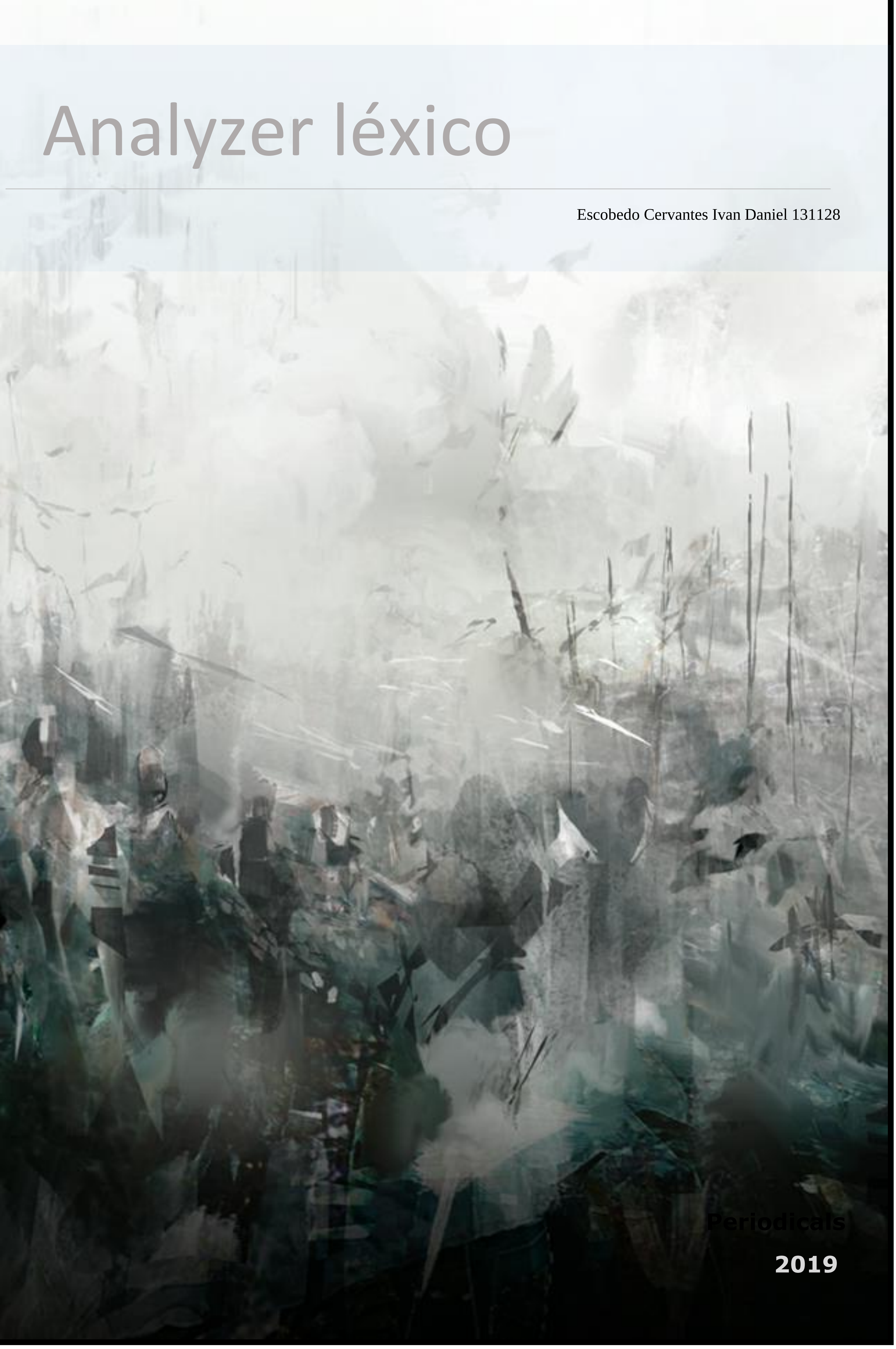
Analyzer léxico

Escobedo Cervantes Ivan Daniel 131128



Periodicals

2019

Lexical Analyzer Report

Autor: Escobedo cervantes Iván Daniel 131128

***Summary*— These instructions are a basic guide to programming a lexical analyzer, programmed in the Java language, which describes how it works, such as the logic implemented.**

The summary is limited to more than 150 words and contending male XLRs equations, figures, tables, or references' Concealment to enunciate that it was done, as was done, main results, and its transcendence.

I. Introduction

SAnd began, as a first step, to establish the logic of the lexical analyzer, using the Java programming language, for this was the construction of a finite deterministic machine (AFD) and two finite non-deterministic machines (AFND), which will be exposed.
To carry out the implementation of the three AFD, it supports the JFLAP program, and then it is implemented in the Java high-level programming language.
Making use of its resources and classes that offers this language, thus facilitating its implementation.
Then it will be shown in table 1, as is the body of the program in Java

TABLE I
Cuerpo del programa

Clase	Funcion
Analizador	es el main del programa, es el inicio del léxico
interfaz	Es el cuerpo del analizador léxico
léxico	Implementación de los ADF Y AFND
fRAMEWORK	entorno GRAFICO DEL PROGRAMA

Tabla 1 Table body of lexicon programming

Explication about the sections of this document.

Next, it will expose how the C++ language works and what is implants in the lexical analyzer. As well as explain the machines AFD, with their respective tables to continue exposing the operation of the lexical analyzer in Java, explaining its functionality for both the user and the programmer

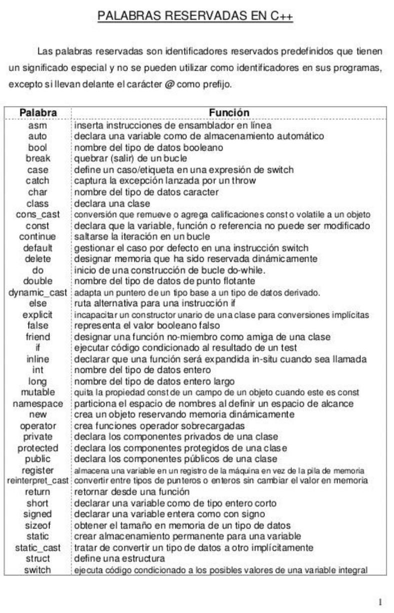
II. C++

A. *Bodi and Reserve words*

To begin, we had to do a preliminary investigation of the language to Lexicar which in this case is **C++,** for that was carried out a basic search of its operation and structure.

Para eso se dividio la búsqueda en los siguientes puntos.

- Logical operators
- Aritmeticos operators
- Relational operators
- Reserved signs
- Reserved words



PALABRAS RESERVADAS EN C++

Las palabras reservadas son identificadores reservados predefinidos que tienen un significado especial y no se pueden utilizar como identificadores en sus programas, excepto si llevan delante el carácter @ como prefijo.

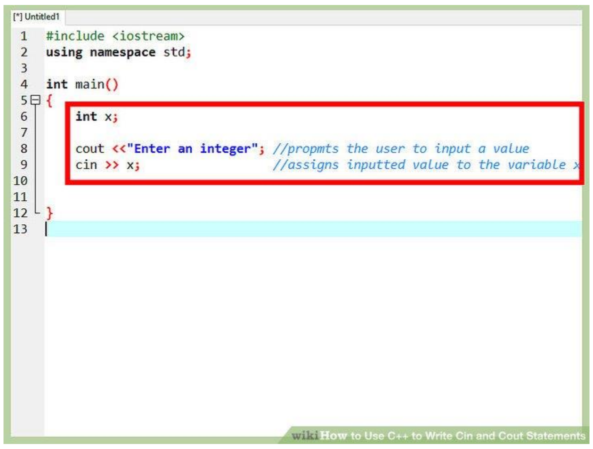
Palabra	Función
asm	inserta instrucciones de ensamblador en línea
auto	declara una variable como de almacenamiento automático
bool	nombre del tipo de datos booleano
break	quebrar (salir) de un bucle
case	define un caso/etiqueta en una expresión de switch
catch	captura la excepción lanzada por un throw
char	nombre del tipo de datos caracter
class	declara una clase
cons_cast	conversión que remueve o agrega calificaciones const o volatile a un objeto
const	declara que la variable, función o referencia no puede ser modificado
continue	saltarse la iteración en un bucle
default	gestionar el caso por defecto en una instrucción switch
delete	designar memoria que ha sido reservada dinámicamente
do	inicio de una construcción de bucle do-while.
double	nombre del tipo de datos de punto flotante
dynamic_cast	adapta un puntero de un tipo base a un tipo de datos derivado.
else	ruta alternativa para una instrucción if
explicit	incapacitar un constructor unario de una clase para conversiones implícitas
false	representa el valor booleano falso
friend	designar una función no-miembro como amiga de una clase
if	ejecutar código condicionado al resultado de un test
inline	declarar que una función será expandida in-situ cuando sea llamada
int	nombre del tipo de datos entero
long	nombre del tipo de datos entero largo
mutable	quita la propiedad const de un campo de un objeto cuando este es const
namespace	particiona el espacio de nombres al definir un espacio de alcance
new	crea un objeto reservando memoria dinámicamente
operator	crea funciones operador sobrecargadas
private	declara los componentes privados de una clase
protected	declara los componentes protegidos de una clase
public	declara los componentes públicos de una clase
register	almacena una variable en un registro de la máquina en vez de la pila de memoria
reinterpret_cast	convertir entre tipos de punteros o enteros sin cambiar el valor en memoria
return	retornar desde una función
short	declarar una variable como de tipo entero corto
signed	declarar una variable entera como con signo
sizeof	obtener el tamaño en memoria de un tipo de datos
static	crear almacenamiento permanente para una variable
static_cast	tratar de convertir un tipo de datos a otro implícitamente
struct	define una estructura
switch	ejecuta código condicionado a los posibles valores de una variable integral

1

Illustration 1 Reserved words

[*] Revista Argentina de Trabajos Estudiantiles. Patrocinada por la IEEE.

B. *Body c++*



```
#include <iostream>
using namespace std;

int main()
{
    int x;

    cout <<"Enter an integer"; //propmts the user to input a value
    cin >> x;                  //assigns inputted value to the variable

}
```

Ilustración 2 body c++

As you can see the structure of the C++ language, where in the analyzer, the "include" the "#", "< >" got into the reserved signs section, the "CIN" and the "cout" would come as C++ reserved words,, the "int", the "X", would come to be words belonging to C and C++. It explains all this because it implantations in the AFD, Separate and analyzing by chain.

C. *Logic implementación*

The machines AFD and AFND will be exposed, which according to the structure of the C++ language, is divided.

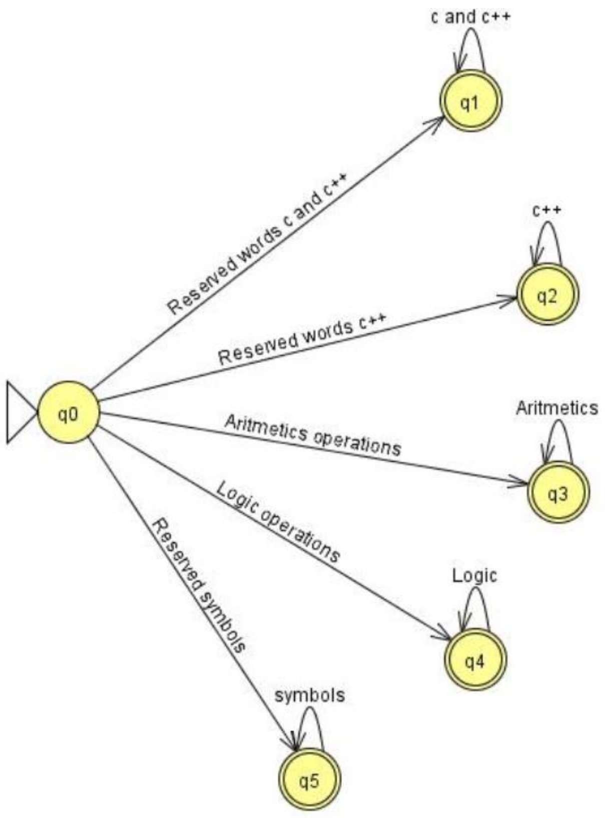


Illustration 3 AFD

As you can see the structure of the AFD, it is basic, siendo A single initial state, towards 5 valid states, validating the reserved words, signs, logical, arithmetical, relational, if it does not enter any of those categories, then it may belong to a declaration of a variable, or real numbers as Floating.

D. *varaibles*

Para poder validar las variables, o identificadores, se tuvo que crear un AFDN, que validara que sea un identificador valido, como se puede observar a continuación.

int variable1, variable2 etc….
(1)

Esas variables como se puede observar pueden tener un digito en la parte final, pero no pueden tener otro signo en a parte inicial de la declaración de la variable.

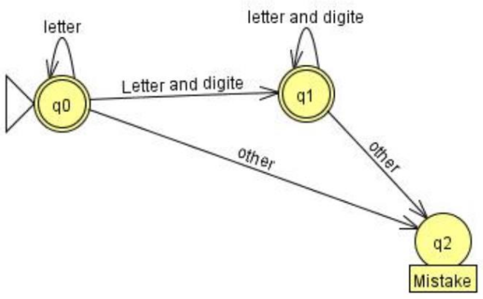


Illustration 4 Variables

As can be observer, in the state q0 a valid and initial state is initiated, that if continua with letter or digit will pass to a state Q1, that in form recursive is Hira filling.

NOTE: It is only valid if starts with a letter of the alphabet, concerted with a digit as shown in the following ennal regular price.

E®= $[A-Z]^{+}$ * [O-9]

If you start with another sign, you will go straight to the Q2 state, where it will be invalid.

A continuación, se expondrá la tabla de verdad de este autómata.

	Letter	Number	Other Symbols
Q0	Q0	Q1	Q2
Q1	Q1	Q1	Q2
Q2	Q2	Q2	Q2

Illustration 5 TABLE VARIABLES

E. Numbers

To be able to implanter the numbers, an AFN was created, where it was validated if they are real or floating, as shown in the following automata.

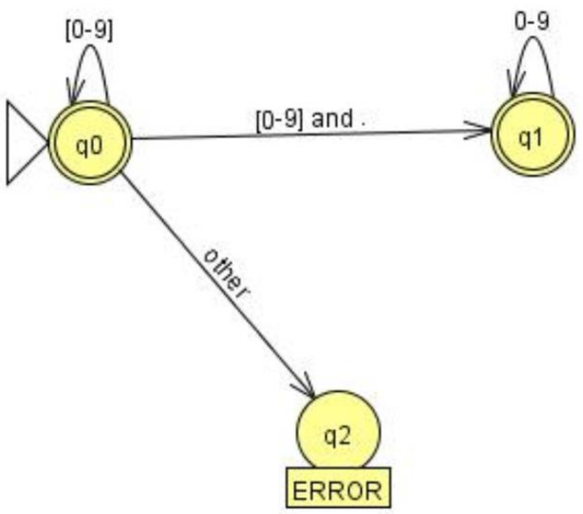


Illustration 6 Number integer and floating

As you can observe in the illustration 6, in Qo, you should start with number, validating so all the integers, if a point arises, either at the beginning or in the middle of the chain, it will take you to a Q1 state.

If you iniciate with any other symbol will take you to the state Q2, which in this case is invalid. A continuation se expander la table of true

	Letter	Number	Other Symbols
Q0	Q0	Q1	Q2
Q1	Q1	Q2	Q2
Q2	Q2	Q2	Q2

Illustration 7 Table integers and floating

Expression regular.

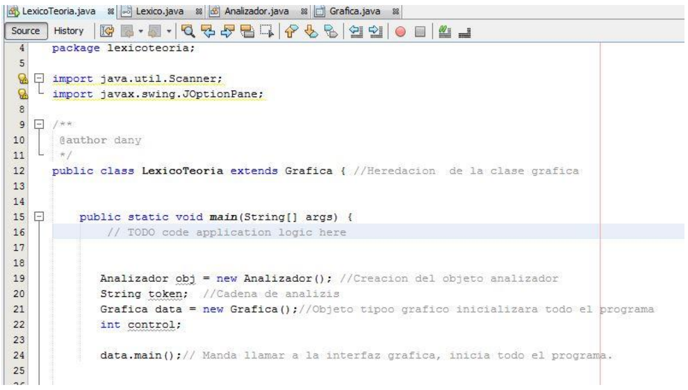
E®= $[0\text{-}9]^{+}$ * . * $[O\text{-}9]^{+}$

III. MANUAL PROGRAMING

In the next section you will see how it was programmed in Java, having a structure of 4 classes, described below:

- Lexicon Theory: Main Program
- Lexicon: interface type, is the skeleton of the program
- Analyzer: AFD and AFND implementation
- Graphic: Graphic environment of the program.

As explained in the preceding section the class **LexicoTeoria**, is the main of the program invokes the graphical method **,** which is where the strings are entered. See Figure 8.

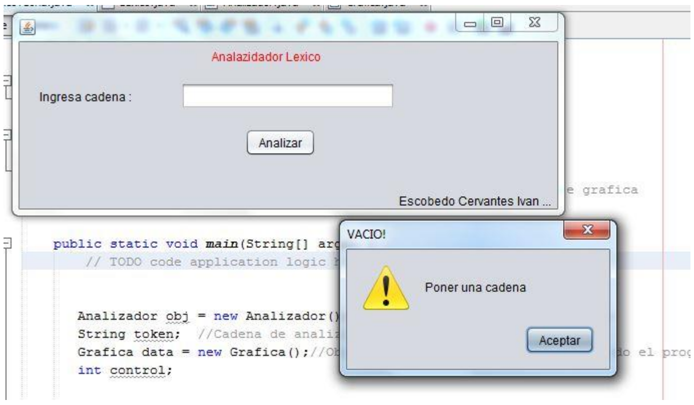


```
package lexicoteoria;

import java.util.Scanner;
import javax.swing.JOptionPane;

/**
 @author dany
 */
public class LexicoTeoria extends Grafica { //Heredacion  de la clase grafica


    public static void main(String[] args) {
        // TODO code application logic here


           Analizador obj = new Analizador(); //Creacion del objeto analizador
           String token;  //Cadena de analizis
           Grafica data = new Grafica();//Objeto tipoo grafico inicializara todo el programa
           int control;

           data.main();// Manda llamar a la interfaz grafica, inicia todo el programa.

```

Ilustración 8 Main

Then in the Java graphical environment, the **swing** library was used, which is the Java framework, where the string is entered, the **graphic** class saves the string "Token" and sends it to the Analyzer class **,** where it entered Al AFD or AFDN.

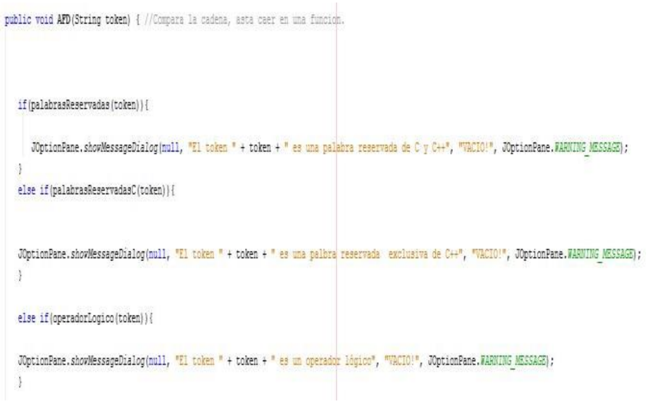
It should be noted that the input of the chain is validated for empty chains, "Landa", as shown in the following figure.



```
    public static void main(String[] ar
        // TODO code application logic

           Analizador obj = new Analizador()
           String token;   //Cadena de anali
           Grafica data = new Grafica();//Ob
           int control;
```

Illustration 9 Landa

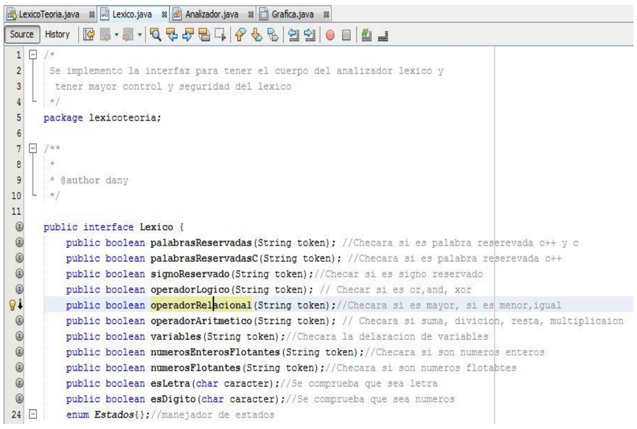
When you enter the analyzer, you pass the touch, reaching the main class, **AFD.**



```
public void AFD(String token) { //Compara la cadena, asta caer en una funcion.



    if(palabrasReservadas(token)){

        JOptionPane.showMessageDialog(null, "El token " + token + " es una palabra reservada de C y C++", "VACIO!", JOptionPane.WARNING_MESSAGE);
    }
    else if(palabrasReservadasC(token)){


    JOptionPane.showMessageDialog(null, "El token " + token + " es una palbra reservada  exclusiva de C++", "VACIO!", JOptionPane.WARNING_MESSAGE);
    }

    else if(operadorLogico(token)){

    JOptionPane.showMessageDialog(null, "El token " + token + " es un operador lógico", "VACIO!", JOptionPane.WARNING_MESSAGE);
    }
```

Illustration 10 AFD MAIN

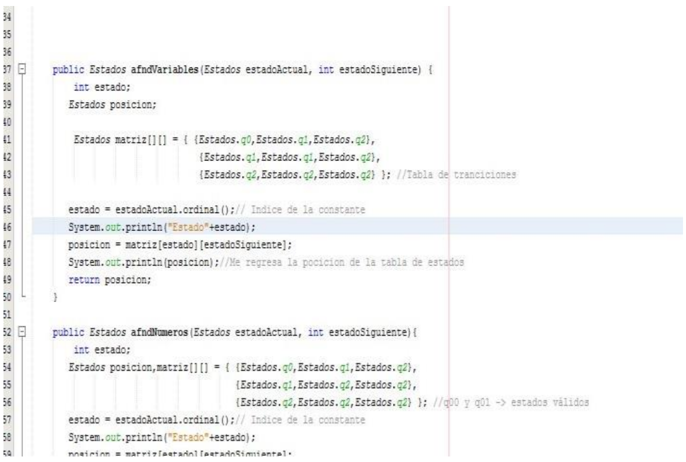
All the methods Implantations in this analyzer class, are implemented by means of an interface, "Lexico" that overwrites all the methods, see Figure 11.



```java
/*
 Se implemento la interfaz para tener el cuerpo del analizador lexico y
  tener mayor control y seguridad del lexico
 */
package lexicoteoria;

/**
 *
 * @author dany
 */

public interface Lexico {
    public boolean palabrasReservadas(String token); //Checara si es palabra reserevada c++ y c
    public boolean palabrasReservadasC(String token); //Checara si es palabra reserevada c++
    public boolean signoReservado(String token);//Checar si es signo reservado
    public boolean operadorLogico(String token); // Checar si es or,and, xor
    public boolean operadorRelacional(String token);//Checara si es mayor, si es menor,igual
    public boolean operadorAritmetico(String token); // Checara si suma, divicion, resta, multiplicaion
    public boolean variables(String token);//Checara la delaracion de variables
    public boolean numerosEnterosFlotantes(String token);//Checara si son numeros enteros
    public boolean numerosFlotantes(String token);//Checara si son numeros flotabtes
    public boolean esLetra(char caracter);//Se comprueba que sea letra
    public boolean esDigito(char caracter);//Se comprueba que sea numeros
    enum Estados{};//manejador de estados
```

Ilustración 11 Lexico

After he comes in, the AFD. We used the Automatas AFD and AFDN, for the Lexicar the token, to carry it out, you use the truth table of Tranciciones, as you can see in Figure 12.



```java
    public Estados afndVariables(Estados estadoActual, int estadoSiguiente) {
         int estado;
        Estados posicion;

         Estados matriz[][] = { {Estados.q0,Estados.q1,Estados.q2},
                                {Estados.q1,Estados.q1,Estados.q2},
                                {Estados.q2,Estados.q2,Estados.q2} }; //Tabla de tranciciones

        estado = estadoActual.ordinal();// Indice de la constante
        System.out.println("Estado"+estado);
        posicion = matriz[estado][estadoSiguiente];
        System.out.println(posicion);//Me regresa la pocicion de la tabla de estados
        return posicion;
    }

    public Estados afndNumeros(Estados estadoActual, int estadoSiguiente){
         int estado;
        Estados posicion,matriz[][] = { {Estados.q0,Estados.q1,Estados.q2},
                                        {Estados.q1,Estados.q2,Estados.q2},
                                        {Estados.q2,Estados.q2,Estados.q2} }; //q00 y q01 -> estados válidos
        estado = estadoActual.ordinal();// Indice de la constante
        System.out.println("Estado"+estado);
        posicion = matriz[estado][estadoSiguiente];
```

Ilustración 12 Table of true

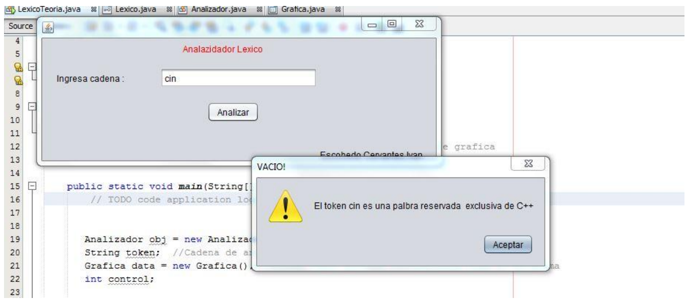
As can be observer is used a matrix of transition to be able to carry out, the movement of States, said movement, is returned, and compares if it fell in a valid state, returning a Boolean.

IV. USER MANUAL

Just analyze the previous section, which strings we can analyze, for example, if you use a reserved word, in this case "CIN", is accepted.

It will show you a message as shown in Figure 13, that the chain has been accepted, depends on what you have entered is where it is sent, the output of the automaton:

1. Chain: will be sent to the automaton in Figure 3
2. Number: If actual or floating number is sent to the automaton in Figure 6
3. Variable: If variable, enter the automaton in Figure 4.



Analazidador Lexico

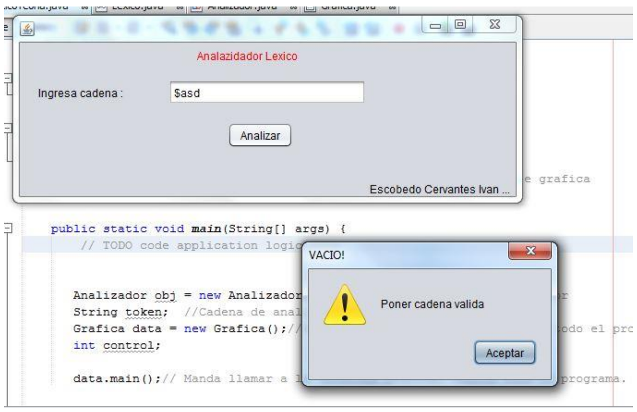
Ingresa cadena : cin

VACIO!

El token cin es una palbra reservada exclusiva de C++

Ilustración 13 CIN

Otherwise, if a wrong string is entered, the error message will be thrown as shown in Figure 14.



Analazidador Lexico

Ingresa cadena : $asd

Escobedo Cervantes Ivan ...

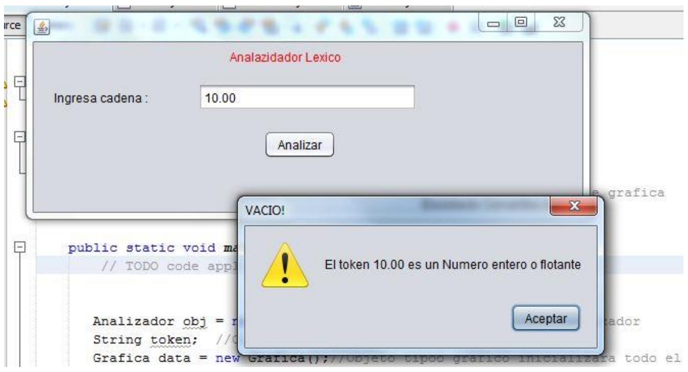
VACIO!

Poner cadena valida

Ilustración 14 Error

In this case you enter a wrong string because it starts with a symbol that does not go according to the C++ syntax.

You can also enter numbers, whether integers or decimals, as shown in Figure 15.



Analazidador Lexico

Ingresa cadena : 10.00

VACIO!

El token 10.00 es un Numero entero o flotante

Ilustración 15 Number

Only enough to visualize which strings belong to C++ and not, the user is not going to battle, because they are only input of strings, or variables or numbers belonging to the Lexicon of C++.

V. BIBLIOGRAFÍA

Dueñas, E. S. (s.f.). Compiladores e intérpretes. En E. S. Dueñas, *Compiladores e intérpretes.*

Louden, K. C. (1997). Construcción de compiladores: principios y práctica. En K. C. Louden, *Construcción de compiladores: principios y práctica.*