

Language Name: Sparky

General System Requirements:

Processor: Intel and AMD processors (Processors with instruction set capable of imperative paradigm)

Operating System on which compiler and runtime are built: Windows OS.

Type of Language: Imperative

Data Structure Used: Abstract Syntax Tree, Hash Map, Stack

Tools Used: Git, Eclipse, Antlr (<https://www.antlr.org/>), ANT

Parsing Technique Employed: Antlr (Feeding a grammar (.g4) file to generate an abstract syntax tree)

Steps to install and run Antlr on Windows:

- Download <https://www.antlr.org/download/antlr-4.8-complete.jar>.
- Add antlr4-complete.jar to CLASSPATH, either:
- Permanently: Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable
- Temporarily, at command line:
- SET CLASSPATH=.;C:\Javalib\antlr4-complete.jar;%CLASSPATH%
- Create batch commands for ANTLR Tool, TestRig in dir in PATH
- antlr4.bat: java org.antlr.v4.Tool %*
- grun.bat: java org.antlr.v4.gui.TestRig %*

Alternative Steps using Dos Key commands to run Antlr:

- Doskey antlr4=java org.antlr.v4.Tool \$*
- Doskey grun =java org.antlr.v4.gui.TestRig \$*

Directions/instructions to install Sparky language

Follow the below steps to install via GitHub:

- Clone the git project from <https://github.com/MayankBatra005/SER502-Spring2020-Team25>
- Download this git project and Unzip the project in a new folder.
< Make sure there should be no spaces or invalid characters>
- Open the project in Eclipse using following steps:
- Files >> Open Project From File System. Browse your project folder here upto extracted project directory.

Steps to build JARS for sparky

1. Right click on the project folder at the top.
2. Click on Export -> Under the Java Option, select Runnable JAR File option. -> Click Next.

3. Select the destination directory in which you want to export the jar.
4. Under Library handling chose "package required libraries into generated Jar"
6. Click on Finish

Note : Jar will be generated under selected destination folder mentioned in step 3

→Please refer to Installation steps as shown in YouTube video

How to run any program using Sparky language

Run via Eclipse:

1. Select the Compiler.java class under src>sparkyCompiler>Compiler.java
2. Right click and select run as Run Configurations
3. Select Arguments tab
4. Provide the complete path of the file located on your disk with extension as sparky

**** Make sure the file should be stored on path containing no white spaces or invalid characters such as _ / - etc. ****

5. Click on Run
6. Output can be seen in Eclipse console

Run using compiler Jars:

Pre requisite:

1. Jar should be generated as illustrated in above steps
2. Source code with extension as ".sparky" is created and path to this file is known

Steps to run on console(Windows command prompt):

1. Navigate to the folder where compiler.jar was created
2. open command prompt (CMD) on this location
3. Type the following command Java - jar compiler.jar "path\Filename.sparky"

Path stands for the path to the file

Filename stands for the name of the file which contains the source code

4. Hit Enter
5. Code is executed on command prompt

Grammar Snippet:

```
1. grammar Sparky;
2. program: LIVE ball DIE;
3. ball: expression* | declare* expression*;
4.
5. declare:
6. (datatype STUFF EQUALTO NUMBER SEMICOLON)|
7. (datatype STUFF SEMICOLON)|
8. (HAINA STUFF EQUALTO booleanvalue SEMICOLON)|
9. (HAINA STUFF SEMICOLON)| stringdatatype STUFF EQUALTO STRINGLITERAL SEMICOLON | stringd
   atatype STUFF SEMICOLON;
10.
11. expression
12. : assignment
13. | ifte
14. | loopum
15. |ternary_operator
16. |print;
17.
18. assignment
19. : STUFF EQUALTO expr SEMICOLON |
20. STUFF EQUALTO yesnostatement SEMICOLON
21. ;
22.
23. ifte
24. : IF yesnostatement in_loop ('warna' in_loop)? FI
25. ;
26.
27.
28. loopum : loop_for|loop_while | loop_for_range;
29. loop_for: 'for' LSmoothBrace for_declare? ';' for_expression? ';' for_expr? RSmoothBrac
   e in_loop;
30. loop_while
31. : WHILE yesnostatement in_loop
32. ;
33.
34. loop_for_range: 'for' STUFF 'in' 'range' LSmoothBrace NUMBER COMMA NUMBER RSmoothBrac
   e in_loop;
35.
36. in_loop: LCurlyBrace ball RCurlyBrace| expression;
37.
38. for_expr: STUFF EQUALTO expr;
39. for_expression :expr YESNOOPERATOR expr;
40. for_declare:datatype STUFF EQUALTO NUMBER;
41.
42. term: NUMBER | STUFF | STUFF op=(MUL | DIV) term | NUMBER op=(MUL | DIV) term;
43. expr: term | term op=(PLUS | MINUS) expr | NOT expr;
44. yesnostatement : booleanvalue | expr YESNOOPERATOR expr |yesnostatement ANDOROPERATOR y
   esnostatement;
45. ANDOROPERATOR: AND|OR;
46.
47. AND: 'and';
48. OR: 'or';
49. NOT:'not';
50.
51. ternary_operator: yesnostatement '?' in_loop ':' in_loop;
52.
53. print:'print' LSmoothBrace expr RSmoothBrace SEMICOLON;
```

```

54. LIVE: 'Live';
55. DIE: 'Die';
56. FI: 'fi';
57.
58.
59. YESNOOPERATOR: ASSEQ| LESS_THAN| MORE_THAN | LESS_THAN_EQ | MORE_THAN_EQ ;
60. EQUALTO : '=';
61. ASSEQ : '==';
62. LESS_THAN : '<';
63. MORE_THAN: '>';
64. LESS_THAN_EQ : '<=';
65. MORE_THAN_EQ : '>=';
66.
67. warna : 'else';
68.
69. PLUS : '+';
70. MINUS : '-';
71. MUL : '*';
72. DIV : '/';
73. SEMICOLON : ';';
74. COMMA : ',';
75.
76. LSmoothBrace : '(';
77. RSmoothBrace : ')';
78. LCurlyBrace : '{';
79. RCurlyBrace : '}';
80. DQ: '"';
81.
82. STRINGLITERAL: DQ (~["\\r\\n])* DQ;
83. HAINA: 'haina';
84. haina: 'bool';
85. datatype: INTEGER| DOUBLE | HAINA;
86. stringdatatype: STRING;
87. INTEGER: 'int';
88. STRING: 'string';
89. DOUBLE: 'double';
90. IF : 'if';
91. WHILE : 'while';
92. STUFF:[a-zA-Z_] [a-zA-Z_0-9]*;
93. NUMBER:[0-9]+;
94. WS: [ \t\r\n] -> skip;
95. booleanvalue: 'yup' | 'nup';
96. yup: 'true';
97. nup: 'false';

```

Model class to store intermediate Code

```

1. package Model;
2.
3. /**
4.  * This purpose of this class is to be a model for writing the intermediate code keywords
5.  * @author Mayank Batra
6.  * @since April-27-2020
7.  * @version 1.0
8.  */
9.
10. public class IntermediateCodeWriter {

```

```

11.
12.     private String icOutput = "";
13.     public static IntermediateCodeWriter singletonInstance;
14.
15.     private IntermediateCodeWriter() {}
16.
17.     public void addOutput(String output) {
18.         this.icOutput += output + "\n";
19.     }
20.
21.
22.
23.     public String getIcOutput() {
24.         return icOutput;
25.     }
26.
27.     public void setIcOutput(String icOutput) {
28.         this.icOutput = icOutput;
29.     }
30.
31.     public static IntermediateCodeWriter getInstance()
32.     {
33.         if (singletonInstance==null)
34.         {
35.             singletonInstance=new IntermediateCodeWriter ();
36.         }
37.         return singletonInstance;
38.     }
39. }

```

Files Auto generated by ANTLR (Lexer, Parser , Token etc)

```

1. // Generated from Sparky.g4 by ANTLR 4.8
2. package sparky;
3.
4. import org.antlr.v4.runtime.ParserRuleContext;
5. import org.antlr.v4.runtime.tree.ErrorNode;
6. import org.antlr.v4.runtime.tree.TerminalNode;
7.
8. /**
9.  * This class provides an empty implementation of {@link SparkyListener},
10.  * which can be extended to create a listener which only needs to handle a subset
11.  * of the available methods.
12.  */
13. public class SparkyBaseListener implements SparkyListener {
14.     /**
15.      * {@inheritDoc}
16.      *
17.      * <p>The default implementation does nothing.</p>
18.      */
19.     @Override public void enterProgram(SparkyParser.ProgramContext ctx) { }
20.     /**
21.      * {@inheritDoc}
22.      *
23.      * <p>The default implementation does nothing.</p>
24.      */
25.     @Override public void exitProgram(SparkyParser.ProgramContext ctx) { }
26.     /**
27.      * {@inheritDoc}
28.      *

```

```

29.     * <p>The default implementation does nothing.</p>
30.     */
31.     @Override public void enterBall(SparkyParser.BallContext ctx) { }
32.     /**
33.     * {@inheritDoc}
34.     *
35.     * <p>The default implementation does nothing.</p>
36.     */
37.     @Override public void exitBall(SparkyParser.BallContext ctx) { }
38.     /**
39.     * {@inheritDoc}
40.     *
41.     * <p>The default implementation does nothing.</p>
42.     */
43.     @Override public void enterDeclare(SparkyParser.DeclareContext ctx) { }
44.     /**
45.     * {@inheritDoc}
46.     *
47.     * <p>The default implementation does nothing.</p>
48.     */
49.     @Override public void exitDeclare(SparkyParser.DeclareContext ctx) { }
50.     /**
51.     * {@inheritDoc}
52.     *
53.     * <p>The default implementation does nothing.</p>
54.     */
55.     @Override public void enterExpression(SparkyParser.ExpressionContext ctx) { }
56.     /**
57.     * {@inheritDoc}
58.     *
59.     * <p>The default implementation does nothing.</p>
60.     */
61.     @Override public void exitExpression(SparkyParser.ExpressionContext ctx) { }
62.     /**
63.     * {@inheritDoc}
64.     *
65.     * <p>The default implementation does nothing.</p>
66.     */
67.     @Override public void enterAssignment(SparkyParser.AssignmentContext ctx) { }
68.     /**
69.     * {@inheritDoc}
70.     *
71.     * <p>The default implementation does nothing.</p>
72.     */
73.     @Override public void exitAssignment(SparkyParser.AssignmentContext ctx) { }
74.     /**
75.     * {@inheritDoc}
76.     *
77.     * <p>The default implementation does nothing.</p>
78.     */
79.     @Override public void enterIfte(SparkyParser.IfteContext ctx) { }
80.     /**
81.     * {@inheritDoc}
82.     *
83.     * <p>The default implementation does nothing.</p>
84.     */
85.     @Override public void exitIfte(SparkyParser.IfteContext ctx) { }
86.     /**
87.     * {@inheritDoc}
88.     *
89.     * <p>The default implementation does nothing.</p>

```

```

90.     */
91.     @Override public void enterLoopum(SparkyParser.LoopumContext ctx) { }
92.     /**
93.      * {@inheritDoc}
94.      *
95.      * <p>The default implementation does nothing.</p>
96.      */
97.     @Override public void exitLoopum(SparkyParser.LoopumContext ctx) { }
98.     /**
99.      * {@inheritDoc}
100.      *
101.      * <p>The default implementation does nothing.</p>
102.      */
103.     @Override public void enterLoop_for(SparkyParser.Loop_forContext ctx) { }
104.     /**
105.      * {@inheritDoc}
106.      *
107.      * <p>The default implementation does nothing.</p>
108.      */
109.     @Override public void exitLoop_for(SparkyParser.Loop_forContext ctx) { }
110.     /**
111.      * {@inheritDoc}
112.      *
113.      * <p>The default implementation does nothing.</p>
114.      */
115.     @Override public void enterLoop_while(SparkyParser.Loop_whileContext ctx) {
116.     }
117.     /**
118.      * {@inheritDoc}
119.      *
120.      * <p>The default implementation does nothing.</p>
121.      */
122.     @Override public void exitLoop_while(SparkyParser.Loop_whileContext ctx) { }
123.     /**
124.      * {@inheritDoc}
125.      *
126.      * <p>The default implementation does nothing.</p>
127.      */
128.     @Override public void enterLoop_for_range(SparkyParser.Loop_for_rangeContext
129.     ctx) { }
130.     /**
131.      * {@inheritDoc}
132.      *
133.      * <p>The default implementation does nothing.</p>
134.      */
135.     @Override public void exitLoop_for_range(SparkyParser.Loop_for_rangeContext
136.     ctx) { }
137.     /**
138.      * {@inheritDoc}
139.      *
140.      * <p>The default implementation does nothing.</p>
141.      */
142.     @Override public void enterIn_loop(SparkyParser.In_loopContext ctx) { }
143.     /**
144.      * {@inheritDoc}
145.      *
146.      * <p>The default implementation does nothing.</p>
147.      */
148.     @Override public void exitIn_loop(SparkyParser.In_loopContext ctx) { }

```

```

147.         * {@inheritDoc}
148.         *
149.         * <p>The default implementation does nothing.</p>
150.         */
151.     @Override public void enterFor_expr(SparkyParser.For_exprContext ctx) { }
152.     /**
153.         * {@inheritDoc}
154.         *
155.         * <p>The default implementation does nothing.</p>
156.         */
157.     @Override public void exitFor_expr(SparkyParser.For_exprContext ctx) { }
158.     /**
159.         * {@inheritDoc}
160.         *
161.         * <p>The default implementation does nothing.</p>
162.         */
163.     @Override public void enterFor_expression(SparkyParser.For_expressionContext
    ctx) { }
164.     /**
165.         * {@inheritDoc}
166.         *
167.         * <p>The default implementation does nothing.</p>
168.         */
169.     @Override public void exitFor_expression(SparkyParser.For_expressionContext
    ctx) { }
170.     /**
171.         * {@inheritDoc}
172.         *
173.         * <p>The default implementation does nothing.</p>
174.         */
175.     @Override public void enterFor_declare(SparkyParser.For_declareContext ctx)
    { }
176.     /**
177.         * {@inheritDoc}
178.         *
179.         * <p>The default implementation does nothing.</p>
180.         */
181.     @Override public void exitFor_declare(SparkyParser.For_declareContext ctx) {
    }
182.     /**
183.         * {@inheritDoc}
184.         *
185.         * <p>The default implementation does nothing.</p>
186.         */
187.     @Override public void enterTerm(SparkyParser.TermContext ctx) { }
188.     /**
189.         * {@inheritDoc}
190.         *
191.         * <p>The default implementation does nothing.</p>
192.         */
193.     @Override public void exitTerm(SparkyParser.TermContext ctx) { }
194.     /**
195.         * {@inheritDoc}
196.         *
197.         * <p>The default implementation does nothing.</p>
198.         */
199.     @Override public void enterExpr(SparkyParser.ExprContext ctx) { }
200.     /**
201.         * {@inheritDoc}
202.         *
203.         * <p>The default implementation does nothing.</p>

```



```

204.         */
205.         @Override public void exitExpr(SparkyParser.ExprContext ctx) { }
206.         /**
207.          * {@inheritDoc}
208.          *
209.          * <p>The default implementation does nothing.</p>
210.          */
211.         @Override public void enterYesnostatement(SparkyParser.YesnostatementContext
    ctx) { }
212.         /**
213.          * {@inheritDoc}
214.          *
215.          * <p>The default implementation does nothing.</p>
216.          */
217.         @Override public void exitYesnostatement(SparkyParser.YesnostatementContext
    ctx) { }
218.         /**
219.          * {@inheritDoc}
220.          *
221.          * <p>The default implementation does nothing.</p>
222.          */
223.         @Override public void enterTernary_operator(SparkyParser.Ternary_operatorCon
    text ctx) { }
224.         /**
225.          * {@inheritDoc}
226.          *
227.          * <p>The default implementation does nothing.</p>
228.          */
229.         @Override public void exitTernary_operator(SparkyParser.Ternary_operatorCont
    ext ctx) { }
230.         /**
231.          * {@inheritDoc}
232.          *
233.          * <p>The default implementation does nothing.</p>
234.          */
235.         @Override public void enterPrint(SparkyParser.PrintContext ctx) { }
236.         /**
237.          * {@inheritDoc}
238.          *
239.          * <p>The default implementation does nothing.</p>
240.          */
241.         @Override public void exitPrint(SparkyParser.PrintContext ctx) { }
242.         /**
243.          * {@inheritDoc}
244.          *
245.          * <p>The default implementation does nothing.</p>
246.          */
247.         @Override public void enterWarna(SparkyParser.WarnContext ctx) { }
248.         /**
249.          * {@inheritDoc}
250.          *
251.          * <p>The default implementation does nothing.</p>
252.          */
253.         @Override public void exitWarna(SparkyParser.WarnContext ctx) { }
254.         /**
255.          * {@inheritDoc}
256.          *
257.          * <p>The default implementation does nothing.</p>
258.          */
259.         @Override public void enterHaina(SparkyParser.HainaContext ctx) { }
260.         /**

```

```

261.         * {@inheritDoc}
262.         *
263.         * <p>The default implementation does nothing.</p>
264.         */
265.     @Override public void exitHaina(SparkyParser.HainaContext ctx) { }
266.     /**
267.         * {@inheritDoc}
268.         *
269.         * <p>The default implementation does nothing.</p>
270.         */
271.     @Override public void enterDatatype(SparkyParser.DatatypeContext ctx) { }
272.     /**
273.         * {@inheritDoc}
274.         *
275.         * <p>The default implementation does nothing.</p>
276.         */
277.     @Override public void exitDatatype(SparkyParser.DatatypeContext ctx) { }
278.     /**
279.         * {@inheritDoc}
280.         *
281.         * <p>The default implementation does nothing.</p>
282.         */
283.     @Override public void enterStringdatatype(SparkyParser.StringdatatypeContext
284.         ctx) { }
285.     /**
286.         * {@inheritDoc}
287.         *
288.         * <p>The default implementation does nothing.</p>
289.         */
290.     @Override public void exitStringdatatype(SparkyParser.StringdatatypeContext
291.         ctx) { }
292.     /**
293.         * {@inheritDoc}
294.         *
295.         * <p>The default implementation does nothing.</p>
296.         */
297.     @Override public void enterBooleanvalue(SparkyParser.BooleanvalueContext ctx
298.         ) { }
299.     /**
300.         * {@inheritDoc}
301.         *
302.         * <p>The default implementation does nothing.</p>
303.         */
304.     @Override public void exitBooleanvalue(SparkyParser.BooleanvalueContext ctx)
305.         { }
306.     /**
307.         * {@inheritDoc}
308.         *
309.         * <p>The default implementation does nothing.</p>
310.         */
311.     @Override public void enterYup(SparkyParser.YupContext ctx) { }
312.     /**
313.         * {@inheritDoc}
314.         *
315.         * <p>The default implementation does nothing.</p>
316.         */
317.     @Override public void exitYup(SparkyParser.YupContext ctx) { }

```

```

318.         */
319.         @Override public void enterNope(SparkyParser.NopeContext ctx) { }
320.         /**
321.          * {@inheritDoc}
322.          *
323.          * <p>The default implementation does nothing.</p>
324.          */
325.         @Override public void exitNope(SparkyParser.NopeContext ctx) { }
326.
327.         /**
328.          * {@inheritDoc}
329.          *
330.          * <p>The default implementation does nothing.</p>
331.          */
332.         @Override public void enterEveryRule(ParserRuleContext ctx) { }
333.         /**
334.          * {@inheritDoc}
335.          *
336.          * <p>The default implementation does nothing.</p>
337.          */
338.         @Override public void exitEveryRule(ParserRuleContext ctx) { }
339.         /**
340.          * {@inheritDoc}
341.          *
342.          * <p>The default implementation does nothing.</p>
343.          */
344.         @Override public void visitTerminal(TerminalNode node) { }
345.         /**
346.          * {@inheritDoc}
347.          *
348.          * <p>The default implementation does nothing.</p>
349.          */
350.         @Override public void visitErrorNode(ErrorNode node) { }
351.     }

```

```

1. // Generated from Sparky.g4 by ANTLR 4.8
2. package sparky;
3.
4. import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;
5.
6. /**
7.  * This class provides an empty implementation of {@link SparkyVisitor},
8.  * which can be extended to create a visitor which only needs to handle a subset
9.  * of the available methods.
10.  *
11.  * @param <T> The return type of the visit operation. Use {@link Void} for
12.  * operations with no return type.
13.  */
14. public class SparkyBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements Sparky
    Visitor<T> {
15.     /**
16.      * {@inheritDoc}
17.      *
18.      * <p>The default implementation returns the result of calling
19.      * {@link #visitChildren} on {@code ctx}.</p>
20.      */

```

```

21.     @Override public T visitProgram(SparkyParser.ProgramContext ctx) { return visitChil
dren(ctx); }
22.     /**
23.      * {@inheritDoc}
24.      *
25.      * <p>The default implementation returns the result of calling
26.      * {@link #visitChildren} on {@code ctx}.</p>
27.      */
28.     @Override public T visitBall(SparkyParser.BallContext ctx) { return visitChildren(c
tx); }
29.     /**
30.      * {@inheritDoc}
31.      *
32.      * <p>The default implementation returns the result of calling
33.      * {@link #visitChildren} on {@code ctx}.</p>
34.      */
35.     @Override public T visitDeclare(SparkyParser.DeclareContext ctx) { return visitChil
dren(ctx); }
36.     /**
37.      * {@inheritDoc}
38.      *
39.      * <p>The default implementation returns the result of calling
40.      * {@link #visitChildren} on {@code ctx}.</p>
41.      */
42.     @Override public T visitExpression(SparkyParser.ExpressionContext ctx) { return vis
itChildren(ctx); }
43.     /**
44.      * {@inheritDoc}
45.      *
46.      * <p>The default implementation returns the result of calling
47.      * {@link #visitChildren} on {@code ctx}.</p>
48.      */
49.     @Override public T visitAssignment(SparkyParser.AssignmentContext ctx) { return vis
itChildren(ctx); }
50.     /**
51.      * {@inheritDoc}
52.      *
53.      * <p>The default implementation returns the result of calling
54.      * {@link #visitChildren} on {@code ctx}.</p>
55.      */
56.     @Override public T visitIfte(SparkyParser.IfteContext ctx) { return visitChildren(c
tx); }
57.     /**
58.      * {@inheritDoc}
59.      *
60.      * <p>The default implementation returns the result of calling
61.      * {@link #visitChildren} on {@code ctx}.</p>
62.      */
63.     @Override public T visitLoopum(SparkyParser.LoopumContext ctx) { return visitChildr
en(ctx); }
64.     /**
65.      * {@inheritDoc}
66.      *
67.      * <p>The default implementation returns the result of calling
68.      * {@link #visitChildren} on {@code ctx}.</p>
69.      */
70.     @Override public T visitLoop_for(SparkyParser.Loop_forContext ctx) { return visitCh
ildren(ctx); }
71.     /**
72.      * {@inheritDoc}
73.      *

```

```

74.     * <p>The default implementation returns the result of calling
75.     * {@link #visitChildren} on {@code ctx}.</p>
76.     */
77.     @Override public T visitLoop_while(SparkyParser.Loop_whileContext ctx) { return vis
    itChildren(ctx); }
78.     /**
79.     * {@inheritDoc}
80.     *
81.     * <p>The default implementation returns the result of calling
82.     * {@link #visitChildren} on {@code ctx}.</p>
83.     */
84.     @Override public T visitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx) { re
    turn visitChildren(ctx); }
85.     /**
86.     * {@inheritDoc}
87.     *
88.     * <p>The default implementation returns the result of calling
89.     * {@link #visitChildren} on {@code ctx}.</p>
90.     */
91.     @Override public T visitIn_loop(SparkyParser.In_loopContext ctx) { return visitChil
    dren(ctx); }
92.     /**
93.     * {@inheritDoc}
94.     *
95.     * <p>The default implementation returns the result of calling
96.     * {@link #visitChildren} on {@code ctx}.</p>
97.     */
98.     @Override public T visitFor_expr(SparkyParser.For_exprContext ctx) { return visitCh
    ildren(ctx); }
99.     /**
100.    * {@inheritDoc}
101.    *
102.    * <p>The default implementation returns the result of calling
103.    * {@link #visitChildren} on {@code ctx}.</p>
104.    */
105.    @Override public T visitFor_expression(SparkyParser.For_expressionContext ctx) {
    return visitChildren(ctx); }
106.    /**
107.    * {@inheritDoc}
108.    *
109.    * <p>The default implementation returns the result of calling
110.    * {@link #visitChildren} on {@code ctx}.</p>
111.    */
112.    @Override public T visitFor_declare(SparkyParser.For_declareContext ctx) { r
    eturn visitChildren(ctx); }
113.    /**
114.    * {@inheritDoc}
115.    *
116.    * <p>The default implementation returns the result of calling
117.    * {@link #visitChildren} on {@code ctx}.</p>
118.    */
119.    @Override public T visitTerm(SparkyParser.TermContext ctx) { return visitChi
    ldren(ctx); }
120.    /**
121.    * {@inheritDoc}
122.    *
123.    * <p>The default implementation returns the result of calling
124.    * {@link #visitChildren} on {@code ctx}.</p>
125.    */
126.    @Override public T visitExpr(SparkyParser.ExprContext ctx) { return visitChi
    ldren(ctx); }

```

```

127.         /**
128.         * {@inheritDoc}
129.         *
130.         * <p>The default implementation returns the result of calling
131.         * {@link #visitChildren} on {@code ctx}.</p>
132.         */
133.         @Override public T visitYesnostatement(SparkyParser.YesnostatementContext ctx) { return visitChildren(ctx); }
134.         /**
135.         * {@inheritDoc}
136.         *
137.         * <p>The default implementation returns the result of calling
138.         * {@link #visitChildren} on {@code ctx}.</p>
139.         */
140.         @Override public T visitTernary_operator(SparkyParser.Ternary_operatorContext ctx) { return visitChildren(ctx); }
141.         /**
142.         * {@inheritDoc}
143.         *
144.         * <p>The default implementation returns the result of calling
145.         * {@link #visitChildren} on {@code ctx}.</p>
146.         */
147.         @Override public T visitPrint(SparkyParser.PrintContext ctx) { return visitChildren(ctx); }
148.         /**
149.         * {@inheritDoc}
150.         *
151.         * <p>The default implementation returns the result of calling
152.         * {@link #visitChildren} on {@code ctx}.</p>
153.         */
154.         @Override public T visitWarna(SparkyParser.WarnaContext ctx) { return visitChildren(ctx); }
155.         /**
156.         * {@inheritDoc}
157.         *
158.         * <p>The default implementation returns the result of calling
159.         * {@link #visitChildren} on {@code ctx}.</p>
160.         */
161.         @Override public T visitHaina(SparkyParser.HainaContext ctx) { return visitChildren(ctx); }
162.         /**
163.         * {@inheritDoc}
164.         *
165.         * <p>The default implementation returns the result of calling
166.         * {@link #visitChildren} on {@code ctx}.</p>
167.         */
168.         @Override public T visitDatatype(SparkyParser.DatatypeContext ctx) { return visitChildren(ctx); }
169.         /**
170.         * {@inheritDoc}
171.         *
172.         * <p>The default implementation returns the result of calling
173.         * {@link #visitChildren} on {@code ctx}.</p>
174.         */
175.         @Override public T visitStringdatatype(SparkyParser.StringdatatypeContext ctx) { return visitChildren(ctx); }
176.         /**
177.         * {@inheritDoc}
178.         *
179.         * <p>The default implementation returns the result of calling
180.         * {@link #visitChildren} on {@code ctx}.</p>

```

```

181.         */
182.         @Override public T visitBooleanvalue(SparkyParser.BooleanvalueContext ctx) {
183.             return visitChildren(ctx); }
184.         /**
185.          * {@inheritDoc}
186.          * <p>The default implementation returns the result of calling
187.          * {@link #visitChildren} on {@code ctx}.</p>
188.          */
189.         @Override public T visitYup(SparkyParser.YupContext ctx) { return visitChild
190.             ren(ctx); }
191.         /**
192.          * {@inheritDoc}
193.          * <p>The default implementation returns the result of calling
194.          * {@link #visitChildren} on {@code ctx}.</p>
195.          */
196.         @Override public T visitNope(SparkyParser.NopeContext ctx) { return visitChi
197.             ldren(ctx); }
198.     }

```

```

1. // Generated from Sparky.g4 by ANTLR 4.8
2. package sparky;
3.
4. import org.antlr.v4.runtime.Lexer;
5. import org.antlr.v4.runtime.CharStream;
6. import org.antlr.v4.runtime.Token;
7. import org.antlr.v4.runtime.TokenStream;
8. import org.antlr.v4.runtime.*;
9. import org.antlr.v4.runtime.atn.*;
10. import org.antlr.v4.runtime.dfa.DFA;
11. import org.antlr.v4.runtime.misc.*;
12.
13. @SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
14. public class SparkyLexer extends Lexer {
15.     static { RuntimeMetaData.checkVersion("4.8", RuntimeMetaData.VERSION); }
16.
17.     protected static final DFA[] _decisionToDFA;
18.     protected static final PredictionContextCache _sharedContextCache =
19.         new PredictionContextCache();
20.     public static final int
21.         T__0=1, T__1=2, T__2=3, T__3=4, T__4=5, T__5=6, T__6=7, T__7=8, T__8=9,
22.         T__9=10, T__10=11, T__11=12, T__12=13, ANDOROPERATOR=14, AND=15, OR=16,
23.         NOT=17, LIVE=18, DIE=19, FI=20, YESNOOPERATOR=21, EQUALTO=22, ASSEQ=23,
24.         LESS_THAN=24, MORE_THAN=25, LESS_THAN_EQ=26, MORE_THAN_EQ=27, PLUS=28,
25.         MINUS=29, MUL=30, DIV=31, SEMICOLON=32, COMMA=33, LSmoothBrace=34, RSmoothBrace
26.         =35,
27.         LCurlyBrace=36, RCurlyBrace=37, DQ=38, STRINGLITERAL=39, HAINA=40, INTEGER=41,
28.         STRING=42, DOUBLE=43, DECIMAL=44, CHAR=45, IF=46, WHILE=47, STUFF=48,
29.         NUMBER=49, WS=50;
30.     public static String[] channelNames = {
31.         "DEFAULT_TOKEN_CHANNEL", "HIDDEN"
32.     };
33.     public static String[] modeNames = {
34.         "DEFAULT_MODE"
35.     };

```

```

36.
37.     private static String[] makeRuleNames() {
38.         return new String[] {
39.             "T_0", "T_1", "T_2", "T_3", "T_4", "T_5", "T_6", "T_7", "T_8",
40.             "T_9", "T_10", "T_11", "T_12", "ANDOROPERATOR", "AND", "OR", "NOT",
41.             "LIVE", "DIE", "FI", "YESNOOPERATOR", "EQUALTO", "ASSEQ", "LESS_THAN",
42.             "MORE_THAN", "LESS_THAN_EQ", "MORE_THAN_EQ", "PLUS", "MINUS", "MUL",
43.             "DIV", "SEMICOLON", "COMMA", "LSmoothBrace", "RSmoothBrace", "LCurlyBrace",
44.             "RCurlyBrace", "DQ", "STRINGLITERAL", "HAINA", "INTEGER", "STRING", "DOUBLE
",
45.             "DECIMAL", "CHAR", "IF", "WHILE", "STUFF", "NUMBER", "WS"
46.         };
47.     }
48.     public static final String[] ruleNames = makeRuleNames();
49.
50.     private static String[] makeLiteralNames() {
51.         return new String[] {
52.             null, "warna", "'for'", "'in'", "'range'", "'?'", "':'", "'print'",
53.             "'else'", "'bool'", "'yup'", "'nope'", "'true'", "'false'", null, "'and'",
54.             "'or'", "'not'", "'live'", "'die'", "'fi'", null, "'='", "'=='", "'<'",
55.             "'>'", "'<='", "'>='", "'+'", "'-'", "'*'", "'/'", "';'", "','", "'('",
56.             "')'", "'{'", "'}'" , "'\"'", null, "haina", "int", "string", "double
",
57.             "float", "char", "if", "while"
58.         };
59.     }
60.     private static final String[] _LITERAL_NAMES = makeLiteralNames();
61.     private static String[] makeSymbolicNames() {
62.         return new String[] {
63.             null, null, null, null, null, null, null, null, null, null, null, null,
64.             null, null, "ANDOROPERATOR", "AND", "OR", "NOT", "LIVE", "DIE", "FI",
65.             "YESNOOPERATOR", "EQUALTO", "ASSEQ", "LESS_THAN", "MORE_THAN", "LESS_THAN_E
Q",
66.             "MORE_THAN_EQ", "PLUS", "MINUS", "MUL", "DIV", "SEMICOLON", "COMMA",
67.             "LSmoothBrace", "RSmoothBrace", "LCurlyBrace", "RCurlyBrace", "DQ", "STRING
LITERAL",
68.             "HAINA", "INTEGER", "STRING", "DOUBLE", "DECIMAL", "CHAR", "IF", "WHILE",
69.             "STUFF", "NUMBER", "WS"
70.         };
71.     }
72.     private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
73.     public static final Vocabulary VOCABULARY = new VocabularyImpl(_LITERAL_NAMES, _SYM
BOLIC_NAMES);
74.
75.     /**
76.      * @deprecated Use {@link #VOCABULARY} instead.
77.      */
78.     @Deprecated
79.     public static final String[] tokenNames;
80.     static {
81.         tokenNames = new String[_SYMBOLIC_NAMES.length];
82.         for (int i = 0; i < tokenNames.length; i++) {
83.             tokenNames[i] = VOCABULARY.getLiteralName(i);
84.             if (tokenNames[i] == null) {
85.                 tokenNames[i] = VOCABULARY.getSymbolicName(i);
86.             }
87.
88.             if (tokenNames[i] == null) {

```



```

89.         tokenNames[i] = "<INVALID>";
90.     }
91. }
92. }
93.
94. @Override
95. @Deprecated
96. public String[] getTokenNames() {
97.     return tokenNames;
98. }
99.
100. @Override
101. public Vocabulary getVocabulary() {
102.     return VOCABULARY;
103. }
104.
105.
106. public SparkyLexer(CharStream input) {
107.     super(input);
108.     _interp = new LexerATNSimulator(this, _ATN, _decisionToDFA, _sharedContextC
109.     ache);
110. }
111.
112. @Override
113. public String getGrammarFileName() { return "Sparky.g4"; }
114.
115. @Override
116. public String[] getRuleNames() { return ruleNames; }
117.
118. @Override
119. public String getSerializedATN() { return _serializedATN; }
120.
121. @Override
122. public String[] getChannelNames() { return channelNames; }
123.
124. @Override
125. public String[] getModeNames() { return modeNames; }
126.
127. @Override
128. public ATN getATN() { return _ATN; }
129.
130. public static final String _serializedATN =
131.     "\3\u608b\ua72a\u8133\u9ed\u417c\u3be7\u7786\u5964\2\64\u012e\b\1\4\2"+
132.     "\t\2\4\3\t\3\4\4\t\4\4\5\t\5\4\6\t\6\4\7\t\7\4\b\t\b\4\t\t\t\4\n\t\n\4"
133.     +
134.     "\13\t\13\4\f\t\f\4\r\t\r\4\16\t\16\4\17\t\17\4\20\t\20\4\21\t\21\4\22"+
135.     "\t\22\4\23\t\23\4\24\t\24\4\25\t\25\4\26\t\26\4\27\t\27\4\30\t\30\4\31"
136.     +
137.     "\t\31\4\32\t\32\4\33\t\33\4\34\t\34\4\35\t\35\4\36\t\36\4\37\t\37\4 \t"
138.     +
139.     " \4!\t!\4"\t"\4#\t#\4$\t$\4%\t%\4&\t&\4'\t'\4(\t(\4)\t)\4*\t*\4+\t"
140.     +
141.     "+\4,\t,\4-\t-
142.     \4.\t.\4/\t/\4\60\t\60\4\61\t\61\4\62\t\62\4\63\t\63\3\2"+
143.     "\3\2\3\2\3\2\3\2\3\2\3\3\3\3\3\3\3\4\3\4\3\4\3\5\3\5\3\5\3\5\3\5\3"
144.     +
145.     "\5\3\6\3\6\3\7\3\7\3\b\3\b\3\b\3\b\3\b\3\b\3\t\3\t\3\t\3\t\3\t\3\n\3\n"
146.     +

```

[illegible]

170. "\u00df\3\2\2\2G\u00e1\3\2\2\2I\u00e3\3\2\2\2K\u00e5\3\2\2\2M\u00e7\3\2"
+
171. "\2\20\u00e9\3\2\2\2Q\u00f2\3\2\2\2S\u00f8\3\2\2\2U\u00fc\3\2\2\2W\u00103
"+
172. "\3\2\2\2Y\u010a\3\2\2\2[\u0110\3\2\2\2]\u0115\3\2\2\2_\u0118\3\2\2\2a"+
173. "\u011e\3\2\2\2c\u0126\3\2\2\2e\u012a\3\2\2\2gh\7y\2\2hi\7c\2\2ij\7t\2"+
174. "\2jk\7p\2\2kl\7c\2\2l\4\3\2\2\2mn\7h\2\2no\7q\2\2op\7t\2\2p\6\3\2\2\2"+
175. "qr\7k\2\2rs\7p\2\2s\b\3\2\2\2tu\7t\2\2uv\7c\2\2vw\7p\2\2wx\7i\2\2xy\7"+
176. "g\2\2y\n\3\2\2\2z{\7A\2\2{\f\3\2\2\2|}\7<\2\2}\16\3\2\2\2~\177\7r\2\2"+
177. "\177\u0080\7t\2\2\u0080\u0081\7k\2\2\u0081\u0082\7p\2\2\u0082\u0083\7"+
178. "v\2\2\u0083\20\3\2\2\2\u0084\u0085\7g\2\2\u0085\u0086\7n\2\2\u0086\u008
7"+
179. "\7u\2\2\u0087\u0088\7g\2\2\u0088\22\3\2\2\2\u0089\u008a\7d\2\2\u008a\u0
08b"+
180. "\7q\2\2\u008b\u008c\7q\2\2\u008c\u008d\7n\2\2\u008d\24\3\2\2\2\u008e\u0
08f"+
181. "\7{\2\2\u008f\u0090\7w\2\2\u0090\u0091\7r\2\2\u0091\26\3\2\2\2\u0092\u0
093"+
182. "\7p\2\2\u0093\u0094\7q\2\2\u0094\u0095\7r\2\2\u0095\u0096\7g\2\2\u0096"
+
183. "\30\3\2\2\2\u0097\u0098\7v\2\2\u0098\u0099\7t\2\2\u0099\u009a\7w\2\2\u0
09a"+
184. "\u009b\7g\2\2\u009b\32\3\2\2\2\u009c\u009d\7h\2\2\u009d\u009e\7c\2\2\u0
09e"+
185. "\u009f\7n\2\2\u009f\u00a0\7u\2\2\u00a0\u00a1\7g\2\2\u00a1\34\3\2\2\2\u0
0a2"+
186. "\u00a5\5\37\20\2\u00a3\u00a5\5!\21\2\u00a4\u00a2\3\2\2\2\u00a4\u00a3\3"
+
187. "\2\2\2\u00a5\36\3\2\2\2\u00a6\u00a7\7c\2\2\u00a7\u00a8\7p\2\2\u00a8\u00
a9"+
188. "\7f\2\2\u00a9 \3\2\2\2\u00aa\u00ab\7q\2\2\u00ab\u00ac\7t\2\2\u00ac"\3"
+
189. "\2\2\2\u00ad\u00ae\7p\2\2\u00ae\u00af\7q\2\2\u00af\u00b0\7v\2\2\u00b0"+
190. "\$\3\2\2\2\u00b1\u00b2\7N\2\2\u00b2\u00b3\7k\2\2\u00b3\u00b4\7x\2\2\u00b
4"+
191. "\u00b5\7g\2\2\u00b5&\3\2\2\2\u00b6\u00b7\7F\2\2\u00b7\u00b8\7k\2\2\u00b
8"+
192. "\u00b9\7g\2\2\u00b9(\3\2\2\2\u00ba\u00bb\7h\2\2\u00bb\u00bc\7k\2\2\u00b
c"+
193. "*\3\2\2\2\u00bd\u00c3\5/\30\2\u00be\u00c3\5\61\31\2\u00bf\u00c3\5\63\32
"+
194. "\2\u00c0\u00c3\5\65\33\2\u00c1\u00c3\5\67\34\2\u00c2\u00bd\3\2\2\2\u00c
2"+
195. "\u00be\3\2\2\2\u00c2\u00bf\3\2\2\2\u00c2\u00c0\3\2\2\2\u00c2\u00c1\3\2"
+
196. "\2\2\u00c3,\3\2\2\2\u00c4\u00c5\7?\2\2\u00c5.\3\2\2\2\u00c6\u00c7\7?\2"
+
197. "\2\u00c7\u00c8\7?\2\2\u00c8\60\3\2\2\2\u00c9\u00ca\7>\2\2\u00ca\62\3\2"
+
198. "\2\2\u00cb\u00cc\7@\2\2\u00cc\64\3\2\2\2\u00cd\u00ce\7>\2\2\u00ce\u00cf
"+
199. "\7?\2\2\u00cf\66\3\2\2\2\u00d0\u00d1\7@\2\2\u00d1\u00d2\7?\2\2\u00d28"+

```
200.         "\3\2\2\2\u00d3\u00d4\7-
201.         \2\2\u00d4:\3\2\2\2\u00d5\u00d6\7/\2\2\u00d6<\3"+
202.         "\2\2\2\u00d7\u00d8\7,\2\2\u00d8>\3\2\2\2\u00d9\u00da\7\61\2\2\u00da@\3"
203.         +
204.         "\2\2\2\u00db\u00dc\7=\2\2\u00dcB\3\2\2\2\u00dd\u00de\7.\2\2\u00deD\3\2"
205.         +
206.         "\2\2\u00df\u00e0\7*\2\2\u00e0F\3\2\2\2\u00e1\u00e2\7+\2\2\u00e2H\3\2\2"
207.         +
208.         "\2\u00e3\u00e4\7}\2\2\u00e4J\3\2\2\2\u00e5\u00e6\7\177\2\2\u00e6L\3\2"+
209.         "\2\2\u00e7\u00e8\7$\2\2\u00e8N\3\2\2\2\u00e9\u00ed\5M\' \2\2\u00ea\u00ec"+
210.         "\n\2\2\2\u00eb\u00ea\3\2\2\2\u00ec\u00ef\3\2\2\2\u00ed\u00eb\3\2\2\2\u0
211.         0ed"+
212.         "\u00ee\3\2\2\2\u00ee\u00f0\3\2\2\2\u00ef\u00ed\3\2\2\2\u00f0\u00f1\5M"+
213.         "\' \2\2\u00f1P\3\2\2\2\u00f2\u00f3\7j\2\2\u00f3\u00f4\7c\2\2\u00f4\u00f5"+
214.         "\7k\2\2\u00f5\u00f6\7p\2\2\u00f6\u00f7\7c\2\2\u00f7R\3\2\2\2\u00f8\u00f
215.         9"+
216.         "\7k\2\2\u00f9\u00fa\7p\2\2\u00fa\u00fb\7v\2\2\u00fbT\3\2\2\2\u00fc\u00f
217.         d"+
218.         "\7u\2\2\u00fd\u00fe\7v\2\2\u00fe\u00ff\7t\2\2\u00ff\u0100\7k\2\2\u0100"
219.         +
220.         "\u0101\7p\2\2\u0101\u0102\7i\2\2\u0102V\3\2\2\2\u0103\u0104\7f\2\2\u010
221.         4"+
222.         "\u0105\7q\2\2\u0105\u0106\7w\2\2\u0106\u0107\7d\2\2\u0107\u0108\7n\2\2"
223.         +
224.         "\u0108\u0109\7g\2\2\u0109X\3\2\2\2\u010a\u010b\7h\2\2\u010b\u010c\7n\2"
225.         +
226.         "\2\u010c\u010d\7q\2\2\u010d\u010e\7c\2\2\u010e\u010f\7v\2\2\u010fZ\3\2"
227.         +
228.         "\2\2\u0110\u0111\7e\2\2\u0111\u0112\7j\2\2\u0112\u0113\7c\2\2\u0113\u01
229.         14"+
230.         "\7t\2\2\u0114\\\3\2\2\2\u0115\u0116\7k\2\2\u0116\u0117\7h\2\2\u0117^\3"
231.         +
232.         "\2\2\2\u0118\u0119\7y\2\2\u0119\u011a\7j\2\2\u011a\u011b\7k\2\2\u011b"+
233.         "\u011c\7n\2\2\u011c\u011d\7g\2\2\u011d` \3\2\2\2\u011e\u0122\t\3\2\2\u01
234.         1f"+
235.         "\u0121\t\4\2\2\u0120\u011f\3\2\2\2\u0121\u0124\3\2\2\2\u0122\u0120\3\2"
236.         +
237.         "\2\2\u0122\u0123\3\2\2\2\u0123b\3\2\2\2\u0124\u0122\3\2\2\2\u0125\u0127
238.         "+
239.         "\t\5\2\2\u0126\u0125\3\2\2\2\u0127\u0128\3\2\2\2\u0128\u0126\3\2\2\2\u0
240.         128"+
241.         "\u0129\3\2\2\2\u0129d\3\2\2\2\u012a\u012b\t\6\2\2\u012b\u012c\3\2\2\2"+
242.         "\u012c\u012d\b\63\2\2\u012df\3\2\2\2\b\2\u00a4\u00c2\u00ed\u0122\u0128"
243.         +
244.         "\3\b\2\2";
245.         public static final ATN _ATN =
246.         new ATNDeserializer().deserialize(_serializedATN.toCharArray());
247.         static {
248.             _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
249.             for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
250.                 _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
251.             }
252.         }
253.     }
254. }
```

```

1. // Generated from Sparky.g4 by ANTLR 4.8
2. package sparky;
3.
4. import org.antlr.v4.runtime.tree.ParseTreeListener;
5.
6. /**
7.  * This interface defines a complete listener for a parse tree produced by
8.  * {@link SparkyParser}.
9.  */
10. public interface SparkyListener extends ParseTreeListener {
11.     /**
12.      * Enter a parse tree produced by {@link SparkyParser#program}.
13.      * @param ctx the parse tree
14.      */
15.     void enterProgram(SparkyParser.ProgramContext ctx);
16.     /**
17.      * Exit a parse tree produced by {@link SparkyParser#program}.
18.      * @param ctx the parse tree
19.      */
20.     void exitProgram(SparkyParser.ProgramContext ctx);
21.     /**
22.      * Enter a parse tree produced by {@link SparkyParser#ball}.
23.      * @param ctx the parse tree
24.      */
25.     void enterBall(SparkyParser.BallContext ctx);
26.     /**
27.      * Exit a parse tree produced by {@link SparkyParser#ball}.
28.      * @param ctx the parse tree
29.      */
30.     void exitBall(SparkyParser.BallContext ctx);
31.     /**
32.      * Enter a parse tree produced by {@link SparkyParser#declare}.
33.      * @param ctx the parse tree
34.      */
35.     void enterDeclare(SparkyParser.DeclareContext ctx);
36.     /**
37.      * Exit a parse tree produced by {@link SparkyParser#declare}.
38.      * @param ctx the parse tree
39.      */
40.     void exitDeclare(SparkyParser.DeclareContext ctx);
41.     /**
42.      * Enter a parse tree produced by {@link SparkyParser#expression}.
43.      * @param ctx the parse tree
44.      */
45.     void enterExpression(SparkyParser.ExpressionContext ctx);
46.     /**
47.      * Exit a parse tree produced by {@link SparkyParser#expression}.
48.      * @param ctx the parse tree
49.      */
50.     void exitExpression(SparkyParser.ExpressionContext ctx);
51.     /**
52.      * Enter a parse tree produced by {@link SparkyParser#assignment}.
53.      * @param ctx the parse tree
54.      */
55.     void enterAssignment(SparkyParser.AssignmentContext ctx);
56.     /**
57.      * Exit a parse tree produced by {@link SparkyParser#assignment}.
58.      * @param ctx the parse tree

```

```

59.     */
60. void exitAssignment(SparkyParser.AssignmentContext ctx);
61. /**
62.  * Enter a parse tree produced by {@link SparkyParser#ifte}.
63.  * @param ctx the parse tree
64.  */
65. void enterIfte(SparkyParser.IfteContext ctx);
66. /**
67.  * Exit a parse tree produced by {@link SparkyParser#ifte}.
68.  * @param ctx the parse tree
69.  */
70. void exitIfte(SparkyParser.IfteContext ctx);
71. /**
72.  * Enter a parse tree produced by {@link SparkyParser#loopum}.
73.  * @param ctx the parse tree
74.  */
75. void enterLoopum(SparkyParser.LoopumContext ctx);
76. /**
77.  * Exit a parse tree produced by {@link SparkyParser#loopum}.
78.  * @param ctx the parse tree
79.  */
80. void exitLoopum(SparkyParser.LoopumContext ctx);
81. /**
82.  * Enter a parse tree produced by {@link SparkyParser#loop_for}.
83.  * @param ctx the parse tree
84.  */
85. void enterLoop_for(SparkyParser.Loop_forContext ctx);
86. /**
87.  * Exit a parse tree produced by {@link SparkyParser#loop_for}.
88.  * @param ctx the parse tree
89.  */
90. void exitLoop_for(SparkyParser.Loop_forContext ctx);
91. /**
92.  * Enter a parse tree produced by {@link SparkyParser#loop_while}.
93.  * @param ctx the parse tree
94.  */
95. void enterLoop_while(SparkyParser.Loop_whileContext ctx);
96. /**
97.  * Exit a parse tree produced by {@link SparkyParser#loop_while}.
98.  * @param ctx the parse tree
99.  */
100. void exitLoop_while(SparkyParser.Loop_whileContext ctx);
101. /**
102.  * Enter a parse tree produced by {@link SparkyParser#loop_for_range}.
103.  * @param ctx the parse tree
104.  */
105. void enterLoop_for_range(SparkyParser.Loop_for_rangeContext ctx);
106. /**
107.  * Exit a parse tree produced by {@link SparkyParser#loop_for_range}.
108.  * @param ctx the parse tree
109.  */
110. void exitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx);
111. /**
112.  * Enter a parse tree produced by {@link SparkyParser#in_loop}.
113.  * @param ctx the parse tree
114.  */
115. void enterIn_loop(SparkyParser.In_loopContext ctx);
116. /**
117.  * Exit a parse tree produced by {@link SparkyParser#in_loop}.
118.  * @param ctx the parse tree
119.  */

```

```

120.     void exitIn_loop(SparkyParser.In_loopContext ctx);
121.     /**
122.      * Enter a parse tree produced by {@link SparkyParser#for_expr}.
123.      * @param ctx the parse tree
124.      */
125.     void enterFor_expr(SparkyParser.For_exprContext ctx);
126.     /**
127.      * Exit a parse tree produced by {@link SparkyParser#for_expr}.
128.      * @param ctx the parse tree
129.      */
130.     void exitFor_expr(SparkyParser.For_exprContext ctx);
131.     /**
132.      * Enter a parse tree produced by {@link SparkyParser#for_expression}.
133.      * @param ctx the parse tree
134.      */
135.     void enterFor_expression(SparkyParser.For_expressionContext ctx);
136.     /**
137.      * Exit a parse tree produced by {@link SparkyParser#for_expression}.
138.      * @param ctx the parse tree
139.      */
140.     void exitFor_expression(SparkyParser.For_expressionContext ctx);
141.     /**
142.      * Enter a parse tree produced by {@link SparkyParser#for_declare}.
143.      * @param ctx the parse tree
144.      */
145.     void enterFor_declare(SparkyParser.For_declareContext ctx);
146.     /**
147.      * Exit a parse tree produced by {@link SparkyParser#for_declare}.
148.      * @param ctx the parse tree
149.      */
150.     void exitFor_declare(SparkyParser.For_declareContext ctx);
151.     /**
152.      * Enter a parse tree produced by {@link SparkyParser#term}.
153.      * @param ctx the parse tree
154.      */
155.     void enterTerm(SparkyParser.TermContext ctx);
156.     /**
157.      * Exit a parse tree produced by {@link SparkyParser#term}.
158.      * @param ctx the parse tree
159.      */
160.     void exitTerm(SparkyParser.TermContext ctx);
161.     /**
162.      * Enter a parse tree produced by {@link SparkyParser#expr}.
163.      * @param ctx the parse tree
164.      */
165.     void enterExpr(SparkyParser.ExprContext ctx);
166.     /**
167.      * Exit a parse tree produced by {@link SparkyParser#expr}.
168.      * @param ctx the parse tree
169.      */
170.     void exitExpr(SparkyParser.ExprContext ctx);
171.     /**
172.      * Enter a parse tree produced by {@link SparkyParser#yesnostatement}.
173.      * @param ctx the parse tree
174.      */
175.     void enterYesnostatement(SparkyParser.YesnostatementContext ctx);
176.     /**
177.      * Exit a parse tree produced by {@link SparkyParser#yesnostatement}.
178.      * @param ctx the parse tree
179.      */
180.     void exitYesnostatement(SparkyParser.YesnostatementContext ctx);

```

```

181.      /**
182.       * Enter a parse tree produced by {@link SparkyParser#ternary_operator}.
183.       * @param ctx the parse tree
184.       */
185.      void enterTernary_operator(SparkyParser.Ternary_operatorContext ctx);
186.      /**
187.       * Exit a parse tree produced by {@link SparkyParser#ternary_operator}.
188.       * @param ctx the parse tree
189.       */
190.      void exitTernary_operator(SparkyParser.Ternary_operatorContext ctx);
191.      /**
192.       * Enter a parse tree produced by {@link SparkyParser#print}.
193.       * @param ctx the parse tree
194.       */
195.      void enterPrint(SparkyParser.PrintContext ctx);
196.      /**
197.       * Exit a parse tree produced by {@link SparkyParser#print}.
198.       * @param ctx the parse tree
199.       */
200.      void exitPrint(SparkyParser.PrintContext ctx);
201.      /**
202.       * Enter a parse tree produced by {@link SparkyParser#warna}.
203.       * @param ctx the parse tree
204.       */
205.      void enterWarna(SparkyParser.WarnContext ctx);
206.      /**
207.       * Exit a parse tree produced by {@link SparkyParser#warna}.
208.       * @param ctx the parse tree
209.       */
210.      void exitWarna(SparkyParser.WarnContext ctx);
211.      /**
212.       * Enter a parse tree produced by {@link SparkyParser#haina}.
213.       * @param ctx the parse tree
214.       */
215.      void enterHaina(SparkyParser.HainaContext ctx);
216.      /**
217.       * Exit a parse tree produced by {@link SparkyParser#haina}.
218.       * @param ctx the parse tree
219.       */
220.      void exitHaina(SparkyParser.HainaContext ctx);
221.      /**
222.       * Enter a parse tree produced by {@link SparkyParser#datatype}.
223.       * @param ctx the parse tree
224.       */
225.      void enterDatatype(SparkyParser.DatatypeContext ctx);
226.      /**
227.       * Exit a parse tree produced by {@link SparkyParser#datatype}.
228.       * @param ctx the parse tree
229.       */
230.      void exitDatatype(SparkyParser.DatatypeContext ctx);
231.      /**
232.       * Enter a parse tree produced by {@link SparkyParser#stringdatatype}.
233.       * @param ctx the parse tree
234.       */
235.      void enterStringdatatype(SparkyParser.StringdatatypeContext ctx);
236.      /**
237.       * Exit a parse tree produced by {@link SparkyParser#stringdatatype}.
238.       * @param ctx the parse tree
239.       */
240.      void exitStringdatatype(SparkyParser.StringdatatypeContext ctx);
241.      /**

```



```

242.         * Enter a parse tree produced by {@link SparkyParser#booleanvalue}.
243.         * @param ctx the parse tree
244.         */
245.         void enterBooleanvalue(SparkyParser.BooleanvalueContext ctx);
246.         /**
247.         * Exit a parse tree produced by {@link SparkyParser#booleanvalue}.
248.         * @param ctx the parse tree
249.         */
250.         void exitBooleanvalue(SparkyParser.BooleanvalueContext ctx);
251.         /**
252.         * Enter a parse tree produced by {@link SparkyParser#yup}.
253.         * @param ctx the parse tree
254.         */
255.         void enterYup(SparkyParser.YupContext ctx);
256.         /**
257.         * Exit a parse tree produced by {@link SparkyParser#yup}.
258.         * @param ctx the parse tree
259.         */
260.         void exitYup(SparkyParser.YupContext ctx);
261.         /**
262.         * Enter a parse tree produced by {@link SparkyParser#nope}.
263.         * @param ctx the parse tree
264.         */
265.         void enterNope(SparkyParser.NopeContext ctx);
266.         /**
267.         * Exit a parse tree produced by {@link SparkyParser#nope}.
268.         * @param ctx the parse tree
269.         */
270.         void exitNope(SparkyParser.NopeContext ctx);
271.     }

```

```

1. // Generated from Sparky.g4 by ANTLR 4.8
2. package sparky;
3.
4. import org.antlr.v4.runtime.atn.*;
5. import org.antlr.v4.runtime.dfa.DFA;
6. import org.antlr.v4.runtime.*;
7. import org.antlr.v4.runtime.misc.*;
8. import org.antlr.v4.runtime.tree.*;
9. import java.util.List;
10. import java.util.Iterator;
11. import java.util.ArrayList;
12.
13. @SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
14. public class SparkyParser extends Parser {
15.     static { RuntimeMetaData.checkVersion("4.8", RuntimeMetaData.VERSION); }
16.
17.     protected static final DFA[] _decisionToDFA;
18.     protected static final PredictionContextCache _sharedContextCache =
19.         new PredictionContextCache();
20.     public static final int
21.         T__0=1, T__1=2, T__2=3, T__3=4, T__4=5, T__5=6, T__6=7, T__7=8, T__8=9,
22.         T__9=10, T__10=11, T__11=12, T__12=13, ANDOROPERATOR=14, AND=15, OR=16,
23.         NOT=17, LIVE=18, DIE=19, FI=20, YESNOOPERATOR=21, EQUALTO=22, ASSEQ=23,
24.         LESS_THAN=24, MORE_THAN=25, LESS_THAN_EQ=26, MORE_THAN_EQ=27, PLUS=28,

```

```

25.         MINUS=29, MUL=30, DIV=31, SEMICOLON=32, COMMA=33, LSmoothBrace=34, RSmoothBrace
    =35,
26.         LCurlyBrace=36, RCurlyBrace=37, DQ=38, STRINGLITERAL=39, HAINA=40, INTEGER=41,
27.         STRING=42, DOUBLE=43, DECIMAL=44, CHAR=45, IF=46, WHILE=47, STUFF=48,
28.         NUMBER=49, WS=50;
29.     public static final int
30.         RULE_program = 0, RULE_ball = 1, RULE_declare = 2, RULE_expression = 3,
31.         RULE_assignment = 4, RULE_ifte = 5, RULE_loopum = 6, RULE_loop_for = 7,
32.         RULE_loop_while = 8, RULE_loop_for_range = 9, RULE_in_loop = 10, RULE_for_expr
    = 11,
33.         RULE_for_expression = 12, RULE_for_declare = 13, RULE_term = 14, RULE_expr = 15
    ,
34.         RULE_yesnostatement = 16, RULE_ternary_operator = 17, RULE_print = 18,
35.         RULE_warna = 19, RULE_haina = 20, RULE_datatype = 21, RULE_stringdatatype = 22,
36.         RULE_booleanvalue = 23, RULE_yup = 24, RULE_nope = 25;
37.     private static String[] makeRuleNames() {
38.         return new String[] {
39.             "program", "ball", "declare", "expression", "assignment", "ifte", "loopum",
40.             "loop_for", "loop_while", "loop_for_range", "in_loop", "for_expr", "for_exp
    resion",
41.             "for_declare", "term", "expr", "yesnostatement", "ternary_operator",
42.             "print", "warna", "haina", "datatype", "stringdatatype", "booleanvalue",
43.             "yup", "nope"
44.         };
45.     }
46.     public static final String[] ruleNames = makeRuleNames();
47.
48.     private static String[] makeLiteralNames() {
49.         return new String[] {
50.             null, "'warna'", "'for'", "'in'", "'range'", "'?'", "':'", "'print'",
51.             "'else'", "'bool'", "'yup'", "'nope'", "'true'", "'false'", null, "'and'",
52.             "'or'", "'not'", "'Live'", "'Die'", "'fi'", null, "'='", "'=='", "'<'",
53.             "'>'", "'<='", "'>='", "'+'", "'-'", "'*'", "'/'", "':'", "';'", "'('",
54.             "')'", "'{'", "'}'" , "'\\'", null, "'haina'", "'int'", "'string'", "'double
    '",
55.             "'float'", "'char'", "'if'", "'while'"
56.         };
57.     }
58.     private static final String[] _LITERAL_NAMES = makeLiteralNames();
59.     private static String[] makeSymbolicNames() {
60.         return new String[] {
61.             null, null, null, null, null, null, null, null, null, null, null,
62.             null, null, "ANDOROPERATOR", "AND", "OR", "NOT", "LIVE", "DIE", "FI",
63.             "YESNOOPERATOR", "EQUALTO", "ASSEQ", "LESS_THAN", "MORE_THAN", "LESS_THAN_E
    Q",
64.             "MORE_THAN_EQ", "PLUS", "MINUS", "MUL", "DIV", "SEMICOLON", "COMMA",
65.             "LSmoothBrace", "RSmoothBrace", "LCurlyBrace", "RCurlyBrace", "DQ", "STRING
    LITERAL",
66.             "HAINA", "INTEGER", "STRING", "DOUBLE", "DECIMAL", "CHAR", "IF", "WHILE",
67.             "STUFF", "NUMBER", "WS"
68.         };
69.     }
70.     private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
71.     public static final Vocabulary VOCABULARY = new VocabularyImpl(_LITERAL_NAMES, _SYM
    BOLIC_NAMES);
72.

```

```

73.  /**
74.   * @deprecated Use {@link #VOCABULARY} instead.
75.   */
76.   @Deprecated
77.   public static final String[] tokenNames;
78.   static {
79.       tokenNames = new String[_SYMBOLIC_NAMES.length];
80.       for (int i = 0; i < tokenNames.length; i++) {
81.           tokenNames[i] = VOCABULARY.getLiteralName(i);
82.           if (tokenNames[i] == null) {
83.               tokenNames[i] = VOCABULARY.getSymbolicName(i);
84.           }
85.
86.           if (tokenNames[i] == null) {
87.               tokenNames[i] = "<INVALID>";
88.           }
89.       }
90.   }
91.
92.   @Override
93.   @Deprecated
94.   public String[] getTokenNames() {
95.       return tokenNames;
96.   }
97.
98.   @Override
99.
100.      public Vocabulary getVocabulary() {
101.          return VOCABULARY;
102.      }
103.
104.      @Override
105.      public String getGrammarFileName() { return "Sparky.g4"; }
106.
107.      @Override
108.      public String[] getRuleNames() { return ruleNames; }
109.
110.      @Override
111.      public String getSerializedATN() { return _serializedATN; }
112.
113.      @Override
114.      public ATN getATN() { return _ATN; }
115.
116.      public SparkyParser(TokenStream input) {
117.          super(input);
118.          _interp = new ParserATNSimulator(this, _ATN, _decisionToDFA, _sharedContext
Cache);
119.      }
120.
121.      public static class ProgramContext extends ParserRuleContext {
122.          public TerminalNode LIVE() { return getToken(SparkyParser.LIVE, 0); }
123.          public BallContext ball() {
124.              return getRuleContext(BallContext.class, 0);
125.          }
126.          public TerminalNode DIE() { return getToken(SparkyParser.DIE, 0); }
127.          public ProgramContext(ParserRuleContext parent, int invokingState) {
128.              super(parent, invokingState);
129.          }
130.          @Override public int getRuleIndex() { return RULE_program; }
131.          @Override
132.          public void enterRule(ParseTreeListener listener) {

```

```

133.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
    .enterProgram(this);
134.     }
135.     @Override
136.     public void exitRule(ParseTreeListener listener) {
137.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
    .exitProgram(this);
138.     }
139.     @Override
140.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
141.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitProgram(this);
142.         else return visitor.visitChildren(this);
143.     }
144. }
145.
146.     public final ProgramContext program() throws RecognitionException {
147.         ProgramContext _localctx = new ProgramContext(_ctx, getState());
148.         enterRule(_localctx, 0, RULE_program);
149.         try {
150.             enterOuterAlt(_localctx, 1);
151.             {
152.                 setState(52);
153.                 match(LIVE);
154.                 setState(53);
155.                 ball();
156.                 setState(54);
157.                 match(DIE);
158.             }
159.         }
160.         catch (RecognitionException re) {
161.             _localctx.exception = re;
162.             _errHandler.reportError(this, re);
163.             _errHandler.recover(this, re);
164.         }
165.         finally {
166.             exitRule();
167.         }
168.         return _localctx;
169.     }
170.
171.     public static class BallContext extends ParserRuleContext {
172.         public List<ExpressionContext> expression() {
173.             return getRuleContexts(ExpressionContext.class);
174.         }
175.         public ExpressionContext expression(int i) {
176.             return getRuleContext(ExpressionContext.class,i);
177.         }
178.         public List<DeclareContext> declare() {
179.             return getRuleContexts(DeclareContext.class);
180.         }
181.         public DeclareContext declare(int i) {
182.             return getRuleContext(DeclareContext.class,i);
183.         }
184.         public BallContext(ParserRuleContext parent, int invokingState) {
185.             super(parent, invokingState);
186.         }
187.         @Override public int getRuleIndex() { return RULE_ball; }
188.         @Override
189.         public void enterRule(ParseTreeListener listener) {

```

```

190.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
    .enterBall(this);
191.     }
192.     @Override
193.     public void exitRule(ParseTreeListener listener) {
194.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
    .exitBall(this);
195.     }
196.     @Override
197.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
198.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
    ends T>)visitor).visitBall(this);
199.         else return visitor.visitChildren(this);
200.     }
201. }
202.
203.     public final BallContext ball() throws RecognitionException {
204.         BallContext _localctx = new BallContext(_ctx, getState());
205.         enterRule(_localctx, 2, RULE_ball);
206.         int _la;
207.         try {
208.             setState(74);
209.             _errHandler.sync(this);
210.             switch ( getInterpreter().adaptivePredict(_input,3,_ctx) ) {
211.             case 1:
212.                 enterOuterAlt(_localctx, 1);
213.                 {
214.                     setState(59);
215.                     _errHandler.sync(this);
216.                     _la = _input.LA(1);
217.                     while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << T__1) | (
    1L << T__6) | (1L << T__9) | (1L << T__10) | (1L << NOT) | (1L << IF) | (1L << WHILE) |
    (1L << STUFF) | (1L << NUMBER)))) != 0)) {
218.                         {
219.                         {
220.                             setState(56);
221.                             expression();
222.                         }
223.                     }
224.                     setState(61);
225.                     _errHandler.sync(this);
226.                     _la = _input.LA(1);
227.                 }
228.             }
229.             break;
230.             case 2:
231.                 enterOuterAlt(_localctx, 2);
232.                 {
233.                     setState(65);
234.                     _errHandler.sync(this);
235.                     _la = _input.LA(1);
236.                     while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << HAINA) |
    (1L << INTEGER) | (1L << STRING) | (1L << DOUBLE) | (1L << DECIMAL) | (1L << CHAR)))) !=
    0)) {
237.                         {
238.                         {
239.                             setState(62);
240.                             declare();
241.                         }
242.                     }
243.                     setState(67);

```

```

244.         _errHandler.sync(this);
245.         _la = _input.LA(1);
246.     }
247.     setState(71);
248.     _errHandler.sync(this);
249.     _la = _input.LA(1);
250.     while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << T__1) | (
1L << T__6) | (1L << T__9) | (1L << T__10) | (1L << NOT) | (1L << IF) | (1L << WHILE) |
(1L << STUFF) | (1L << NUMBER)))) != 0)) {
251.     {
252.     {
253.         setState(68);
254.         expression();
255.     }
256.     }
257.     setState(73);
258.     _errHandler.sync(this);
259.     _la = _input.LA(1);
260.     }
261.     }
262.     break;
263.     }
264. }
265. catch (RecognitionException re) {
266.     _localctx.exception = re;
267.     _errHandler.reportError(this, re);
268.     _errHandler.recover(this, re);
269. }
270. finally {
271.     exitRule();
272. }
273. return _localctx;
274. }
275.
276. public static class DeclareContext extends ParserRuleContext {
277.     public DatatypeContext datatype() {
278.         return getRuleContext(DatatypeContext.class,0);
279.     }
280.     public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
281.     public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0)
; }
282.     public TerminalNode NUMBER() { return getToken(SparkyParser.NUMBER, 0);
}
283.     public TerminalNode SEMICOLON() { return getToken(SparkyParser.SEMICOLON
, 0); }
284.     public TerminalNode HAINA() { return getToken(SparkyParser.HAINA, 0); }
285.     public BooleanvalueContext booleanvalue() {
286.         return getRuleContext(BooleanvalueContext.class,0);
287.     }
288.     public StringdatatypeContext stringdatatype() {
289.         return getRuleContext(StringdatatypeContext.class,0);
290.     }
291.     public TerminalNode STRINGLITERAL() { return getToken(SparkyParser.STRIN
GLITERAL, 0); }
292.     public DeclareContext(ParserRuleContext parent, int invokingState) {
293.         super(parent, invokingState);
294.     }
295.     @Override public int getRuleIndex() { return RULE_declare; }
296.     @Override

```

```

297.         public void enterRule(ParseTreeListener listener) {
298.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterDeclare(this);
299.         }
300.         @Override
301.         public void exitRule(ParseTreeListener listener) {
302.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitDeclare(this);
303.         }
304.         @Override
305.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
306.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitDeclare(this);
307.             else return visitor.visitChildren(this);
308.         }
309.     }
310.
311.     public final DeclareContext declare() throws RecognitionException {
312.         DeclareContext _localctx = new DeclareContext(_ctx, getState());
313.         enterRule(_localctx, 4, RULE_declare);
314.         try {
315.             setState(105);
316.             _errHandler.sync(this);
317.             switch ( getInterpreter().adaptivePredict(_input,4,_ctx) ) {
318.                 case 1:
319.                     enterOuterAlt(_localctx, 1);
320.                     {
321.                         {
322.                             setState(76);
323.                             datatype();
324.                             setState(77);
325.                             match(STUFF);
326.                             setState(78);
327.                             match(EQUALTO);
328.                             setState(79);
329.                             match(NUMBER);
330.                             setState(80);
331.                             match(SEMICOLON);
332.                         }
333.                     }
334.                     break;
335.                 case 2:
336.                     enterOuterAlt(_localctx, 2);
337.                     {
338.                         {
339.                             setState(82);
340.                             datatype();
341.                             setState(83);
342.                             match(STUFF);
343.                             setState(84);
344.                             match(SEMICOLON);
345.                         }
346.                     }
347.                     break;
348.                 case 3:
349.                     enterOuterAlt(_localctx, 3);
350.                     {
351.                         {
352.                             setState(86);
353.                             match(HAINA);
354.                             setState(87);

```

```

355.         match(STUFF);
356.         setState(88);
357.         match(EQUALTO);
358.         setState(89);
359.         booleanvalue();
360.         setState(90);
361.         match(SEMICOLON);
362.     }
363. }
364. break;
365. case 4:
366.     enterOuterAlt(_localctx, 4);
367.     {
368.     {
369.         setState(92);
370.         match(HAINA);
371.         setState(93);
372.         match(STUFF);
373.         setState(94);
374.         match(SEMICOLON);
375.     }
376.     }
377.     break;
378. case 5:
379.     enterOuterAlt(_localctx, 5);
380.     {
381.         setState(95);
382.         stringdatatype();
383.         setState(96);
384.         match(STUFF);
385.         setState(97);
386.         match(EQUALTO);
387.         setState(98);
388.         match(STRINGLITERAL);
389.         setState(99);
390.         match(SEMICOLON);
391.     }
392.     break;
393. case 6:
394.     enterOuterAlt(_localctx, 6);
395.     {
396.         setState(101);
397.         stringdatatype();
398.         setState(102);
399.         match(STUFF);
400.         setState(103);
401.         match(SEMICOLON);
402.     }
403.     break;
404.     }
405. }
406. catch (RecognitionException re) {
407.     _localctx.exception = re;
408.     _errHandler.reportError(this, re);
409.     _errHandler.recover(this, re);
410. }
411. finally {
412.     exitRule();
413. }
414. return _localctx;
415. }

```


[illegible]

```

473.         case 3:
474.             enterOuterAlt(_localctx, 3);
475.             {
476.                 setState(109);
477.                 loopum();
478.             }
479.             break;
480.         case 4:
481.             enterOuterAlt(_localctx, 4);
482.             {
483.                 setState(110);
484.                 ternary_operator();
485.             }
486.             break;
487.         case 5:
488.             enterOuterAlt(_localctx, 5);
489.             {
490.                 setState(111);
491.                 print();
492.             }
493.             break;
494.     }
495. }
496. catch (RecognitionException re) {
497.     _localctx.exception = re;
498.     _errHandler.reportError(this, re);
499.     _errHandler.recover(this, re);
500. }
501. finally {
502.     exitRule();
503. }
504. return _localctx;
505. }
506.
507. public static class AssignmentContext extends ParserRuleContext {
508.     public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
509.
510.     public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0); }
511.
512.     public ExprContext expr() {
513.         return getRuleContext(ExprContext.class,0);
514.     }
515.     public TerminalNode SEMICOLON() { return getToken(SparkyParser.SEMICOLON
516. , 0); }
517.     public YesnostatementContext yesnostatement() {
518.         return getRuleContext(YesnostatementContext.class,0);
519.     }
520.     public AssignmentContext(ParserRuleContext parent, int invokingState) {
521.         super(parent, invokingState);
522.     }
523.     @Override public int getRuleIndex() { return RULE_assignment; }
524.     @Override
525.     public void enterRule(ParseTreeListener listener) {
526.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
527. .enterAssignment(this);
528.     }
529.     @Override
530.     public void exitRule(ParseTreeListener listener) {
531.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
532. .exitAssignment(this);

```

```

528.         }
529.         @Override
530.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
531.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitAssignment(this);
532.             else return visitor.visitChildren(this);
533.         }
534.     }
535.
536.     public final AssignmentContext assignment() throws RecognitionException {
537.         AssignmentContext _localctx = new AssignmentContext(_ctx, getState());
538.         enterRule(_localctx, 8, RULE_assignment);
539.         try {
540.             setState(124);
541.             _errHandler.sync(this);
542.             switch ( getInterpreter().adaptivePredict(_input,6,_ctx) ) {
543.                 case 1:
544.                     enterOuterAlt(_localctx, 1);
545.                     {
546.                         setState(114);
547.                         match(STUFF);
548.                         setState(115);
549.                         match(EQUALTO);
550.                         setState(116);
551.                         expr();
552.                         setState(117);
553.                         match(SEMICOLON);
554.                     }
555.                     break;
556.                 case 2:
557.                     enterOuterAlt(_localctx, 2);
558.                     {
559.                         setState(119);
560.                         match(STUFF);
561.                         setState(120);
562.                         match(EQUALTO);
563.                         setState(121);
564.                         yesnostatement(0);
565.                         setState(122);
566.                         match(SEMICOLON);
567.                     }
568.                     break;
569.             }
570.         }
571.         catch (RecognitionException re) {
572.             _localctx.exception = re;
573.             _errHandler.reportError(this, re);
574.             _errHandler.recover(this, re);
575.         }
576.         finally {
577.             exitRule();
578.         }
579.         return _localctx;
580.     }
581.
582.     public static class IfteContext extends ParserRuleContext {
583.         public TerminalNode IF() { return getToken(SparkyParser.IF, 0); }
584.         public YesnostatementContext yesnostatement() {
585.             return getRuleContext(YesnostatementContext.class,0);
586.         }
587.         public List<In_loopContext> in_loop() {

```

```

588.         return getRuleContexts(In_loopContext.class);
589.     }
590.     public In_loopContext in_loop(int i) {
591.         return getRuleContext(In_loopContext.class,i);
592.     }
593.     public TerminalNode FI() { return getToken(SparkyParser.FI, 0); }
594.     public IfteContext(ParserRuleContext parent, int invokingState) {
595.         super(parent, invokingState);
596.     }
597.     @Override public int getRuleIndex() { return RULE_ifte; }
598.     @Override
599.     public void enterRule(ParseTreeListener listener) {
600.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .enterIfte(this);
601.     }
602.     @Override
603.     public void exitRule(ParseTreeListener listener) {
604.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .exitIfte(this);
605.     }
606.     @Override
607.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
608.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
        ends T>)visitor).visitIfte(this);
609.         else return visitor.visitChildren(this);
610.     }
611. }
612.
613.     public final IfteContext ifte() throws RecognitionException {
614.         IfteContext _localctx = new IfteContext(_ctx, getState());
615.         enterRule(_localctx, 10, RULE_ifte);
616.         int _la;
617.         try {
618.             enterOuterAlt(_localctx, 1);
619.             {
620.                 setState(126);
621.                 match(IF);
622.                 setState(127);
623.                 yesnoStatement(0);
624.                 setState(128);
625.                 in_loop();
626.                 setState(131);
627.                 _errHandler.sync(this);
628.                 _la = _input.LA(1);
629.                 if (_la==T__0) {
630.                     {
631.                         setState(129);
632.                         match(T__0);
633.                         setState(130);
634.                         in_loop();
635.                     }
636.                 }
637.
638.                 setState(133);
639.                 match(FI);
640.             }
641.         }
642.         catch (RecognitionException re) {
643.             _localctx.exception = re;
644.             _errHandler.reportError(this, re);
645.             _errHandler.recover(this, re);

```

```

646.     }
647.     finally {
648.         exitRule();
649.     }
650.     return _localctx;
651. }
652.
653. public static class LoopumContext extends ParserRuleContext {
654.     public Loop_forContext loop_for() {
655.         return getRuleContext(Loop_forContext.class,0);
656.     }
657.     public Loop_whileContext loop_while() {
658.         return getRuleContext(Loop_whileContext.class,0);
659.     }
660.     public Loop_for_rangeContext loop_for_range() {
661.         return getRuleContext(Loop_for_rangeContext.class,0);
662.     }
663.     public LoopumContext(ParserRuleContext parent, int invokingState) {
664.         super(parent, invokingState);
665.     }
666.     @Override public int getRuleIndex() { return RULE_loopum; }
667.     @Override
668.     public void enterRule(ParseTreeListener listener) {
669.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .enterLoopum(this);
670.     }
671.     @Override
672.     public void exitRule(ParseTreeListener listener) {
673.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .exitLoopum(this);
674.     }
675.     @Override
676.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
677.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
        ends T>)visitor).visitLoopum(this);
678.         else return visitor.visitChildren(this);
679.     }
680. }
681.
682. public final LoopumContext loopum() throws RecognitionException {
683.     LoopumContext _localctx = new LoopumContext(_ctx, getState());
684.     enterRule(_localctx, 12, RULE_loopum);
685.     try {
686.         setState(138);
687.         _errHandler.sync(this);
688.         switch ( getInterpreter().adaptivePredict(_input,8,_ctx) ) {
689.             case 1:
690.                 enterOuterAlt(_localctx, 1);
691.                 {
692.                     setState(135);
693.                     loop_for();
694.                 }
695.                 break;
696.             case 2:
697.                 enterOuterAlt(_localctx, 2);
698.                 {
699.                     setState(136);
700.                     loop_while();
701.                 }
702.                 break;
703.             case 3:

```

```

704.         enterOuterAlt(_localctx, 3);
705.         {
706.             setState(137);
707.             loop_for_range();
708.         }
709.         break;
710.     }
711. }
712. catch (RecognitionException re) {
713.     _localctx.exception = re;
714.     _errHandler.reportError(this, re);
715.     _errHandler.recover(this, re);
716. }
717. finally {
718.     exitRule();
719. }
720. return _localctx;
721. }
722.
723. public static class Loop_forContext extends ParserRuleContext {
724.     public TerminalNode LSmoothBrace() { return getToken(SparkyParser.LSmooth
hBrace, 0); }
725.     public List<TerminalNode> SEMICOLON() { return getTokens(SparkyParser.SEM
MICOLON); }
726.     public TerminalNode SEMICOLON(int i) {
727.         return getToken(SparkyParser.SEMICOLON, i);
728.     }
729.     public TerminalNode RSmoothBrace() { return getToken(SparkyParser.RSmooth
hBrace, 0); }
730.     public In_loopContext in_loop() {
731.         return getRuleContext(In_loopContext.class,0);
732.     }
733.     public For_declareContext for_declare() {
734.         return getRuleContext(For_declareContext.class,0);
735.     }
736.     public For_expressionContext for_expression() {
737.         return getRuleContext(For_expressionContext.class,0);
738.     }
739.     public For_exprContext for_expr() {
740.         return getRuleContext(For_exprContext.class,0);
741.     }
742.     public Loop_forContext(ParserRuleContext parent, int invokingState) {
743.         super(parent, invokingState);
744.     }
745.     @Override public int getRuleIndex() { return RULE_loop_for; }
746.     @Override
747.     public void enterRule(ParseTreeListener listener) {
748.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterLoop_for(this);
749.     }
750.     @Override
751.     public void exitRule(ParseTreeListener listener) {
752.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitLoop_for(this);
753.     }
754.     @Override
755.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
756.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitLoop_for(this);
757.         else return visitor.visitChildren(this);
758.     }

```

```

759.         }
760.
761.         public final Loop_forContext loop_for() throws RecognitionException {
762.             Loop_forContext _localctx = new Loop_forContext(_ctx, getState());
763.             enterRule(_localctx, 14, RULE_loop_for);
764.             int _la;
765.             try {
766.                 enterOuterAlt(_localctx, 1);
767.                 {
768.                     setState(140);
769.                     match(T__1);
770.                     setState(141);
771.                     match(LSmoothBrace);
772.                     setState(143);
773.                     _errHandler.sync(this);
774.                     _la = _input.LA(1);
775.                     if ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << HAINA) | (1L <<
INTEGER) | (1L << DOUBLE) | (1L << DECIMAL) | (1L << CHAR))) != 0)) {
776.                         {
777.                             setState(142);
778.                             for_declare();
779.                         }
780.                     }
781.
782.                     setState(145);
783.                     match(SEMICOLON);
784.                     setState(147);
785.                     _errHandler.sync(this);
786.                     _la = _input.LA(1);
787.                     if ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << NOT) | (1L << ST
UFF) | (1L << NUMBER))) != 0)) {
788.                         {
789.                             setState(146);
790.                             for_expression();
791.                         }
792.                     }
793.
794.                     setState(149);
795.                     match(SEMICOLON);
796.                     setState(151);
797.                     _errHandler.sync(this);
798.                     _la = _input.LA(1);
799.                     if (_la==STUFF) {
800.                         {
801.                             setState(150);
802.                             for_expr();
803.                         }
804.                     }
805.
806.                     setState(153);
807.                     match(RSmoothBrace);
808.                     setState(154);
809.                     in_loop();
810.                 }
811.             }
812.             catch (RecognitionException re) {
813.                 _localctx.exception = re;
814.                 _errHandler.reportError(this, re);
815.                 _errHandler.recover(this, re);
816.             }
817.             finally {

```

```

818.         exitRule();
819.     }
820.     return _localctx;
821. }
822.
823.     public static class Loop_whileContext extends ParserRuleContext {
824.         public TerminalNode WHILE() { return getToken(SparkyParser.WHILE, 0); }
825.
826.         public YesnostatementContext yesnostatement() {
827.             return getRuleContext(YesnostatementContext.class,0);
828.         }
829.         public In_loopContext in_loop() {
830.             return getRuleContext(In_loopContext.class,0);
831.         }
832.         public Loop_whileContext(ParserRuleContext parent, int invokingState) {
833.             super(parent, invokingState);
834.         }
835.         @Override public int getRuleIndex() { return RULE_loop_while; }
836.         @Override
837.         public void enterRule(ParseTreeListener listener) {
838.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
839. .enterLoop_while(this);
840.         }
841.         @Override
842.         public void exitRule(ParseTreeListener listener) {
843.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
844. .exitLoop_while(this);
845.         }
846.         @Override
847.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
848.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
849. ends T>)visitor).visitLoop_while(this);
850.             else return visitor.visitChildren(this);
851.         }
852.     }
853.
854.     public final Loop_whileContext loop_while() throws RecognitionException {
855.         Loop_whileContext _localctx = new Loop_whileContext(_ctx, getState());
856.         enterRule(_localctx, 16, RULE_loop_while);
857.         try {
858.             enterOuterAlt(_localctx, 1);
859.             {
860.                 setState(156);
861.                 match(WHILE);
862.                 setState(157);
863.                 yesnostatement(0);
864.                 setState(158);
865.                 in_loop();
866.             }
867.         }
868.         catch (RecognitionException re) {
869.             _localctx.exception = re;
870.             _errHandler.reportError(this, re);
871.             _errHandler.recover(this, re);
872.         }
873.         finally {
874.             exitRule();
875.         }
876.         return _localctx;
877.     }

```



```

874.
875.         public static class Loop_for_rangeContext extends ParserRuleContext {
876.             public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
877.
878.             public TerminalNode LSmoothBrace() { return getToken(SparkyParser.LSmooth
hBrace, 0); }
879.             public List<TerminalNode> NUMBER() { return getTokens(SparkyParser.NUMBE
R); }
880.             public TerminalNode NUMBER(int i) {
881.                 return getToken(SparkyParser.NUMBER, i);
882.             }
883.             public TerminalNode COMMA() { return getToken(SparkyParser.COMMA, 0); }
884.
885.             public TerminalNode RSmoothBrace() { return getToken(SparkyParser.RSmooth
hBrace, 0); }
886.             public In_loopContext in_loop() {
887.                 return getRuleContext(In_loopContext.class,0);
888.             }
889.             public Loop_for_rangeContext(ParserRuleContext parent, int invokingState
) {
890.                 super(parent, invokingState);
891.             }
892.             @Override public int getRuleIndex() { return RULE_loop_for_range; }
893.             @Override
894.             public void enterRule(ParseTreeListener listener) {
895.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
896.                 .enterLoop_for_range(this);
897.             }
898.             @Override
899.             public void exitRule(ParseTreeListener listener) {
900.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
901.                 .exitLoop_for_range(this);
902.             }
903.             @Override
904.             public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
905.                 if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitLoop_for_range(this);
906.                 else return visitor.visitChildren(this);
907.             }
908.         }
909.
910.         public final Loop_for_rangeContext loop_for_range() throws RecognitionExcept
ion {
911.             Loop_for_rangeContext _localctx = new Loop_for_rangeContext(_ctx, getSta
te());
912.             enterRule(_localctx, 18, RULE_loop_for_range);
913.             try {
914.                 enterOuterAlt(_localctx, 1);
915.                 {
916.                     setState(160);
917.                     match(T__1);
918.                     setState(161);
919.                     match(STUFF);
920.                     setState(162);
921.                     match(T__2);
922.                     setState(163);
923.                     match(T__3);
924.                     setState(164);
925.                     match(LSmoothBrace);
926.                     setState(165);
927.                     match(NUMBER);

```

```

924.         setState(166);
925.         match(COMMA);
926.         setState(167);
927.         match(NUMBER);
928.         setState(168);
929.         match(RSmoothBrace);
930.         setState(169);
931.         in_loop();
932.     }
933. }
934. catch (RecognitionException re) {
935.     _localctx.exception = re;
936.     _errHandler.reportError(this, re);
937.     _errHandler.recover(this, re);
938. }
939. finally {
940.     exitRule();
941. }
942. return _localctx;
943. }
944.
945. public static class In_loopContext extends ParserRuleContext {
946.     public TerminalNode LCurlyBrace() { return getToken(SparkyParser.LCurlyB
race, 0); }
947.     public BallContext ball() {
948.         return getRuleContext(BallContext.class,0);
949.     }
950.     public TerminalNode RCurlyBrace() { return getToken(SparkyParser.RCurlyB
race, 0); }
951.     public ExpressionContext expression() {
952.         return getRuleContext(ExpressionContext.class,0);
953.     }
954.     public In_loopContext(ParserRuleContext parent, int invokingState) {
955.         super(parent, invokingState);
956.     }
957.     @Override public int getRuleIndex() { return RULE_in_loop; }
958.     @Override
959.     public void enterRule(ParseTreeListener listener) {
960.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterIn_loop(this);
961.     }
962.     @Override
963.     public void exitRule(ParseTreeListener listener) {
964.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitIn_loop(this);
965.     }
966.     @Override
967.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
968.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitIn_loop(this);
969.         else return visitor.visitChildren(this);
970.     }
971. }
972.
973. public final In_loopContext in_loop() throws RecognitionException {
974.     In_loopContext _localctx = new In_loopContext(_ctx, getState());
975.     enterRule(_localctx, 20, RULE_in_loop);
976.     try {
977.         setState(176);
978.         _errHandler.sync(this);
979.         switch (_input.LA(1)) {

```

```

980.         case LCurlyBrace:
981.             enterOuterAlt(_localctx, 1);
982.             {
983.                 setState(171);
984.                 match(LCurlyBrace);
985.                 setState(172);
986.                 ball();
987.                 setState(173);
988.                 match(RCurlyBrace);
989.             }
990.             break;
991.         case T__1:
992.         case T__6:
993.         case T__9:
994.         case T__10:
995.         case NOT:
996.         case IF:
997.         case WHILE:
998.         case STUFF:
999.         case NUMBER:
1000.             enterOuterAlt(_localctx, 2);
1001.             {
1002.                 setState(175);
1003.                 expression();
1004.             }
1005.             break;
1006.         default:
1007.             throw new NoViableAltException(this);
1008.     }
1009. }
1010. catch (RecognitionException re) {
1011.     _localctx.exception = re;
1012.     _errHandler.reportError(this, re);
1013.     _errHandler.recover(this, re);
1014. }
1015. finally {
1016.     exitRule();
1017. }
1018. return _localctx;
1019. }
1020.
1021. public static class For_exprContext extends ParserRuleContext {
1022.     public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
1023.
1024.     public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0); }
1025. }
1026.
1027. public ExprContext expr() {
1028.     return getRuleContext(ExprContext.class, 0);
1029. }
1030. public For_exprContext(ParserRuleContext parent, int invokingState) {
1031.     super(parent, invokingState);
1032. }
1033. @Override public int getRuleIndex() { return RULE_for_expr; }
1034. @Override
1035. public void enterRule(ParseTreeListener listener) {
1036.     if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .enterFor_expr(this);
1037. }
1038. @Override
1039. public void exitRule(ParseTreeListener listener) {

```

```

1037.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .exitFor_expr(this);
1038.     }
1039.     @Override
1040.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1041.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
        ends T>)visitor).visitFor_expr(this);
1042.         else return visitor.visitChildren(this);
1043.     }
1044. }
1045.
1046.     public final For_exprContext for_expr() throws RecognitionException {
1047.         For_exprContext _localctx = new For_exprContext(_ctx, getState());
1048.         enterRule(_localctx, 22, RULE_for_expr);
1049.         try {
1050.             enterOuterAlt(_localctx, 1);
1051.             {
1052.                 setState(178);
1053.                 match(STUFF);
1054.                 setState(179);
1055.                 match(EQUALTO);
1056.                 setState(180);
1057.                 expr();
1058.             }
1059.         }
1060.         catch (RecognitionException re) {
1061.             _localctx.exception = re;
1062.             _errHandler.reportError(this, re);
1063.             _errHandler.recover(this, re);
1064.         }
1065.         finally {
1066.             exitRule();
1067.         }
1068.         return _localctx;
1069.     }
1070.
1071.     public static class For_expressionContext extends ParserRuleContext {
1072.         public List<ExprContext> expr() {
1073.             return getRuleContexts(ExprContext.class);
1074.         }
1075.         public ExprContext expr(int i) {
1076.             return getRuleContext(ExprContext.class,i);
1077.         }
1078.         public TerminalNode YESNOOPERATOR() { return getToken(SparkyParser.YESNO
        OPERATOR, 0); }
1079.         public For_expressionContext(ParserRuleContext parent, int invokingState
        ) {
1080.             super(parent, invokingState);
1081.         }
1082.         @Override public int getRuleIndex() { return RULE_for_expression; }
1083.         @Override
1084.         public void enterRule(ParseTreeListener listener) {
1085.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .enterFor_expression(this);
1086.         }
1087.         @Override
1088.         public void exitRule(ParseTreeListener listener) {
1089.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
        .exitFor_expression(this);
1090.         }
1091.         @Override

```

```

1092.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1093.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitFor_expression(this);
1094.             else return visitor.visitChildren(this);
1095.         }
1096.     }
1097.
1098.     public final For_expressionContext for_expression() throws RecognitionExcept
ion {
1099.         For_expressionContext _localctx = new For_expressionContext(_ctx, getSta
te());
1100.         enterRule(_localctx, 24, RULE_for_expression);
1101.         try {
1102.             enterOuterAlt(_localctx, 1);
1103.             {
1104.                 setState(182);
1105.                 expr();
1106.                 setState(183);
1107.                 match(YESNOOPERATOR);
1108.                 setState(184);
1109.                 expr();
1110.             }
1111.         }
1112.         catch (RecognitionException re) {
1113.             _localctx.exception = re;
1114.             _errHandler.reportError(this, re);
1115.             _errHandler.recover(this, re);
1116.         }
1117.         finally {
1118.             exitRule();
1119.         }
1120.         return _localctx;
1121.     }
1122.
1123.     public static class For_declareContext extends ParserRuleContext {
1124.         public DatatypeContext datatype() {
1125.             return getRuleContext(DatatypeContext.class,0);
1126.         }
1127.         public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
1128.         public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0)
; }
1129.         public TerminalNode NUMBER() { return getToken(SparkyParser.NUMBER, 0);
}
1130.         public For_declareContext(ParserRuleContext parent, int invokingState) {
1131.             super(parent, invokingState);
1132.         }
1133.         @Override public int getRuleIndex() { return RULE_for_declare; }
1134.         @Override
1135.         public void enterRule(ParseTreeListener listener) {
1136.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterFor_declare(this);
1137.         }
1138.         @Override
1139.         public void exitRule(ParseTreeListener listener) {
1140.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitFor_declare(this);
1141.         }
1142.         @Override
1143.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {

```

```

1144.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitFor_declare(this);
1145.         else return visitor.visitChildren(this);
1146.     }
1147. }
1148.
1149.     public final For_declareContext for_declare() throws RecognitionException {
1150.
1151.         For_declareContext _localctx = new For_declareContext(_ctx, getState());
1152.
1153.         enterRule(_localctx, 26, RULE_for_declare);
1154.         try {
1155.             enterOuterAlt(_localctx, 1);
1156.             {
1157.                 setState(186);
1158.                 datatype();
1159.                 setState(187);
1160.                 match(STUFF);
1161.                 setState(188);
1162.                 match(EQUALTO);
1163.                 setState(189);
1164.                 match(NUMBER);
1165.             }
1166.         }
1167.         catch (RecognitionException re) {
1168.             _localctx.exception = re;
1169.             _errHandler.reportError(this, re);
1170.             _errHandler.recover(this, re);
1171.         }
1172.         finally {
1173.             exitRule();
1174.         }
1175.         return _localctx;
1176.     }
1177.
1178.     public static class TermContext extends ParserRuleContext {
1179.         public Token op;
1180.         public TerminalNode NUMBER() { return getToken(SparkyParser.NUMBER, 0); }
1181.     }
1182.     public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
1183.
1184.     public TermContext term() {
1185.         return getRuleContext(TermContext.class,0);
1186.     }
1187.     public TerminalNode MUL() { return getToken(SparkyParser.MUL, 0); }
1188.     public TerminalNode DIV() { return getToken(SparkyParser.DIV, 0); }
1189.     public TermContext(ParserRuleContext parent, int invokingState) {
1190.         super(parent, invokingState);
1191.     }
1192.     @Override public int getRuleIndex() { return RULE_term; }
1193.     @Override
1194.     public void enterRule(ParseTreeListener listener) {
1195.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1196.         .enterTerm(this);
1197.     }
1198.     @Override
1199.     public void exitRule(ParseTreeListener listener) {
1200.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1201.         .exitTerm(this);
1202.     }
1203.     @Override

```

```

1198.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1199.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitTerm(this);
1200.             else return visitor.visitChildren(this);
1201.         }
1202.     }
1203.
1204.     public final TermContext term() throws RecognitionException {
1205.         TermContext _localctx = new TermContext(_ctx, getState());
1206.         enterRule(_localctx, 28, RULE_term);
1207.         int _la;
1208.         try {
1209.             setState(199);
1210.             _errHandler.sync(this);
1211.             switch ( getInterpreter().adaptivePredict(_input,13,_ctx) ) {
1212.             case 1:
1213.                 enterOuterAlt(_localctx, 1);
1214.                 {
1215.                     setState(191);
1216.                     match(NUMBER);
1217.                 }
1218.                 break;
1219.             case 2:
1220.                 enterOuterAlt(_localctx, 2);
1221.                 {
1222.                     setState(192);
1223.                     match(STUFF);
1224.                 }
1225.                 break;
1226.             case 3:
1227.                 enterOuterAlt(_localctx, 3);
1228.                 {
1229.                     setState(193);
1230.                     match(STUFF);
1231.                     setState(194);
1232.                     ((TermContext)_localctx).op = _input.LT(1);
1233.                     _la = _input.LA(1);
1234.                     if ( !(_la==MUL || _la==DIV) ) {
1235.                         ((TermContext)_localctx).op = (Token)_errHandler.recoverInli
ne(this);
1236.                     }
1237.                     else {
1238.                         if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
1239.                         _errHandler.reportMatch(this);
1240.                         consume();
1241.                     }
1242.                     setState(195);
1243.                     term();
1244.                 }
1245.                 break;
1246.             case 4:
1247.                 enterOuterAlt(_localctx, 4);
1248.                 {
1249.                     setState(196);
1250.                     match(NUMBER);
1251.                     setState(197);
1252.                     ((TermContext)_localctx).op = _input.LT(1);
1253.                     _la = _input.LA(1);
1254.                     if ( !(_la==MUL || _la==DIV) ) {
1255.                         ((TermContext)_localctx).op = (Token)_errHandler.recoverInli
ne(this);

```

```

1256.         }
1257.         else {
1258.             if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
1259.             _errHandler.reportMatch(this);
1260.             consume();
1261.         }
1262.         setState(198);
1263.         term();
1264.     }
1265.     break;
1266. }
1267. }
1268. catch (RecognitionException re) {
1269.     _localctx.exception = re;
1270.     _errHandler.reportError(this, re);
1271.     _errHandler.recover(this, re);
1272. }
1273. finally {
1274.     exitRule();
1275. }
1276. return _localctx;
1277. }
1278.
1279. public static class ExprContext extends ParserRuleContext {
1280.     public Token op;
1281.     public TermContext term() {
1282.         return getRuleContext(TermContext.class,0);
1283.     }
1284.     public ExprContext expr() {
1285.         return getRuleContext(ExprContext.class,0);
1286.     }
1287.     public TerminalNode PLUS() { return getToken(SparkyParser.PLUS, 0); }
1288.     public TerminalNode MINUS() { return getToken(SparkyParser.MINUS, 0); }
1289.     public TerminalNode NOT() { return getToken(SparkyParser.NOT, 0); }
1290.     public ExprContext(ParserRuleContext parent, int invokingState) {
1291.         super(parent, invokingState);
1292.     }
1293.     @Override public int getRuleIndex() { return RULE_expr; }
1294.     @Override
1295.     public void enterRule(ParseTreeListener listener) {
1296.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1297.             .enterExpr(this);
1298.     }
1299.     @Override
1300.     public void exitRule(ParseTreeListener listener) {
1301.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1302.             .exitExpr(this);
1303.     }
1304.     @Override
1305.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1306.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitExpr(this);
1307.         else return visitor.visitChildren(this);
1308.     }
1309. }
1310.
1311. public final ExprContext expr() throws RecognitionException {
1312.     ExprContext _localctx = new ExprContext(_ctx, getState());
1313.     enterRule(_localctx, 30, RULE_expr);
1314.     int _la;

```



```

1313.         try {
1314.             setState(208);
1315.             _errHandler.sync(this);
1316.             switch ( getInterpreter().adaptivePredict(_input,14,_ctx) ) {
1317.                 case 1:
1318.                     enterOuterAlt(_localctx, 1);
1319.                     {
1320.                         setState(201);
1321.                         term();
1322.                     }
1323.                     break;
1324.                 case 2:
1325.                     enterOuterAlt(_localctx, 2);
1326.                     {
1327.                         setState(202);
1328.                         term();
1329.                         setState(203);
1330.                         ((ExprContext)_localctx).op = _input.LT(1);
1331.                         _la = _input.LA(1);
1332.                         if ( !(_la==PLUS || _la==MINUS) ) {
1333.                             ((ExprContext)_localctx).op = (Token)_errHandler.recoverInli
1334. ne(this);
1335.                         }
1336.                         else {
1337.                             if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
1338.                             _errHandler.reportMatch(this);
1339.                             consume();
1340.                         }
1341.                         setState(204);
1342.                         expr();
1343.                     }
1344.                     break;
1345.                 case 3:
1346.                     enterOuterAlt(_localctx, 3);
1347.                     {
1348.                         setState(206);
1349.                         match(NOT);
1350.                         setState(207);
1351.                         expr();
1352.                     }
1353.                     break;
1354.             }
1355.         } catch (RecognitionException re) {
1356.             _localctx.exception = re;
1357.             _errHandler.reportError(this, re);
1358.             _errHandler.recover(this, re);
1359.         }
1360.         finally {
1361.             exitRule();
1362.         }
1363.         return _localctx;
1364.     }
1365.
1366.     public static class YesnostatementContext extends ParserRuleContext {
1367.         public BooleanvalueContext booleanvalue() {
1368.             return getRuleContext(BooleanvalueContext.class,0);
1369.         }
1370.         public List<ExprContext> expr() {
1371.             return getRuleContexts(ExprContext.class);
1372.         }

```

```

1373.         public ExprContext expr(int i) {
1374.             return getRuleContext(ExprContext.class,i);
1375.         }
1376.         public TerminalNode YESNOOPERATOR() { return getToken(SparkyParser.YESNO
OPERATOR, 0); }
1377.         public List<YesnostatementContext> yesnostatement() {
1378.             return getRuleContexts(YesnostatementContext.class);
1379.         }
1380.         public YesnostatementContext yesnostatement(int i) {
1381.             return getRuleContext(YesnostatementContext.class,i);
1382.         }
1383.         public TerminalNode ANDOROPERATOR() { return getToken(SparkyParser.ANDOR
OPERATOR, 0); }
1384.         public YesnostatementContext(ParserRuleContext parent, int invokingState
) {
1385.             super(parent, invokingState);
1386.         }
1387.         @Override public int getRuleIndex() { return RULE_yesnostatement; }
1388.         @Override
1389.         public void enterRule(ParseTreeListener listener) {
1390.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterYesnostatement(this);
1391.         }
1392.         @Override
1393.         public void exitRule(ParseTreeListener listener) {
1394.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitYesnostatement(this);
1395.         }
1396.         @Override
1397.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1398.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitYesnostatement(this);
1399.             else return visitor.visitChildren(this);
1400.         }
1401.     }
1402.
1403.     public final YesnostatementContext yesnostatement() throws RecognitionExcept
ion {
1404.         return yesnostatement(0);
1405.     }
1406.
1407.     private YesnostatementContext yesnostatement(int _p) throws RecognitionExcep
tion {
1408.         ParserRuleContext _parentctx = _ctx;
1409.         int _parentState = getState();
1410.         YesnostatementContext _localctx = new YesnostatementContext(_ctx, _paren
tState);
1411.         YesnostatementContext _prevctx = _localctx;
1412.         int _startState = 32;
1413.         enterRecursionRule(_localctx, 32, RULE_yesnostatement, _p);
1414.         try {
1415.             int _alt;
1416.             enterOuterAlt(_localctx, 1);
1417.             {
1418.                 setState(216);
1419.                 _errHandler.sync(this);
1420.                 switch (_input.LA(1)) {
1421.                     case T__9:
1422.                     case T__10:
1423.                     {
1424.                         setState(211);

```

```

1425.         booleanvalue();
1426.     }
1427.     break;
1428.     case NOT:
1429.     case STUFF:
1430.     case NUMBER:
1431.     {
1432.         setState(212);
1433.         expr();
1434.         setState(213);
1435.         match(YESNOOPERATOR);
1436.         setState(214);
1437.         expr();
1438.     }
1439.     break;
1440.     default:
1441.         throw new NoViableAltException(this);
1442.     }
1443.     _ctx.stop = _input.LT(-1);
1444.     setState(223);
1445.     _errHandler.sync(this);
1446.     _alt = getInterpreter().adaptivePredict(_input,16,_ctx);
1447.     while ( _alt!=2 && _alt!=org.antlr.v4.runtime.atn.ATN.INVALID_ALT_NUM
MBER ) {
1448.         if ( _alt==1 ) {
1449.             if ( _parseListeners!=null ) triggerExitRuleEvent();
1450.             _prevctx = _localctx;
1451.             {
1452.             {
1453.                 _localctx = new YesnostatementContext(_parentctx, _parentSta
te);
1454.                 pushNewRecursionContext(_localctx, _startState, RULE_yesnost
atement);
1455.                 setState(218);
1456.                 if (!(precpred(_ctx, 1))) throw new FailedPredicateException
(this, "precpred(_ctx, 1)");
1457.                 setState(219);
1458.                 match(ANDOROPERATOR);
1459.                 setState(220);
1460.                 yesnostatement(2);
1461.             }
1462.             }
1463.         }
1464.         setState(225);
1465.         _errHandler.sync(this);
1466.         _alt = getInterpreter().adaptivePredict(_input,16,_ctx);
1467.     }
1468. }
1469. }
1470. catch (RecognitionException re) {
1471.     _localctx.exception = re;
1472.     _errHandler.reportError(this, re);
1473.     _errHandler.recover(this, re);
1474. }
1475. finally {
1476.     unrollRecursionContexts(_parentctx);
1477. }
1478. return _localctx;
1479. }
1480.
1481. public static class Ternary_operatorContext extends ParserRuleContext {

```

```

1482.         public YesnostatementContext yesnostatement() {
1483.             return getRuleContext(YesnostatementContext.class,0);
1484.         }
1485.         public List<In_loopContext> in_loop() {
1486.             return getRuleContexts(In_loopContext.class);
1487.         }
1488.         public In_loopContext in_loop(int i) {
1489.             return getRuleContext(In_loopContext.class,i);
1490.         }
1491.         public Ternary_operatorContext(ParserRuleContext parent, int invokingSta
te) {
1492.             super(parent, invokingState);
1493.         }
1494.         @Override public int getRuleIndex() { return RULE_ternary_operator; }
1495.         @Override
1496.         public void enterRule(ParseTreeListener listener) {
1497.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterTernary_operator(this);
1498.         }
1499.         @Override
1500.         public void exitRule(ParseTreeListener listener) {
1501.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitTernary_operator(this);
1502.         }
1503.         @Override
1504.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1505.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitTernary_operator(this);
1506.             else return visitor.visitChildren(this);
1507.         }
1508.     }
1509.
1510.     public final Ternary_operatorContext ternary_operator() throws RecognitionEx
ception {
1511.         Ternary_operatorContext _localctx = new Ternary_operatorContext(_ctx, ge
tState());
1512.         enterRule(_localctx, 34, RULE_ternary_operator);
1513.         try {
1514.             enterOuterAlt(_localctx, 1);
1515.             {
1516.                 setState(226);
1517.                 yesnostatement(0);
1518.                 setState(227);
1519.                 match(T__4);
1520.                 setState(228);
1521.                 in_loop();
1522.                 setState(229);
1523.                 match(T__5);
1524.                 setState(230);
1525.                 in_loop();
1526.             }
1527.         }
1528.         catch (RecognitionException re) {
1529.             _localctx.exception = re;
1530.             _errHandler.reportError(this, re);
1531.             _errHandler.recover(this, re);
1532.         }
1533.         finally {
1534.             exitRule();
1535.         }
1536.         return _localctx;

```

```

1537.         }
1538.
1539.         public static class PrintContext extends ParserRuleContext {
1540.             public TerminalNode LSmoothBrace() { return getToken(SparkyParser.LSmoot
hBrace, 0); }
1541.             public ExprContext expr() {
1542.                 return getRuleContext(ExprContext.class,0);
1543.             }
1544.             public TerminalNode RSmoothBrace() { return getToken(SparkyParser.RSmoot
hBrace, 0); }
1545.             public TerminalNode SEMICOLON() { return getToken(SparkyParser.SEMICOLON
, 0); }
1546.             public PrintContext(ParserRuleContext parent, int invokingState) {
1547.                 super(parent, invokingState);
1548.             }
1549.             @Override public int getRuleIndex() { return RULE_print; }
1550.             @Override
1551.             public void enterRule(ParseTreeListener listener) {
1552.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterPrint(this);
1553.             }
1554.             @Override
1555.             public void exitRule(ParseTreeListener listener) {
1556.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitPrint(this);
1557.             }
1558.             @Override
1559.             public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1560.                 if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitPrint(this);
1561.                 else return visitor.visitChildren(this);
1562.             }
1563.         }
1564.
1565.         public final PrintContext print() throws RecognitionException {
1566.             PrintContext _localctx = new PrintContext(_ctx, getState());
1567.             enterRule(_localctx, 36, RULE_print);
1568.             try {
1569.                 enterOuterAlt(_localctx, 1);
1570.                 {
1571.                     setState(232);
1572.                     match(T__6);
1573.                     setState(233);
1574.                     match(LSmoothBrace);
1575.                     setState(234);
1576.                     expr();
1577.                     setState(235);
1578.                     match(RSmoothBrace);
1579.                     setState(236);
1580.                     match(SEMICOLON);
1581.                 }
1582.             }
1583.             catch (RecognitionException re) {
1584.                 _localctx.exception = re;
1585.                 _errHandler.reportError(this, re);
1586.                 _errHandler.recover(this, re);
1587.             }
1588.             finally {
1589.                 exitRule();
1590.             }
1591.             return _localctx;

```

```

1592.         }
1593.
1594.         public static class WarnaContext extends ParserRuleContext {
1595.             public WarnaContext(ParserRuleContext parent, int invokingState) {
1596.                 super(parent, invokingState);
1597.             }
1598.             @Override public int getRuleIndex() { return RULE_warna; }
1599.             @Override
1600.             public void enterRule(ParseTreeListener listener) {
1601.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1602.                 .enterWarna(this);
1603.             }
1604.             @Override
1605.             public void exitRule(ParseTreeListener listener) {
1606.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1607.                 .exitWarna(this);
1608.             }
1609.             @Override
1610.             public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1611.                 if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
1612.                 ends T>)visitor).visitWarna(this);
1613.                 else return visitor.visitChildren(this);
1614.             }
1615.         }
1616.
1617.         public final WarnaContext warna() throws RecognitionException {
1618.             WarnaContext _localctx = new WarnaContext(_ctx, getState());
1619.             enterRule(_localctx, 38, RULE_warna);
1620.             try {
1621.                 enterOuterAlt(_localctx, 1);
1622.                 {
1623.                     setState(238);
1624.                     match(T__7);
1625.                 }
1626.             }
1627.             catch (RecognitionException re) {
1628.                 _localctx.exception = re;
1629.                 _errHandler.reportError(this, re);
1630.                 _errHandler.recover(this, re);
1631.             }
1632.             finally {
1633.                 exitRule();
1634.             }
1635.             return _localctx;
1636.         }
1637.
1638.         public static class HainaContext extends ParserRuleContext {
1639.             public HainaContext(ParserRuleContext parent, int invokingState) {
1640.                 super(parent, invokingState);
1641.             }
1642.             @Override public int getRuleIndex() { return RULE_haina; }
1643.             @Override
1644.             public void enterRule(ParseTreeListener listener) {
1645.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1646.                 .enterHaina(this);
1647.             }
1648.             @Override
1649.             public void exitRule(ParseTreeListener listener) {
1650.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1651.                 .exitHaina(this);
1652.             }
1653.         }

```

```

1648.         @Override
1649.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1650.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitHaina(this);
1651.             else return visitor.visitChildren(this);
1652.         }
1653.     }
1654.
1655.     public final HainaContext haina() throws RecognitionException {
1656.         HainaContext _localctx = new HainaContext(_ctx, getState());
1657.         enterRule(_localctx, 40, RULE_haina);
1658.         try {
1659.             enterOuterAlt(_localctx, 1);
1660.             {
1661.                 setState(240);
1662.                 match(T__8);
1663.             }
1664.         }
1665.         catch (RecognitionException re) {
1666.             _localctx.exception = re;
1667.             _errHandler.reportError(this, re);
1668.             _errHandler.recover(this, re);
1669.         }
1670.         finally {
1671.             exitRule();
1672.         }
1673.         return _localctx;
1674.     }
1675.
1676.     public static class DatatypeContext extends ParserRuleContext {
1677.         public TerminalNode INTEGER() { return getToken(SparkyParser.INTEGER, 0)
; }
1678.         public TerminalNode DOUBLE() { return getToken(SparkyParser.DOUBLE, 0);
}
1679.         public TerminalNode DECIMAL() { return getToken(SparkyParser.DECIMAL, 0)
; }
1680.         public TerminalNode CHAR() { return getToken(SparkyParser.CHAR, 0); }
1681.         public TerminalNode HAINA() { return getToken(SparkyParser.HAINA, 0); }
1682.
1683.         public DatatypeContext(ParserRuleContext parent, int invokingState) {
super(parent, invokingState);
1684.     }
1685.     @Override public int getRuleIndex() { return RULE_datatype; }
1686.     @Override
1687.     public void enterRule(ParseTreeListener listener) {
1688.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterDatatype(this);
1689.     }
1690.     @Override
1691.     public void exitRule(ParseTreeListener listener) {
1692.         if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitDatatype(this);
1693.     }
1694.     @Override
1695.     public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1696.         if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitDatatype(this);
1697.         else return visitor.visitChildren(this);
1698.     }
1699. }
1700.

```

```

1701.         public final DatatypeContext datatype() throws RecognitionException {
1702.             DatatypeContext _localctx = new DatatypeContext(_ctx, getState());
1703.             enterRule(_localctx, 42, RULE_datatype);
1704.             int _la;
1705.             try {
1706.                 enterOuterAlt(_localctx, 1);
1707.                 {
1708.                     setState(242);
1709.                     _la = _input.LA(1);
1710.                     if ( !((( (_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << HAINA) | (1L
1711. << INTEGER) | (1L << DOUBLE) | (1L << DECIMAL) | (1L << CHAR))) != 0)) ) {
1712.                         _errHandler.recoverInline(this);
1713.                     }
1714.                     else {
1715.                         if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
1716.                         _errHandler.reportMatch(this);
1717.                         consume();
1718.                     }
1719.                 }
1720.             } catch (RecognitionException re) {
1721.                 _localctx.exception = re;
1722.                 _errHandler.reportError(this, re);
1723.                 _errHandler.recover(this, re);
1724.             }
1725.             finally {
1726.                 exitRule();
1727.             }
1728.             return _localctx;
1729.         }
1730.
1731.         public static class StringdatatypeContext extends ParserRuleContext {
1732.             public TerminalNode STRING() { return getToken(SparkyParser.STRING, 0);
1733.         }
1734.             public StringdatatypeContext(ParserRuleContext parent, int invokingState
1735. ) {
1736.                 super(parent, invokingState);
1737.             }
1738.             @Override public int getRuleIndex() { return RULE_stringdatatype; }
1739.             @Override
1740.             public void enterRule(ParseTreeListener listener) {
1741.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1742. .enterStringdatatype(this);
1743.             }
1744.             @Override
1745.             public void exitRule(ParseTreeListener listener) {
1746.                 if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1747. .exitStringdatatype(this);
1748.             }
1749.             @Override
1750.             public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1751.                 if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
1752. ends T>)visitor).visitStringdatatype(this);
1753.                 else return visitor.visitChildren(this);
1754.             }
1755.         }
1756.
1757.         public final StringdatatypeContext stringdatatype() throws RecognitionExcept
1758. ion {
1759.             StringdatatypeContext _localctx = new StringdatatypeContext(_ctx, getSta
1760. te());

```



```

1754.         enterRule(_localctx, 44, RULE_stringdatatype);
1755.         try {
1756.             enterOuterAlt(_localctx, 1);
1757.             {
1758.                 setState(244);
1759.                 match(STRING);
1760.             }
1761.         }
1762.         catch (RecognitionException re) {
1763.             _localctx.exception = re;
1764.             _errHandler.reportError(this, re);
1765.             _errHandler.recover(this, re);
1766.         }
1767.         finally {
1768.             exitRule();
1769.         }
1770.         return _localctx;
1771.     }
1772.
1773.     public static class BooleanvalueContext extends ParserRuleContext {
1774.         public BooleanvalueContext(ParserRuleContext parent, int invokingState)
1775.         {
1776.             super(parent, invokingState);
1777.         }
1778.         @Override public int getRuleIndex() { return RULE_booleanvalue; }
1779.         @Override
1780.         public void enterRule(ParseTreeListener listener) {
1781.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1782.             .enterBooleanvalue(this);
1783.         }
1784.         @Override
1785.         public void exitRule(ParseTreeListener listener) {
1786.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1787.             .exitBooleanvalue(this);
1788.         }
1789.         @Override
1790.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1791.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
1792.             ends T>)visitor).visitBooleanvalue(this);
1793.             else return visitor.visitChildren(this);
1794.         }
1795.     }
1796.
1797.     public final BooleanvalueContext booleanvalue() throws RecognitionException
1798.     {
1799.         BooleanvalueContext _localctx = new BooleanvalueContext(_ctx, getState()
1800.         );
1801.         enterRule(_localctx, 46, RULE_booleanvalue);
1802.         int _la;
1803.         try {
1804.             enterOuterAlt(_localctx, 1);
1805.             {
1806.                 setState(246);
1807.                 _la = _input.LA(1);
1808.                 if ( !(_la==T__9 || _la==T__10) ) {
1809.                     _errHandler.recoverInline(this);
1810.                 }
1811.                 else {
1812.                     if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
1813.                     _errHandler.reportMatch(this);
1814.                     consume();

```

```

1809.         }
1810.     }
1811. }
1812.     catch (RecognitionException re) {
1813.         _localctx.exception = re;
1814.         _errHandler.reportError(this, re);
1815.         _errHandler.recover(this, re);
1816.     }
1817.     finally {
1818.         exitRule();
1819.     }
1820.     return _localctx;
1821. }
1822.
1823.     public static class YupContext extends ParserRuleContext {
1824.         public YupContext(ParserRuleContext parent, int invokingState) {
1825.             super(parent, invokingState);
1826.         }
1827.         @Override public int getRuleIndex() { return RULE_yup; }
1828.         @Override
1829.         public void enterRule(ParseTreeListener listener) {
1830.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1831. .enterYup(this);
1832.         }
1833.         @Override
1834.         public void exitRule(ParseTreeListener listener) {
1835.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1836. .exitYup(this);
1837.         }
1838.         @Override
1839.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1840.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
1841. ends T>)visitor).visitYup(this);
1842.             else return visitor.visitChildren(this);
1843.         }
1844.     }
1845.
1846.     public final YupContext yup() throws RecognitionException {
1847.         YupContext _localctx = new YupContext(_ctx, getState());
1848.         enterRule(_localctx, 48, RULE_yup);
1849.         try {
1850.             enterOuterAlt(_localctx, 1);
1851.             {
1852.                 setState(248);
1853.                 match(T__11);
1854.             }
1855.         }
1856.         catch (RecognitionException re) {
1857.             _localctx.exception = re;
1858.             _errHandler.reportError(this, re);
1859.             _errHandler.recover(this, re);
1860.         }
1861.         finally {
1862.             exitRule();
1863.         }
1864.         return _localctx;
1865.     }
1866.
1867.     public static class NopeContext extends ParserRuleContext {
1868.         public NopeContext(ParserRuleContext parent, int invokingState) {
1869.             super(parent, invokingState);

```

```

1867.         }
1868.         @Override public int getRuleIndex() { return RULE_nope; }
1869.         @Override
1870.         public void enterRule(ParseTreeListener listener) {
1871.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1872. .enterNope(this);
1873.         }
1874.         @Override
1875.         public void exitRule(ParseTreeListener listener) {
1876.             if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
1877. .exitNope(this);
1878.         }
1879.         @Override
1880.         public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
1881.             if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
1882. ends T>)visitor).visitNope(this);
1883.             else return visitor.visitChildren(this);
1884.         }
1885.     }
1886.     public final NopeContext nope() throws RecognitionException {
1887.         NopeContext _localctx = new NopeContext(_ctx, getState());
1888.         enterRule(_localctx, 50, RULE_nope);
1889.         try {
1890.             enterOuterAlt(_localctx, 1);
1891.             {
1892.                 setState(250);
1893.                 match(T__12);
1894.             }
1895.         }
1896.         catch (RecognitionException re) {
1897.             _localctx.exception = re;
1898.             _errHandler.reportError(this, re);
1899.             _errHandler.recover(this, re);
1900.         }
1901.         finally {
1902.             exitRule();
1903.         }
1904.         return _localctx;
1905.     }
1906.     public boolean sempred(RuleContext _localctx, int ruleIndex, int predIndex)
1907.     {
1908.         switch (ruleIndex) {
1909.             case 16:
1910.                 return yesnostatement_sempred((YesnostatementContext)_localctx, pred
1911. Index);
1912.         }
1913.         return true;
1914.     }
1915.     private boolean yesnostatement_sempred(YesnostatementContext _localctx, int
1916. predIndex) {
1917.         switch (predIndex) {
1918.             case 0:
1919.                 return precpred(_ctx, 1);
1920.         }
1921.         return true;
1922.     }
1923.     public static final String _serializedATN =

```

[illegible]

1951. "\30\2\2QR\7\63\2\2RS\7\" \2\2S1\3\2\2\2TU\5, \27\2UV\7\62\2\2VW\7\" \2\2"+

1952. "wI\3\2\2\2XY\7*\2\2YZ\7\62\2\2Z[\7\30\2\2[\5\60\31\2\2\7\" \2\2]1\3"+

1953. "\2\2\2^_\7*\2\2_\7\62\2\2`1\7\" \2\2ab\5. \30\2bc\7\62\2\2cd\7\30\2\2d"+

1954. "e\7)\2\2ef\7\" \2\2f1\3\2\2\2gh\5. \30\2hi\7\62\2\2ij\7\" \2\2j1\3\2\2\2"+

1955. "kN\3\2\2\2kT\3\2\2\2kX\3\2\2\2k^\3\2\2\2ka\3\2\2\2kg\3\2\2\2l\7\3\2\2"+

1956. "\2ms\5\n\6\2ns\5\f\7\2os\5\16\b\2ps\5\$\23\2qs\5&\24\2rm\3\2\2\2rn\3\2"+

1957. "\2\2ro\3\2\2\2rp\3\2\2\2rq\3\2\2\2s\t\3\2\2\2tu\7\62\2\2uv\7\30\2\2vw"+

1958. "\5 \21\2wx\7\" \2\2x\177\3\2\2\2yz\7\62\2\2z{\7\30\2\2{| \5\" \22\2|}\7\" "

+
1959. "\2\2}\177\3\2\2\2~t\3\2\2\2~y\3\2\2\2\177\13\3\2\2\2\u0080\u0081\7\60"+

1960. "\2\2\u0081\u0082\5\" \22\2\u0082\u0085\5\26\f\2\u0083\u0084\7\3\2\2\u0084"+
4"+

1961. "\u0086\5\26\f\2\u0085\u0083\3\2\2\2\u0085\u0086\3\2\2\2\u0086\u0087\3"+

1962. "\2\2\2\u0087\u0088\7\26\2\2\u0088\r\3\2\2\2\u0089\u008d\5\20\t\2\u008a"+
+

1963. "\u008d\5\22\n\2\u008b\u008d\5\24\13\2\u008c\u0089\3\2\2\2\u008c\u008a"+

1964. "\3\2\2\2\u008c\u008b\3\2\2\2\u008d\17\3\2\2\2\u008e\u008f\7\4\2\2\u008f"+
"+

1965. "\u0091\7\$\2\2\u0090\u0092\5\34\17\2\u0091\u0090\3\2\2\2\u0091\u0092\3"+

1966. "\2\2\2\u0092\u0093\3\2\2\2\u0093\u0095\7\" \2\2\u0094\u0096\5\32\16\2\u0095"+
095"+

1967. "\u0094\3\2\2\2\u0095\u0096\3\2\2\2\u0096\u0097\3\2\2\2\u0097\u0099\7\" "
+

1968. "\2\2\u0098\u009a\5\30\r\2\u0099\u0098\3\2\2\2\u0099\u009a\3\2\2\2\u009a"+
"+

1969. "\u009b\3\2\2\2\u009b\u009c\7%\2\2\u009c\u009d\5\26\f\2\u009d\21\3\2\2"+

1970. "\2\u009e\u009f\7\61\2\2\u009f\u00a0\5\" \22\2\u00a0\u00a1\5\26\f\2\u00a1"+
"+

1971. "\23\3\2\2\2\u00a2\u00a3\7\4\2\2\u00a3\u00a4\7\62\2\2\u00a4\u00a5\7\5\2"+
+

1972. "\2\u00a5\u00a6\7\6\2\2\u00a6\u00a7\7\$\2\2\u00a7\u00a8\7\63\2\2\u00a8\u00a9"+
0a9"+

1973. "\7#\2\2\u00a9\u00aa\7\63\2\2\u00aa\u00ab\7%\2\2\u00ab\u00ac\5\26\f\2\u00ac"+
0ac"+

1974. "\25\3\2\2\2\u00ad\u00ae\7&\2\2\u00ae\u00af\5\4\3\2\u00af\u00b0\7\" \2\2"+
+

1975. "\u00b0\u00b3\3\2\2\2\u00b1\u00b3\5\b\5\2\u00b2\u00ad\3\2\2\2\u00b2\u00b3"+
1"+

1976. "\3\2\2\2\u00b3\27\3\2\2\2\u00b4\u00b5\7\62\2\2\u00b5\u00b6\7\30\2\2\u00b6"+
b6"+

1977. "\u00b7\5 \21\2\u00b7\31\3\2\2\2\u00b8\u00b9\5 \21\2\u00b9\u00ba\7\27\2"+
+

1978. "\2\u00ba\u00bb\5 \21\2\u00bb\33\3\2\2\2\u00bc\u00bd\5, \27\2\u00bd\u00be"+
"+

1979. "\7\62\2\2\u00be\u00bf\7\30\2\2\u00bf\u00c0\7\63\2\2\u00c0\35\3\2\2\2\u00c1"+
0c1"+

1980. "\u00ca\7\63\2\2\u00c2\u00ca\7\62\2\2\u00c3\u00c4\7\62\2\2\u00c4\u00c5"+

```

1981.      "\t\2\2\2\u00c5\u00ca\5\36\20\2\u00c6\u00c7\7\63\2\2\u00c7\u00c8\t\2\2"+
1982.      "\2\u00c8\u00ca\5\36\20\2\u00c9\u00c1\3\2\2\2\u00c9\u00c2\3\2\2\2\u00c9"
+
1983.      "\u00c3\3\2\2\2\u00c9\u00c6\3\2\2\2\u00ca\37\3\2\2\2\u00cb\u00d3\5\36\20
"+
1984.      "\2\u00cc\u00cd\5\36\20\2\u00cd\u00ce\t\3\2\2\u00ce\u00cf\5 \21\2\u00cf"
+
1985.      "\u00d3\3\2\2\2\u00d0\u00d1\7\23\2\2\u00d1\u00d3\5 \21\2\u00d2\u00cb\3"+
1986.      "\2\2\2\u00d2\u00cc\3\2\2\2\u00d2\u00d0\3\2\2\2\u00d3!\3\2\2\2\u00d4\u00
d5"+
1987.      "\b\22\1\2\u00d5\u00db\5\60\31\2\u00d6\u00d7\5 \21\2\u00d7\u00d8\7\27\2"
+
1988.      "\2\u00d8\u00d9\5 \21\2\u00d9\u00db\3\2\2\2\u00da\u00d4\3\2\2\2\u00da\u0
0d6"+
1989.      "\3\2\2\2\u00db\u00e1\3\2\2\2\u00dc\u00dd\f\3\2\2\u00dd\u00de\7\20\2\2"+
1990.      "\u00de\u00e0\5\"22\4\u00df\u00dc\3\2\2\2\u00e0\u00e3\3\2\2\2\u00e1\u00
df"+
1991.      "\3\2\2\2\u00e1\u00e2\3\2\2\2\u00e2#\3\2\2\2\u00e3\u00e1\3\2\2\2\u00e4"+
1992.      "\u00e5\5\"22\2\u00e5\u00e6\7\7\2\2\u00e6\u00e7\5\26\f\2\u00e7\u00e8\7"
+
1993.      "\b\2\2\u00e8\u00e9\5\26\f\2\u00e9%\3\2\2\2\u00ea\u00eb\7\t\2\2\u00eb\u0
0ec"+
1994.      "\7$\2\2\u00ec\u00ed\5 \21\2\u00ed\u00ee\7%\2\2\u00ee\u00ef\7\"2\2\u00e
f"+
1995.      "\'\3\2\2\2\u00f0\u00f1\7\n\2\2\u00f1)\3\2\2\2\u00f2\u00f3\7\13\2\2\u00f
3"+
1996.      "+\3\2\2\2\u00f4\u00f5\t\4\2\2\u00f5-
\3\2\2\2\u00f6\u00f7\7,\2\2\u00f7"+
1997.      "/\3\2\2\2\u00f8\u00f9\t\5\2\2\u00f9\61\3\2\2\2\u00fa\u00fb\7\16\2\2\u00
fb"+
1998.      "\63\3\2\2\2\u00fc\u00fd\7\17\2\2\u00fd\65\3\2\2\2\23=CILkr~\u0085\u008c
"+
1999.      "\u0091\u0095\u0099\u00b2\u00c9\u00d2\u00da\u00e1";
2000.      public static final ATN _ATN =
2001.      new ATNDeserializer().deserialize(_serializedATN.toCharArray());
2002.      static {
2003.      _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
2004.      for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
2005.      _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
2006.      }
2007.      }
2008.      }

```

```

1. // Generated from Sparky.g4 by ANTLR 4.8
2. package sparky;
3.
4. import org.antlr.v4.runtime.tree.ParseTreeVisitor;
5.
6. /**
7.  * This interface defines a complete generic visitor for a parse tree produced
8.  * by {@link SparkyParser}.
9.  *
10.  * @param <T> The return type of the visit operation. Use {@link Void} for
11.  * operations with no return type.

```

```

12. */
13. public interface SparkyVisitor<T> extends ParseTreeVisitor<T> {
14.     /**
15.      * Visit a parse tree produced by {@link SparkyParser#program}.
16.      * @param ctx the parse tree
17.      * @return the visitor result
18.      */
19.     T visitProgram(SparkyParser.ProgramContext ctx);
20.     /**
21.      * Visit a parse tree produced by {@link SparkyParser#ball}.
22.      * @param ctx the parse tree
23.      * @return the visitor result
24.      */
25.     T visitBall(SparkyParser.BallContext ctx);
26.     /**
27.      * Visit a parse tree produced by {@link SparkyParser#declare}.
28.      * @param ctx the parse tree
29.      * @return the visitor result
30.      */
31.     T visitDeclare(SparkyParser.DeclareContext ctx);
32.     /**
33.      * Visit a parse tree produced by {@link SparkyParser#expression}.
34.      * @param ctx the parse tree
35.      * @return the visitor result
36.      */
37.     T visitExpression(SparkyParser.ExpressionContext ctx);
38.     /**
39.      * Visit a parse tree produced by {@link SparkyParser#assignment}.
40.      * @param ctx the parse tree
41.      * @return the visitor result
42.      */
43.     T visitAssignment(SparkyParser.AssignmentContext ctx);
44.     /**
45.      * Visit a parse tree produced by {@link SparkyParser#ifte}.
46.      * @param ctx the parse tree
47.      * @return the visitor result
48.      */
49.     T visitIfte(SparkyParser.IfteContext ctx);
50.     /**
51.      * Visit a parse tree produced by {@link SparkyParser#loopum}.
52.      * @param ctx the parse tree
53.      * @return the visitor result
54.      */
55.     T visitLoopum(SparkyParser.LoopumContext ctx);
56.     /**
57.      * Visit a parse tree produced by {@link SparkyParser#loop_for}.
58.      * @param ctx the parse tree
59.      * @return the visitor result
60.      */
61.     T visitLoop_for(SparkyParser.Loop_forContext ctx);
62.     /**
63.      * Visit a parse tree produced by {@link SparkyParser#loop_while}.
64.      * @param ctx the parse tree
65.      * @return the visitor result
66.      */
67.     T visitLoop_while(SparkyParser.Loop_whileContext ctx);
68.     /**
69.      * Visit a parse tree produced by {@link SparkyParser#loop_for_range}.
70.      * @param ctx the parse tree
71.      * @return the visitor result
72.      */

```

```

73.     T visitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx);
74.     /**
75.      * Visit a parse tree produced by {@link SparkyParser#in_loop}.
76.      * @param ctx the parse tree
77.      * @return the visitor result
78.      */
79.     T visitIn_loop(SparkyParser.In_loopContext ctx);
80.     /**
81.      * Visit a parse tree produced by {@link SparkyParser#for_expr}.
82.      * @param ctx the parse tree
83.      * @return the visitor result
84.      */
85.     T visitFor_expr(SparkyParser.For_exprContext ctx);
86.     /**
87.      * Visit a parse tree produced by {@link SparkyParser#for_expression}.
88.      * @param ctx the parse tree
89.      * @return the visitor result
90.      */
91.     T visitFor_expression(SparkyParser.For_expressionContext ctx);
92.     /**
93.      * Visit a parse tree produced by {@link SparkyParser#for_declare}.
94.      * @param ctx the parse tree
95.      * @return the visitor result
96.      */
97.     T visitFor_declare(SparkyParser.For_declareContext ctx);
98.     /**
99.      * Visit a parse tree produced by {@link SparkyParser#term}.
100.      * @param ctx the parse tree
101.      * @return the visitor result
102.      */
103.     T visitTerm(SparkyParser.TermContext ctx);
104.     /**
105.      * Visit a parse tree produced by {@link SparkyParser#expr}.
106.      * @param ctx the parse tree
107.      * @return the visitor result
108.      */
109.     T visitExpr(SparkyParser.ExprContext ctx);
110.     /**
111.      * Visit a parse tree produced by {@link SparkyParser#yesnostatement}.
112.      * @param ctx the parse tree
113.      * @return the visitor result
114.      */
115.     T visitYesnostatement(SparkyParser.YesnostatementContext ctx);
116.     /**
117.      * Visit a parse tree produced by {@link SparkyParser#ternary_operator}.
118.      * @param ctx the parse tree
119.      * @return the visitor result
120.      */
121.     T visitTernary_operator(SparkyParser.Ternary_operatorContext ctx);
122.     /**
123.      * Visit a parse tree produced by {@link SparkyParser#print}.
124.      * @param ctx the parse tree
125.      * @return the visitor result
126.      */
127.     T visitPrint(SparkyParser.PrintContext ctx);
128.     /**
129.      * Visit a parse tree produced by {@link SparkyParser#warna}.
130.      * @param ctx the parse tree
131.      * @return the visitor result
132.      */
133.     T visitWarna(SparkyParser.WarnContext ctx);

```



```

134.      /**
135.       * Visit a parse tree produced by {@link SparkyParser#haina}.
136.       * @param ctx the parse tree
137.       * @return the visitor result
138.       */
139.      T visitHaina(SparkyParser.HainaContext ctx);
140.      /**
141.       * Visit a parse tree produced by {@link SparkyParser#datatype}.
142.       * @param ctx the parse tree
143.       * @return the visitor result
144.       */
145.      T visitDatatype(SparkyParser.DatatypeContext ctx);
146.      /**
147.       * Visit a parse tree produced by {@link SparkyParser#stringdatatype}.
148.       * @param ctx the parse tree
149.       * @return the visitor result
150.       */
151.      T visitStringdatatype(SparkyParser.StringdatatypeContext ctx);
152.      /**
153.       * Visit a parse tree produced by {@link SparkyParser#booleanvalue}.
154.       * @param ctx the parse tree
155.       * @return the visitor result
156.       */
157.      T visitBooleanvalue(SparkyParser.BooleanvalueContext ctx);
158.      /**
159.       * Visit a parse tree produced by {@link SparkyParser#yup}.
160.       * @param ctx the parse tree
161.       * @return the visitor result
162.       */
163.      T visitYup(SparkyParser.YupContext ctx);
164.      /**
165.       * Visit a parse tree produced by {@link SparkyParser#nope}.
166.       * @param ctx the parse tree
167.       * @return the visitor result
168.       */
169.      T visitNope(SparkyParser.NopeContext ctx);
170.  }

```

Sparky Compiler Code

```

1.  package sparkyCompiler;
2.
3.  import java.io.BufferedWriter;
4.  import java.io.File;
5.  import java.io.FileWriter;
6.  import java.io.IOException;
7.  import java.util.Arrays;
8.  import java.util.List;
9.
10. import org.antlr.v4.runtime.CharStream;
11. import org.antlr.v4.runtime.CharStreams;
12. import org.antlr.v4.runtime.CommonTokenStream;
13. import org.antlr.v4.runtime.tree.ParseTree;
14.
15. import sparky.SparkyLexer;
16. import sparky.SparkyParser;
17. import sparkyRuntime.IntermediateCodeReader;
18.

```

```

19. public class Compiler {
20.
21.     public static void main(String[] args) throws Exception {
22.         String inputIcfile = null;
23.
24.         try {
25.
26.             if (args.length > 0) {
27.                 String inputFileName = args[0];
28.                 inputIcfile = args[0];
29.
30.                 CharStream sourceCode = CharStreams.fromFileName(inputFileName);
31.                 SparkyLexer lx = new SparkyLexer(sourceCode);
32.                 CommonTokenStream tokenStream = new CommonTokenStream(lx);
33.                 SparkyParser parser = new SparkyParser(tokenStream);
34.                 ParseTree parseTree = parser.program();
35.
36.                 IntermediateCodeGenerator iCodeGen = new IntermediateCodeGenerator();
37.                 iCodeGen.visit(parseTree);
38.
39.                 List<String> iCodeArray = Arrays.asList(iCodeGen.getOutput().split("\\n
40.             ));
41.
42.             try {
43.                 File iCodeFile = new File(inputFileName.replace("sparky", "sparkyic
44.             ));
45.                 BufferedWriter bufferWriter = null;
46.                 FileWriter fileWriter = null;
47.                 if (iCodeArray.size() > 1) {
48.                     try {
49.                         if (iCodeFile.exists()) {
50.                             iCodeFile.delete();
51.                             iCodeFile.createNewFile();
52.                         } else {
53.                             iCodeFile.createNewFile();
54.                         }
55.
56.                         fileWriter = new FileWriter(iCodeFile);
57.                         bufferWriter = new BufferedWriter(fileWriter);
58.                         for (int i = 0; i < iCodeArray.size(); i++) {
59.                             bufferWriter.write(iCodeArray.get(i) + " " + "\n");
60.                         }
61.                     } catch (IOException e) {
62.                         e.printStackTrace();
63.                     }
64.                     finally {
65.                         try {
66.                             if (bufferWriter != null)
67.                                 bufferWriter.close();
68.                             if (fileWriter != null)
69.                                 fileWriter.close();
70.                         } catch (IOException ex) {
71.                             ex.printStackTrace();
72.                         }
73.                     }
74.                 }
75.             }
76.
77.             catch (Exception e) {

```

```

78.         e.printStackTrace();
79.     }
80. }
81. } catch (Exception e) {
82.     System.out.println("Input filename is incorrect");
83.     e.printStackTrace();
84. }
85. if (inputIcfile != null) {
86.     executeRuntime(inputIcfile + ".ic");
87. }
88. } else {
89.     System.out.println("Error File not created ");
90. }
91. }
92. }
93.
94. public static void executeRuntime(String filename) throws Exception {
95.     new IntermediateCodeReader(filename);
96.     // System.out.println("call to runtime");
97. }
98. }
99.
100. }

```

Intermediat Code Generator class

```

1. package sparkyCompiler;
2.
3. import Model.IntermediateCodeWriter;
4.
5. /**
6.  * In this class methods from SparkyBaseVisitor are overloaded for intermediate
7.  * code generation according to Sparky grammar.
8.  * @author Sayali Tanawade
9.  * @author Mayank Batra
10.  * @since April-20-2020
11.  * @version 2.0
12.  */
13.
14. import sparky.SparkyBaseVisitor;
15. import sparky.SparkyParser;
16. import sparkyRuntime.RuntimeConstantKeywords;
17.
18. public class IntermediateCodeGenerator extends SparkyBaseVisitor<Object> {
19.     String regexStr = "[0-9]*";
20.
21.     public String getOutput() {
22.         return IntermediateCodeWriter.getInstance().getIcOutput();
23.     }
24.
25.     @Override
26.     public Object visitProgram(SparkyParser.ProgramContext ctx) {
27.         return visitChildren(ctx);
28.     }
29.
30.     @Override
31.     public Object visitBall(SparkyParser.BallContext ctx) {
32.         return visitChildren(ctx);

```

```

33.     }
34.
35.     @Override
36.     public Object visitDeclare(SparkyParser.DeclareContext ctx) {
37.         IntermediateCodeWriter.getInstance().addOutput(
38.             RuntimeConstantKeywords.DECLARE + " " + ctx.getChild(0).getText() + " "
39.             + ctx.STUFF().getText());
40.         if (ctx.NUMBER() != null) {
41.             IntermediateCodeWriter.getInstance()
42.                 .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.NU
43.                 MBER().getText());
44.             IntermediateCodeWriter.getInstance()
45.                 .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STU
46.                 FF().getText());
47.         } else if (ctx.getChild(0).getText().contains("string")) {
48.             visit(ctx.stringdatatype());
49.             IntermediateCodeWriter.getInstance()
50.                 .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.ST
51.                 RINGLITERAL().getText());
52.             IntermediateCodeWriter.getInstance()
53.                 .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STU
54.                 FF().getText());
55.         }
56.         return null;
57.     }
58.
59.     @Override
60.     public Object visitExpression(SparkyParser.ExpressionContext ctx) {
61.         return visitChildren(ctx);
62.     }
63.
64.     @Override
65.     public Object visitExpr(SparkyParser.ExprContext ctx) {
66.
67.         if (ctx.getChildCount() == 1) {
68.             if (ctx.term().getChildCount() == 1) {
69.                 if (ctx.getChild(0).getText().matches(regexStr)) {
70.                     IntermediateCodeWriter.getInstance()
71.                         .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " "
72.                         + ctx.getChild(0).getText());
73.                 } else {
74.                     IntermediateCodeWriter.getInstance()
75.                         .addOutput(RuntimeConstantKeywords.GET + " " + ctx.getChild
76.                         (0).getText());
77.                 }
78.             } else {
79.                 IntermediateCodeWriter.getInstance()
80.                     .addOutput(RuntimeConstantKeywords.GET + " " + ctx.term().getCh
81.                     ild(0).getText());
82.                 if (ctx.term().getChild(2).getText().matches(regexStr)) {
83.                     IntermediateCodeWriter.getInstance().addOutput(
84.                         RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.term(
85.                         ).getChild(2).getText());
86.                 } else {
87.                     IntermediateCodeWriter.getInstance()
88.                         .addOutput(RuntimeConstantKeywords.GET + " " + ctx.term().g
89.                         etChild(2).getText());
90.                 }
91.             }
92.         }
93.     }

```

```

84.         getArithmeticOperator(ctx.term().getChild(1).getText());
85.     }
86. } else {
87.     IntermediateCodeWriter.getInstance()
88.         .addOutput(RuntimeConstantKeywords.GET + " " + ctx.getChild(0).getText());
89.     if (ctx.getChild(2).getText().matches(regexStr)) {
90.         IntermediateCodeWriter.getInstance()
91.             .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx
92.                 .getChild(2).getText());
93.     } else {
94.         IntermediateCodeWriter.getInstance()
95.             .addOutput(RuntimeConstantKeywords.GET + " " + ctx.getChild(2).
96.                 getText());
97.     }
98.     return null;
99. }
100.
101. public void getArithmeticOperator(String operator) {
102.     switch (operator) {
103.         case "+":
104.             IntermediateCodeWriter.getInstance()
105.                 .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeC
106.                     onstantKeywords.ADDITION);
107.             break;
108.         case "-":
109.             IntermediateCodeWriter.getInstance()
110.                 .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeC
111.                     onstantKeywords.SUBTRACTION);
112.             break;
113.         case "*":
114.             IntermediateCodeWriter.getInstance()
115.                 .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeC
116.                     onstantKeywords.MULTIPLICATION);
117.             break;
118.         case "/":
119.             IntermediateCodeWriter.getInstance()
120.                 .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeC
121.                     onstantKeywords.DIVISION);
122.             break;
123.     }
124. }
125.
126. @Override
127. public Object visitAssignment(SparkyParser.AssignmentContext ctx) {
128.     visit(ctx.expr());
129.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.P
130.         USH + " " + ctx.STUFF().getText());
131.     return null;
132. }
133.
134. @Override
135. public Object visitIfte(SparkyParser.IfteContext ctx) {
136.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
137.         FTE_START);
138.     if (ctx.yesnostatement().getText().contains("yup") || ctx.yesnostatement
139.         ().getText().contains("nup")) {
140.         IntermediateCodeWriter.getInstance()

```

```

135.         .addOutput(RuntimeConstantKeywords.CHECK_CONDITION + " " + c
    tx.yesnostatement().getText());
136.     } else {
137.         visit(ctx.yesnostatement());
138.     }
139.
140.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.C
    ONDITION_FALSE + " "
141.         + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.E
    LSE_START);
142.
143.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
    F_START);
144.
145.     visit(ctx.in_loop(0));
146.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
    F_END);
147.     if (ctx.in_loop(1) != null) {
148.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
    ds.ELSE_START);
149.         visit(ctx.in_loop(1));
150.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
    ds.ELSE_END);
151.     }
152.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
    FTE_END);
153.
154.     return null;
155. }
156.
157. @Override
158. public Object visitLoopum(SparkyParser.LoopumContext ctx) {
159.     return visitChildren(ctx);
160. }
161.
162. @Override
163. public Object visitLoop_for(SparkyParser.Loop_forContext ctx) {
164.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.F
    OR_START);
165.     visit(ctx.for_declare());
166.     visit(ctx.for_expression());
167.     visit(ctx.in_loop());
168.     visit(ctx.for_expr());
169.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.F
    OR_STOP);
170.
171.     return null;
172. }
173.
174. @Override
175. public Object visitFor_declare(SparkyParser.For_declareContext ctx) {
176.     IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.D
    ECLARE + " "
177.         + ctx.datatype().getChild(0).getText() + " " + ctx.STUFF().getTe
    xt());
178.     IntermediateCodeWriter.getInstance()
179.         .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx
    .NUMBER().getText());
180.     IntermediateCodeWriter.getInstance()
181.         .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.
    STUFF().getText());

```

```

182.
183.         return null;
184.     }
185.
186.     @Override
187.     public Object visitFor_expression(SparkyParser.For_expressionContext ctx) {
188.
189.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.F
OR_CONDITION_START);
190.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.G
ET + " " + ctx.expr(0).getText());
191.         if (ctx.expr(1).getText().matches(regexStr)) {
192.             IntermediateCodeWriter.getInstance()
193.                 .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " +
ctx.expr(1).getText());
194.         } else {
195.             IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
ds.GET + " " + ctx.expr(1).getText());
196.         }
197.
198.         IntermediateCodeWriter.getInstance()
199.             .addOutput(RuntimeConstantKeywords.COMPARE_OPERATOR + " " + ctx.
YESNOOPERATOR().getText());
200.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.C
ONDITION_FALSE + " "
201.             + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.F
OR_STOP);
202.
203.         return null;
204.     }
205.
206.     @Override
207.     public Object visitFor_expr(SparkyParser.For_exprContext ctx) {
208.         IntermediateCodeWriter.getInstance()
209.             .addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr().getChi
ld(0).getText());
210.         IntermediateCodeWriter.getInstance()
211.             .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx
.expr().getChild(2).getText());
212.
213.         getArithmeticOperator(ctx.expr().getChild(1).getText());
214.
215.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.P
USH + " " + ctx.STUFF().getText());
216.         IntermediateCodeWriter.getInstance()
217.             .addOutput(RuntimeConstantKeywords.JUMP + " " + RuntimeConstantK
eywords.FOR_CONDITION_START);
218.
219.         return null;
220.     }
221.
222.     @Override
223.     public Object visitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx) {
224.
225.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.F
OR_START);
226.         IntermediateCodeWriter.getInstance()
227.             .addOutput(RuntimeConstantKeywords.DECLARE + " int " + ctx.STUFF
().getText());

```

```

227.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
    NSTRUCTION_STORE + " " + ctx.NUMBER(0));
228.         IntermediateCodeWriter.getInstance()
229.             .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.
    STUFF().getText());
230.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.F
    OR_CONDITION_START);
231.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.G
    ET + " " + ctx.STUFF().getText());
232.         if (ctx.NUMBER(1).getText().matches(regexStr)) {
233.             IntermediateCodeWriter.getInstance()
234.                 .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " +
    ctx.NUMBER(1));
235.         } else {
236.             IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
    ds.GET + " " + ctx.NUMBER(1));
237.         }
238.
239.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.C
    OMPARE_OPERATOR + " " + "<");
240.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.C
    ONDITION_FALSE + " "
241.             + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.F
    OR_STOP);
242.         visit(ctx.in_loop());
243.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.G
    ET + " " + ctx.STUFF().getText());
244.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
    NSTRUCTION_STORE + " " + ctx.NUMBER(0));
245.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.O
    PERATOR + " " + "ADD");
246.         IntermediateCodeWriter.getInstance()
247.             .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.
    STUFF().getText());
248.         IntermediateCodeWriter.getInstance()
249.             .addOutput(RuntimeConstantKeywords.JUMP + " " + RuntimeConstantK
    eywords.FOR_CONDITION_START);
250.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.F
    OR_STOP);
251.         return null;
252.     }
253.
254.     @Override
255.     public Object visitLoop_while(SparkyParser.Loop_whileContext ctx) {
256.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.W
    HILE_BEGIN);
257.         if (ctx.yesnostatement().getText().contains("yup") || ctx.yesnostatement
    ().getText().contains("nup")) {
258.             IntermediateCodeWriter.getInstance()
259.                 .addOutput(RuntimeConstantKeywords.CHECK_CONDITION + " " + c
    tx.yesnostatement().getText());
260.         } else {
261.             visit(ctx.yesnostatement());
262.         }
263.
264.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.C
    ONDITION_FALSE + " "
265.             + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.W
    HILE_END);
266.         visit(ctx.in_loop());
267.         IntermediateCodeWriter.getInstance()

```



```

268.         .addOutput(RuntimeConstantKeywords.JUMP + " " + RuntimeConstantK
eywords.WHILE_BEGIN);
269.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.W
HILE_END);
270.
271.         return null;
272.     }
273.
274.     @Override
275.     public Object visitIn_loop(SparkyParser.In_loopContext ctx) {
276.         visit(ctx.ball());
277.         return null;
278.     }
279.
280.     @Override
281.     public Object visitTerm(SparkyParser.TermContext ctx) {
282.         return visitChildren(ctx);
283.     }
284.
285.     @Override
286.     public Object visitYesnostatement(SparkyParser.YesnostatementContext ctx) {
287.         if (ctx.expr(0) != null) {
288.             IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
ds.GET + " " + ctx.expr(0).getText());
289.             if (ctx.expr(1).getText().matches(regexStr)) {
290.                 IntermediateCodeWriter.getInstance()
291.                     .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + "
" + ctx.expr(1).getText());
292.             } else {
293.                 IntermediateCodeWriter.getInstance()
294.                     .addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr(
1).getText());
295.             }
296.             IntermediateCodeWriter.getInstance()
297.                 .addOutput(RuntimeConstantKeywords.COMPARE_OPERATOR + " " +
ctx.YESNOOPERATOR().getText());
298.             } else {
299.                 visit(ctx.yesnostatement(0));
300.                 visit(ctx.yesnostatement(1));
301.                 IntermediateCodeWriter.getInstance()
302.                     .addOutput(RuntimeConstantKeywords.AND_OR_OPERATOR + " " + c
tx.ANDOROPERATOR().getText());
303.             }
304.
305.             return null;
306.         }
307.
308.     @Override
309.     public Object visitTernary_operator(SparkyParser.Ternary_operatorContext ctx
) {
310.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
FTE_START);
311.         if (ctx.yesnostatement().getText().contains("yup") || ctx.yesnostatement
().getText().contains("nup")) {
312.             IntermediateCodeWriter.getInstance()
313.                 .addOutput(RuntimeConstantKeywords.CHECK_CONDITION + " " + c
tx.yesnostatement().getText());
314.             } else {
315.                 visit(ctx.yesnostatement());
316.             }

```

```

317.
318.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.C
CONDITION_FALSE + " "
319.         + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.E
LSE_START);
320.
321.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
F_START);
322.         visit(ctx.in_loop(0));
323.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
F_END);
324.         if (ctx.in_loop(1) != null) {
325.             IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
ds.ELSE_START);
326.             visit(ctx.in_loop(1));
327.             IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywor
ds.ELSE_END);
328.         }
329.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.I
FTE_END);
330.
331.         return null;
332.     }
333.
334.     @Override
335.     public Object visitPrint(SparkyParser.PrintContext ctx) {
336.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.G
ET + " " + ctx.expr().getText());
337.         IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.P
RINT);
338.
339.         return null;
340.     }
341.
342.     @Override
343.     public Object visitWarna(SparkyParser.WarnaContext ctx) {
344.         return visitChildren(ctx);
345.     }
346.
347.     @Override
348.     public Object visitHaina(SparkyParser.HainaContext ctx) {
349.         return visitChildren(ctx);
350.     }
351.
352.     @Override
353.     public Object visitDatatype(SparkyParser.DatatypeContext ctx) {
354.         return visitChildren(ctx);
355.     }
356.
357.     @Override
358.     public Object visitStringdatatype(SparkyParser.StringdatatypeContext ctx) {
359.         return visitChildren(ctx);
360.     }
361.
362.     @Override
363.     public Object visitYup(SparkyParser.YupContext ctx) {
364.         return visitChildren(ctx);
365.     }
366.
367.     @Override

```

```

368.         public Object visitNope(SparkyParser.NopeContext ctx) {
369.             return visitChildren(ctx);
370.         }
371.
372.     }

```

Run time Classes

```

1. package sparkyRuntime;
2.
3. /**
4.  *
5.  */
6. public class DataTypes {
7.
8.     Object value;
9.     DataTypes(Object value){
10.         this.value = value;
11.     }
12.
13.     public String toString() {
14.         return String.valueOf(value);
15.     }
16.
17.     public String checkDataType() {
18.         try {
19.             @SuppressWarnings("unused")
20.             int i =(Integer)value;
21.             return "int";
22.         }
23.         catch(ClassCastException fu) {
24.             return checkDouble();
25.         }
26.     }
27.
28.     private String checkDouble() {
29.         try {
30.             @SuppressWarnings("unused")
31.             Double i = (Double)value;
32.             return "double";
33.         }
34.         catch(ClassCastException fu) {
35.             return checkBoolean();
36.         }
37.     }
38.
39.     private String checkBoolean() {
40.         try {
41.             @SuppressWarnings("unused")
42.             Boolean bo = (Boolean)value;
43.             return "BOOLEAN";
44.         }
45.         catch(ClassCastException fu) {
46.             return "STRING";
47.         }
48.     }
49.
50.     public int Integer() {

```

```

51.         return (Integer)value;
52.     }
53.
54.     public double Double() {
55.         return (Double)value;
56.     }
57.
58.     public Boolean Boolean() {
59.         return (Boolean)value;
60.     }
61.
62.     public String String() {
63.         return (String)value;
64.     }
65. }

```

Intermediate Code Reader Class

```

1. package sparkyRuntime;
2.
3. import java.io.BufferedReader;
4. import java.io.IOException;
5. import java.io.StringReader;
6. import java.util.HashMap;
7. import java.util.Stack;
8.
9. import org.antlr.v4.runtime.CharStream;
10. import org.antlr.v4.runtime.CharStreams;
11.
12. public class IntermediateCodeReader {
13.
14.     public IntermediateCodeReader(String filename) throws Exception {
15.         program(filename);
16.     }
17.
18.     /**
19.      * Program function reads the Intermediate code file line by line. Variables are
20.      * stored in Hashmap having value of different data types. To handle this we
21.      * created class DataTypes.
22.      *
23.      * @throws Exception
24.      */
25.
26.     private String readFromICFile(String filename) throws IOException {
27.         String intermediateCode;
28.         CharStream code = CharStreams.fromFileName(filename);
29.         intermediateCode = (code.toString().replaceAll("\r", ""));
30.         return intermediateCode;
31.     }
32.
33.     private void program(String filename) throws Exception {
34.
35.         String fileContent = readFromICFile(filename);
36.         BufferedReader read = new BufferedReader(new StringReader(fileContent));
37.         String content;
38.         HashMap<String, DataTypes> map = new HashMap<String, DataTypes>();
39.         Stack<DataTypes> local = new Stack<DataTypes>();
40.         Stack<String> forVariable = new Stack<String>();

```

```

41.     int counter = 0;
42.     Stack<String> whileVariable = new Stack<String>();
43.     int counter1 = 0;
44.     try {
45.
46.         while ((content = read.readLine()) != null) {
47.             String[] line = content.split(" ");
48.             DataTypes value;
49.
50.             if (line[0].equals("DECLARE")) {
51.                 // If variable name is already present.
52.                 if (map.containsKey(line[2])) {
53.                     throw new Exception("Variable " + line[2] + " is already declar
54. ed.");
55.                 }
56.                 // If variable is declared without definition.
57.                 if (line.length < 5) {
58.                     map.put(line[2], null);
59.                     if (counter != 0) {
60.                         forVariable.push(line[2]);
61.                     }
62.                     if (counter1 != 0) {
63.                         whileVariable.push(line[2]);
64.                     }
65.                 }
66.                 // If variable is declared with definition.
67.                 else {
68.                     value = setValueDataTypes(line[4]);
69.                     if (line[1].equals(value.checkDataType())) {
70.                         map.put(line[2], value);
71.                         if (counter != 0) {
72.                             forVariable.push(line[2]);
73.                         }
74.                         if (counter1 != 0) {
75.                             whileVariable.push(line[2]);
76.                         }
77.                     } else {
78.
79.                         throw new Exception("Datatype Mismatch during Variable decl
80. aration");
81.                     }
82.                 }
83.             }
84.
85.             else if (line[0].equals("STORE")) {
86.                 if (line[1].charAt(0) != '"') {
87.                     value = setValueDataTypes(line[1]);
88.                     local.push(value);
89.                 } else if (line[1].charAt(0) == '"') {
90.                     String output = "";
91.                     int count = 0;
92.                     for (int i = 1; i < line.length; i++) {
93.                         if (i == 1 && line[i].length() != 1) {
94.                             output += line[i].substring(1, line[i].length()) + " ";
95.
96.                             count = 1;
97.                         } else if (i == line.length - 1 && line[i].length() != 1) {
98.
99.                             output += line[i].substring(0, line[i].length() - 1);

```

```

98.             } else if (i != 1 && i != line.length - 1) {
99.                 if (count == 0 && i == 2) {
100.                     output += " " + line[i] + " ";
101.                 } else {
102.                     output += line[i] + " ";
103.                 }
104.             }
105.         }
106.         value = setValueDataTypes(output);
107.         local.push(value);
108.     }
109. }
110.
111.         else if (line[0].equals("PUSH")) {
112.             if (!map.containsKey(line[1])) {
113.                 throw new Exception("Variable " + line[1] + " might not
have been declared.");
114.             } else {
115.                 DataTypes mapvar = map.get(line[1]);
116.                 DataTypes localvar = local.pop();
117.                 if (mapvar == null || mapvar.checkDataType().equals(localvar.checkDataType())) {
118.                     map.put(line[1], localvar);
119.                 } else {
120.                     throw new Exception("Datatype mismatch while assignment");
121.                 }
122.             }
123.         }
124.     }
125.
126.         else if (line[0].equals("GET")) {
127.             if (!map.containsKey(line[1])) {
128.                 throw new Exception("Variable " + line[1] + " might not
have been declared.");
129.             } else {
130.                 local.push(map.get(line[1]));
131.             }
132.         }
133.
134.         else if (line[0].equals("OPERATOR")) {
135.             operator(line[1], local);
136.         }
137.
138.         else if (line[0].equals("COMPARE_OPERATOR")) {
139.             compare(line[1], local);
140.         }
141.
142.         else if (line[0].equals("AND_OR_OPERATOR")) {
143.             if (line[1].equals("and")) {
144.                 Boolean decand = local.pop().Boolean();
145.                 DataTypes outand = new DataTypes(decand.equals(local.pop().Boolean()) ? decand : false);
146.                 local.push(outand);
147.             }
148.
149.             else if (line[1].equals("or")) {
150.                 Boolean decor = local.pop().Boolean();
151.                 DataTypes outor = new DataTypes(decor.equals(local.pop().Boolean()) ? decor : true);
152.                 local.push(utor);

```

```

153.         }
154.
155.         else if (line[1].equals("not")) {
156.             Boolean decnot = local.pop().Boolean();
157.             DataTypes outnot = new DataTypes(decnot.equals(true) ? f
158.         else : true);
159.             local.push(outnot);
160.         }
161.     }
162.     else if (line[0].equals("CONDITION_FALSE") && !local.pop().Boole
163. an()) {
164.         if (line[2].equals("ELSE_START")) {
165.             while ((content = read.readLine()) != null) {
166.                 line = content.split(" ");
167.                 if (line[0].equals("ELSE_START")) {
168.                     break;
169.                 }
170.             }
171.         } else if (line[2].equals("WHILEEND")) {
172.             while ((content = read.readLine()) != null) {
173.                 line = content.split(" ");
174.                 if (line[0].equals("WHILEEND")) {
175.                     break;
176.                 }
177.             }
178.             counter1 = 0;
179.             while (!whileVariable.empty()) {
180.                 map.remove(whileVariable.pop());
181.             }
182.         } else if (line[2].equals("FOR_STOP")) {
183.             while ((content = read.readLine()) != null) {
184.                 line = content.split(" ");
185.                 if (line[0].equals("FOR_STOP")) {
186.                     break;
187.                 }
188.             }
189.             counter = 0;
190.             while (!forVariable.empty()) {
191.                 map.remove(forVariable.pop());
192.             }
193.         }
194.     }
195.
196.     else if (line[0].equals("FOR_START")) {
197.         if (counter == 0) {
198.             counter = counter + 1;
199.         }
200.
201.     }
202.
203.     else if (line[0].equals("IF_END")) {
204.         while ((content = read.readLine()) != null) {
205.             line = content.split(" ");
206.             if (line[0].equals("ELSE_END")) {
207.                 break;
208.             }
209.         }
210.
211.     }

```

```

212.
213.         else if (line[0].equals("WHILEBEGIN")) {
214.             if (counter1 == 0) {
215.                 counter1 = counter1 + 1;
216.             }
217.
218.         }
219.
220.         else if (line[0].equals("JUMP")) {
221.
222.             if (line[1].equals("FOR_CONDITION_START")) {
223.                 fileContent = readFromICFile(filename);
224.                 read = new BufferedReader(new StringReader(fileContent))
;
225.                 while ((content = read.readLine()) != null) {
226.                     line = content.split(" ");
227.                     if (line[0].equals("FOR_CONDITION_START")) {
228.                         break;
229.                     }
230.                 }
231.             } else if (line[1].equals("WHILEBEGIN")) {
232.                 fileContent = readFromICFile(filename);
233.                 read = new BufferedReader(new StringReader(fileContent))
;
234.                 while ((content = read.readLine()) != null) {
235.                     line = content.split(" ");
236.                     if (line[0].equals("WHILEBEGIN")) {
237.                         break;
238.                     }
239.                 }
240.             }
241.         }
242.
243.         else if (line[0].equals("PRINT")) {
244.             if (local.isEmpty()) {
245.                 throw new Exception("Nothing to Print.");
246.             }
247.             System.out.println(local.pop().toString());
248.         }
249.     }
250. }
251.
252. catch (IOException fu) {
253.     fu.printStackTrace();
254. }
255.
256. }
257.
258. // Checks the data type of input and creates an object of class DataTypes.
259. private DataTypes setValueDataTypes(String string) {
260.     if (isInt(string)) {
261.         return new DataTypes(Integer.parseInt(string));
262.     } else if (isdouble(string)) {
263.         return new DataTypes(Double.parseDouble(string));
264.     } else if (isbool(string)) {
265.         return new DataTypes(Boolean.parseBoolean(string));
266.     } else {
267.         return new DataTypes(string);
268.     }
269. }
270.

```



```

271.         private boolean isInt(String strin) {
272.             try {
273.                 Integer.parseInt(strin);
274.                 return true;
275.             } catch (NumberFormatException fk) {
276.                 return false;
277.             }
278.         }
279.
280.         private boolean isdouble(String strin) {
281.             try {
282.                 Double.parseDouble(strin);
283.                 return true;
284.             } catch (NumberFormatException fk) {
285.                 return false;
286.             }
287.         }
288.
289.         private boolean isbool(String string) {
290.             try {
291.                 Boolean bo = Boolean.parseBoolean(string);
292.                 if (!bo && !string.equalsIgnoreCase("FALSE")) {
293.                     return false;
294.                 }
295.                 return true;
296.             } catch (NumberFormatException fk) {
297.                 return false;
298.             }
299.         }
300.
301.         // Compares the 2 input and pushes the boolean result in stack.
302.         private void compare(String comparison, Stack<DataTypes> local) throws Excep
tion {
303.
304.             DataTypes locop2 = local.pop();
305.             DataTypes locop1 = local.pop();
306.             DataTypes outcome;
307.             if (locop1.checkDataType().equals(locop2.checkDataType())) {
308.                 if (comparison.equals("==")) {
309.                     outcome = new DataTypes(locop2.toString().equals(locop1.toStrin
g()) ? true : false);
310.                     local.push(outcome);
311.                 } else if (comparison.equals("<") && locop1.checkDataType().equals("
int")) {
312.                     outcome = new DataTypes(locop2.Integer() > locop1.Integer() ? tr
ue : false);
313.                     local.push(outcome);
314.                 } else if (comparison.equals("<") && locop1.checkDataType().equals("
double")) {
315.                     outcome = new DataTypes(locop2.Double() > locop1.Double() ? true
: false);
316.                     local.push(outcome);
317.                 } else if (comparison.equals("<=") && locop1.checkDataType().equals(
"int")) {
318.                     outcome = new DataTypes(locop2.Integer() >= locop1.Integer() ? t
rue : false);
319.                     local.push(outcome);
320.                 } else if (comparison.equals("<=") && locop1.checkDataType().equals(
"double")) {
321.                     outcome = new DataTypes(locop2.Double() >= locop1.Double() ? tru
e : false);

```

```

322.         local.push(outcome);
323.     } else if (comparison.equals(">") && locop1.checkDataType().equals("
    int")) {
324.         outcome = new DataTypes(locop2.Integer() < locop1.Integer() ? tr
        ue : false);
325.         local.push(outcome);
326.     } else if (comparison.equals(">") && locop1.checkDataType().equals("
    double")) {
327.         outcome = new DataTypes(locop2.Double() < locop1.Double() ? true
            : false);
328.         local.push(outcome);
329.     } else if (comparison.equals(">=") && locop1.checkDataType().equals(
        "int")) {
330.         outcome = new DataTypes(locop2.Integer() <= locop1.Integer() ? t
            rue : false);
331.         local.push(outcome);
332.     } else if (comparison.equals(">=") && locop1.checkDataType().equals(
        "double")) {
333.         outcome = new DataTypes(locop2.Double() <= locop1.Double() ? tru
            e : false);
334.         local.push(outcome);
335.     } else {
336.         throw new Exception("Incorrect Datatype while comparison");
337.     }
338. } else {
339.     throw new Exception("Datatype mismatch while comparison");
340. }
341.
342. }
343.
344. // Performs Arithmetic operation on inputs and pushes the result in stack.
345. private void operator(String operation, Stack<DataTypes> local) throws Excep
    tion {
346.     DataTypes locop2 = local.pop();
347.     DataTypes locop1 = local.pop();
348.     DataTypes outcome;
349.     if (locop1.checkDataType().equals(locop2.checkDataType()) && locop1.chec
        kDataType().equals("int")) {
350.         if (operation.equals("ADD")) {
351.             outcome = new DataTypes(locop1.Integer() + locop2.Integer());
352.             local.push(outcome);
353.         } else if (operation.equals("SUBTRACT")) {
354.             outcome = new DataTypes(locop1.Integer() - locop2.Integer());
355.             local.push(outcome);
356.         } else if (operation.equals("MULTIPLY")) {
357.             outcome = new DataTypes(locop1.Integer() * locop2.Integer());
358.             local.push(outcome);
359.         } else if (operation.equals("DIVIDE")) {
360.             if (locop2.Integer() != 0) {
361.                 outcome = new DataTypes(locop1.Integer() / locop2.Integer())
                    ;
362.                 local.push(outcome);
363.             } else {
364.                 throw new Exception("Denominator can't be zero.");
365.             }
366.         }
367.     } else if (locop1.checkDataType().equals(locop2.checkDataType()) && loco
        p1.checkDataType().equals("double")) {
368.         if (operation.equals("ADD")) {
369.             outcome = new DataTypes(locop1.Double() + locop2.Double());
370.             local.push(outcome);

```

```

371.         } else if (operation.equals("SUBTRACT")) {
372.             outcome = new DataTypes(locop1.Double() - locop2.Double());
373.             local.push(outcome);
374.         } else if (operation.equals("MULTIPLY")) {
375.             outcome = new DataTypes(locop1.Double() * locop2.Double());
376.             local.push(outcome);
377.         } else if (operation.equals("DIVIDE")) {
378.             if (locop2.Integer() != 0) {
379.                 outcome = new DataTypes(locop1.Double() / locop2.Double());
380.                 local.push(outcome);
381.             } else {
382.                 throw new Exception("Denominator can't be zero.");
383.             }
384.         }
385.     } else {
386.         throw new Exception("Datatype of both the variables should be same."
387.     );
388.     }
389. }

```

Runtime Constant Class

```

1. package sparkyRuntime;
2.
3. public interface RuntimeConstantKeywords {
4.
5.     public static final String equal_to = "EQUAL_TO";
6.     public static final String INSTRUCTION_STORE = "STORE";
7.     public static final String INSTRUCTION_PUSH = "PUSH";
8.     public static final String DECLARE = "DECLARE";
9.     public static final String INT_DEFAULT = "0";
10.
11.     public static final String GET = "GET";
12.     public static final String OPERATOR = "OPERATOR";
13.     public static final String PUSH = "PUSH";
14.
15.     public static final String ADDITION = "ADD";
16.     public static final String SUBTRACTION = "SUBTRACT";
17.     public static final String MULTIPLICATION = "MULTIPLY";
18.     public static final String DIVISION = "DIVIDE";
19.     public static final String BOOLEAN = "YESNO";
20.     public static final String CONDITION_TRUE = "CONDITION_TRUE";
21.
22.     public static final String IF_START = "IF_START";
23.     public static final String IF_END = "IF_END";
24.     public static final String ELSE_START = "ELSE_START";
25.     public static final String ELSE_END = "ELSE_END";
26.     public static final String IFTE_START = "IFTE_START";
27.     public static final String IFTE_END = "IFTE_END";
28.     public static final String CHECK_CONDITION = "CHECK_CONDITION";
29.     public static final String CONDITION_FALSE = "CONDITION_FALSE";
30.     public static final String JUMP = "JUMP";
31.
32.     public static final String COMPARE_OPERATOR = "COMPARE_OPERATOR";
33.
34.     public static final String FOR_START = "FOR_START";

```

```

35.     public static final String FOR_STOP = "FOR_STOP";
36.     public static final String FOR_INIT = "FOR_INIT";
37.     public static final String FOR_EXPRESSION = "FOR_EXPRESSION";
38.     public static final String FOR_CONDITION_START = "FOR_CONDITION_START";
39.     public static final String FOR_CONDITION_STOP = "FOR_CONDITION_STOP";
40.     public static final String FOR_UPDATE_START = "FOR_UPDATE_START";
41.     public static final String FOR_UPDATE_STOP = "FOR_UPDATE_STOP";
42.     public static final String FOR_VARIABLE = "FOR_VARIABLE";
43.
44.     // Keywords for While
45.     public static final String WHILE_BEGIN = "WHILEBEGIN";
46.     public static final String WHILE_END = "WHILEEND";
47.
48.     /*
49.      * Existing keywords used for while GET , COMPARE_OPERATOR,CONDITION_NOT_TRUE,
50.      * STORE, OPERATOR etc.
51.      *
52.      */
53.     public static final String PRINT = "PRINT";
54.     public static final String AND_OR_OPERATOR = "AND_OR_OPERATOR";
55.
56. }

```

Sample Codes: (<https://github.com/MayankBatra005/SER502-Spring2020-Team25/tree/master/data>)

Example1: arithmeticOps.sparky

```

1. Live
2. int a=40;
3. int b=8;
4. int result;
5. result=a+b;
6. print(result);
7. result=a-b;
8. print(result);
9. result=a*b;
10. print(result);
11. result=a/b;
12. print(result);
13. Die

```

Corresponding intermediate code

```

1. DECLARE int a
2. STORE 40
3. PUSH a
4. DECLARE int b
5. STORE 8
6. PUSH b
7. DECLARE int result
8. GET a
9. GET b
10. OPERATOR ADD
11. PUSH result
12. GET result
13. PRINT

```

```
14. GET a
15. GET b
16. OPERATOR SUBTRACT
17. PUSH result
18. GET result
19. PRINT
20. GET a
21. GET b
22. OPERATOR MULTIPLY
23. PUSH result
24. GET result
25. PRINT
26. GET a
27. GET b
28. OPERATOR DIVIDE
29. PUSH result
30. GET result
31. PRINT
```

Output

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17763.1158]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\build>java -jar Compiler.jar arithmeticOps.sparky
48
32
320
5
C:\build>_
```

Example 2: fibonacci.sparky

```
1. Live
2. int count = 7;
3. int counter = 1;
4. int firstFib = 1;
5. int secondFib = 1;
6. int sum;
7. while counter <= count
8. {
9.   print(firstFib);
10.  sum = firstFib + secondFib;
11.  firstFib = secondFib;
12.  secondFib = sum;
13.  counter = counter + 1;
14. }
15. Die
```

Corresponding Intermediate code

```
1. DECLARE int count
2. STORE 7
3. PUSH count
4. DECLARE int counter
5. STORE 1
6. PUSH counter
7. DECLARE int firstFib
8. STORE 1
9. PUSH firstFib
10. DECLARE int secondFib
11. STORE 1
12. PUSH secondFib
13. DECLARE int sum
14. WHILEBEGIN
15. GET counter
16. GET count
17. COMPARE_OPERATOR <=
18. CONDITION_FALSE JUMP WHILEEND
19. GET firstFib
20. PRINT
21. GET firstFib
22. GET secondFib
23. OPERATOR ADD
24. PUSH sum
25. GET secondFib
26. PUSH firstFib
27. GET sum
28. PUSH secondFib
29. GET counter
30. STORE 1
31. OPERATOR ADD
32. PUSH counter
33. JUMP WHILEBEGIN
34. WHILEEND
```

Output

```
C:\build>java -jar Compiler.jar fibonacci.sparky
1
1
2
3
5
8
13
C:\build>
```

Example 3: factorial.sparky

```
1. Live
2. int result = 1;
3. int n=5;
```

```
4. for (int i=2; i<=n; i=i+1)
5. {result=result*i;}
6. print(result);
7. Die
```

Corresponding Intermediate Code

```
1. DECLARE int result
2. STORE 1
3. PUSH result
4. DECLARE int n
5. STORE 5
6. PUSH n
7. FOR_START
8. DECLARE int i
9. STORE 2
10. PUSH i
11. FOR_CONDITION_START
12. GET i
13. GET n
14. COMPARE_OPERATOR <=
15. CONDITION_FALSE JUMP FOR_STOP
16. GET result
17. GET i
18. OPERATOR MULTIPLY
19. PUSH result
20. GET i
21. STORE 1
22. OPERATOR ADD
23. PUSH i
24. JUMP FOR_CONDITION_START
25. FOR_STOP
26. GET result
27. PRINT
```

Output

```
C:\build>java -jar Compiler.jar factorial.sparky
120
C:\build>
```