Bilkent University

CS 315- PROGRAMMING LANGUAGES

PROJECT 2

Rapton Programming Language

Ali Eren Günaltılı 21801897 Section 1

Bora Altınok 21902206 Section 3

Ceyda Şahin 21903448 Section 3

Contents

BNF

<program> ::= <begin><declare_function><main><end>

<main> ::= <MAIN><LP><RP><LCB><stmts><RCB>

<stmts> ::= <stmt>|<stmt><stmts>

<stmt> ::= <matched_stmt> | <unmatched_stmt>

<matched_stmt> ::=
<if><LP><logic_expr><RP><LB><matched_stmt><RB><else><LB><matched
_stmt><RB> | <init_stmt> | <declaration_stmt> | <return_stmt> | <comment>
|<struct_stmt> | <loop_stmt> | <call_function_stmt>
|<array_stmt>|<declare_function_stmt> |<output_stmt>|<input_stmt>

<unmatched_stmt> ::= <if><LP><logic_expr><RP><LB><matched><RB>
| <if><LP><logic_expr><RP><LB><unmatched><RB>
|<if><LP><logic_expr><RP><LB><matched_stmt><RB><else><LB><unmatc
hed_stmt><RB>

<logic_expr> ::= <id> | <id><space><comparison_expr><space><id>
<comparision_expr>::=<and>|<or>|<equality_identifier><equals_op>|<great
er_op> |<lesser_op>|<notEquals_op>|<greaterOrEqual_op>
|<lesserOrEqual_op>

<input_stmt> ::= <raptin><LP><type_identifier><identifier><RP><end_stmt>

<type_identifier>::= <int><space>|<float><space>|<char><space>

|<string><space>|<bool><space>|<surface><space>

<identifier> ::= <letter> | <identifier><digit> |<identifier><letter>|
<heading>|<altitude>|<drone_x>|<drone_y>

<init_stmt> ::= <type_identifier><identifier><assign_op><term><end_stmt> |
<surface_init_stmt> | <identifier><assign_op><term><end_stmt>

<declaration_stmt> ::= <type_identifier><identifier><end_stmt> |
<const><type_identifier><identifier><end_stmt>

<surface_init_stmt> ::=
<type_identifier><identifier><assign_op><surface_term><end_stmt>

<surface_term> ::=
<LP><surface_width><assign_op><id><comma><surface_height><assign_op
><id>
<comma><surface_centerX><assign_op><id><comma><surface_centerY><a
ssign_op> <id><RP> // Surface
surf=(sWidth=5,sHeight=11,sCenterX=0,sCenterY=0);

<struct_stmt> ::=<struct><space><identifier><LB><stmts><RB><end_stmt>

<array_stmt> ::= <type_identifier><identifier><dimension><end_stmt>

<dimension> ::= <dimension><LSB><int_value><RSB> |
<LSB><int_value><RSB>

<output_stmt> ::= <raptout><LP><id><RP><end_stmt>

<call_function_stmt> ::=

<call><arrow><func_name><func_body><end_stmt>

<func_name> ::= <identifier>

<predefined_funcs> ::= |<get_heading>|<get_altitude>|<move_forward>|<move_backward
>|<get_temp>|<nozzle_on_off>|<stop_drone>|<heading_left>|<heading_right>
|<climb_up>|<drop_down>|
<surface_get_height>|<surface_get_width>|<surface_get_centerX>|<surface_
get_centerY> |<connect_wifi>

<func_body> ::= <LP><body_context><RP>

<body_context> ::= |<identifier><comma><body_context> | <identifier>
| <type_identifier><identifier><comma><body_context>
| <type_identifier><identifier>

<declare_function_stmt> ::=
<function><space><identifier><func_body><LB><stmts><RB>

<return_stmt> ::= <return><space><id><end_stmt>

<loop_stmt> ::= <while_stmt> | <for_stmt>

<for_stmt> ::=
<for><LP><int><space><identifier><space><in><space><int_value><arrow>
<int_value>
<space><step><space><arith_op><space><id><RP><LB><stmts><RB>

<while_stmt> ::= <while><LP><logic_expr><RP><LB><stmts><RB>

<term> ::= <factor> | <term> + <id> | <term> - <id>

<factor> ::= <factor> * <id> | <factor> / <id> | <id>

<id> ::= <identifier> | <value>

<value> ::= <int_value> | <float_value> | <string_value> | <char_value>|
<bool_value>

<int_value> ::= <number><digit> | <digit>

<string_value> ::= <string_identifier><word><string_identifier>

<char_value> ::= <char_identifier><letter><char_identifier>

<bool_value> ::= <true_value> | <false_value>

<true_value>::= <true>| <not><false>

<false_value>::= <false>| <not><true>

<comment> ::= <begin_comment><sentence><end_comment>

<begin_comment> ::= <HTS>

<end_comment> ::= <EXM><HTS>

<sentence> ::= <identifier><space> | <identifier>

<word> ::= <letter><word> | <letter>

1. General Program Features and Tokens

In this section, the program structure of Rapton will be given.

1.1 Program Definition

<program> ::= <beginning><declare_function_stmt><main><end>

This non-terminal is the starting variable for our program. Everything derives from this initialization. All the things we want to include in our language had to be derived from <stmts> if our BNF is true. The program had to start with “begin” and had to end with “end”.

1.2 Statements Definition

<main>::= <MAIN><LP><RP><LCB><stmts><RCB>

<stmts> ::= <stmt>|<stmt><stmts>

This is a recursive definition of <stmts> non-terminal to make any valid sentence derivation possible.

1.3 A Statement Definition

<stmt> ::= <matched_stmt> | <unmatched_stmt>

This non-terminal is the summary of the whole sentences and a fundamental part of Rapton’s BNF description. By using <stmt>, <matched_stmt> and, <unmatched_stmt> it is possible to build the BNF for every valid statement in our language. There is a simple parse tree example showing the importance of <stmt> non-terminal. (See Appendix A.)

1.3.1 <matched_stmt> ::= |<if><LP><logic_expr><RP><LCB><matched_stmt><RCB><else><LCB><matched_stmt><RCB>|<init_stmt> | <declaration_stmt> | <return_stmt> | <comment> |<struct_stmt> | <loop_stmt> | <call_function_stmt> |<array_stmt> |<declare_function_stmt> |<output_stmt>|<input_stmt>

<matched_stmt> statement refers to balanced else-if statements. In fact, every type of statement derives from this non-terminal. Non-if type statements are also considered <matched_stmt>. This rule is recursive since it’s possible to have more than one statement inside the if or else conditions. Primary attribute of this statement is that it obliges the user to use consecutive else statements for each if statement. A matched if-else statement block is as depicted below:

```
  if(){
  }
else{
    if(){
}
  }
```

<unmatched_stmt> and <matched_stmt> are defined distinctly to prevent ambiguity of statements. Finally, <matched_stmt> can be null in case we don't have anything inside the if or else statement.

1.3.2 <unmatched_stmt> ::= <if><LP><logic_expr><RP><LB><matched><RB>

| <if><LP><logic_expr><RP><LB><unmatched><RB>

|<if><LP><logic_expr><RP><LB><matched_stmt><RB><else><LB><unmatc

hed_stmt><RB>

<unmatched_stmt> also provides a solution for unbalanced if-else ambiguity. We have <stmts> to the left of *or;* in case of an abundance of statements inside the if condition. To the right of *or*, explains a case similar to below:

```
if(){
}
else{
  if(){
  }
}
```

1.4 BNF of Tokens & Non - Terminals

Rapton has defined tokens for important reserved words, keywords, drone properties and built-in functions.

<beginning> ::= begin

<end> ::= end

<not> ::= <EXM>

<LSB> ::= [

<RSB> ::=]

<HTS> ::= #

<EXM> ::= !

<LP>::= (

<RP>::=)

<LCB>::= {

<RCB>::= }

<MAIN> ::= main

```
<comma>::= ,

<end_stmt>::= ;

<if>::= if

<else>::= else

<function> ::= function

<for> ::= for

<in> ::= in

<step> ::= step

<while> ::= while

<and> ::= and

<or> ::= or

<arrow> ::= ->

<string_identifier> ::= “

<char_identifier> ::= ‘

<equals_op>::= ==

<equality_identifier>::=

<greater_op>::= >

<lesser_op>::= <

<notEquals_op>::= !=

<greaterOrEqual>::= >=

<lesserOrEqual_op>::= <=

<arith_op> ::= +|-|*|/

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

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

<letter> ::=
A|a|B|b|C|c|D|d|E|e|F|f|G|g|H|h|I|i|J|j|K|k|L|l|M|m|N|n|O|o|P|p|Q|q|R|r|
S|s|T|t|U|u|V|v|W|w|X|x|Y|y|Z
```

<int> ::= int

<float> ::= float

<bool> ::= bool

<char> ::= char

<string> ::= string

<const> ::= const

<true>::= true

<false>::= false

<drone_x> ::= droneX

<drone_y> ::= droneY

<struct> ::= struct

<surface> ::= Surface

<altitude> ::= altitude

<heading> ::= heading

<surface_height> ::= surfaceHeight

<surface_width> ::= surfaceWidth

<surface_centerX>::= sCenterX

<surface_centerY>:: sCenterY

<surface_get_height> ::= sgetHeight

<surface_get_width> ::= sgetWidth

<surface_get_centerX> ::= sgetCenterX

<surface_get_centerY> ::= sgetCenterY

<power_function> ::= mathPow

<get_heading> ::= getHeading

<get_altitude> ::= getAltitude

<move_forward> ::= moveForward

<get_temp> ::= getTemp

<nozzle_on_off> ::= nozzleOnOff

<stop_drone> ::= stopDrone

<heading_left> ::= headingLeft

<heading_right> ::= headingRight

<climb_up> ::= climbUp

<drop_down> ::= dropDown

<connect_wifi> ::= connectWifi

<call>::= call

<callp>::= callp

1.5 Description of Non-Trivial Tokens

int: Reserved token for implying integer type data or variable.

float: Reserved token for implying float type data or variable.

string: Reserved token for implying string type data or variable.

char: Reserved token for implying char type data or variable.

bool: Reserved token for implying boolean type data or variable.

raptout: Reserved token for output function that is predefined in Rapton. It is easy to remember. It is easy to use by considering the name of the language. That's why we choose this word since it is easy to remember and write. We want to create a short word for output and input statements, unlike Java.

raptin: Similar to raptout, it can be associated with Rapton too. Therefore, users can easily use raptin for scanning input.

while: Reserved token for while loop. "While" name is consensus for most of the language. Thus, we also want to add it to our language.

for: Reserved token for "for". It is easy to write and remember since it is a common token for most programming languages.

if: Reserved token for if statements.

else: Reserved token for else statements.

return: Reserved token for “return” statement which occurs in the body of the function. We add return statements to make functions more convenient and usable.

true: Reserved token for “true” boolean value.

false: Reserved token for “false” boolean value.

heading: Reserved token for heading attribute of the drone. We add this token as it belongs to a drone. As every drone keeps heading value as an integer between 0-359 we need to reach it and with this token, we can reach it easily.

altitude: Reserved token for altitude attribute of the drone.

droneX and droneY: Reserved tokens for x and y points of the drone’s position. It is convenient for finding the drone’s position over the area.

surface: Reserved token for Surface type identifier. We decided to give a predefined new type called surface. As our drone flies above some defined areas, the surface data type will be useful for users.

sCenterX,sCenterY, surfaceHeight, surfaceWidth: Reserved tokens for surface object’s predefined properties. As every surface had to have width and height, these tokens are useful to initialize them. Also helps with standardization.

sgetHeight, sgetWidth, sgetCenterX, sgetCenterY: Reserved tokens for surface object’s predefined functions. Users can use these functions to reach properties.

getHeading, getAltitude, getTemp: Reserved tokens for built-in functions of the drone. By using these predefined functions users can reach current information about the drone.

headingLeft, headingRight, climbUp, dropDown: Reserved tokens for built-in functions of the drone to control its movement over the area. All of these functions take one parameter called int distance. By considering that parameter, the drone moves.

connectWifi,nozzleOnOf: Reserved tokens for built-in functions of the drone for controlling its connection and spraying.

mathPow: Reserved name for built in math power function. Users can use this function to calculate square or square root of some integer values.

2. Types of Statements

2.1 Declaration Statement

<declaration_stmt> ::= <type_identifier><identifier><end_stmt>

In Rapton, there are multiple ways of creating a variable. You can directly create a variable and initialize it or can declare and assign some value later. Although this declaration gives flexibility, it causes an error if the user tries to use the declared variable without initialization.

2.2 Initialization Statement

<init_stmt> ::= <type_identifier><identifier><assign_op><term><end_stmt> | <surface_init_stmt>

This assignment statement is very similar to the C-type language assignment statements as it contains a type identifier.

❖ int b = 5; char c = 'w'

2.3 Assignment Statement

<assign_stmt> ::= <identifier><assign_op><term><end_stmt>

2.3 Return Statement

<return_stmt> ::= <return><space><id><end_stmt>

This non-terminal is created to be used inside of the functions. In Rapton language, there is no void return type. Users have to return something inside the function. It is space sensitive, it requires space after the <return>.

2.4 If Statement

The if statement is defined above in Section 1.3.1 and 1.3.2.

2.5 Struct Statement

<struct_stmt> ::=<struct><space><identifier><LCB><stmts><RCB><end_stmt>

This statement is for creating struct in Rapton language. Structs are useful and convenient to store data. With this struct definition, users can build their primitive objects. In this way, they can use the Rapton language for their purposes easily. They can create what they need and then use it too. Struct statements can contain statements inside of them. Users can declare variables and statements inside this struct assignment statement. Then, they have a chance to create the same objects with different variable values.

```
❖ Struct s1{
      String color;
      int number;
  };
```

Inside of the struct, we use only declaration statements, not assignment or any other statements.

2.6 Array Statement

<array_stmt>::=<type_identifier><space><identifier><dimension><end_stmt>

Rapton language provides users to perform operations on the collection of data items by using an array structure. All of the items in the array must be the same type, in other words homogenous.

<dimension> ::= <dimension><LSB><int_value><RSB> |<LSB><int_value><RSB>

In Rapton language you can use arrays as a primitive data type. More than that you can declare 2D, 3D arrays by considering this non-terminal definition. Therefore, this non-terminal consists of a recursive definition to allow choosing the dimension type.

2.7 Input Statement

<input_stmt> ::= <raptin><LP><type_identifier><identifier><RP><end_stmt>
Special input scanner statement to read user's input for a specified variable type. Our input reader

2.8 Output Statement

<output_stmt> ::= <raptout><LP><id><RP><end_stmt>

Rapout is the Rapton language's output statement. Rapout statement is simple to understand and it's readability is good. Users can output a idenitifer or value by putting it inside the brackets.

2.9 Surface Initialization Statement

**<surface_init_stmt>
::<type_identifier><identifier><assign_op><surface_term><end_stmt>**

Surface is an predefined non-primitive data type for Rapton language to ease user's jobs when their drone needs to spray a rectangular or square area.

**<surface_term> ::=
<LP><surface_width><assign_op><id><comma><surface_height><assign_op><id><comma><surface_centerX><assign_op><id><comma><surface_centerY><assign_op> <id><RP> // Surface surf=(sWidth=5,sHeight=11,sCenterX=0,sCenterY=0);**

2.10 Function Statements

<call_function_stmt> ::= <call><arrow><func_name><func_param_list><end_stmt>

| <callp><arrow><predefined_funcs><func_param_list><end_stmt>

Call function statement is used for calling our created functions. To simplify the function calling statement, Rapton language uses the "call->" keyword to call a created function. Functions are called by using their unique name.

**<declare_function_stmt>::=
<function><identifier><func_param_list><RCB><stmts><LCB>**

<func_name> ::= <identifier>

The “function” name is used for creating a unique named function. Rapton language also calls the function by using its name. Every function name should be unique, overriding a function is not allowed. Functions prevent unnecessary coding and repetition of the same code. Instead, it acts as a shortcut for the user by providing the same functioning code in the function’s body.

<predefined_funcs> ::= |<get_heading>|<get_altitude>|<move_forward>|<move_backward>|<get_temp> |<nozzle_on_of>|<stop_drone>|<heading_left>|<heading_right>|<climb_up>|<drop_down>| <surface_get_height>|<surface_get_width>|<surface_get_centerX>|<surface_get_centerY>

Rapton language has lots of predefined functions for drone’s movement or its situation. Users can call these functions by using specific non-terminal callp instead of using call which is a non-terminal for calling non-predefined functions.

<func_param_list> ::= <LP><param_context><RP>

This non-terminal is created for passing or declaring multiple variables inside the function. Thus, <param_context> is written recursively. <func_param_list> is just an organizer statement to make the BNF more clear.

<param_context> ::= |<identifier><comma><param_context> | <identifier>

| <type_identifier><identifier><comma><param_context>

| <type_identifier><identifier>

Param context is our parameter part of the function. Param context can either be blank, have one or multiple parameters(identifiers). Parameters help functions to perform their functioning properly and accordingly.

2.11 Comment

<comment> ::= <begin_comment><sentence><end_comment>

The syntax for comments in Rapton language is identified as the “begin_comment” keyword which is “#” After that, the user can write the comment. When a user wants to end the comment, the user shall use “end_comment” which is “!#”.

<begin_comment> ::= <HTS>

<end_comment> ::= <EXM><HTS><end_stmt>

3. Precedence, Associativity

1. <logic_expr> ::= <id> | <id><space><comparison_expr><space><id>
2. <comparision_expr>::= <and>|<or>|<equality_identifier>
3. <equals_op>|<greater_op>|<lesser_op>|<notEquals_op>|<greaterOrEqual_op> |<lesserOrEqual_op>
4. <type_identifier>::= <int><space>|<float><space>|<char><space> |<string><space>|<bool><space>|<surface><space>
5. <identifier> ::= <letter> | <identifier><digit> |<identifier><letter>| <heading>|<altitude>|<drone_x>|<drone_y>
6. <term> ::= <factor> | <term> + <id> | <term> - <id> | <id>
7. <factor> ::= <factor> * <id> | <factor> / <id> | <id>
8. <id> ::= <identifier> | <value>
9. <value> ::= <int_value> | <float_value> | <string_value> | <char_value>| <bool_value>
10. <int_value> ::= <number><digit> | <digit>
11. <string_value> ::= <string_identifier><word><string_identifier>
12. <char_value> ::= <char_identifier><letter><char_identifier>
13. <bool_value> ::= <true_value> | <false_value> | 0 | 1
14. <true_value>::= <true>| <not><false>
15. <false_value>::= <false>| <not><true>
16. <sentence> ::= <identifier><space> | <identifier>
17. <word> ::= <letter><word> | <letter>

4. Loops

<loop_stmt> ::= <while_stmt> | <for_stmt>

This non-terminal helps Rapton language users to create loops.

4.1 For Loop

<for_stmt> ::= <for><LP><int><space><identifier><space><in><space><id><arrow><id><space><step><space><arith_oprt><space><id><RP><LCB><stmts> <RCB>

This statement is used for defining the for loops in Rapton language. Rapton language adds ->(arrow) feature on top of C type of for loops to help users understand the for loop's condition and update according to it. Not only can we update our loop variable by increasing it, but Rapton language also makes it possible to update by using multiplication and division. A few examples are shown below:

```
for(int i in 1->50 step + 1){

}

for(int i in 1->25 step * i){

}
```

4.2 While Loop

<while_stmt> ::= <while><LP><logic_expr><RP><LCB><stmts><RCB>

In Rapton language there are two types of the loop; for and while. For loops are mostly used in integer comparison or if we want to make certain operations at a certain time. However, the loop maintains its process while its condition is satisfied. Mostly in while loops, the number of steps is not determined clearly as for loops. As we want to do some different operations inside the while, we need some bridge to <matched_stmt> or <unmatched_stmt>. We can use the <stmts> non-terminal as a bridge in this case.

Readability of Rapton

The readability of a language is a capability of a programming language to be understood by its targeted audience and users. Readable programs tend to require less maintenance. Rapton language considers this readability and tries to come up with an easily understandable syntax design. Rapton language's function calling syntax prioritizes readability. In order to call a function, you have to write it as call-> (your function name). This way the user can clearly see whether they called the function or not. Rapton language's "for loop" uses a very readable syntax. for(int i in 1->50 step + 2):

Condition for staying in the for loop is the 1->50 part.

Rapton language believes that using arrow(->) is unconventional but the most readable representation of "staying in the loop representation".

Writability of Rapton

The writability of a language is a measure of how easily a language can be used to create programs for a chosen problem domain. Writable programming languages tend to gain more popularity and will be used more frequently. Rapton language is not specifically designed to be written easily. Since sometimes making languages too much writable, can damage the language in different ways. However, in the Rapton language, logical operators such as “and” and “or” are written just like in English. In Rapton language also most of the syntax is carefully designed to be short. In the Rapton language, there is also a difference for loop syntax created by considering Kotlin for loop. However, there are some built-in methods and variables to make writing harder in Rapton. But they are also required for well-functioning code writing. Therefore, we designed Rapton by considering two simple things; It has to be easy for beginners but it also has to be for experienced and expert coders.

Reliability of Rapton

Reliability. In general, think of it as the ability of a program to comply with its specifications, under all conditions. We try to design Rapton as reliable as we can. We use type checking for this reliability since run-time checking is expensive and not desired. Instead, we want to check types on compile-time just like Rapton does. We also design the for loops and logical operations to make space sensitive to make the language more reliable. When we are making our language more reliable though we have to sacrifice the writability.

Unresolved and Resolved Conflicts in Rapton

We had a problem about our comment prompt and it still causes problems when we type more than one string value for the command line. Although, our lex's comment definition contains {string_value}*, comment line gives an error for multiple strings belonging to the same comment. Also, we had to add end_stmt (;) at the end of the comment. Otherwise, it is still reads the next line and causes an error.

For instance;

```
int a = 5;

#comment!#

int b = 11;
```

In this example, if we don't have semicolon to end comment line, #comment!# becomes

the part of the coming line int b = 11; and creates syntax error. As there is not a possible

assignment statement like <comment><type_identifier><identifier><assign_op><id><end_stmt>.

Another example that we solved is about operator precedence statements. We provide operator precedence for multiplication and division operators with term and factor statements.

However, when these two statements looks like in yacc file;

```
term: factor | term PLUS id | term MINUS id | id

factor: factor MULTIPLY id | factor DIVIDE id | id
```

It gives a shift/reduce error. As we can reach the same element id from two different paths.

Therefore, we change it as:

term: factor | term PLUS id | term MINUS id

factor | factor MULTIPLY id | factor DIVIDE id | id

Appendix

A. Test Codes

Test 1

```
begin
main(){
    bool isConnect = true;
    callp->connectWifi(isConnect);
    int droneX = 0; #initializer!#;
    int heading = 0;
    int altitude = 100;
    int droneY = 0;
    Surface s = (surfaceWidth = 8, surfaceHeight = 8, sCenterX = 0, sCenterY = 0);
    int k = callp->sgetCenterX();
    int w = callp->sgetWidth();
    int l = callp->sgetCenterY();
    int h = callp->sgetHeight();
    bool check = true;
    callp->nozzleOnOff(check);
    k = w / 2 + k;
    l = h / 2 + l;
    droneX = -1 * k;
    droneY = -1 * l;
    int j = 0;
    altitude = altitude + 100;
    callp->climbUp(altitude);
```

```
        for(int i in 0 -> w step + 1){
            j = 0;
            callp->moveRight(); #moveOneUnitRight!#;
            while(j < h){
                callp->moveForward(); #moveOneUnitForward!#;
                j = j + 1;
            }
            heading = heading + 180;
        }
        int ground = 0;
        callp->dropDown(ground);
        callp->stopDrone();
        raptout(a);
    }
    end
```

Test2

```
begin

function calc(int a, int b, int c){
    a = 10; #onesideofthetriangle!#;
    b = 10; #anotherside!#;
    c = 16; #thirdsideoftrianglearea!#;
    return c;
}

main(){
    int droneX = 0; #xPositionOfdroneinitialized!#;
    int droneY = 0; #yPositionOfdroneinitialized!#;
    int baseLength;
    int side1;
    int side2;
    int baseLength = call->calc(side1, side2, baseLength);
    int altitude = 150; #changingPredefinedVariablesValue!#;
    callp->climbUp(altitude);
    int heightHelper = baseLength / 2;
    int squareExp = 2; #expConstFormathPowFunct !#;
    float squareRoot = 0.5;  #expConstFormathPowFunct !#;
    int height = callp->mathPow(heightHelper, squareExp);
    #hypotenuseCalcwithSquareingSide1!#;
    if(side1 == side2){
       int hypotenuse = callp->mathPow(side1, squareExp);
    }
    else{
       height = 10;
    }
    height = hypotenuse - height;
    height = callp->mathPow(height, squareRoot);
    int k = 0;
    int heading = 90;
    bool flag = true;
    callp->connectWifi(flag);
    callp->nozzleOnOff(flag);
    for(int i in 0 -> height step + 1){
       for(int j in 0 -> baseLength - k step + 1){
          callp->moveRight();
       }
       if(k < baseLength){
          k = k + 1;
```

```
            }
            callp->moveForward();
        }
        bool stopper = false;
        callp->stopDrone();
        callp->connectWifi(stopper);
        if(heading == 0){
            raptout("STOPPED");
        }
        else{
            raptout("MOVING");
        }
    }
    end
```

B. Lex File

```
/* drone.l */
%option yylineno
NL        \n
LSB    \[
RSB    \]
LCB  \{
RCB \}
HTS    #
EXM    !
LP    \(
RP    \)
divide \/
comma    ,
end_stmt    \;
assign_operator     =
beginning begin
callpredefined callp
end end
main main
if    if
else    else
function    function
call call
for     for
in    in
step     step
while     while
and    and
or    or
arrow    ->
string_identifier    \"
char_identifier    \'
equals_op     ==
greater_op     >
lesser_op       <
not     {EXM}
notEquals_op    {not}{assign_operator}
greaterOrEqual     {greater_op}{assign_operator}
lesserOrEqual_op    {lesser_op}{assign_operator}
arith_op       [+-/*]
const       const
bool_type bool
bool     (true|false|0|1)
true             true
```

false	false
int	int
number	[1-9]
digit	[0-9]
int_value	[+-]?({digit}+)
float	float
float_value	[+-]?({digit})+(\.)({digit}*)
char	char
string	string
string_value	([A-Za-z])
raptin	raptin
raptout	raptout
comment	{HTS}{string_value}{EXM}{HTS}
drone_x	droneX
drone_y	droneY
struct	struct
surface	Surface
altitude	altitude
heading	heading
surface_height	surfaceHeight
surface_width	surfaceWidth
surface_centerX	sCenterX
surface_centerY	sCenterY
surface_get_height	sgetHeight
surface_get_width	sgetWidth
surface_get_centerX	sgetCenterX
surface_get_centerY	sgetCenterY
get_heading	getHeading
get_altitude	getAltitude
get_temp	getTemp
nozzle_on_off	nozzleOnOff

```
heading_left        headingLeft
heading_right       headingRight
stop_drone       stopDrone
move_right                      moveRight
move_left                       moveLeft
move_forward        moveForward
move_backward                   moveBackward
climb_up          climbUp
drop_down        dropDown
power_function       mathPow
connect_wifi           connectWifi
return          return
%%
[ \t]       ;
{divide}    return(DIVIDE);
{bool_type}    return(BOOL);
{callpredefined} return(CALL_PREDEFINED);
{LSB}     return(LSB);
{RSB}     return(RSB);
{LCB}             return(LCB);
{RCB}             return(RCB);
{LP}              return(LP);
{RP}              return(RP);
{comma}   return(COMMA);
{beginning}   return(BEGINNING);
{end}     return(END);
{main}    return(MAIN);
{end_stmt}  return(END_STATEMENT);
{assign_operator}    return(ASSIGNOP);
{if}  return(IF);
{else}          return(ELSE );
```

```
{function}            return(FUNCTION_IDENTIFIER);
{in}          return(IN);
{step}                return(STEP);
{while}               return(WHILE);
{and}                 return(AND);
{or}                  return(OR);
{arrow}               return(ARROW);
{return}              return(RETURN);
{string_identifier}   return(STRING_IDENTIFIER);
{char_identifier}     return(CHAR_IDENTIFIER);
{equals_op}           return(EQUALS_OP);
{greater_op}          return(GREATER_THAN);
{greaterOrEqual}  return(GREATER_OR_EQUAL);
{lesser_op}           return(LESS_THAN);
{notEquals_op}        return(NOT_EQUALS);
{lesserOrEqual_op}            return(LESS_OR_EQUAL);
{const}       return(CONST_TYPE);
{true}                 return(TRUE);
{false}                return(FALSE);
{float}                return(FLOAT_TYPE);
{char}                 return(CHAR_TYPE);
{string}            return(STRING_TYPE);
{int}         return(INTEGER_TYPE);
{raptin}      return(RAPTIN);
{raptout}     return(RAPTOUT);
{power_function} return(MATH_POW);
{int_value}           return(INTEGER_VALUE);
{float_value}         return(FLOAT_VALUE);
{drone_x}           return(DRONE_X);
{drone_y}           return(DRONE_Y);
{struct}            return(STRUCT);
```

```
{surface}              return(SURFACE);
{altitude}             return(ALTITUDE);
{heading}              return(HEADING);
{surface_height}                   return(SURFACE_HEIGHT);
{surface_width}                       return(SURFACE_WIDTH);
{surface_centerX}                  return(SURFACE_CENTER_X);
{surface_centerY}                  return(SURFACE_CENTER_Y);
{surface_get_height}          return(SGET_HEIGHT);
{surface_get_width}           return(SGET_WIDTH);
{surface_get_centerX}         return(SGET_CENTER_X);
{surface_get_centerY}         return(SGET_CENTER_Y);
{call}                             return(CALL_FUNCTION);
{get_heading}        return(FUNCTION_GET_HEADING);
{get_altitude}       return(FUNCTION_GET_ALTITUDE);
{get_temp}           return(FUNCTION_GET_TEMP);
{nozzle_on_off}      return(FUNCTION_NOZZLE_ON_OFF);
{heading_left}       return(FUNCTION_HEADING_LEFT);
{heading_right}      return(FUNCTION_HEADING_RIGHT);
{stop_drone}         return(FUNCTION_STOP_DRONE);
{move_left}                   return(FUNCTION_MOVE_LEFT);
{move_right}                  return(FUNCTION_MOVE_RIGHT);
{move_backward}               return(FUNCTION_MOVE_BACKWARD);
{move_forward}       return(FUNCTION_MOVE_FORWARD);
{climb_up}           return(FUNCTION_CLIMB_UP);
{drop_down}          return(FUNCTION_DROP_DOWN);
{connect_wifi}       return(FUNCTION_CONNECT_TO_WIFI);
{for}         return(FOR);


\+            return(PLUS);
\-            return(MINUS);
\*         return(MULTIPLY);
```

```
{comment}               return(COMMENT);

{string_value}+{digit}*         return(GENERAL_IDENTIFIER);

[ \t\n]           ;

%%

int yywrap(){

      return 1;

}
```

C. Yacc File

```
/* drone.y */

%token LSB

%token RSB

%token LCB

%token RCB

%token LP

%token RP

%token COMMA

%token END_STATEMENT

%token IF

%token ELSE

%token FUNCTION_IDENTIFIER

%token IN

%token STEP

%token WHILE

%token AND

%token OR

%token ARROW

%token RETURN

%token STRING_IDENTIFIER

%token CHAR_IDENTIFIER

%token EQUALS_OP

%token GREATER_THAN
```

```
%token LESS_THAN

%token NOT_EQUALS

%token LESS_OR_EQUAL

%token CONST_TYPE

%token TRUE

%token FALSE

%token FLOAT_TYPE

%token CHAR_TYPE

%token STRING_TYPE

%token INTEGER_TYPE

%token RAPTIN

%token RAPTOUT

%token INTEGER_VALUE

%token FLOAT_VALUE

%token DRONE_X

%token DRONE_Y

%token STRUCT

%token SURFACE

%token ALTITUDE

%token HEADING

%token SURFACE_HEIGHT

%token SURFACE_WIDTH

%token SURFACE_CENTER_X

%token SURFACE_CENTER_Y

%token SGET_HEIGHT

%token SGET_WIDTH

%token SGET_CENTER_X

%token SGET_CENTER_Y

%token CALL_FUNCTION

%token FUNCTION_GET_HEADING

%token FUNCTION_GET_ALTITUDE
```

```
%token FUNCTION_GET_TEMP

%token FUNCTION_NOZZLE_ON_OFF

%token FUNCTION_HEADING_LEFT

%token FUNCTION_HEADING_RIGHT

%token FUNCTION_STOP_DRONE

%token FUNCTION_MOVE_FORWARD

%token FUNCTION_CLIMB_UP

%token FUNCTION_DROP_DOWN

%token FUNCTION_CONNECT_TO_WIFI

%token FOR

%token PLUS

%token MINUS

%token NL

%token COMMENT

%token GENERAL_IDENTIFIER

%token UNDEFINED

%token BEGINNING

%token END

%token MAIN

%token ASSIGNOP

%token GREATER_OR_EQUAL

%token BOOL

%token DIVIDE

%token MULTIPLY

%token CALL_PREDEFINED

%token FUNCTION_MOVE_BACKWARD

%token FUNCTION_MOVE_LEFT

%token FUNCTION_MOVE_RIGHT

%token MATH_POW

%%
```

```
program: BEGINNING declare_function main END {printf("Input program is valid\n"); return 0;}

main: MAIN LP RP LCB stmts RCB

stmts: stmt | stmts stmt

stmt: unmatched | matched

matched: IF LP logic_expr RP LCB matched RCB ELSE LCB matched RCB
        | init_stmt
        | declaration_stmt
        | call_function
        | return
        | comment
        | struct
        | loop
        | array
        | output
        | input

unmatched: IF LP logic_expr RP LCB matched RCB
        | IF LP logic_expr RP LCB unmatched RCB
        | IF LP logic_expr RP LCB matched RCB ELSE LCB unmatched RCB

return: RETURN id END_STATEMENT

input: RAPTIN LP type_identifier GENERAL_IDENTIFIER RP END_STATEMENT

init_stmt: type_identifier identifier ASSIGNOP term END_STATEMENT
        | type_identifier identifier ASSIGNOP values END_STATEMENT
        | surface_init_stmt
        | type_identifier identifier ASSIGNOP call_function

values: TRUE
        | FALSE
        | FLOAT_VALUE

call_function: CALL_FUNCTION ARROW func_name func_param END_STATEMENT
        | CALL_PREDEFINED ARROW predefined_funcs func_param END_STATEMENT

predefined_funcs: FUNCTION_GET_HEADING
        | FUNCTION_GET_ALTITUDE
        | FUNCTION_MOVE_FORWARD
        | FUNCTION_HEADING_LEFT
        | FUNCTION_HEADING_RIGHT
        | FUNCTION_NOZZLE_ON_OFF
        | FUNCTION_GET_TEMP
        | FUNCTION_CLIMB_UP
        | FUNCTION_DROP_DOWN
        | FUNCTION_CONNECT_TO_WIFI
        | SGET_HEIGHT
        | SGET_WIDTH
        | SGET_CENTER_Y
        | SGET_CENTER_X
        | FUNCTION_MOVE_BACKWARD
        | FUNCTION_MOVE_RIGHT
        | FUNCTION_MOVE_LEFT
        | FUNCTION_STOP_DRONE
        | MATH_POW

declaration_stmt: type_identifier identifier END_STATEMENT
        | CONST_TYPE type_identifier identifier END_STATEMENT
```

```
surface_init_stmt: SURFACE identifier ASSIGNOP surface_term END_STATEMENT

logic_expr: identifier
        | id comparison_expr id

comparison_expr: AND
        | OR
        | EQUALS_OP
        |  GREATER_THAN
        | LESS_THAN
        | LESS_OR_EQUAL
        | NOT_EQUALS
        | GREATER_OR_EQUAL

type_identifier:
        | FLOAT_TYPE
        | INTEGER_TYPE
        | STRING_TYPE
        | CHAR_TYPE
        | BOOL

identifier: GENERAL_IDENTIFIER
        | HEADING
        | ALTITUDE
        | DRONE_Y
        | DRONE_X

term:   factor
        | term PLUS id
        | term MINUS id
```

```
factor:
        | factor MULTIPLY id
        | factor DIVIDE id
        | id

id: identifier
        | INTEGER_VALUE
        | string_value

surface_term: LP SURFACE_WIDTH ASSIGNOP id COMMA SURFACE_HEIGHT ASSIGNOP id
COMMA SURFACE_CENTER_X ASSIGNOP id COMMA SURFACE_CENTER_Y ASSIGNOP id RP

struct: STRUCT GENERAL_IDENTIFIER LCB stmts RCB END_STATEMENT

array: type_identifier GENERAL_IDENTIFIER dimension END_STATEMENT

dimension: dimension LSB INTEGER_VALUE RSB
        | LSB INTEGER_VALUE RSB

string_value: STRING_IDENTIFIER GENERAL_IDENTIFIER STRING_IDENTIFIER

output: RAPTOUT LP id RP END_STATEMENT

func_name: identifier

func_param: LP param_context RP

param_context:   | identifier COMMA param_context
        | identifier
```

```
declare_function:
	| FUNCTION_IDENTIFIER GENERAL_IDENTIFIER LP func_param_declare RP LCB stmts RCB

func_param_declare:  | type_identifier identifier COMMA func_param_declare
	| type_identifier identifier

loop: while_stmt
	| for_stmt

while_stmt: WHILE LP logic_expr RP LCB stmts RCB

for_stmt: FOR LP INTEGER_TYPE GENERAL_IDENTIFIER IN INTEGER_VALUE ARROW
term STEP arith_op id RP LCB stmts RCB

arith_op: PLUS
	| MINUS
	| DIVIDE
	| MULTIPLY

comment: COMMENT

%%
#include "lex.yy.c"

int yyerror(char* s){
  fprintf(stderr, "%s on line %d\n",s, yylineno);
  return 1;
}

int main(){
  yyparse();
  return 0;
}
```