Word count: 2082 words

1. Overview:

This project involves developing a compiler that translates TurtleScript programs into PostScript, reinforcing key programming concepts such as lexing, parsing, and code generation. Working in groups, we were required to modify and extend the provided starter code to implement a lexer and parser that validate TurtleScript programs, ensuring correct syntax recognition. Additionally, we implemented code generation, converting valid TurtleScript commands into their PostScript equivalents. The project also emphasized collaboration through version control and teamwork. All primary requirements were met, including lexical analysis, syntax validation, and basic PostScript translation. Furthermore, Secondary and Tertiary requirements were also met successfully.

2. Team Working:

Throughout the project, our team implemented the technique of pair programming to improve productivity and communication. Usually, one person was the driver, actively developing the code, while the other member was the navigator, continuously examining the code, making suggestions for enhancements, and spotting any problems. To guarantee a fair contribution and a better understanding of every facet of the implementation, we often exchanged positions. Regular communication through messages allowed us to stay in touch while discussing our progress, allocating work, and resolving any obstacles. Additionally, we used Git version control to easily track changes and combine contributions. Most of the time during the project both of us worked on the program together.

3. Code Design:

3.1 Lexer:

3.1.i Type.java:

To start off our project, we started off by completing the list of tokens provided to us in the Type.java file.

3.1.ii Token.java:

We followed the same format provided in the constructor and completed all the tokens.

3.1.iii Lexer.java:

In Lexer.java, we implemented the next() method, which retrieves the next token from the input stream. The method processes characters sequentially and determines the type of token

to return. If the current character is a digit (0-9), we use a StringBuilder to build a numeric token. A while loop continues collecting digits until a non-digit character is encountered. Once the number is fully formed, a Token representing a numeric value is returned. If the character is a letter (a-z or A-Z), the lexer behaves similarly, using a loop to collect an identifier (a variable name or keyword). The extracted string is then checked against predefined keywords (e.g., "learn" or "forward"). If it matches a keyword, the corresponding keyword token (like PROC_TOKEN or FORWARD_TOKEN) is returned. Otherwise, it is treated as an identifier token. A switch statement handles single-character operators like +, -, *, /, and comparison operators such as ==, !=, <, >, etc. Tokens are returned based on the matched symbol. If the character is a dollar sign another StringBuilder is used to build a parameter token, similar to how numeric and identifier tokens are constructed. If -1 is encountered (which commonly represents the end of input stream in many languages), the lexer recognizes this as the end of the file and returns the EOI (End of Input) token.

To complete the tertiary requirements, we created an integer field called lineCounter which keeps track of the line the lexer is currently at. If there is an error in that line, parser will call getLineCounter() and print out the error with the line it is in.

3.2 Parser:

3.2.i Parser.java:

In Parser.java, the first change was adding an error message in the ParseProg() method. An if statement is used to check if the input token is null, and if so, an error message is displayed. This ensures that parsing does not proceed with invalid input. Since the PROC token (representing "learn") is not needed for parsing, an if statement was added to check whether the current token's type is PROC. If true, the next() method is called to skip it and move to the next token. When parsing function parameters, commas are used as separators. If the token type is a comma (COMMA_TOKEN), the parser skips it by calling next() to move forward. This ensures that parameters are parsed correctly without unnecessary processing. All the error messages are implemented with the help of if statements and if the token type is exactly what was expected. For getting the procedure body was utilized and it looped until it detected a right brace. Further, looking at the parseStmt() method, the first 3 cases are the same just for 3 different directions. This case handles tokens of type FORWARD. The current token (which would be FORWARD) is stored in forwardToken. The parser then advances to the next token in the input stream. Next, the program calls parseExpr() to evaluate the expression that follows FORWARD, representing the number of steps the turtle should move. It then creates a ForwardStmt object using the FORWARD token and the parsed expression. Finally, the switch statement exits as the FORWARD case has been successfully handled. The same happens with RIGHT and LEFT tokens.

For the REPEAT token, the parser first checks if there is a number (which determines how many times the loop should run). Then, it verifies the presence of a left brace, which marks

the beginning of the loop body. An ArrayList of statements (stmts) is created to store the statements inside the loop. The while loop continuously adds statements to stmts until it detects a right brace. Once the right brace is found, a RepeatStmt object is created using the repeat token, the parsed number, and the list of statements. For the IF statement, the parser first checks if the condition expression is valid. If it is null, a syntax error is reported. Then, it checks for a left brace, marking the beginning of the if block. A token is stored for ifLBrace, and a list of statements inside the block is created. The parser ensures that a right brace is present to close the block. Next, the parser checks if the next token is ELSE. If an ELSE block is present, the same parsing process occurs it verifies a left brace, collects statements inside the block, and checks for a closing right brace. Finally, an IfElseStmt object is created with all the required parameters. If there is no ELSE block, an IfElseStmt object is still created, but the else parameters are set to null. The last case handles the IDENT token, which represents procedure calls. The parser checks if there is an expression following the identifier. While the next token is either a parameter or a number, the expression gets added to the ArrayList<Expr> exprs. If a comma is present between expressions, it is skipped to ensure proper parsing. Once all arguments have been collected, a CallStmt object is created using the identifier token and the list of parsed expressions.

The last method edited was parsePrimaryExpr(), which determines the type of expression. If the token is a number, a new NumExpr object is created, with the current token stored as its value. If the token is a parameter, a new ParamExpr object is created in a similar manner. If the token is neither a number nor a parameter, an error message is displayed, indicating a syntax error.

3.3 Grammar:

3.3.i Proc.java:

The instance variable args declaration was changed into a list, we decided this because there are cases where there can be more than one parameter. These args were then appended to the String builders using for loops. Finally, two minor changes were made to toPostScript(): if there is more than one arg, the String “2 dict begin\n” should be “3 dict begin\n”, and each arg should be headed with “exch def\n” and outputted in reverse order.

3.3.ii Expr.java remains unmodified. **NumExpr.java** and **ParamExpr.java** inherit from it.

These two classes were created to represent the two types of primary expressions. They contain the same attributes and methods; the only difference is the value of the Token, which will be the appropriate type for each expression.

3.3.iii Stmt.java also remains unmodified. **ForwardStmt.java**, **RightStmt.java** and **LeftStmt.java** are inherited from it.

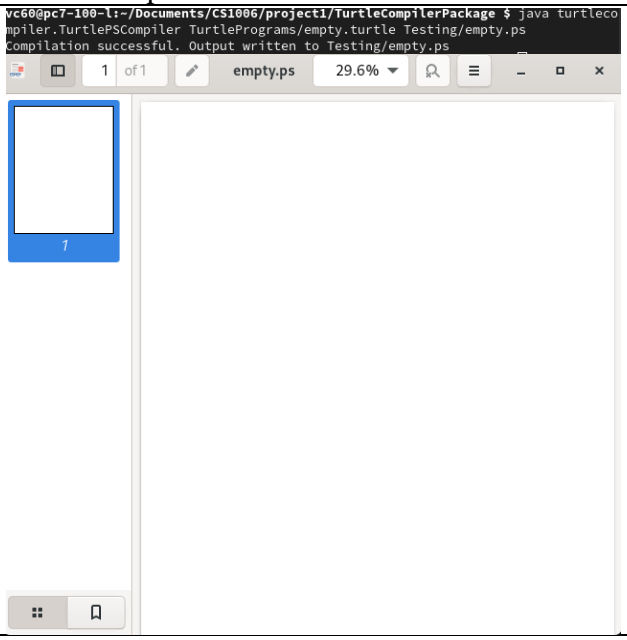
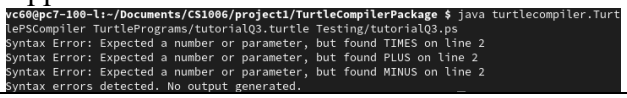
These three classes were created to represent each statement separately. They contain the same attributes and methods; the only difference is the value of Token name which will be the appropriate one for each statement. To generate correct code, spaces and new lines were added, and toPostScript() outputs the expression first and then the token name.

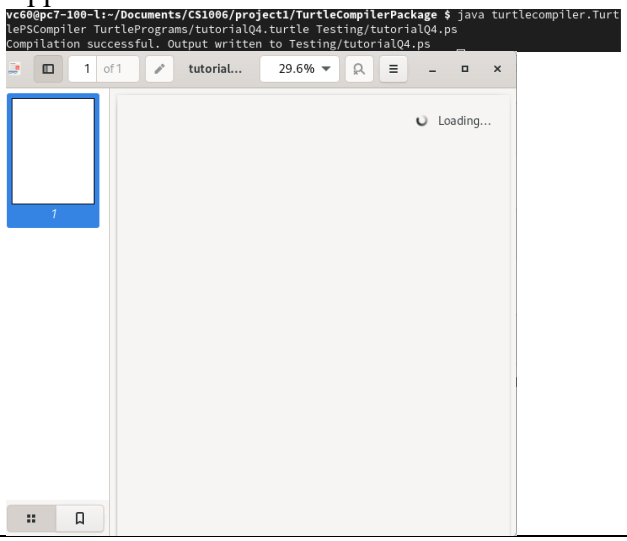
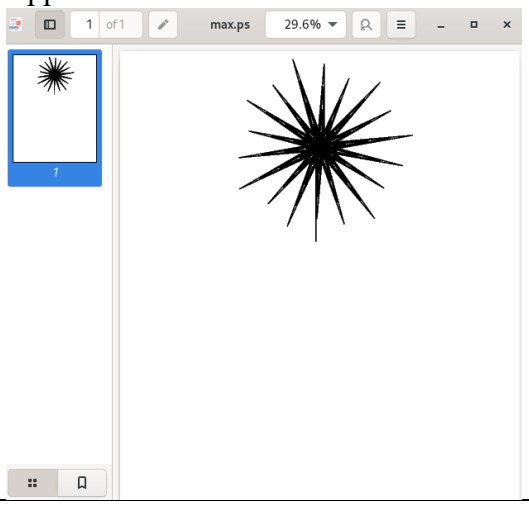
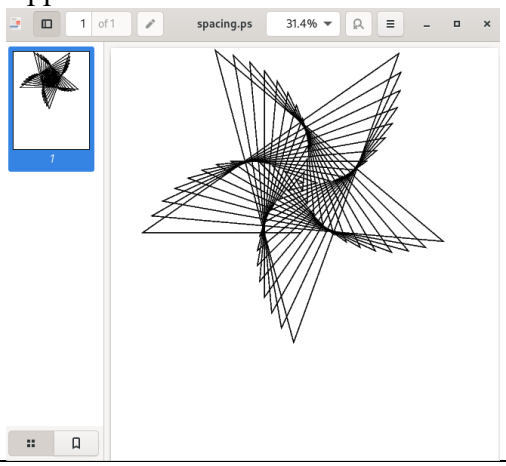
3.3.iv RepeatStmt.java and **ifElseStmt.java**

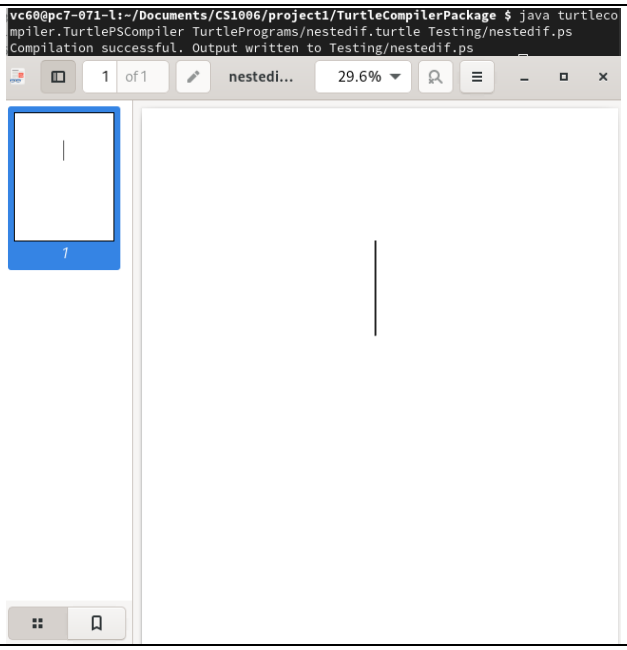
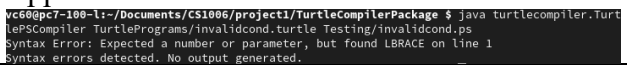
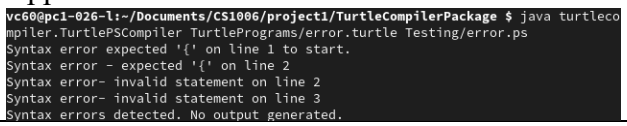
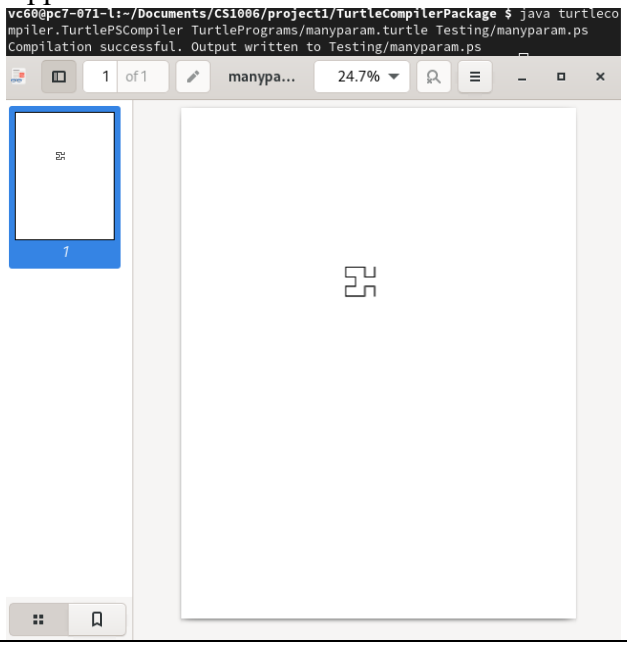
These classes are inherited from Stmt.java and it were created to represent the repeat and if-else statements independently. Based on the definition "repeat" num '{' { stmt } '}', the class contains 5 attributes, one for each element in the definition. The two methods were modified to correctly generate each script. As for ifElseStmt.java, this class contains 9 attributes, following the structure from the definition: "if" expr '{' { stmt } '}' "else" '{' { stmt } '}'. In the constructor, an if statement was added to make sure the attributes for the else part of the statement are initialised with null values in the case that there is an if without an else. Similar to RepeatStmt.java, the two methods were modified to correctly generate the two types of script.

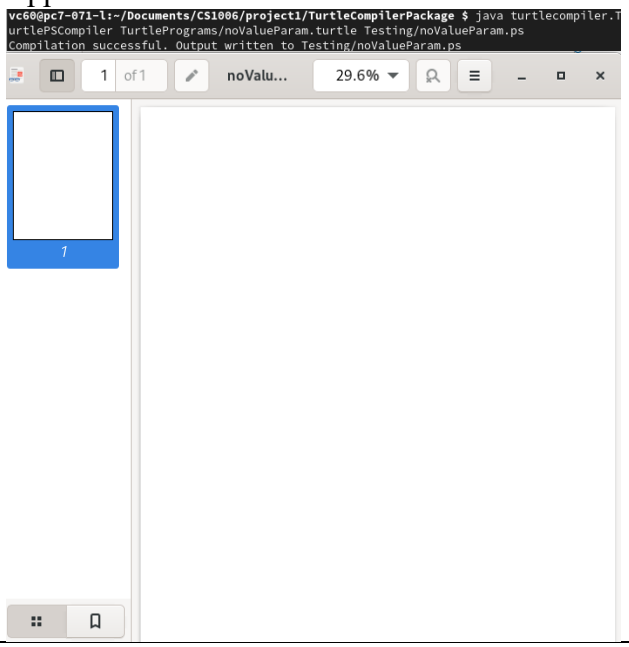
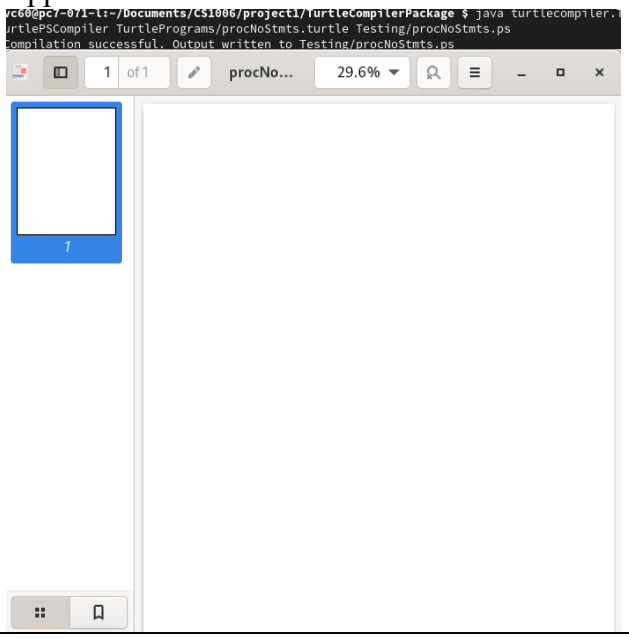
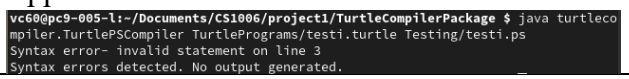
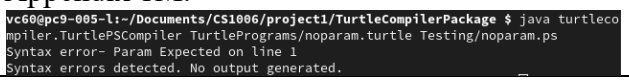
3.3.v BinaryExpr.java, **CallStmt.java**, **Prog.java** were not modified.

4. Testing:

Case	Output	Pass/Fail
Compiling all the turtle files provided in the Package.	All the outputs were generated and compared to the samples provided, the ones which did not have a sample were tested on Turtle	Pass
Empty file		Pass
Multiple operators in 1 line	The code for this has been provided in Appendix 1A. 	Pass

Syntactically correct code but not semantically	The code for this has been provided in Appendix 1B. 	Pass
Code in 1 line	The code for this has been provided in Appendix 1C. 	Pass
Multiple spacing between the code	The code for this has been provided in Appendix 1D. 	Pass
Multiple nested if statements and nested else statements.	The code for this has been provided in Appendix 1E.	Pass

		
Invalid if conditions	The code for this has been provided in Appendix 1F. 	Pass
Multiple errors in 1 code file.	The code for this has been provided in Appendix 1G. 	Pass
Multiple parameters in a procedure	The code for this has been provided in Appendix 1H. 	Pass
Invalid Parameter Scenario missing parameter in proc call	The code for this has been provided in Appendix 1I. 	Pass

Param with no value	The code for this has been provided in Appendix 1J. 	Pass
Procedure with no statements	The code for this has been provided in Appendix 1K. 	Pass
Starting a prog with a statement instead of a proc	The code for this has been provided in Appendix 1L. 	Pass
No parameter	The code for this has been provided in Appendix 1M. 	Pass
Param without dollar sign	The code for this has been provided in Appendix 1N.	Pass

	vc60@pc7-071-l:~/Documents/CS1006/project1/TurtleCompil urtlePSCompiler TurtlePrograms/paramWithNoDollar.turtle Syntax error- Param Expected on line 1 Syntax error expected '{' on line 1 to start. Syntax error- invalid statement on line 1 Syntax error- invalid statement on line 6 Syntax errors detected. No output generated. vc60@pc7-071-l:~/Documents/CS1006/project1/TurtleCompil	
Ident starting with a number	The code for this has been provided in Appendix 1O. vc60@pc7-071-l:~/Documents/CS1006/project1/TurtleComp urtlePSCompiler TurtlePrograms/identWithDigit.turtle Syntax error- expected procedure name on line 1 Syntax error- Param Expected on line 1 Syntax error expected '{' on line 1 to start. Syntax error- invalid statement on line 1 Syntax error- invalid statement on line 1 Syntax errors detected. No output generated. vc60@pc7-071-l:~/Documents/CS1006/project1/TurtleCompr	Pass
Proc without learn	The code for this has been provided in Appendix 1P. vc60@pc7-071-l:~/Documents/CS1006/project1/Tur urtlePSCompiler TurtlePrograms/procWithoutLear Syntax error- invalid statement on line 1 Syntax errors detected. No output generated. vc60@pc7-071-l:~/Documents/CS1006/project1/Tur	Pass
Stmt without expr	The code for this has been provided in Appendix 1Q. vc60@pc7-071-l:~/Documents/CS1006/project1/TurtleCompilerPackage $ java turtle urtlePSCompiler TurtlePrograms/stmtWithNoExpr.turtle Testing/stmtWithNoExpr.ps Syntax Error: Expected a number or parameter, but found EOI on line 1 Syntax errors detected. No output generated. vc60@pc7-071-l:~/Documents/CS1006/project1/TurtleCompilerPackage $	Pass

5. Evaluation:

All the requirements—primary, secondary, and tertiary—have been successfully implemented in our compiler. Throughout the development process, we gained a strong understanding of lexing, parsing, and code generation, which greatly enhanced our programming skills. Our compiler correctly translates TurtleScript programs into PostScript, handling all expected syntax and producing accurate output. Implementing error handling and improving error messages further strengthened our solution.

Pair programming proved to be an effective approach, ensuring both team members to have a thorough understanding of the concepts and contributing equally. While our implementation meets all specifications, future enhancements could include further optimizations for efficiency and improved debugging support. Overall, we are satisfied with our submission, as it not only fulfills all the assignment requirements but also solidifies our understanding of compiler construction.

6. Conclusion:

We thoroughly enjoyed working on this project, particularly the challenge of implementing a fully functional TurtleScript-to-PostScript compiler. The process of designing the lexer, parser, and code generator was both engaging and rewarding. Pair programming was an

invaluable technique, allowing us to collaborate effectively, exchange ideas, and refine our understanding of compiler development. One of the most satisfying aspects was successfully implementing all the required features and seeing our compiler generate correct PostScript output. Celebrating after each small achievement made us truly understand what we had accomplished. Debugging and refining error handling presented some challenges, but through iterative testing and continuous improvements, we ensured the compiler functions as expected. This project deepened our knowledge of programming language translation and strengthened our problem-solving abilities. Given more time, we would explore further optimizations and additional language extensions such as making our compiler handle semantic errors. Overall, this experience has been instrumental in enhancing our programming skills, and we look forward to applying these concepts to future software development challenges.

References:

[1] https://studres.cs.st-andrews.ac.uk/CS1006/

[2] https://docs.oracle.com/javase/8/docs/api/

[3] https://apps.kde.org/en-gb/kturtle/

[4] https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

[5] https://www.geeksforgeeks.org/stringbuilder-class-in-java-with-examples/

[6] https://dev.to/codingwithadam/introduction-to-lexers-parsers-and-interpreters-with-chevrotain-5c7b

[7] https://stackoverflow.com/

7. Appendix:

Appendix 1A

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Tutorials/MiniCompiler-T2.pdf

```
learn ldragon $level {
    if $level == 0 {
        forward *+- 5
    }
    else {
        ldragon $level - 1
        left 90
        rdragon $level - 1
    }
}
ldragon 3
```

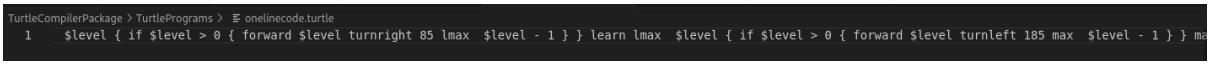
Appendix 1B

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Tutorials/MiniCompiler-T2.pdf

```
learn ldragon $level {
    if $level == 0 {
        forward $distance
    }
    else {
        ldragon $level-1
        left 90
        rdragon $level-1
    }
}
ldragon 3
```

Appendix 1C

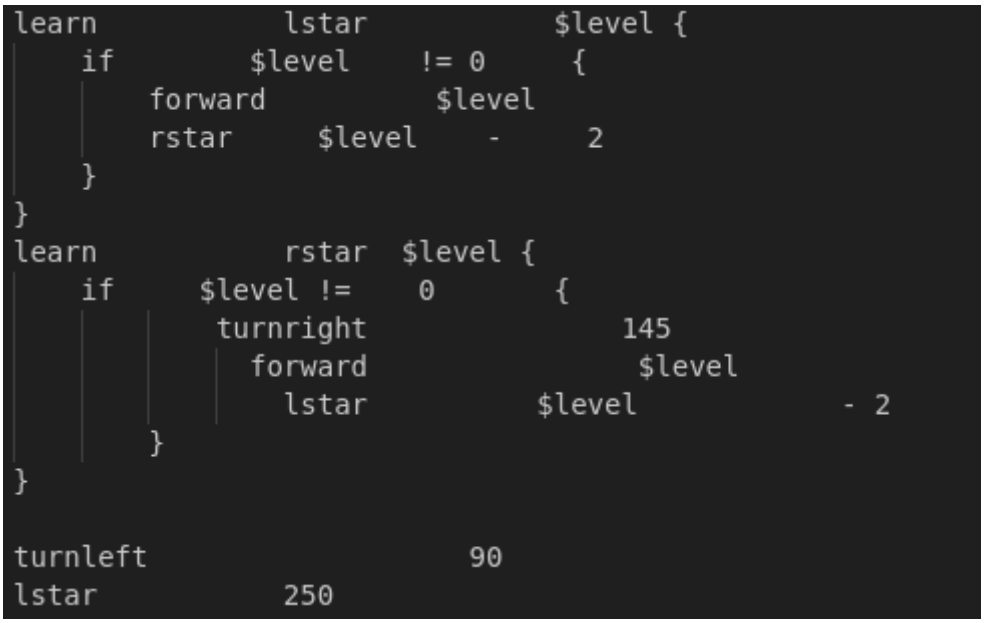
Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/



The code in this screenshot it taken from max.turtle and put it in 1 line

Appendix 1D

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/



```
learn           lstar           $level {
    if        $level    != 0     {
        forward          $level
        rstar     $level    -     2
    }
}
learn           rstar  $level {
    if     $level !=    0       {
            turnright               145
              forward                $level
                lstar          $level            - 2
        }
}

turnleft                  90
lstar           250
```

The code in this screenshot it taken from starfractal.turtle with a lot of spacing.

Appendix 1E

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn ldragon $level {
    if $level >= 0 {
        forward 122
        if $level == 0 {
            forward 333
        }
    }
    else {
        turnleft 190
    }
}

learn rdragon $level {
    if $level >= 0 {
        forward 222
        if $level == 0 {
            forward 333
        }
        else {
            forward 111
        }
    } else {
            turnright 90
        }
}
ldragon 11
```

Appendix 1F

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn koch $depth, $length {
  if $depth >  {
    koch $depth-1, $length/3
    turnleft 60
    koch $depth-1, $length/3
    turnright 120
    koch $depth-1, $length/3
    turnleft 60
    koch $depth-1, $length/3
  } else {
      forward 3*$length
  }
}

koch 6, 60
```

The code in this screenshot it taken from koch.turtle with a lot of spacing.

Appendix 1G

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn ldragon $level
    if $level == 0
        forward
    }
    else
        ldragon level - 1
        turnleft 90
        rdragon $level - 1

}
```

Appendix 1H

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn lhilbert $level, $length, $penSize, $draw {

    if $level != 0 {
        turnleft 90
        rhilbert $level - 1, $length, $penSize, $draw
        forward $length
        turnright 90
        lhilbert $level - 1, $length, $penSize, $draw
        forward $length
        lhilbert $level - 1, $length, $penSize, $draw
        turnright 90
        forward $length
        rhilbert $level - 1, $length, $penSize, $draw
        turnleft 90
    }
}

learn rhilbert $level, $length, $penSize, $draw {
    if $level != 0 {
        turnright 90
        lhilbert $level - 1, $length, $penSize, $draw
        forward $length
        turnleft 90
        rhilbert $level - 1, $length, $penSize, $draw
        forward $length
        rhilbert $level - 1, $length, $penSize, $draw
        turnleft 90
        forward $length
        lhilbert $level - 1, $length, $penSize, $draw
        turnright 90
    }
}

lhilbert 2, 15, 20, 4
```

Appendix 1I

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn square {
    forward $size
    turnright 90
    forward $size
    turnright 90
    forward $size
    turnright 90
    forward $size
    turnright 90
}

square 100
```

Appendix 1J

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn printValue $ {
  if $ > 0 {
    forward $
  }
}

printValue 0
```

Appendix 1K

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn printValue $value {
  if $value > 0 {

  }
}
printValue 4
```

Appendix 1L

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
forward 100

learn lstar $level {
    if $level != 0 {
      forward $level
      rstar  $level - 2
    }
}
```

Appendix 1M

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
learn printValue {
  if $value > 0 {
    forward $value
  }
}

printValue 0
```

Appendix 1N

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
1  learn lstar level {
2      if $level != 0 {
3          forward $level
4          rstar  $level - 2
5      }
6  }
```

Appendix 1O

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
1  learn 1star $level {
2      if $level != 0 {
3          forward $level
4          rstar  $level - 2
5      }
6  }
```

Appendix 1P

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
lstar $level {
    if $level != 0 {
        forward $level
        rstar  $level - 2
    }
}
```

Appendix 1Q

Source of this turtlescript: https://studres.cs.st-andrews.ac.uk/CS1006/Coursework/Project1/

```
forward
```