Padilla Pimentel Carlos Eduardo

Gramática

PROGRAMA → START INSTRUCCIONES END

INSTRUCCIONES → [[INSTRUCCION;]] *

INSTRUCCION →

→ DATA_TYPE IDENTIFICADOR |
→IDENTIFICADOR = EXPRESION |
→WHILE EXPRESION {INSTRUCCIONES} |
→IF CONDICION {INSTRUCCIONES} [[ELSE {INSTRUCCIONES}]]?

DATA_TYPE → INT | STRING | DOUBLE

EXPRESION →

→ IDENTIFICADOR |
→NUMERO |
→ (EXPRESION OPERADOR EXPRESION)

CONDICION → EXPRESION OPERADOR_LOGICO EXPRESION

OPERADOR → + | - | *

OPERADOR_LOGICO → == |! = | < | > | <= | >=

IDENTIFICADOR → CHARS [[CHAR]] *

NUMERO → DIGITO [[DIGITO]] *

CHAR → A | B | C | ... | Z | a | b | c | .... |z

DIGITO → 0 | 1 | ... | 9

Padilla Pimentel Carlos Eduardo

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Semantic {
  private List<Token> tokens;
  private Map<String, String> symbolTable;

  public Semantic(List<Token> tokens) {
    this.tokens = tokens;
    this.symbolTable = new HashMap<>();
  }
  public String analyze() {
    StringBuilder errores = new StringBuilder();
    for (int i = 0; i < tokens.size(); i++) {
      Token currentToken = tokens.get(i);
      switch (currentToken.getTipo()) {
        case INT:
        case BOOLEAN:
        case STRING:
          Declaration(i, currentToken, errores);
          break;
        case IDENTIFICADOR:
          checkVariableUsage(currentToken, errores);
          break;
        case IGUAL:
          checkAssignment(i, errores);
          break;
        default:
```

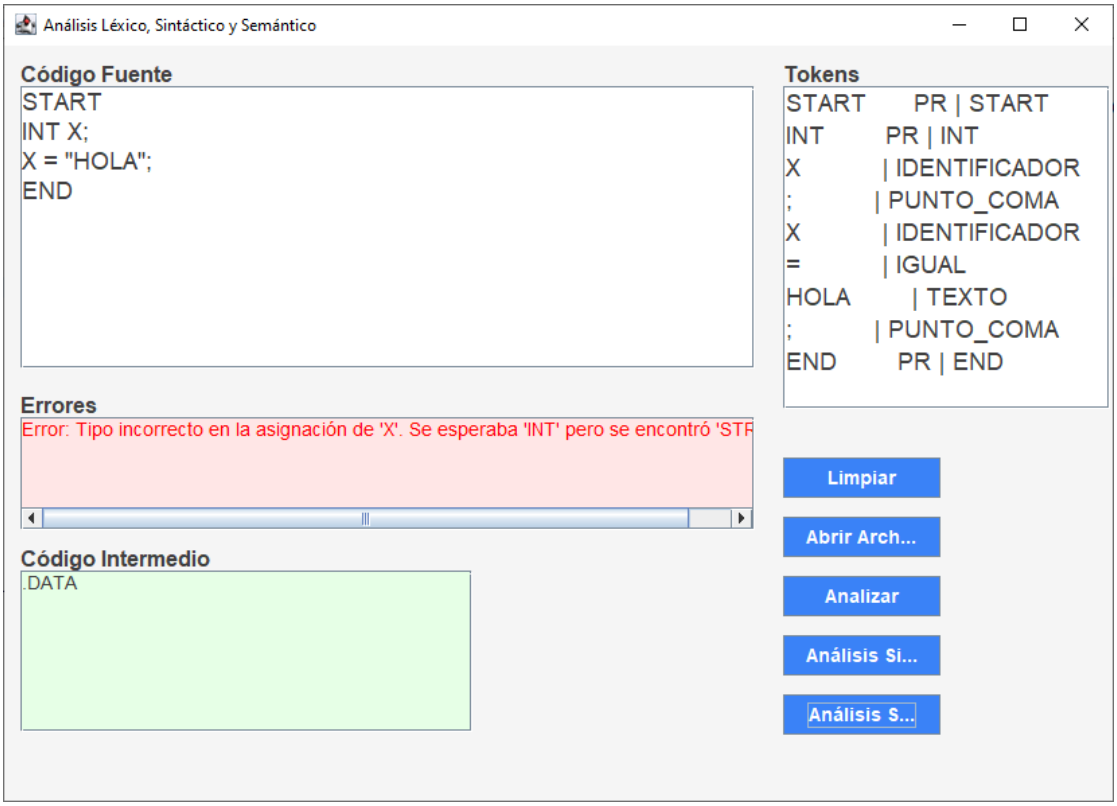
```
    return errores.length() == 0 ? "Análisis semántico completado sin errores." :
errores.toString();
  }
  private void Declaration(int index, Token token, StringBuilder errores) {
    if (index + 1 < tokens.size() && tokens.get(index + 1).getTipo() ==
TokenType.IDENTIFICADOR) {
      String variableName = tokens.get(index + 1).getValor();
      String variableType = token.getValor();
      if (symbolTable.containsKey(variableName)) {
        errores.append("Error: La variable '").append(variableName).append("' ya está
declarada.\n");
      } else {
        symbolTable.put(variableName, variableType);
  private void checkVariableUsage(Token token, StringBuilder errores) {
    String variableName = token.getValor();
    if (!symbolTable.containsKey(variableName)) {
      errores.append("Error: La variable '").append(variableName).append("' no está
declarada.\n");
  private void checkAssignment(int index, StringBuilder errores) {
    if (index > 0 && tokens.get(index - 1).getTipo() == TokenType.IDENTIFICADOR) {
      String variableName = tokens.get(index - 1).getValor();
      if (symbolTable.containsKey(variableName)) {
        String variableType = symbolTable.get(variableName);
        if (index + 1 < tokens.size()) {
          String expressionType = evaluateExpression(index + 1, errores);
          if ("BOOLEAN".equals(variableType)) {
            if (!"INT".equals(expressionType) || !("0".equals(expressionType) ||
"1".equals(expressionType))) {
              errores.append("Error: Asignación incorrecta para la variable '")
                  .append(variableName).append("'. Las asignaciones válidas para BOOLEAN
son '0' y '1'.\n");
```

```
                }
            }
            if ("String".equals(variableType)) {
                if (!"TEXTO".equals(expressionType)) {
                    errores.append("Error: Asignación incorrecta para la variable '")
                            .append(variableName).append("'. Se esperaba un valor de tipo 'String' (entre comillas dobles), pero se encontró '")
                            .append(expressionType).append("'.\n");
                }
            }
            if (!variableType.equals(expressionType)) {
                errores.append("Error: Tipo incorrecto en la asignación de '")
                        .append(variableName).append("'. Se esperaba '").append(variableType)
                        .append("' pero se encontró '").append(expressionType).append("'.\n");
            }
        } else {
            errores.append("Error: Asignación incompleta para la variable '").append(variableName).append("'.\n");
        }
    } else {
        errores.append("Error: La variable '").append(variableName).append("' no está declarada.\n");
    private String evaluateExpression(int index, StringBuilder errores) {
    String expressionType = null;
    boolean expectingOperand = true;
    boolean hasOperand = false;
    for (int i = index; i < tokens.size(); i++) {
        Token token = tokens.get(i);
        switch (token.getTipo()) {
            case NUMERO:
                if (expectingOperand) {
```

```
            if (expressionType == null) {
                expressionType = "INT";
            } else if (!expressionType.equals("INT")) {
                errores.append("Error: Tipo incompatible en la expresión. Se esperaba
'").append(expressionType).append("'.\n");
            }
            expectingOperand = false;
            hasOperand = true;
        } else {
            errores.append("Error: Operador faltante antes de
'").append(token.getValor()).append("'.\n");
        }
        break;
    case IDENTIFICADOR:
        if (expectingOperand) {
            if (symbolTable.containsKey(token.getValor())) {
                String variableType = symbolTable.get(token.getValor());
                if (expressionType == null) {
                    expressionType = variableType;
                } else if (!expressionType.equals(variableType)) {
                    errores.append("Error: Tipo incompatible en la expresión. Se esperaba
'").append(expressionType).append("'.\n");
                }
                expectingOperand = false;
                hasOperand = true;
            } else {
                errores.append("Error: La variable '").append(token.getValor()).append("' no está
declarada.\n");
            }
        } else {
```

```
                errores.append("Error: Operador faltante antes de la variable
'").append(token.getValor()).append("'.\n");
            }
            break;
        case TEXTO:
            if (expectingOperand) {
                    if (expressionType == null) {
                expressionType = "STRING";
            } else if (!expressionType.equals("STRING")) {
                errores.append("Error: Tipo incompatible en la expresión. Se esperaba
'").append(expressionType).append("'.\n");
            }
            expectingOperand = false;
            hasOperand = true;
        } else {
            errores.append("Error: Operador faltante antes de
'").append(token.getValor()).append("'.\n");
            }
            break;
        case OPERADOR:
            if (!expectingOperand) {
                if (token.getValor().equals("+") || token.getValor().equals("-") ||
token.getValor().equals("*")) {
                    expectingOperand = true;
                } else {
                    errores.append("Error: Operador '").append(token.getValor()).append("' no
soportado.\n");
                }
            } else {
                errores.append("Error: Se esperaba un operando antes de
'").append(token.getValor()).append("'.\n");
```

```
            }
            break;
        case PARENTESIS_DERECHO:
        case PUNTO_COMA:
            if (!hasOperand) {
                errores.append("Error: La expresión está vacía.\n");
            }
            return expressionType != null ? expressionType : "UNKNOWN";
        default:
            return expressionType != null ? expressionType : "UNKNOWN";
    }
}
if (expectingOperand) {
    errores.append("Error: La expresión termina con un operador inválido.\n");
}
```



}