Tecnológico de Monterrey

PROFESIONAL

# “ACTIVIDAD INTEGRADORA 2: RESALTADOR DE SINTAXIS Y EJECUCIÓN DE PROGRAMAS DEL LENGUAJE DE MATRICES”

IMPLEMENTACIÓN DE MÉTODOS COMPUTACIONALES


| | |
|---|---|
| Julio Alfredo Gordillo Covarrubias | #A01284202 |
| Fernando José Cabrera Blanco | #A00831380 |
| Hugo Edgar Palomares Estrella | #A01741537 |
| Eduardo Maldonado Guzman | #A00832631 |


MONTERREY, NUEVO LEÓN A 6 DE JUNIO DEL 2022

# INTRODUCCIÓN

El objetivo de nuestro segundo entregable es buscar plasmar lo aprendido durante las semanas conociendo las diferentes funciones, programas y pasos detrás de un compilador al ejecutar algún lenguaje de programación. Utilizando el primer entregable como aprendizaje, nuestro segundo entregable tiene como objetivo crear un nuevo lenguaje de programación llamado lenguaje de matrices, y construir un compilador que verifique la implementación correcta de este mediante un analizador léxico, sintáctico y semántico.

Se buscará utilizar una herramienta en donde se pueda codificar el programa, luego poder ser traducido al lenguaje JAVA y finalmente ser ejecutado para poder obtener la solución al problema ingresado por el usuario. En nuestro trabajo se utilizarán los nuevos métodos y modelos vistos durante el curso, como por ejemplo, la biblioteca de ANTLR para realizar parte de nuestro compilador. Utilizando las nuevas lecciones y reflexionando sobre la retroalimentación obtenida en nuestro primer entregable, se busca realizar la mejor implementación para la creación y verificación de nuestro programa del lenguaje para operaciones de matrices.

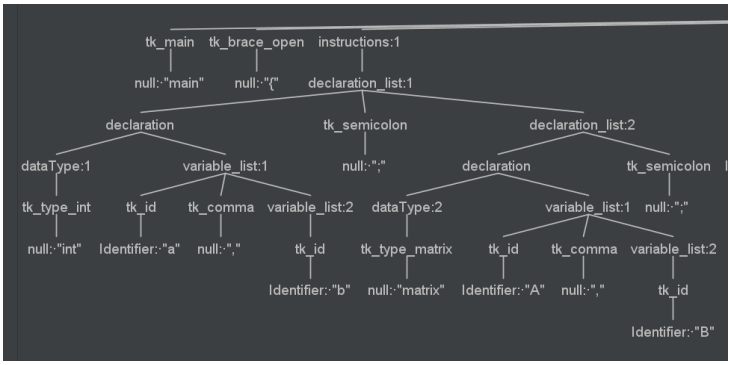
# REPORTE

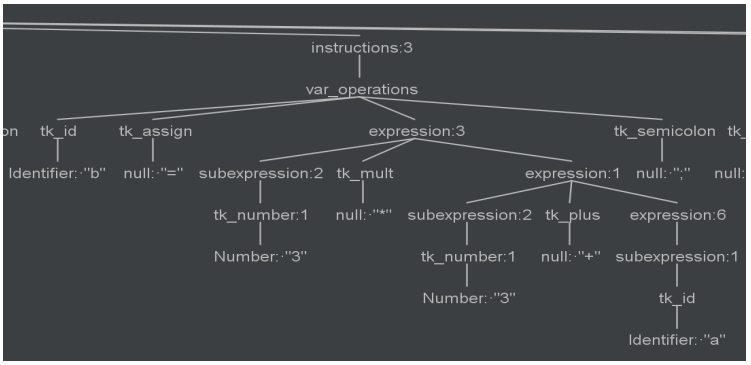
El primer paso para realizar nuestro lenguaje de matrices fue diseñar y escribir nuestra gramática de tipo LL1. Después de crear la gramática, se utilizó la librería de ANTLR para poder realizar los analizadores del compilador. La librería de ANTLR nos ayuda a crear el analizador sintáctico y léxico de nuestra gramática creada. Se tiene que especificar el fichero en donde está escrito la gramática y este genera todos nuestros ficheros de salida que se utilizaran en el programa.

```
grammar MAT;

/*
 * Parser Rules
 */

programDeclaration
      : tk_main tk_brace_open
      instructions *
      tk_brace_close
      ;

instructions
    : declaration_list | print | var_operations | matrix_constructor
    ;

declaration_list
      : declaration tk_semicolon declaration_list
      | declaration tk_semicolon
      ;

declaration
      : dataType variable_list
      ;

variable_list
      : tk_id tk_comma variable_list
      | tk_id
      ;

dataType
      : tk_type_int
      | tk_type_matrix
      ;

print
    : tk_write tk_id tk_semicolon;

var_operations
    : tk_id tk_assign expression tk_semicolon
    ;

expression
    : subexpression tk_plus expression
    | subexpression tk_minus expression
    | subexpression tk_mult expression
    | subexpression tk_div expression
    | subexpression tk_transpos expression
    | subexpression
    ;
```

```
subexpression
    : tk_id | tk_number | tk_par_open expression tk_par_close;

matrix_constructor
    : tk_id tk_assign matrix_values tk_semicolon
    ;

matrix_values
    : tk_bracket_open matrix_numbers* tk_bracket_close
    ;

matrix_numbers
    : Number
    | Number tk_semicolon
    | Number tk_comma matrix_numbers
    ;

/*
 * Tokens
 */

tk_main
      : 'main'
      ;

tk_semicolon
      : ';'
      ;

tk_comma
      : ','
      ;

tk_assign
    : '='
    ;

tk_id
      : Identifier
      ;

tk_number
    : Number | tk_minus Number
    ;

tk_type_int
      : 'int'
      ;

tk_type_matrix
      : 'matrix'
```

Se realizó en un fichero con extensión g4 las reglas para definir la gramática de nuestro lenguaje de matrices, las reglas del perser y las reglas léxicas. Las reglas léxicas se escriben con la primera letra en mayúscula y las reglas del perser con letra minúscula. Los nodos del árbol que se visitará durante el recorrido del árbol de parsing serán las reglas especificadas con la letra minúscula.




Nuestro siguiente paso, fue realizar la implementación de una clase con diferentes métodos y funciones para la implementación del analizador semántico. El analizador semántico sigue después del analizador sintáctico y se explora el árbol de sintaxis abstracta con el fin de detectar los errores semánticos. Se interactúa con la tabla de símbolos creada. Nuestro compilador está diseñado para detectar las variables correctas e incorrectas con las reglas de nuestro lenguaje de matrices.

```
public class MyTranslateListener extends MATBaseListener
{

    private final LinkedList<Variable> variableList = new LinkedList();
    private final LinkedList<Variable> int_variables_list = new LinkedList();
    private final LinkedList<Variable> matrix_variables_list = new LinkedList();
    private final LinkedList<Variable> declared_variables_list = new LinkedList();
    private final LinkedList<String> numberList = new LinkedList();
    private final LinkedList<Matrix> matrixList = new LinkedList();
    private final LinkedList<Matrix> declaredMatrixList = new LinkedList();
    private int counter = 0;


    Variable myvar;
    private String current_dataType_declaration;
```

```
public String getVarType(Variable varToVerify)
{
    for (int i = 0; i < declared_variables_list.size(); i++)
    {
        if (varToVerify.variableName.equals(declared_variables_list.get(i).variableName))
        {
            return declared_variables_list.get(i).variableType;
        }
    }

    try {
        throw new SemanticException(varToVerify.get_token_location() + " variable '"
                + varToVerify.variableName + "' is not defined");
    } catch (SemanticException ex) {
        Logger.getLogger(MyTranslateListener.class.getName()).log(Level.SEVERE, null, ex);
    }
    return "None";
}
```

Por último, en nuestro paso final se tiene como objetivo principal crear la traducción de nuestro lenguaje de matrices al lenguaje de programación JAVA. Para esto se creó una clase en donde se encuentra la personalización del recorrido durante el árbol de parsing. Se crean diferentes “listeners” que se activarán al ocurrir algún evento, es decir algún nodo de interés. Los nodos fueron implementados en el fichero de la gramática y son los que empiezan con letras minúsculas.

```
public class MATBaseListener implements MATListener {
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void enterProgramDeclaration(MATParser.ProgramDeclarationContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void exitProgramDeclaration(MATParser.ProgramDeclarationContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void enterInstructions(MATParser.InstructionsContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
        @Override public void exitInstructions(MATParser.InstructionsContext ctx) { }
        /**
         * {@inheritDoc}
         *
         * <p>The default implementation does nothing.</p>
         */
```

La salida de nuestro programa es el resultado del problema de matrices ingresado por el usuario. En el caso de que haya algún error, se marca en el output y se específica el tipo de error junto con la línea del código en la que se encuentra.

Todo lo explicado anteriormente se realiza por detrás de nuestro programa final. El usuario, tendrá la oportunidad de ingresar cualquier problema desde una interfaz de usuario creada, es decir, una IDE. Realizamos la creación de una pequeña IDE en dónde el usuario es capaz de escribir el problema con las reglas de nuestro lenguaje y recibir la respuesta. En caso de que el problema sea mal escrito o haya algún tipo de error por parte del programador, el IDE es capaz de marcar el error y explicar un poco más sobre qué tipo de error es y en qué parte del código. Si el código está implementado de manera correcta, todos los pasos explicados anteriormente se realizan para compilarlo, traducirlo, realizar la operación y regresar la respuesta. Nuestra IDE está conectada y diseñada en netbeans con el lenguaje de java y es capaz de resaltar cualquier error o sintaxis del lenguaje, cambiando de color. Con nuestra IDE y el compilador detrás, se puede crear el lenguaje de matrices.

## CONCLUSIÓN

En conclusión, el segundo entregable nos ayuda a entender y practicar los nuevos conocimientos aprendidos y las nuevas herramientas vistas en clase para poder facilitar el desarrollo de un compilador. Con el uso de la biblioteca ANTLR, se facilitó la creación y el desarrollo de nuestro analizador sintáctico y léxico. Durante la creación de nuestro entregable, se realizó la implementación y optimización de los algoritmos y modelos computacionales aprendidos durante el curso para realizar nuestra solución al problema dado. Se realizó una extensa investigación y práctica, así como trabajo en equipo para poder encontrar la solución al ejercicio cumpliendo las reglas de nuestro lenguaje. Todo esto nos ayuda a entender de mejor manera el funcionamiento de un compilador y todo el proceso que se realiza al ejecutar un programa de algún lenguaje de programación. El proyecto también nos ayuda a practicar nuestra manera de aprender, investigar y trabajar en equipo. El reto tiene una complejidad alta, lo cual nos ayuda a practicar para nuestros futuros proyectos y mejorar nuestras habilidades como futuros ingenieros.

Nuestro programa es capaz de realizar de manera sencilla diferentes operaciones de matrices ingresando el problema y regresando el resultado en matriz. El lenguaje de matrices puede ser utilizado en la sociedad como herramienta de programación para facilitar la operación matemática de matrices y obtener datos de manera rápida y sencilla. También se puede aprender la manera de realizar un compilador con los métodos computacionales utilizados en nuestro IDE creado y

buscar encontrar mejores soluciones a programas con la optimización de métodos y modelos en la computación.

**Referencias**


- Cesar Garcia. (2022). Conferencia_5. 17-04-22, de TEC Sitio web: https://experiencia21.tec.mx/courses/241488/files/93748136?module_item_id=16301428
- Dr. Liesner Acevedo Martínez. (2011). 3. En Técnicas de Compilación: Manual Práctico para estudiantes de Informática.(209). -: UCI.
- Terence Par. (-). What is ANTLR?. 4-6-22, de ANTLR Sitio web: https://www.antlr.org/
- Cesar Garcia. (2022). ANTLR_ActividadIntegradora. 25-04-22, de TEC Sitio web: ANTLR_Actividad Integradora.mp4 - Google Drive