Contenido

[Introducción 2](#_Toc535912345)

[Objetivo General 2](#_Toc535912346)

[Objetivo Particular 2](#_Toc535912347)

[Desarrollo 2](#_Toc535912348)

[Conclusiones 2](#_Toc535912349)

[Caja Negra (Requisitos funcionales y no funcionales) 2](#_Toc535912350)

[Ciclomatica 2](#_Toc535912351)

[Grafos 2](#_Toc535912352)

[Cocomo 2](#_Toc535912353)

[Apéndices 2](#_Toc535912354)

[Casos de Uso 2](#_Toc535912355)

[Diagrama de Clases 2](#_Toc535912356)

[Acrónimo 2](#_Toc535912357)

# Introducción

El presente reporte tiene como objetivo mostrar la documentación necesaria para la realización de la primera parte del proyecto que es el Análisis Léxico. Para ello primeramente mostraremos los objetivos de la actividad, después tendremos capturas de pantalla del programa funcionando el cual será programado en c++. Finalmente presentamos la ingeniería de software que consiste en los requisitos la ciclomática diagramas como grafos, clases y casos de uso así como la estimación de costos con Cocomo 1.

# Objetivo General

Como objetivo general en esta práctica se tiene el deseo de elaborar un analizador léxico que reconozca una gramática en especial, la cual esta misma será definida por los creadores dejando a su imaginación la cantidad de reglas gramaticales necesarias para la creación de dicho analizador, teniendo como mínimo 20 reglas gramaticales y siendo extendidas hasta donde la imaginación del programador lo llevara.

# Objetivo Particular

Como objetivo particular se tiene la inspiración de poder crear un analizador programado léxico que nos permita detectar las palabras ingresadas y que el analizador las reconozca como algún objeto en especial, un identificador, una variable, un carácter, cualquier cosa que sea previamente establecida por el programador.

Esta práctica en particular nos ayudará a comenzar a entender el funcionamiento de un compilador comenzando desde lo más simple, las reglas gramaticales, las reglas en cuanto a lo léxico que se debe tener así como las bases establecidas y fundamentadas que ayudarán a tener un programa sólido y estable que pueda ser fácilmente desarrollado para que no genere problemas en un futuro al avanzar en dicho compilador, evitando así una pérdida de tiempo innecesaria gastando recursos valiosos en la corrección de errores.

# Desarrollo

Para el desarrollo se necesitó primordialmente de tener establecido un lenguaje que será la base de todo el futuro proyecto, en este caso fue implementado con c++ usando la técnica de archivos de estructura de datos y de programación orientada a objetos para tener una mejor relación respecto a la información que se iba a almacenar, se utilizará como compilador principal “CodeBlocks” ya que es un entorno de programación muy estable y de los más avanzados que ayudan a la generación de código con compatibilidad.

Analizador Léxico:

1 #include <cstring>

2

3 #include "AnalizadorLexico.h"

4

5 #define FILENAME "Database/lexico.txt"

6

7 **using namespace std**;

8

9 AnalizadorLexico::AnalizadorLexico(){

10 listData = **new** Collection<Lexico>;

11 readFromDisk();

12 }

13

14

15 **std**::**string** AnalizadorLexico::getStringToAnalize() {

16 **return** stringToAnalize;

17 }

18

19 **void** AnalizadorLexico::setStringToAnalize(**std**::**string**& aux) {

20 stringToAnalize = aux;

21 }

22

23 **void** AnalizadorLexico::receiveString() {

24 }

25

26 **void** AnalizadorLexico::analizeString() {

27 Lexico lexico;

28 Tools\* tool;

29 **string** aux = stringToAnalize;

30 **int** i(0);

31 **int** qActual(0);

32 **bool** exit(**false**);

33 aux = tool->strToCapLet(aux);

34 **do**{

35 **switch**(qActual){

36

37 **case** 0:

38 //cout <<"estado 0"<< endl;

39 **if**(('A' <= aux[i] && aux[i] <= 'Z') || ('0' <= aux[i] && aux[i] <= '9') ||

40 aux[i] == '"' || aux[i] == '\'' || aux[i] == ';' || aux[i] == ',' ||

41 aux[i] == '(' || aux[i] == ')' || aux[i] == '{' || aux[i] == '}' ||

42 aux[i] == '!' || aux[i] == '=' || aux[i] == '+' || aux[i] == '\*' ||

43 aux[i] == '-' || aux[i] == '$' || aux[i] == '#' || aux[i] == '<' ||

44 aux[i] == '>'){

45 qActual = 1;

46 }

47 **else if**(aux[i] == '\0'){

48 qActual = 2;

49 }

50 **else**{

51 qActual = 3;

52 }

53 **break**;

54 **case** 1:

55 //cout <<"estado 1"<< endl;

56 **if**(('A' <= aux[i] && aux[i] <= 'Z') || ('0' <= aux[i] && aux[i] <= '9')) {

57 qActual = 1;

58 }

59 **else if**(aux[i] == '\0'){

60 qActual = 2;

61 }

62 **else** {

63 qActual = 3;

64 }

65 **break**;

66 **case** 2:

67 //cout <<"estado 2"<< endl;

68 **cout** <<"DescripciÃ³n: ";

69 lexico.setName(aux);

70 **if**(listData->findData(lexico) != **nullptr**){

71 lexico = listData->getData(listData->findData(lexico));

72 **cout** <<lexico.getDescription()<< **endl**;

73 exit = **true**;

74 }

75 **else**{

76 **cout** <<"Es una variable"<< **endl**;

77 //qActual = 3;

78 exit = **true**;

79 }

80 **break**;

81 **case** 3:

82 **cout** <<"Error, esto no existe en la gramÃ¡tica"<< **endl**;

83 exit = **true**;

84 **break**;

85 **default**:

86 **cout** <<"Error"<< **endl**;

87 exit = **true**;

88 }

89 i++;

90 } **while**(!exit);

91

92 }

93

94 **void** AnalizadorLexico::readFromDisk() {

95 //fstream readerFile;

96 Lexico lexico;

97 **fstream** readerFile;

98 **string** str1, str2, str3;

99 readerFile.open(FILENAME, ios::in);

100 **if**(!readerFile.is\_open()){

101 **cout** <<"Error, archivo no abierto para lectura TABOP"<< **endl**;

102 **return**;

103 }

104 **while**(!readerFile.eof()){

105 getline(readerFile,str1,'Â°');

106 getline(readerFile,str2,'Â°');

107 getline(readerFile,str3,'Â°');

108 **if**(!readerFile.eof()){

109 lexico.setId(atoi(str1.c\_str()));

110 lexico.setName(str2);

111 lexico.setDescription(str3);

112 listData->insertData(lexico);

113 }

114

115 }

116 }

117

Main:

1 #include <iostream>

2 #include "AnalizadorLexico.h"

3

4 **using namespace std**;

5

6 **int** main()

7 {

8 setlocale(LC\_CTYPE,"Spanish");

9 AnalizadorLexico\* analizador = **new** AnalizadorLexico();

10 **string** str;

11 **while**(1){

12 **cout** <<"Diga la cadena: ";

13 getline(**cin**, str);

14 analizador->setStringToAnalize(str);

15 analizador->analizeString();

16 }

17 **return** 0;

18 }

Clase Léxico:

1 #ifndef LEXICO\_H\_INCLUDED

2 #define LEXICO\_H\_INCLUDED

3

4 #include <iostream>

5 #include <cstring>

6

7 **class** Lexico {

8 **private**:

9 **int** id;

10 **std**::**string** name;

11 **std**::**string** description;

12

13 **public**:

14 **int** getId();

15 **std**::**string** getName();

16 **std**::**string** getDescription();

17

18 **void** setId(**const int**& idL);

19 **void** setName(**std**::**string**& nameL);

20 **void** setDescription(**std**::**string**& descriptionL);

21

22 **std**::**string** toString();

23 };

24

25 #endif // LEXICO\_H\_INCLUDED

# Conclusiones

Se obtuvo un programa el cual nos empezó a describir el uso y la etapa inicial de un compilador, inicializando por el analizador léxico que nos detecte que todas las palabras escritas sean reconocidas por una gramática en específico y de esta manera sean válidas para comenzar la compilación y la traducción a lenguaje máquina.  
Resultó bastante peculiar la creación de este programa ya que, aunque se tenían conocimientos previos en cuanto a la función de dicho analizador, se empezó a obtener una mejor visión respecto a lo que realmente hacía un compilador en el fondo, lo que nadie ve pero ahí está, funcionando y ayudándonos a crear magia con un par de teclas.

# Caja Negra (Requisitos funcionales y no funcionales)

# Ciclomática

# Grafos

# Cocomo

# Apéndices

# Casos de Uso

# Diagrama de Clases

# Acrónimo