	INSTITUTO TECNOLÓGICO DE CELAYA
PROGRAM	**AUTOR: Ramirez García Juana Rubi** **Valle Rodríguez Julio Cesar**

CARRERA	NOMBRE DE LA ASIGNATURA
INGENIERÍA EN SISTEMAS COMPUTACIONALES	LENGUAJES Y AUTÓMATAS II

PROGRAMA No.	NOMBRE DEL PROGRAMA
1 . 1	GENERACIÓN DE TOKENS Y TABLA DE SÍMBOLOS PROGRAMA CON AUTÓMATAS

1 INTRODUCCIÓN

Sabemos que todo compilador está compuesto de diferentes analizadores(léxico, sintáctico, semántico) y que las frases de un lenguaje constan de una cadena de componentes léxicos, de forma que a un traductor se le añade un analizador léxico que lea y convierta la entrada en una cadena de componentes léxicos que se enviaran al analizador sintáctico para avanzar en la gramática, funcionando como una subrutina de este último, cuando recibe la orden de obtener el siguiente componente léxico considerando que son los símbolos terminales de la gramática.

Así como también se sabe que los autómatas son modelos matemáticos, un reconocedor de lenguaje, el cual nos ayuda a saber si una palabra pertenece a nuestro lenguaje previamente definido.

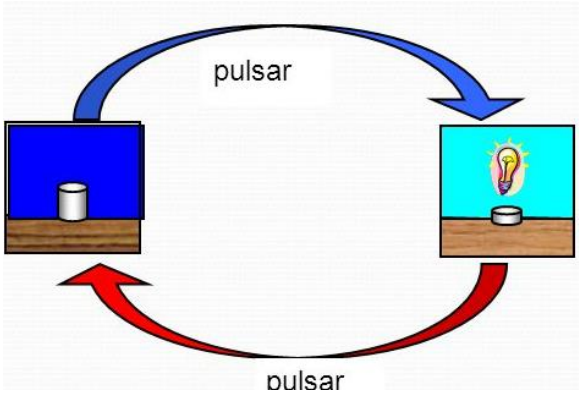
2 OBJETIVO (COMPETENCIA)

El alumno aprenderá a implementar los autómatas previamente diseñados dentro del código del programa 1 previamente codificado, para poder reconocer los tokens leídos del archivo y poder categorizarlos dentro de nuestra tabla de símbolos.

3 MARCO TEÓRICO

Un autómata o máquina de estado es el modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce valiéndose de estados. Por esto es que los autómatas son **Analizadores Léxicos** (llamados en inglés ``**Parsers''**).

Ejemplo:
El ejemplo más simple de autómata que podemos comenzar a analizar y como preámbulo al estudio de
autómatas sería un interruptor de apagado encendido. Donde podemos observar que comenzamos en un
estado, este estado de partida seria el estado **off** (**apagado**), el cual una vez que ocurre un evento en este caso el evento “Pulsar”, nos permite llegar a otro estado, el estado **on**(**encendido** , del cual podremos salir y regresar al de apagado una vez que ocurre de nuevo el evento pulsar.



Un autómata es un modelo matemático, reconocedor de un lenguaje para una máquina de estados finitos. Dada una cadena de entrada, salta a través de una serie de estados de acuerdo a una función de transición.

La cadena de entrada se lee símbolo por símbolo, hasta que se lea por completo, una vez que se completó la lectura el autómata se detiene. Dependiendo del estado en el que se encuentre el autómata una vez finalizada la lectura, se dice si la cadena se ha aceptado o rechazado, es decir, si la cadena pertenece o no al lenguaje.

Los conjuntos de todas las palabras aceptadas por el autómata constituyen el lenguaje aceptado por el mismo.

Tipos De Autómatas: (Menor A Mayor Jerarquía)

- Autómatas Finitos
- Autómatas Intermedios
  - De pila
  - De memoria limitada
- Máquinas de Turing

Los elementos de un autómata son:

- Símbolos: dato arbitrario que tiene algún significado o efecto en la máquina.
- Palabras: cadena finita formada por concatenación de un número de símbolos (tokens).
- Lenguaje: conjunto de palabras formadas por símbolos de un alfabeto dado. }

Los autómatas pueden ser representados por medio de grafos o también llamados diagramas de estados finitos en donde tienen como elementos:

- ***Estados:*** representados como vértices, etiquetados con su nombre en el interior.
- ***Transiciones:*** una transición de un estado a otro, depende de un símbolo del alfabeto, se representa mediante una arista dirigida que une a estos vértices, y que está etiquetada con el símbolo.
- ***Estado inicial:*** se caracteriza por tener una arista que llega a él, proveniente de ningún otro vértice.
- ***Estado final:*** representados mediante vértices que están encerrados a su vez por otra circunferencia.

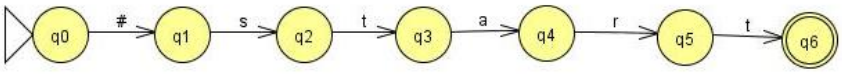
Se hicieron autómatas para poder clasificar los tokens en las siguientes categorías:

- Palabras reservadas
- Aritméticas
- Comparativas
- Identificadores
- Signos
- Números

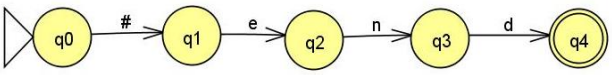
Se diseñaron los siguientes autómatas y son sobre los que se codificó el programa.

PALABRAS RESERVADAS

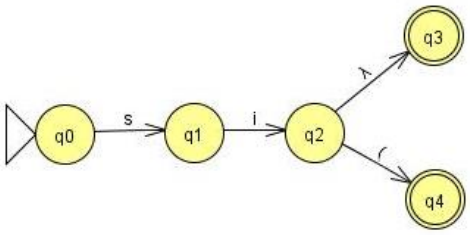
- #start



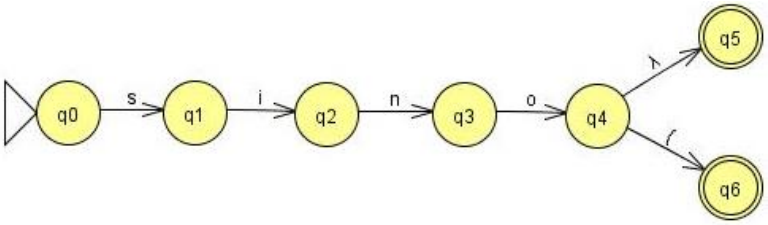
- #end



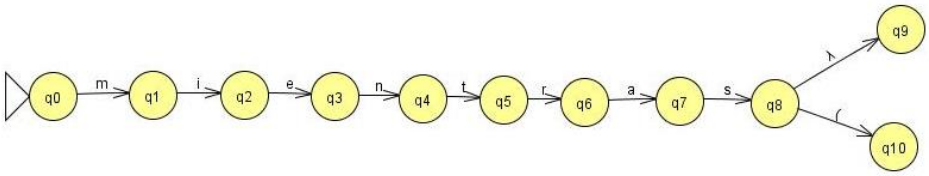
- Si



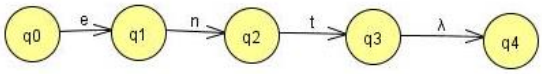
- sino



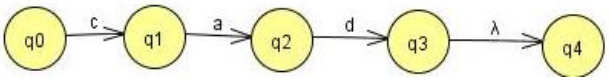
- mientras



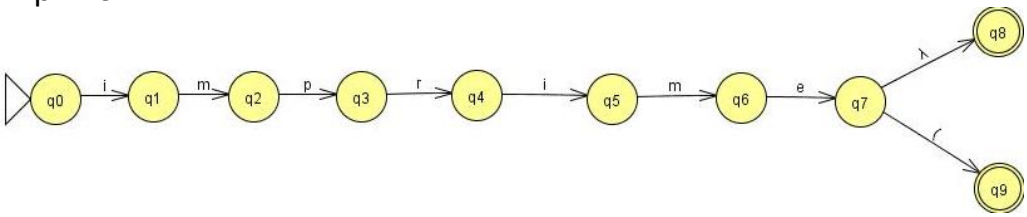
- ent



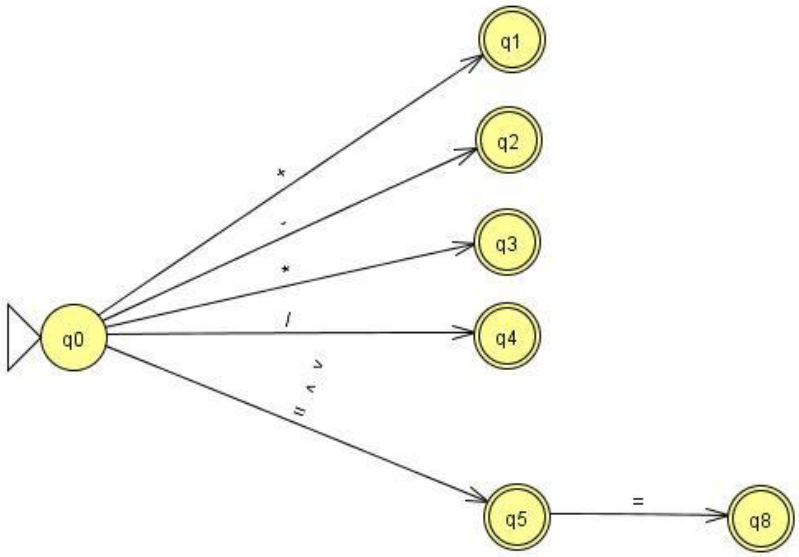
- cad



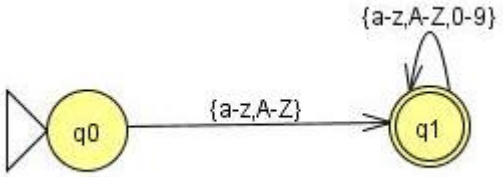
- imprime



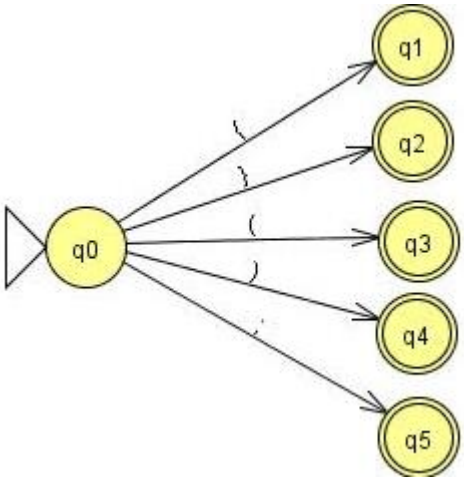
ARITMÉTICAS Y COMPARATIVAS



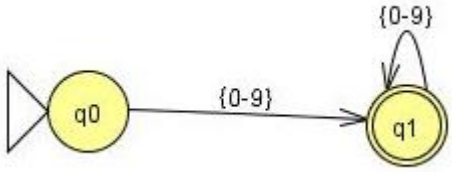
IDENTIFICADORES



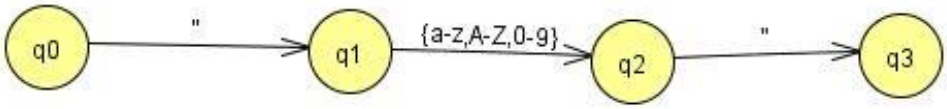
DELIMITADORES



NÚMEROS



CADENAS



4	DESARROLLO

Autómata Palabras reservadas

El autómata para identificar las palabras reservadas usa métodos como estados y de manera recursiva viaja a través de ellos y retorna el número de caracteres que ha leído satisfactoriamente la palabra.

```
int getPalabrasReservadas(String p_Palabra){
        int v_Indice=0;
        if(start(p_Palabra)>0){
            v_Indice=start(p_Palabra);
        }
        else{
            if(end(p_Palabra)>0)
                v_Indice=end(p_Palabra);
            else{
                if(si(p_Palabra)>0)
                    v_Indice=si(p_Palabra);
                else{
                    if(cad(p_Palabra)>0)
                        v_Indice=cad(p_Palabra);
                    else{
                        if(ent(p_Palabra)>0)
                            v_Indice=ent(p_Palabra);
                        else{
                            if(mientras(p_Palabra)>0)
                                v_Indice=mientras(p_Palabra);
                            else{
                                if(imprime(p_Palabra)>0)
                                    v_Indice=imprime(p_Palabra);
                            }
                        }
                    }
                }
            }
        }
        return v_Indice;
    }

    //#start
    private int start(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_startQ0(p_Palabra,0);
        return v_Recorrido;
    }

    int m_startQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='#'){
                v_Recorrido=m_startQ1(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }
```

```
int m_startQ1(String p_Palabra,int p_Index){
    int v_Recorrido=0;
    if(p_Palabra.length()-1>=p_Index){
        if(p_Palabra.charAt(p_Index)=='s'){
            v_Recorrido=m_startQ2(p_Palabra,2);
        }
    }
    return v_Recorrido;
}

int m_startQ2(String p_Palabra,int p_Index){
    int v_Recorrido=0;
    if(p_Palabra.length()-1>=p_Index){
        if(p_Palabra.charAt(p_Index)=='t'){
            v_Recorrido=m_startQ3(p_Palabra,3);
        }
    }
    return v_Recorrido;
}

int m_startQ3(String p_Palabra,int p_Index){
    int v_Recorrido=0;
    if(p_Palabra.length()-1>=p_Index){
        if(p_Palabra.charAt(p_Index)=='a'){
            v_Recorrido=m_startQ4(p_Palabra,4);
        }
    }
    return v_Recorrido;
}

int m_startQ4(String p_Palabra,int p_Index){
    int v_Recorrido=0;
    if(p_Palabra.length()-1>=p_Index){
        if(p_Palabra.charAt(p_Index)=='r'){
            v_Recorrido=m_startQ5(p_Palabra,5);
        }
    }
    return v_Recorrido;
}

int m_startQ5(String p_Palabra,int p_Index){
    int v_Recorrido=0;
    if(p_Palabra.length()-1>=p_Index){
        if(p_Palabra.charAt(p_Index)=='t'){
            v_Recorrido=m_startQ6(p_Palabra,6);
        }
    }
    return v_Recorrido;
}

int m_startQ6(String p_Palabra,int p_Index){
    int v_Recorrido=0;
    if(p_Palabra.length()-1>=p_Index){
        if(p_Palabra.charAt(p_Index)==' '){
            v_Recorrido=6;
        }
```

```
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=6;
            }
        }
        else{
            v_Recorrido=6;
        }
        return v_Recorrido;
    }

    //#end
    private int end(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_endQ0(p_Palabra,0);
        return v_Recorrido;
    }


    int m_endQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='#'){
                v_Recorrido=m_endQ1(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }

    int m_endQ1(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='e'){
                v_Recorrido=m_endQ2(p_Palabra,2);
            }
        }
        return v_Recorrido;
    }

    int m_endQ2(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='n'){
                v_Recorrido=m_endQ3(p_Palabra,3);
            }
        }
        return v_Recorrido;
    }

    int m_endQ3(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='d'){
                v_Recorrido=m_endQ4(p_Palabra,4);
            }
        }
        return v_Recorrido;
    }
```

```
    int m_endQ4(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=4;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=4;
            }
        }else{
            v_Recorrido=4;
        }
        return v_Recorrido;
    }

    //si - sino
    private int si(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_siQ0(p_Palabra,0);
        return v_Recorrido;
    }

    int m_siQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='s'){
                v_Recorrido=m_siQ1(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }

    int m_siQ1(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='i'){
                v_Recorrido=m_siQ2(p_Palabra,2);
            }
        }
        return v_Recorrido;
    }

    int m_siQ2(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='('){
                v_Recorrido=2;
            }
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=2;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=2;
            }
            if(p_Palabra.charAt(p_Index)=='n'){
                v_Recorrido=m_siQ3(p_Palabra,3);
```

```
            }
        }else{
            v_Recorrido=2;
        }
        return v_Recorrido;
    }

    int m_siQ3(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='o'){
                v_Recorrido=m_siQ4(p_Palabra,4);
            }
        }
        return v_Recorrido;
    }

    int m_siQ4(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='{'){
                v_Recorrido=4;
            }
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=4;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=4;
            }
        }else{
            v_Recorrido=4;
        }
        return v_Recorrido;
    }

    //ent
    private int ent(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_entQ0(p_Palabra,0);
        return v_Recorrido;
    }

    int m_entQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='e'){
                v_Recorrido=m_entQ1(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_entQ1(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='n'){
                v_Recorrido=m_entQ2(p_Palabra,p_Index+1);
```

```
            }
        }
        return v_Recorrido;
    }

    int m_entQ2(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='t'){
                v_Recorrido=m_entQ3(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_entQ3(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=3;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=3;
            }
        }else{
            v_Recorrido=3;
        }
        return v_Recorrido;
    }

    //cad
    private int cad(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_cadQ0(p_Palabra,0);
        return v_Recorrido;
    }

    int m_cadQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='c'){
                v_Recorrido=m_cadQ1(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_cadQ1(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='a'){
                v_Recorrido=m_cadQ2(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }
```

```
    int m_cadQ2(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='d'){
                v_Recorrido=m_cadQ3(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_cadQ3(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=3;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=3;
            }
        }else{
            v_Recorrido=3;
        }
        return v_Recorrido;
    }

    //mientras
    private int mientras(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_mientrasQ0(p_Palabra,0);
        return v_Recorrido;
    }

    int m_mientrasQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='m'){
                v_Recorrido=m_mientrasQ1(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ1(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='i'){
                v_Recorrido=m_mientrasQ2(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ2(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='e'){
                v_Recorrido=m_mientrasQ3(p_Palabra,p_Index+1);
```

```
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ3(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='n'){
                v_Recorrido=m_mientrasQ4(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ4(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='t'){
                v_Recorrido=m_mientrasQ5(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ5(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='r'){
                v_Recorrido=m_mientrasQ6(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ6(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='a'){
                v_Recorrido=m_mientrasQ7(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ7(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='s'){
                v_Recorrido=m_mientrasQ8(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_mientrasQ8(String p_Palabra,int p_Index){
        int v_Recorrido=0;
```

```
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=8;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=8;
            }
            if(p_Palabra.charAt(p_Index)=='('){
                v_Recorrido=8;
            }
        }else{
            v_Recorrido=8;
        }
        return v_Recorrido;
    }

    //imprime
    private int imprime(String p_Palabra){
        int v_Recorrido=0;
        v_Recorrido=m_imprimeQ0(p_Palabra,0);
        return v_Recorrido;
    }

    int m_imprimeQ0(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='i'){
                v_Recorrido=m_imprimeQ1(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ1(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='m'){
                v_Recorrido=m_imprimeQ2(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ2(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='p'){
                v_Recorrido=m_imprimeQ3(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ3(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='r'){
```

```
                v_Recorrido=m_imprimeQ4(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ4(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='i'){
                v_Recorrido=m_imprimeQ5(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ5(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='m'){
                v_Recorrido=m_imprimeQ6(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ6(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)=='e'){
                v_Recorrido=m_imprimeQ7(p_Palabra,p_Index+1);
            }
        }
        return v_Recorrido;
    }

    int m_imprimeQ7(String p_Palabra,int p_Index){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Index){
            if(p_Palabra.charAt(p_Index)==' '){
                v_Recorrido=7;
            }
            if(p_Palabra.charAt(p_Index)==10){
                v_Recorrido=7;
            }
            if(p_Palabra.charAt(p_Index)=='('){
                v_Recorrido=7;
            }
        }else{
            v_Recorrido=p_Index;
        }
        return v_Recorrido;
    }
```

Autómata Operadores Aritméticos y Comparativos

La clase para los operadores solo usa métodos recursivos dentro de los operadores comparativos ya que cuentan con más de un estado.

```
/**
 * @author Ramirez García Juana Rubi
 * @author Valle Rodriguez Julio Cesar
 */
package compilador;

public class Operadores {

    int getOperadores(String p_Palabra){
        int v_Indice=0;
        if(Mas(p_Palabra) >0){
            v_Indice=Mas(p_Palabra);
        }
        else{
            if(Menos(p_Palabra) >0){
                v_Indice=Menos(p_Palabra);
            }
            else{
                if(Por(p_Palabra) >0){
                    v_Indice=Por(p_Palabra);
                }
                else{
                    if(Entre(p_Palabra) >0){
                        v_Indice=Entre(p_Palabra);
                    }
                    else{
                        if(MenorQue(p_Palabra,v_Indice) >0){
                            v_Indice=MenorQue(p_Palabra,v_Indice);
                        }
                        else{
                            if(MayorQue(p_Palabra,v_Indice) >0){
                                v_Indice=MayorQue(p_Palabra,v_Indice);
                            }
                            else{
                                if(Igual(p_Palabra,v_Indice) >0){
                                    v_Indice=Igual(p_Palabra,v_Indice);
                                }
                                else{
                                    v_Indice=0;
                                }
                            }
                        }
                    }
                }
            }
        }

        return v_Indice;
    }

    private int Mas(String p_Palabra){
```

```
        if(p_Palabra.charAt(0)=='+')
            return 1;
        else
            return 0;
    }

    private int Menos(String p_Palabra){
        if(p_Palabra.charAt(0)=='-')
            return 1;
        else
            return 0;
    }

    private int Por(String p_Palabra){
        if(p_Palabra.charAt(0)=='*')
            return 1;
        else
            return 0;
    }

    private int Entre(String p_Palabra){
        if(p_Palabra.charAt(0)=='/')
            return 1;
        else
            return 0;
    }

    private int MenorQue(String p_Palabra,int p_Indice){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>p_Indice){
            if(p_Palabra.charAt(0)=='<'){
                v_Recorrido=IgualQ(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }

    private int MayorQue(String p_Palabra,int p_Indice){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>p_Indice){
            if(p_Palabra.charAt(0)=='>'){
                v_Recorrido=IgualQ(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }

    private int Igual(String p_Palabra,int p_Indice){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>p_Indice){
            if(p_Palabra.charAt(0)=='='){
                v_Recorrido=IgualQ(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }
```

```
    private int IgualQ(String p_Palabra,int p_Indice){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Indice){
            if(p_Palabra.charAt(p_Indice)=='='){
                v_Recorrido=2;
            }else{
                v_Recorrido=1;
            }
        }else{
            v_Recorrido=1;
        }
        return v_Recorrido;
    }
}
```

Autómatas Identificadores

El autómata de identificadores solo cuenta con solo 2 estados ya que solo debe verificar que un identificador debe iniciar con una letra minúscula o mayúscula como regla de nuestro lenguaje

```
/**
 * @author Ramirez García Juana Rubi
 * @author Valle Rodriguez Julio Cesar
 */
package compilador;

public class Identificadores {

    int getIndentificador(String p_Palabra){
        int v_Indice=0;
        v_Indice=m_IdentQ0(p_Palabra,0);
        return v_Indice;
    }

    private int m_IdentQ0(String p_Palabra,int p_Indice){
        int v_Indice=0;

if(p_Palabra.charAt(p_Indice)>=65&&p_Palabra.charAt(v_Indice)<=90||

p_Palabra.charAt(p_Indice)>=97&&p_Palabra.charAt(v_Indice)<=122)
            v_Indice=m_IdentQ1(p_Palabra,v_Indice+1);
        return v_Indice;
    }

    private int m_IdentQ1(String p_Palabra,int p_Indice){
        int v_Indice=p_Indice;
        if (p_Palabra.length()-1>=p_Indice){

if(p_Palabra.charAt(p_Indice)>=48&&p_Palabra.charAt(v_Indice)<=57||

p_Palabra.charAt(p_Indice)>=65&&p_Palabra.charAt(v_Indice)<=90||

p_Palabra.charAt(p_Indice)>=97&&p_Palabra.charAt(v_Indice)<=122)
                v_Indice=m_IdentQ1(p_Palabra,v_Indice+1);
        }
        return v_Indice;
    }
}
```

Autómatas Delimitadores

El autómata de los delimitadores solo cuenta con un dos estados (inicial y final) ya que solo hace la labor de identificar un signo como tal, por lo que solo regresa el valor de 1 de encontrar un resultado positivo

```
/**
 * @author Ramirez García Juana Rubi
 * @author Valle Rodriguez Julio Cesar
 */
package compilador;

public class Delimitadores {

    int  getDelimitadores(String p_Palabra){
        int v_Indice=0;
        if(m_PuntoComa(p_Palabra)!=0){
            v_Indice=1;
        }else{
            if(m_ParentesisAbierto(p_Palabra)!=0){
                v_Indice=1;
            }else{
                if(m_ParentesisCerrado(p_Palabra)!=0){
                    v_Indice=1;
                }else{
                    if(m_LlavesAbierto(p_Palabra)!=0){
                        v_Indice=1;
                    }else{
                        if(m_LlavesCerrado(p_Palabra)!=0){
                            v_Indice=1;
                        }
                    }
                }
            }
        }
        return v_Indice;
    }

    //Punto y coma
    int m_PuntoComa(String p_Palabra){
        if(p_Palabra.charAt(0)==';')
            return 1;
        else
            return 0;
    }

    //ParentesisAbierto
    int m_ParentesisAbierto(String p_Palabra){
        if(p_Palabra.charAt(0)=='(')
            return 1;
        else
            return 0;
    }

    //ParentesisCerrado
    int m_ParentesisCerrado(String p_Palabra){
        if(p_Palabra.charAt(0)==')')
            return 1;
```

```
        else
            return 0;
    }

    //ParentesisCerrado
    int m_LlavesAbierto(String p_Palabra){
        if(p_Palabra.charAt(0)=='{')
            return 1;
        else
            return 0;
    }

    //ParentesisCerrado
    int m_LlavesCerrado(String p_Palabra){
        if(p_Palabra.charAt(0)=='}')
            return 1;
        else
            return 0;
    }
}
```

Autómata Enteros y Cadenas

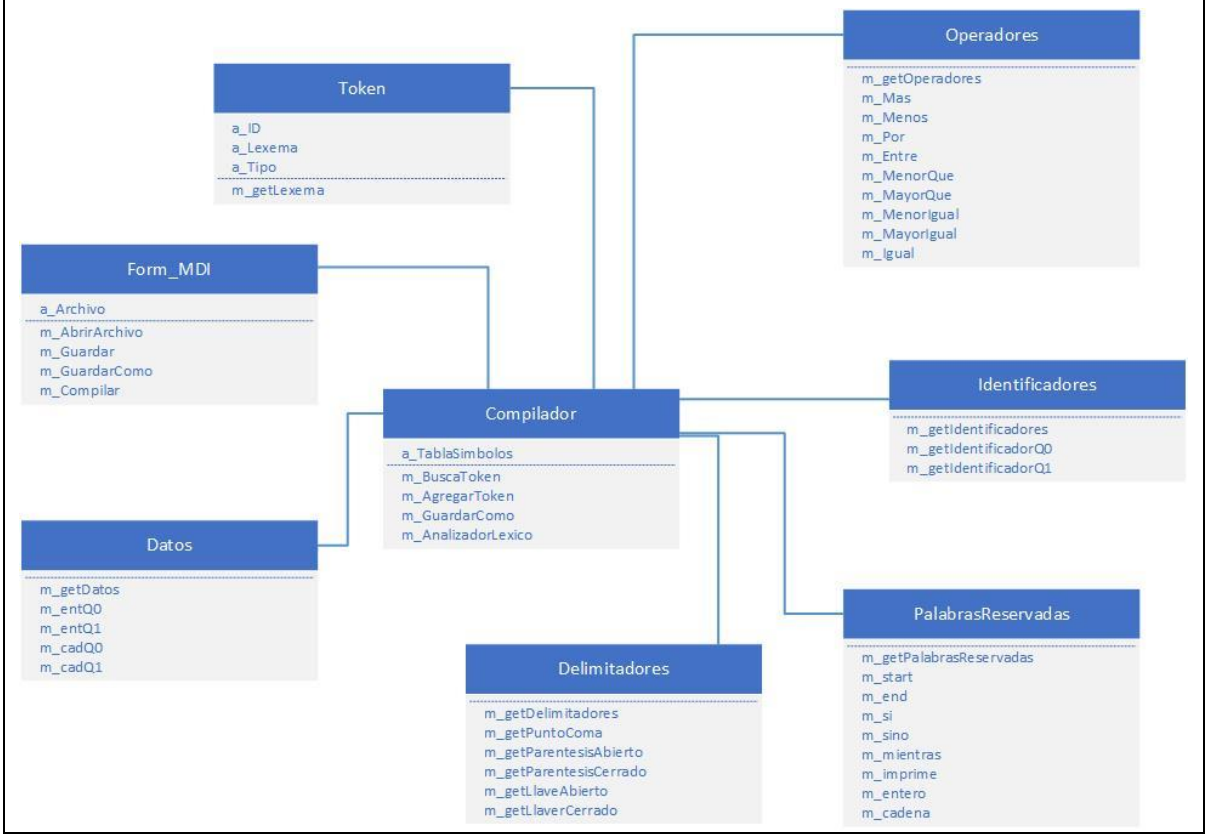
Los autómatas para detectar los tipos de datos enteros y de cadena fueron ubicados en una sola clase ya que son autómatas cortos donde la única condición para los enteros es que cuenten con números enteros y que las cadenas inicien y cierren con comillas dobles, de lo contrario se enviaría un resultado negativo para generar el error de construcción de una cadena.

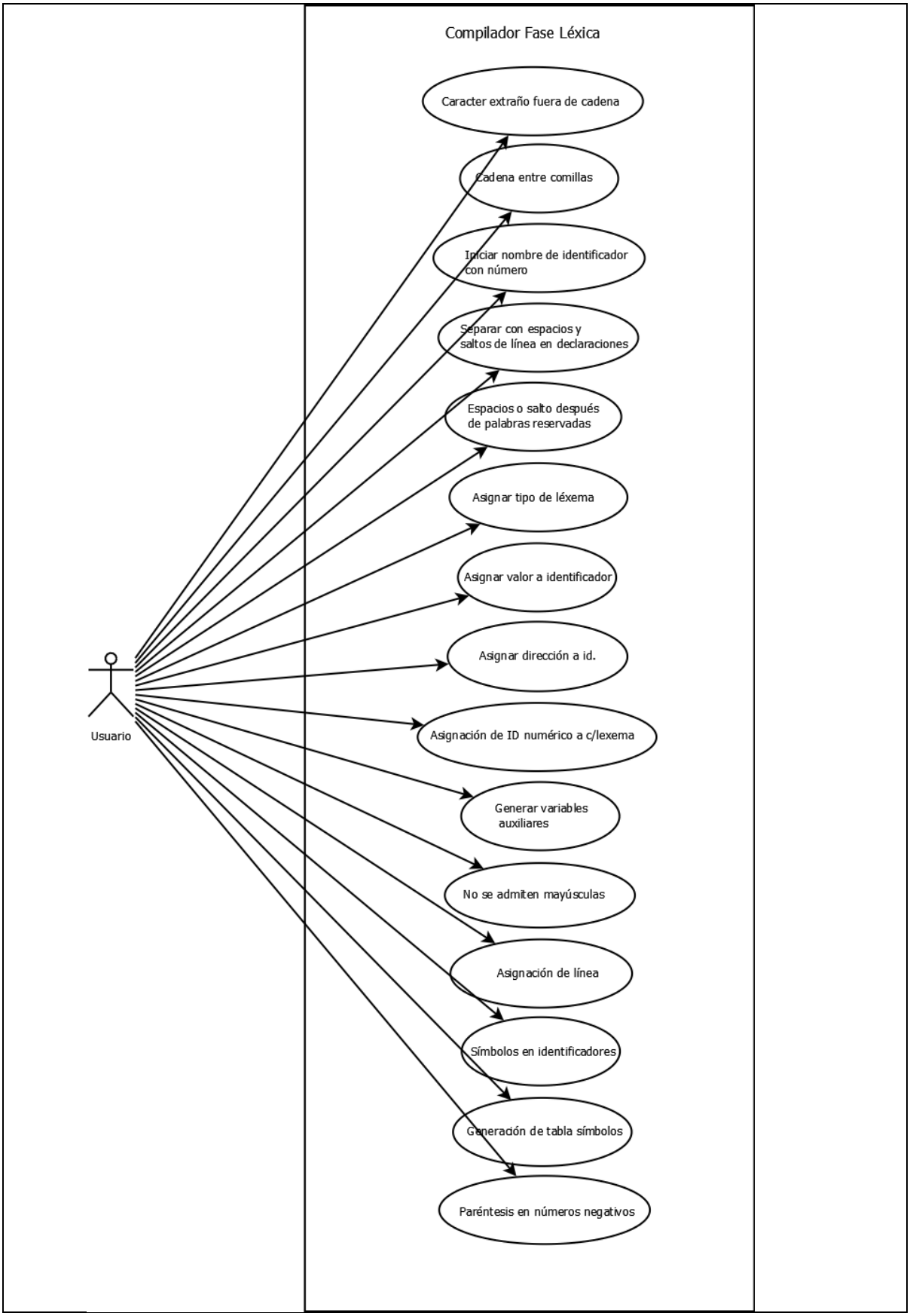
```
/**
 * @author Ramirez García Juana Rubi
 * @author Valle Rodriguez Julio Cesar
 */
package compilador;
public class Datos {

    int getDatos(String p_Palabra){
        int v_Index=0;
        if (m_entQ0(p_Palabra,0)>0){
            v_Index=m_entQ0(p_Palabra,0);
        }else{
            if(m_cadQ0(p_Palabra,0)>0)
                v_Index=m_cadQ0(p_Palabra,0);
        }
        return v_Index;
    }

    int m_entQ0(String p_Palabra,int p_Indice){
        int v_Index=p_Indice;

if(p_Palabra.charAt(p_Indice)>=48&&p_Palabra.charAt(p_Indice)<=57){
            v_Index=m_entQ1(p_Palabra,v_Index+1);
        }
        return v_Index;
    }

    int m_entQ1(String p_Palabra,int p_Indice){
```

```
        int v_Index=p_Indice;
        if (p_Palabra.length()-1>=p_Indice){

if(p_Palabra.charAt(p_Indice)>=48&&p_Palabra.charAt(p_Indice)<=57){
                v_Index=m_entQ1(p_Palabra,v_Index+1);
            }
        }
        return v_Index;
    }

    int m_cadQ0(String p_Palabra,int p_Indice){
        int v_Recorrido=0;
        if(p_Palabra.length()-1>=p_Indice){
            if(p_Palabra.charAt(p_Indice)==""){
                v_Recorrido=m_cadQ1(p_Palabra,1);
            }
        }
        return v_Recorrido;
    }

    int m_cadQ1(String p_Palabra,int p_Indice){
        int v_Indice=p_Indice;
        if (p_Palabra.length()-1>=p_Indice){
            if(p_Palabra.charAt(p_Indice)!=""){
                v_Indice=m_cadQ1(p_Palabra,v_Indice+1);
            }
            else{
                return v_Indice+1;
            }
        }
        else{
            return -1;
        }
        return v_Indice;
    }
}
```

5	Anexos

Diagrama




Compilador Fase Léxica
Caracter extraño fuera de cadena
Cadena entre comillas
Iniciar nombre de identificador con número
Separar con espacios y saltos de línea en declaraciones
Espacios o salto después de palabras reservadas
Asignar tipo de léxema
Asignar valor a identificador
Asignar dirección a id.
Asignación de ID numérico a c/lexema
Generar variables auxiliares
No se admiten mayúsculas
Asignación de línea
Símbolos en identificadores
Generación de tabla símbolos
Paréntesis en números negativos
Usuario

Control de versiones

Versión	Detalles	Fecha
1.1	Implementación de analizador léxico.	14/10/2016
1.2	Optimización de analizador léxico aplicando autómatas para el reconocimiento y clasificación de tokens.	28/10/2016

Bitácora

5 REFERENCIAS

- http://icteoriadeautomatas.blogspot.mx/2010/10/elementos-que-conforman-un-automata.html
- https://es.wikipedia.org/wiki/Teor%C3%ADa_de_aut%C3%B3matas
- http://pendientedemigracion.ucm.es/info/pslogica/automatas.pdf