as using for loops instead of while loops.

**Reliability**

We tried to create the language so that it performs to its specifications under all sorts of conditions. For instance, in terms of type checking, we have enabled one of our functions to accept speed values for both int types and float types. At this stage; we did not implement any sort of exception handling system, because we only wrote the BNF and the lex specification file. The same applies to aliasing. For readability and writability, we made sure that all the conditions for them are met, thus enhancing the reliability of our language.