MANUAL TÉCNICO

Práctica 1 | Organización de Lenguajes y Compiladores 1
Notebook Mobile

Estudiante: Carlos Alfredo López de León
Carnet: 202231051

Índice

Introducción

El presente manual muestra las herramientas y la lógica utilizada para desarrollar una solución de software para una empresa la cual requiere de un programa que ejecute sectores de código y texto individualmente.

La aplicación debe simular el funcionamiento de un Notebook donde se puedan hacer anotaciones sobre los datos que están siendo calculados o graficados, esto por medio de celdas que podrán ser de dos tipos: "Texto" y "Código". Ambas celdas se podrán "Ejecutar" para obtener algún resultado, y las celdas se pueden "Ejecutar" en cualquier orden.

Las celdas de Texto servirán para mostrar solamente texto en formato Markdown. Al ejecutar la celda de Texto al texto ingresado se le dará un formato basado en Markdown ya que estas celdas sirven para hacer anotaciones dentro del Notebook.

Las celdas de Código servirán para realizar asignación de variables así como operaciones matemáticas y la graficación de funciones y ecuaciones, estas funcionalidades se harán en un lenguaje de scripting. Al "Ejecutar" esta celda se realizará la acción del código que hay en la misma, ya sea la declaración de una variable o el despliegue de alguna gráfica o variable. Si es necesario mostrar algo según el código de la celda, este resultado se mostrará debajo de la misma.

Objetivos

General

- Proporcionar al lector una explicación de la lógica, clases y atributos implementados en el desarrollo de la aplicación programada para facilitar el mantenimiento y futuras modificaciones realizadas por terceros.

Específicos

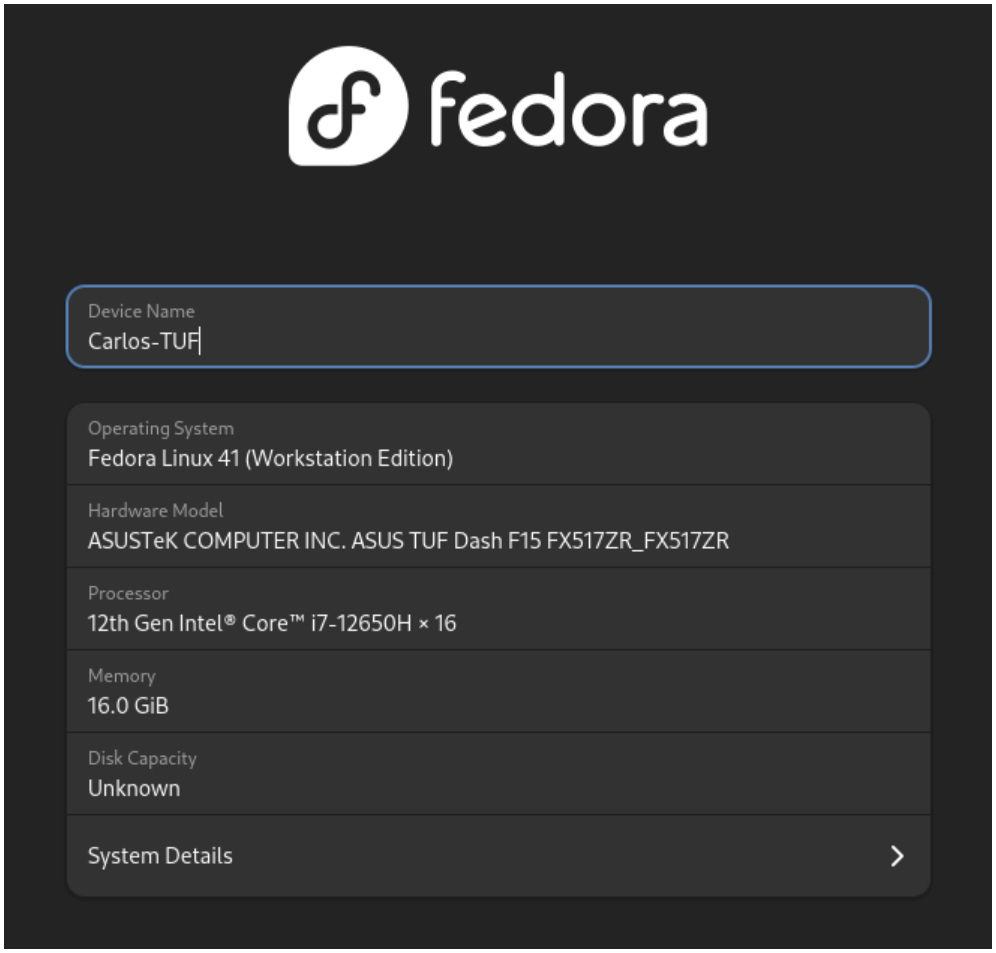
- Exponer una descripción del SO, IDE y otros elementos utilizados durante el desarrollo de la aplicación.
- Otorgar al lector una explicación técnica de los métodos y atributos que componen la parte operativa de la aplicación.
- Proporcionar el conocimiento necesario para el mantenimiento sencillo del software.

Entorno de desarrollo

En el desarrollo de la aplicación fue utilizado el IDE Android Studio, debido a ser un entorno de trabajo completo y que nos proporciona una gran cantidad de funcionalidades, entre ellas la herramienta de Copilot, la cual agiliza el trabajo de desarrollo.

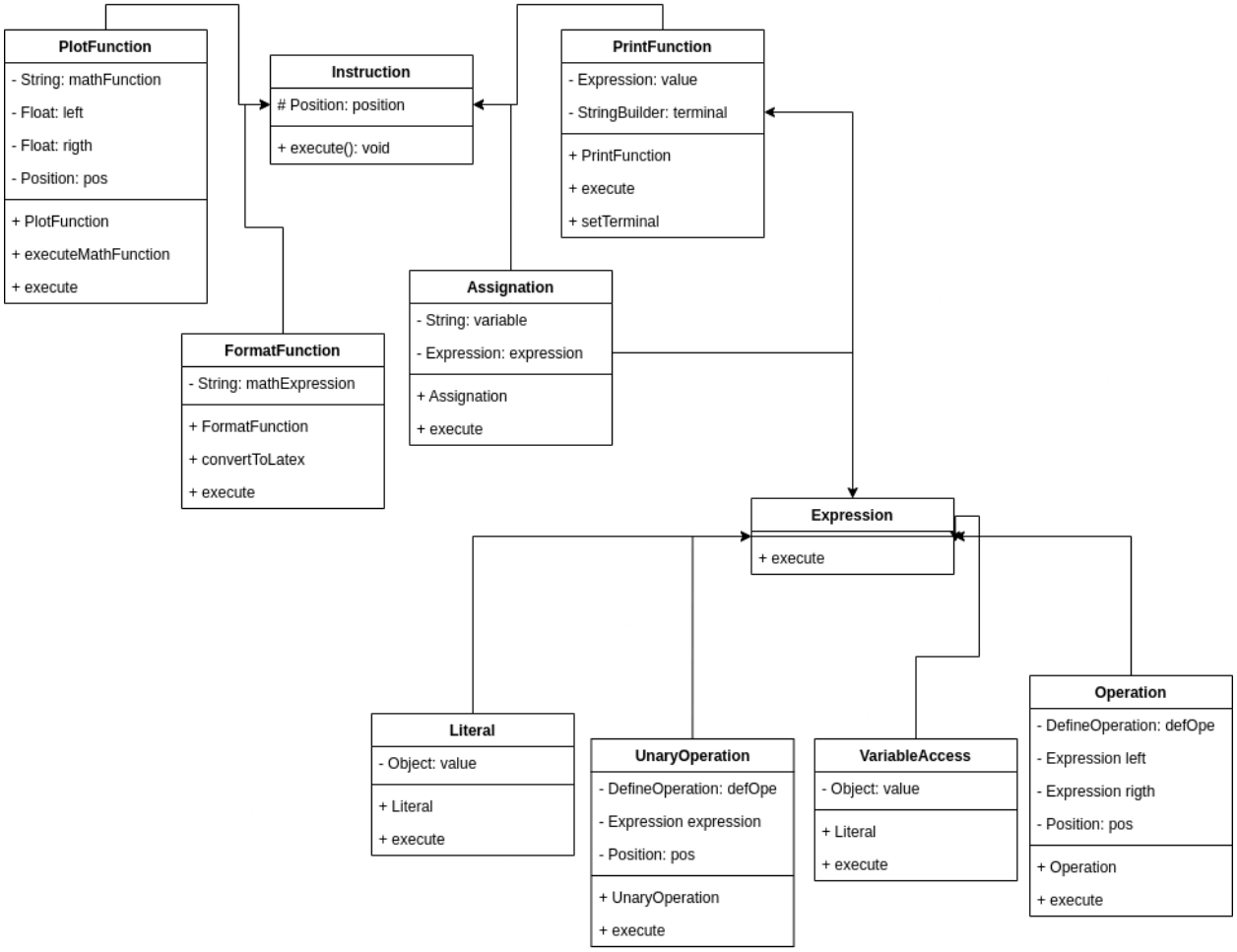
Lenguaje de programación:
Kotlin, Java

Sistema operativo:
Fedora Linux 41



Arquitectura del proyecto

Diagrama de clases



Análisis Léxico para Código

```
/****************** USERS CODE ******************************/
package com.example.notebookmobile.code_analysis;

import java_cup.runtime.*;
import java.util.*;

%%

/*************** Options and declarations *****************/
%public
%unicode
%class CodeLexer
%cup
%line
%column
%debug
%init{
    errorsList = new LinkedList<>();
    string = new StringBuilder();
%init}

/************************** States *********************************/
%state STRING
%state PLOT_FUNCTION
%state FORMAT_FUNCTION
%state MATH_FUNCTION
%state MATH_EXP

/************************** macros *********************************/
LineTerminator = \r|\n|\r\n
InputCharacter = [^\r\n]
```

```
WhiteSpace = {LineTerminator} | [ \t\f]
Identifier = [:jletter:] ([:jletterdigit:]|_)*
SimpleBoolean = 0|1
DecIntegerLiteral = [0-9]+
DecFloatLiteral = {DecIntegerLiteral}\.{DecIntegerLiteral}
StringCharacter = [^\r\n\"\\]
OctDigit          = [0-7]

%{
    StringBuilder string;
  /*--------------------------------------------------------------
    Error handling
  --------------------------------------------------------------*/
    private List<String> errorsList;
    public List<String> symbols = new ArrayList();

    private int parenCount = 0; // Counter for parenthesis

    public List<String> getErrors(){
        if (this.errorsList == null) {
            this.errorsList = new LinkedList<>();
        }
        return this.errorsList;
    }

    /*--------------------------------------------------------------
     Parser code
    --------------------------------------------------------------*/
    private Symbol symbol(int type) {
        symbols.add(yytext());
        return new Symbol(type, yyline+1, yycolumn+1);
```

```
    }

    private Symbol symbol(int type, Object value) {
        symbols.add(value.toString());
        return new Symbol(type, yyline+1, yycolumn+1, value);
    }

    private void error(String message) {
            errorsList.add("Error en la linea: " + (yyline+1) + ", columna: " +
(yycolumn+1) + " : " + message);
    }

%}

%%

/* Lexic Rules */

    /*-------------------------------------------------------------------------
                            Text Marks
    -------------------------------------------------------------------------*/


    /*-------------------------------------------------------------
     *                    Default State
     -------------------------------------------------------------*/
    <YYINITIAL> {

        /* Operators */
        "+"             { return symbol(sym.PLUS); }
        "-"             { return symbol(sym.MINUS); }
        "*"             { return symbol(sym.TIMES); }
```

```
        "/"             { return symbol(sym.DIV); }
        "^"             { return symbol(sym.POWER); }
        "="             { return symbol(sym.EQUALS); }

        /* delimitators */
        "("             { return symbol(sym.LPAREN); }
        ")"             { return symbol(sym.RPAREN); }
        ","             { return symbol(sym.COMMA); }

        /* keywords */
        "print"       { return symbol(sym.PRINT); }
        "format"      { yybegin(FORMAT_FUNCTION); return symbol(sym.FORMAT);
}
        "plot"        { yybegin(PLOT_FUNCTION); return symbol(sym.PLOT); }


        /* literals */
        {Identifier}    { return symbol( sym.ID, yytext() ); }

                        {DecIntegerLiteral}             {  return  symbol(sym.FLOAT_LIT,
Float.valueOf(yytext())); }

                          {DecFloatLiteral}             {  return  symbol(sym.FLOAT_LIT,
Float.valueOf(yytext())); }

        \"              { string.setLength(0); yybegin(STRING); }
    }

    <PLOT_FUNCTION> {
        "("             { yybegin(MATH_FUNCTION); return symbol(sym.LPAREN); }
    }
```

```
<FORMAT_FUNCTION> {
    "("             { yybegin(MATH_EXP); return symbol(sym.LPAREN); }
}

<MATH_EXP> {
    "x"                 { string.append("x"); }
    "+"                 { string.append("+"); }
    "-"                 { string.append("-"); }
    "*"                 { string.append("*"); }
    "/"                 { string.append("/"); }
    "^"                 { string.append("^"); }
      "("                       { string.append("("); parenCount++; } // Counter for
parenthesis to close
    {DecFloatLiteral}       { string.append(yytext()); }
    {DecIntegerLiteral}     { string.append(yytext()); }

    ")" {
      string.append(")");
      if (parenCount > 0) {
        parenCount--; // Close one parenthesis
      } else {
            yybegin(YYINITIAL); // No more parenthesis to close, return to
default state
        string.deleteCharAt(string.length() - 1); // Delete last parenthesis
        return symbol(sym.MATH_EXP, string.toString());
      }
    }
}

<MATH_FUNCTION> {
    "x"                 { string.append("x"); }
    "+"                 { string.append("+"); }
```

```
        "-"                 { string.append("-"); }
        "*"                 { string.append("*"); }
        "/"                 { string.append("/"); }
        "^"                 { string.append("^"); }
        "("                 { string.append("("); }
        ")"                 { string.append(")"); }
        {DecFloatLiteral}       { string.append(yytext()); }
        {DecIntegerLiteral}     { string.append(yytext()); }
                    ","                                         {  yybegin(YYINITIAL);  return
symbol(sym.MATH_FUNCTION, string.toString()); }
    }

    <STRING> {
          \"                        { yybegin(YYINITIAL); return symbol(sym.STRING_LIT,
string.toString()); }

        {StringCharacter}+          { string.append( yytext() ); }

        /* escape sequences */
        "\\b"                       { string.append( '\b' ); }
        "\\t"                       { string.append( '\t' ); }
        "\\n"                       { string.append( '\n' ); }
        "\\f"                       { string.append( '\f' ); }
        "\\r"                       { string.append( '\r' ); }
        "\\\""                      { string.append( '\"' ); }
        "\\'"                       { string.append( '\'' ); }
        "\\\\"                      { string.append( '\\' ); }
                            \\[0-3]?{OctDigit}?{OctDigit}       {   char   val   =   (char)
Integer.parseInt(yytext().substring(1),8);
                                        string.append( val ); }

        /* error cases */
```

```
        \\.                     { error("Secuencia ilegal de escape \""+yytext()+"\""); }
        {LineTerminator}        { error("Literal de cadena sin terminar al final de la línea"); }
    }

    {WhiteSpace}    {/* ignore */}

    /* error fallback */
    .               { error("Simbolo invalido <"+ yytext()+">");}
    <<EOF>>         { return symbol(sym.EOF); }
```

Análisis Sintáctico para Código

```
package com.example.notebookmobile.code_analysis;

import java_cup.runtime.*;
import java.util.*;

import com.example.notebookmobile.code_analysis.instructions.*;
import com.example.notebookmobile.code_analysis.expressions.*;
import com.example.notebookmobile.code_analysis.math_expressions.*;
import com.example.notebookmobile.code_analysis.utils.*;

parser code {:
    // Fields
    CodeLexer lex;
    private List<String> syntaxErrors;
    private Analyzer analyzer;

    // Connect the parser with the lexer
    public CodeParser(CodeLexer lex, Analyzer analyzer){
        super(lex);
        this.analyzer = analyzer;
        this.syntaxErrors = new LinkedList<>();
    }


    // Getters
    public CodeLexer getLexer(){
        return this.lex;
    }
    public List<String> getSyntaxErrors() {
        if (this.syntaxErrors == null) {
            this.syntaxErrors = new LinkedList<>();
        }
        return this.syntaxErrors;
    }

    /* Overwrite error methods */
    public void syntax_error(Symbol cur_token) {
        StringBuilder mssBuilder = new StringBuilder("Simbolo: ");
        mssBuilder.append(symbl_name_from_id(cur_token.sym));

        if(cur_token.value != null){
            mssBuilder.append(", lexema <");
```

```
            mssBuilder.append(cur_token.value.toString());
            mssBuilder.append(">");
        }
        mssBuilder.append(", linea: ");
        mssBuilder.append(cur_token.left);
        mssBuilder.append(", columna: ");
        mssBuilder.append(cur_token.right);

        if (expected_token_ids().isEmpty()) {
            mssBuilder.append(" -- ya no se esperaba ningun simbolo");
        } else {
            mssBuilder.append(" -- Se esperaba [");
            for (Integer expected_token_id : expected_token_ids()) {
                if(!symbl_name_from_id(expected_token_id).equals("error")){
                    mssBuilder.append(symbl_name_from_id(expected_token_id));
                    mssBuilder.append(" ");
                }

            }
            mssBuilder.append("]");
        }
        syntaxErrors.add(mssBuilder.toString());
    }

    public void report_error(String message, Object info){
        try{
            Symbol cur_token = (Symbol) info;
            StringBuilder mssBuilder = new StringBuilder("Simbolo: ");
            mssBuilder.append(symbl_name_from_id(cur_token.sym));
            mssBuilder.append(", linea: ");
            mssBuilder.append(cur_token.left);
            mssBuilder.append(", columna: ");
            mssBuilder.append(cur_token.right);
            if(cur_token != null){
                mssBuilder.append(", Lexema: ");
                mssBuilder.append(cur_token.value);
            }

            if(message != null){
                mssBuilder.append(", Info: ");
                mssBuilder.append(message);
            }

            syntaxErrors.add(mssBuilder.toString());
```

```
        } catch (Exception e){
            syntaxErrors.add(message);
        }
    }


    public void unrecovered_syntax_error(Symbol cur_token) {
        syntaxErrors.add("Errores de sintaxis severos detectados, revisa municiosamente el
codigo");
    }
:}

/*---------------------------------------
            Declarations
-----------------------------------------*/
/* Terminals (tokens returned by the scanner). */
terminal PLUS, MINUS, TIMES, POWER, DIV,
          LPAREN, RPAREN, EQUALS,
          PRINT, PLOT, FORMAT,
          UMINUS, COMMA;

terminal String     ID, STRING_LIT, MATH_FUNCTION, MATH_EXP;
terminal Float    INTEGER_LIT, FLOAT_LIT;


/*---------------------------- Non-terminals ------------------------------------*/

non terminal s;
non terminal List<Instruction>  instructions;

non terminal Expression operable, expression;
non terminal MathExpression math_expression;
non terminal Instruction print_function, plot_function, format_function, assignation, instruction;

/*---------------------------------------------------------------------------
                              Precedences
-----------------------------------------------------------------------------*/
precedence left PLUS, MINUS;
precedence left TIMES, DIV;
precedence left POWER;
precedence right UMINUS;
```

```
/* ----------------------------------
            Grammar
-----------------------------------------*/
start with s;

s ::= instructions:list
        {:
            if (list == null) {
                list = new LinkedList<>();
            }
            analyzer.execute(list);
        :}
    ;

instructions ::= instruction:i
        {:
            List<Instruction> list = new LinkedList<Instruction>();
            list.add(i);
            RESULT = list;
        :}
    | instruction:i instructions:l
        {:
            l.add(0, i);
            RESULT = l;
        :}
      | error
        {:
            RESULT = null;
        :}
    ;

instruction ::= assignation:i
        {: RESULT = i; :}
    | print_function:i
        {: RESULT = i; :}
    | plot_function:i
        {: RESULT = i; :}
    | format_function:i
        {: RESULT = i; :}
    ;

assignation ::= ID:i EQUALS operable:e
        {:
            RESULT = new Assignation(i, e, new Position(ileft, iright));
```

```
        :}
    ;

print_function ::= PRINT:p LPAREN expression:e RPAREN
        {:
            RESULT = new PrintFunction(e, new Position(pleft, pright));
        :}
    ;

plot_function ::= PLOT:p LPAREN MATH_FUNCTION:e FLOAT_LIT:e1 COMMA FLOAT_LIT:e2
RPAREN
        {:

            RESULT = new PlotFunction(e, e1, e2, new Position(pleft, pright));
        :}
    ;

math_expression ::= math_expression:e1 PLUS:o math_expression:e2
            {:
                RESULT = new OperationMathExp(
                    DefineOperation.PLUS,
                    e1, e2,
                    new Position(oleft, oright)
                );
            :}
    | math_expression:e1 MINUS:o math_expression:e2
            {:
                RESULT = new OperationMathExp(
                    DefineOperation.MINUS,
                    e1, e2,
                    new Position(oleft, oright)
                );
            :}
    | math_expression:e1 DIV:o math_expression:e2
            {:
                RESULT = new OperationMathExp(
                    DefineOperation.DIV,
                    e1, e2,
                    new Position(oleft, oright)
                );
            :}
    | math_expression:e1 TIMES:o math_expression:e2
            {:
                RESULT = new OperationMathExp(
```

```
            DefineOperation.TIMES,
            e1, e2,
            new Position(oleft, oright)
        );
    :}
| math_expression:e1 POWER:o math_expression:e2
    {:
        RESULT = new OperationMathExp(
            DefineOperation.POWER,
            e1, e2,
            new Position(oleft, oright)
        );
    :}
| LPAREN
    {:
        RESULT = new LiteralMathExp("(");
    :}
| RPAREN
    {:
        RESULT = new LiteralMathExp(")");
    :}
| PLUS:o math_expression:e
    {:
        RESULT = new UnaryOperationMathExp(
            DefineOperation.PLUS, e, new Position(oleft, oright)
        );
    :}
| MINUS:o math_expression:e
    {:
        RESULT = new UnaryOperationMathExp(
            DefineOperation.MINUS, e, new Position(oleft, oright)
        );
    :}
| INTEGER_LIT:l
    {:
        RESULT = new LiteralMathExp(l);
    :}
| FLOAT_LIT:l
    {:
        RESULT = new LiteralMathExp(l);
    :}
| ID:i
    {:
        RESULT = new VariableMathExp(i, new Position(ileft, iright));
```

```
        :}
    ;

format_function ::= FORMAT:f LPAREN MATH_EXP:s
        {:
            RESULT = new FormatFunction(s, new Position(fleft, fright));
        :}
    ;

expression ::= operable:e
        {: RESULT = e; :}
    | STRING_LIT:l
        {: RESULT = new Literal(l); :}
    ;

operable ::= operable:e1 PLUS:o operable:e2
        {:
            RESULT = new Operation(
                DefineOperation.PLUS,
                e1, e2,
                new Position(oleft, oright)
            );
        :}
    | operable:e1 MINUS:o operable:e2
        {:
            RESULT = new Operation(
                DefineOperation.MINUS,
                e1, e2,
                new Position(oleft, oright)
            );
        :}
    | operable:e1 DIV:o operable:e2
        {:
            RESULT = new Operation(
                DefineOperation.DIV,
                e1, e2,
                new Position(oleft, oright)
            );
        :}
    | operable:e1 TIMES:o operable:e2
        {:
            RESULT = new Operation(
                DefineOperation.TIMES,
                e1, e2,
```

```
            new Position(oleft, oright)
        );
    :}
| operable:e1 POWER:o operable:e2
    {:
        RESULT = new Operation(
            DefineOperation.POWER,
            e1, e2,
            new Position(oleft, oright)
        );
    :}
%prec UMINUS
| LPAREN operable:e RPAREN
    {: RESULT = e; :}
%prec UMINUS
| PLUS:o operable:e
    {:
        RESULT = new UnaryOperation(
            DefineOperation.PLUS, e, new Position(oleft, oright)
        );
    :}
%prec UMINUS
| MINUS:o operable:e
    {:
        RESULT = new UnaryOperation(
            DefineOperation.MINUS, e, new Position(oleft, oright)
        );
    :}
| INTEGER_LIT:l
    {: RESULT = new Literal(l); :}
| FLOAT_LIT:l
    {: RESULT = new Literal(l); :}
| ID:i
    {:
        RESULT = new VariableAccess(i, new Position(ileft, iright)) ;
    :}
;
```

Análisis Léxico para Texto

```
/****************** USERS CODE ******************************/
package com.example.notebookmobile.text_analysis;

import java_cup.runtime.*;
import java.util.*;

%%

/**************** Options and declarations *****************/
%public
%unicode
%class TextLexer
%cup
%line
%column
%init{
   errorsList = new LinkedList<>();
   string = new StringBuilder();
%init}

/***************************** States ***********************************/
%state HEADER6, HEADER5, HEADER4, HEADER3, HEADER2, HEADER1
%state BOLD_ITALIC, BOLD, ITALIC
%state ORDERED_LIST, NOT_ORDERED_LIST
%state STRING_LIT

/***************************** macros ***********************************/
LineTerminator = \r|\n|\r\n
InputCharacter = [^\r\n]

WhiteSpace = {LineTerminator} | [ \t\f]
Identifier = [:jletter:] ([:jletterdigit:]|_)*
SimpleBoolean = 0|1
DecIntegerLiteral = [0-9]+
DecFloatLiteral = {DecIntegerLiteral}\.{DecIntegerLiteral}
OctDigit = [0-7]

%{
   StringBuilder string;

   /*--------------------------------------------------------------
   Error handling
```

```
    ------------------------------------------------------------------*/
    private List<String> errorsList = new LinkedList<>();
    public List<String> symbols = new ArrayList<>();

    public List<String> getErrors() {
        return this.errorsList;
    }

    /*------------------------------------------------------------
     Parser code
    ------------------------------------------------------------*/
    private Symbol symbol(int type) {
        symbols.add(yytext());
        return new Symbol(type, yyline+1, yycolumn+1);
    }

    private Symbol symbol(int type, Object value) {
        symbols.add(value.toString());
        return new Symbol(type, yyline+1, yycolumn+1, value);
    }

    private void error(String message) {
        String errorMessage = "Error en la línea " + (yyline+1) + ", columna " + (yycolumn+1) + " : "
+ message;
        errorsList.add(errorMessage);
        System.out.println(errorMessage);
    }

%}

/* Regular Definitions */

NUMBER = [0-9]+
DOT = \.

%%

/* Lexic Rules */

<YYINITIAL> {
    // Headers
    "###### " { string.setLength(0); yybegin(HEADER6); }
    "##### " { string.setLength(0); yybegin(HEADER5); }
    "#### " { string.setLength(0); yybegin(HEADER4); }
```

```
    "### " { string.setLength(0); yybegin(HEADER3); }
    "## " { string.setLength(0); yybegin(HEADER2); }
    "# " { string.setLength(0); yybegin(HEADER1); }

    // Bold and Italic
    "***" { string.setLength(0); yybegin(BOLD_ITALIC); }
    "**" { string.setLength(0); yybegin(BOLD); }
    "*" { string.setLength(0); yybegin(ITALIC); }

    // Ordered list
    {NUMBER}{DOT} { string.setLength(0); yybegin(ORDERED_LIST); }

    // Not ordered list
    "+" { string.setLength(0); yybegin(NOT_ORDERED_LIST); }

    // Text
    . { string.setLength(0); string.append(yytext()); yybegin(STRING_LIT); }
}

/* Headers */
<HEADER6, HEADER5, HEADER4, HEADER3, HEADER2, HEADER1> {
    [^\r\n]+ { string.append(yytext()); }

    {LineTerminator}? {
        int type =
            yystate() == HEADER6 ? sym.HEADER6 :
            yystate() == HEADER5 ? sym.HEADER5 :
            yystate() == HEADER4 ? sym.HEADER4 :
            yystate() == HEADER3 ? sym.HEADER3 :
            yystate() == HEADER2 ? sym.HEADER2 :
            sym.HEADER1;

        yybegin(YYINITIAL);
        return symbol(type, string.toString().trim()); // Delete extra spaces
    }

    <<EOF>> {
         int type =
             yystate() == HEADER6 ? sym.HEADER6 :
             yystate() == HEADER5 ? sym.HEADER5 :
             yystate() == HEADER4 ? sym.HEADER4 :
             yystate() == HEADER3 ? sym.HEADER3 :
             yystate() == HEADER2 ? sym.HEADER2 :
             sym.HEADER1;
```

```
            yybegin(YYINITIAL);
            return symbol(type, string.toString().trim()); // Delete extra spaces
        }
}

/* Bold, Italic, and Bold-Italic */
<BOLD_ITALIC> {
    "***" { yybegin(YYINITIAL); return symbol(sym.BOLD_ITALIC, string.toString().trim()); }
        [^\r\n] { string.append(yytext()); }
        {LineTerminator} { string.append("\n"); }
}
<BOLD> {
    "**" { yybegin(YYINITIAL); return symbol(sym.BOLD, string.toString().trim()); }
    [^\r\n] { string.append(yytext()); }
    {LineTerminator} { string.append("\n"); }
}
<ITALIC> {
    "*" { yybegin(YYINITIAL); return symbol(sym.ITALIC, string.toString().trim()); }
    [^\r\n] { string.append(yytext()); }
    {LineTerminator} { string.append("\n"); }
}

/* Ordered and Not Ordered Lists */
<ORDERED_LIST> {
    {LineTerminator} { yybegin(YYINITIAL); return symbol(sym.ORDERED_LIST_ITEM,
string.toString()); }
    <<EOF>> { yybegin(YYINITIAL); return symbol(sym.ORDERED_LIST_ITEM,
string.toString()); }
    . { string.append(yytext()); }
}
<NOT_ORDERED_LIST> {
    {LineTerminator} { yybegin(YYINITIAL); return symbol(sym.NOT_ORDERED_LIST_ITEM,
string.toString()); }
     <<EOF>> { yybegin(YYINITIAL); return symbol(sym.NOT_ORDERED_LIST_ITEM,
string.toString()); }
    . { string.append(yytext()); }
}

<STRING_LIT> {
    [^\r\n]+ { string.append(yytext()); }
    {LineTerminator} {
        yybegin(YYINITIAL);
        return symbol(sym.STRING_LIT, string.toString().trim());
```

```
    }
    <<EOF>> {
        yybegin(YYINITIAL);
        return symbol(sym.STRING_LIT, string.toString().trim());
    }
}

/* Ignored whitespace */
{WhiteSpace} { /* Ignore */ }

/* Error handling */
. { error("Símbolo inválido <" + yytext() + ">"); }
<<EOF>> { return symbol(sym.EOF); }
```

Análisis Sintáctico para Texto

```
package com.example.notebookmobile.text_analysis;

import java_cup.runtime.*;
import java.util.*;
import com.example.notebookmobile.text_analysis.elements.*;

parser code {:
    // Fields
    private TextLexer lex;
    private List<String> syntaxErrors;
    private List<Element> elements;

    // Connect the parser with the lexer
    public TextParser(TextLexer lex) {
        super(lex);
        this.lex = lex;
        this.syntaxErrors = new LinkedList<>();
        this.elements = new ArrayList<>();
    }

    // Getters
    public TextLexer getLexer() {
        return this.lex;
    }

    public List<String> getSyntaxErrors() {
        return this.syntaxErrors;
    }

    public List<Element> getElements() {
        return this.elements;
    }
```

```
/* Overwrite error methods */
public void syntax_error(Symbol cur_token) {
    StringBuilder mssBuilder = new StringBuilder("Símbolo: ");
    mssBuilder.append(symbl_name_from_id(cur_token.sym));

    if (cur_token.value != null) {
        mssBuilder.append(", lexema <");
        mssBuilder.append(cur_token.value.toString());
        mssBuilder.append(">");
    }
    mssBuilder.append(", línea: ");
    mssBuilder.append(cur_token.left);
    mssBuilder.append(", columna: ");
    mssBuilder.append(cur_token.right);

    if (expected_token_ids().isEmpty()) {
        mssBuilder.append(" -- No se esperaba ningún símbolo");
    } else {
        mssBuilder.append(" -- Se esperaba [");
        for (Integer expected_token_id : expected_token_ids()) {
            if (!symbl_name_from_id(expected_token_id).equals("error")) {
                mssBuilder.append(symbl_name_from_id(expected_token_id));
                mssBuilder.append(" ");
            }
        }
        mssBuilder.append("]");
    }
    syntaxErrors.add(mssBuilder.toString());
}

public void report_error(String message, Object info) {
    try {
        Symbol cur_token = (Symbol) info;
        StringBuilder mssBuilder = new StringBuilder("Símbolo: ");
```

```
            mssBuilder.append(symbl_name_from_id(cur_token.sym));
            mssBuilder.append(", línea: ");
            mssBuilder.append(cur_token.left);
            mssBuilder.append(", columna: ");
            mssBuilder.append(cur_token.right);
            if (cur_token != null) {
                mssBuilder.append(", Lexema: ");
                mssBuilder.append(cur_token.value);
            }

            if (message != null) {
                mssBuilder.append(", Info: ");
                mssBuilder.append(message);
            }

            syntaxErrors.add(mssBuilder.toString());
        } catch (Exception e) {
            syntaxErrors.add(message);
        }
    }

    public void unrecovered_syntax_error(Symbol cur_token) {
            syntaxErrors.add("Errores de sintaxis severos detectados, revisa minuciosamente el código.");
    }

    // Method to add an element to the list
    public void addElement(Element element) {
        elements.add(element);
    }
:}

/*---------------------------------------
            Declarations
```

```
----------------------------------------*/

/* Tokens */
terminal String STRING_LIT;
terminal String BOLD;
terminal String ITALIC;
terminal String BOLD_ITALIC;
terminal String HEADER1;
terminal String HEADER2;
terminal String HEADER3;
terminal String HEADER4;
terminal String HEADER5;
terminal String HEADER6;
terminal String ORDERED_LIST_ITEM;
terminal String NOT_ORDERED_LIST_ITEM;

/* No terminales */
non terminal List<Element> s, elements;
non terminal Element element, header, list, text;

/*----------------------------------------
            Precedences
----------------------------------------*/


/* ----------------------------------------
             Grammar
----------------------------------------*/

start with s;

s ::= elements:e {:
    for (Element elem : e) {
        parser.addElement(elem);
```

```
    }
    RESULT = e;
:};

elements ::= elements:e element:t {:
    e.add(t);
    RESULT = e;
:}
        | element:t {:
    List<Element> list = new ArrayList<>();
    list.add(t);
    RESULT = list;
:};

element ::= header:h {: RESULT = h; :}
        | list:l {: RESULT = l; :}
        | text:t {: RESULT = t; :};

header ::= HEADER1:h {: RESULT = new Header(1, h); :}
        | HEADER2:h {: RESULT = new Header(2, h); :}
        | HEADER3:h {: RESULT = new Header(3, h); :}
        | HEADER4:h {: RESULT = new Header(4, h); :}
        | HEADER5:h {: RESULT = new Header(5, h); :}
        | HEADER6:h {: RESULT = new Header(6, h); :};

list ::= ORDERED_LIST_ITEM:l {: RESULT = new OrderedList(l); :}
      | NOT_ORDERED_LIST_ITEM:l {: RESULT = new UnorderedList(l); :};

text ::= STRING_LIT:t {: RESULT = new StringLit(t); :}
      | BOLD:t {: RESULT = new Bold(t); :}
      | ITALIC:t {: RESULT = new Italic(t); :}
      | BOLD_ITALIC:t {: RESULT = new BoldItalic(t); :};
```

Librerías utilizadas

Las librerías utilizadas en el proyecto son las siguientes:

- JFlex y CUP | Utilizados para el análisis léxico, sintáctico y semántico
- exp4j | Utilizado para la operación de funciones matemáticas a base de un String recibido
- jlatexmath | Utilizado para trabajar con funciones matemáticas y mostrar texto en formato JLaTeX

Descripción de la solución

Para poder desarrollar este proyecto, se analizó lo solicitado por el cliente, sus restricciones, ambiente y forma de trabajo.

Entre estas, las principales son:

- El usuario selecciona el tipo de casilla que necesita para ejecutar, siendo estas de Texto o de Código.
- En la casilla seleccionada, ingresara texto que será reconocido por el analizador para texto o ingresará código que será reconocido por el analizador de código, esto en su respectiva casilla.
- Podrá ejecutar casillas de manera independiente.
- Al ejecutar se mostrará el resultado de dicho input si no se reconocen errores, de lo contrario se mostrarán los errores reconocidos para su pronta corrección.

Lógica de la solución

La lógica utilizada para el desarrollo de la aplicación es la siguiente:

Al ejecutar el archivo, este es analizado por el analizador correspondiente con el fin de hacer los análisis

1. Léxico
2. Sintáctico y Semántico

El lexer en conjunto con el cup, analizarán el contenido ingresado, siendo que reconocerán los tokens y el orden de los mismos. Una vez sean reconocidos correctamente se procederá a la lógica individual en cada una de las funcionalidades solicitadas por el usuario.

Solo los archivos que no contengan errores podrán ser ejecutados, donde se mostrará la información deseada.