

**FACULTAD DE INGENIERÍA Y ARQUITECTURA**

**CARRERA: “INGENIERÍA EN SISTEMAS COMPUTACIONALES”**

**CURSO :**

Compiladores y Lenguajes de Programación

**TEMA :**

“Diseño e implementación de un Analizador Léxico-Sintáctico para un lenguaje de programación de alto nivel”

**DOCENTE :**

Castillo Diestra, Carlos Enrique

**INTEGRANTES :**

|  |  |
| --- | --- |
| Hernández Pérez, Thania | zhany91@hotmail.com |
| Moreno Núñez, Jimmy | kalip\_kalit\_500@hotmail.com |
| Rivera Pérez, Moisés | moy\_572@hotmail.com |

**TRUJILLO – PERÚ**

**2013**

**INDICE**

1. **INTRODUCCION 3**
2. **DESARROLLO 4**
3. **Analizador Léxico 4**

**a.1.** lexico2.l 4

**a.2.** parser2.h 6

**b. Analizador Sintáctico 7**

**b.1.** parser2.c 7

**c. Diseño del Lenguaje 7**

**c.1.** parser2.y 7

**d. Diseño de Interfaz 9**

**e. Casos de Prueba 10**

1. **CONCLUSIONES 11**
2. **RECOMENDACIONES 12**
3. **INTRODUCCION**

En el siguiente proyecto de investigación tras analizar los requerimientos y características mínimas para la creación del analizador, el equipo tomo como lenguajes guías a los siguientes lenguajes de programación Prolog los cuales fueron de gran ayuda para la creación de nuestros token´s iniciales dando así el inicio al desarrollo gramatical, léxico y sintáctico requeridos.

1. **DESARROLLO**

En los siguientes puntos se podrán ver el avance del proyecto según sus iteraciones y pasos realizados por el grupo.

1. **Analizador Léxico**

**a.1. lexico2.l**

%{

#include <stdio.h>

#include <conio.h>

int yystopparser=0;

%}

%option noyywrap

%option yylineno

ignora " "|\t|\n

digito [0-9]+

letra [A-Za-z]

iden [a-zA-Z]([a-zA-Z]|[0-9]|\_|" ")\*

cadena '({letra}|{digito}|" ")+'

%%

{ignora}\* {;}

{iden} {return (ID);}

{digito} {return (ENTERO);}

{digito}"."{digito} {return (REAL);}

{cadena} {return (CADENA);}

"write" {return (ESCRIBIR);}

"read" {return (LEER);}

"IF" {return (SI);}

"THEN" {return (ENTONCES);}

"ELSE" {return (ELSE);}

"DO" {return (DO);}

"<" {return ('<');}

"/" {return ('/');}

"(" {return ('(');}

")" {return (')');}

"." {return (ENDLINE);}

. {printf("En linea %d: ERROR: Caracter ilegal\n", yylineno);}

%%

yyerror(char \* msg)

{

printf ("%s\n", msg);

}

int main(void)

{

char NomArch[30];

printf("Ingrese archivo a procesar:");

gets(NomArch);

yyin=fopen(NomArch, "rt");

if (yyin==NULL)

printf("\n No se puede abrir el archivo:");

else

yyparse();

fclose(yyin);

getch();

return(0);

}

**a.2. parser2.h**

/\* Tokens. \*/

#ifndef YYTOKENTYPE

# define YYTOKENTYPE

/\* Put the tokens into the symbol table, so that GDB and other debuggers

know about them. \*/

enum yytokentype {

ID = 258,

ENDLINE = 259,

SI = 260,

ENTONCES = 261,

COMPARAR = 262,

MAYORIGUAL = 263,

MENORIGUAL = 264,

LEER = 265,

ESCRIBIR = 266,

ELSE = 267,

ENTERO = 268,

REAL = 269,

CADENA = 270,

DO = 271

};

#endif

#if ! defined YYSTYPE && ! defined YYSTYPE\_IS\_DECLARED

typedef int YYSTYPE;

# define YYSTYPE\_IS\_TRIVIAL 1

# define yystype YYSTYPE /\* obsolescent; will be withdrawn \*/

# define YYSTYPE\_IS\_DECLARED 1

#endif

extern YYSTYPE yylval;

1. **Analizador Sintáctico**

**b.1. parser2.c**

Referencia al archivo en el CD

1. **Diseño del Lenguaje – gramática**

**c.1. parser2.y**

%token ID ENDLINE SI ENTONCES COMPARAR MAYORIGUAL MENORIGUAL

%token LEER ESCRIBIR ELSE ENTERO REAL CADENA DO

%start inicio

%%

inicio : listas

;

listas : nombre '(' listNombre ')' sentencias ENDLINE nuevalista

;

casoContrario : resp2

| ')'

;

resp2 : ELSE elemento ',' elemento ')'

;

elemento : leer

| escribir

;

operador : '>'

| '<'

| '\*'

| '-'

| '/'

| '+'

| COMPARAR

| MAYORIGUAL

| MENORIGUAL

;

nuevalista : otroList

|

;

otroList : listas

;

listNombre : listNombre ',' nombre

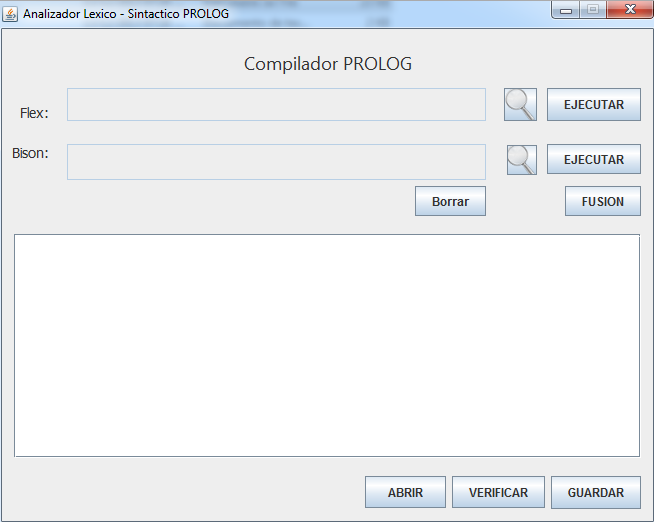
| nombre

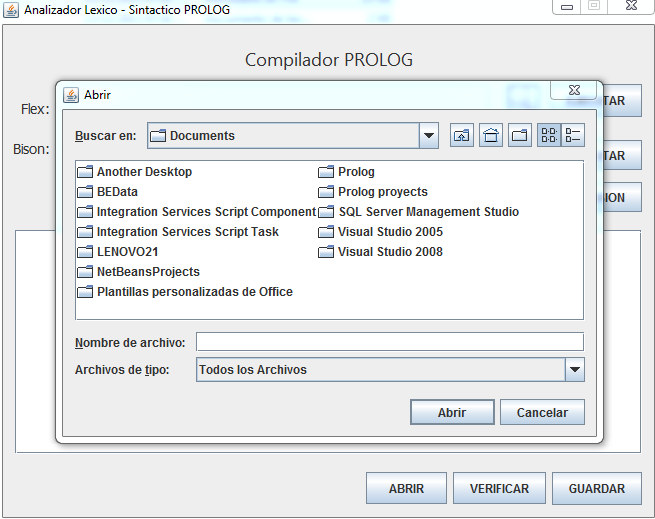
;

1. **Diseño de la Interface**

La interfaz de nuestro proyecto ha sido creada con el afán de optimizar y simplificar el generador de archivos para los usuarios, los cuales son necesarios para el desarrollo de este proyecto (en el modo consola). Para poder ejecutar el analizador léxico y sintáctico de esta forma, genera los archivos extensión .c y .h y ejecutar el caso de prueba con los botones buscar el cual se puede identificar con el dibujo de una lupa, “Ejecutar “, “Verificar” y “Fusión”.

Además de otros botones que serán usados en la interface como: “Abrir” y “Guardar”.





1. **Casos de Prueba**

**e.1. Correctos**

**Ejemplo 01:**

otro(brasil,peru,argntina,chile,etc).

imprime\_mayor(X,Y) @IF ( X > Y @THEN @\_write(x), @\_read('asd asd 123123') @ELSE @\_read(A), @\_read('ooo lll 111')).

**Ejemplo 02:**

padre(Ciro,jorge).

**Ejemplo 03:**

imprime\_mayor(X,Y) @IF ( X > Y @THEN @\_write(x), @\_read('X es mayor ') @ELSE @\_read(A), @\_read('Y es mayor)).

**e.2. Incorrectos**

**Ejemplo 01:**

imprime\_mayor(X,+,Y) @IF ( X > Y @THEN @\_write(x), @\_read(prueba A') @ELSE @\_read(A), @\_read('prueba B')).

**Ejemplo 02:**

imprime\_mayor(X,Y) @IF ( X > Y @THEN @\_write(x), @\_write('Hola') @ELSE @\_read(A), @\_read('Adios')).

**Ejemplo 03:**

paises\_latinos(brasil,peru,argntina,chile,etc)

1. **CONCLUSIONES**

Con el desarrollo de este proyecto se ha logrado la creación de un analizador léxico-sintáctico el cual reconoce el lenguaje Prolog. Además se ha conseguido desarrollar posibles aplicaciones a partir del analizador con el fin de poder mostrar el potencial de éste como lo es nuestra interfaz simplificada.

1. **RECOMENDACIONES**

Al ser un módulo que aporta los códigos más usados en los lenguajes de programación su funcionabilidad de reconocimiento aun no es tan grande para algunos aspectos, siempre se puede ir añadiendo funcionalidad que se vaya necesitando en cada momento y además otro punto a tomar en cuenta es que nosotros solo nos basamos en el reconocimiento del lenguaje mas no en la función del programa a analizar.