VictorDronhen Language Specifications

Umut Utku Erşahince	**22202606**	**Sec 2**
Emir Görgülü	**22202834**	**Sec 2**
Berke Kuzey Ardıç	**22103340**	**Sec 2**

BNF Description:

```
<string>                    ::= "[^"]*"
<number>                    ::= [-+]?[0-9]*\.?[0-9]+
<identifier>                ::= [a-zA-Z_]+
<boolean>                   ::= True | False
<comment>                   ::= //.*
<comma_seperated_list>      ::= ε | <string> | <identifier> | <number> | <boolean>|
                                <number>,<comma_seperated_list> |
                                <identifier>,<comma_seperated_list> |
                                <string>,<comma_seperated_list> |
                                <boolean>,<comma_seperated_list>

<program>                   ::= <stmt_list>
<stmt_list>                 ::= <stmt> | <stmt> <stmt_list> | <comment> |
                                <comment>\n<stmt_list>
<stmt>                      ::= <assignment>; | <loop> | <conditional> | <function_def> |
                                <expression>;

<assignment>                ::= <identifier> = <expression>;
<loop>                      ::= while (<expression>){<stmt_list>}
<conditional>               ::= if (<expression>){<stmt_list>} |
                                if (<expression>){<stmt_list>} else{<stmt_list>}
<function_def>              ::= function <identifier>(<comma_seperated_list>){<stmt_list>}
<expression>                ::= <arithmetic_expression> | <relational_expression> |
                                <boolean_expression> | <function_call> | <identifier>

<function_call>             ::= <identifier>(<comma_seperated_list>)

<arithmetic_expression>     ::= <term> | <term> + <arithmetic_expression> |
                                <term> - <arithmetic_expression>
<term>                      ::=  <factor> | <factor> * <term> | <factor> / <term>
<factor>                    ::=  <number> | <identifier> | <function_call> | (<arithmetic_expression>) | ε

<relational_expression>     ::= <arithmetic_expression> “<” <arithmetic_expression>   |
                                <arithmetic_expression> “<=” <arithmetic_expression> |
                                <arithmetic_expression> “>” <arithmetic_expression>   |
                                <arithmetic_expression> “>=” <arithmetic_expression> |
                                <arithmetic_expression> “==” <arithmetic_expression> |
                                <arithmetic_expression> “!=” <arithmetic_expression> |
                                <arithmetic_expression>


<boolean_expression>        ::= <relational_expression> and <boolean_expression> |
                                <relational_expression> or <boolean_expression>    |
                                not <relational_expression> | <relational_expression>
```

Primitive Functions:

getHeading() : Returns a number, [0, 359], indicating the angle between north and heading direction. A return value of 1 indicates 1 degree deviation to the west
getAltitude() : Returns the altitude in meters
getTime() : Returns Unix time in seconds
sprayOn() : Turn the spray on
sprayOff() : Turn the spray off
climbUp() : Start climbing up at 0.25 m/s,
dropDown() : Start dropping at 0.25 m/s
stopVertical() : Stops vertical movement
stopHorizontal() : Stops horizontal movement
moveForward() : Start moving forward at 0.75 m/s
turnLeft() : Turn left 1 degree
turnRight() : Turn right 1 degree
print(variable) : Print to terminal, useful for displaying information. Adds a new line character
input() : Read input

Terminals:

<number> : The number literal represents either a floating point or an integer value. These are used for numerical calculations and arithmetic. They also support relational comparison.
<identifier> : These are used in naming variables as well as functions in the programming language. The variables can be either a number (an integer or a float), a string or a boolean.
<string> : String literals are used for representing a sequence of characters. They are used mostly for input / output purposes and are not directly related to the control mechanism of the drone.
<boolean> : Boolean literal, either True or False. Denotes a boolean value. Used for conditional statements.

Nontrivial Tokens:

<comment> : Comments are explanatory notes written in a separate line.
logical reserved : [and, or, not] are used to work with boolean content.
relational reserved : [<, >, <=, >=, ==, !=] used for comparison between number types.
arithmetic reserved : [+, -, /, *, (,)] used for arithmetic between number types.
primitive functions : See designated section above. We paid attention to readability and writability by giving very clear names.

Standard Library:

We enhanced the user experience by building a standard library of especially useful functions. This greatly increases the writability criterion of this language by providing essential functions for controlling drones. For the implementation of this library, see below:

STANDARD LIBRARY: (Every Program Has Access to These Functions)

```
// Turns right at a given angle. Works for negative angles.
function turnRightDegrees(degree){
      if (degree > 0) {
            j = 0;
            while (j < degree) {
                  turnRight();
                  j = j + 1;
            }
      }
      else{
            j = 0;
            while (j > degree) {
                  turnLeft();
                  j = j - 1;
            }
      }
}
```

```
// move forwards with the given distance and then stop. Works for negative dist.
function moveForwardDistance(distance) {
      startTime = getTime();
      necessaryTime = distance / 0.75;
      if (distance < 0) {
            necessaryTime = - necessaryTime;
            turnRightDegrees(180);
      }
      moveForward();
      while (getTime() - startTime < necessaryTime) {}
      stopHorizontal();

      // do not change rotation if we move backward.
      if (distance < 0) {
            turnRightDegrees(180);
      }
}
```

```
// moves to the specified height and stops.
function moveToAltitude(height){
      altitude = getAltitude() - height;
      startTime = getTime();
      if (altitude > 0) {
            dropDown();
      }
      else{
            altitude = - altitude;
            climbUp();
      }
      while (getTime() - startTime < altitude * 4) {}
      stopHorizontal();
}
```

```
function turnNorth(){
      turnRightDegrees(getHeading());
}

function turnWest(){
      turnNorth();
      turnRightDegrees(270);
}

function turnEast(){
      turnNorth();
      turnRightDegrees(90);
}

function turnSouth(){
      turnNorth();
      turnRightDegrees(180);
}
```

TEST PROGRAM 1:

```
function droneController(heading, distance){
      currentDir = getHeading();
      turnRightDegrees(heading - current_dir);
      moveForwardDistance(distance);
      stopHorizontal();
}
```

TEST PROGRAM 2:

```
function sprayRectangle(curX, curY, rectX, rectY, rectW, rectH){
      xMove = rectX - curX;
      yMove = rectY - curY;
      moveToAltitude(4);

      turnEast();
      moveForwardDistance(xMove);
      turnNorth();
      moveForwardDistance(yMove);

      sprayOn();
      moveForwardDistance(rectH);

      turnEast();
      moveForwardDistance(rectW);

      turnSouth();
      moveForwardDistance(rectH);

      turnWest();
      moveForwardDistance(rectW);

      sprayOff();
}
```

TEST PROGRAM 3:

```
// this function creates movements and prints the results
function main(){
      // testing output functionality, printing the drone's heading and altitude.
      print("Heading is : ");
      print(getHeading());
      print("Altitude is: ");
      print(getAltitude());

      // testing the standard library and primitive functions
      moveForwardDistance(25);
      moveToAltitude(20);
      turnEast();
      moveForwardDistance(12);

      // testing a time based operation
      startTime = getTime();
      while (getTime() - startTime < 3) {
            sprayOn();
      }
      sprayOff();

      // testing if current location is indeed correct
      print("Heading is: ");
      print(getHeading());
      print("Altitude is: ");
      print(getAltitude());
}

// Execute the main function
main();
```

TEST PROGRAM 4:

```
// this function makes the drone traverse a hilbert curve
// https://medium.com/@nico727272/drawing-hilbert-curves-using-turtle-46f6d628732b
// call with angle = 90 for standard Hilbert Curves
function traverseHilbertCurve(order, angle){
      if(order != 0){
            turnRightDegrees(360 - angle);
            traverseHilbertCurve(order - 1, - angle);
            moveForwardDistance(1);
            turnRightDegrees(angle - 360);
            traverseHilbertCurve(order-1, angle);
            moveForwardDistance(1);
            traverseHilbertCurve(order-1, angle);
            turnRightDegrees(angle - 360);
            moveForwardDistance(1);
            traverseHilbertCurve(order - 1, - angle);
            turnRightDegrees(360 - angle);
      }
}
```

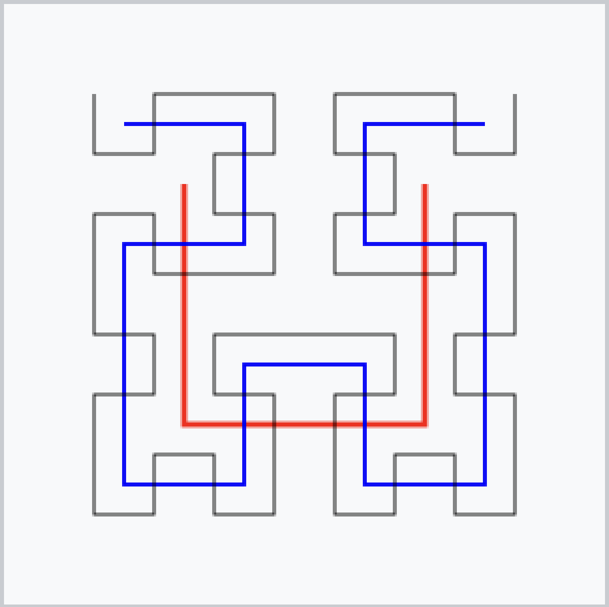

Fig 1. Hilbert Curves of order 1, 2 and 3, image taken from wikipedia.

Lex File:

```
%%
"//".*                          { printf("COMMENT"); }
";"                             { printf(" SEMICOLON ");}
"("                             { printf(" LP ");}
")"                             { printf(" RP ");}
"{"                             { printf(" LBRACE ");}
"}"                             { printf(" RBRACE ");}
“,”                             { printf(" COMMA ");}
“<”                             { printf(" LT ");}
“>”                             { printf(" GT ");}
“<=”                            { printf(" LT_EQ ");}
“>=”                            { printf(" GT_EQ ");}
“==”                            { printf(" EQ ");}
"!="                            { printf(" NOT_EQ "); }
“=”                             { printf(" ASSIGN ");}
“+”                             { printf(" PLUS ");}
“-”                             { printf(" MINUS ");}
“*”                             { printf(" MULT ");}
“/”                             { printf(" DIV ");}
“function”                      { printf(" FUNCTION ");}
“while”                         { printf(" WHILE ");}
“if”                            { printf(" IF ");}
“else”                          { printf(" ELSE ");}
“and”                           { printf(" LOGICAL_AND ");}
“or”                            { printf(" LOGICAL_OR ");}
“not”                           { printf(" LOGICAL_NOT ");}
“True”                          { printf(" TRUE_LIT ");}
“False”                         { printf(" FALSE_LIT ");}
[-+]?[0-9]*\.?[0-9]+            { printf(" NUMBER_LIT ");}
[a-zA-Z_]+                      { printf(" IDENTIFIER ");}
\"([^\\\"]|\\.)*\"              { printf(" STRING_LIT "); }


%%
int yywrap() {
   return 1;
}
int main(void) {
   yylex();
   return 0;
}
```