```java
package Gramatica;

import Lectura.LeerArchivo_1;
import Lectura.NoEsDeString;

public class AcomodoGramatica {

    private String[] gramaticaCompleta;
    private String[][] todasLasProducciones;
    private String[] ladosDerechos;
    public String[] simbolosNoTerminales;
    public String[] simbolosTerminales;
    private String[] temp;
    private final LeerArchivo_1 leerA;
    private final NoEsDeString nes;

    public AcomodoGramatica() {
        leerA = new LeerArchivo_1();
        nes = new NoEsDeString();
    }

    private void leerGramatica() {
        leerA.leerArchivo();
    }

    public void ini() {
        leerGramatica();
        separarProducciones();
        definirArreglos();
        todasLasProducciones();
        ladosDerechos();
        simbolosNoTerminales();
        simbolosTerminales();
        //System.out.println(nes.cantidadDeCadenas());
        //System.out.println(nes.NoEsTrim(gramatica));
    }

    private void definirArreglos() {
        todasLasProducciones = new String[nes.cantidadDeCadenas()]
[nes.cantidadDeCadenas()];
        ladosDerechos = new String[nes.cantidadDeCadenas()];
        simbolosNoTerminales = new String[nes.cantidadDeCadenas()];
        simbolosTerminales = new String[nes.cantidadDeCadenas()+30];
        temp = new String[3];
    }

    private void todasLasProducciones() {
        for (int i = 0; i < gramaticaCompleta.length-1; i++) {
            temp = nes.noEsEsplit(gramaticaCompleta[i], ">");
            todasLasProducciones[i][0] = (temp[0]).trim(); // lado izquierdo
            todasLasProducciones[i][1] = (temp[1]).trim(); // lado derecho
        }
    }

    private void ladosDerechos() {
        for (int i = 0; i < todasLasProducciones.length; i++) {
            ladosDerechos[i] = todasLasProducciones[i][1];
        }
    }

    private void simbolosNoTerminales() {
        int i = 0, j = 0;
        while (j < todasLasProducciones.length) {
            if (!existeEnArreglo(simbolosNoTerminales,
todasLasProducciones[j][0])) {
                simbolosNoTerminales[i] = todasLasProducciones[j][0].trim();
                i++;
            }
            j++;
        }
    }

    private void simbolosTerminales() {
        int i = 0, j = 0;
//            imprime(simbolosNoTerminales);
        while (j < todasLasProducciones.length) {
            temp = nes.noEsEsplit(todasLasProducciones[j][1], " ");
            for (String tempDerecha : temp) {           //  949 - vacio - 'ε'
                if (!existeEnArreglo(simbolosNoTerminales, tempDerecha)&&
((int)tempDerecha.charAt(0) != 949)
                        && !existeEnArreglo(simbolosTerminales, tempDerecha))
{
                    simbolosTerminales[i] =
tempDerecha;//todasLasProducciones[j][1];
                    i++;
                }
            }
            j++;
        }
    }

    private boolean existeEnArreglo(String[] arreglo, String buscar) {
        for (String buscarEn : arreglo) {
            if (buscarEn != null)
                if (buscarEn.equals(buscar))
                    return true;
        }
        return false;
    }

    public int indiceNoTerminal(String buscar) {
        for (int i = 0; i < simbolosNoTerminales.length; i++) {
            if (simbolosNoTerminales[i] != null)
//                System.out.println("simbolo nt:"+simbolosNoTerminales[i]);
                if (simbolosNoTerminales[i].equals(buscar)){
//                    System.out.println("encontrado no terminal "+i);
                    return i;
                }
        }
        return 0;
    }

    public int indiceTerminal(String buscar) {
        for (int i = 0; i < simbolosTerminales.length; i++) {
            if (simbolosTerminales[i] != null)
//                System.out.println("simbolo t:"+simbolosTerminales[i]);
                if (simbolosTerminales[i].equals(buscar)){
//                    System.out.println("encontrado terminal "+i);
                    return i;
                }
        }
        return 0;
    }

    public String[] produccionDerecha(int produccion) {
        if (todasLasProducciones[produccion - 1][1] != null) {
            return nes.noEsEsplit(todasLasProducciones[produccion - 1][1], " ");
        }
        return null;
    }

    private void imprimeGramaticaCompleta() {
        for (String[] prod : todasLasProducciones)
            //System.out.println(prod[0] + "\t->\t" + prod[1]);
            System.out.printf("10s 3s 10s", prod[0] );
    }

    public void imprime() {
        System.out.println("\tGramática completa");
        imprimeGramaticaCompleta();
        System.out.println("\n\tLados Derechos");
        imprime(ladosDerechos);
        System.out.println("\n\tSímbolos No Terminales");
        imprime(simbolosNoTerminales);
        System.out.println("\n\tSímbolos Terminales");
        imprime(simbolosTerminales);
    }

    private void separarProducciones() {
        gramaticaCompleta = nes.noEsEsplit(leerA.datos(), "\n");
    }

    private void imprime(String[] arreglo) {
        for (String prod : arreglo) {
            if (prod != null)
                System.out.println(prod);
        }
    }

    public String simboloInicial() {
        return simbolosNoTerminales[0];
    }
}

class test {
    public static void main(String[] args) {
        AcomodoGramatica ag = new AcomodoGramatica();
        ag.ini();
        ag.imprime();
    }
}
```

```
package analizador.lexico.c;

import Lectura.LeerArchivo;
import Estructuras.ListasR;

public class ClasificaRebuilt {

    Tipos tipo = new Tipos();
    ConversionCaracter conv = new ConversionCaracter();
    String archivo = "", token;
    LeerArchivo leer = new LeerArchivo();
    ListasR listaTab;
    PalabraReservada palR;
    int actual = 0;

    public ClasificaRebuilt() {
        leer.leerArchivo();
        archivo = leer.datos();
        palR = new PalabraReservada();
        crearListasT();
    }

    public void retroceder(){
        actual=actual-6;
        listaTab.agregarElementoLSimbolosR("begin", tipoPalabra("begin"),
listaTab.buscaRepR("begin") - 1, calValToken("begin"), -1, null);
    }

    private void crearListasT() {
        listaTab = new ListasR();
    }

    public String pedirToken() {
        q0(archivo);
        System.out.println(token);
        return token;
    }

    public void q0(String archivo) {
        if (actual < archivo.length()) {
            conv.convertirCaracter(archivo.charAt(actual));
            if (tipo.esEspacio(conv.getAscii())) {
                if (actual + 1 < archivo.length()) {
                    actual++;
                    q0(archivo);
                }
                //NOTA: CREO FALTA AGREGAR ELSE
            } else if ((tipo.esMinuscula(conv.getAscii()) == true) ||
(tipo.esMayuscula(conv.getAscii()) == true)) {
                actual++;
                q1Identificador(archivo);
            } else if (tipo.esNumero(conv.getAscii()) == true) {
                actual++;
                q2NumeroEntero(archivo);
            } else if (tipo.esParentesis1(conv.getAscii()) == true) {
                actual++;
                token = crearCadena(actual - 1, actual + 1, archivo);
                /*Token|Tipo Token|Valor Token|Valor Identf|Tipo Identif|#Veces Repite|*/
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
//                q1Identificador(archivo);
            } else if (tipo.esParentesis2(conv.getAscii()) == true) {
                actual++;
                token = crearCadena(actual - 1, actual + 1, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
//                q1Identificador(archivo);
            } else if (tipo.esComa(conv.getAscii()) == true) {
                actual++;
                token = crearCadena(actual - 1, actual + 1, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
//                q1Identificador(archivo);
            } else if (tipo.esPyC(conv.getAscii()) == true) {
                actual++;
                token = crearCadena(actual - 1, actual + 1, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
//                q1Identificador(archivo);
            } else if (tipo.esDPuntos(conv.getAscii()) == true) {
                actual++;
                q3Asignacion(archivo);
            } else if (tipo.esMas(conv.getAscii()) == true) {
                actual++;
                token = crearCadena(actual - 1, actual + 1, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
//                q1Identificador(archivo);
            } else if (tipo.esMenos(conv.getAscii()) == true) {
                actual++;
                token = crearCadena(actual - 1, actual + 1, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
//                q1Identificador(archivo);
            } else {
                actual++;
                qErrorLexico(archivo, 0);
            }
        }
    }

    public void q1Identificador(String archivo) {
        int movs = 1;
        for (int i = actual; i < archivo.length(); i++) {
            conv.convertirCaracter(archivo.charAt(i));
            if (tipo.esEspacio(conv.getAscii())) {
                token = crearCadena(actual - 1, actual + movs, archivo);
                listaTab.agregarElementoLSimbolosR(token, tipoPalabra(token),
listaTab.buscaRepR(token) + 1, calValToken(token), -1, null);
                if (!palR.ExistePalabraReservada(token)) {
                    token = "ID";
                }
                actual = actual + movs;
                break;
            } else if ((tipo.esMinuscula(conv.getAscii()) == true)
                    || (tipo.esMayuscula(conv.getAscii()) == true)
                    || (tipo.esNumero(conv.getAscii()) == true)) {
                movs++;
            } else {
                token = crearCadena(actual - 1, actual + movs, archivo);
                listaTab.agregarElementoLSimbolosR(token, tipoPalabra(token),
listaTab.buscaRepR(token) + 1, calValToken(token), -1, null);
                if (!palR.ExistePalabraReservada(token)) {
                    token = "ID";
                }
                actual = actual + movs - 1;
//                qErrorLexico(archivo, movs);
                break;
            }
        }
    }

    public void q2NumeroEntero(String archivo) {
        int movs = 1;
        for (int i = actual; i < archivo.length(); i++) {
            conv.convertirCaracter(archivo.charAt(i));
            if (tipo.esEspacio(conv.getAscii())) {
                token = crearCadena(actual - 1, actual + movs, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Número", listaTab.buscaRepR(token) +
1, 500, token, null);
                token = "int";
                actual = actual + movs;
                break;
            } else if (tipo.esNumero(conv.getAscii()) == true) {
                movs++;
            } else {
                token = crearCadena(actual - 1, actual + movs, archivo);
                token = "int";
                actual = actual + movs - 1;
//                qErrorLexico(archivo, movs);
                break;
            }
        }
    }

    public void q3Asignacion(String archivo) {
        int movs = 1;
        for (int i = actual; i < archivo.length(); i++) {
            conv.convertirCaracter(archivo.charAt(i));
            if (tipo.esEspacio(conv.getAscii())) {
                token = crearCadena(actual - 1, actual + movs, archivo);
                listaTab.agregarElementoLSimbolosR(token, "Simb. Esp.",
listaTab.buscaRepR(token) + 1, (int) (token.charAt(0)) + 300, 0, null);
                actual = actual + movs;
                break;
            } else if (tipo.esIgual(conv.getAscii()) == true) {
                movs++;
                //NOTA: SOLO DEBE HABER UNO
            } else {
                token = crearCadena(actual - 1, actual + movs, archivo);
                actual = actual + movs - 1;
//                qErrorLexico(archivo, movs);
                break;
            }
        }
    }

    public void qErrorLexico(String archivo, int movs) {
        for (int i = actual + movs; i < archivo.length(); i++) {
            conv.convertirCaracter(archivo.charAt(i));
            if (tipo.esEspacio(conv.getAscii()) == true) {
                movs++;
                token = crearCadena(actual - 1, actual + movs, archivo);
                actual = actual + movs;
                break;
            } else {
                movs++;
            }
        }

    }

    public String crearCadena(int i, int f, String archivo) {
        String cad = "";

        for (int j = i; j < f - 1; j++) {
            cad = cad + archivo.charAt(j);
        }

        return cad;
    }

    public void imprimeTablas() {
        listaTab.mostrarListaSimbolosR();
    }

    public String tipoPalabra(String token) {
        if (palR.ExistePalabraReservada(token)) {
            return "Palabra Re.";
        } else {
            return "Identificador";
        }
    }

    public int calValToken(String Token) {
        if (palR.ExistePalabraReservada(token)) {
            return palR.getValorPalabraReservada(Token);
        } else {
            if (listaTab.ExistePalabraT(token)) {
                listaTab.buscaRepR(token);
            } else {
                if (listaTab.ExistePalabraT(token)) {
                    return listaTab.buscaIdent(token);
                } else {
                    return listaTab.ultimoEnFila() + 1;
                }
                //return -1;
            }
            return 0;
        }
    }

    public static void main(String[] args) {
        ClasificaRebuilt obj = new ClasificaRebuilt();
//        LeerArchivo leer = new LeerArchivo();
//        leer.leerArchivo();

        //String archivo = "Este .9 Este 0 00 .9 es 002 00t un Ar rA rA3 rA3.s 4.3e4 archivo
archivo2 archi. 0. de prueba 12 12.12 0 0 00 edwsd 12 12.1 . . .. edsd # # ##  ";
        //String archivo2 = ";";
        //String archivo3 = leer.datos();
        //obj.q0(archivo2);
        //        obj.im(archivo);
        while (!obj.pedirToken().equals("end")) {
            obj.pedirToken();
        }
        obj.imprimeTablas();
    }
}
```

```
package analizador.lexico.c;

public class ConversionCaracter {

    private int ascii = 0;
    private char carac;

    public void convertirCaracter(char caracter){       //convierte caracter a
ascii
        ascii = (int) caracter;
    }

    public int getAscii(){                              //devuelve el valor en
ascii
        return ascii;
    }

    public int getAscii(char caracter){                             //devuelve el
valor en ascii
        return (int) caracter;
    }

    public void convertirAscii (int numero) {           //convierte ascii a
caracter
        carac = (char)numero;
    }

    public char getCaracter(){                          //devuelve el caracter
        return carac;
    }
}
```

```
package Lectura;

import java.awt.HeadlessException;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class LeerArchivo extends javax.swing.JFrame {

    private JFileChooser ventana = null;
    private File archivo = null;
    private String cadena = "";

    public void leerArchivo(){
        String bfRead;
        ventana = new JFileChooser();
        ventana.setCurrentDirectory(new File("src/Archivos/"));
        FileNameExtensionFilter filtroTxt = new FileNameExtensionFilter("TXT",
"txt");
        ventana.setFileFilter(filtroTxt);
        ventana.setDialogTitle("Abrir archivo");
        if (ventana.showOpenDialog(ventana) == JFileChooser.APPROVE_OPTION) {
            archivo = ventana.getSelectedFile();
            try{
                BufferedReader bf = new BufferedReader(new FileReader(archivo));
                while ((bfRead = bf.readLine()) != null)
                    cadena += bfRead + ' ';
            } catch (HeadlessException | IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }

    public String rutaArchivo() {
        return ventana.getSelectedFile().toString();
    }

    public String datos() {
        return cadena + ' ';
    }
}
```

```java
package Estructuras;

public class ListasR<dato> {
    private NodoTSimR inicioSim, finSim;
    private NodoTErrores inicioErr, finErr;
    private NodoTToken inicioTok, finTok;
    private NodoTReservadas inicioR, finR;

    /*
            Supuesta tabla de símbolos
    Token   |   Tipo Token    |  Valor Token   |   Valor Identf  |  Tipo Identif
|  #Veces Repite  |
      w        identificador         115               0                int
3
    */

    class NodoTSimR<dato> {
        public String token, tipoToken, tipoIdentificador;
        dato valorToken, valorIdentificador, vecesRepite;
        public NodoTSimR siguiente;

        public NodoTSimR(String token, String tipoToken, dato valorToken, dato
valorIdentificador, String tipoIdentificador, dato vecesRepite) {
            this.token = token;
            this.tipoToken = tipoToken;
            this.valorToken = valorToken;
            this.valorIdentificador = valorIdentificador;
            this.tipoIdentificador = tipoIdentificador;
            this.vecesRepite = vecesRepite;
        }
    }

    class NodoTErrores {
        public String error;
        public NodoTErrores siguiente;

        public NodoTErrores (String error) {
            this.error = error;
        }
    }

    class NodoTToken {
        public dato palabra; String tipoTok; int tokenTok;
        public NodoTToken siguiente;

        public NodoTToken (dato palabra, String tipoTok, int tokenTok) {
            this.palabra = palabra;
            this.tipoTok = tipoTok;
            this.tokenTok = tokenTok;
        }
    }

    class NodoTReservadas {
        public String palabraR; int tokenR;
        public NodoTReservadas siguiente;

        public NodoTReservadas (String palabraR, int tokenR) {
            this.palabraR = palabraR;
            this.tokenR = tokenR;
        }
    }

        /* Inserta nuevos elementos
                Símbolos
            Tipo    Identificador   Valor
        */
        public void agregarElementoLSimbolosR(String token, String tipoToken,
dato vecesRepite, dato valorToken, dato valorIdentificador, String
tipoIdentificador) {
            if (!ExistePalabraT(token)) {
                NodoTSimR agregarElemento = new NodoTSimR(token, tipoToken,
valorToken, valorIdentificador, tipoIdentificador, vecesRepite);
                if (inicioSim != null) {  // Existe el inicio
                    finSim.siguiente = agregarElemento;  //Agregar al final de la
                    finSim = agregarElemento;
                } else {
                    inicioSim = finSim = agregarElemento; //Crea la lista con su
primer Nodo
                }
            } else
                siExiste(token, vecesRepite);
        }

        public void siExiste(String token, dato vecesRepite) {
            NodoTSimR recorrer = inicioSim;
            while (recorrer != null) {
                if (recorrer.token.equals(token)) {
//                    System.out.println("vr: " + vecesRepite);
                    recorrer.vecesRepite = vecesRepite;
                    break;
                }
                recorrer = recorrer.siguiente;
            }
        }


        /*
                Errores
            Palabra  Error
        */
        public void agregarElementoLErroresR(String palabra) {
            NodoTErrores agregarElemento = new NodoTErrores(palabra);
            if (inicioErr != null) {
                finErr.siguiente = agregarElemento;
                finErr = agregarElemento;
            } else {
                inicioErr = finErr = agregarElemento;
            }
        }

        /*
                Tokens
            Palabra  Tipo   Token
        */
        public void agregarElementoLTokensR(dato palabra, String tipo, int token)
{
            NodoTToken agregarElemento = new NodoTToken(palabra, tipo, token);
            if (inicioTok != null) {
                finTok.siguiente = agregarElemento;
                finTok = agregarElemento;
            } else {
                inicioTok = finTok = agregarElemento;
            }
        }

        /*
                Reservadas
            Palabra  Token
        */
        public void agregarElementoLReservadasR(String palabra, int token) {
            NodoTReservadas agregarElemento = new NodoTReservadas(palabra,
token);
            if (inicioR != null) {
                finR.siguiente = agregarElemento;
                finR = agregarElemento;
            } else {
                inicioR = finR = agregarElemento;
            }
        }

        public void mostrarListaErroresR() {
            NodoTErrores recorrer = inicioErr;
            while (recorrer != null) {
                System.out.println(recorrer.error);
                recorrer = recorrer.siguiente;
            }
        }

        public void mostrarListaSimbolosR() {
            NodoTSimR recorrer = inicioSim;
            System.out.println("Token   |   Tipo Token    |  Valor Token   |
Valor Identf  |  Tipo Identif  |  #Veces Repite");
            while (recorrer != null) {
                System.out.printf("%-10s %13s %10s %17s %18s %15s %n",
                                  recorrer.token,
                                  recorrer.tipoToken,
                                  recorrer.valorToken,
                                  recorrer.valorIdentificador,
                                  recorrer.tipoIdentificador,
                                  recorrer.vecesRepite);
                recorrer = recorrer.siguiente;
            }
        }

        public void mostrarListaTokensR() {
            NodoTToken recorrer = inicioTok;
            while (recorrer != null) {
                System.out.println(recorrer.palabra + "\t" +
                                   recorrer.tipoTok + "\t" +
                                   recorrer.tokenTok);
                recorrer = recorrer.siguiente;
            }
        }

        public void mostrarListaReservadasR() {
            NodoTReservadas recorrer = inicioR;
            while (recorrer != null) {
                System.out.println(recorrer.palabraR + "\t" +
                                   recorrer.tokenR + "\t");
                recorrer = recorrer.siguiente;
            }
        }

        public boolean ExistePalabraT(String token) {
            NodoTSimR recorrer = inicioSim;
            while (recorrer != null) {
                if (recorrer.token.equals(token))
                    return true;
                else
                    recorrer = recorrer.siguiente;
            }
            return false;
        }

        public int buscaIdent(String token) {
            NodoTSimR recorrer = inicioSim;
            while (recorrer != null) {
                if (recorrer.token.equals(token))
                    return (int) recorrer.valorToken;
                recorrer = recorrer.siguiente;
            }
            return 0;
        }

        public int buscaRepR(String token) {
            NodoTSimR recorrer = inicioSim;
            int sumEn = 0;
            while (recorrer != null) {
                if (recorrer.token.equals(token)) {
                    sumEn = (int) recorrer.vecesRepite;
                }
                recorrer = recorrer.siguiente;
            }
            return sumEn;
        }

        public int ultimoEnFila() {
            NodoTSimR recorrer = inicioSim;
            int idValue = 0;
            while (recorrer != null) {
                if (recorrer.tipoToken.equals("Identificador"))
                    idValue = (int) recorrer.valorToken;
                recorrer = recorrer.siguiente;
            }
            return idValue;
        }
}
```

```
package Sintactico;

import Estructuras.Pila;
import Gramatica.AcomodoGramatica;
import analizador.lexico.c.ClasificaRebuilt;

public class MatrizPredictiva {

    Pila pila;
    AcomodoGramatica gramatica;
    ClasificaRebuilt lexico;
    boolean error = false;

    MatrizPredictiva() {
        pila = new Pila();
        gramatica = new AcomodoGramatica();
        gramatica.ini();  // pedimos la gramatica y se trabaja
        lexico = new ClasificaRebuilt(); // pedimos el programa a analizar
    }

    public int matriz(int x, int y) {
                                    /* |beg|end| id| :=| ; |rea| ( | ) |wri| , |int|
+ | - | $| */
        int[][] matrizPredictiva = {{ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0},
                                    { 0,  0,  2,  0,  0,  2,  0,  0,  2,  0,  0,
0,  0,  0},
                                    { 0,  4,  3,  0,  0,  3,  0,  0,  3,  0,  0,
0,  0,  0},
                                    { 0,  0,  5,  0,  0,  6,  0,  0,  7,  0,  0,
0,  0,  0},
                                    { 0,  0,  8,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0},
                                    { 0,  0,  0,  0,  0,  0,  0, 10,  0,  9,  0,
0,  0,  0},
                                    { 0,  0, 11,  0,  0,  0, 11,  0,  0,  0, 11,
0,  0,  0},
                                    { 0,  0,  0,  0, 12,  0,  0, 13,  0,  0,  0,
0,  0,  0},
                                    { 0,  0, 14,  0,  0,  0, 14,  0,  0,  0, 14,
0,  0,  0},
                                    { 0,  0,  0,  0, 16,  0,  0, 16,  0, 16,  0,
15, 15,  0},
                                    { 0,  0, 18,  0,  0,  0, 17,  0,  0,  0, 19,
0,  0,  0},
                                    { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
20, 21,  0},
                                    {22,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0}};
        return matrizPredictiva[x][y];
    }

    public int obtenProduccionMatrizP(String x, String a) {
        int posX = gramatica.indiceNoTerminal(x); // 0
        int posY = gramatica.indiceTerminal(a);  // 0
        return matriz(posX, posY);
    }

    public boolean noEsTerminal(String buscar) {
        for (String buscarEn : gramatica.simbolosNoTerminales) {
            if (buscarEn != null)
                if (buscarEn.equals(buscar))
                    return true;
        }
        return false;
    }

    public void LlDiver() { //a sera el lexema enviado del analizador lexico
        pila.push(gramatica.simboloInicial());
        String x = pila.peak(); // tope de la pila
//        System.out.println("x: "+x);
        String a = lexico.pedirToken(); // pedir la primer palabra
//        System.out.println("pedir token 1");
//        System.out.println("a: "+a);
//        System.out.println("inicia pila:");
//        pila.imprime();
//        System.out.println("termina pila");
//                    System.out.print(a+" ");
        while(pila.isEmpty()) {
            if (noEsTerminal(x)) {
                if(obtenProduccionMatrizP(x, a) != 0) {
//                    System.out.println("no-matriz: "+obtenProduccionMatrizP(x,
a));
                    pila.pop(); //y un ciclo push();
                    cicloPush(obtenProduccionMatrizP(x, a)); // derecha a
izquierda
//                                System.out.println("inicia pila:");
//        pila.imprime();
//        System.out.println("termina pila");
//                    System.out.println("despues");
                    x = pila.peak();
//                    System.out.println("x: "+x);
                } else {
//                    System.out.println("error 1");
                    errorSintactico(a);
                    lexico.imprimeTablas();
                    break;
                }
            } else {
                if(x.equals(a)) {
                    pila.pop();
                    x = pila.peak();
//                    System.out.println("x: "+x);
                    a = lexico.pedirToken();
//                    System.out.println("pedir token 2");
//                    System.out.println("a: "+a);
                    System.out.print(a+" ");
                }
                else if(x.equals("ε")) {
                    pila.pop();
//                    pila.pop();
                    x = pila.peak();
                    System.out.println("vacio x: "+x);
                }
                else {
//                    System.out.println("error 2");
                    errorSintactico(a);
                    lexico.imprimeTablas();
                    break;
                }
            }
        }
        if(error == false){
            if (!a.equals("$")) {

//        System.out.println("inicia pila:");
//        pila.imprime();
//        System.out.println("termina pila");
                System.out.println("");
                LlDiver();
            } else {
                System.out.println("Análisis terminado");
                lexico.imprimeTablas();
            }
        }
    }

    private void cicloPush(int produccion) {
        String[] deriva = gramatica.produccionDerecha(produccion);     // la
produccion a ingresar
        int i = deriva.length;
        while (i != 0) {
            pila.push(deriva[i-1]);
            i=i-1;
        }
    }

    private void errorSintactico(String x) {
        if(!x.equals("$")){
            System.out.println("\nError sintactico en: " + x);
            error = true;
        }
    }
}

class test {
    public static void main(String[] args) {
        MatrizPredictiva m = new MatrizPredictiva();
        m.LlDiver();
        //m.obtenProduccionMatrizP(0, 7);
    }
}
```

```java
package Lectura;

public class NoEsDeString {
    private int numero;

    public int cantidadDeCadenas() {
        return numero;
    }

    //Método Split
    public String[] noEsEsplit(String Cad, String comodin) {
        //cuantos separadores existen en la cadena
        numero = 0;
        for (int i = 0; i < Cad.length(); i++) {
            if (Cad.charAt(i) == comodin.charAt(0)) {
                numero++;
            }
        }
        String[] vectemp = new String[numero + 1];
        int i = 0;
        while (i < numero) {
            int posicion = Cad.indexOf(comodin);

            vectemp[i] = Cad.substring(0, posicion);
            Cad = Cad.substring(posicion + 1);
            i++;
        }
        vectemp[i] = Cad;
        return (vectemp);
        //Victor Viera Balanta;
    }

    public String NoEsTrim(String palabra) {
        int inicio = 0, fin = 0;
        // recorre la palabra hasta encontrar un carcater diferente de espacio
        for(int i = 0, c = palabra.length(); i < c; i++) {
            if(palabra.charAt(i) == ' ')
                inicio++;
            else
                break;
        }

        for(int i = palabra.length() - 1; i >= 0; i--) {
            if(palabra.charAt(i) == ' ')
                fin++;
            else
                break;
        }
        return palabra.substring(inicio, palabra.length() - fin);
    }
}
```

```
package analizador.lexico.c;

//**********            Enumerado Palabras Reservadas            **********//

enum PalabrasReservadas { // lista de palabras reservadas a usar
        READ (200),
        ID (201),
        WRITE (202),
        INT (203),
        END (204),
        BEGIN (205);

    private final int valor;  // valor numérico de cada palabra reservada

    private PalabrasReservadas (int valor){  // constructor de la clase enum
PalabrasReservadas
        this.valor = valor;
    }

    public int getValor(){ // método para conseguir el valor numérico de cada
palabrareservada
        return valor;
    }
}

public class PalabraReservada {
    // método que recibe un string (palabra reservada) y verifica si existe en el
enumerado
    public boolean ExistePalabraReservada(String pReservada) {
        // ciclo que recorre el enumerado hasta encontrar la palabra recibida
        for (PalabrasReservadas pr : PalabrasReservadas.values()) {
                if (pr.name().toLowerCase().equals(pReservada.toLowerCase()))
                    return true; // si se encuentra regresa un true
        }
        return false; // en caso de que no existe se regresa un false
    }
    // método que devuelve el valor numerico de una palabra reservada
    public int getValorPalabraReservada(String pReservada) {
        // ciclo que recorre el enumerado hasta encontrar la palabra recibida
        for (PalabrasReservadas pr : PalabrasReservadas.values()) {
                if (pr.name().toLowerCase().equals(pReservada.toLowerCase()))
                    return pr.getValor(); // si la encuentra devuelve su valor
numérico
        }
        return -1; // en caso contrario masna un -1 que seria el caso de error
(no encontrado)
    }

    public void reservadas() {
        for (PalabrasReservadas pr : PalabrasReservadas.values()) {
            System.out.println(pr.name() + "\t" + pr.getValor());
        }
    }

    public String tipoPalabra(String token) {
        if (ExistePalabraReservada(token))
            return "Palabra Re.";
        else
            return "Identificador";
    }
}

class test {
    public static void main(String[] args) {
        PalabraReservada pr = new PalabraReservada();
        System.out.println(pr.tipoPalabra("bololean"));
        System.out.println("Existe: " + pr.ExistePalabraReservada("boolean"));
        System.out.println("Palabra: " + pr.getValorPalabraReservada("boolean"));

    }
}
```

```java
package Estructuras;

public class Pila {
    Nodo inicio, fin;

    public Pila() {
        inicio = null;
    }

    public class Nodo {
        Nodo sig;
        String dato;

        public Nodo(String dato) {
            this.dato = dato;
        }

        @Override
        public String toString() {//conversion de posicion de memoria a string
            return dato.toString();
        }
    }

    public void push(String dato) {
        Nodo nuevo = new Nodo(dato);
        push(nuevo);
    }

    private void push(Nodo dato) {
        if (inicio != null) {
            dato.sig = inicio;
            inicio = dato;
        } else
            inicio = fin = dato;
    }

    public void pop() {
        Nodo temp = inicio;
        if (inicio != null) {
            inicio = temp.sig;
            temp.sig = null;
        } else
            System.out.println("Pila vacia");
    }

    public String peak() {
        if (inicio != null) {
            return  inicio.dato;
        } else
            return null;
    }

    public boolean isEmpty() {
        return inicio != null;
    }

    public void imprime() {
        Nodo temp = inicio;
        while (temp != null) {
            System.out.println(temp);
            temp = temp.sig;
        }
    }
}

class test {
//    public static void main(String[] args) {
//        Pila p = new Pila();
//        p.push("1");
//        p.push("2");
//        p.push("3");
//        p.push("4");
//        System.out.println(":" + p.peak());
//        p.pop();
//        p.imprime();
//    }
}
```

```
package Sintactico;

public class tesst {
    public static void main(String[] args) {
        MatrizPredictiva m = new MatrizPredictiva();
        m.L1Diver();
    }
}
```

```
package Estructuras;

public class TestLista {

    ListasR al;

    TestLista() {
        al = new ListasR();
    }

    public void test(String token, String tipo) {
        al.agregarElementoLSimbolosR(token, tipo, al.buscaRepR(token) + 1,
(int)(token.charAt(0)), 0, null);
    }

    public void imp() {
        System.out.println(al.ExistePalabraT("0"));
        al.mostrarListaSimbolosR();
        System.out.println(al.buscaIdent("!"));
        System.out.println(al.ultimoEnFila());
    }

    public static void main(String[] args) {
        TestLista t = new TestLista();
        t.test(":=", "Identificador");
        t.test("!", "Sim. Esp.");
        t.test("!", "Identificador");
        t.test(":=", "Sim. Esp.");
        t.test("#", "Sim. Esp.");
        t.test("#", "Identificador");
        t.test("!", "Sim. Esp.");
        t.imp();

//        char c[] = {
//                    ',',
//                    '(',
//                    ')',
//                    '+',
//                    '-',
//                    ':',
//                    ';',
//                    '='};
//            for (char ci : c)
//                System.out.println( Integer.valueOf(ci) );
    }
}
```

```
package analizador.lexico.c;

public class Tipos {

    public boolean esEspacio(int ascii){
        if(ascii == 32 || ascii == 3 || ascii == 9)
            return true;
        else
            return false;
    }

    public boolean esCero(int ascii){
        if(ascii == 48 )
            return true;
        else
            return false;
    }

    public boolean esMayuscula(int ascii){              //verifica si el caracter
es mayuscula
        if(ascii >= 65 && ascii <= 90)
            return true;
        else
            return false;
    }

    public boolean esMinuscula(int ascii){              //verifica si el caracter
es minuscula
        if(ascii >= 97 && ascii <= 122)
            return true;
        else
            return false;
    }

    public boolean esNumero(int ascii){                 //verifica si el caracter
es numero
        if(ascii >= 49 && ascii <= 57)
            return true;
        else
            return false;
    }

    public boolean esParentesis1(int ascii){
        if(ascii == 40 )
            return true;
        else
            return false;
    }

    public boolean esParentesis2(int ascii){
        if(ascii == 41 )
            return true;
        else
            return false;
    }

    public boolean esComa(int ascii){
        if(ascii == 44 )
            return true;
        else
            return false;
    }

    public boolean esPyC(int ascii){
        if(ascii == 59 )
            return true;
        else
            return false;
    }

    public boolean esIgual(int ascii){
        if(ascii == 61 )
            return true;
        else
            return false;
    }

    public boolean esDPuntos(int ascii){
        if(ascii == 58 )
            return true;
        else
            return false;
    }

    public boolean esMas(int ascii){
        if(ascii == 43 )
            return true;
        else
            return false;
    }

    public boolean esMenos(int ascii){
        if(ascii == 45 )
            return true;
        else
            return false;
    }
}
```