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. Browser 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>Complier.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:

```
grammar Sparky;
program: LIVE ball DIE;
ball: expression* | declare* expression*;

declare:
(datatype STUFF EQUALTO NUMBER SEMICOLON)|
(datatype STUFF SEMICOLON)|
(HAINA STUFF EQUALTO booleanvalue SEMICOLON)|
(HAINA STUFF SEMICOLON)| stringdatatype STUFF EQUALTO STRINGLITERAL SEMICOLON | stringdatatype STUFF SEMICOLON;

expression
 : assignment
 | ifte
 | loopum
 |ternary_operator
 |print;

assignment
 : STUFF EQUALTO expr SEMICOLON |
 STUFF EQUALTO yesnostatement SEMICOLON
 ;

ifte
 : IF yesnostatement in_loop ('warna' in_loop)? FI
 ;


loopum : loop_for|loop_while | loop_for_range;
loop_for: 'for' LSmoothBrace for_declare? ';' for_expression? ';' for_expr? RSmoothBrace in_loop;
loop_while
 : WHILE yesnostatement in_loop
 ;

 loop_for_range: 'for'  STUFF 'in' 'range' LSmoothBrace NUMBER COMMA NUMBER RSmoothBrace in_loop;

in_loop: LCurlyBrace ball RCurlyBrace| expression;

for_expr: STUFF EQUALTO expr;
for_expression :expr YESNOOPERATOR expr;
for_declare:datatype STUFF EQUALTO NUMBER;

term: NUMBER | STUFF | STUFF op=(MUL | DIV) term | NUMBER  op=(MUL | DIV) term;
expr: term | term op=(PLUS | MINUS) expr | NOT expr;
yesnostatement : booleanvalue | expr YESNOOPERATOR expr |yesnostatement ANDOROPERATOR yesnostatement;
ANDOROPERATOR: AND|OR;

AND: 'and';
OR: 'or';
NOT:'not';


ternary_operator: yesnostatement '?' in_loop ':' in_loop;

print:'print' LSmoothBrace expr RSmoothBrace SEMICOLON;
```

```
LIVE: 'Live';
DIE: 'Die';
FI: 'fi';


YESNOOPERATOR: ASSEQ| LESS_THAN| MORE_THAN | LESS_THAN_EQ | MORE_THAN_EQ ;
EQUALTO : '=';
ASSEQ : '==';
LESS_THAN : '<';
MORE_THAN: '>';
LESS_THAN_EQ : '<=';
MORE_THAN_EQ : '>=';

warna :'else';

PLUS : '+';
MINUS :'-';
MUL : '*';
DIV : '/';
SEMICOLON : ';';
COMMA : ',';

LSmoothBrace : '(';
RSmoothBrace : ')';
LCurlyBrace : '{';
RCurlyBrace : '}';
DQ: '"';

STRINGLITERAL: DQ (~["\\\r\n])* DQ;
HAINA: 'haina';
haina:'bool';
datatype: INTEGER| DOUBLE | HAINA;
stringdatatype: STRING;
INTEGER: 'int';
STRING: 'string';
DOUBLE: 'double';
IF : 'if';
WHILE : 'while';
STUFF:[a-zA-Z_] [a-zA-Z_0-9]*;
NUMBER:[0-9]+;
WS: [ \t\r\n] -> skip;
booleanvalue: 'yup' | 'nup';
yup:'true';
nup:'false';
```

Model class to store intermediate Code

```
package Model;

/**
 * This purpose of this class is to be a model for writing the intermediate code keywor
ds
 * @author Mayank Batra
 * @since April-27-2020
 * @version 1.0
 */

public class IntermediateCodeWriter {

    private String icOutput = "";
    public static IntermediateCodeWriter singeltonInstance;

    private IntermediateCodeWriter() {}

    public void addOutput(String output) {
        this.icOutput += output + "\n";
    }



    public String getIcOutput() {
        return icOutput;
    }

    public void setIcOutput(String icOutput) {
        this.icOutput = icOutput;
    }

    public static IntermediateCodeWriter getInstance()
    {
        if (singeltonInstance==null)
        {
            singeltonInstance=new IntermediateCodeWriter ();
        }
        return singeltonInstance;
    }
}
```

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

```
// Generated from Sparky.g4 by ANTLR 4.8
package sparky;

import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.tree.ErrorNode;
import org.antlr.v4.runtime.tree.TerminalNode;

/**
 * This class provides an empty implementation of {@link SparkyListener},
 * which can be extended to create a listener which only needs to handle a subset
 * of the available methods.
 */
public class SparkyBaseListener implements SparkyListener {
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation does nothing.</p>
     */
    @Override public void enterProgram(SparkyParser.ProgramContext ctx) { }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation does nothing.</p>
     */
    @Override public void exitProgram(SparkyParser.ProgramContext ctx) { }
    /**
     * {@inheritDoc}
     *
```

```
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterBall(SparkyParser.BallContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitBall(SparkyParser.BallContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterDeclare(SparkyParser.DeclareContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitDeclare(SparkyParser.DeclareContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterExpression(SparkyParser.ExpressionContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitExpression(SparkyParser.ExpressionContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterAssignment(SparkyParser.AssignmentContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitAssignment(SparkyParser.AssignmentContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterIfte(SparkyParser.IfteContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitIfte(SparkyParser.IfteContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
```

```java
     */
    @Override public void enterLoopum(SparkyParser.LoopumContext ctx) { }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation does nothing.</p>
     */
    @Override public void exitLoopum(SparkyParser.LoopumContext ctx) { }
    /**
     * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void enterLoop_for(SparkyParser.Loop_forContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void exitLoop_for(SparkyParser.Loop_forContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void enterLoop_while(SparkyParser.Loop_whileContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void exitLoop_while(SparkyParser.Loop_whileContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void enterLoop_for_range(SparkyParser.Loop_for_rangeContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void exitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void enterIn_loop(SparkyParser.In_loopContext ctx) { }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation does nothing.</p>
            */
           @Override public void exitIn_loop(SparkyParser.In_loopContext ctx) { }
           /**
```

```java
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterFor_expr(SparkyParser.For_exprContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitFor_expr(SparkyParser.For_exprContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterFor_expression(SparkyParser.For_expressionContext
 ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitFor_expression(SparkyParser.For_expressionContext
ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterFor_declare(SparkyParser.For_declareContext ctx)
{ }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitFor_declare(SparkyParser.For_declareContext ctx) {
 }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterTerm(SparkyParser.TermContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitTerm(SparkyParser.TermContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterExpr(SparkyParser.ExprContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitExpr(SparkyParser.ExprContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterYesnostatement(SparkyParser.YesnostatementContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitYesnostatement(SparkyParser.YesnostatementContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterTernary_operator(SparkyParser.Ternary_operatorContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitTernary_operator(SparkyParser.Ternary_operatorContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterPrint(SparkyParser.PrintContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitPrint(SparkyParser.PrintContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterWarna(SparkyParser.WarnaContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitWarna(SparkyParser.WarnaContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterHaina(SparkyParser.HainaContext ctx) { }
	/**
```

```
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitHaina(SparkyParser.HainaContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterDatatype(SparkyParser.DatatypeContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitDatatype(SparkyParser.DatatypeContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterStringdatatype(SparkyParser.StringdatatypeContext
 ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitStringdatatype(SparkyParser.StringdatatypeContext
ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterBooleanvalue(SparkyParser.BooleanvalueContext ctx
) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitBooleanvalue(SparkyParser.BooleanvalueContext ctx)
 { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void enterYup(SparkyParser.YupContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
	 */
	@Override public void exitYup(SparkyParser.YupContext ctx) { }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation does nothing.</p>
```

```java
         */
        @Override public void enterNope(SparkyParser.NopeContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void exitNope(SparkyParser.NopeContext ctx) { }

        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void enterEveryRule(ParserRuleContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void exitEveryRule(ParserRuleContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void visitTerminal(TerminalNode node) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void visitErrorNode(ErrorNode node) { }
    }
```

```java
// Generated from Sparky.g4 by ANTLR 4.8
package sparky;

import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;

/**
 * This class provides an empty implementation of {@link SparkyVisitor},
 * which can be extended to create a visitor which only needs to handle a subset
 * of the available methods.
 *
 * @param <T> The return type of the visit operation. Use {@link Void} for
 * operations with no return type.
 */
public class SparkyBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements SparkyVisitor<T> {
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitProgram(SparkyParser.ProgramContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitBall(SparkyParser.BallContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitDeclare(SparkyParser.DeclareContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitExpression(SparkyParser.ExpressionContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitAssignment(SparkyParser.AssignmentContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitIfte(SparkyParser.IfteContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitLoopum(SparkyParser.LoopumContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitLoop_for(SparkyParser.Loop_forContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitLoop_while(SparkyParser.Loop_whileContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitIn_loop(SparkyParser.In_loopContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitFor_expr(SparkyParser.For_exprContext ctx) { return visitChildren(ctx); }
    /**
            * {@inheritDoc}
            *
            * <p>The default implementation returns the result of calling
            * {@link #visitChildren} on {@code ctx}.</p>
            */
           @Override public T visitFor_expression(SparkyParser.For_expressionContext ctx) { return visitChildren(ctx); }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation returns the result of calling
            * {@link #visitChildren} on {@code ctx}.</p>
            */
           @Override public T visitFor_declare(SparkyParser.For_declareContext ctx) { return visitChildren(ctx); }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation returns the result of calling
            * {@link #visitChildren} on {@code ctx}.</p>
            */
           @Override public T visitTerm(SparkyParser.TermContext ctx) { return visitChildren(ctx); }
           /**
            * {@inheritDoc}
            *
            * <p>The default implementation returns the result of calling
            * {@link #visitChildren} on {@code ctx}.</p>
            */
           @Override public T visitExpr(SparkyParser.ExprContext ctx) { return visitChildren(ctx); }
```

```java
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitYesnostatement(SparkyParser.YesnostatementContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitTernary_operator(SparkyParser.Ternary_operatorContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitPrint(SparkyParser.PrintContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitWarna(SparkyParser.WarnaContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitHaina(SparkyParser.HainaContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitDatatype(SparkyParser.DatatypeContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
	 */
	@Override public T visitStringdatatype(SparkyParser.StringdatatypeContext ctx) { return visitChildren(ctx); }
	/**
	 * {@inheritDoc}
	 *
	 * <p>The default implementation returns the result of calling
	 * {@link #visitChildren} on {@code ctx}.</p>
```

```java
             */
            @Override public T visitBooleanvalue(SparkyParser.BooleanvalueContext ctx) { return visitChildren(ctx); }
            /**
             * {@inheritDoc}
             *
             * <p>The default implementation returns the result of calling
             * {@link #visitChildren} on {@code ctx}.</p>
             */
            @Override public T visitYup(SparkyParser.YupContext ctx) { return visitChildren(ctx); }
            /**
             * {@inheritDoc}
             *
             * <p>The default implementation returns the result of calling
             * {@link #visitChildren} on {@code ctx}.</p>
             */
            @Override public T visitNope(SparkyParser.NopeContext ctx) { return visitChildren(ctx); }
        }
```

```java
// Generated from Sparky.g4 by ANTLR 4.8
package sparky;

import org.antlr.v4.runtime.Lexer;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.TokenStream;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.misc.*;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
public class SparkyLexer extends Lexer {
    static { RuntimeMetaData.checkVersion("4.8", RuntimeMetaData.VERSION); }

    protected static final DFA[] _decisionToDFA;
    protected static final PredictionContextCache _sharedContextCache =
        new PredictionContextCache();
    public static final int
        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,
        T__9=10, T__10=11, T__11=12, T__12=13, ANDOROPERATOR=14, AND=15, OR=16,
        NOT=17, LIVE=18, DIE=19, FI=20, YESNOOPERATOR=21, EQUALTO=22, ASSEQ=23,
        LESS_THAN=24, MORE_THAN=25, LESS_THAN_EQ=26, MORE_THAN_EQ=27, PLUS=28,
        MINUS=29, MUL=30, DIV=31, SEMICOLON=32, COMMA=33, LSmoothBrace=34, RSmoothBrace=35,
        LCurlyBrace=36, RCurlyBrace=37, DQ=38, STRINGLITERAL=39, HAINA=40, INTEGER=41,
        STRING=42, DOUBLE=43, DECIMAL=44, CHAR=45, IF=46, WHILE=47, STUFF=48,
        NUMBER=49, WS=50;
    public static String[] channelNames = {
        "DEFAULT_TOKEN_CHANNEL", "HIDDEN"
    };

    public static String[] modeNames = {
        "DEFAULT_MODE"
    };

    private static String[] makeRuleNames() {
        return new String[] {
            "T__0", "T__1", "T__2", "T__3", "T__4", "T__5", "T__6", "T__7", "T__8",
            "T__9", "T__10", "T__11", "T__12", "ANDOROPERATOR", "AND", "OR", "NOT",
            "LIVE", "DIE", "FI", "YESNOOPERATOR", "EQUALTO", "ASSEQ", "LESS_THAN",
            "MORE_THAN", "LESS_THAN_EQ", "MORE_THAN_EQ", "PLUS", "MINUS", "MUL",
            "DIV", "SEMICOLON", "COMMA", "LSmoothBrace", "RSmoothBrace", "LCurlyBrace",
            "RCurlyBrace", "DQ", "STRINGLITERAL", "HAINA", "INTEGER", "STRING", "DOUBLE",
            "DECIMAL", "CHAR", "IF", "WHILE", "STUFF", "NUMBER", "WS"
        };
    }
    public static final String[] ruleNames = makeRuleNames();

    private static String[] makeLiteralNames() {
        return new String[] {
            null, "'warna'", "'for'", "'in'", "'range'", "'?'", "':'", "'print'",
            "'else'", "'bool'", "'yup'", "'nope'", "'true'", "'false'", null, "'and'",
            "'or'", "'not'", "'Live'", "'Die'", "'fi'", null, "'='", "'=='", "'<'",
            "'>'", "'<='", "'>='", "'+'", "'-'", "'*'", "'/'", "';'", "','", "'('",
            "')'", "'{'", "'}'", "'\"'", null, "'haina'", "'int'", "'string'", "'double'",
            "'float'", "'char'", "'if'", "'while'"
        };
    }
    private static final String[] _LITERAL_NAMES = makeLiteralNames();
    private static String[] makeSymbolicNames() {
        return new String[] {
            null, null, null, null, null, null, null, null, null, null, null, null,
            null, null, "ANDOROPERATOR", "AND", "OR", "NOT", "LIVE", "DIE", "FI",
            "YESNOOPERATOR", "EQUALTO", "ASSEQ", "LESS_THAN", "MORE_THAN", "LESS_THAN_EQ",
            "MORE_THAN_EQ", "PLUS", "MINUS", "MUL", "DIV", "SEMICOLON", "COMMA",
            "LSmoothBrace", "RSmoothBrace", "LCurlyBrace", "RCurlyBrace", "DQ", "STRINGLITERAL",
            "HAINA", "INTEGER", "STRING", "DOUBLE", "DECIMAL", "CHAR", "IF", "WHILE",
            "STUFF", "NUMBER", "WS"
        };
    }
    private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
    public static final Vocabulary VOCABULARY = new VocabularyImpl(_LITERAL_NAMES, _SYMBOLIC_NAMES);

    /**
     * @deprecated Use {@link #VOCABULARY} instead.
     */
    @Deprecated
    public static final String[] tokenNames;
    static {
        tokenNames = new String[_SYMBOLIC_NAMES.length];
        for (int i = 0; i < tokenNames.length; i++) {
            tokenNames[i] = VOCABULARY.getLiteralName(i);
            if (tokenNames[i] == null) {
                tokenNames[i] = VOCABULARY.getSymbolicName(i);
            }

            if (tokenNames[i] == null) {
                tokenNames[i] = "<INVALID>";
            }
        }
    }

    @Override
    @Deprecated
    public String[] getTokenNames() {
        return tokenNames;
    }

        @Override

        public Vocabulary getVocabulary() {
            return VOCABULARY;
        }


        public SparkyLexer(CharStream input) {
            super(input);
            _interp = new LexerATNSimulator(this,_ATN,_decisionToDFA,_sharedContextCache);
        }

        @Override
        public String getGrammarFileName() { return "Sparky.g4"; }

        @Override
        public String[] getRuleNames() { return ruleNames; }

        @Override
        public String getSerializedATN() { return _serializedATN; }

        @Override
        public String[] getChannelNames() { return channelNames; }

        @Override
        public String[] getModeNames() { return modeNames; }

        @Override
        public ATN getATN() { return _ATN; }

        public static final String _serializedATN =
            "\3\u608b\ua72a\u8133\ub9ed\u417c\u3be7\u7786\u5964\2\64\u012e\b\1\4\2"+
            "\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"+
            "\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"+
            "\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"+
            "\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"+
            " \4!\t!\4\"\t\"\4#\t#\4$\t$\4%\t%\4&\t&\4\'\t\'\4(\t(\4)\t)\4*\t*\4+\t"+
            "+\4,\t,\4-\t-\4.\t.\4/\t/\4\60\t\60\4\61\t\61\4\62\t\62\4\63\t\63\3\2"+
            "\3\2\3\2\3\2\3\2\3\2\3\3\3\3\3\3\3\3\3\4\3\4\3\4\3\5\3\5\3\5\3\5\3\5\3"+
            "\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"+
            "\3\n\3\n\3\n\3\13\3\13\3\13\3\13\3\f\3\f\3\f\3\f\3\f\3\r\3\r\3\r\3\r\3" +
            "\r\3\16\3\16\3\16\3\16\3\16\3\16\3\17\3\17\5\17\u00a5\n\17\3\20\3\20\3" +
            "\20\3\20\3\21\3\21\3\21\3\22\3\22\3\22\3\22\3\23\3\23\3\23\3\23\3\23\3" +
            "\24\3\24\3\24\3\24\3\25\3\25\3\25\3\26\3\26\3\26\3\26\3\26\5\26\u00c3"+
            "\n\26\3\27\3\27\3\30\3\30\3\30\3\31\3\31\3\32\3\32\3\33\3\33\3\33\3\34" +
            "\3\34\3\34\3\35\3\35\3\36\3\36\3\37\3\37\3 \3 \3!\3!\3\"\3\"\3#\3#\3$"+
            "\3$\3%\3%\3&\3&\3\'\3\'\3(\3(\7(\u00ec\n(\f(\16(\u00ef\13(\3(\3(\3)\3"+
            ")\3)\3)\3)\3)\3*\3*\3*\3*\3+\3+\3+\3+\3+\3+\3,\3,\3,\3,\3,\3,\3,\3"+
            "-\3-\3-\3-\3-\3-\3.\3.\3.\3.\3.\3/\3/\3/\3\60\3\60\3\60\3\60\3\60\3\60"+
            "\3\61\3\61\7\61\u0121\n\61\f\61\16\61\u0124\13\61\3\62\6\62\u0127\n\62" +
            "\r\62\16\62\u0128\3\63\3\63\3\63\3\63\2\2\64\3\3\5\4\7\5\t\6\13\7\r\b"+
            "\17\t\21\n\23\13\25\f\27\r\31\16\33\17\35\20\37\21!\22#\23%\24\'\25)\26"+
            "+\27-\30/\31\61\32\63\33\65\34\67\359\36;\37= ?!A\"C#E$G%I&K\'M(O)Q*S"+
            "+U,W-Y.[/]\60_\61a\62c\63e\64\3\2\7\6\2\f\f\17\17$$^^\5\2C\\aac|\6\2\62"+
            ";C\\aac|\3\2\62;\5\2\13\f\17\17\"\"\2\u0135\2\3\3\2\2\2\2\5\3\2\2\2\2"+
            "\7\3\2\2\2\2\t\3\2\2\2\2\13\3\2\2\2\2\r\3\2\2\2\2\17\3\2\2\2\2\21\3\2"+
            "\2\2\2\23\3\2\2\2\2\25\3\2\2\2\2\27\3\2\2\2\2\31\3\2\2\2\2\33\3\2\2\2"+
            "\2\35\3\2\2\2\2\37\3\2\2\2\2!\3\2\2\2\2#\3\2\2\2\2%\3\2\2\2\2\'\3\2\2"+
            "\2\2)\3\2\2\2\2+\3\2\2\2\2-\3\2\2\2\2/\3\2\2\2\2\61\3\2\2\2\2\63\3\2\2"+
            "\2\2\65\3\2\2\2\2\67\3\2\2\2\29\3\2\2\2\2;\3\2\2\2\2=\3\2\2\2\2?\3\2\2" +
            "\2\2A\3\2\2\2\2C\3\2\2\2\2E\3\2\2\2\2G\3\2\2\2\2I\3\2\2\2\2K\3\2\2\2\2" +
            "M\3\2\2\2\2O\3\2\2\2\2Q\3\2\2\2\2S\3\2\2\2\2U\3\2\2\2\2W\3\2\2\2\2Y\3"+
            "\2\2\2\2[\3\2\2\2\2]\3\2\2\2\2_\3\2\2\2\2a\3\2\2\2\2c\3\2\2\2\2e\3\2\2" +
            "\2\3g\3\2\2\2\5m\3\2\2\2\7q\3\2\2\2\tt\3\2\2\2\13z\3\2\2\2\r|\3\2\2\2"+
            "\17~\3\2\2\2\21\u0084\3\2\2\2\23\u0089\3\2\2\2\25\u008e\3\2\2\2\27\u0092"+
            "\3\2\2\2\31\u0097\3\2\2\2\33\u009c\3\2\2\2\35\u00a4\3\2\2\2\37\u00a6\3" +
            "\2\2\2!\u00aa\3\2\2\2#\u00ad\3\2\2\2%\u00b1\3\2\2\2\'\u00b6\3\2\2\2)\u00ba"+
            "\3\2\2\2+\u00c2\3\2\2\2-\u00c4\3\2\2\2/\u00c6\3\2\2\2\61\u00c9\3\2\2\2"+
            "\63\u00cb\3\2\2\2\65\u00cd\3\2\2\2\67\u00d0\3\2\2\29\u00d3\3\2\2\2;\u00d5"+
            "\3\2\2\2=\u00d7\3\2\2\2?\u00d9\3\2\2\2A\u00db\3\2\2\2C\u00dd\3\2\2\2E"+
            "\u00df\3\2\2\2G\u00e1\3\2\2\2I\u00e3\3\2\2\2K\u00e5\3\2\2\2M\u00e7\3\2"+
            "\2\2O\u00e9\3\2\2\2Q\u00f2\3\2\2\2S\u00f8\3\2\2\2U\u00fc\3\2\2\2W\u0103"+
            "\3\2\2\2Y\u010a\3\2\2\2[\u0110\3\2\2\2]\u0115\3\2\2\2_\u0118\3\2\2\2a"+
            "\u011e\3\2\2\2c\u0126\3\2\2\2e\u012a\3\2\2\2gh\7y\2\2hi\7c\2\2ij\7t\2"+
            "\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"+
            "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"+
            "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\u0080\7t\2\2\u0080\u0081\7k\2\2\u0081\u0082\7p\2\2\u0082\u0083\7"+
            "v\2\2\u0083\20\3\2\2\2\u0084\u0085\7g\2\2\u0085\u0086\7n\2\2\u0086\u0087"+
            "\7u\2\2\u0087\u0088\7g\2\2\u0088\22\3\2\2\2\u0089\u008a\7d\2\2\u008a\u008b"+
            "\7q\2\2\u008b\u008c\7q\2\2\u008c\u008d\7n\2\2\u008d\24\3\2\2\2\u008e\u008f"+
            "\7{\2\2\u008f\u0090\7w\2\2\u0090\u0091\7r\2\2\u0091\26\3\2\2\2\u0092\u0093"+
            "\7p\2\2\u0093\u0094\7q\2\2\u0094\u0095\7r\2\2\u0095\u0096\7g\2\2\u0096"+
            "\30\3\2\2\2\u0097\u0098\7v\2\2\u0098\u0099\7t\2\2\u0099\u009a\7w\2\2\u009a"+
            "\u009b\7g\2\2\u009b\32\3\2\2\2\u009c\u009d\7h\2\2\u009d\u009e\7c\2\2\u009e"+
            "\u009f\7n\2\2\u009f\u00a0\7u\2\2\u00a0\u00a1\7g\2\2\u00a1\34\3\2\2\2\u00a2"+
            "\u00a5\5\37\20\2\u00a3\u00a5\5!\21\2\u00a4\u00a2\3\2\2\2\u00a4\u00a3\3"+
            "\2\2\2\u00a5\36\3\2\2\2\u00a6\u00a7\7c\2\2\u00a7\u00a8\7p\2\2\u00a8\u00a9"+
            "\7f\2\2\u00a9 \3\2\2\2\u00aa\u00ab\7q\2\2\u00ab\u00ac\7t\2\2\u00ac\"\3"+
            "\2\2\2\u00ad\u00ae\7p\2\2\u00ae\u00af\7q\2\2\u00af\u00b0\7v\2\2\u00b0"+
            "$\3\2\2\2\u00b1\u00b2\7N\2\2\u00b2\u00b3\7k\2\2\u00b3\u00b4\7x\2\2\u00b4"+
            "\u00b5\7g\2\2\u00b5&\3\2\2\2\u00b6\u00b7\7F\2\2\u00b7\u00b8\7k\2\2\u00b8"+
            "\u00b9\7g\2\2\u00b9(\3\2\2\2\u00ba\u00bb\7h\2\2\u00bb\u00bc\7k\2\2\u00bc"+
            "*\3\2\2\2\u00bd\u00c3\5/\30\2\u00be\u00c3\5\61\31\2\u00bf\u00c3\5\63\32"+
            "\2\u00c0\u00c3\5\65\33\2\u00c1\u00c3\5\67\34\2\u00c2\u00bd\3\2\2\2\u00c2"+
            "\u00be\3\2\2\2\u00c2\u00bf\3\2\2\2\u00c2\u00c0\3\2\2\2\u00c2\u00c1\3\2"+
            "\2\2\u00c3,\3\2\2\2\u00c4\u00c5\7?\2\2\u00c5.\3\2\2\2\u00c6\u00c7\7?\2"+
            "\2\u00c7\u00c8\7?\2\2\u00c8\60\3\2\2\2\u00c9\u00ca\7>\2\2\u00ca\62\3\2"+
            "\2\2\u00cb\u00cc\7@\2\2\u00cc\64\3\2\2\2\u00cd\u00ce\7>\2\2\u00ce\u00cf"+
            "\7?\2\2\u00cf\66\3\2\2\2\u00d0\u00d1\7@\2\2\u00d1\u00d2\7?\2\2\u00d28"+
```

```java
            "\3\2\2\2\u00d3\u00d4\7-\2\2\u00d4:\3\2\2\2\u00d5\u00d6\7/\2\2\u00d6<\3"+
            "\2\2\2\u00d7\u00d8\7,\2\2\u00d8>\3\2\2\2\u00d9\u00da\7\61\2\2\u00da@\3"+
            "\2\2\2\u00db\u00dc\7=\2\2\u00dcB\3\2\2\2\u00dd\u00de\7.\2\2\u00deD\3\2"+
            "\2\2\u00df\u00e0\7*\2\2\u00e0F\3\2\2\2\u00e1\u00e2\7+\2\2\u00e2H\3\2\2"+
            "\2\u00e3\u00e4\7}\2\2\u00e4J\3\2\2\2\u00e5\u00e6\7\177\2\2\u00e6L\3\2"+
            "\2\2\u00e7\u00e8\7$\2\2\u00e8N\3\2\2\2\u00e9\u00ed\5M\'\2\u00ea\u00ec"+
            "\n\2\2\2\u00eb\u00ea\3\2\2\2\u00ec\u00ef\3\2\2\2\u00ed\u00eb\3\2\2\2\u00ed"+
            "\u00ee\3\2\2\2\u00ee\u00f0\3\2\2\2\u00ef\u00ed\3\2\2\2\u00f0\u00f1\5M"+
            "\'\2\u00f1P\3\2\2\2\u00f2\u00f3\7j\2\2\u00f3\u00f4\7c\2\2\u00f4\u00f5"+
            "\7k\2\2\u00f5\u00f6\7p\2\2\u00f6\u00f7\7c\2\2\u00f7R\3\2\2\2\u00f8\u00f9"+
            "\7k\2\2\u00f9\u00fa\7p\2\2\u00fa\u00fb\7v\2\2\u00fbT\3\2\2\2\u00fc\u00fd"+
            "\7u\2\2\u00fd\u00fe\7v\2\2\u00fe\u00ff\7t\2\2\u00ff\u0100\7k\2\2\u0100"+
            "\u0101\7p\2\2\u0101\u0102\7i\2\2\u0102V\3\2\2\2\u0103\u0104\7f\2\2\u0104"+
            "\u0105\7q\2\2\u0105\u0106\7w\2\2\u0106\u0107\7d\2\2\u0107\u0108\7n\2\2"+
            "\u0108\u0109\7g\2\2\u0109X\3\2\2\2\u010a\u010b\7h\2\2\u010b\u010c\7n\2"+
            "\2\u010c\u010d\7q\2\2\u010d\u010e\7c\2\2\u010e\u010f\7v\2\2\u010fZ\3\2"+
            "\2\2\u0110\u0111\7e\2\2\u0111\u0112\7j\2\2\u0112\u0113\7c\2\2\u0113\u0114"+
            "\7t\2\2\u0114\\\3\2\2\2\u0115\u0116\7k\2\2\u0116\u0117\7h\2\2\u0117^\3"+
            "\2\2\2\u0118\u0119\7y\2\2\u0119\u011a\7j\2\2\u011a\u011b\7k\2\2\u011b"+
            "\u011c\7n\2\2\u011c\u011d\7g\2\2\u011d`\3\2\2\2\u011e\u0122\t\3\2\2\u011f"+
            "\u0121\t\4\2\2\u0120\u011f\3\2\2\2\u0121\u0124\3\2\2\2\u0122\u0120\3\2"+
            "\2\2\u0122\u0123\3\2\2\2\u0123b\3\2\2\2\u0124\u0122\3\2\2\2\u0125\u0127"+
            "\t\5\2\2\u0126\u0125\3\2\2\2\u0127\u0128\3\2\2\2\u0128\u0126\3\2\2\2\u0128"+
            "\u0129\3\2\2\2\u0129d\3\2\2\2\u012a\u012b\t\6\2\2\u012b\u012c\3\2\2\2"+
            "\u012c\u012d\b\63\2\2\u012df\3\2\2\2\b\2\u00a4\u00c2\u00ed\u0122\u0128"+
            "\3\b\2\2";
        public static final ATN _ATN =
            new ATNDeserializer().deserialize(_serializedATN.toCharArray());
        static {
            _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
            for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
                _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
            }
        }
    }
```

```java
// Generated from Sparky.g4 by ANTLR 4.8
package sparky;

import org.antlr.v4.runtime.tree.ParseTreeListener;

/**
 * This interface defines a complete listener for a parse tree produced by
 * {@link SparkyParser}.
 */
public interface SparkyListener extends ParseTreeListener {
    /**
     * Enter a parse tree produced by {@link SparkyParser#program}.
     * @param ctx the parse tree
     */
    void enterProgram(SparkyParser.ProgramContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#program}.
     * @param ctx the parse tree
     */
    void exitProgram(SparkyParser.ProgramContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#ball}.
     * @param ctx the parse tree
     */
    void enterBall(SparkyParser.BallContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#ball}.
     * @param ctx the parse tree
     */
    void exitBall(SparkyParser.BallContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#declare}.
     * @param ctx the parse tree
     */
    void enterDeclare(SparkyParser.DeclareContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#declare}.
     * @param ctx the parse tree
     */
    void exitDeclare(SparkyParser.DeclareContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#expression}.
     * @param ctx the parse tree
     */
    void enterExpression(SparkyParser.ExpressionContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#expression}.
     * @param ctx the parse tree
     */
    void exitExpression(SparkyParser.ExpressionContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#assignment}.
     * @param ctx the parse tree
     */
    void enterAssignment(SparkyParser.AssignmentContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#assignment}.
     * @param ctx the parse tree
     */
    void exitAssignment(SparkyParser.AssignmentContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#ifte}.
     * @param ctx the parse tree
     */
    void enterIfte(SparkyParser.IfteContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#ifte}.
     * @param ctx the parse tree
     */
    void exitIfte(SparkyParser.IfteContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#loopum}.
     * @param ctx the parse tree
     */
    void enterLoopum(SparkyParser.LoopumContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#loopum}.
     * @param ctx the parse tree
     */
    void exitLoopum(SparkyParser.LoopumContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#loop_for}.
     * @param ctx the parse tree
     */
    void enterLoop_for(SparkyParser.Loop_forContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#loop_for}.
     * @param ctx the parse tree
     */
    void exitLoop_for(SparkyParser.Loop_forContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#loop_while}.
     * @param ctx the parse tree
     */
    void enterLoop_while(SparkyParser.Loop_whileContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#loop_while}.
     * @param ctx the parse tree
     */
    void exitLoop_while(SparkyParser.Loop_whileContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#loop_for_range}.
     * @param ctx the parse tree
     */
    void enterLoop_for_range(SparkyParser.Loop_for_rangeContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#loop_for_range}.
     * @param ctx the parse tree
     */
    void exitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx);
    /**
     * Enter a parse tree produced by {@link SparkyParser#in_loop}.
     * @param ctx the parse tree
     */
    void enterIn_loop(SparkyParser.In_loopContext ctx);
    /**
     * Exit a parse tree produced by {@link SparkyParser#in_loop}.
     * @param ctx the parse tree
     */
```

```java
	void exitIn_loop(SparkyParser.In_loopContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#for_expr}.
	 * @param ctx the parse tree
	 */
	void enterFor_expr(SparkyParser.For_exprContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#for_expr}.
	 * @param ctx the parse tree
	 */
	void exitFor_expr(SparkyParser.For_exprContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#for_expression}.
	 * @param ctx the parse tree
	 */
	void enterFor_expression(SparkyParser.For_expressionContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#for_expression}.
	 * @param ctx the parse tree
	 */
	void exitFor_expression(SparkyParser.For_expressionContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#for_declare}.
	 * @param ctx the parse tree
	 */
	void enterFor_declare(SparkyParser.For_declareContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#for_declare}.
	 * @param ctx the parse tree
	 */
	void exitFor_declare(SparkyParser.For_declareContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#term}.
	 * @param ctx the parse tree
	 */
	void enterTerm(SparkyParser.TermContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#term}.
	 * @param ctx the parse tree
	 */
	void exitTerm(SparkyParser.TermContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#expr}.
	 * @param ctx the parse tree
	 */
	void enterExpr(SparkyParser.ExprContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#expr}.
	 * @param ctx the parse tree
	 */
	void exitExpr(SparkyParser.ExprContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#yesnostatement}.
	 * @param ctx the parse tree
	 */
	void enterYesnostatement(SparkyParser.YesnostatementContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#yesnostatement}.
	 * @param ctx the parse tree
	 */
	void exitYesnostatement(SparkyParser.YesnostatementContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#ternary_operator}.
	 * @param ctx the parse tree
	 */
	void enterTernary_operator(SparkyParser.Ternary_operatorContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#ternary_operator}.
	 * @param ctx the parse tree
	 */
	void exitTernary_operator(SparkyParser.Ternary_operatorContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#print}.
	 * @param ctx the parse tree
	 */
	void enterPrint(SparkyParser.PrintContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#print}.
	 * @param ctx the parse tree
	 */
	void exitPrint(SparkyParser.PrintContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#warna}.
	 * @param ctx the parse tree
	 */
	void enterWarna(SparkyParser.WarnaContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#warna}.
	 * @param ctx the parse tree
	 */
	void exitWarna(SparkyParser.WarnaContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#haina}.
	 * @param ctx the parse tree
	 */
	void enterHaina(SparkyParser.HainaContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#haina}.
	 * @param ctx the parse tree
	 */
	void exitHaina(SparkyParser.HainaContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#datatype}.
	 * @param ctx the parse tree
	 */
	void enterDatatype(SparkyParser.DatatypeContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#datatype}.
	 * @param ctx the parse tree
	 */
	void exitDatatype(SparkyParser.DatatypeContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#stringdatatype}.
	 * @param ctx the parse tree
	 */
	void enterStringdatatype(SparkyParser.StringdatatypeContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#stringdatatype}.
	 * @param ctx the parse tree
	 */
	void exitStringdatatype(SparkyParser.StringdatatypeContext ctx);
	/**
```

```java
	 * Enter a parse tree produced by {@link SparkyParser#booleanvalue}.
	 * @param ctx the parse tree
	 */
	void enterBooleanvalue(SparkyParser.BooleanvalueContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#booleanvalue}.
	 * @param ctx the parse tree
	 */
	void exitBooleanvalue(SparkyParser.BooleanvalueContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#yup}.
	 * @param ctx the parse tree
	 */
	void enterYup(SparkyParser.YupContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#yup}.
	 * @param ctx the parse tree
	 */
	void exitYup(SparkyParser.YupContext ctx);
	/**
	 * Enter a parse tree produced by {@link SparkyParser#nope}.
	 * @param ctx the parse tree
	 */
	void enterNope(SparkyParser.NopeContext ctx);
	/**
	 * Exit a parse tree produced by {@link SparkyParser#nope}.
	 * @param ctx the parse tree
	 */
	void exitNope(SparkyParser.NopeContext ctx);
}
```

```java
// Generated from Sparky.g4 by ANTLR 4.8
package sparky;

import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.misc.*;
import org.antlr.v4.runtime.tree.*;
import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
public class SparkyParser extends Parser {
    static { RuntimeMetaData.checkVersion("4.8", RuntimeMetaData.VERSION); }

    protected static final DFA[] _decisionToDFA;
    protected static final PredictionContextCache _sharedContextCache =
        new PredictionContextCache();
    public static final int
        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,
        T__9=10, T__10=11, T__11=12, T__12=13, ANDOROPERATOR=14, AND=15, OR=16,
        NOT=17, LIVE=18, DIE=19, FI=20, YESNOOPERATOR=21, EQUALTO=22, ASSEQ=23,
        LESS_THAN=24, MORE_THAN=25, LESS_THAN_EQ=26, MORE_THAN_EQ=27, PLUS=28,
        MINUS=29, MUL=30, DIV=31, SEMICOLON=32, COMMA=33, LSmoothBrace=34, RSmoothBrace=35,
        LCurlyBrace=36, RCurlyBrace=37, DQ=38, STRINGLITERAL=39, HAINA=40, INTEGER=41,
        STRING=42, DOUBLE=43, DECIMAL=44, CHAR=45, IF=46, WHILE=47, STUFF=48,
        NUMBER=49, WS=50;
    public static final int
        RULE_program = 0, RULE_ball = 1, RULE_declare = 2, RULE_expression = 3,
        RULE_assignment = 4, RULE_ifte = 5, RULE_loopum = 6, RULE_loop_for = 7,
        RULE_loop_while = 8, RULE_loop_for_range = 9, RULE_in_loop = 10, RULE_for_expr = 11,
        RULE_for_expression = 12, RULE_for_declare = 13, RULE_term = 14, RULE_expr = 15,
        RULE_yesnostatement = 16, RULE_ternary_operator = 17, RULE_print = 18,
        RULE_warna = 19, RULE_haina = 20, RULE_datatype = 21, RULE_stringdatatype = 22,
        RULE_booleanvalue = 23, RULE_yup = 24, RULE_nope = 25;
    private static String[] makeRuleNames() {
        return new String[] {
            "program", "ball", "declare", "expression", "assignment", "ifte", "loopum",
            "loop_for", "loop_while", "loop_for_range", "in_loop", "for_expr", "for_expression",
            "for_declare", "term", "expr", "yesnostatement", "ternary_operator",
            "print", "warna", "haina", "datatype", "stringdatatype", "booleanvalue",
            "yup", "nope"
        };
    }
    public static final String[] ruleNames = makeRuleNames();

    private static String[] makeLiteralNames() {
        return new String[] {
            null, "'warna'", "'for'", "'in'", "'range'", "'?'", "':'", "'print'",
            "'else'", "'bool'", "'yup'", "'nope'", "'true'", "'false'", null, "'and'",
            "'or'", "'not'", "'Live'", "'Die'", "'fi'", null, "'='", "'=='", "'<'",
            "'>'", "'<='", "'>='", "'+'", "'-'", "'*'", "'/'", "';'", "','", "'('",
            "')'", "'{'", "'}'", "'\"'", null, "'haina'", "'int'", "'string'", "'double'",
            "'float'", "'char'", "'if'", "'while'"
        };
    }
    private static final String[] _LITERAL_NAMES = makeLiteralNames();
    private static String[] makeSymbolicNames() {
        return new String[] {
            null, null, null, null, null, null, null, null, null, null, null, null,
            null, null, "ANDOROPERATOR", "AND", "OR", "NOT", "LIVE", "DIE", "FI",
            "YESNOOPERATOR", "EQUALTO", "ASSEQ", "LESS_THAN", "MORE_THAN", "LESS_THAN_EQ",
            "MORE_THAN_EQ", "PLUS", "MINUS", "MUL", "DIV", "SEMICOLON", "COMMA",
            "LSmoothBrace", "RSmoothBrace", "LCurlyBrace", "RCurlyBrace", "DQ", "STRINGLITERAL",
            "HAINA", "INTEGER", "STRING", "DOUBLE", "DECIMAL", "CHAR", "IF", "WHILE",
            "STUFF", "NUMBER", "WS"
        };
    }
    private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
    public static final Vocabulary VOCABULARY = new VocabularyImpl(_LITERAL_NAMES, _SYMBOLIC_NAMES);

    /**
     * @deprecated Use {@link #VOCABULARY} instead.
     */
    @Deprecated
    public static final String[] tokenNames;
    static {
        tokenNames = new String[_SYMBOLIC_NAMES.length];
        for (int i = 0; i < tokenNames.length; i++) {
            tokenNames[i] = VOCABULARY.getLiteralName(i);
            if (tokenNames[i] == null) {
                tokenNames[i] = VOCABULARY.getSymbolicName(i);
            }

            if (tokenNames[i] == null) {
                tokenNames[i] = "<INVALID>";
            }
        }
    }

    @Override
    @Deprecated
    public String[] getTokenNames() {
        return tokenNames;
    }

    @Override

            public Vocabulary getVocabulary() {
                return VOCABULARY;
            }

            @Override
            public String getGrammarFileName() { return "Sparky.g4"; }

            @Override
            public String[] getRuleNames() { return ruleNames; }

            @Override
            public String getSerializedATN() { return _serializedATN; }

            @Override
            public ATN getATN() { return _ATN; }

            public SparkyParser(TokenStream input) {
                super(input);
                _interp = new ParserATNSimulator(this,_ATN,_decisionToDFA,_sharedContext
Cache);
            }

            public static class ProgramContext extends ParserRuleContext {
                public TerminalNode LIVE() { return getToken(SparkyParser.LIVE, 0); }
                public BallContext ball() {
                    return getRuleContext(BallContext.class,0);
                }
                public TerminalNode DIE() { return getToken(SparkyParser.DIE, 0); }
                public ProgramContext(ParserRuleContext parent, int invokingState) {
                    super(parent, invokingState);
                }
                @Override public int getRuleIndex() { return RULE_program; }
                @Override
                public void enterRule(ParseTreeListener listener) {
```

```java
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterProgram(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitProgram(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitProgram(this);
                else return visitor.visitChildren(this);
            }
        }

        public final ProgramContext program() throws RecognitionException {
            ProgramContext _localctx = new ProgramContext(_ctx, getState());
            enterRule(_localctx, 0, RULE_program);
            try {
                enterOuterAlt(_localctx, 1);
                {
                setState(52);
                match(LIVE);
                setState(53);
                ball();
                setState(54);
                match(DIE);
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class BallContext extends ParserRuleContext {
            public List<ExpressionContext> expression() {
                return getRuleContexts(ExpressionContext.class);
            }
            public ExpressionContext expression(int i) {
                return getRuleContext(ExpressionContext.class,i);
            }
            public List<DeclareContext> declare() {
                return getRuleContexts(DeclareContext.class);
            }
            public DeclareContext declare(int i) {
                return getRuleContext(DeclareContext.class,i);
            }
            public BallContext(ParserRuleContext parent, int invokingState) {
                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_ball; }
            @Override
            public void enterRule(ParseTreeListener listener) {
```

```java
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterBall(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitBall(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitBall(this);
            else return visitor.visitChildren(this);
        }
    }

    public final BallContext ball() throws RecognitionException {
        BallContext _localctx = new BallContext(_ctx, getState());
        enterRule(_localctx, 2, RULE_ball);
        int _la;
        try {
            setState(74);
            _errHandler.sync(this);
            switch ( getInterpreter().adaptivePredict(_input,3,_ctx) ) {
            case 1:
                enterOuterAlt(_localctx, 1);
                {
                setState(59);
                _errHandler.sync(this);
                _la = _input.LA(1);
                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)) {
                    {
                    {
                    setState(56);
                    expression();
                    }
                    }
                    setState(61);
                    _errHandler.sync(this);
                    _la = _input.LA(1);
                }
                }
                break;
            case 2:
                enterOuterAlt(_localctx, 2);
                {
                setState(65);
                _errHandler.sync(this);
                _la = _input.LA(1);
                while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << HAINA) | (1L << INTEGER) | (1L << STRING) | (1L << DOUBLE) | (1L << DECIMAL) | (1L << CHAR))) != 0)) {
                    {
                    {
                    setState(62);
                    declare();
                    }
                    }
                    setState(67);
                    _errHandler.sync(this);
                    _la = _input.LA(1);
                }
                setState(71);
                _errHandler.sync(this);
                _la = _input.LA(1);
                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)) {
                    {
                    {
                    setState(68);
                    expression();
                    }
                    }
                    setState(73);
                    _errHandler.sync(this);
                    _la = _input.LA(1);
                }
                }
                break;
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class DeclareContext extends ParserRuleContext {
        public DatatypeContext datatype() {
            return getRuleContext(DatatypeContext.class,0);
        }
        public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
        public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0); }
        public TerminalNode NUMBER() { return getToken(SparkyParser.NUMBER, 0); }
        public TerminalNode SEMICOLON() { return getToken(SparkyParser.SEMICOLON, 0); }
        public TerminalNode HAINA() { return getToken(SparkyParser.HAINA, 0); }
        public BooleanvalueContext booleanvalue() {
            return getRuleContext(BooleanvalueContext.class,0);
        }
        public StringdatatypeContext stringdatatype() {
            return getRuleContext(StringdatatypeContext.class,0);
        }
        public TerminalNode STRINGLITERAL() { return getToken(SparkyParser.STRINGLITERAL, 0); }
        public DeclareContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_declare; }
        @Override
```

```java
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterDeclare(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitDeclare(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitDeclare(this);
                else return visitor.visitChildren(this);
            }
        }

        public final DeclareContext declare() throws RecognitionException {
            DeclareContext _localctx = new DeclareContext(_ctx, getState());
            enterRule(_localctx, 4, RULE_declare);
            try {
                setState(105);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,4,_ctx) ) {
                case 1:
                    enterOuterAlt(_localctx, 1);
                    {
                    {
                    setState(76);
                    datatype();
                    setState(77);
                    match(STUFF);
                    setState(78);
                    match(EQUALTO);
                    setState(79);
                    match(NUMBER);
                    setState(80);
                    match(SEMICOLON);
                    }
                    }
                    break;
                case 2:
                    enterOuterAlt(_localctx, 2);
                    {
                    {
                    setState(82);
                    datatype();
                    setState(83);
                    match(STUFF);
                    setState(84);
                    match(SEMICOLON);
                    }
                    }
                    break;
                case 3:
                    enterOuterAlt(_localctx, 3);
                    {
                    {
                    setState(86);
                    match(HAINA);
                    setState(87);
```

```
                        match(STUFF);
                        setState(88);
                        match(EQUALTO);
                        setState(89);
                        booleanvalue();
                        setState(90);
                        match(SEMICOLON);
                        }
                        }
                        break;
                    case 4:
                        enterOuterAlt(_localctx, 4);
                        {
                        {
                        setState(92);
                        match(HAINA);
                        setState(93);
                        match(STUFF);
                        setState(94);
                        match(SEMICOLON);
                        }
                        }
                        break;
                    case 5:
                        enterOuterAlt(_localctx, 5);
                        {
                        setState(95);
                        stringdatatype();
                        setState(96);
                        match(STUFF);
                        setState(97);
                        match(EQUALTO);
                        setState(98);
                        match(STRINGLITERAL);
                        setState(99);
                        match(SEMICOLON);
                        }
                        break;
                    case 6:
                        enterOuterAlt(_localctx, 6);
                        {
                        setState(101);
                        stringdatatype();
                        setState(102);
                        match(STUFF);
                        setState(103);
                        match(SEMICOLON);
                        }
                        break;
                    }
                }
                catch (RecognitionException re) {
                    _localctx.exception = re;
                    _errHandler.reportError(this, re);
                    _errHandler.recover(this, re);
                }
                finally {
                    exitRule();
                }
                return _localctx;
            }
```

```java

        public static class ExpressionContext extends ParserRuleContext {
            public AssignmentContext assignment() {
                return getRuleContext(AssignmentContext.class,0);
            }
            public IfteContext ifte() {
                return getRuleContext(IfteContext.class,0);
            }
            public LoopumContext loopum() {
                return getRuleContext(LoopumContext.class,0);
            }
            public Ternary_operatorContext ternary_operator() {
                return getRuleContext(Ternary_operatorContext.class,0);
            }
            public PrintContext print() {
                return getRuleContext(PrintContext.class,0);
            }
            public ExpressionContext(ParserRuleContext parent, int invokingState) {

                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_expression; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterExpression(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitExpression(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitExpression(this);
                else return visitor.visitChildren(this);
            }
        }

        public final ExpressionContext expression() throws RecognitionException {
            ExpressionContext _localctx = new ExpressionContext(_ctx, getState());
            enterRule(_localctx, 6, RULE_expression);
            try {
                setState(112);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,5,_ctx) ) {
                case 1:
                    enterOuterAlt(_localctx, 1);
                    {
                    setState(107);
                    assignment();
                    }
                    break;
                case 2:
                    enterOuterAlt(_localctx, 2);
                    {
                    setState(108);
                    ifte();
                    }
                    break;
                case 3:
                    enterOuterAlt(_localctx, 3);
                    {
                    setState(109);
                    loopum();
                    }
                    break;
                case 4:
                    enterOuterAlt(_localctx, 4);
                    {
                    setState(110);
                    ternary_operator();
                    }
                    break;
                case 5:
                    enterOuterAlt(_localctx, 5);
                    {
                    setState(111);
                    print();
                    }
                    break;
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class AssignmentContext extends ParserRuleContext {
            public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }

            public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0); }
            public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
            }
            public TerminalNode SEMICOLON() { return getToken(SparkyParser.SEMICOLON, 0); }
            public YesnostatementContext yesnostatement() {
                return getRuleContext(YesnostatementContext.class,0);
            }
            public AssignmentContext(ParserRuleContext parent, int invokingState) {

                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_assignment; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterAssignment(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitAssignment(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitAssignment(this);
                else return visitor.visitChildren(this);
            }
        }

        public final AssignmentContext assignment() throws RecognitionException {
            AssignmentContext _localctx = new AssignmentContext(_ctx, getState());
            enterRule(_localctx, 8, RULE_assignment);
            try {
                setState(124);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,6,_ctx) ) {
                case 1:
                    enterOuterAlt(_localctx, 1);
                    {
                    setState(114);
                    match(STUFF);
                    setState(115);
                    match(EQUALTO);
                    setState(116);
                    expr();
                    setState(117);
                    match(SEMICOLON);
                    }
                    break;
                case 2:
                    enterOuterAlt(_localctx, 2);
                    {
                    setState(119);
                    match(STUFF);
                    setState(120);
                    match(EQUALTO);
                    setState(121);
                    yesnostatement(0);
                    setState(122);
                    match(SEMICOLON);
                    }
                    break;
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class IfteContext extends ParserRuleContext {
            public TerminalNode IF() { return getToken(SparkyParser.IF, 0); }
            public YesnostatementContext yesnostatement() {
                return getRuleContext(YesnostatementContext.class,0);
            }
            public List<In_loopContext> in_loop() {
                return getRuleContexts(In_loopContext.class);
            }
            public In_loopContext in_loop(int i) {
                return getRuleContext(In_loopContext.class,i);
            }
            public TerminalNode FI() { return getToken(SparkyParser.FI, 0); }
            public IfteContext(ParserRuleContext parent, int invokingState) {
                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_ifte; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterIfte(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitIfte(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitIfte(this);
                else return visitor.visitChildren(this);
            }
        }

        public final IfteContext ifte() throws RecognitionException {
            IfteContext _localctx = new IfteContext(_ctx, getState());
            enterRule(_localctx, 10, RULE_ifte);
            int _la;
            try {
                enterOuterAlt(_localctx, 1);
                {
                setState(126);
                match(IF);
                setState(127);
                yesnostatement(0);
                setState(128);
                in_loop();
                setState(131);
                _errHandler.sync(this);
                _la = _input.LA(1);
                if (_la==T__0) {
                    {
                    setState(129);
                    match(T__0);
                    setState(130);
                    in_loop();
                    }
                }

                setState(133);
                match(FI);
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class LoopumContext extends ParserRuleContext {
            public Loop_forContext loop_for() {
                return getRuleContext(Loop_forContext.class,0);
            }
            public Loop_whileContext loop_while() {
                return getRuleContext(Loop_whileContext.class,0);
            }
            public Loop_for_rangeContext loop_for_range() {
                return getRuleContext(Loop_for_rangeContext.class,0);
            }
            public LoopumContext(ParserRuleContext parent, int invokingState) {
                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_loopum; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterLoopum(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitLoopum(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitLoopum(this);
                else return visitor.visitChildren(this);
            }
        }

        public final LoopumContext loopum() throws RecognitionException {
            LoopumContext _localctx = new LoopumContext(_ctx, getState());
            enterRule(_localctx, 12, RULE_loopum);
            try {
                setState(138);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,8,_ctx) ) {
                case 1:
                    enterOuterAlt(_localctx, 1);
                    {
                    setState(135);
                    loop_for();
                    }
                    break;
                case 2:
                    enterOuterAlt(_localctx, 2);
                    {
                    setState(136);
                    loop_while();
                    }
                    break;
                case 3:
```

```
                enterOuterAlt(_localctx, 3);
                {
                setState(137);
                loop_for_range();
                }
                break;
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class Loop_forContext extends ParserRuleContext {
        public TerminalNode LSmoothBrace() { return getToken(SparkyParser.LSmoothBrace, 0); }
        public List<TerminalNode> SEMICOLON() { return getTokens(SparkyParser.SEMICOLON); }
        public TerminalNode SEMICOLON(int i) {
            return getToken(SparkyParser.SEMICOLON, i);
        }
        public TerminalNode RSmoothBrace() { return getToken(SparkyParser.RSmoothBrace, 0); }
        public In_loopContext in_loop() {
            return getRuleContext(In_loopContext.class,0);
        }
        public For_declareContext for_declare() {
            return getRuleContext(For_declareContext.class,0);
        }
        public For_expressionContext for_expression() {
            return getRuleContext(For_expressionContext.class,0);
        }
        public For_exprContext for_expr() {
            return getRuleContext(For_exprContext.class,0);
        }
        public Loop_forContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_loop_for; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterLoop_for(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitLoop_for(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitLoop_for(this);
            else return visitor.visitChildren(this);
        }
```

```java
        }

        public final Loop_forContext loop_for() throws RecognitionException {
            Loop_forContext _localctx = new Loop_forContext(_ctx, getState());
            enterRule(_localctx, 14, RULE_loop_for);
            int _la;
            try {
                enterOuterAlt(_localctx, 1);
                {
                setState(140);
                match(T__1);
                setState(141);
                match(LSmoothBrace);
                setState(143);
                _errHandler.sync(this);
                _la = _input.LA(1);
                if ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << HAINA) | (1L << INTEGER) | (1L << DOUBLE) | (1L << DECIMAL) | (1L << CHAR))) != 0)) {
                    {
                    setState(142);
                    for_declare();
                    }
                }

                setState(145);
                match(SEMICOLON);
                setState(147);
                _errHandler.sync(this);
                _la = _input.LA(1);
                if ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << NOT) | (1L << STUFF) | (1L << NUMBER))) != 0)) {
                    {
                    setState(146);
                    for_expression();
                    }
                }

                setState(149);
                match(SEMICOLON);
                setState(151);
                _errHandler.sync(this);
                _la = _input.LA(1);
                if (_la==STUFF) {
                    {
                    setState(150);
                    for_expr();
                    }
                }

                setState(153);
                match(RSmoothBrace);
                setState(154);
                in_loop();
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class Loop_whileContext extends ParserRuleContext {
            public TerminalNode WHILE() { return getToken(SparkyParser.WHILE, 0); }

            public YesnostatementContext yesnostatement() {
                return getRuleContext(YesnostatementContext.class,0);
            }
            public In_loopContext in_loop() {
                return getRuleContext(In_loopContext.class,0);
            }
            public Loop_whileContext(ParserRuleContext parent, int invokingState) {

                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_loop_while; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.enterLoop_while(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener)
.exitLoop_while(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? ext
ends T>)visitor).visitLoop_while(this);
                else return visitor.visitChildren(this);
            }
        }

        public final Loop_whileContext loop_while() throws RecognitionException {
            Loop_whileContext _localctx = new Loop_whileContext(_ctx, getState());
            enterRule(_localctx, 16, RULE_loop_while);
            try {
                enterOuterAlt(_localctx, 1);
                {
                setState(156);
                match(WHILE);
                setState(157);
                yesnostatement(0);
                setState(158);
                in_loop();
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }
```

```java

    public static class Loop_for_rangeContext extends ParserRuleContext {
        public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
        public TerminalNode LSmoothBrace() { return getToken(SparkyParser.LSmoothBrace, 0); }
        public List<TerminalNode> NUMBER() { return getTokens(SparkyParser.NUMBER); }
        public TerminalNode NUMBER(int i) {
            return getToken(SparkyParser.NUMBER, i);
        }
        public TerminalNode COMMA() { return getToken(SparkyParser.COMMA, 0); }
        public TerminalNode RSmoothBrace() { return getToken(SparkyParser.RSmoothBrace, 0); }
        public In_loopContext in_loop() {
            return getRuleContext(In_loopContext.class,0);
        }
        public Loop_for_rangeContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_loop_for_range; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterLoop_for_range(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitLoop_for_range(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitLoop_for_range(this);
            else return visitor.visitChildren(this);
        }
    }

    public final Loop_for_rangeContext loop_for_range() throws RecognitionException {
        Loop_for_rangeContext _localctx = new Loop_for_rangeContext(_ctx, getState());
        enterRule(_localctx, 18, RULE_loop_for_range);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(160);
            match(T__1);
            setState(161);
            match(STUFF);
            setState(162);
            match(T__2);
            setState(163);
            match(T__3);
            setState(164);
            match(LSmoothBrace);
            setState(165);
            match(NUMBER);
            setState(166);
            match(COMMA);
            setState(167);
            match(NUMBER);
            setState(168);
            match(RSmoothBrace);
            setState(169);
            in_loop();
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class In_loopContext extends ParserRuleContext {
        public TerminalNode LCurlyBrace() { return getToken(SparkyParser.LCurlyBrace, 0); }
        public BallContext ball() {
            return getRuleContext(BallContext.class,0);
        }
        public TerminalNode RCurlyBrace() { return getToken(SparkyParser.RCurlyBrace, 0); }
        public ExpressionContext expression() {
            return getRuleContext(ExpressionContext.class,0);
        }
        public In_loopContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_in_loop; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterIn_loop(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitIn_loop(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitIn_loop(this);
            else return visitor.visitChildren(this);
        }
    }

    public final In_loopContext in_loop() throws RecognitionException {
        In_loopContext _localctx = new In_loopContext(_ctx, getState());
        enterRule(_localctx, 20, RULE_in_loop);
        try {
            setState(176);
            _errHandler.sync(this);
            switch (_input.LA(1)) {
```

```java
            case LCurlyBrace:
                enterOuterAlt(_localctx, 1);
                {
                setState(171);
                match(LCurlyBrace);
                setState(172);
                ball();
                setState(173);
                match(RCurlyBrace);
                }
                break;
            case T__1:
            case T__6:
            case T__9:
            case T__10:
            case NOT:
            case IF:
            case WHILE:
            case STUFF:
            case NUMBER:
                enterOuterAlt(_localctx, 2);
                {
                setState(175);
                expression();
                }
                break;
            default:
                throw new NoViableAltException(this);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class For_exprContext extends ParserRuleContext {
        public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
        public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0); }
        public ExprContext expr() {
            return getRuleContext(ExprContext.class,0);
        }
        public For_exprContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_for_expr; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterFor_expr(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitFor_expr(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitFor_expr(this);
            else return visitor.visitChildren(this);
        }
    }

    public final For_exprContext for_expr() throws RecognitionException {
        For_exprContext _localctx = new For_exprContext(_ctx, getState());
        enterRule(_localctx, 22, RULE_for_expr);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(178);
            match(STUFF);
            setState(179);
            match(EQUALTO);
            setState(180);
            expr();
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class For_expressionContext extends ParserRuleContext {
        public List<ExprContext> expr() {
            return getRuleContexts(ExprContext.class);
        }
        public ExprContext expr(int i) {
            return getRuleContext(ExprContext.class,i);
        }
        public TerminalNode YESNOOPERATOR() { return getToken(SparkyParser.YESNOOPERATOR, 0); }
        public For_expressionContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_for_expression; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterFor_expression(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitFor_expression(this);
        }
        @Override
```

```
			public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
				if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitFor_expression(this);
				else return visitor.visitChildren(this);
			}
		}

		public final For_expressionContext for_expression() throws RecognitionException {
			For_expressionContext _localctx = new For_expressionContext(_ctx, getState());
			enterRule(_localctx, 24, RULE_for_expression);
			try {
				enterOuterAlt(_localctx, 1);
				{
				setState(182);
				expr();
				setState(183);
				match(YESNOOPERATOR);
				setState(184);
				expr();
				}
			}
			catch (RecognitionException re) {
				_localctx.exception = re;
				_errHandler.reportError(this, re);
				_errHandler.recover(this, re);
			}
			finally {
				exitRule();
			}
			return _localctx;
		}

		public static class For_declareContext extends ParserRuleContext {
			public DatatypeContext datatype() {
				return getRuleContext(DatatypeContext.class,0);
			}
			public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
			public TerminalNode EQUALTO() { return getToken(SparkyParser.EQUALTO, 0); }
			public TerminalNode NUMBER() { return getToken(SparkyParser.NUMBER, 0); }
			public For_declareContext(ParserRuleContext parent, int invokingState) {
				super(parent, invokingState);
			}
			@Override public int getRuleIndex() { return RULE_for_declare; }
			@Override
			public void enterRule(ParseTreeListener listener) {
				if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterFor_declare(this);
			}
			@Override
			public void exitRule(ParseTreeListener listener) {
				if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitFor_declare(this);
			}
			@Override
			public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
```

```java
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitFor_declare(this);
                else return visitor.visitChildren(this);
            }
        }

        public final For_declareContext for_declare() throws RecognitionException {
            For_declareContext _localctx = new For_declareContext(_ctx, getState());
            enterRule(_localctx, 26, RULE_for_declare);
            try {
                enterOuterAlt(_localctx, 1);
                {
                setState(186);
                datatype();
                setState(187);
                match(STUFF);
                setState(188);
                match(EQUALTO);
                setState(189);
                match(NUMBER);
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class TermContext extends ParserRuleContext {
            public Token op;
            public TerminalNode NUMBER() { return getToken(SparkyParser.NUMBER, 0); }
            public TerminalNode STUFF() { return getToken(SparkyParser.STUFF, 0); }
            public TermContext term() {
                return getRuleContext(TermContext.class,0);
            }
            public TerminalNode MUL() { return getToken(SparkyParser.MUL, 0); }
            public TerminalNode DIV() { return getToken(SparkyParser.DIV, 0); }
            public TermContext(ParserRuleContext parent, int invokingState) {
                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_term; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterTerm(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitTerm(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitTerm(this);
                else return visitor.visitChildren(this);
            }
        }

        public final TermContext term() throws RecognitionException {
            TermContext _localctx = new TermContext(_ctx, getState());
            enterRule(_localctx, 28, RULE_term);
            int _la;
            try {
                setState(199);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,13,_ctx) ) {
                case 1:
                    enterOuterAlt(_localctx, 1);
                    {
                    setState(191);
                    match(NUMBER);
                    }
                    break;
                case 2:
                    enterOuterAlt(_localctx, 2);
                    {
                    setState(192);
                    match(STUFF);
                    }
                    break;
                case 3:
                    enterOuterAlt(_localctx, 3);
                    {
                    setState(193);
                    match(STUFF);
                    setState(194);
                    ((TermContext)_localctx).op = _input.LT(1);
                    _la = _input.LA(1);
                    if ( !(_la==MUL || _la==DIV) ) {
                        ((TermContext)_localctx).op = (Token)_errHandler.recoverInline(this);
                    }
                    else {
                        if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
                        _errHandler.reportMatch(this);
                        consume();
                    }
                    setState(195);
                    term();
                    }
                    break;
                case 4:
                    enterOuterAlt(_localctx, 4);
                    {
                    setState(196);
                    match(NUMBER);
                    setState(197);
                    ((TermContext)_localctx).op = _input.LT(1);
                    _la = _input.LA(1);
                    if ( !(_la==MUL || _la==DIV) ) {
                        ((TermContext)_localctx).op = (Token)_errHandler.recoverInline(this);
                    }
                    else {
                        if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
                        _errHandler.reportMatch(this);
                        consume();
                    }
                    setState(198);
                    term();
                    }
                    break;
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                exitRule();
            }
            return _localctx;
        }

        public static class ExprContext extends ParserRuleContext {
            public Token op;
            public TermContext term() {
                return getRuleContext(TermContext.class,0);
            }
            public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
            }
            public TerminalNode PLUS() { return getToken(SparkyParser.PLUS, 0); }
            public TerminalNode MINUS() { return getToken(SparkyParser.MINUS, 0); }
            public TerminalNode NOT() { return getToken(SparkyParser.NOT, 0); }
            public ExprContext(ParserRuleContext parent, int invokingState) {
                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_expr; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterExpr(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitExpr(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitExpr(this);
                else return visitor.visitChildren(this);
            }
        }

        public final ExprContext expr() throws RecognitionException {
            ExprContext _localctx = new ExprContext(_ctx, getState());
            enterRule(_localctx, 30, RULE_expr);
            int _la;
```

```
			try {
				setState(208);
				_errHandler.sync(this);
				switch ( getInterpreter().adaptivePredict(_input,14,_ctx) ) {
				case 1:
					enterOuterAlt(_localctx, 1);
					{
					setState(201);
					term();
					}
					break;
				case 2:
					enterOuterAlt(_localctx, 2);
					{
					setState(202);
					term();
					setState(203);
					((ExprContext)_localctx).op = _input.LT(1);
					_la = _input.LA(1);
					if ( !(_la==PLUS || _la==MINUS) ) {
						((ExprContext)_localctx).op = (Token)_errHandler.recoverInli
ne(this);
					}
					else {
						if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
						_errHandler.reportMatch(this);
						consume();
					}
					setState(204);
					expr();
					}
					break;
				case 3:
					enterOuterAlt(_localctx, 3);
					{
					setState(206);
					match(NOT);
					setState(207);
					expr();
					}
					break;
				}
			}
			catch (RecognitionException re) {
				_localctx.exception = re;
				_errHandler.reportError(this, re);
				_errHandler.recover(this, re);
			}
			finally {
				exitRule();
			}
			return _localctx;
		}

		public static class YesnostatementContext extends ParserRuleContext {
			public BooleanvalueContext booleanvalue() {
				return getRuleContext(BooleanvalueContext.class,0);
			}
			public List<ExprContext> expr() {
				return getRuleContexts(ExprContext.class);
			}
```

```java
            public ExprContext expr(int i) {
                return getRuleContext(ExprContext.class,i);
            }
            public TerminalNode YESNOOPERATOR() { return getToken(SparkyParser.YESNOOPERATOR, 0); }
            public List<YesnostatementContext> yesnostatement() {
                return getRuleContexts(YesnostatementContext.class);
            }
            public YesnostatementContext yesnostatement(int i) {
                return getRuleContext(YesnostatementContext.class,i);
            }
            public TerminalNode ANDOROPERATOR() { return getToken(SparkyParser.ANDOROPERATOR, 0); }
            public YesnostatementContext(ParserRuleContext parent, int invokingState) {
                super(parent, invokingState);
            }
            @Override public int getRuleIndex() { return RULE_yesnostatement; }
            @Override
            public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterYesnostatement(this);
            }
            @Override
            public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitYesnostatement(this);
            }
            @Override
            public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitYesnostatement(this);
                else return visitor.visitChildren(this);
            }
        }

        public final YesnostatementContext yesnostatement() throws RecognitionException {
            return yesnostatement(0);
        }

        private YesnostatementContext yesnostatement(int _p) throws RecognitionException {
            ParserRuleContext _parentctx = _ctx;
            int _parentState = getState();
            YesnostatementContext _localctx = new YesnostatementContext(_ctx, _parentState);
            YesnostatementContext _prevctx = _localctx;
            int _startState = 32;
            enterRecursionRule(_localctx, 32, RULE_yesnostatement, _p);
            try {
                int _alt;
                enterOuterAlt(_localctx, 1);
                {
                setState(216);
                _errHandler.sync(this);
                switch (_input.LA(1)) {
                case T__9:
                case T__10:
                    {
                    setState(211);
                    booleanvalue();
                    }
                    break;
                case NOT:
                case STUFF:
                case NUMBER:
                    {
                    setState(212);
                    expr();
                    setState(213);
                    match(YESNOOPERATOR);
                    setState(214);
                    expr();
                    }
                    break;
                default:
                    throw new NoViableAltException(this);
                }
                _ctx.stop = _input.LT(-1);
                setState(223);
                _errHandler.sync(this);
                _alt = getInterpreter().adaptivePredict(_input,16,_ctx);
                while ( _alt!=2 && _alt!=org.antlr.v4.runtime.atn.ATN.INVALID_ALT_NUMBER ) {
                    if ( _alt==1 ) {
                        if ( _parseListeners!=null ) triggerExitRuleEvent();
                        _prevctx = _localctx;
                        {
                        {
                        _localctx = new YesnostatementContext(_parentctx, _parentState);
                        pushNewRecursionContext(_localctx, _startState, RULE_yesnostatement);
                        setState(218);
                        if (!(precpred(_ctx, 1))) throw new FailedPredicateException(this, "precpred(_ctx, 1)");
                        setState(219);
                        match(ANDOROPERATOR);
                        setState(220);
                        yesnostatement(2);
                        }
                        }
                    }
                    setState(225);
                    _errHandler.sync(this);
                    _alt = getInterpreter().adaptivePredict(_input,16,_ctx);
                }
                }
            }
            catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
            }
            finally {
                unrollRecursionContexts(_parentctx);
            }
            return _localctx;
        }

        public static class Ternary_operatorContext extends ParserRuleContext {
```

```java
        public YesnostatementContext yesnostatement() {
            return getRuleContext(YesnostatementContext.class,0);
        }
        public List<In_loopContext> in_loop() {
            return getRuleContexts(In_loopContext.class);
        }
        public In_loopContext in_loop(int i) {
            return getRuleContext(In_loopContext.class,i);
        }
        public Ternary_operatorContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_ternary_operator; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterTernary_operator(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitTernary_operator(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitTernary_operator(this);
            else return visitor.visitChildren(this);
        }
    }

    public final Ternary_operatorContext ternary_operator() throws RecognitionException {
        Ternary_operatorContext _localctx = new Ternary_operatorContext(_ctx, getState());
        enterRule(_localctx, 34, RULE_ternary_operator);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(226);
            yesnostatement(0);
            setState(227);
            match(T__4);
            setState(228);
            in_loop();
            setState(229);
            match(T__5);
            setState(230);
            in_loop();
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class PrintContext extends ParserRuleContext {
        public TerminalNode LSmoothBrace() { return getToken(SparkyParser.LSmoothBrace, 0); }
        public ExprContext expr() {
            return getRuleContext(ExprContext.class,0);
        }
        public TerminalNode RSmoothBrace() { return getToken(SparkyParser.RSmoothBrace, 0); }
        public TerminalNode SEMICOLON() { return getToken(SparkyParser.SEMICOLON, 0); }
        public PrintContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_print; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterPrint(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitPrint(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitPrint(this);
            else return visitor.visitChildren(this);
        }
    }

    public final PrintContext print() throws RecognitionException {
        PrintContext _localctx = new PrintContext(_ctx, getState());
        enterRule(_localctx, 36, RULE_print);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(232);
            match(T__6);
            setState(233);
            match(LSmoothBrace);
            setState(234);
            expr();
            setState(235);
            match(RSmoothBrace);
            setState(236);
            match(SEMICOLON);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class WarnaContext extends ParserRuleContext {
        public WarnaContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_warna; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterWarna(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitWarna(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitWarna(this);
            else return visitor.visitChildren(this);
        }
    }

    public final WarnaContext warna() throws RecognitionException {
        WarnaContext _localctx = new WarnaContext(_ctx, getState());
        enterRule(_localctx, 38, RULE_warna);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(238);
            match(T__7);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class HainaContext extends ParserRuleContext {
        public HainaContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_haina; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterHaina(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitHaina(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitHaina(this);
            else return visitor.visitChildren(this);
        }
    }

    public final HainaContext haina() throws RecognitionException {
        HainaContext _localctx = new HainaContext(_ctx, getState());
        enterRule(_localctx, 40, RULE_haina);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(240);
            match(T__8);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class DatatypeContext extends ParserRuleContext {
        public TerminalNode INTEGER() { return getToken(SparkyParser.INTEGER, 0); }
        public TerminalNode DOUBLE() { return getToken(SparkyParser.DOUBLE, 0); }
        public TerminalNode DECIMAL() { return getToken(SparkyParser.DECIMAL, 0); }
        public TerminalNode CHAR() { return getToken(SparkyParser.CHAR, 0); }
        public TerminalNode HAINA() { return getToken(SparkyParser.HAINA, 0); }
        public DatatypeContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_datatype; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterDatatype(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitDatatype(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitDatatype(this);
            else return visitor.visitChildren(this);
        }
    }

```

```java
    public final DatatypeContext datatype() throws RecognitionException {
        DatatypeContext _localctx = new DatatypeContext(_ctx, getState());
        enterRule(_localctx, 42, RULE_datatype);
        int _la;
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(242);
            _la = _input.LA(1);
            if ( !((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L << HAINA) | (1L << INTEGER) | (1L << DOUBLE) | (1L << DECIMAL) | (1L << CHAR))) != 0)) ) {
            _errHandler.recoverInline(this);
            }
            else {
                if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
                _errHandler.reportMatch(this);
                consume();
            }
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class StringdatatypeContext extends ParserRuleContext {
        public TerminalNode STRING() { return getToken(SparkyParser.STRING, 0); }
        public StringdatatypeContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_stringdatatype; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterStringdatatype(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitStringdatatype(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitStringdatatype(this);
            else return visitor.visitChildren(this);
        }
    }

    public final StringdatatypeContext stringdatatype() throws RecognitionException {
        StringdatatypeContext _localctx = new StringdatatypeContext(_ctx, getState());
        enterRule(_localctx, 44, RULE_stringdatatype);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(244);
            match(STRING);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public static class BooleanvalueContext extends ParserRuleContext {
        public BooleanvalueContext(ParserRuleContext parent, int invokingState) {
            super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_booleanvalue; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterBooleanvalue(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitBooleanvalue(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitBooleanvalue(this);
            else return visitor.visitChildren(this);
        }
    }

    public final BooleanvalueContext booleanvalue() throws RecognitionException {
        BooleanvalueContext _localctx = new BooleanvalueContext(_ctx, getState());
        enterRule(_localctx, 46, RULE_booleanvalue);
        int _la;
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(246);
            _la = _input.LA(1);
            if ( !(_la==T__9 || _la==T__10) ) {
            _errHandler.recoverInline(this);
            }
            else {
                if ( _input.LA(1)==Token.EOF ) matchedEOF = true;
                _errHandler.reportMatch(this);
                consume();
```

```
                    }
                    }
                }
                catch (RecognitionException re) {
                    _localctx.exception = re;
                    _errHandler.reportError(this, re);
                    _errHandler.recover(this, re);
                }
                finally {
                    exitRule();
                }
                return _localctx;
            }

            public static class YupContext extends ParserRuleContext {
                public YupContext(ParserRuleContext parent, int invokingState) {
                    super(parent, invokingState);
                }
                @Override public int getRuleIndex() { return RULE_yup; }
                @Override
                public void enterRule(ParseTreeListener listener) {
                    if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterYup(this);
                }
                @Override
                public void exitRule(ParseTreeListener listener) {
                    if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitYup(this);
                }
                @Override
                public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
                    if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitYup(this);
                    else return visitor.visitChildren(this);
                }
            }

            public final YupContext yup() throws RecognitionException {
                YupContext _localctx = new YupContext(_ctx, getState());
                enterRule(_localctx, 48, RULE_yup);
                try {
                    enterOuterAlt(_localctx, 1);
                    {
                    setState(248);
                    match(T__11);
                    }
                }
                catch (RecognitionException re) {
                    _localctx.exception = re;
                    _errHandler.reportError(this, re);
                    _errHandler.recover(this, re);
                }
                finally {
                    exitRule();
                }
                return _localctx;
            }

            public static class NopeContext extends ParserRuleContext {
                public NopeContext(ParserRuleContext parent, int invokingState) {
                    super(parent, invokingState);
```

```java
        }
        @Override public int getRuleIndex() { return RULE_nope; }
        @Override
        public void enterRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).enterNope(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
            if ( listener instanceof SparkyListener ) ((SparkyListener)listener).exitNope(this);
        }
        @Override
        public <T> T accept(ParseTreeVisitor<? extends T> visitor) {
            if ( visitor instanceof SparkyVisitor ) return ((SparkyVisitor<? extends T>)visitor).visitNope(this);
            else return visitor.visitChildren(this);
        }
    }

    public final NopeContext nope() throws RecognitionException {
        NopeContext _localctx = new NopeContext(_ctx, getState());
        enterRule(_localctx, 50, RULE_nope);
        try {
            enterOuterAlt(_localctx, 1);
            {
            setState(250);
            match(T__12);
            }
        }
        catch (RecognitionException re) {
            _localctx.exception = re;
            _errHandler.reportError(this, re);
            _errHandler.recover(this, re);
        }
        finally {
            exitRule();
        }
        return _localctx;
    }

    public boolean sempred(RuleContext _localctx, int ruleIndex, int predIndex) {
        switch (ruleIndex) {
        case 16:
            return yesnostatement_sempred((YesnostatementContext)_localctx, predIndex);
        }
        return true;
    }
    private boolean yesnostatement_sempred(YesnostatementContext _localctx, int predIndex) {
        switch (predIndex) {
        case 0:
            return precpred(_ctx, 1);
        }
        return true;
    }

    public static final String _serializedATN =
        "\3\u608b\ua72a\u8133\ub9ed\u417c\u3be7\u7786\u5964\3\64\u00ff\4\2\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\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\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\t\31"+
        "\4\32\t\32\4\33\t\33\3\2\3\2\3\2\3\2\3\3\7\3<\n\3\f\3\16\3?\13\3\3\3\7"+
        "\3B\n\3\f\3\16\3E\13\3\3\3\7\3H\n\3\f\3\16\3K\13\3\5\3M\n\3\3\4\3\4\3"+
        "\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4"+
        "\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\3\4\5\4l\n\4\3\5\3\5\3\5\3\5\3\5\5\5"+
        "s\n\5\3\6\3\6\3\6\3\6\3\6\3\6\3\6\3\6\3\6\3\6\5\6\177\n\6\3\7\3\7\3\7"+
        "\3\7\3\7\5\7\u0086\n\7\3\7\3\7\3\b\3\b\3\b\5\b\u008d\n\b\3\t\3\t\3\t\5"+
        "\t\u0092\n\t\3\t\3\t\5\t\u0096\n\t\3\t\3\t\5\t\u009a\n\t\3\t\3\t\3\t\3"+
        "\n\3\n\3\n\3\n\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13\3\13"+
        "\3\f\3\f\3\f\3\f\3\f\5\f\u00b3\n\f\3\r\3\r\3\r\3\r\3\16\3\16\3\16\3\16"+
        "\3\17\3\17\3\17\3\17\3\17\3\20\3\20\3\20\3\20\3\20\3\20\3\20\3\20\5\20"+
        "\u00ca\n\20\3\21\3\21\3\21\3\21\3\21\3\21\3\21\5\21\u00d3\n\21\3\22\3"+
        "\22\3\22\3\22\3\22\3\22\5\22\u00db\n\22\3\22\3\22\3\22\3\22\7\22\u00e0\n\22"+
        "\f\22\16\22\u00e3\13\22\3\23\3\23\3\23\3\23\3\23\3\23\3\24\3\24\3\24\3"+
        "\24\3\24\3\24\3\25\3\25\3\26\3\26\3\27\3\27\3\30\3\30\3\31\3\31\3\32\3"+
        "\32\3\33\3\33\3\33\2\3\"\34\2\4\6\b\n\f\16\20\22\24\26\30\32\34\36 \""+
        "$&(*,.\60\62\64\2\6\3\2 !\3\2\36\37\4\2*+-/\3\2\f\r\2\u0100\2\66\3\2\2"+
        "\2\4L\3\2\2\2\6k\3\2\2\2\br\3\2\2\2\n~\3\2\2\2\f\u0080\3\2\2\2\16\u008c"+
        "\3\2\2\2\20\u008e\3\2\2\2\22\u009e\3\2\2\2\24\u00a2\3\2\2\2\26\u00b2\3"+
        "\2\2\2\30\u00b4\3\2\2\2\32\u00b8\3\2\2\2\34\u00bc\3\2\2\2\36\u00c9\3\2"+
        "\2\2 \u00d2\3\2\2\2\"\u00da\3\2\2\2$\u00e4\3\2\2\2&\u00ea\3\2\2\2(\u00f0"+
        "\3\2\2\2*\u00f2\3\2\2\2,\u00f4\3\2\2\2.\u00f6\3\2\2\2\60\u00f8\3\2\2\2"+
        "\62\u00fa\3\2\2\2\64\u00fc\3\2\2\2\66\67\7\24\2\2\678\5\4\3\289\7\25\2"+
        "\29\3\3\2\2\2:<\5\b\5\2;:\3\2\2\2<?\3\2\2\2=;\3\2\2\2=>\3\2\2\2>M\3\2"+
        "\2\2?=\3\2\2\2@B\5\6\4\2A@\3\2\2\2BE\3\2\2\2CA\3\2\2\2CD\3\2\2\2DI\3\2"+
        "\2\2EC\3\2\2\2FH\5\b\5\2GF\3\2\2\2HK\3\2\2\2IG\3\2\2\2IJ\3\2\2\2JM\3\2"+
        "\2\2KI\3\2\2\2L=\3\2\2\2LC\3\2\2\2M\5\3\2\2\2NO\5,\27\2OP\7\62\2\2PQ\7"+
        "\30\2\2QR\7\63\2\2RS\7\"\2\2Sl\3\2\2\2TU\5,\27\2UV\7\62\2\2VW\7\"\2\2"+
        "Wl\3\2\2\2XY\7*\2\2YZ\7\62\2\2Z[\7\30\2\2[\\\5\60\31\2\\]\7\"\2\2]l\3"+
        "\2\2\2^_\7*\2\2_`\7\62\2\2`l\7\"\2\2ab\5.\30\2bc\7\62\2\2cd\7\30\2\2d"+
        "e\7)\2\2ef\7\"\2\2fl\3\2\2\2gh\5.\30\2hi\7\62\2\2ij\7\"\2\2jl\3\2\2\2"+
        "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"+
        "\2ms\5\n\6\2ns\5\f\7\2os\5\16\b\2ps\5$\23\2qs\5&\24\2rm\3\2\2\2rn\3\2"+
        "\2\2ro\3\2\2\2rp\3\2\2\2rq\3\2\2\2s\t\3\2\2\2tu\7\62\2\2uv\7\30\2\2vw"+
        "\5 \21\2wx\7\"\2\2x\177\3\2\2\2yz\7\62\2\2z{\7\30\2\2{|\5\"\22\2|}\7\""
        +
        "\2\2}\177\3\2\2\2~t\3\2\2\2~y\3\2\2\2\177\13\3\2\2\2\u0080\u0081\7\60"+
        "\2\2\u0081\u0082\5\"\22\2\u0082\u0085\5\26\f\2\u0083\u0084\7\3\2\2\u0084"+
        "\u0086\5\26\f\2\u0085\u0083\3\2\2\2\u0085\u0086\3\2\2\2\u0086\u0087\3"+
        "\2\2\2\u0087\u0088\7\26\2\2\u0088\r\3\2\2\2\u0089\u008d\5\20\t\2\u008a"
        +
        "\u008d\5\22\n\2\u008b\u008d\5\24\13\2\u008c\u0089\3\2\2\2\u008c\u008a"+
        "\3\2\2\2\u008c\u008b\3\2\2\2\u008d\17\3\2\2\2\u008e\u008f\7\4\2\2\u008f"+
        "\u0091\7$\2\2\u0090\u0092\5\34\17\2\u0091\u0090\3\2\2\2\u0091\u0092\3"+
        "\2\2\2\u0092\u0093\3\2\2\2\u0093\u0095\7\"\2\2\u0094\u0096\5\32\16\2\u0095"+
        "\u0094\3\2\2\2\u0095\u0096\3\2\2\2\u0096\u0097\3\2\2\2\u0097\u0099\7\""
        +
        "\2\2\u0098\u009a\5\30\r\2\u0099\u0098\3\2\2\2\u0099\u009a\3\2\2\2\u009a"+
        "\u009b\3\2\2\2\u009b\u009c\7%\2\2\u009c\u009d\5\26\f\2\u009d\21\3\2\2"+
        "\2\u009e\u009f\7\61\2\2\u009f\u00a0\5\"\22\2\u00a0\u00a1\5\26\f\2\u00a1"+
        "\23\3\2\2\2\u00a2\u00a3\7\4\2\2\u00a3\u00a4\7\62\2\2\u00a4\u00a5\7\5\2"
        +
        "\2\u00a5\u00a6\7\6\2\2\u00a6\u00a7\7$\2\2\u00a7\u00a8\7\63\2\2\u00a8\u00a9"+
        "\7#\2\2\u00a9\u00aa\7\63\2\2\u00aa\u00ab\7%\2\2\u00ab\u00ac\5\26\f\2\u00ac"+
        "\25\3\2\2\2\u00ad\u00ae\7&\2\2\u00ae\u00af\5\4\3\2\u00af\u00b0\7\'\2\2"
        +
        "\u00b0\u00b3\3\2\2\2\u00b1\u00b3\5\b\5\2\u00b2\u00ad\3\2\2\2\u00b2\u00b1"+
        "\3\2\2\2\u00b3\27\3\2\2\2\u00b4\u00b5\7\62\2\2\u00b5\u00b6\7\30\2\2\u00b6"+
        "\u00b7\5 \21\2\u00b7\31\3\2\2\2\u00b8\u00b9\5 \21\2\u00b9\u00ba\7\27\2"
        +
        "\2\u00ba\u00bb\5 \21\2\u00bb\33\3\2\2\2\u00bc\u00bd\5,\27\2\u00bd\u00be"+
        "\7\62\2\2\u00be\u00bf\7\30\2\2\u00bf\u00c0\7\63\2\2\u00c0\35\3\2\2\2\u00c1"+
        "\u00ca\7\63\2\2\u00c2\u00ca\7\62\2\2\u00c3\u00c4\7\62\2\2\u00c4\u00c5"+
```

```java
                "\t\2\2\2\u00c5\u00ca\5\36\20\2\u00c6\u00c7\7\63\2\2\u00c7\u00c8\t\2\2"+
                "\2\u00c8\u00ca\5\36\20\2\u00c9\u00c1\3\2\2\2\u00c9\u00c2\3\2\2\2\u00c9"+
                "\u00c3\3\2\2\2\u00c9\u00c6\3\2\2\2\u00ca\37\3\2\2\2\u00cb\u00d3\5\36\20"+
                "\2\u00cc\u00cd\5\36\20\2\u00cd\u00ce\t\3\2\2\u00ce\u00cf\5 \21\2\u00cf"+
                "\u00d3\3\2\2\2\u00d0\u00d1\7\23\2\2\u00d1\u00d3\5 \21\2\u00d2\u00cb\3"+
                "\2\2\2\u00d2\u00cc\3\2\2\2\u00d2\u00d0\3\2\2\2\u00d3!\3\2\2\2\u00d4\u00d5"+
                "\b\22\1\2\u00d5\u00db\5\60\31\2\u00d6\u00d7\5 \21\2\u00d7\u00d8\7\27\2"+
                "\2\u00d8\u00d9\5 \21\2\u00d9\u00db\3\2\2\2\u00da\u00d4\3\2\2\2\u00da\u00d6"+
                "\3\2\2\2\u00db\u00e1\3\2\2\2\u00dc\u00dd\f\3\2\2\u00dd\u00de\7\20\2\2"+
                "\u00de\u00e0\5\"\22\4\u00df\u00dc\3\2\2\2\u00e0\u00e3\3\2\2\2\u00e1\u00df"+
                "\3\2\2\2\u00e1\u00e2\3\2\2\2\u00e2#\3\2\2\2\u00e3\u00e1\3\2\2\2\u00e4"+
                "\u00e5\5\"\22\2\u00e5\u00e6\7\7\2\2\u00e6\u00e7\5\26\f\2\u00e7\u00e8\7"+
                "\b\2\2\u00e8\u00e9\5\26\f\2\u00e9%\3\2\2\2\u00ea\u00eb\7\t\2\2\u00eb\u00ec"+
                "\7$\2\2\u00ec\u00ed\5 \21\2\u00ed\u00ee\7%\2\2\u00ee\u00ef\7\"\2\2\u00ef"+
                "\'\3\2\2\2\u00f0\u00f1\7\n\2\2\u00f1)\3\2\2\2\u00f2\u00f3\7\13\2\2\u00f3"+
                "+\3\2\2\2\u00f4\u00f5\t\4\2\2\u00f5-\3\2\2\2\u00f6\u00f7\7,\2\2\u00f7"+
                "/\3\2\2\2\u00f8\u00f9\t\5\2\2\u00f9\61\3\2\2\2\u00fa\u00fb\7\16\2\2\u00fb"+
                "\63\3\2\2\2\u00fc\u00fd\7\17\2\2\u00fd\65\3\2\2\2\23=CILkr~\u0085\u008c"+
                "\u0091\u0095\u0099\u00b2\u00c9\u00d2\u00da\u00e1";
            public static final ATN _ATN =
                new ATNDeserializer().deserialize(_serializedATN.toCharArray());
            static {
                _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
                for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
                    _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
                }
            }
        }
```

```java
// Generated from Sparky.g4 by ANTLR 4.8
package sparky;

import org.antlr.v4.runtime.tree.ParseTreeVisitor;

/**
 * This interface defines a complete generic visitor for a parse tree produced
 * by {@link SparkyParser}.
 *
 * @param <T> The return type of the visit operation. Use {@link Void} for
 * operations with no return type.
 */
public interface SparkyVisitor<T> extends ParseTreeVisitor<T> {
    /**
     * Visit a parse tree produced by {@link SparkyParser#program}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitProgram(SparkyParser.ProgramContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#ball}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitBall(SparkyParser.BallContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#declare}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitDeclare(SparkyParser.DeclareContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#expression}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitExpression(SparkyParser.ExpressionContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#assignment}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitAssignment(SparkyParser.AssignmentContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#ifte}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitIfte(SparkyParser.IfteContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#loopum}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitLoopum(SparkyParser.LoopumContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#loop_for}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitLoop_for(SparkyParser.Loop_forContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#loop_while}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitLoop_while(SparkyParser.Loop_whileContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#loop_for_range}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#in_loop}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitIn_loop(SparkyParser.In_loopContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#for_expr}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitFor_expr(SparkyParser.For_exprContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#for_expression}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitFor_expression(SparkyParser.For_expressionContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#for_declare}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitFor_declare(SparkyParser.For_declareContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#term}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitTerm(SparkyParser.TermContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#expr}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitExpr(SparkyParser.ExprContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#yesnostatement}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitYesnostatement(SparkyParser.YesnostatementContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#ternary_operator}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitTernary_operator(SparkyParser.Ternary_operatorContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#print}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitPrint(SparkyParser.PrintContext ctx);
    /**
     * Visit a parse tree produced by {@link SparkyParser#warna}.
     * @param ctx the parse tree
     * @return the visitor result
     */
    T visitWarna(SparkyParser.WarnaContext ctx);
```

```java
        /**
         * Visit a parse tree produced by {@link SparkyParser#haina}.
         * @param ctx the parse tree
         * @return the visitor result
         */
        T visitHaina(SparkyParser.HainaContext ctx);
        /**
         * Visit a parse tree produced by {@link SparkyParser#datatype}.
         * @param ctx the parse tree
         * @return the visitor result
         */
        T visitDatatype(SparkyParser.DatatypeContext ctx);
        /**
         * Visit a parse tree produced by {@link SparkyParser#stringdatatype}.
         * @param ctx the parse tree
         * @return the visitor result
         */
        T visitStringdatatype(SparkyParser.StringdatatypeContext ctx);
        /**
         * Visit a parse tree produced by {@link SparkyParser#booleanvalue}.
         * @param ctx the parse tree
         * @return the visitor result
         */
        T visitBooleanvalue(SparkyParser.BooleanvalueContext ctx);
        /**
         * Visit a parse tree produced by {@link SparkyParser#yup}.
         * @param ctx the parse tree
         * @return the visitor result
         */
        T visitYup(SparkyParser.YupContext ctx);
        /**
         * Visit a parse tree produced by {@link SparkyParser#nope}.
         * @param ctx the parse tree
         * @return the visitor result
         */
        T visitNope(SparkyParser.NopeContext ctx);
    }
```

Sparky Compiler Code

```java
package sparkyCompiler;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;

import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;

import sparky.SparkyLexer;
import sparky.SparkyParser;
import sparkyRuntime.IntermediateCodeReader;

```

```java
public class Compiler {

    public static void main(String[] args) throws Exception {
        String inputIcfile = null;

        try {

            if (args.length > 0) {
                String inputFileName = args[0];
                inputIcfile = args[0];

                CharStream sourceCode = CharStreams.fromFileName(inputFileName);
                SparkyLexer lx = new SparkyLexer(sourceCode);
                CommonTokenStream tokenStream = new CommonTokenStream(lx);
                SparkyParser parser = new SparkyParser(tokenStream);
                ParseTree parseTree = parser.program();

                IntermediateCodeGenerator iCodeGen = new IntermediateCodeGenerator();
                iCodeGen.visit(parseTree);

                List<String> iCodeArray = Arrays.asList(iCodeGen.getOutput().split("\\n"));

                try {
                    File iCodeFile = new File(inputFileName.replace("sparky", "sparkyic"));
                    BufferedWriter bufferWriter = null;
                    FileWriter fileWriter = null;
                    if (iCodeArray.size() > 1) {
                        try {
                            if (iCodeFile.exists()) {
                                iCodeFile.delete();
                                iCodeFile.createNewFile();
                            } else {
                                iCodeFile.createNewFile();
                            }

                            fileWriter = new FileWriter(iCodeFile);
                            bufferWriter = new BufferedWriter(fileWriter);
                            for (int i = 0; i < iCodeArray.size(); i++) {
                                bufferWriter.write(iCodeArray.get(i) + " " + "\n");
                            }

                        } catch (IOException e) {
                            e.printStackTrace();

                        } finally {
                            try {
                                if (bufferWriter != null)
                                    bufferWriter.close();
                                if (fileWriter != null)
                                    fileWriter.close();

                            } catch (IOException ex) {
                                ex.printStackTrace();
                            }
                        }
                    }
                }

                catch (Exception e) {
                    e.printStackTrace();
                }
            }
        } catch (Exception e) {
            System.out.println("Input filename is incorrect");
            e.printStackTrace();
        }
        if (inputIcfile != null) {
            executeRuntime(inputIcfile + "ic");

        } else {

            System.out.println("Error File not created ");
        }
    }

    public static void executeRuntime(String filename) throws Exception {
        new IntermediateCodeReader(filename);
        // System.out.println("call to runtime");

    }

       }
```

Intermediat Code Generator class

```
package sparkyCompiler;

import Model.IntermediateCodeWriter;

/**
 * In this class methods from SparkyBaseVisitor are overloaded for intermediate
 * course generation according to Sparky grammar.
 * @author Sayali Tanawade
 * @author Mayank Batra
 * @since April-20-2020
 * @version 2.0
 */

import sparky.SparkyBaseVisitor;
import sparky.SparkyParser;
import sparkyRuntime.RuntimeConstantKeywords;

public class IntermediateCodeGenerator extends SparkyBaseVisitor<Object> {
    String regexStr = "^[0-9]*$";

    public String getOutput() {
        return IntermediateCodeWriter.getInstance().getIcOutput();
    }

    @Override
    public Object visitProgram(SparkyParser.ProgramContext ctx) {
        return visitChildren(ctx);
    }

    @Override
    public Object visitBall(SparkyParser.BallContext ctx) {
        return visitChildren(ctx);
```

```java
    }

    @Override
    public Object visitDeclare(SparkyParser.DeclareContext ctx) {
        IntermediateCodeWriter.getInstance().addOutput(
                RuntimeConstantKeywords.DECLARE + " " + ctx.getChild(0).getText() + " " + ctx.STUFF().getText());
        if (ctx.NUMBER() != null) {
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.NUMBER().getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STUFF().getText());
        } else if (ctx.getChild(0).getText().contains("string")) {
            visit(ctx.stringdatatype());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.STRINGLITERAL().getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STUFF().getText());
        }

        return null;
    }

    @Override
    public Object visitExpression(SparkyParser.ExpressionContext ctx) {
        return visitChildren(ctx);
    }

    @Override
    public Object visitExpr(SparkyParser.ExprContext ctx) {

        if (ctx.getChildCount() == 1) {
            if (ctx.term().getChildCount() == 1) {
                if (ctx.getChild(0).getText().matches(regexStr)) {
                    IntermediateCodeWriter.getInstance()
                            .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.getChild(0).getText());
                } else {
                    IntermediateCodeWriter.getInstance()
                            .addOutput(RuntimeConstantKeywords.GET + " " + ctx.getChild(0).getText());
                }
            } else {

                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.GET + " " + ctx.term().getChild(0).getText());

                if (ctx.term().getChild(2).getText().matches(regexStr)) {
                    IntermediateCodeWriter.getInstance().addOutput(
                            RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.term().getChild(2).getText());
                } else {
                    IntermediateCodeWriter.getInstance()
                            .addOutput(RuntimeConstantKeywords.GET + " " + ctx.term().getChild(2).getText());
                }
                getArithmaticOperator(ctx.term().getChild(1).getText());
            }
        } else {
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.GET + " " + ctx.getChild(0).getText());
            if (ctx.getChild(2).getText().matches(regexStr)) {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.getChild(2).getText());
            } else {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.GET + " " + ctx.getChild(2).getText());
            }
            getArithmaticOperator(ctx.getChild(1).getText());
        }
        return null;
    }

    public void getArithmaticOperator(String operator) {
        switch (operator) {
        case "+":
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeConstantKeywords.ADDITION);
            break;
        case "-":
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeConstantKeywords.SUBTRACTION);
            break;
        case "*":
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeConstantKeywords.MULTIPLICATION);
            break;
        case "/":
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.OPERATOR + " " + RuntimeConstantKeywords.DIVSION);
            break;
        }
    }

    @Override
    public Object visitAssignment(SparkyParser.AssignmentContext ctx) {
        visit(ctx.expr());
        IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.PUSH + " " + ctx.STUFF().getText());

        return null;
    }

    @Override
    public Object visitIfte(SparkyParser.IfteContext ctx) {
        IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IFTE_START);
        if (ctx.yesnostatement().getText().contains("yup") || ctx.yesnostatement().getText().contains("nup")) {
            IntermediateCodeWriter.getInstance()
```

```java
                        .addOutput(RuntimeConstantKeywords.CHECK_CONDITION + " " + ctx.yesnostatement().getText());
            } else {
                visit(ctx.yesnostatement());
            }

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.CONDITION_FALSE + " "
                    + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.ELSE_START);

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IF_START);

            visit(ctx.in_loop(0));
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IF_END);
            if (ctx.in_loop(1) != null) {
                IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.ELSE_START);
                visit(ctx.in_loop(1));
                IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.ELSE_END);
            }
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IFTE_END);

            return null;
        }

        @Override
        public Object visitLoopum(SparkyParser.LoopumContext ctx) {
            return visitChildren(ctx);
        }

        @Override
        public Object visitLoop_for(SparkyParser.Loop_forContext ctx) {
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.FOR_START);
            visit(ctx.for_declare());
            visit(ctx.for_expression());
            visit(ctx.in_loop());
            visit(ctx.for_expr());
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.FOR_STOP);

            return null;
        }

        @Override
        public Object visitFor_declare(SparkyParser.For_declareContext ctx) {
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.DECLARE + " "
                    + ctx.datatype().getChild(0).getText() + " " + ctx.STUFF().getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.NUMBER().getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STUFF().getText());

            return null;
        }

        @Override
        public Object visitFor_expression(SparkyParser.For_expressionContext ctx) {

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.FOR_CONDITION_START);
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr(0).getText());
            if (ctx.expr(1).getText().matches(regexStr)) {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.expr(1).getText());
            } else {
                IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr(1).getText());
            }

            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.COMPARE_OPERATOR + " " + ctx.YESNOOPERATOR().getText());
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.CONDITION_FALSE + " "
                    + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.FOR_STOP);

            return null;
        }

        @Override
        public Object visitFor_expr(SparkyParser.For_exprContext ctx) {
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr().getChild(0).getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.expr().getChild(2).getText());

            getArithmaticOperator(ctx.expr().getChild(1).getText());

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.PUSH + " " + ctx.STUFF().getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.FOR_CONDITION_START);

            return null;
        }

        @Override
        public Object visitLoop_for_range(SparkyParser.Loop_for_rangeContext ctx) {

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.FOR_START);
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.DECLARE + " int " + ctx.STUFF().getText());
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.NUMBER(0));
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STUFF().getText());
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.FOR_CONDITION_START);
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.STUFF().getText());
            if (ctx.NUMBER(1).getText().matches(regexStr)) {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.NUMBER(1));
            } else {
                IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.NUMBER(1));
            }

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.COMPARE_OPERATOR + " " + "<");
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.CONDITION_FALSE + " "
                    + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.FOR_STOP);
            visit(ctx.in_loop());
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.STUFF().getText());
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.NUMBER(0));
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.OPERATOR + " " + "ADD");
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.INSTRUCTION_PUSH + " " + ctx.STUFF().getText());
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.FOR_CONDITION_START);
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.FOR_STOP);
            return null;
        }

        @Override
        public Object visitLoop_while(SparkyParser.Loop_whileContext ctx) {
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.WHILE_BEGIN);
            if (ctx.yesnostatement().getText().contains("yup") || ctx.yesnostatement().getText().contains("nup")) {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.CHECK_CONDITION + " " + ctx.yesnostatement().getText());
            } else {
                visit(ctx.yesnostatement());
            }

            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.CONDITION_FALSE + " "
                    + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.WHILE_END);
            visit(ctx.in_loop());
            IntermediateCodeWriter.getInstance()
```

```java
                .addOutput(RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.WHILE_BEGIN);
        IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.WHILE_END);

        return null;
    }

    @Override
    public Object visitIn_loop(SparkyParser.In_loopContext ctx) {
        visit(ctx.ball());
        return null;
    }

    @Override
    public Object visitTerm(SparkyParser.TermContext ctx) {
        return visitChildren(ctx);
    }

    @Override
    public Object visitYesnostatement(SparkyParser.YesnostatementContext ctx) {
        if (ctx.expr(0) != null) {
            IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr(0).getText());
            if (ctx.expr(1).getText().matches(regexStr)) {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.INSTRUCTION_STORE + " " + ctx.expr(1).getText());
            } else {
                IntermediateCodeWriter.getInstance()
                        .addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr(1).getText());
            }
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.COMPARE_OPERATOR + " " + ctx.YESNOOPERATOR().getText());
        } else {
            visit(ctx.yesnostatement(0));
            visit(ctx.yesnostatement(1));
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.AND_OR_OPERATOR + " " + ctx.ANDOROPERATOR().getText());
        }

        return null;
    }

    @Override
    public Object visitTernary_operator(SparkyParser.Ternary_operatorContext ctx) {
        IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IFTE_START);
        if (ctx.yesnostatement().getText().contains("yup") || ctx.yesnostatement().getText().contains("nup")) {
            IntermediateCodeWriter.getInstance()
                    .addOutput(RuntimeConstantKeywords.CHECK_CONDITION + " " + ctx.yesnostatement().getText());
        } else {
            visit(ctx.yesnostatement());
        }
```

```java

      IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.CONDITION_FALSE + " "
          + RuntimeConstantKeywords.JUMP + " " + RuntimeConstantKeywords.ELSE_START);

      IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IF_START);
      visit(ctx.in_loop(0));
      IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IF_END);
      if (ctx.in_loop(1) != null) {
        IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.ELSE_START);
        visit(ctx.in_loop(1));
        IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.ELSE_END);
      }
      IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.IFTE_END);

      return null;
    }

    @Override
    public Object visitPrint(SparkyParser.PrintContext ctx) {
      IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.GET + " " + ctx.expr().getText());
      IntermediateCodeWriter.getInstance().addOutput(RuntimeConstantKeywords.PRINT);

      return null;
    }

    @Override
    public Object visitWarna(SparkyParser.WarnaContext ctx) {
      return visitChildren(ctx);
    }

    @Override
    public Object visitHaina(SparkyParser.HainaContext ctx) {
      return visitChildren(ctx);
    }

    @Override
    public Object visitDatatype(SparkyParser.DatatypeContext ctx) {
      return visitChildren(ctx);
    }

    @Override
    public Object visitStringdatatype(SparkyParser.StringdatatypeContext ctx) {

      return visitChildren(ctx);
    }

    @Override
    public Object visitYup(SparkyParser.YupContext ctx) {
      return visitChildren(ctx);
    }

    @Override
```

```
            public Object visitNope(SparkyParser.NopeContext ctx) {
                return visitChildren(ctx);
            }

        }
```

Run time Classes

```
package sparkyRuntime;

/**
 *
 */
public class DataTypes {

    Object value;
    DataTypes(Object value){
        this.value = value;
    }

    public String toString() {
        return String.valueOf(value);
    }

    public String checkDataType() {
        try {
            @SuppressWarnings("unused")
            int i =(Integer)value;
            return "int";
        }
        catch(ClassCastException fu) {
            return checkDouble();
        }
    }

    private String checkDouble() {
        try {
            @SuppressWarnings("unused")
            Double i = (Double)value;
            return "double";
        }
        catch(ClassCastException fu) {
            return checkBoolean();
        }
    }

    private String checkBoolean() {
        try {
            @SuppressWarnings("unused")
            Boolean bo = (Boolean)value;
            return "BOOLEAN";
        }
        catch(ClassCastException fu) {
            return "STRING";
        }
    }

    public int Integer() {
```

```java
        return (Integer)value;
    }

    public double Double() {
        return (Double)value;
    }

    public Boolean Boolean() {
        return (Boolean)value;
    }

    public String String() {
        return (String)value;
    }
}
```

Intermediate Code Reader Class

```java
package sparkyRuntime;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.StringReader;
import java.util.HashMap;
import java.util.Stack;

import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;

public class IntermediateCodeReader {

    public IntermediateCodeReader(String filename) throws Exception {
        program(filename);
    }

    /**
     * Program function reads the Intermediate code file line by line. Variables are
     * stored in Hashmap having value of different data types. To handle this we
     * created class Datatypes.
     *
     * @throws Exception
     */

    private String readFromICFile(String filename) throws IOException {
        String intermediateCode;
        CharStream code = CharStreams.fromFileName(filename);
        intermediateCode = (code.toString().replaceAll("\r", ""));
        return intermediateCode;
    }

    private void program(String filename) throws Exception {

        String fileContent = readFromICFile(filename);
        BufferedReader read = new BufferedReader(new StringReader(fileContent));
        String content;
        HashMap<String, DataTypes> map = new HashMap<String, DataTypes>();
        Stack<DataTypes> local = new Stack<DataTypes>();
        Stack<String> forVariable = new Stack<String>();
        int counter = 0;
        Stack<String> whileVariable = new Stack<String>();
        int counter1 = 0;
        try {

            while ((content = read.readLine()) != null) {
                String[] line = content.split(" ");
                DataTypes value;

                if (line[0].equals("DECLARE")) {
                    // If variable name is already present.
                    if (map.containsKey(line[2])) {
                        throw new Exception("Variable " + line[2] + " is already declared.");
                    }
                    // If variable is declared without definition.
                    if (line.length < 5) {
                        map.put(line[2], null);
                        if (counter != 0) {
                            forVariable.push(line[2]);
                        }
                        if (counter1 != 0) {
                            whileVariable.push(line[2]);
                        }
                    }
                    // If variable is declared with definition.
                    else {

                        value = setValueDataTypes(line[4]);
                        if (line[1].equals(value.checkDataType())) {
                            map.put(line[2], value);
                            if (counter != 0) {
                                forVariable.push(line[2]);
                            }
                            if (counter1 != 0) {
                                whileVariable.push(line[2]);
                            }
                        } else {

                            throw new Exception("Datatype Mismatch during Variable declaration");

                        }
                    }
                }

                else if (line[0].equals("STORE")) {
                    if (line[1].charAt(0) != '"') {
                        value = setValueDataTypes(line[1]);
                        local.push(value);
                    } else if (line[1].charAt(0) == '"') {
                        String output = "";
                        int count = 0;
                        for (int i = 1; i < line.length; i++) {
                            if (i == 1 && line[i].length() != 1) {
                                output += line[i].substring(1, line[i].length()) + " ";

                                count = 1;
                            } else if (i == line.length - 1 && line[i].length() != 1) {

                                output += line[i].substring(0, line[i].length() - 1);
```

```java
                             } else if (i != 1 && i != line.length - 1) {
                                 if (count == 0 && i == 2) {
                                            output += " " + line[i] + " ";
                                        } else {
                                            output += line[i] + " ";
                                        }
                                    }
                                }
                                value = setValueDataTypes(output);
                                local.push(value);
                            }
                        }

                        else if (line[0].equals("PUSH")) {
                            if (!map.containsKey(line[1])) {
                                throw new Exception("Variable " + line[1] + " might not have been declared.");
                            } else {
                                DataTypes mapvar = map.get(line[1]);
                                DataTypes localvar = local.pop();
                                if (mapvar == null || mapvar.checkDataType().equals(localvar.checkDataType())) {
                                    map.put(line[1], localvar);
                                } else {
                                    throw new Exception("Datatype mismatch while assignment");

                                }
                            }
                        }

                        else if (line[0].equals("GET")) {
                            if (!map.containsKey(line[1])) {
                                throw new Exception("Variable " + line[1] + " might not have been declared.");
                            } else {
                                local.push(map.get(line[1]));
                            }
                        }

                        else if (line[0].equals("OPERATOR")) {
                            operator(line[1], local);
                        }

                        else if (line[0].equals("COMPARE_OPERATOR")) {
                            compare(line[1], local);
                        }

                        else if (line[0].equals("AND_OR_OPERATOR")) {
                            if (line[1].equals("and")) {
                                Boolean decand = local.pop().Boolean();
                                DataTypes outand = new DataTypes(decand.equals(local.pop().Boolean()) ? decand : false);
                                local.push(outand);
                            }

                            else if (line[1].equals("or")) {
                                Boolean decor = local.pop().Boolean();
                                DataTypes outor = new DataTypes(decor.equals(local.pop().Boolean()) ? decor : true);
                                local.push(outor);
                            }

                            else if (line[1].equals("not")) {
                                Boolean decnot = local.pop().Boolean();
                                DataTypes outnot = new DataTypes(decnot.equals(true) ? false : true);
                                local.push(outnot);
                            }
                        }

                        else if (line[0].equals("CONDITION_FALSE") && !local.pop().Boolean()) {

                            if (line[2].equals("ELSE_START")) {
                                while ((content = read.readLine()) != null) {
                                    line = content.split(" ");
                                    if (line[0].equals("ELSE_START")) {
                                        break;
                                    }
                                }
                            } else if (line[2].equals("WHILEEND")) {
                                while ((content = read.readLine()) != null) {
                                    line = content.split(" ");
                                    if (line[0].equals("WHILEEND")) {
                                        break;
                                    }
                                }
                                counter1 = 0;
                                while (!whileVariable.empty()) {
                                    map.remove(whileVariable.pop());
                                }
                            } else if (line[2].equals("FOR_STOP")) {
                                while ((content = read.readLine()) != null) {
                                    line = content.split(" ");
                                    if (line[0].equals("FOR_STOP")) {
                                        break;
                                    }
                                }
                                counter = 0;
                                while (!forVariable.empty()) {
                                    map.remove(forVariable.pop());
                                }
                            }
                        }

                        else if (line[0].equals("FOR_START")) {
                            if (counter == 0) {
                                counter = counter + 1;
                            }

                        }

                        else if (line[0].equals("IF_END")) {
                            while ((content = read.readLine()) != null) {
                                line = content.split(" ");
                                if (line[0].equals("ELSE_END")) {
                                    break;
                                }
                            }

                        }
```

```java

                else if (line[0].equals("WHILEBEGIN")) {
                    if (counter1 == 0) {
                        counter1 = counter1 + 1;
                    }

                }

                else if (line[0].equals("JUMP")) {

                    if (line[1].equals("FOR_CONDITION_START")) {
                        fileContent = readFromICFile(filename);
                        read = new BufferedReader(new StringReader(fileContent));
                        while ((content = read.readLine()) != null) {
                            line = content.split(" ");
                            if (line[0].equals("FOR_CONDITION_START")) {
                                break;
                            }
                        }
                    } else if (line[1].equals("WHILEBEGIN")) {
                        fileContent = readFromICFile(filename);
                        read = new BufferedReader(new StringReader(fileContent));
                        while ((content = read.readLine()) != null) {
                            line = content.split(" ");
                            if (line[0].equals("WHILEBEGIN")) {
                                break;
                            }
                        }
                    }
                }

                else if (line[0].equals("PRINT")) {
                    if (local.isEmpty()) {
                        throw new Exception("Nothing to Print.");
                    }
                    System.out.println(local.pop().toString());
                }
            }
        }

        catch (IOException fu) {
            fu.printStackTrace();
        }

    }

    // Checks the data type of input and creates an object of class DataTypes.
    private DataTypes setValueDataTypes(String string) {
        if (isInt(string)) {
            return new DataTypes(Integer.parseInt(string));
        } else if (isdouble(string)) {
            return new DataTypes(Double.parseDouble(string));
        } else if (isbool(string)) {
            return new DataTypes(Boolean.parseBoolean(string));
        } else {
            return new DataTypes(string);
        }
    }

```

```java
            private boolean isInt(String strin) {
                try {
                    Integer.parseInt(strin);
                    return true;
                } catch (NumberFormatException fk) {
                    return false;
                }
            }

            private boolean isdouble(String strin) {
                try {
                    Double.parseDouble(strin);
                    return true;
                } catch (NumberFormatException fk) {
                    return false;
                }
            }

            private boolean isbool(String string) {
                try {
                    Boolean bo = Boolean.parseBoolean(string);
                    if (!bo && !string.equalsIgnoreCase("FALSE")) {
                        return false;
                    }
                    return true;
                } catch (NumberFormatException fk) {
                    return false;
                }
            }

            // Compares the 2 input and pushes the boolean result in stack.
            private void compare(String comparison, Stack<DataTypes> local) throws Exception {

                DataTypes locop2 = local.pop();
                DataTypes locop1 = local.pop();
                DataTypes outcome;
                if (locop1.checkDataType().equals(locop2.checkDataType())) {
                    if (comparison.equals("==")) {
                        outcome = new DataTypes(locop2.toString().equals(locop1.toString()) ? true : false);
                        local.push(outcome);
                    } else if (comparison.equals("<") && locop1.checkDataType().equals("int")) {
                        outcome = new DataTypes(locop2.Integer() > locop1.Integer() ? true : false);
                        local.push(outcome);
                    } else if (comparison.equals("<") && locop1.checkDataType().equals("double")) {
                        outcome = new DataTypes(locop2.Double() > locop1.Double() ? true : false);
                        local.push(outcome);
                    } else if (comparison.equals("<=") && locop1.checkDataType().equals("int")) {
                        outcome = new DataTypes(locop2.Integer() >= locop1.Integer() ? true : false);
                        local.push(outcome);
                    } else if (comparison.equals("<=") && locop1.checkDataType().equals("double")) {
                        outcome = new DataTypes(locop2.Double() >= locop1.Double() ? true : false);
                    local.push(outcome);
                } else if (comparison.equals(">") && locop1.checkDataType().equals("int")) {
                    outcome = new DataTypes(locop2.Integer() < locop1.Integer() ? true : false);
                    local.push(outcome);
                } else if (comparison.equals(">") && locop1.checkDataType().equals("double")) {
                    outcome = new DataTypes(locop2.Double() < locop1.Double() ? true : false);
                    local.push(outcome);
                } else if (comparison.equals(">=") && locop1.checkDataType().equals("int")) {
                    outcome = new DataTypes(locop2.Integer() <= locop1.Integer() ? true : false);
                    local.push(outcome);
                } else if (comparison.equals(">=") && locop1.checkDataType().equals("double")) {
                    outcome = new DataTypes(locop2.Double() <= locop1.Double() ? true : false);
                    local.push(outcome);
                } else {
                    throw new Exception("Incorrect Datatype while comparison");
                }
            } else {
                throw new Exception("Datatype mismatch while comparison");
            }

        }

        // Performs Arithmetic operation on inputs and pushes the result in stack.
        private void operator(String operation, Stack<DataTypes> local) throws Exception {
            DataTypes locop2 = local.pop();
            DataTypes locop1 = local.pop();
            DataTypes outcome;
            if (locop1.checkDataType().equals(locop2.checkDataType()) && locop1.checkDataType().equals("int")) {
                if (operation.equals("ADD")) {
                    outcome = new DataTypes(locop1.Integer() + locop2.Integer());
                    local.push(outcome);
                } else if (operation.equals("SUBTRACT")) {
                    outcome = new DataTypes(locop1.Integer() - locop2.Integer());
                    local.push(outcome);
                } else if (operation.equals("MULTIPLY")) {
                    outcome = new DataTypes(locop1.Integer() * locop2.Integer());
                    local.push(outcome);
                } else if (operation.equals("DIVIDE")) {
                    if (locop2.Integer() != 0) {
                        outcome = new DataTypes(locop1.Integer() / locop2.Integer());
                        local.push(outcome);
                    } else {
                        throw new Exception("Denominator can't be zero.");
                    }
                }
            } else if (locop1.checkDataType().equals(locop2.checkDataType()) && locop1.checkDataType().equals("double")) {
                if (operation.equals("ADD")) {
                    outcome = new DataTypes(locop1.Double() + locop2.Double());
                    local.push(outcome);
                    } else if (operation.equals("SUBTRACT")) {
                        outcome = new DataTypes(locop1.Double() - locop2.Double());
                        local.push(outcome);
                    } else if (operation.equals("MULTIPLY")) {
                        outcome = new DataTypes(locop1.Double() * locop2.Double());
                        local.push(outcome);
                    } else if (operation.equals("DIVIDE")) {
                        if (locop2.Integer() != 0) {
                            outcome = new DataTypes(locop1.Double() / locop2.Double());

                            local.push(outcome);
                        } else {
                            throw new Exception("Denominator can't be zero.");
                        }
                    }
                } else {
                    throw new Exception("Datatype of both the variables should be same."
);
                }
            }
        }
```

Runtime Constant Class

```
package sparkyRuntime;

public interface RuntimeConstantKeywords {

    public static final String equal_to = "EQUAL_TO";
    public static final String INSTRUCTION_STORE = "STORE";
    public static final String INSTRUCTION_PUSH = "PUSH";
    public static final String DECLARE = "DECLARE";
    public static final String INT_DEFAULT = "0";

    public static final String GET = "GET";
    public static final String OPERATOR = "OPERATOR";
    public static final String PUSH = "PUSH";

    public static final String ADDITION = "ADD";
    public static final String SUBTRACTION = "SUBTRACT";
    public static final String MULTIPLICATION = "MULTIPLY";
    public static final String DIVSION = "DIVIDE";
    public static final String BOOLEAN = "YESNO";
    public static final String CONDITION_TRUE = "CONDITION_TRUE";

    public static final String IF_START = "IF_START";
    public static final String IF_END = "IF_END";
    public static final String ELSE_START = "ELSE_START";
    public static final String ELSE_END = "ELSE_END";
    public static final String IFTE_START = "IFTE_START";
    public static final String IFTE_END = "IFTE_END";
    public static final String CHECK_CONDITION = "CHECK_CONDITION";
    public static final String CONDITION_FALSE = "CONDITION_FALSE";
    public static final String JUMP = "JUMP";

    public static final String COMPARE_OPERATOR = "COMPARE_OPERATOR";

    public static final String FOR_START = "FOR_START";
```

```java
    public static final String FOR_STOP = "FOR_STOP";
    public static final String FOR_INIT = "FOR_INIT";
    public static final String FOR_EXPRESSION = "FOR_EXPRESSION";
    public static final String FOR_CONDITION_START = "FOR_CONDITION_START";
    public static final String FOR_CONDITION_STOP = "FOR_CONDITION_STOP";
    public static final String FOR_UPDATE_START = "FOR_UPDATE_START";
    public static final String FOR_UPDATE_STOP = "FOR_UPDATE_STOP";
    public static final String FOR_VARIABLE = "FOR_VARIABLE";

    // Keywords for While
    public static final String WHILE_BEGIN = "WHILEBEGIN";
    public static final String WHILE_END = "WHILEEND";

    /*
     * Existing keywords used for while GET , COMPARE_OPERATOR,CONDITION_NOT_TRUE,
     * STORE, OPERATOR etc.
     *
     */
    public static final String PRINT = "PRINT";
    public static final String AND_OR_OPERATOR = "AND_OR_OPERATOR";

}
```

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

Example1: arithmaticOps.sparky

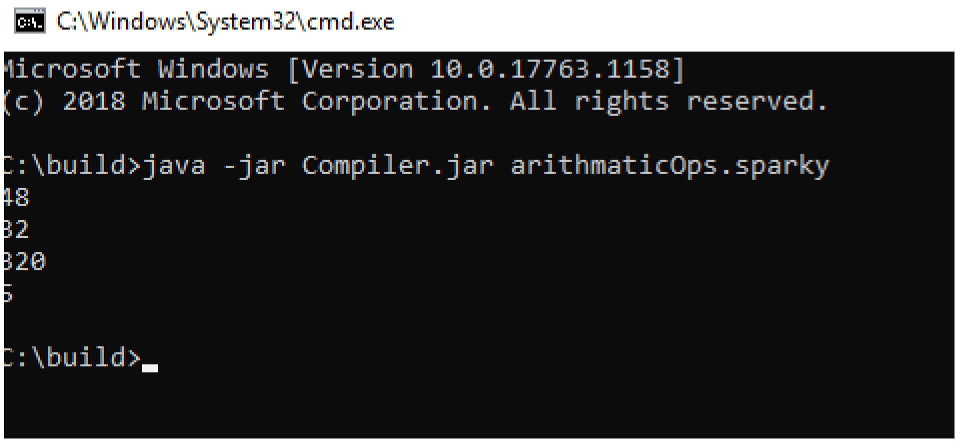
```
Live
int a=40;
int b=8;
int result;
result=a+b;
print(result);
result=a-b;
print(result);
result=a*b;
print(result);
result=a/b;
print(result);
Die
```

Corresponding intermediate code

```
DECLARE int a
STORE 40
PUSH a
DECLARE int b
STORE 8
PUSH b
DECLARE int result
GET a
GET b
OPERATOR ADD
PUSH result
GET result
PRINT
```

```
GET a
GET b
OPERATOR SUBTRACT
PUSH result
GET result
PRINT
GET a
GET b
OPERATOR MULTIPLY
PUSH result
GET result
PRINT
GET a
GET b
OPERATOR DIVIDE
PUSH result
GET result
PRINT
```

Output



C:\Windows\System32\cmd.exe

```
icrosoft Windows [Version 10.0.17763.1158]
(c) 2018 Microsoft Corporation. All rights reserved.

:\build>java -jar Compiler.jar arithmaticOps.sparky
48
32
320
5

:\build>_
```

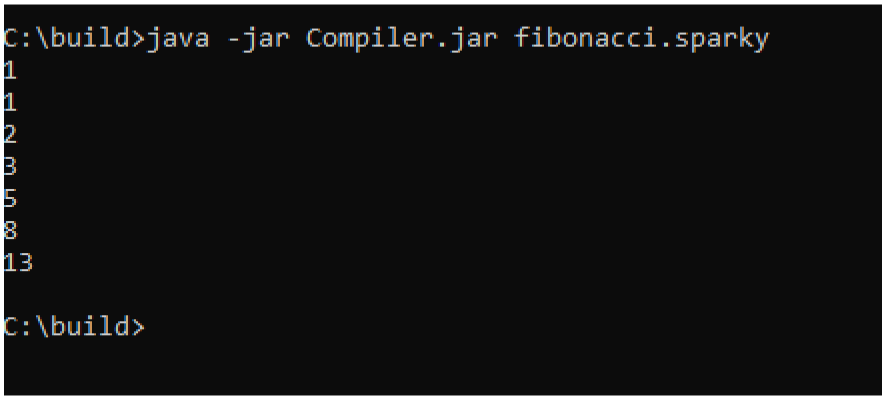
Example 2: fibonacci.sparky

```
Live
int count = 7;
int counter =1;
int firstFib=1;
int secondFib=1;
int sum;
while counter<=count
{
print(firstFib);
sum=firstFib+secondFib;
firstFib = secondFib;
secondFib=sum;
counter=counter+1;
}
Die
```

Corresponding Intermediate code

```
DECLARE int count
STORE 7
PUSH count
DECLARE int counter
STORE 1
PUSH counter
DECLARE int firstFib
STORE 1
PUSH firstFib
DECLARE int secondFib
STORE 1
PUSH secondFib
DECLARE int sum
WHILEBEGIN
GET counter
GET count
COMPARE_OPERATOR <=
CONDITION_FALSE JUMP WHILEEND
GET firstFib
PRINT
GET firstFib
GET secondFib
OPERATOR ADD
PUSH sum
GET secondFib
PUSH firstFib
GET sum
PUSH secondFib
GET counter
STORE 1
OPERATOR ADD
PUSH counter
JUMP WHILEBEGIN
WHILEEND
```

Output



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

C:\build>
```

Example 3: factorial.sparky

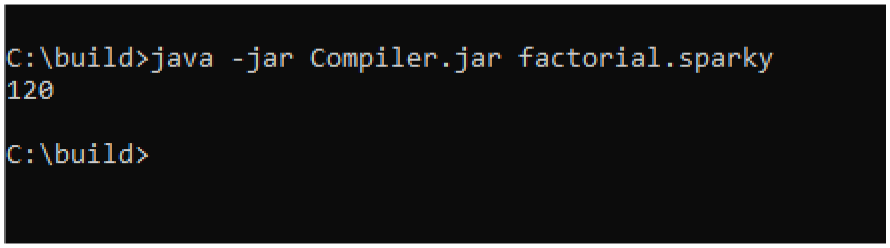
```
Live
int result = 1;
int n=5;
```

```
for (int i=2; i<=n; i=i+1)
{result=result*i;}
print(result);
Die
```

Corresponding Intermediate Code

```
DECLARE int result
STORE 1
PUSH result
DECLARE int n
STORE 5
PUSH n
FOR_START
DECLARE int i
STORE 2
PUSH i
FOR_CONDITION_START
GET i
GET n
COMPARE_OPERATOR <=
CONDITION_FALSE JUMP FOR_STOP
GET result
GET i
OPERATOR MULTIPLY
PUSH result
GET i
STORE 1
OPERATOR ADD
PUSH i
JUMP FOR_CONDITION_START
FOR_STOP
GET result
PRINT
```

Output



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

C:\build>
```