2020-2021 FALL SEMESTER

CS315 – PROGRAMMING LANGUAGES

TEAM: 34

PROJECT REPORT

TURNA

GROUP MEMBERS:

1) Burak Yetiştiren, 21802608, Section 1
2) Akın Parkan, 21703249, Section 1
3) Mustafa Hakan Kara, 21703317, Section 1

Table of Contents

1. BNF Description of Turna

<program> → <statements>

<statements> → <statement> SEMICOLON <statements> | <return_statement> SEMICOLON <statements> | COMMENT <statements> | <conditional_statement> <statements> | <loop_statement> <statements> | <function_declaration> <statements> | <empty>

<statement> → <const_declaration> | <variable_declaration> | <assignment_statement> | <function_call> | <empty>

<loop_statement> → <while_statement> | <for_statement>

<conditional_statement> → IF LP <expression> RP START <statements> END

| IF LP <expression> RP START <statements> END ELSE START <statements> END

<return_statement> → RETURN <expression>

<while_statement> → WHILE LP <expression> RP START <statements> END

<for_statement> → FOR LP <statement> SEMICOLON <expression> SEMICOLON <assignment_statement> RP START <statements> END

<argument> → <expression>

<expression_list> → <expression> COMMA <expression_list> | <expression>

<arguments> → <expression_list> | <empty>

<type> → TYPE_BOOL | TYPE_CHAR | TYPE_INT | TYPE_FLOAT | TYPE_STRING

<assignment_statement> → IDENTIFIER ASSIGN_OP <expression> | IDENTIFIER INCREMENT_OP | IDENTIFIER DECREMENT_OP

<const_declaration> → CONST <type> <declaration_list>

<variable_declaration> → <type> <declaration_list>

<declaration_list> → IDENTIFIER ASSIGN_OP <expression> COMMA <declaration_list> | IDENTIFIER COMMA <declaration_list> | IDENTIFIER ASSIGN_OP <expression> | IDENTIFIER

<expression>→ STRING | <or_expression>

<or_expression> → <or_expression> OR <and_expression> | <and_expression>

<and_expression> → <and_expression> AND <relational_expression> | <relational_expression>

<relational_expression> → <relational_expression> <relational_operator> <arithmetic_expression> | <arithmetic_expression>

<arithmetic_expression> → <arithmetic_expression> PLUS <term> | <arithmetic _expression> MINUS <term> | <term>

<term> → <term> MULTIPLICATION <factor> | <term> DIVISION <factor> | <term> MODULO <factor> | <factor> | NOT <factor>

<factor> → LP <or_expression> RP | <function_call> | IDENTIFIER | INTEGER | FLOAT | TRUE | FALSE | CHAR

<relational_operator> → EQUAL_OP | NOT_EQUAL_OP | LESS_THAN_OP | LESS_THAN_OR_EQUAL_OP | GREATER_THAN_OP | GREATER_THAN_OR_EQUAL_OP

<function_call> → <custom_function_call> | <primitive_function_call>

<custom_function_call> → IDENTIFIER LP <arguments> RP

<primitive_function_call> → <drone_function_call> | <print_function_call> | <scan_function_call>

<drone_function_call> → GET_INCL LP RP

| GET_ALTITUDE LP RP

| GET_SPEED LP RP

| GET_TEMP LP RP

| GET_ACCELERATION LP RP

| SET_ALTITUDE LP <argument> RP

| SET_SPEED LP <argument> RP

| SET_ACCELERATION LP <argument> RP

| SET_INCL LP <argument> RP

| TURN_ON_CAM LP RP

| TURN_OFF_CAM LP RP

| TAKE_PICT LP RP

| GET_CURRENT_TIME_STAMP LP RP

| CONNECT_WIFI LP <argument> COMMA <argument> RP

| TAKE_OFF LP RP

| LAND LP RP

| FLY_UP_WITH_DISTANCE LP <argument> RP

| FLY_DOWN_WITH_DISTANCE LP <argument> RP

| FLY_RIGHT_WITH_DISTANCE LP <argument> RP

| FLY_FORWARD_WITH_DISTANCE LP <argument> RP

| FLY_BACK_WITH_DISTANCE LP <argument> RP

| ROTATE_CW_WITH_ANGLE LP <argument> RP

| ROTATE_CCW_WITH_ANGLE LP <argument> RP

| FLIP_FORWARD LP RP

| FLIP_BACKWARD LP RP

| FLIP_LEFT LP RP

| FLIP_RIGHT LP RP

| HOVER_FOR_SECONDS LP <argument> RP

| YAW_RIGHT_WITH_DEGREES LP <argument> RP

| YAW_LEFT_WITH_DEGREES LP <argument> RP

<scan_function_call> → SCAN_INT LP RP | SCAN_STR LP RP | SCAN_FLOAT LP RP | SCAN_BOOL LP RP | SCAN_CHAR LP RP

<print_function_call> → PRINT LP <argument> RP | PRINT_NL LP RP | PRINT_NL LP <argument> RP

<parameter_list> → <type> IDENTIFIER COMMA <parameter_list> | <type> IDENTIFIER

<parameters> → <parameter_list> | <empty>

<function_type> → VOID | <type>

<function_declaration> → FUNC <function_type> IDENTIFIER LP <parameters> RP START <statements> END

<empty> → ε

2. Language Constructs

<program> The program consists of a set of statements.

<statements> Statements can consist of a statement followed by a semicolon and statements; statements can be preceded by a comment. Types of statements include conditional, loop, function declaration statements. Statements can also be empty.

<statement> Statement can be a constant or variable declaration, an assignment statement, a function call, or it can be empty.

<loop_statement> Collects the two types of loop statements present in the language: For statement and while statement. These statements are basically loops, those the programmers will use for executing a certain part of their code one or more times.

<conditional_statement> This construct embraces certain statements with start, and end terminals. Execution of the code is controlled by <expression> embraced in parenthesis. If the logic expression is true, then the embraced statements are executed. If not either the conditional statement is abandoned, or if the conditional statement has the else part that part is executed. Hence, in the language conditional statements can be written in two ways:

```
if(getAltitude() lt 10)
start
  setAltitude(10);
end

OR

if(getSpeed() lt 2)
start
  setSpeed(3);
end
else
start
  setSpeed(getSpeed()--);
end
```

<return_statement> Statement used to return required value or identifier from a function.

<while_statement> While statement is controlled by the language construct <expression>. As long as the logic expression is true statements embraced with the start, and end terminals will be executed. For example:

```
while( getHeight() gt 3)
start
  setHeight( getHeight()--);
end
```

<for_statement> For statement is controlled by the language construct <expression> and driven by the assignment statement. The programmer has to declare a variable first, define a logic expression and drive the loop with the assignment statement. Then the executed statement must be written between the start, and end terminals. For example:

```
for( type_int = 0; i lte 15; i++)
start
 flipForward();
end
```

<argument> Every argument is an expression. It only exists for clarification purposes.

<expression_list> Expression list is either only one expression or a set of expressions separated by commas.

<arguments> Arguments can either be empty, or it can be an expression list.

<type> Type construct accumulates the integer, char, float, string, and boolean variable type names.

<assignment_statement> Used to assign desired values to variables or constants. There are also increment (++) and decrement (--) operators as assignment statements. For example:

```
type_float degree = PI * 0.75;
```

```
degree++;
time--;
```

<const_decleration> Structures the declaration of constants. For example:

```
const type_int PI = 3.14;
```

<variable_declaration> Structures the declaration of the variables. For example:

```
type_int numberOfTakeoffs;
type_int numberOfLandings  = 0;
```

<declaration_list> Used to declare a set of variables or constants of the same type in one statement at once, by separating them with commas.

```
type_int distance = 10, speed, acceleration = 2;
```

<expression> An expression is either a string or an or expression. It encompasses relational and arithmetic expressions, function calls, identifiers, and combination of these.

For example, the following are valid expressions in Turna:

```
"Turna"

type_int exp = 'T' + a * 5 - !(6 < 7) && foo() || false;
```

<or_expression> An or expression is either an And expression or an expression that is obtained by combining an or expression and an and expression with an OR operation.

<and_expression> An and expression is either a relational expression or an expression that is obtained by combining an and expression and a relational expression with an AND operation.

<relational_expression> A relational expression is either an arithmetic expression or an expression that is obtained by combining a relational and an arithmetic expression with a relational operator.

<arithmetic_expression> An arithmetic expression is either a term or an expression that is obtained by combining an and expression and a term with an ADDITION or SUBTRACTION operation.

<term> A term is either a factor or a factor preceded by a NOT operator or an expression that is obtained by combining a term and a factor with a MULTIPLICATION, DIVISION or MODULO operation.

<factor> A factor is either an or expression inside parenthesis or one of the following:

- An integer, float, boolean or character literal
- An identifier
- A function call

<relational_operator> Cumulates all the logical operators in the language: equal, not equal, less than, less than or equal, greater than, greater than or equal operators

<function_call> This language construct merges two types of function calls: custom function calls and primitive function calls.

<custom_function_call> Defines the structure of the custom function calls. An identifier is followed by a <expression> in parenthesis. For example:

```
name = getPilotName();
```

<primitive_function_call> Used to group three types of primitive function calls. They are drone, scan, and print function calls.

<drone_function_call> Cumulates the primitive function calls regarding the drone such as, getAltitude(), getIncl(), turnOnCam(), takePict().

<scan_function_call> Cumulates the primitive scan functions. They are: scanInt(), scanStr, scanFloat() ,scanBool(), scanChar()

<print_function_call> Used to gather two primitive print functions print and print_nl.

```
print("Flight completed.");
```

```
print_nl("Flight initialized.");
print_nl();
```

<parameter_list> Is used for declaration of custom function parameters separated by commas. It can contain more than one parameter or multiple parameters.

<parameters> Delineates the parameters of functions. Parameters are followed by a parameter. Either one, multiple or no parameters can be present. Parameters are separated by commas. For example:

```
type_int numberOfLoops, type_float degrees
```

<function_type> Defines the types of custom functions. In addition to variable types functions can also be void type.

<function_decleration> Defines the structure of the user defined function declarations. Firstly the reserved word func and function type are written. Then the identifier and the parameters are provided. The statements of the function are embraced in start and end terminals. An example is:

```
func type_bool getVerification( type_int altitude, type_int speed)
start
  if( speed lte 10 and altitude eq 5)
  start
     return true;
  end
  else
  start
      return false;
  end;
end;
```

<empty> → ε

3. Nontrivial Tokens

Our language does not contain any tokens to specify the beginning and the end of the execution. The code is directly executed from the first statement written by the programmer until the end. So, it can be said that the structure of Turna is like the famous programming language Python.

3.1. Comments

Comments in Turna have two implementations. Programmers can use single and multi-line comments. Comments that are enclosed in "{" and "}" are multi-line comments. On the other hand, comments those initialize with "#" and end with "\n" are single line comments. We wanted programmers to be able to have multi-line comments as well as single line comments so that the writability of Turna would increase.

3.2. Identifiers

Identifiers in Turna must start with a set of characters. They are any letter in English alphabet (A-Za-z), or the dollar sign ("$"), or the underscore character ("_"). For the following characters, programmers can choose any initials or digits they wish to use. We have introduced initials; our ambition was to make the language more reliable. On the negative side, the language became less writable, because the programmers have to remember the rules for writing initializers.

3.3. Literals

The rules we have determined to represent the literals in Turna followed the conventions used in most of the C group languages. We believe that the rules are readable because they follow similar rules, when compared to the mathematical terminology, also the conventions that are being used in natural languages too. On the other hand, when writability is considered, because we have followed the natural ways to write the literals and there is approximately one way to write them, Turna is writable as well.

- **true** and f**alse** literals are boolean literals that are representing the true and false statements in logic.
- **Character** literals are enclosed between two apostrophe (‘) characters.
- **String** literals are enclosed between two quotation marks (“), similar to C group languages.
- **Integer** literals are similar to the most programming languages. They can have signs before them, if they are negative integers, they must have a minus sign (-) before them.

- **Float** literals can be positive and negative or neutral. If they are between 0 and 1 the programmer can put the decimal point without writing 0 in the beginning: .9 for 0.9.

3.4. Reserved words

While we were selecting the reserved words in Turna, our main ambition was to make them readable. Because of this ambition, we have chosen to use the abbreviation of some boolean operators. This has decreased the writability for some accounts, that the programmer would have learned both ways to write such operators. We want to emphasize the primitive functions we have defined for Turna. We believe that the primitive functions implemented, help the drone programmer to program his/her drone more conveniently with Turna.

Types

- **type_int** - used to precede the identifier for integer type
- **type_float** - used to precede the identifier for float type
- **type_string** - used to precede the identifier for string type
- **type_char** - used to precede the identifier for character type
- **type_bool** - used to precede the identifier for boolean type

Function-related

- **return** - used to return variables from the functions
- **func -** used at the beginning of function declarations

Output

- **print** - used to print the inputs in one terminal line
- **print_nl** - used to print the inputs in one terminal line and skip to the next terminal line

Conditional operators

- **if** - used to initialize the conditional statements, precedes the logic expression in parenthesis
- **else** - precedes the reserved word start, that precedes statements for the conditional operator

Loops

- **for** - reserved word for for loops. After using this word programmer must continue with providing the required parts for the for loop

- **while** - reserved word for while loops, which precedes the logical expression that must be satisfied to enter the loop

Boundary statements

- **start** - reserved word for specifying the beginning of the code block of conditional and loop statements
- **end** - reserved word for specifying the ending of the code block of conditional and loop statements

Logical operators

- **eq -** equivalent of the terminal “==” checks if the given elements are the same
- **neq -** equivalent of the terminal “!=” checks if the given elements are different
- **and -** equivalent of the terminal “&&” checks if the given logical statements are both true
- **or -** equivalent of the terminal “||” checks if one of the given elements is true
- **lt -** equivalent of the terminal “<” checks if the given element is less than the other element
- **lte -** equivalent of the terminal “<=” checks if the given element is less than or equal to the other element
- **gt -** equivalent of the terminal “>” checks if the given element is greater than the other element
- **gte -** equivalent of the terminal “>=” checks if the given element is greater than or equal to the other element

Primitive functions

- **getIncl** - reserved word for specifying the primitive function for getting the inclination of the drone.
- **getAltitude** - reserved word for specifying the primitive function for getting the altitude of the drone
- **getSpeed** - reserved word for specifying the primitive function for getting the speed of the drone
- **getTemp** - reserved word for specifying the primitive function for getting the temperature of the drone
- **getAcceleration** - reserved word for specifying the primitive function for getting the acceleration of the drone

- **setIncl** - reserved word for specifying the primitive function for setting the inclination of the drone.
- **setAltitude** - reserved word for specifying the primitive function for setting the altitude of the drone.
- **setSpeed** - reserved word for specifying the primitive function for setting the speed of the drone.
- **setAcceleration** - reserved word for specifying the primitive function for setting the acceleration of the drone.
- **turnOnCam** - reserved word for specifying the primitive function for turning on the camera of the drone
- **turnOffCam** - reserved word for specifying the primitive function for turning off the drone
- **takePict** - reserved word for specifying the primitive function for taking a picture with camera of the drone
- **getCurrentTimeStamp** - reserved word for specifying the primitive function for getting the current timestamp of the drone
- **connectWifi** - reserved word for specifying the primitive function for connecting the drone to a Wi-Fi network.
- **takeOff** - reserved word for specifying the primitive function for taking off for the drone
- **land** - reserved word for specifying the primitive function for landing for the drone
- **flyUpWithDistance** - reserved word for specifying the primitive function for flying up with a distance of the drone
- **flyDownWithDistance** - reserved word for specifying the primitive function for flying down with a distance of the drone
- **flyLeftWithDistance** - reserved word for specifying the primitive function for flying left with a distance of the drone
- **flyRightWithDistance** - reserved word for specifying the primitive function for flying right with a distance of the drone
- **flyForwardWithDistance** - reserved word for specifying the primitive function for flying forward with a distance of the drone
- **flyBackWithDistance** - reserved word for specifying the primitive function for flying back with a distance of the drone
- **rotateCwWithAngle** - reserved word for specifying the primitive function for rotating the drone clockwise with given angle

- **rotateCcwWithAngle** - reserved word for specifying the primitive function for rotating the drone counterclockwise with given angle

- **flipForward** - reserved word for specifying the primitive function for flipping the drone forward

- **flipBackward** - reserved word for specifying the primitive function for flipping the drone backward

- **flipLeft** - reserved word for specifying the primitive function for flipping the drone left

- **flipRight** - reserved word for specifying the primitive function for flipping the drone right

- **hoverForSeconds** - reserved word for specifying the primitive function for making the drone stay in the hover for user defined duration

- **yawRightWithDegrees** - reserved word for specifying the primitive function for yawing the drone right with user defined degrees

- **yawLeftWithDegrees** - reserved word for specifying the primitive function for yawing the drone left with user defined degrees

- **scanInt** - reserved word for specifying the primitive function for scanning the integer the programmer will provide

- **scanStr** - reserved word for specifying the primitive function for scanning the string the programmer will provide

- **scanFloat** - reserved word for specifying the primitive function for scanning the float the programmer will provide

- **scanBool** - reserved word for specifying the primitive function for scanning the boolean the programmer will provide

- **scanChar** - reserved word for specifying the primitive function for scanning the character the programmer will provide

4. Language Evaluation

4.1. Readability

Readability of a language is important because a readable language provides programmers of a language to be familiarized easily, with that learning cost of the language would be decreased. For that purpose, we try to increase our language's readability as much as we could, while not compromising writability of our language as much as possible. Reserved words of our language are not much, and they cannot be used as variable names or user defined function names. These reserved words include primitive function names related to Drone programming since we are designing a language for drones. So having less reserved words and not letting programmers use them for defining variables or functions increase the readability of our language. Our language supports output functions similar to Python language as print() and print_nl(), and input functions similar to Java which are scanInt(), scanBool() etc. with that, it makes our language easier for the programmers to familiarize. Moreover our language includes single line and multi-line commands, it also increases our language's readability. Since our language requires start and end reserved words for loops' and conditional statements' for specifying the code blocks of the loop statement or conditional statement; our language's readability is increased since it is easier to follow the statements with specified code blocks. Our language will not allow programmers to overload operators, with that readability of the language increases. In contrast to python and JavaScript, our language has data types, with that also the readability increases. Lastly, in contrast to c, our language includes boolean type. With that ambiguity related to understanding if the boolean type holds 0/1 or true/false we solve this problem by specifying bool data type to hold true\false.

4.2. Writability

Writability of a language is a measurement which can be specified as how easy programmers can code by using a language. Therefore, we also aim to have a language with high writability as much as we can by supporting expressivity and simplicity. Our language supports expressivity by providing programmers to use different sets of loops which can be specified as for, while loops and also providing ++ and -- operators for integer types of the language in addition to classic increment operator (count = count +1), another thing that supports expressivity can be supported by multiple declaration on a single line, which is supported by our language. Since our language has a few primitive types it is a simple language. Since there is a trade-off between readability and writability and while forcing the usage of start - end reserved words for loops and conditionals and not supporting operator overloading increase the readability of the language, due to readability - writability trade-off language's writability decreases.

4.3. Reliability

In order to make our programming language as reliable as possible, we have done the following: Since variable types have different sizes in programming languages we provide the variable types as primitive types and require programmers to use variable types while defining the variables, and through the primitive input functions we specify the input type with scanInt(), scanBool(), scanFloat() etc. and also we force programmers to obey parameter types of functions (type checking) otherwise lexical analyzer will not supply the expected tokens. In addition, with following hierarchy we provide the operator precedence in our language:

- Expressions inside the parenthesis
- NOT operator
- Multiplication, division, and modulo operators
- Addition, and subtraction operators
- Relational operators
- AND operator
- OR operator

5. Lex of the Language

```
digit   [0-9]
sign    [+-]
integer {sign}?{digit}+
float {sign}?{digit}*(\.)?{digit}+
letter  [a-zA-Z]
char    \'[^']\'
string    \"[^"]*\"
initial ({letter}|$|_)
identifier {initial}({initial}|{digit})*
literal ({integer}|{float}|{char}|{string})

func "func"
call "call"
get_incl "getIncl"
get_altitude "getAltitude"
get_speed "getSpeed"
get_temp "getTemp"
get_acceleration "getAcceleration"
set_incl "setIncl"
set_altitude "setAltitude"
set_speed "setSpeed"
set_temp "setTemp"
set_acceleration "setAcceleration"
turn_on_cam "turnOnCam"
turn_off_cam "turnOffCam"
take_pict "takePict"
get_current_time_stamp "getCurrentTimeStamp"
connect_wifi "connectWifi"
take_off "takeOff"
land "land"
```

```
fly_up_with_distance "flyUpWithDistance"
fly_down_with_distance "flyDownWithDistance"
fly_left_with_distance "flyLeftWithDistance"
fly_right_with_distance "flyRightWithDistance"
fly_forward_with_distance "flyForwardWithDistance"
fly_back_with_distance "flyBackWithDistance"
rotate_cw_with_angle "rotateCwWithAngle"
rotate_ccw_with_angle "rotateCcwWithAngle"
flip_forward "flipForward"
flip_backward "flipBackward"
flip_left "flipLeft"
flip_right "flipRight"
hover_for_seconds "hoverForSeconds"
yaw_right_with_degrees "yawRightWithDegrees"
yaw_left_with_degrees "yawLeftWithDegrees"
hover "hover"
scan_int "scanInt"
scan_str "scanStr"
scan_float "scanFloat"
scan_bool "scanBool"
scan_char "scanChar"
print "print"
print_nl "print_nl"

comment "#".*|"{"[^\}]*"}"

assign_op "="
semicolon ";"
incrmnt_op "++"
decrmnt_op "--"

true "true"
false "false"
if "if"
else "else"
while "while"
for "for"
return "return"
start "start"
end "end"

lp "("
rp ")"
comma ","

const "const"
type_bool "type_bool"
type_int "type_int"
type_float "type_float"
type_char "type_char"
type_string "type_string"
void "void"
```

```
plus "+"
minus "-"
mul "*"
div "/"
mod "%"

eq_op (==|eq)
neq_op (!=|neq)
lt_op (<|lt)
lte_op (<=|lte)
gt_op (>|gt)
gte_op (>=|gte)
not_op "!"
and (&&|and)
or (\|\||or)

%option yylineno
%%
{comment} return(COMMENT);
{func} return(FUNC);
{call} return(CALL);
{lp} return(LP);
{rp} return(RP);
{get_incl} return(GET_INCL);
{get_altitude} return(GET_ALTITUDE);
{get_temp} return(GET_TEMP);
{get_speed} return(GET_SPEED);
{get_acceleration} return(GET_ACCELERATION);
{set_incl} return(SET_INCL);
{set_altitude} return(SET_ALTITUDE);
{set_temp} return(SET_TEMP);
{set_speed} return(SET_SPEED);
```

```
{set_acceleration} return(SET_ACCELERATION);
{turn_on_cam} return(TURN_ON_CAM);
{turn_off_cam} return(TURN_OFF_CAM);
{take_pict} return(TAKE_PICT);
{get_current_time_stamp} return(GET_CURRENT_TIME_STAMP);
{connect_wifi} return(CONNECT_WIFI);
{take_off} return(TAKE_OFF);
{land} return(LAND);
{fly_up_with_distance} return(FLY_UP_WITH_DISTANCE);
{fly_down_with_distance} return(FLY_DOWN_WITH_DISTANCE);
{fly_left_with_distance} return(FLY_LEFT_WITH_DISTANCE);
{fly_right_with_distance} return(FLY_RIGHT_WITH_DISTANCE);
{fly_forward_with_distance} return(FLY_FORWARD_WITH_DISTANCE);
{fly_back_with_distance} return(FLY_BACK_WITH_DISTANCE);
{rotate_cw_with_angle} return(ROTATE_CW_WITH_ANGLE);
{rotate_ccw_with_angle} return(ROTATE_CCW_WITH_ANGLE);
{flip_forward} return(FLIP_FORWARD);
{flip_backward} return(FLIP_BACKWARD);
{flip_left} return(FLIP_LEFT);
{flip_right} return(FLIP_RIGHT);
{hover_for_seconds} return(HOVER_FOR_SECONDS);
{yaw_right_with_degrees} return(YAW_RIGHT_WITH_DEGREES);
{yaw_left_with_degrees} return(YAW_LEFT_WITH_DEGREES);
{hover} return(HOVER);
{scan_int} return(SCAN_INT);
{scan_str} return(SCAN_STR);
{scan_float} return(SCAN_FLOAT);
{scan_bool} return(SCAN_BOOL);
{scan_char} return(SCAN_CHAR);
{print} return(PRINT);
{print_nl} return(PRINT_NL);
{semicolon} return(SEMICOLON);
```

```
{comma} return(COMMA);
{assign_op} return(ASSIGN_OP);
{incrmnt_op} return(INCREMENT_OP);
{decrmnt_op} return(DECREMENT_OP);
{eq_op} return(EQUAL_OP);
{neq_op} return(NOT_EQUAL_OP);
{lt_op} return(LESS_THAN_OP);
{lte_op} return(LESS_THAN_OR_EQUAL_OP);
{gt_op} return(GREATER_THAN_OP);
{gte_op} return(GREATER_THAN_OR_EQUAL_OP);
{not_op} return(NOT);
{and} return(AND);
{or} return(OR);
{const} return(CONST);
{type_bool} return(TYPE_BOOL);
{type_char} return(TYPE_CHAR);
{type_int} return(TYPE_INT);
{type_float} return(TYPE_FLOAT);
{type_string} return(TYPE_STRING);
{void} return(VOID);
{true} return(TRUE);
{false} return(FALSE);
{while} return(WHILE);
{for} return(FOR);
{start} return(START);
{end} return(END);
{if} return(IF);
{else} return(ELSE);
{return} return(RETURN);
{integer} return(INTEGER);
{float} return(FLOAT);
{identifier} return(IDENTIFIER);
```

```
{char} return(CHAR);
{string} return(STRING);
{plus} return(PLUS);
{minus} return(MINUS);
{mul} return(MULTIPLICATION);
{div} return(DIVISION);
{mod} return(MODULO);
{rbrace} return(RBRACE);
"\n"  { }
%%
int yywrap() { return 1; }
```

6. Yacc Description of the Language

```
%token COMMENT FUNC CALL GET_INCL GET_ALTITUDE GET_TEMP GET_SPEED
GET_ACCELERATION SET_INCL SET_ALTITUDE SET_TEMP SET_SPEED
SET_ACCELERATION TURN_ON_CAM TURN_OFF_CAM TAKE_PICT
GET_CURRENT_TIME_STAMP CONNECT_WIFI TAKE_OFF LAND
FLY_UP_WITH_DISTANCE FLY_DOWN_WITH_DISTANCE
FLY_LEFT_WITH_DISTANCE FLY_RIGHT_WITH_DISTANCE
FLY_FORWARD_WITH_DISTANCE FLY_BACK_WITH_DISTANCE
ROTATE_CW_WITH_ANGLE ROTATE_CCW_WITH_ANGLE FLIP_FORWARD
FLIP_BACKWARD FLIP_LEFT FLIP_RIGHT HOVER_FOR_SECONDS
YAW_RIGHT_WITH_DEGREES YAW_LEFT_WITH_DEGREES HOVER SCAN_INT
SCAN_STR SCAN_FLOAT SCAN_BOOL SCAN_CHAR PRINT PRINT_NL SEMICOLON
COMMA ASSIGN_OP INCREMENT_OP DECREMENT_OP EQUAL_OP
NOT_EQUAL_OP LESS_THAN_OP LESS_THAN_OR_EQUAL_OP
GREATER_THAN_OP GREATER_THAN_OR_EQUAL_OP NOT AND OR CONST
TYPE_BOOL TYPE_CHAR TYPE_INT TYPE_FLOAT TYPE_STRING VOID TRUE
FALSE WHILE FOR START END IF ELSE RETURN INTEGER FLOAT IDENTIFIER
CHAR STRING LP RP PLUS MINUS MULTIPLICATION DIVISION MODULO RBRACE

%%

program: statements

statements: statement SEMICOLON statements | return_statement SEMICOLON
statements | COMMENT statements | conditional_statement statements | loop_statement
statements | function_declaration statements | empty;

statement: const_declaration | variable_declaration | assignment_statement |
function_call | empty;

loop_statement: while_statement | for_statement;

conditional_statement: IF LP expression RP START statements END

| IF LP expression RP START statements END ELSE START statements END;

return_statement: RETURN expression;

while_statement: WHILE LP expression RP START statements END;

for_statement: FOR LP statement SEMICOLON expression SEMICOLON
assignment_statement RP START statements END;

argument: expression;

expression_list: expression COMMA expression_list | expression;

arguments: expression_list | empty;

type: TYPE_BOOL | TYPE_CHAR | TYPE_INT | TYPE_FLOAT | TYPE_STRING;

assignment_statement: IDENTIFIER ASSIGN_OP expression | IDENTIFIER
INCREMENT_OP | IDENTIFIER DECREMENT_OP;

const_declaration: CONST type declaration_list;
```

variable_declaration: type declaration_list;

declaration_list: IDENTIFIER ASSIGN_OP expression COMMA declaration_list | IDENTIFIER COMMA declaration_list | IDENTIFIER ASSIGN_OP expression | IDENTIFIER;

expression: STRING | or_expression;

or_expression: or_expression OR and_expression | and_expression;

and_expression: and_expression AND relational_expression | relational_expression;

relational_expression: relational_expression relational_operator arithmetic_expression | arithmetic_expression;

arithmetic_expression: arithmetic_expression PLUS term | arithmetic_expression MINUS term | term;

term: term MULTIPLICATION factor | term DIVISION factor | term MODULO factor | factor | NOT factor;

factor: LP or_expression RP | function_call | IDENTIFIER | INTEGER | FLOAT | TRUE | FALSE | CHAR;

relational_operator: EQUAL_OP | NOT_EQUAL_OP | LESS_THAN_OP | LESS_THAN_OR_EQUAL_OP | GREATER_THAN_OP | GREATER_THAN_OR_EQUAL_OP;

function_call: custom_function_call | primitive_function_call;

custom_function_call: IDENTIFIER LP arguments RP;

primitive_function_call: drone_function_call | print_function_call | scan_function_call;

drone_function_call: GET_INCL LP RP | GET_ALTITUDE LP RP | GET_SPEED LP RP | GET_TEMP LP RP | GET_ACCELERATION LP RP | SET_ALTITUDE LP argument RP | SET_SPEED LP argument RP | SET_ACCELERATION LP argument RP | SET_INCL LP argument RP | TURN_ON_CAM LP RP | TURN_OFF_CAM LP RP | TAKE_PICT LP RP | GET_CURRENT_TIME_STAMP LP RP | CONNECT_WIFI LP argument COMMA argument RP | TAKE_OFF LP RP | LAND LP RP | FLY_UP_WITH_DISTANCE LP argument RP | FLY_DOWN_WITH_DISTANCE LP argument RP | FLY_RIGHT_WITH_DISTANCE LP argument RP | FLY_FORWARD_WITH_DISTANCE LP argument RP | FLY_BACK_WITH_DISTANCE LP argument RP | ROTATE_CW_WITH_ANGLE LP argument RP | ROTATE_CCW_WITH_ANGLE LP argument RP | FLIP_FORWARD LP RP | FLIP_BACKWARD LP RP | FLIP_LEFT LP RP | FLIP_RIGHT LP RP | HOVER_FOR_SECONDS LP argument RP | YAW_LEFT_WITH_DEGREES LP argument RP | YAW_RIGHT_WITH_DEGREES LP argument RP;

scan_function_call: SCAN_INT LP RP | SCAN_STR LP RP | SCAN_FLOAT LP RP | SCAN_BOOL LP RP | SCAN_CHAR LP RP;

print_function_call: PRINT LP argument RP | PRINT_NL LP RP | PRINT_NL LP argument RP;

parameter_list: type IDENTIFIER COMMA parameter_list | type IDENTIFIER

```
parameters: parameter_list | empty;

function_type: VOID | type;

function_declaration: FUNC function_type IDENTIFIER LP parameters RP START
statements END;

empty: ;

%%

#include "lex.yy.c"

main() {


if(yyparse() == 0)

    printf("Your program is syntactically correct.\n");

return 0;

}

int yyerror( char *s ) {

fprintf(stderr,"There is a syntax error at line %d\n", yylineno);

}
```

7. Example Programs

7.1. Example Flight Scenario

```
{
    An example flight scenerio written in Turna.
}
type_string wifiName = scanStr();
type_string wifiPassword = scanStr();
connectWifi(wifiName, wifiPassword);
print_nl(getCurrentTimeStamp());

turnOnCam();
takeOff();
type_int count = 0;
while(count lt 10)
start
        flipLeft();
        takePict();
        flipRight();
        takePict();
        rotateCwWithAngle(30.0);
        takePict();
        rotateCwWithAngle(30.0);
        takePict();
        if(getAcceleration() gt 10.0)
        start
                setAcceleration(5.0);
        end
        a++;
end
```

```
func void flipAndFly(type_float altitude)
start
	flipForward();
	flyForwardWithDistance(5);
	flipBack();
	flyBackWithDistance(3);
	print_nl(altitude);
end

for(type_int a = 10; a lte 2; a--)
start
	flipAndFly(getAltitude());
end

func type_bool hoverAndYaw()
start

	hoverForSeconds(60);
	yawRightWithDegrees(180.0);
	yawLeftWithDegrees(90.0);

	if(getTemp() gt 30.5)
	start
		return true;
	end
	return false;

end
```

```
if(hoverAndYaw())
start
	print_nl("Overheating, please land.");
end


func void setFlightParameters(type_float inclination, type_float altitude, type_float speed, type_float acceleration)
start
	setIncl(inclination);
	setAltitude(altitude);
	setSpeed(speed);
	setAcceleration(acceleration);
end


setFlightParameters(45.0, 12.3, 6.2, 2.7);


land();
turnOffCam();
print_nl("Goodbye!");
```

7.2. Other Test Programs

```
{
	Other test programs:
}
#Constant & variable declarations
const type_int A = 5, B = 10;
const type_bool ALWAYS_TRUE = true;
const type_float PI = 3.14;
const type_char FIRST_LETTER = 'T';
```

```
const type_string NAME_OF_THE_LANGUAGE = "TURNA";


type_int a;
a = 4;
type_int b = 3;
type_int c = 2, d = 1, e;

type_bool f = true, g = false, h, i, j;
type_float k = 1.5;
type_char ch = 'T';
type_string str = "TURNA";

e = (a + b) * c / d - a;

#Boolean operations
h = f && g;
h = !h and (a < 4 || b <= 3 or c > 1 || c >= 2);
i = a lt 4 or b lte 3 or c gt 1 or c gte 2;
j = h == i and h eq i or h != i and h neq i;
#Expressions
type_int exp = 'T' + a * 5 - !(6 < 7) && foo() || false;

#Example conditional statements
if(e gt a) start
 if(!i or j)
 start
          h = false;
 end
end
```

```
#IO
type_int $int_in = scanInt();
type_string $str_in = scanStr();
type_float $float_in = scanFloat();
type_bool $bool_in = scanBool();
type_char $char_in = scanChar();

print("The integer input by the user is: ");
print_nl($int_in);

if($str_in lt 5)
start
	print_nl("The integer input by the user is less than 5");
end

#LOOPS
print("Please enter an integer: ");
type_int limit = scanInt();
type_int count = 0;

while(count < limit)
start
	print_nl(count);
	count++;
end
for(type_int x = 0, y = limit; x lt limit; x++) start y--; end
```

```
#Function Declerations & Calls

func void my_function() start print_nl("Hello"); end
func type_int my_function2() start return 0; end
func type_bool my_function3() start return true; end
func type_float my_function4() start return 1.5; end
func type_char my_function5() start return 'T'; end
func type_string my_function6() start return "TURNA"; end



func type_int pow(type_int x, type_int y) start

const type_int FIRST_X = x;

for(type_int i = 0; i lt y; i++) start x = x * FIRST_X; end

end

print("Please enter the base: ");
type_int base = scanInt();

print("Please enter the exponent: ");
type_int exp = scanInt();

print("The result is: ");
print_nl(pow(base, exp));
```

```
#Comments

#This is a one-line comment

{

    This is a multi-line comment

}

{
{{ Even if there are more than one left curly braces, we won't get any errors. }
```