CS315 PROGRAMMING LANGUAGES
PROJECT 2

JETCOT

Team 32

Asuman Aydın, 21502604, Section 02

Metehan Gürbüz, 21602687, Section 02

Yusup Alpdemir, 21803035, Section 02

Fall, 2020

TABLE OF CONTENT

1 BNF - Revised

< program > → **begin** < functions > **end**
| <empty>

< functions > → < functions > < function > | < function >

<function> → <ident_type> <function_name>[<parameter list>]: <body>

<function_name> → <letters>

<func_start> → <function_name> **[** <parameter_list>**]**

<parameter_list> → <ident_type> <parameter>, <parameter_list>
|<ident_type> <parameter>
| <empty>

<parameter> → <variable>

<variable> → <letters> | **Const**

<body> → <stmts> **this** <variable>

<stmts> → <stmt>
| <stmt> <stmts>
| $<comments>$ <stmts>
| <stmts> $<comments>

<stmt> → <matched> | <unmatched>

<matched> → if <log_stmts>: <matched> else: <matched>

| <non-if_stmts>

<unmatched> → if <log_stmts>: <stmt>

| if <log_stmts>: <matched> else: <unmatched>

<non-if_stmts> → <assignment>
| <primitive_func>
| <while-loop>
| <for>
| <print>

<comments> → <comment>
| <comment> <comments>
| <empty>

<comment> → all characters except $

<primitive_func> → <inclination>
| <altitude>
| <temperature>
| <acceleration>
| <camera-on>
| <camera-off>
| <picture>
| <capture>
| <connect>
| <time>
| <get-timer>

<while-loop> → **while** <log_stmts>: <stmts>
| **while** <comparison>: <stmts>

<for> → **for** <ident_type> <variable> **;** <comparison> **;** <assignment>: <stmts>

<log_stmts> → <log_stmts> **or** <and_cond>
| <and_cond>

<and_cond> → <and_cond> **and** <not_cond>
| <not_cond>

<not_cond> → **not** <log_stmts>
| <log_stmts>

```
<assignment> → <ident_type> <var> = <log_stmts>
                    |<ident_type>  <var> = <math_expr>
                    | <ident_type> <var> = <var><dot><func_start>
                    |<ident_type>  <var> = <dronic>
                    | <var> = <var>
                    | <var>  = <log_stmts>
                    | <var> = <math_expr>
                    | <var> = <var><dot><func_start>

<comparison> → <var> > <var>
                    | <dronic> > <dronic>
                    | <var> < <var>
                    | <dronic>  < <dronic>
                    | <var> >=  <var>
                    | <dronic> >= <dronic>
                    | <var> <=  <var>
                    | <dronic> <= <dronic>
                    | <var> == <var>
                    | <dronic> == <dronic>
                    | <var> == <var>
<var> → <letters>

<letters> → <letter>
             | <letter><letters>

<dot> → .

<math_expr> → <math_expr> + <term>
                    | <math_expr> - <term>
                    | <term>

<term> → <term> * <factor>
                    | <term> | <factor>
                    | <factor>

<factor> →
          | + <factor>
          | - <factor>
```

<inclination> → **readIncline**[]

<altitude> → **readAltitude**[]:

<temperature> → **readTemperature**[]:

<acceleration> → **readAcceleration**[]:

<camera-on> → **turnOnCamera**[<variable>]:

<camera-off> → **turnOffCamera**[<variable>]:

<picture> → <ident_type> **takePicture**[<variable>]:

<capture> → <ident_type> **capture**[]:

<connect> → <ident_type> **connect**[<variable>, <variable>]:

<time> → <ident_type> **readTime**[<variable>]:

<get-timer> → <ident_type> **getTimer**[<variable>]:

<print> → **print[<dronic>]**
| print[<log_stmts>]
| print[<primitive_func>]
| print[<variable>]

<input> →

<dronic> → <string>
| <float>
| <int>
| <boolean>

<string> → "<letters>" | "<empty>"

<float> → <digit><int>.<digit><int> | <digit>.<digit><int>

<int> → <digit><int> | <digit>

<ident_type> → **int | boolean | float | string | nada | Drone | Timer |** <empty>

<boolean> → **true | false**

<digit> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

<letter> → **a | b | ... | z | A | B | ... | Z**

<empty> →

1.1 EXPLANATION OF BNF - Revised

Revision:

- Addition of comparison operators to bnf and lex.
- Elimination of unnecessary definitions on BNF
- Some matching problems fixed
- Logical Expression recursion problem is solved.

JETCOT is a simple programming language for everyone to program their drone anywhere any time. If you are familiar with python, this language first gives the impression of simplicity of Python programming language. However, it also contains the definitive beginnings of variables and functions. We choose to keep identifiers since it will diminish the compile errors and runtime errors. It will ease the error checking in run-time.

Our language requires primitive functions to program the drone and contains a main function to give users the freedom to program their drone in the way they want. JETCOT also requires "**begin-end**" statements to start and finish the program. Main function is also included in the process of begin and end. The language does not allow global statements in the program. This is because we want to keep things short and simple for functionality matters. If we do not have variable declaration complexity, users will use less variables to define the program. Also, if they want to program their drone, only one instance of variables are required as it can be seen in example programs.

We implement the general mathematical expression knowledge from Java and Python. However, we do not allow increment with only one side of assignment operation. It will decrease the complexity but also give some restrictions to the language. We think these kinds of restrictions keep the language in a better shape. We have string, float, int etc. in <dronic> definition. Dronic is a group of variable types which are mostly used during programming a drone. Users can define a function and can declare a dronic variable inside of functions. Also, the main method allows the user to declare a variable and use it to call the functions.

There are no much reserved words to eliminate unnecessary, unused definitions. Everything with definitive types are defined under their distinct types. This will give the language a reliability since it mostly eliminates the aliasing.

Our language does not require an endpoint for the statements like dots. However, ":" is needed to proceed the if, while, function statements. This is for distinction criteria and it increases the readability because it distinguishes the body and the main statements.

<program>: This is a start statement for the program. It should start with begin and it should end with end. Also, it can be empty. Between begin and end, programs can contain only functions. Users should program their drone inside begin and end statements. These increase the readability and the writability of the code since the programmer needs to program the drone only within one scope. This also increases the reliability since there is not much complexity between the statements.

<functions>: Functions are used to determine what to do with a program. Checking incline of drone and taking pictures of the camera capture are some of the primitive functions defined under the function definition. Functions take parameters with their identified types. Each parameter has its own definitive type. There are no restrictions on the parameter types. Users can either return a variable or function can be nada which means there are no return statements. Return statement is defined with "this" word which indicates that this function returns this variable. Functions can only return one variable at a time. It will ease the debugging of the program in the sense of complexity and steps.

<function>: functions contain parameters, types and body statements. Bod

<function_name>: Consists of letters. This is used to name the functions created.

<parameter_list>: Parameter list is a list of different variables to pass to the function. There are no limits on the number of parameters.Also there can be empty parameter lists which function does not need any variable pass.

<parameter>: Parameter is a variable with identified type. There are different types such as Drone, Timer, int etc. It is passed to the function to use inside of the function.

<body>: Body of the function is a block of statements with different types. Body is important in terms of writability. It gives optimal scale to the language.

<stmts>: Statements are used to make the drone take defined actions. Statements are used for any physical action taken by the drone since they are a part of functions. Examples of statements are if statements, while loops, assignments, primitive functions, and printing operations.

<stmt>: A statement is either a matched statement or an unmatched statement.

<matched>: A matched statement can be an if statement or a non-if statement. In order to combat the ambiguity which occurs when there are more if statements inside an if statement, matched and unmatched are introduced. Matched refers to the statement inside the if statement being either a matched statement in itself or a statement that does not contain an else, so that ambiguity does not occur.

<unmatched>: An if statement being unmatched means that the statement in the if statement is already matched and its else is unmatched, or the statement simply does not have an else.

<non-if_stmts>: Non if statements consist of assignment operations, loops, printing operation, and primitive functions such as turning the camera on and off and checking the time. This was mainly added to solve the ambiguity in if statements.

<comment>: Comment is for a programmer to write anything between two "$" symbols for any reason without influencing how the code works.

<comments>: This is a collection of comments for writing multiple comments in a row.

<primitive_func> In the description of the requirements, we had a couple of primitive functions which are must functions to implement this programming language. These functions must be used in terms of the drone functionalities.

<while-loop>: A while loop, as long as the result of the logical statement is true, runs a statement until the result of the logical statement is false.

<log_stmts> Logical expressions are a must of a programming language. As our language is based on logic of a drone, and, or not logic are expressed in terms of logical expressions. Logical expressions are left recursive expressions and in terms of our language or have more precedence over and and not.

<and_cond>: An and condition is one of the logical expressions which returns if the statement is true in two conditions or not. And condition has higher precedence over the not condition.

<not_cond>: Not condition is a logical expression which indicates the reverse of the boolean value of the statement of variable.

<assignment>: The assignment expression which assigns the value of the right-hand side of the "=" symbol, to the left-hand side.

<var>: Variables are used to determine definitive objects of Drone like time, incline number etc.

<letters>: Letters consisting of letters which is simply one thing from the alphabet.

<func_start>: The function call which is needed for the get or set functions on variables of drone. After dot you can simply call any function with parameters.

<dot>: It is used to call a function inside the body of a function.

<math_expr>: General mathematical expression used for addition, multiplication, division, subtraction.

<high_pre>: Mathematical expression with higher precedence.

<inclination>: Primitive function for reading the inclination. Takes no parameters and returns an integer.

<altitude>: Primitive function for reading the altitude. Takes no parameters and returns a double.

<temperature>: Primitive function for reading the temperature. Takes no parameters and returns an integer.

<acceleration>: Primitive function for reading the acceleration. Takes no parameters and returns an integer.

<camera-on>: Primitive function for turning the camera on. Takes a boolean parameter and has no return value.

<camera-off>: Primitive function for turning the camera off. Takes a boolean parameter and has no return value.

<picture>: Primitive function for taking pictures with the camera. Takes a boolean parameter and has no return value.

<capture>: Primitive function for capturing video footage with the camera. Takes no parameters and returns a drone object.

<connect>: Primitive function for connecting the drone to the base computer through wi-fi connection. Takes a string and a boolean for parameters, has no return value. It checks whether the password entered is correct or not, then proceeds to the connection.
<time>: ReadTime function is a primitive function which reads the timestamp of the function and then returns the time. There function call to the getTimer function of Drone Object.
<get-timer>: Simply returns the time variable of Drone Object.
<print>: The function simply returns the string between the quotation mark.
<dronic>: There are multiple variable types. In the sense of this project, we thought "dronic" word may define the meaning of the variables used in the program. String, float, integer etc are some of the dronic variables defined.
<string>: Words within the quote marks. These consist of letters.
<float<: Numbers which contains digits with points to use to declare the altitude or incline of the drone.
<int>: Numbers which contain digits or digits.
<low_pre_op>: Addition and subtraction have lower precedence over the division and the multiplication.
<high_pre_op>: Division and multiplication have higher precedence over the addition and subtraction.
<ident_type>: The type is used to define the functions and the variable. As the python seems to not use such variable types, we consider the compile time errors. We want to eliminate any difficult process during the programming of a drone for the users.
<boolean>: It is a logical variable declaration. It is either true or false.
<digit>: It contains the natural numbers.
<letter>: It contains the alphabet.
<empty>: It is simply an empty statement. Total Blank.

2 Lex

```
integer [0-9]+
letters [a-zA-Z]+
float [0-9]+\.[0-9]+
%{
int lineCounter = 1;
%}
%%
Drone           return(DRONE );
begin           return(PROGRAM_BEGIN );
end             return(PROGRAM_END );
Timer           return(TIMER );
string          return(TYPE_STRING );
float           return(TYPE_FLOAT );
int             return(TYPE_INT );
boolean         return(TYPE_BOOL );
const           return(CONST );
this            return(RETURN );
if              return(IF );
else            return(ELSE );
true            return(TRUE );
false           return(FALSE );
while           return(WHILE_LOOP );
for                                   return(FOR_LOOP );
print           return(PRINT_STATEMENT );
or              return(OR );
and             return(AND );
not             return(NOT );
nada            return(VOID_FUNC );
wifiPassword    return(WİFİ_IDENTIFIER );
readIncline     return(INCLINE_FUNC );
readAltitude    return(ALTITUDE_FUNC );
readAcceleration return(ACCEL_FUNC );
readTemperature  return(TEMPR_FUNC );
turnOnCamera    return(ON_FUNC );
turnOffCamera   return(OFF_FUNC );
```

```
takePicture     return(PICTURE_FUNC );
capture         return(CAPTURE_FUNC );
connect         return(CONNECT_FUNC );
main            return(MAIN_FUNC );
readTime        return(TIME_FUNC );
getTimer        return(TIMER_FUNC );
{letters}       return(VARIABLE );
{integer}       return(INTEGER );
\,              return(COMMA );
\.              return(BELONGS );
\+              return(ADD );
\-              return(SUBTRACT );
\|              return(DIVIDE );
\#              return(MULTIPLY );
\(              return(LP );
\)              return(RP );
\{              return(LC );
\}              return(RC );
\[              return(LSQ );
\:              return(END_STATEMENT );
\]              return(RSQ );
\>              return(BIGGER_THAN );
\<              return(LESS_THAN );
\>=             return(BIGGER_EQUAL );
\<=             return(LESS_EQUAL );
\==             return(EQUAL );
\!=             return(NOT_EQUAL );
\;              return(SEMI_COL );
\=              return(ASGN_OP );
$.+\$           return(COMMENT );
\"              return(QUOTE);
\n              {lineCounter++;}
.;
%%
int yywrap(){return 1;}
```

3 Example Programs

```
$functions$
begin
int readIncline[]:
    int incline = x
    this incline
float readAltitude[]:
    this altitude
int readTemperature[]:
    this temperature
int readAcceleration[]:
    this acceleration
nada turnOnCamera[boolean cam]:
    cam = true
nada turnOffCamera[boolean cam]:
    cam = false
nada takePicture[boolean cam]:
    if cam == true:
       capture[]
    else:
       print("camera is off")
Drone capture[]:
    this picture
nada connect[string password, boolean connection]:
    if wifiPassword == password:
       connection = true
    else:
       connection = false
float readTime[Timer thisTimer]:
    float currentTime = thisTimer.getTimer[]
    this currentTime
nada main[]:
    Drone drone1
    string passwordEnter = "jetcot123"
    boolean connected = false
    connect[passwordEnter,connected]
    print("drone is connected to wifi")
    int count = 5
    while count < 10:
      count = count + 1
      readAcceleration[]
      readIncline[]
end
```

4 Yacc

```
%token DRONE PROGRAM_BEGIN PROGRAM_END TIMER TYPE_STRING
TYPE_FLOAT TYPE_INT TYPE_BOOL CONST RETURN IF ELSE TRUE FALSE
WHILE_LOOP FOR_LOOP PRINT_STATEMENT OR AND NOT VOID_FUNC
WIFI_IDENTIFIER INCLINE_FUNC ALTITUDE_FUNC ACCEL_FUNC TEMPR_FUNC
ON_FUNC OFF_FUNC PICTURE_FUNC CAPTURE_FUNC CONNECT_FUNC
MAIN_FUNC TIME_FUNC TIMER_FUNC FOR_LOOP VARIABLE INTEGER COMMA
BELONGS ADD SUBTRACT DIVIDE MULTIPLY LP RP LC RC LSQ
END_STATEMENT RSQ BIGGER_THAN LESS_THAN BIGGER_EQUAL
LESS_EQUAL EQUAL NOT_EQUAL  SEMI_COL ASGN_OP COMMENT QUOTE
%right '='
%left '+' '-'
%left '#"|'
%%
start: program
;
program:            PROGRAM_BEGIN functions PROGRAM_END
                    |
;
functions:          functions function | function
;
function:           ident_type function_name LSQ parameter_list RSQ
END_STATEMENT body
;
func_start:         function_name LSQ parameter_list RSQ
;
function_name:      VARIABLE
;
parameter_list:            ident_type parameter COMMA LESS_THAN parameter_list
                           BIGGER_THAN
| ident_type  parameter
|
;
```

```
parameter:          variable
;
variable:           VARIABLE | CONST
;
body:               stmts RETURN variable
;
stmts:              stmt
| stmt stmts
| COMMENT stmts
| stmts COMMENT
;
stmt:               matched | unmatched
;
matched:            IF log_stmts END_STATEMENT matched ELSE
END_STATEMENT                         matched

                    | non_if_stmts
;
unmatched:          IF log_stmts END_STATEMENT stmt
                    | IF log_stmts END_STATEMENT matched ELSE
END_STATEMENT                         unmatched
;
non_if_stmts:       assignment
| primitive_func
| WHILE_LOOP
| FOR_LOOP
| PRINT_STATEMENT
;
primitive_func:     inclination
                    | altitude
                    | temperature
                    | acceleration
                    | camera_on
                    | camera_off
                    | picture
                    | capture
                    | connect
                    | time
                    | get_timer
;
```

```
while_loop: WHILE log_stmts END_STATEMENT stmts
                | WHILE comparison END_STATEMENT stmts
;
for: FOR_LOOP ident_type var COMMA comparison COMMA assignment
END_STATEMENT stmts
;
inclination:        INCLINE_FUNC LSQ RSQ
;
altitude:           ALTITUDE_FUNC LSQ RSQ END_STATEMENT
;
temperature:        TEMPR_FUNC LSQ RSQ END_STATEMENT
;
acceleration:       ACCEL_FUNC LSQ RSQ END_STATEMENT
;
camera_on:          ON_FUNC LSQ variable RSQ END_STATEMENT
;
camera_off:         OFF_FUNC LSQ variable RSQ END_STATEMENT
;
picture:            ident_type PICTURE_FUNC LSQ variable RSQ
END_STATEMENT
;
capture:            ident_type CAPTURE_FUNC LSQ RSQ END_STATEMENT
;
connect:            ident_type CONNECT_FUNC LSQ variable COMMA variable RSQ
                    END_STATEMENT
;
time:               ident_type TIME_FUNC LSQ variable RSQ END_STATEMENT
;
get_timer:          ident_type TIMER_FUNC LSQ variable RSQ END_STATEMENT
;
log_stmts:          log_stmts OR and_cond
                    | and_cond
;
and_cond:           and_cond AND not_cond
                            | not_cond
;
not_cond:           NOT log_stmts
                            | log_stmts
;
assignment:         ident_type var ASGN_OP log_stmts
```

```
                        | ident_type var ASGN_OP math_expr
                  | ident_type var ASGN_OP var BELONGS func_start
                  | ident_type var ASGN_OP dronic
                        | var ASGN_OP var
| var  ASGN_OP log_stmts
| var ASGN_OP math_expr
                | var ASGN_OP VARIABLE BELONGS func_start
;
comparison: var BIGGER_THAN var
        | dronic BIGGER_THAN dronic
        |var LESS_THAN var
        | dronic LESS_THAN dronic
        |var BIGGER_EQUAL var
        | dronic BIGGER_EQUAL dronic
        |var LESS_EQUAL var
        | dronic LESS_EQUAL dronic
        |var EQUAL var
        | dronic EQUAL dronic
        |var NOT_EQUAL var
        | dronic NOT_EQUAL dronic
;
var:                    VARIABLE
;
math_expr:              math_expr ADD term
                        | math_expr SUBTRACT term
                        | term
;
term:                   term MULTIPLY factor
                        | term | factor
                        | factor
;
factor:
                        | ADD factor
                        | SUBTRACT factor
;
print:                  print LSQ dronic RSQ
                        | print LSQ log_stmts RSQ
                | print LSQ primitive_func RSQ
                        | print LSQ variable RSQ
;
```

```
input:
;
dronic:                       TYPE_STRING
                       | TYPE_FLOAT
                       | TYPE_INT
                       | TYPE_BOOL
;
ident_type:            TYPE_INT | TYPE_BOOL | TYPE_FLOAT | TYPE_STRING |
VOID_FUNC | DRONE | TIMER |
;
%%
#include "lex.yy.c"
extern int lineCounter;
int main(){
yyparse();
printf("Valid Input");
return 0;
}
int yyerror(const char *s){fprintf(stderr, "%s in line %d\n", s, lineCounter); }
```