```
// Generated from C:\Users\u005Cuser\Documents\Compiler\Skeleton\src/decaf/
    DecafParser.g4 by ANTLR 4.6
package decaf;
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 DecafParser extends Parser {
        static { RuntimeMetaData.checkVersion("4.6", RuntimeMetaData.VERSION); }

        protected static final DFA[] _decisionToDFA;
        protected static final PredictionContextCache _sharedContextCache =
                new PredictionContextCache();
        public static final int
                BOOLEAN=1, BREAK=2, CALLOUT=3, CLASS=4, CONTINUE=5, ELSE=6,
                    FALSE=7, FOR=8,
                IF=9, INT=10, RETURN=11, TRUE=12, VOID=13, LCURLY=14, RCURLY=15,
                     LBRACE=16,
                RBRACE=17, LSQUARE=18, RSQUARE=19, COMMA=20, ASSIGN=21,
                    PLUSASSIGN=22,
                MINUSASSIGN=23, PLUS=24, MINUS=25, MULT=26, MOD=27, DIV=28, NOT
                    =29, AND=30,
                OR=31, EQ=32, NEQ=33, LT=34, GT=35, LTE=36, GTE=37, END=38, ID
                    =39, WS_=40,
                SL_COMMENT=41, CHAR=42, STRING=43, NUMBER=44;
        public static final int
                RULE_program = 0, RULE_field_name = 1, RULE_field_decl = 2,
                    RULE_method_decl = 3,
                RULE_meth_name = 4, RULE_meth_type = 5, RULE_arg_type = 6,
                    RULE_block = 7,
                RULE_var_decl = 8, RULE_var_name = 9, RULE_type = 10,
                    RULE_statement = 11,
                RULE_assign_op = 12, RULE_math_assign = 13, RULE_method_call =
                    14, RULE_method_name = 15,
                RULE_location = 16, RULE_expr = 17, RULE_callout_arg = 18,
                    RULE_bin_op = 19,
                RULE_arith_op = 20, RULE_rel_op = 21, RULE_eq_op = 22,
                    RULE_cond_op = 23,
                RULE_literal = 24, RULE_bool_literal = 25;
        public static final String[] ruleNames = {
                "program", "field_name", "field_decl", "method_decl", "meth_name
                    ", "meth_type",
                "arg_type", "block", "var_decl", "var_name", "type", "statement"
                    , "assign_op",
                "math_assign", "method_call", "method_name", "location", "expr",
                     "callout_arg",
                "bin_op", "arith_op", "rel_op", "eq_op", "cond_op", "literal", "
                    bool_literal"
        };

        private static final String[] _LITERAL_NAMES = {
```

```
        null, "'boolean'", "'break'", "'callout'", "'class'", "'continue
            '", "'else'",
        "'false'", "'for'", "'if'", "'int'", "'return'", "'true'", "'
            void'", "'{'",
        "'}'", "'('", "')'", "'['", "']'", "','", "'='", "'+='", "'-='",
             "'+'",
        "'-'", "'*'", "'%'", "'/'", "'!'", "'&&'", "'||'", "'=='", "'!='
            ", "'<'",
        "'>'", "'<='", "'>='", "';'"
};
private static final String[] _SYMBOLIC_NAMES = {
        null, "BOOLEAN", "BREAK", "CALLOUT", "CLASS", "CONTINUE", "ELSE"
            , "FALSE",
        "FOR", "IF", "INT", "RETURN", "TRUE", "VOID", "LCURLY", "RCURLY"
            , "LBRACE",
        "RBRACE", "LSQUARE", "RSQUARE", "COMMA", "ASSIGN", "PLUSASSIGN",
             "MINUSASSIGN",
        "PLUS", "MINUS", "MULT", "MOD", "DIV", "NOT", "AND", "OR", "EQ",
             "NEQ",
        "LT", "GT", "LTE", "GTE", "END", "ID", "WS_", "SL_COMMENT", "
            CHAR", "STRING",
        "NUMBER"
};
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 "DecafParser.g4"; }

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

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

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

public DecafParser(TokenStream input) {
        super(input);
        _interp = new ParserATNSimulator(this,_ATN,_decisionToDFA,
            _sharedContextCache);
}
public static class ProgramContext extends ParserRuleContext {
        public TerminalNode CLASS() { return getToken(DecafParser.CLASS,
             0); }
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public TerminalNode LCURLY() { return getToken(DecafParser.
            LCURLY, 0); }
        public TerminalNode RCURLY() { return getToken(DecafParser.
            RCURLY, 0); }
        public TerminalNode EOF() { return getToken(DecafParser.EOF, 0);
             }
        public List<Field_declContext> field_decl() {
                return getRuleContexts(Field_declContext.class);
        }
        public Field_declContext field_decl(int i) {
                return getRuleContext(Field_declContext.class,i);
        }
        public List<Method_declContext> method_decl() {
                return getRuleContexts(Method_declContext.class);
        }
        public Method_declContext method_decl(int i) {
                return getRuleContext(Method_declContext.class,i);
        }
        public ProgramContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_program; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterProgram(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitProgram(this);
        }
}

public final ProgramContext program() throws RecognitionException {
```

```
ProgramContext _localctx = new ProgramContext(_ctx, getState());
enterRule(_localctx, 0, RULE_program);
int _la;
try {
        int _alt;
        enterOuterAlt(_localctx, 1);
        {
        setState(52);
        match(CLASS);
        setState(53);
        match(ID);
        setState(54);
        match(LCURLY);
        setState(58);
        _errHandler.sync(this);
        _alt = getInterpreter().adaptivePredict(_input,0,_ctx);
        while ( _alt!=2 && _alt!=org.antlr.v4.runtime.atn.ATN.
            INVALID_ALT_NUMBER ) {
                if ( _alt==1 ) {
                        {
                        {
                        setState(55);
                        field_decl();
                        }
                        }
                }
                setState(60);
                _errHandler.sync(this);
                _alt = getInterpreter().adaptivePredict(_input
                    ,0,_ctx);
        }
        setState(64);
        _errHandler.sync(this);
        _la = _input.LA(1);
        while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L <<
            BOOLEAN) | (1L << INT) | (1L << VOID))) != 0)) {
                {
                {
                setState(61);
                method_decl();
                }
                }
                setState(66);
                _errHandler.sync(this);
                _la = _input.LA(1);
        }
        setState(67);
        match(RCURLY);
        setState(68);
        match(EOF);
        }
}
catch (RecognitionException re) {
        _localctx.exception = re;
        _errHandler.reportError(this, re);
        _errHandler.recover(this, re);
}
```

```
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Field_nameContext extends ParserRuleContext {
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public TerminalNode LSQUARE() { return getToken(DecafParser.
            LSQUARE, 0); }
        public TerminalNode NUMBER() { return getToken(DecafParser.
            NUMBER, 0); }
        public TerminalNode RSQUARE() { return getToken(DecafParser.
            RSQUARE, 0); }
        public Field_nameContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_field_name; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterField_name(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitField_name(this);
        }
}

public final Field_nameContext field_name() throws RecognitionException
    {
        Field_nameContext _localctx = new Field_nameContext(_ctx,
            getState());
        enterRule(_localctx, 2, RULE_field_name);
        try {
                setState(75);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,2,_ctx)
                     ) {
                case 1:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(70);
                        match(ID);
                        }
                        break;
                case 2:
                        enterOuterAlt(_localctx, 2);
                        {
                        {
                        setState(71);
                        match(ID);
                        setState(72);
                        match(LSQUARE);
                        setState(73);
```

```
                        match(NUMBER);
                        setState(74);
                        match(RSQUARE);
                        }
                        }
                        break;
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Field_declContext extends ParserRuleContext {
        public TypeContext type() {
                return getRuleContext(TypeContext.class,0);
        }
        public List<Field_nameContext> field_name() {
                return getRuleContexts(Field_nameContext.class);
        }
        public Field_nameContext field_name(int i) {
                return getRuleContext(Field_nameContext.class,i);
        }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public List<TerminalNode> COMMA() { return getTokens(DecafParser
            .COMMA); }
        public TerminalNode COMMA(int i) {
                return getToken(DecafParser.COMMA, i);
        }
        public Field_declContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_field_decl; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterField_decl(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitField_decl(this);
        }
}

public final Field_declContext field_decl() throws RecognitionException
    {
        Field_declContext _localctx = new Field_declContext(_ctx,
            getState());
```

```
        enterRule(_localctx, 4, RULE_field_decl);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(77);
                type();
                setState(78);
                field_name();
                setState(83);
                _errHandler.sync(this);
                _la = _input.LA(1);
                while (_la==COMMA) {
                        {
                        {
                        setState(79);
                        match(COMMA);
                        setState(80);
                        field_name();
                        }
                        }
                        setState(85);
                        _errHandler.sync(this);
                        _la = _input.LA(1);
                }
                setState(86);
                match(END);
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Method_declContext extends ParserRuleContext {
        public Meth_nameContext meth_name() {
                return getRuleContext(Meth_nameContext.class,0);
        }
        public TerminalNode LBRACE() { return getToken(DecafParser.
            LBRACE, 0); }
        public TerminalNode RBRACE() { return getToken(DecafParser.
            RBRACE, 0); }
        public BlockContext block() {
                return getRuleContext(BlockContext.class,0);
        }
        public Meth_typeContext meth_type() {
                return getRuleContext(Meth_typeContext.class,0);
        }
        public List<Arg_typeContext> arg_type() {
                return getRuleContexts(Arg_typeContext.class);
        }
```

```
        public Arg_typeContext arg_type(int i) {
                return getRuleContext(Arg_typeContext.class,i);
        }
        public List<TerminalNode> ID() { return getTokens(DecafParser.ID
            ); }
        public TerminalNode ID(int i) {
                return getToken(DecafParser.ID, i);
        }
        public List<TerminalNode> COMMA() { return getTokens(DecafParser
            .COMMA); }
        public TerminalNode COMMA(int i) {
                return getToken(DecafParser.COMMA, i);
        }
        public Method_declContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_method_decl; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMethod_decl(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMethod_decl(this);
        }
}

public final Method_declContext method_decl() throws
    RecognitionException {
        Method_declContext _localctx = new Method_declContext(_ctx,
            getState());
        enterRule(_localctx, 6, RULE_method_decl);
        int _la;
        try {
                int _alt;
                enterOuterAlt(_localctx, 1);
                {
                {
                setState(88);
                meth_type();
                }
                setState(89);
                meth_name();
                setState(90);
                match(LBRACE);
                setState(103);
                _errHandler.sync(this);
                _la = _input.LA(1);
                if (_la==BOOLEAN || _la==INT) {
                        {
                        setState(97);
                        _errHandler.sync(this);
                        _alt = getInterpreter().adaptivePredict(_input
                            ,4,_ctx);
```

```
                        while ( _alt!=2 && _alt!=org.antlr.v4.runtime.
                            atn.ATN.INVALID_ALT_NUMBER ) {
                                if ( _alt==1 ) {
                                        {
                                        {
                                        setState(91);
                                        arg_type();
                                        setState(92);
                                        match(ID);
                                        setState(93);
                                        match(COMMA);
                                        }
                                        }
                                }
                                setState(99);
                                _errHandler.sync(this);
                                _alt = getInterpreter().adaptivePredict(
                                    _input,4,_ctx);
                        }
                        setState(100);
                        arg_type();
                        setState(101);
                        match(ID);
                        }
                }

                setState(105);
                match(RBRACE);
                setState(106);
                block();
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Meth_nameContext extends ParserRuleContext {
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public Meth_nameContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_meth_name; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMeth_name(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
```

```
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMeth_name(this);
        }
}

public final Meth_nameContext meth_name() throws RecognitionException {
        Meth_nameContext _localctx = new Meth_nameContext(_ctx, getState
            ());
        enterRule(_localctx, 8, RULE_meth_name);
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(108);
                match(ID);
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Meth_typeContext extends ParserRuleContext {
        public TypeContext type() {
                return getRuleContext(TypeContext.class,0);
        }
        public TerminalNode VOID() { return getToken(DecafParser.VOID,
            0); }
        public Meth_typeContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_meth_type; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMeth_type(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMeth_type(this);
        }
}

public final Meth_typeContext meth_type() throws RecognitionException {
        Meth_typeContext _localctx = new Meth_typeContext(_ctx, getState
            ());
        enterRule(_localctx, 10, RULE_meth_type);
        try {
                setState(112);
                _errHandler.sync(this);
```

```
                switch (_input.LA(1)) {
                case BOOLEAN:
                case INT:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(110);
                        type();
                        }
                        break;
                case VOID:
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(111);
                        match(VOID);
                        }
                        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 Arg_typeContext extends ParserRuleContext {
        public TypeContext type() {
                return getRuleContext(TypeContext.class,0);
        }
        public Arg_typeContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_arg_type; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterArg_type(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitArg_type(this);
        }
}

public final Arg_typeContext arg_type() throws RecognitionException {
        Arg_typeContext _localctx = new Arg_typeContext(_ctx, getState()
            );
        enterRule(_localctx, 12, RULE_arg_type);
        try {
```

```
                enterOuterAlt(_localctx, 1);
                {
                setState(114);
                type();
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class BlockContext extends ParserRuleContext {
        public TerminalNode LCURLY() { return getToken(DecafParser.
            LCURLY, 0); }
        public TerminalNode RCURLY() { return getToken(DecafParser.
            RCURLY, 0); }
        public List<Var_declContext> var_decl() {
                return getRuleContexts(Var_declContext.class);
        }
        public Var_declContext var_decl(int i) {
                return getRuleContext(Var_declContext.class,i);
        }
        public List<StatementContext> statement() {
                return getRuleContexts(StatementContext.class);
        }
        public StatementContext statement(int i) {
                return getRuleContext(StatementContext.class,i);
        }
        public BlockContext(ParserRuleContext parent, int invokingState)
             {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_block; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterBlock(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitBlock(this);
        }
}

public final BlockContext block() throws RecognitionException {
        BlockContext _localctx = new BlockContext(_ctx, getState());
        enterRule(_localctx, 14, RULE_block);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
```

```
                {
                setState(116);
                match(LCURLY);
                setState(120);
                _errHandler.sync(this);
                _la = _input.LA(1);
                while (_la==BOOLEAN || _la==INT) {
                        {
                        {
                        setState(117);
                        var_decl();
                        }
                        }
                        setState(122);
                        _errHandler.sync(this);
                        _la = _input.LA(1);
                }
                setState(126);
                _errHandler.sync(this);
                _la = _input.LA(1);
                while ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L <<
                    BREAK) | (1L << CALLOUT) | (1L << CONTINUE) | (1L <<
                    FOR) | (1L << IF) | (1L << RETURN) | (1L << LCURLY) |
                     (1L << ID))) != 0)) {
                        {
                        {
                        setState(123);
                        statement();
                        }
                        }
                        setState(128);
                        _errHandler.sync(this);
                        _la = _input.LA(1);
                }
                setState(129);
                match(RCURLY);
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Var_declContext extends ParserRuleContext {
        public TypeContext type() {
                return getRuleContext(TypeContext.class,0);
        }
        public List<Var_nameContext> var_name() {
                return getRuleContexts(Var_nameContext.class);
        }
        public Var_nameContext var_name(int i) {
```

```
                return getRuleContext(Var_nameContext.class,i);
        }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public List<TerminalNode> COMMA() { return getTokens(DecafParser
            .COMMA); }
        public TerminalNode COMMA(int i) {
                return getToken(DecafParser.COMMA, i);
        }
        public Var_declContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_var_decl; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterVar_decl(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitVar_decl(this);
        }
}

public final Var_declContext var_decl() throws RecognitionException {
        Var_declContext _localctx = new Var_declContext(_ctx, getState()
            );
        enterRule(_localctx, 16, RULE_var_decl);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(131);
                type();
                setState(132);
                var_name();
                setState(137);
                _errHandler.sync(this);
                _la = _input.LA(1);
                while (_la==COMMA) {
                        {
                        {
                        setState(133);
                        match(COMMA);
                        setState(134);
                        var_name();
                        }
                        }
                        setState(139);
                        _errHandler.sync(this);
                        _la = _input.LA(1);
                }
                setState(140);
                match(END);
                }
```

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

public static class Var_nameContext extends ParserRuleContext {
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public Var_nameContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_var_name; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterVar_name(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitVar_name(this);
        }
}

public final Var_nameContext var_name() throws RecognitionException {
        Var_nameContext _localctx = new Var_nameContext(_ctx, getState()
            );
        enterRule(_localctx, 18, RULE_var_name);
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(142);
                match(ID);
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class TypeContext extends ParserRuleContext {
        public TerminalNode INT() { return getToken(DecafParser.INT, 0);
             }
        public TerminalNode BOOLEAN() { return getToken(DecafParser.
```

```
            BOOLEAN, 0); }
        public TypeContext(ParserRuleContext parent, int invokingState)
            {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_type; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterType(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitType(this);
        }
}

public final TypeContext type() throws RecognitionException {
        TypeContext _localctx = new TypeContext(_ctx, getState());
        enterRule(_localctx, 20, RULE_type);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(144);
                _la = _input.LA(1);
                if ( !(_la==BOOLEAN || _la==INT) ) {
                _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 StatementContext extends ParserRuleContext {
        public StatementContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_statement; }

        public StatementContext() { }
```

```
        public void copyFrom(StatementContext ctx) {
                super.copyFrom(ctx);
        }
}
public static class ReturnContext extends StatementContext {
        public TerminalNode RETURN() { return getToken(DecafParser.
            RETURN, 0); }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
        }
        public ReturnContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterReturn(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitReturn(this);
        }
}
public static class MCContext extends StatementContext {
        public Method_callContext method_call() {
                return getRuleContext(Method_callContext.class,0);
        }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public MCContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMC(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMC(this);
        }
}
public static class ForContext extends StatementContext {
        public TerminalNode FOR() { return getToken(DecafParser.FOR, 0);
             }
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public TerminalNode ASSIGN() { return getToken(DecafParser.
            ASSIGN, 0); }
        public List<ExprContext> expr() {
                return getRuleContexts(ExprContext.class);
        }
        public ExprContext expr(int i) {
                return getRuleContext(ExprContext.class,i);
        }
        public TerminalNode COMMA() { return getToken(DecafParser.COMMA,
             0); }
        public BlockContext block() {
```

```
                return getRuleContext(BlockContext.class,0);
        }
        public ForContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterFor(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitFor(this);
        }
}
public static class BreakContext extends StatementContext {
        public TerminalNode BREAK() { return getToken(DecafParser.BREAK,
             0); }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public BreakContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterBreak(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitBreak(this);
        }
}
public static class AssignContext extends StatementContext {
        public LocationContext location() {
                return getRuleContext(LocationContext.class,0);
        }
        public Assign_opContext assign_op() {
                return getRuleContext(Assign_opContext.class,0);
        }
        public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
        }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public AssignContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterAssign(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitAssign(this);
        }
}
public static class BlContext extends StatementContext {
        public BlockContext block() {
```

```
                return getRuleContext(BlockContext.class,0);
        }
        public BlContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterBl(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitBl(this);
        }
}
public static class ContinueContext extends StatementContext {
        public TerminalNode CONTINUE() { return getToken(DecafParser.
            CONTINUE, 0); }
        public TerminalNode END() { return getToken(DecafParser.END, 0);
             }
        public ContinueContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterContinue(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitContinue(this);
        }
}
public static class IfContext extends StatementContext {
        public TerminalNode IF() { return getToken(DecafParser.IF, 0); }
        public TerminalNode LBRACE() { return getToken(DecafParser.
            LBRACE, 0); }
        public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
        }
        public TerminalNode RBRACE() { return getToken(DecafParser.
            RBRACE, 0); }
        public List<BlockContext> block() {
                return getRuleContexts(BlockContext.class);
        }
        public BlockContext block(int i) {
                return getRuleContext(BlockContext.class,i);
        }
        public TerminalNode ELSE() { return getToken(DecafParser.ELSE,
            0); }
        public IfContext(StatementContext ctx) { copyFrom(ctx); }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterIf(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
```

```
                    DecafParserListener)listener).exitIf(this);
        }
}

public final StatementContext statement() throws RecognitionException {
        StatementContext _localctx = new StatementContext(_ctx, getState
            ());
        enterRule(_localctx, 22, RULE_statement);
        int _la;
        try {
                setState(181);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,12,_ctx
                    ) ) {
                case 1:
                        _localctx = new AssignContext(_localctx);
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(146);
                        location();
                        setState(147);
                        assign_op();
                        setState(148);
                        expr(0);
                        setState(149);
                        match(END);
                        }
                        break;
                case 2:
                        _localctx = new MCContext(_localctx);
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(151);
                        method_call();
                        setState(152);
                        match(END);
                        }
                        break;
                case 3:
                        _localctx = new IfContext(_localctx);
                        enterOuterAlt(_localctx, 3);
                        {
                        setState(154);
                        match(IF);
                        setState(155);
                        match(LBRACE);
                        setState(156);
                        expr(0);
                        setState(157);
                        match(RBRACE);
                        setState(158);
                        block();
                        setState(161);
                        _errHandler.sync(this);
                        _la = _input.LA(1);
                        if (_la==ELSE) {
                                {
```

```
                setState(159);
                match(ELSE);
                setState(160);
                block();
                }
            }

            }
            break;
        case 4:
            _localctx = new ForContext(_localctx);
            enterOuterAlt(_localctx, 4);
            {
            setState(163);
            match(FOR);
            setState(164);
            match(ID);
            setState(165);
            match(ASSIGN);
            setState(166);
            expr(0);
            setState(167);
            match(COMMA);
            setState(168);
            expr(0);
            setState(169);
            block();
            }
            break;
        case 5:
            _localctx = new ReturnContext(_localctx);
            enterOuterAlt(_localctx, 5);
            {
            setState(171);
            match(RETURN);
            setState(173);
            _errHandler.sync(this);
            _la = _input.LA(1);
            if ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L
                 << CALLOUT) | (1L << FALSE) | (1L << TRUE) |
                 (1L << LBRACE) | (1L << MINUS) | (1L << NOT)
                 | (1L << ID) | (1L << CHAR) | (1L << NUMBER)
                ))) != 0)) {
                {
                setState(172);
                expr(0);
                }
            }

            setState(175);
            match(END);
            }
            break;
        case 6:
            _localctx = new BreakContext(_localctx);
            enterOuterAlt(_localctx, 6);
            {
```

```
                        setState(176);
                        match(BREAK);
                        setState(177);
                        match(END);
                        }
                        break;
                case 7:
                        _localctx = new ContinueContext(_localctx);
                        enterOuterAlt(_localctx, 7);
                        {
                        setState(178);
                        match(CONTINUE);
                        setState(179);
                        match(END);
                        }
                        break;
                case 8:
                        _localctx = new BlContext(_localctx);
                        enterOuterAlt(_localctx, 8);
                        {
                        setState(180);
                        block();
                        }
                        break;
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
}

public static class Assign_opContext extends ParserRuleContext {
        public TerminalNode ASSIGN() { return getToken(DecafParser.
            ASSIGN, 0); }
        public Math_assignContext math_assign() {
                return getRuleContext(Math_assignContext.class,0);
        }
        public Assign_opContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_assign_op; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterAssign_op(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitAssign_op(this);
```

```
        }
}

public final Assign_opContext assign_op() throws RecognitionException {
        Assign_opContext _localctx = new Assign_opContext(_ctx, getState
            ());
        enterRule(_localctx, 24, RULE_assign_op);
        try {
                setState(185);
                _errHandler.sync(this);
                switch (_input.LA(1)) {
                case ASSIGN:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(183);
                        match(ASSIGN);
                        }
                        break;
                case PLUSASSIGN:
                case MINUSASSIGN:
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(184);
                        math_assign();
                        }
                        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 Math_assignContext extends ParserRuleContext {
        public TerminalNode PLUSASSIGN() { return getToken(DecafParser.
            PLUSASSIGN, 0); }
        public TerminalNode MINUSASSIGN() { return getToken(DecafParser.
            MINUSASSIGN, 0); }
        public Math_assignContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_math_assign; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMath_assign(this);
        }
        @Override
```

```
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMath_assign(this);
        }
}

public final Math_assignContext math_assign() throws
    RecognitionException {
        Math_assignContext _localctx = new Math_assignContext(_ctx,
            getState());
        enterRule(_localctx, 26, RULE_math_assign);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(187);
                _la = _input.LA(1);
                if ( !(_la==PLUSASSIGN || _la==MINUSASSIGN) ) {
                _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 Method_callContext extends ParserRuleContext {
        public Method_nameContext method_name() {
                return getRuleContext(Method_nameContext.class,0);
        }
        public TerminalNode LBRACE() { return getToken(DecafParser.
            LBRACE, 0); }
        public TerminalNode RBRACE() { return getToken(DecafParser.
            RBRACE, 0); }
        public List<ExprContext> expr() {
                return getRuleContexts(ExprContext.class);
        }
        public ExprContext expr(int i) {
                return getRuleContext(ExprContext.class,i);
        }
        public List<TerminalNode> COMMA() { return getTokens(DecafParser
            .COMMA); }
        public TerminalNode COMMA(int i) {
                return getToken(DecafParser.COMMA, i);
```

```
        }
        public TerminalNode CALLOUT() { return getToken(DecafParser.
            CALLOUT, 0); }
        public TerminalNode STRING() { return getToken(DecafParser.
            STRING, 0); }
        public List<Callout_argContext> callout_arg() {
                return getRuleContexts(Callout_argContext.class);
        }
        public Callout_argContext callout_arg(int i) {
                return getRuleContext(Callout_argContext.class,i);
        }
        public Method_callContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_method_call; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMethod_call(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMethod_call(this);
        }
}

public final Method_callContext method_call() throws
    RecognitionException {
        Method_callContext _localctx = new Method_callContext(_ctx,
            getState());
        enterRule(_localctx, 28, RULE_method_call);
        int _la;
        try {
                int _alt;
                setState(215);
                _errHandler.sync(this);
                switch (_input.LA(1)) {
                case ID:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(189);
                        method_name();
                        setState(190);
                        match(LBRACE);
                        setState(200);
                        _errHandler.sync(this);
                        _la = _input.LA(1);
                        if ((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L
                             << CALLOUT) | (1L << FALSE) | (1L << TRUE) |
                             (1L << LBRACE) | (1L << MINUS) | (1L << NOT)
                             | (1L << ID) | (1L << CHAR) | (1L << NUMBER)
                            )) != 0)) {
                                {
                                setState(196);
                                _errHandler.sync(this);
```

```
                    _alt = getInterpreter().adaptivePredict(
                        _input,14,_ctx);
                    while ( _alt!=2 && _alt!=org.antlr.v4.
                        runtime.atn.ATN.INVALID_ALT_NUMBER )
                        {
                            if ( _alt==1 ) {
                                    {
                                    {
                                    setState(191);
                                    expr(0);
                                    setState(192);
                                    match(COMMA);
                                    }
                                    }
                            }
                            setState(198);
                            _errHandler.sync(this);
                            _alt = getInterpreter().
                                adaptivePredict(_input,14,
                                _ctx);
                    }
                    setState(199);
                    expr(0);
                    }
            }

            setState(202);
            match(RBRACE);
            }
            break;
    case CALLOUT:
            enterOuterAlt(_localctx, 2);
            {
            setState(204);
            match(CALLOUT);
            setState(205);
            match(LBRACE);
            setState(206);
            match(STRING);
            setState(211);
            _errHandler.sync(this);
            _la = _input.LA(1);
            while (_la==COMMA) {
                    {
                    {
                    setState(207);
                    match(COMMA);
                    setState(208);
                    callout_arg();
                    }
                    }
                    setState(213);
                    _errHandler.sync(this);
                    _la = _input.LA(1);
            }
            setState(214);
            match(RBRACE);
```

```
                        }
                        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 Method_nameContext extends ParserRuleContext {
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public Method_nameContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_method_name; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterMethod_name(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitMethod_name(this);
        }
}

public final Method_nameContext method_name() throws
    RecognitionException {
        Method_nameContext _localctx = new Method_nameContext(_ctx,
            getState());
        enterRule(_localctx, 30, RULE_method_name);
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(217);
                match(ID);
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                exitRule();
        }
        return _localctx;
```

```
}

public static class LocationContext extends ParserRuleContext {
        public TerminalNode ID() { return getToken(DecafParser.ID, 0); }
        public TerminalNode LSQUARE() { return getToken(DecafParser.
            LSQUARE, 0); }
        public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
        }
        public TerminalNode RSQUARE() { return getToken(DecafParser.
            RSQUARE, 0); }
        public LocationContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_location; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterLocation(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitLocation(this);
        }
}

public final LocationContext location() throws RecognitionException {
        LocationContext _localctx = new LocationContext(_ctx, getState()
            );
        enterRule(_localctx, 32, RULE_location);
        try {
                setState(225);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,18,_ctx
                    ) ) {
                case 1:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(219);
                        match(ID);
                        }
                        break;
                case 2:
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(220);
                        match(ID);
                        setState(221);
                        match(LSQUARE);
                        setState(222);
                        expr(0);
                        setState(223);
                        match(RSQUARE);
                        }
                        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 TerminalNode MINUS() { return getToken(DecafParser.MINUS,
             0); }
        public List<ExprContext> expr() {
                return getRuleContexts(ExprContext.class);
        }
        public ExprContext expr(int i) {
                return getRuleContext(ExprContext.class,i);
        }
        public TerminalNode NOT() { return getToken(DecafParser.NOT, 0);
             }
        public LocationContext location() {
                return getRuleContext(LocationContext.class,0);
        }
        public Method_callContext method_call() {
                return getRuleContext(Method_callContext.class,0);
        }
        public LiteralContext literal() {
                return getRuleContext(LiteralContext.class,0);
        }
        public TerminalNode LBRACE() { return getToken(DecafParser.
            LBRACE, 0); }
        public TerminalNode RBRACE() { return getToken(DecafParser.
            RBRACE, 0); }
        public TerminalNode MULT() { return getToken(DecafParser.MULT,
            0); }
        public TerminalNode DIV() { return getToken(DecafParser.DIV, 0);
             }
        public TerminalNode MOD() { return getToken(DecafParser.MOD, 0);
             }
        public TerminalNode PLUS() { return getToken(DecafParser.PLUS,
            0); }
        public Rel_opContext rel_op() {
                return getRuleContext(Rel_opContext.class,0);
        }
        public Eq_opContext eq_op() {
                return getRuleContext(Eq_opContext.class,0);
        }
        public TerminalNode AND() { return getToken(DecafParser.AND, 0);
             }
        public TerminalNode OR() { return getToken(DecafParser.OR, 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 DecafParserListener ) ((
                    DecafParserListener)listener).enterExpr(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitExpr(this);
        }
}

public final ExprContext expr() throws RecognitionException {
        return expr(0);
}

private ExprContext expr(int _p) throws RecognitionException {
        ParserRuleContext _parentctx = _ctx;
        int _parentState = getState();
        ExprContext _localctx = new ExprContext(_ctx, _parentState);
        ExprContext _prevctx = _localctx;
        int _startState = 34;
        enterRecursionRule(_localctx, 34, RULE_expr, _p);
        int _la;
        try {
                int _alt;
                enterOuterAlt(_localctx, 1);
                {
                setState(239);
                _errHandler.sync(this);
                switch ( getInterpreter().adaptivePredict(_input,19,_ctx
                    ) ) {
                case 1:
                        {
                        setState(228);
                        match(MINUS);
                        setState(229);
                        expr(12);
                        }
                        break;
                case 2:
                        {
                        setState(230);
                        match(NOT);
                        setState(231);
                        expr(11);
                        }
                        break;
                case 3:
                        {
                        setState(232);
                        location();
                        }
                        break;
                case 4:
```

```
            {
            setState(233);
            method_call();
            }
            break;
case 5:
            {
            setState(234);
            literal();
            }
            break;
case 6:
            {
            setState(235);
            match(LBRACE);
            setState(236);
            expr(0);
            setState(237);
            match(RBRACE);
            }
            break;
}
_ctx.stop = _input.LT(-1);
setState(263);
_errHandler.sync(this);
_alt = getInterpreter().adaptivePredict(_input,21,_ctx);
while ( _alt!=2 && _alt!=org.antlr.v4.runtime.atn.ATN.
    INVALID_ALT_NUMBER ) {
        if ( _alt==1 ) {
                if ( _parseListeners!=null )
                    triggerExitRuleEvent();
                _prevctx = _localctx;
                {
                setState(261);
                _errHandler.sync(this);
                switch ( getInterpreter().
                    adaptivePredict(_input,20,_ctx) ) {
                case 1:
                        {
                        _localctx = new ExprContext(
                            _parentctx, _parentState);
                        pushNewRecursionContext(
                            _localctx, _startState,
                            RULE_expr);
                        setState(241);
                        if (!(precpred(_ctx, 10))) throw
                             new FailedPredicateException
                            (this, "precpred(_ctx, 10)");
                        setState(242);
                        _la = _input.LA(1);
                        if ( !((((_la) & ~0x3f) == 0 &&
                            ((1L << _la) & ((1L << MULT)
                            | (1L << MOD) | (1L << DIV)))
                             != 0)) ) {
                        _errHandler.recoverInline(this);
                        }
                        else {
```

```
                    if ( _input.LA(1)==Token
                        .EOF ) matchedEOF =
                        true;
                    _errHandler.reportMatch(
                        this);
                    consume();
                }
                setState(243);
                expr(11);
                }
                break;
            case 2:
                {
                _localctx = new ExprContext(
                    _parentctx, _parentState);
                pushNewRecursionContext(
                    _localctx, _startState,
                    RULE_expr);
                setState(244);
                if (!(precpred(_ctx, 9))) throw
                    new FailedPredicateException(
                    this, "precpred(_ctx, 9)");
                setState(245);
                _la = _input.LA(1);
                if ( !(_la==PLUS || _la==MINUS)
                    ) {
                _errHandler.recoverInline(this);
                }
                else {
                    if ( _input.LA(1)==Token
                        .EOF ) matchedEOF =
                        true;
                    _errHandler.reportMatch(
                        this);
                    consume();
                }
                setState(246);
                expr(10);
                }
                break;
            case 3:
                {
                _localctx = new ExprContext(
                    _parentctx, _parentState);
                pushNewRecursionContext(
                    _localctx, _startState,
                    RULE_expr);
                setState(247);
                if (!(precpred(_ctx, 8))) throw
                    new FailedPredicateException(
                    this, "precpred(_ctx, 8)");
                {
                setState(248);
                rel_op();
                }
                setState(249);
                expr(9);
```

```
                }
                break;
            case 4:
                {
                _localctx = new ExprContext(
                    _parentctx, _parentState);
                pushNewRecursionContext(
                    _localctx, _startState,
                    RULE_expr);
                setState(251);
                if (!(precpred(_ctx, 7))) throw
                    new FailedPredicateException(
                    this, "precpred(_ctx, 7)");
                {
                setState(252);
                eq_op();
                }
                setState(253);
                expr(8);
                }
                break;
            case 5:
                {
                _localctx = new ExprContext(
                    _parentctx, _parentState);
                pushNewRecursionContext(
                    _localctx, _startState,
                    RULE_expr);
                setState(255);
                if (!(precpred(_ctx, 6))) throw
                    new FailedPredicateException(
                    this, "precpred(_ctx, 6)");
                {
                setState(256);
                match(AND);
                }
                setState(257);
                expr(7);
                }
                break;
            case 6:
                {
                _localctx = new ExprContext(
                    _parentctx, _parentState);
                pushNewRecursionContext(
                    _localctx, _startState,
                    RULE_expr);
                setState(258);
                if (!(precpred(_ctx, 5))) throw
                    new FailedPredicateException(
                    this, "precpred(_ctx, 5)");
                {
                setState(259);
                match(OR);
                }
                setState(260);
                expr(6);
```

```
                                        }
                                        break;
                                }
                                }
                        }
                        setState(265);
                        _errHandler.sync(this);
                        _alt = getInterpreter().adaptivePredict(_input
                            ,21,_ctx);
                }
                }
        }
        catch (RecognitionException re) {
                _localctx.exception = re;
                _errHandler.reportError(this, re);
                _errHandler.recover(this, re);
        }
        finally {
                unrollRecursionContexts(_parentctx);
        }
        return _localctx;
}

public static class Callout_argContext extends ParserRuleContext {
        public ExprContext expr() {
                return getRuleContext(ExprContext.class,0);
        }
        public TerminalNode STRING() { return getToken(DecafParser.
            STRING, 0); }
        public Callout_argContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_callout_arg; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterCallout_arg(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitCallout_arg(this);
        }
}

public final Callout_argContext callout_arg() throws
    RecognitionException {
        Callout_argContext _localctx = new Callout_argContext(_ctx,
            getState());
        enterRule(_localctx, 36, RULE_callout_arg);
        try {
                setState(268);
                _errHandler.sync(this);
                switch (_input.LA(1)) {
                case CALLOUT:
                case FALSE:
```

```
                case TRUE:
                case LBRACE:
                case MINUS:
                case NOT:
                case ID:
                case CHAR:
                case NUMBER:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(266);
                        expr(0);
                        }
                        break;
                case STRING:
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(267);
                        match(STRING);
                        }
                        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 Bin_opContext extends ParserRuleContext {
        public Arith_opContext arith_op() {
                return getRuleContext(Arith_opContext.class,0);
        }
        public Rel_opContext rel_op() {
                return getRuleContext(Rel_opContext.class,0);
        }
        public Eq_opContext eq_op() {
                return getRuleContext(Eq_opContext.class,0);
        }
        public Cond_opContext cond_op() {
                return getRuleContext(Cond_opContext.class,0);
        }
        public Bin_opContext(ParserRuleContext parent, int invokingState
            ) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_bin_op; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterBin_op(this);
```

```
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitBin_op(this);
        }
}

public final Bin_opContext bin_op() throws RecognitionException {
        Bin_opContext _localctx = new Bin_opContext(_ctx, getState());
        enterRule(_localctx, 38, RULE_bin_op);
        try {
                setState(274);
                _errHandler.sync(this);
                switch (_input.LA(1)) {
                case PLUS:
                case MINUS:
                case MULT:
                case MOD:
                case DIV:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(270);
                        arith_op();
                        }
                        break;
                case LT:
                case GT:
                case LTE:
                case GTE:
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(271);
                        rel_op();
                        }
                        break;
                case EQ:
                case NEQ:
                        enterOuterAlt(_localctx, 3);
                        {
                        setState(272);
                        eq_op();
                        }
                        break;
                case AND:
                case OR:
                        enterOuterAlt(_localctx, 4);
                        {
                        setState(273);
                        cond_op();
                        }
                        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 Arith_opContext extends ParserRuleContext {
        public TerminalNode MULT() { return getToken(DecafParser.MULT,
            0); }
        public TerminalNode DIV() { return getToken(DecafParser.DIV, 0);
             }
        public TerminalNode MOD() { return getToken(DecafParser.MOD, 0);
             }
        public TerminalNode PLUS() { return getToken(DecafParser.PLUS,
            0); }
        public TerminalNode MINUS() { return getToken(DecafParser.MINUS,
             0); }
        public Arith_opContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_arith_op; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterArith_op(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitArith_op(this);
        }
}

public final Arith_opContext arith_op() throws RecognitionException {
        Arith_opContext _localctx = new Arith_opContext(_ctx, getState()
            );
        enterRule(_localctx, 40, RULE_arith_op);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(276);
                _la = _input.LA(1);
                if ( !((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L <<
                    PLUS) | (1L << MINUS) | (1L << MULT) | (1L << MOD) |
                    (1L << DIV))) != 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 Rel_opContext extends ParserRuleContext {
        public TerminalNode LT() { return getToken(DecafParser.LT, 0); }
        public TerminalNode LTE() { return getToken(DecafParser.LTE, 0);
             }
        public TerminalNode GTE() { return getToken(DecafParser.GTE, 0);
             }
        public TerminalNode GT() { return getToken(DecafParser.GT, 0); }
        public Rel_opContext(ParserRuleContext parent, int invokingState
            ) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_rel_op; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterRel_op(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitRel_op(this);
        }
}

public final Rel_opContext rel_op() throws RecognitionException {
        Rel_opContext _localctx = new Rel_opContext(_ctx, getState());
        enterRule(_localctx, 42, RULE_rel_op);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(278);
                _la = _input.LA(1);
                if ( !((((_la) & ~0x3f) == 0 && ((1L << _la) & ((1L <<
                    LT) | (1L << GT) | (1L << LTE) | (1L << GTE))) != 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 Eq_opContext extends ParserRuleContext {
        public TerminalNode EQ() { return getToken(DecafParser.EQ, 0); }
        public TerminalNode NEQ() { return getToken(DecafParser.NEQ, 0);
             }
        public Eq_opContext(ParserRuleContext parent, int invokingState)
             {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_eq_op; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterEq_op(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitEq_op(this);
        }
}

public final Eq_opContext eq_op() throws RecognitionException {
        Eq_opContext _localctx = new Eq_opContext(_ctx, getState());
        enterRule(_localctx, 44, RULE_eq_op);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(280);
                _la = _input.LA(1);
                if ( !(_la==EQ || _la==NEQ) ) {
                _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 Cond_opContext extends ParserRuleContext {
        public TerminalNode AND() { return getToken(DecafParser.AND, 0);
             }
        public TerminalNode OR() { return getToken(DecafParser.OR, 0); }
        public Cond_opContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_cond_op; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterCond_op(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitCond_op(this);
        }
}

public final Cond_opContext cond_op() throws RecognitionException {
        Cond_opContext _localctx = new Cond_opContext(_ctx, getState());
        enterRule(_localctx, 46, RULE_cond_op);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(282);
                _la = _input.LA(1);
                if ( !(_la==AND || _la==OR) ) {
                _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 LiteralContext extends ParserRuleContext {
        public TerminalNode NUMBER() { return getToken(DecafParser.
            NUMBER, 0); }
        public TerminalNode CHAR() { return getToken(DecafParser.CHAR,
            0); }
        public Bool_literalContext bool_literal() {
                return getRuleContext(Bool_literalContext.class,0);
        }
        public LiteralContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_literal; }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterLiteral(this);
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitLiteral(this);
        }
}

public final LiteralContext literal() throws RecognitionException {
        LiteralContext _localctx = new LiteralContext(_ctx, getState());
        enterRule(_localctx, 48, RULE_literal);
        try {
                setState(287);
                _errHandler.sync(this);
                switch (_input.LA(1)) {
                case NUMBER:
                        enterOuterAlt(_localctx, 1);
                        {
                        setState(284);
                        match(NUMBER);
                        }
                        break;
                case CHAR:
                        enterOuterAlt(_localctx, 2);
                        {
                        setState(285);
                        match(CHAR);
                        }
                        break;
                case FALSE:
                case TRUE:
                        enterOuterAlt(_localctx, 3);
                        {
                        setState(286);
                        bool_literal();
```

```
                        }
                        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 Bool_literalContext extends ParserRuleContext {
        public TerminalNode TRUE() { return getToken(DecafParser.TRUE,
            0); }
        public TerminalNode FALSE() { return getToken(DecafParser.FALSE,
             0); }
        public Bool_literalContext(ParserRuleContext parent, int
            invokingState) {
                super(parent, invokingState);
        }
        @Override public int getRuleIndex() { return RULE_bool_literal;
            }
        @Override
        public void enterRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).enterBool_literal(this)
                    ;
        }
        @Override
        public void exitRule(ParseTreeListener listener) {
                if ( listener instanceof DecafParserListener ) ((
                    DecafParserListener)listener).exitBool_literal(this);
        }
}

public final Bool_literalContext bool_literal() throws
    RecognitionException {
        Bool_literalContext _localctx = new Bool_literalContext(_ctx,
            getState());
        enterRule(_localctx, 50, RULE_bool_literal);
        int _la;
        try {
                enterOuterAlt(_localctx, 1);
                {
                setState(289);
                _la = _input.LA(1);
                if ( !(_la==FALSE || _la==TRUE) ) {
                _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 boolean sempred(RuleContext _localctx, int ruleIndex, int
    predIndex) {
        switch (ruleIndex) {
        case 17:
                return expr_sempred((ExprContext)_localctx, predIndex);
        }
        return true;
}
private boolean expr_sempred(ExprContext _localctx, int predIndex) {
        switch (predIndex) {
        case 0:
                return precpred(_ctx, 10);
        case 1:
                return precpred(_ctx, 9);
        case 2:
                return precpred(_ctx, 8);
        case 3:
                return precpred(_ctx, 7);
        case 4:
                return precpred(_ctx, 6);
        case 5:
                return precpred(_ctx, 5);
        }
        return true;
}

public static final String _serializedATN =
        "\3\u0430\ud6d1\u8206\uad2d\u4417\uaef1\u8d80\uaadd\3.\u0126
            \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\7\2;\n\2\f
            \2\16\2>\13\2\3\2\7\2A"+
        "\n\2\f\2\16\2D\13\2\3\2\3\2\3\2\3\2\3\3\3\3\3\3\3\3\3\3\3\5\3N\n
            \3\3\4\3\4\3"+
        "\4\3\4\7\4T\n\4\f\4\16\4W
```

```
    \13\4\3\4\3\4\3\5\3\5\3\5\3\5\3\5\3\5\3\5\7\5"+
"b\n\5\f\5\16\5e\13\5\3\5\3\5\3\5\5\5j\n
    \5\3\5\3\5\3\5\3\6\3\6\3\7\3\7"+
"\5\7s\n\7\3\b\3\b\3\t\3\t\7\ty\n\t\f\t\16\t|\13\t\3\t\7\t\177\n
    \t\f\t"+
"\16\t\u0082\13\t\3\t\3\t\3\n\3\n\3\n\3\n\7\n\u008a\n\n\f\n\16\n
    \u008d"+
"\13\n\3\n\3\n\3\13\3\13\3\f\3\f\3\r\3\r\3\r\3\r\3\r\3\r\3\r\3\r
    \3\r\3"+
"\r\3\r\3\r\3\r\3\r\3\r\5\r\u00a4\n\r\3\r\3\r\3\r\3\r\3\r\3\r\3\
    r\3\r\3"+
"\r\3\r\5\r\u00b0\n\r\3\r\3\r\3\r\3\r\3\r\3\r\3\r\5\r\u00b8\n\r
    \3\16\3\16\5"+
"\16\u00bc\n\16\3\17\3\17\3\20\3\20\3\20\3\20\3\20\7\20\u00c5\n
    \20\f\20"+
"\16\20\u00c8\13\20\3\20\5\20\u00cb\n
    \20\3\20\3\20\3\20\3\20\3\20\3\20\3\20"+
"\3\20\7\20\u00d4\n\20\f\20\16\20\u00d7\13\20\3\20\5\20\u00da\n
    \20\3\21"+
"\3\21\3\22\3\22\3\22\3\22\3\22\3\22\5\22\u00e4\n
    \22\3\23\3\23\3\23\3\23"+
"\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\5\23\u00f2\n
    \23\3\23\3\23\3\23\3\23"+
"
    \3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23\3\23
    "+
"\3\23\3\23\3\23\7\23\u0108\n\23\f\23\16\23\u010b
    \13\23\3\24\3\24\5\24"+
"\u010f\n\24\3\25\3\25\3\25\3\25\5\25\u0115\n
    \25\3\26\3\26\3\27\3\27\3"+
"\30\3\30\3\31\3\31\3\32\3\32\3\32\5\32\u0122\n
    \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\13\4\2"+
"\3\3\f\f\3\2\30\31\3\2\34\36\3\2\32\33\3\2\32\36\3\2$
    \'\3\2\"#\3\2 !\4"+
"\2\t\t\16\16\u0135\2\66\3\2\2\2\4M\3\2\2\2\6O\3\2\2\2\bZ
    \3\2\2\2\nn\3"+
"\2\2\2\fr\3\2\2\2\16t\3\2\2\2\20v\3\2\2\2\22\u0085\3\2\2\2\24\
    u0090\3"+
"\2\2\2\26\u0092\3\2\2\2\30\u00b7\3\2\2\2\32\u00bb\3\2\2\2\34\
    u00bd\3\2"+
"\2\2\36\u00d9\3\2\2\2 \u00db\3\2\2\2\"\u00e3\3\2\2\2$\u00f1
    \3\2\2\2&\u010e"+
"\3\2\2\2(\u0114\3\2\2\2*\u0116\3\2\2\2,\u0118\3\2\2\2.\u011a
    \3\2\2\2\60"+
"\u011c\3\2\2\2\62\u0121\3\2\2\2\64\u0123
    \3\2\2\2\66\67\7\6\2\2\678\7)"+
"
    \2\28<\7\20\2\29;\5\6\4\2:9\3\2\2\2;>\3\2\2\2<:\3\2\2\2<=\3\2\2\2=
    B\3"+
"\2\2\2><\3\2\2\2?A\5\b\5\2@?\3\2\2\2AD\3\2\2\2B@\3\2\2\2BC
    \3\2\2\2CE\3"+
"\2\2\2DB\3\2\2\2EF\7\21\2\2FG\7\2\2\3G\3\3\2\2\2HN\7)\2\2IJ\7)
    \2\2JK\7"+
"\24\2\2KL\7.\2\2LN\7\25\2\2MH\3\2\2\2MI\3\2\2\2N\5\3\2\2\2OP
    \5\26\f\2"+
```

```
"PU\5\4\3\2QR\7\26\2\2RT\5\4\3\2SQ\3\2\2\2TW\3\2\2\2US\3\2\2\2UV
    \3\2\2"+
"\2VX\3\2\2\2WU\3\2\2\2XY\7(\2\2Y\7\3\2\2\2Z[\5\f\7\2[\\\5\n
    \6\2\\i\7\22"+
"\2\2]^\5\16\b\2^_\7)\2\2_`\7\26\2\2`b\3\2\2\2a]\3\2\2\2be
    \3\2\2\2ca\3"+
"\2\2\2cd\3\2\2\2df\3\2\2\2ec\3\2\2\2fg\5\16\b\2gh\7)\2\2hj
    \3\2\2\2ic\3"+
"\2\2\2ij\3\2\2\2jk\3\2\2\2kl\7\23\2\2lm\5\20\t\2m\t\3\2\2\2no
    \7)\2\2o"+
"\13\3\2\2\2ps\5\26\f\2qs\7\17\2\2rp\3\2\2\2rq\3\2\2\2s\r
    \3\2\2\2tu\5\26"+
"\f\2u\17\3\2\2\2vz\7\20\2\2wy\5\22\n\2xw\3\2\2\2y|\3\2\2\2zx
    \3\2\2\2z"+
"{\3\2\2\2{\u0080\3\2\2\2|z\3\2\2\2}\177\5\30\r\2~}\3\2\2\2\177\
    u0082\3"+
"\2\2\2\u0080~\3\2\2\2\u0080\u0081\3\2\2\2\u0081\u0083\3\2\2\2\
    u0082\u0080"+
"\3\2\2\2\u0083\u0084\7\21\2\2\u0084\21\3\2\2\2\u0085\u0086
    \5\26\f\2\u0086"+
"\u008b\5\24\13\2\u0087\u0088\7\26\2\2\u0088\u008a\5\24\13\2\
    u0089\u0087"+
"\3\2\2\2\u008a\u008d\3\2\2\2\u008b\u0089\3\2\2\2\u008b\u008c
    \3\2\2\2\u008c"+
"\u008e\3\2\2\2\u008d\u008b\3\2\2\2\u008e\u008f\7(\2\2\u008f
    \23\3\2\2\2"+
"\u0090\u0091\7)\2\2\u0091\25\3\2\2\2\u0092\u0093\t\2\2\2\u0093
    \27\3\2"+
"\2\2\u0094\u0095\5\"\22\2\u0095\u0096\5\32\16\2\u0096\u0097\5$
    \23\2\u0097"+
"\u0098\7(\2\2\u0098\u00b8\3\2\2\2\u0099\u009a\5\36\20\2\u009a\
    u009b\7"+
"(\2\2\u009b\u00b8\3\2\2\2\u009c\u009d\7\13\2\2\u009d\u009e
    \7\22\2\2\u009e"+
"\u009f\5$\23\2\u009f\u00a0\7\23\2\2\u00a0\u00a3\5\20\t\2\u00a1\
    u00a2\7"+
"\b\2\2\u00a2\u00a4\5\20\t\2\u00a3\u00a1\3\2\2\2\u00a3\u00a4
    \3\2\2\2\u00a4"+
"\u00b8\3\2\2\2\u00a5\u00a6\7\n\2\2\u00a6\u00a7\7)\2\2\u00a7\
    u00a8\7\27"+
"\2\2\u00a8\u00a9\5$\23\2\u00a9\u00aa\7\26\2\2\u00aa\u00ab\5$
    \23\2\u00ab"+
"\u00ac\5\20\t\2\u00ac\u00b8\3\2\2\2\u00ad\u00af\7\r\2\2\u00ae\
    u00b0\5"+
"$\23\2\u00af\u00ae\3\2\2\2\u00af\u00b0\3\2\2\2\u00b0\u00b1
    \3\2\2\2\u00b1"+
"\u00b8\7(\2\2\u00b2\u00b3\7\4\2\2\u00b3\u00b8\7(\2\2\u00b4\
    u00b5\7\7\2"+
"\2\u00b5\u00b8\7(\2\2\u00b6\u00b8\5\20\t\2\u00b7\u0094\3\2\2\2\
    u00b7\u0099"+
"\3\2\2\2\u00b7\u009c\3\2\2\2\u00b7\u00a5\3\2\2\2\u00b7\u00ad
    \3\2\2\2\u00b7"+
"\u00b2\3\2\2\2\u00b7\u00b4\3\2\2\2\u00b7\u00b6\3\2\2\2\u00b8
    \31\3\2\2"+
"\2\u00b9\u00bc\7\27\2\2\u00ba\u00bc\5\34\17\2\u00bb\u00b9
    \3\2\2\2\u00bb"+
"\u00ba\3\2\2\2\u00bc\33\3\2\2\2\u00bd\u00be\t\3\2\2\u00be
```

```
    \35\3\2\2\2\u00bf"+
"\u00c0\5 \21\2\u00c0\u00ca\7\22\2\2\u00c1\u00c2\5$\23\2\u00c2\
    u00c3\7"+
"\26\2\2\u00c3\u00c5\3\2\2\2\u00c4\u00c1\3\2\2\2\u00c5\u00c8
    \3\2\2\2\u00c6"+
"\u00c4\3\2\2\2\u00c6\u00c7\3\2\2\2\u00c7\u00c9\3\2\2\2\u00c8\
    u00c6\3\2"+
"\2\2\u00c9\u00cb\5$\23\2\u00ca\u00c6\3\2\2\2\u00ca\u00cb
    \3\2\2\2\u00cb"+
"\u00cc\3\2\2\2\u00cc\u00cd\7\23\2\2\u00cd\u00da\3\2\2\2\u00ce\
    u00cf\7"+
"\5\2\2\u00cf\u00d0\7\22\2\2\u00d0\u00d5\7-\2\2\u00d1\u00d2
    \7\26\2\2\u00d2"+
"\u00d4\5&\24\2\u00d3\u00d1\3\2\2\2\u00d4\u00d7\3\2\2\2\u00d5\
    u00d3\3\2"+
"\2\2\u00d5\u00d6\3\2\2\2\u00d6\u00d8\3\2\2\2\u00d7\u00d5
    \3\2\2\2\u00d8"+
"\u00da\7\23\2\2\u00d9\u00bf\3\2\2\2\u00d9\u00ce\3\2\2\2\u00da
    \37\3\2\2"+
"\2\u00db\u00dc\7)\2\2\u00dc!\3\2\2\2\u00dd\u00e4\7)\2\2\u00de\
    u00df\7"+
")\2\2\u00df\u00e0\7\24\2\2\u00e0\u00e1\5$\23\2\u00e1\u00e2
    \7\25\2\2\u00e2"+
"\u00e4\3\2\2\2\u00e3\u00dd\3\2\2\2\u00e3\u00de\3\2\2\2\u00e4
    #\3\2\2\2"+
"\u00e5\u00e6\b\23\1\2\u00e6\u00e7\7\33\2\2\u00e7\u00f2\5$
    \23\16\u00e8"+
"\u00e9\7\37\2\2\u00e9\u00f2\5$\23\r\u00ea\u00f2\5\"\22\2\u00eb\
    u00f2\5"+
"\36\20\2\u00ec\u00f2\5\62\32\2\u00ed\u00ee\7\22\2\2\u00ee\u00ef
    \5$\23"+
"\2\u00ef\u00f0\7\23\2\2\u00f0\u00f2\3\2\2\2\u00f1\u00e5
    \3\2\2\2\u00f1"+
"\u00e8\3\2\2\2\u00f1\u00ea\3\2\2\2\u00f1\u00eb\3\2\2\2\u00f1\
    u00ec\3\2"+
"\2\2\u00f1\u00ed\3\2\2\2\u00f2\u0109\3\2\2\2\u00f3\u00f4\f\f
    \2\2\u00f4"+
"\u00f5\t\4\2\2\u00f5\u0108\5$\23\r\u00f6\u00f7\f\13\2\2\u00f7\
    u00f8\t"+
"\5\2\2\u00f8\u0108\5$\23\f\u00f9\u00fa\f\n\2\2\u00fa\u00fb
    \5,\27\2\u00fb"+
"\u00fc\5$\23\13\u00fc\u0108\3\2\2\2\u00fd\u00fe\f\t\2\2\u00fe\
    u00ff\5"+
".\30\2\u00ff\u0100\5$\23\n\u0100\u0108\3\2\2\2\u0101\u0102\f\b
    \2\2\u0102"+
"\u0103\7 \2\2\u0103\u0108\5$\23\t\u0104\u0105\f\7\2\2\u0105\
    u0106\7!\2"+
"\2\u0106\u0108\5$\23\b\u0107\u00f3\3\2\2\2\u0107\u00f6\3\2\2\2\
    u0107\u00f9"+
"\3\2\2\2\u0107\u00fd\3\2\2\2\u0107\u0101\3\2\2\2\u0107\u0104
    \3\2\2\2\u0108"+
"\u010b\3\2\2\2\u0109\u0107\3\2\2\2\u0109\u010a\3\2\2\2\u010a
    %\3\2\2\2"+
"\u010b\u0109\3\2\2\2\u010c\u010f\5$\23\2\u010d\u010f\7-\2\2\
    u010e\u010c"+
"\3\2\2\2\u010e\u010d\3\2\2\2\u010f\'\3\2\2\2\u0110\u0115
    \5*\26\2\u0111"+
```

```
            "\u0115\5,\27\2\u0112\u0115\5.\30\2\u0113\u0115\5\60\31\2\u0114\
                u0110\3"+
            "\2\2\2\u0114\u0111\3\2\2\2\u0114\u0112\3\2\2\2\u0114\u0113
                \3\2\2\2\u0115"+
            ")\3\2\2\2\u0116\u0117\t\6\2\2\u0117+\3\2\2\2\u0118\u0119\t
                \7\2\2\u0119"+
            "-\3\2\2\2\u011a\u011b\t\b\2\2\u011b/\3\2\2\2\u011c\u011d\t\t
                \2\2\u011d"+
            "\61\3\2\2\2\u011e\u0122\7.\2\2\u011f\u0122\7,\2\2\u0120\u0122
                \5\64\33"+
            "\2\u0121\u011e\3\2\2\2\u0121\u011f\3\2\2\2\u0121\u0120\3\2\2\2\
                u0122\63"+
            "\3\2\2\2\u0123\u0124\t\n\2\2\u0124\65\3\2\2\2\33<BMUcirz\u0080\
                u008b\u00a3"+
            "\u00af\u00b7\u00bb\u00c6\u00ca\u00d5\u00d9\u00e3\u00f1\u0107\
                u0109\u010e"+
            "\u0114\u0121";
    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)
                        ;
            }
    }
}
```