Tecnológico de Estudios Superiores de Ecatepec

División de Ingeniería en Sistemas Computacionales

Academia en Ciencias de la Ingeniería

Materia: Lenguajes y Autómatas II

Grupo 5602

Alumno

Campero Granados Luis Daniel

Profesor

Hernández Rojas Rosa María

Proyecto tercer parcial

Índice

1. Introducción

Un compilador es un tipo de traductor que transforma un programa entero de un lenguaje de programación a otro. Usualmente el lenguaje objetivo es código máquina, aunque también puede ser traducido a un código intermedio o a texto.

2. Objetivo general

Compilar código mediante el desarrollo de un programa en el lenguaje de programación JAVA.

3. Objetivos específicos

- Elaborar un compilador capaz de leer variables y errores.
- Utilizar los métodos aprendidos, para el desarrollo del programa.

4. Competencias para desarrollar

- Conocer herramientas que faciliten la creación del autómata y programa.
- Capacidad de análisis y síntesis
- Solución de problemas
- Adaptabilidad en entornos de trabajo
- Planificación y organización

5. Desarrollo

5.1. Código

```
public class Compilador extends javax.swing.JFrame {

    String[] vecCodigo = new String[100];
    String[] vecCodigo1 = new String[100];
    String[] vecTokens1 = new String[100];
    String[] vecTokens2 = new String[100];
    String[] tipoDato = {"int", "boolean", "double", "String"};
    String[] operador = {"+", "-", "*", "/", "%", "="};

    String[][] TErrores = new String[100][2];
    String[][] TVar = new String[100][4];

    int contErorres = 0;
    int contVar = 0;
    int contLineas = 1;
    int contTokens = 1;
    int contLineaTabla = 1;
    int NoInt;

    double numdouble;

    boolean FlagSimboloOp = false;
    boolean FlagVar = false;
    boolean FlagOperador = false;
    boolean FlagConstante = false;
    boolean FlagError = false;
    boolean FlagWhile = false;
    boolean FlagEndWhile = false;
    boolean VarBool;

    String codigo;
    String varString;
    String tipoVar = "";
    String txtCodigo = "Codigo.Txt";

    DefaultTableModel tablaVariables;
    DefaultTableModel tablaError;

    public Compilador() {
        initComponents();
        tablaVariables = (DefaultTableModel) tablaVar.getModel();
        tablaError = (DefaultTableModel) tablaErrores.getModel();
        try{
            FileInputStream obj1 = new FileInputStream(txtCodigo);
            InputStreamReader obj2 = new InputStreamReader(obj1,"utf8");
            BufferedReader obj3 = new BufferedReader(obj2);
            String lineaCodigo;
            BufferedReader lectura;
            try{
                lectura = new BufferedReader(new FileReader(txtCodigo));
                while (lectura.ready()) {
                    codigo = lectura.readLine();

txtAreaCodigo.setText(txtAreaCodigo.getText() + contLineas + ":     " +  codigo + "\n");
```

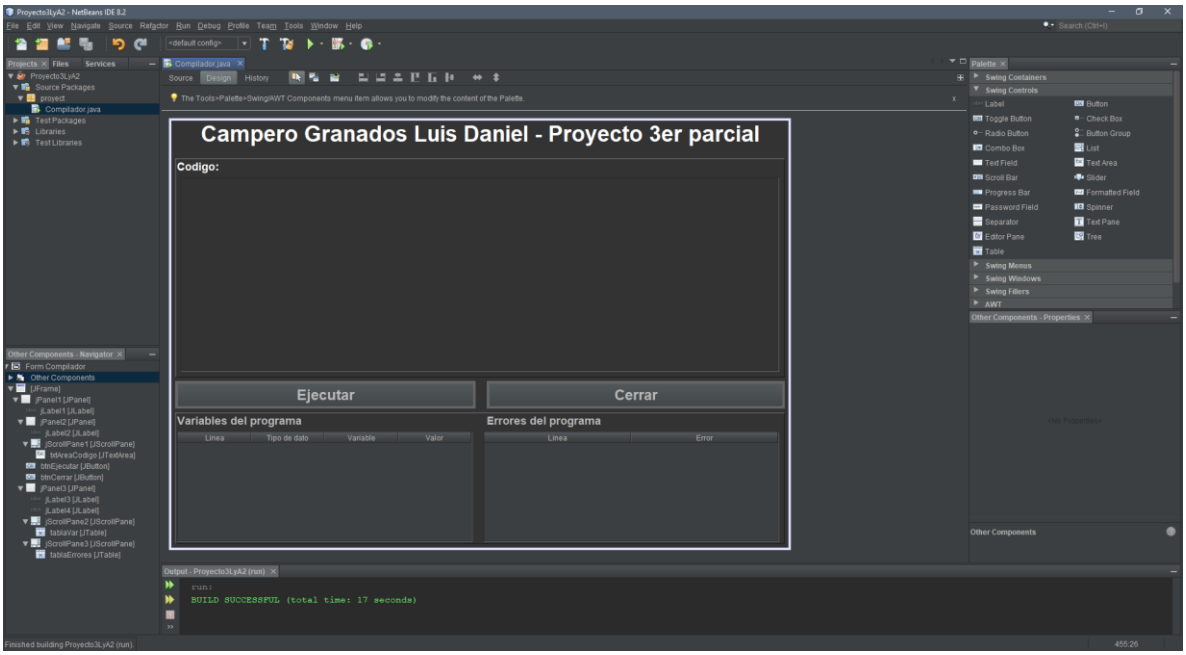
```
            contLineas++;
        }
    }catch (IOException e){}
    contLineas = 0;

    while((lineaCodigo = obj3.readLine()) != null){
        codigo = lineaCodigo;
        contLineas++;
        for (int i = 0; i < codigo.length(); i++){
            vecCodigo[i] = "" + codigo.charAt(i);
        }
        for (int i = 0; i < codigo.length(); i++){
            if (" ".equals(vecCodigo[i])) contTokens++;
        }
        for (int i = 0; i < contTokens; i++){
            vecTokens1[i] = "";
        }
        contTokens = 0;
        for (int i = 0; i < codigo.length(); i++){
            if (!" ".equals(vecCodigo[i])){
                if (";".equals(vecCodigo[i+1])){
                    vecTokens1[contTokens] = "" + vecTokens1[contTokens] + vecCodigo[i];
                    contTokens++;
                    vecTokens1[contTokens] = "" + vecCodigo[i+1];
                    i++;
                }else{
                    vecTokens1[contTokens] = "" + vecTokens1[contTokens] + vecCodigo[i];
                }
            }else{
                contTokens++;
            }
        }

        for (int i = 0; (i < contTokens+1 && !";".equals(vecTokens1[i])); i++){
            FlagVar = false;
            FlagOperador = false;
            FlagConstante = false;
            for (int x = 0; (x < tipoDato.length && FlagSimboloOp == false); x++){
                if (vecTokens1[i].equals(tipoDato[x])){
                    TVar[contVar][0] = "" + contLineas;
                    TVar[contVar][1] = vecTokens1[0];
                    TVar[contVar][2] = vecTokens1[1];
                    if (!";".equals(vecTokens1[2])){
                        TVar[contVar][3] = vecTokens1[3];
                    }
                    contVar++;
                    FlagSimboloOp = true;
                }
                if (!";".equals(vecTokens1[2])) {
                    if ("int".equals(vecTokens1[0]) && (vecTokens1[3] != null || !"".equals(vecTokens1[3])) && FlagError == false){
                        try{
                            NoInt = Integer.parseInt(vecTokens1[3]);

}catch(NumberFormatException e){
                            TErrores[contErorres][0] = "" + contLineas;
```

```
                        TErrores[contErorres][1] =
"La variable de tipo (int) no acepta el valor
ingresado";

                        contErorres++;

                        FlagError = true;

                    }

                }

                if
("boolean".equals(vecTokens1[0]) &&
(vecTokens1[3] != null || !"".equals(vecTokens1[3]))
&& FlagError == false){

                    if
(!"true".equals(vecTokens1[3]) &&
!"false".equals(vecTokens1[3])){

                        TErrores[contErorres][0] = ""
+ contLineas;

                        TErrores[contErorres][1] =
"La variable de tipo (boolean) no acepta el valor
ingresado";

                        contErorres++;

                        FlagError = true;

                    }

                }

                if
("double".equals(vecTokens1[0]) &&
(vecTokens1[3] != null || !"".equals(vecTokens1[3]))
&& FlagError == false){

                    try{

                        numdouble =
Double.parseDouble(vecTokens1[3]);


}catch(NumberFormatException e){

                        TErrores[contErorres][0] = ""
+ contLineas;

                        TErrores[contErorres][1] =
"La variable de tipo (double) no acepta el valor
ingresado";

                        contErorres++;

                        FlagError = true;

                    }

                }

                if ("String".equals(vecTokens1[0])
&& (vecTokens1[3] != null ||
!"".equals(vecTokens1[3])) && FlagError == false){

                    if ( vecTokens1[3].charAt(0) !=
"" || vecTokens1[3].charAt(vecTokens1[3].length()-
1) != "") {

                        TErrores[contErorres][0] = ""
+ contLineas;

                        TErrores[contErorres][1] =
"La variable de tipo (String) no acepta el valor
ingresado";

                        contErorres++;

                        FlagError = true;

                    }

                }

            }

            if ("WHILE".equals(vecTokens1[0])) {

                FlagWhile = true;

                if (vecTokens1[1].charAt(0) != '(' ||
vecTokens1[contTokens].charAt(vecTokens1[cont
Tokens].length()-1) != ')') {

                    TErrores[contErorres][0] = "" +
contLineas;

                    TErrores[contErorres][1] = "Error
en la declaracion del WHILE";

                    contErorres++;

                }

            }

            if
("ENDWHILE".equals(vecTokens1[0])) {

                if (FlagWhile == false) {

                    TErrores[contErorres][0] = "" +
contLineas;

                    TErrores[contErorres][1] = "Error
en la declaracion del WHILE";

                    contErorres++;

                } else FlagWhile = false;

            }
```

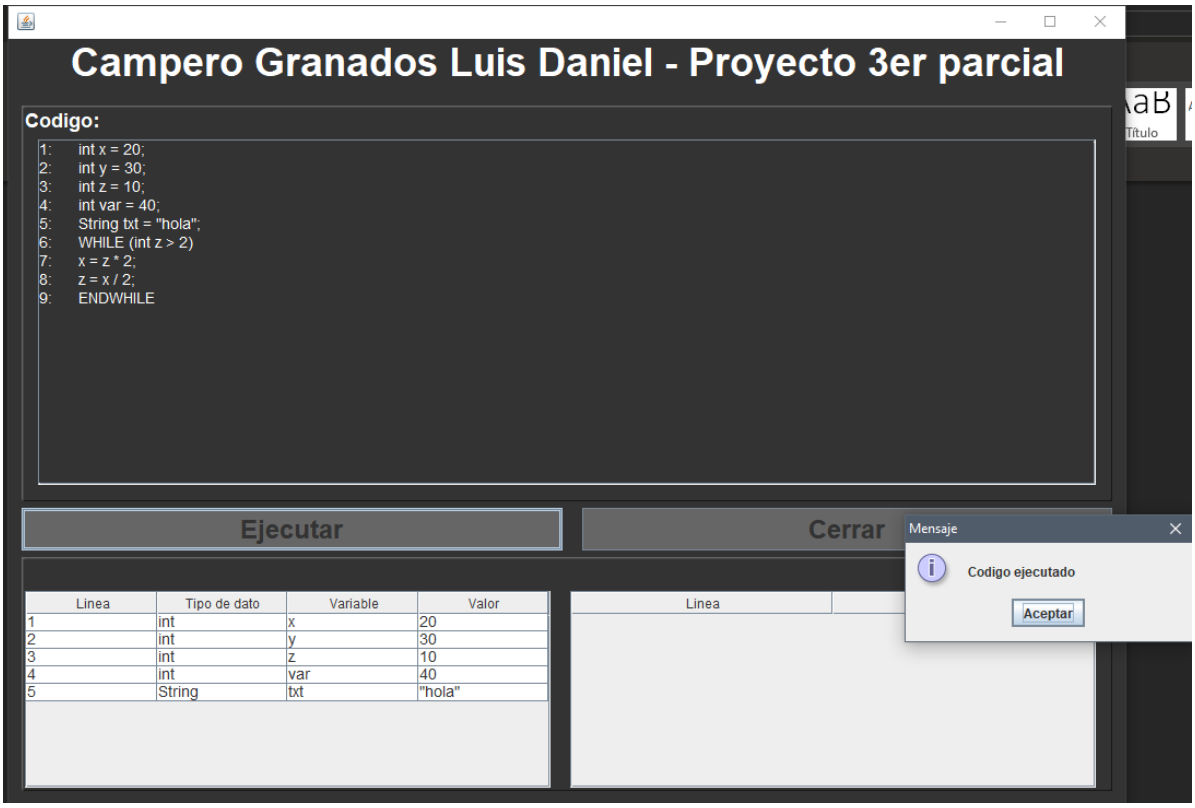
```
                for (int x = 0; (x < TVar.length && FlagSimboloOp == false); x++){

                    if (vecTokens1[i].equals(TVar[x][2])) FlagVar = true;

                }

                for (int x = 0; (x < operador.length && FlagSimboloOp == false); x++){

                    if (vecTokens1[i].equals(operador[x])) FlagOperador = true;

                }

                try{

                    NoInt = Integer.parseInt(vecTokens1[i]);

                    FlagConstante = true;

                }catch(NumberFormatException e){}

                try{

                    numdouble = Double.parseDouble(vecTokens1[i]);

                    FlagConstante = true;

                }catch(NumberFormatException e){}

                if (vecTokens1[i].charAt(0) == "" && vecTokens1[i].charAt(vecTokens1[i].length()-1) == ""){

                    try{

                        varString = String.valueOf((vecTokens1[i]));

                        FlagConstante = true;

                    }catch(NumberFormatException e){}

                }

                if (FlagSimboloOp == false && !"WHILE".equals(vecTokens1[0]) && !"ENDWHILE".equals(vecTokens1[0])){

                    if (FlagVar == false && FlagOperador == false && FlagConstante == false){

                        TErrores[contErorres][0] = "" + contLineas;

                        TErrores[contErorres][1] = "Variable " + vecTokens1[i] + " no declarada";

                        contErorres++;

                        FlagError = true;

                    }

                }

                for (int x = 0; (x < TVar.length && FlagSimboloOp == false); x++){

                    if (vecTokens1[i].equals(TVar[x][2])){

                        if (!"".equals(tipoVar) && !TVar[x][1].equals(tipoVar)){

                            TErrores[contErorres][0] = "" + contLineas;

                            TErrores[contErorres][1] = "Tipo de variables incompatible";

                            contErorres++;

                            FlagError = true;

                        }

                        tipoVar = TVar[x][1];

                    }

                }

            }

            if (!";".equals(vecTokens1[contTokens]) && !"WHILE".equals(vecTokens1[0]) && !"ENDWHILE".equals(vecTokens1[0])){

                TErrores[contErorres][0] = "" + contLineas;

                TErrores[contErorres][1] = "Error en la linea de codigo";

                contErorres++;

            }

            FlagSimboloOp = false;

            FlagVar = false;

            FlagOperador = false;

            FlagConstante = false;

            FlagError = false;

            tipoVar = "";
```

```
            }

            if (FlagWhile == true){

                TErrores[contErorres][0] = "" + contLineas;

                TErrores[contErorres][1] = "Error en la declaracion del WHILE";

                contErorres++;

            }

            obj1.close();

        } catch(IOException e) {}



    }

private void btnEjecutarActionPerformed(java.awt.event.ActionEvent evt) {

        int tablaVarFila = tablaVariables.getRowCount();

        for (int i = 1; i <= tablaVarFila; i++){

            tablaVariables.removeRow(0);

        }

        for (int i = 0; i <contVar; i++){

            tablaVariables.addRow(new Object[]{TVar[i][0], TVar[i][1], TVar[i][2],

                TVar[i][3]});

        }

        int tablaErrorFila = tablaError.getRowCount();

        for (int i = 1; i <= tablaErrorFila; i++){

            tablaError.removeRow(0);

        }

        for (int i = 0; i <contErorres; i++){

            tablaError.addRow(new Object[]{TErrores[i][0], TErrores[i][1]});

        }


        try{

            File archivo2 = new File("Errores.txt");

            FileWriter archi =new FileWriter(archivo2,false);



            archi.write("Linea  ");

            archi.write("Error  \r\n");


            for (int q = 0; q <contErorres; q++){

            archi.write(TErrores[q][0]+"     ");

            archi.write(TErrores[q][1]+"  \r\n");

            }



JOptionPane.showMessageDialog(null,"Codigo ejecutado" );

            archi.close();

        }

        catch(Exception e)

        {

            System.out.println("Error al escribir");



JOptionPane.showMessageDialog(null,"Error al compilar el codigo" );

        }

    }


    private void btnCerrarActionPerformed(java.awt.event.ActionEvent evt) {

        System.exit(0);

    }
```

5.2. Capturas


Campero Granados Luis Daniel - Proyecto 3er parcial
Codigo:
Ejecutar
Cerrar
Variables del programa
Linea
Tipo de dato
Variable
Valor
Errores del programa
Linea
Error
BUILD SUCCESSFUL (total time: 17 seconds)


Campero Granados Luis Daniel - Proyecto 3er parcial
Codigo:
1: int x = 20;
2: int y = 30;
3: int z = 10;
4: int var = 40;
5: String txt = "hola";
6: WHILE (int z > 2)
7: x = z * 2;
8: z = x / 2;
9: ENDWHILE
Ejecutar
Cerrar
Linea
Tipo de dato
Variable
Valor
Linea
Error


Campero Granados Luis Daniel - Proyecto 3er parcial
Codigo:
1: int x = 20;
2: int y = 30;
3: int z = 10;
4: int var = 40;
5: String txt = "hola";
6: WHILE (int z > 2)
7: x = z * 2;
8: z = x / 2;
9: ENDWHILE
Ejecutar
Cerrar
Mensaje
Codigo ejecutado
Aceptar
Linea
Tipo de dato
Variable
Valor
1 int x 20
2 int y 30
3 int z 10
4 int var 40
5 String txt "hola"
Linea


Campero Granados Luis Daniel - Proyecto 3er parcial
Codigo:
1: int x = 20;
2: int y = 30;
3: int z = 10;
4: int var = 40;
5: String txt = "hola"
6: WHILE (int z > 2)
7: x = z * 2;
8: z = x / 2;
9: ENDWHILE
Ejecutar
Cerrar
Linea
Tipo de dato
Variable
Valor
1 int x 20
2 int y 30
3 int z 10
4 int var 40
5 String txt "hola"
Linea
Error
5 Error en la linea de codigo

6. Conclusión

La elaboración de este programa se me complico bastante ya que no tenia muchos conocimientos, pero sobre todo para crear las demás funciones por ejemplo sentencias correctas para las condiciones es más código se vuelve repetitivo y tedioso pero se tendrá que hacer lo mismo para las otras estructuras condicionales, en este programa solamente mostré 1 pero existen más.

7. Referencias

https://www.lawebdelprogramador.com/diccionario/Flag/

https://www.youtube.com/watch?v=3RIeADfiE54

https://www.youtube.com/watch?v=PMlyNM2-B2w

https://www.youtube.com/watch?v=b6kRvD6l7oo

8. Link del video

https://www.youtube.com/watch?v=Fdl7p1Z4plk