**Universidad Tecnológica Nacional - FRRe**

***Ingeniería en Sistemas de***

***Información***

**Sintaxis y Semántica de los Lenguajes**

***Trabajo Final Integrador***

**Alumnos:**

-Lucas, Dania

-Troncoso, Clarise

**Lugar:** Resistencia, Chaco, Argentina

**Primer cuatrimestre**

**Curso académico:** 2019

**Fecha de entrega:** 15/08

***INDICE:***

***Introducción…………………………………….………...3,4***

***Característica de Lenguaje Pseudocodigo………………..4,5***

***Componentes Léxicos………………………………………6***

***Gramática generada………………………………………6,8***

***Analizador Léxico..………………………………………9,11***

***Analizador Sintáctico……………………………………..12,18***

***Conclusión………………………………………………….19***

***Bibliografía…………………………………………………..19***

Introducción:

Durante la primer entrega desarrollamos la gramática libre de contexto donde dejamos asentado las distintas reglas y producciones previas a la implementación de la solución mediante el software.

Para poder realizar la segunda entrega de este trabajo, utilizamos el generador léxico llamado “Flex” en conjunto con el compilador Dev-c++ versión 5.1.1 y la herramienta CMD (Símbolo del Sistema) para la ejecución del léxico.

La entrega del analizador léxico consta de un archivo “.l” que es compilado/escaneado por el flex mediante la plataforma CMD, donde si este mismo no muestra errores genera un fichero “.c” (lex.yy.c) que contiene la función de análisis ‘yylex()’, varias tablas usadas por esta para emparejar tokens, unas cuantas rutinas auxiliares, macros y su respectivo “.exe” para la ejecución.

El archivo “.l” está formado primeramente por el sector de librerías que necesitamos para el programa, correspondientes al lenguaje C. Luego sigue la estructura del código perteneciente al analizador léxico, ésta comienza por la declaración de comandos necesarios para la formación del mismo (que nos ayudan a controlar), seguido se encuentra la sección de definiciones que contiene declaraciones de definiciones con nombres sencillos para simplificar la especificación del escáner, de la forma “nombre (expresión regular)” que responden a los requisitos del apartado 3 de la consigna asignada. Después sigue la parte de reglas, donde llamamos a las definiciones y se asocia la acción “printf” para emitir el mensaje por pantalla si se ha escaneado y encontrado ese token. Por último, la sección donde se coloca de análisis ‘yylex()’, donde siempre que se llame a ‘yylex()’, este analiza tokens desde el fichero de entrada. La función continúa hasta que alcance el final del fichero (punto en el que devuelve el valor 0) o una de sus acciones ejecute una sentencia return.

Por último, en la realización de la tercera entrega, se realizó un analizador sintáctico utilizando el generador Bison (versión 2.4.1), el cual convierte la descripción formal de un lenguaje, escrita como una gramática libre de contexto, en un programa en C, el cual utiliza la gramática que se elaboró previamente, que se representa por el archivo con extensión “.y” que contiene la especificación de una gramática y genera una función en C que reconoce cadenas validas de la gramática.

El archivo “y.tab.c” está formado por las definiciones de todos los ‘%tokens’ empleando conjuntamente el uso de Flex y Bison, dicho archivo se genera normalmente con el comando “bison nombre\_archivo.y” en la pantalla de un cmd que se ejecuta en la misma carpeta en la que se encuentran los dos programas. Posteriormente si se quiere compilar el archivo creado, se usara la opcion gcc y.tab.c.

Características de lenguaje de pseudocódigo

1. **Componentes léxicos o tokens**

**Palabras reservadas**

# \_mod, \_div, # \_o, \_y, \_no,

# leer, # escribir # si, entonces, sino, fin\_si # mientras, hacer, fin\_mientras # repetir, hasta\_que # para, hasta, fin\_para, # accion, \_es, fin\_accion, proceso, ambiente.

**Observaciones**

# No se distingue entre mayúsculas ni minúsculas.

**Identificadores**

**Características**

Identificadores válidos: # dato, dato\_1, dato\_1\_a

**Tipos de Datos:**

**Número** # Se utilizan números enteros y reales de punto fijo.

**Alfanumérico** # Esta compuesta por una serie de caracteres delimitados por comillas simples: ‘Ejemplo de cadena’ ‘Ejemplo de cadena con salto de línea \n y tabulador \t’

**Operadores**

**Operador de asignación**

#asignación

**Operadores aritméticos**

#suma, #resta, #producto, #división real, #división entera, #módulo, #potencia

**Operadores relacionales de números y cadenas:**

#menor que, #menor o igual que, #mayor que, #mayor o igual, #igual que, #distinto que

**Operadores lógicos**

#disyunción lógica, #conjunción lógica, #negación lógica

1. **Sentencias o Inicio y fin de Algoritmo**

ACCION identificador \_es

Ambiente

Proceso

Sentencias

Fin\_accion

o Lectura Leer (identificador)

o Escritura Escribir (expresión alfanumérica y/o identificadores)

o Sentencias de control

# Sentencia condicional simple

si (condición) entonces

sentencias

fin\_si

# Sentencia condicional compuesta

si (condición) entonces

sentencias

sino

sentencias

fin\_si

# Bucle “mientras”

mientras (condición) hacer

sentencias

fin\_mientras

# Bucle “repetir”

repetir

sentencias

hasta\_que (condición)

# Bucle “para”

para (identificador:=valor\_inicial) hasta valor\_final, incremento hacer

sentencias

fin\_para

**Componentes Léxicos:**

Menorque ,mayorque, menorigual, mayorigual, igual, distinto, dosptos, parentesis, MAS, POR, ASIGNACION, MENOS, DIV, MODULO, DIVENTERA, POT, Y, O, Negacion, espacio, caract, COMILLAS, cadena, comentario, findesentencia, Id2, digito, numENTERO, numREAL, si, entonces, sino, fin\_si, mientras, hacer, fin\_mientras, repetir, hasta\_que, para, hasta, fin\_para, alfanumerico, entero, real, accion, ambiente, proceso.

**Gramatica:**

**∑** → SIGMA

SIGMA → **acción** ID **\_es ambiente tokencomentario** VAR **proceso tokencomentario** SENT **fin\_accion; tokencomentario**

VAR → **ƛ** | ID : TIPO

TIPO → ENT | ALF | REAL

ENT → **entero**

REAL → **real**

ALF → **alfanumérico**

ID → **tokenID**

LETRAS → LETRAS

GUIONB → **\_**

ENTERO → NUM

NUM → **tokennum**

NREAL → NUM PTO NUM

NOT→  **ƛ**  | **\_no**

PTO **→ .**

COMA → **,**

ARIT2 → + ID | - ID | \* ID

INC → **ƛ** | COMA ENTERO

OPLOG → **\_y** | **\_o**

A → **ƛ**  | COMA OPERACIÓN A

CAD → **ƛ** | ID CAD

CADENA → **tokencadena**

OPERADOR→ **≤ | ≥ | < | <> | > | =**

ARITMETICO**→ +| - | \* | \*\* | / | \_div | \_mod**

CON → ENTERO | ID

SENT → **ƛ** | **si** NOT CONDICION **entonces** SENT **fin\_si;** SENT

| **si** NOT CONDICION **entonces** SENT **sino** SENT **fin\_si;** SENT

| **mientras** CONDICION **hacer** SENT **fin\_mientras;** SENT

| **repetir** SENT **hasta\_que** CONDICION; SENT

| **para (** ID **:=** CON **) hasta** CON INC **hacer** SENT **fin\_para;** SENT

| **escribir (**  **‘** CAD **‘ );** SENT

| **escribir (** OPERACIÓNA **);** SENT

| **escribir (** CADENA COMA ID A **);** SENT

| **escribir (** HH **) ;** SENT

| **leer (** ID A **) ;** SENT

| ID MENOS SENT

| ID MAS SENT

| ID **:= (** OPERACIÓN **) ;**SENT

| ID **:=** KK ;SENT

| ID **:=**  BB ;SENT

BB → ID | ENTERO | REAL

KK → ( BB ARITMETICO BB ) | BB ARITMETICO BB

HH → ENTERO ARITMETICO ENTERO | ENTERO ARITMETICO ID

| ID ARITMETICO ENTERO | ID ARITMETICO ID

CONDICION → OPERACIONLOGICA | OPERACIÓNLOGICA OPLOG CONDICION

| ( TERM OPERADOR TERM ) | ( TERM ARITMETICO TERM OPERADOR )

OPERACIÓN → ( TERM)

| ( TERM ARITMETICO OPERACIÓN )

| ( TERM DER ARITMETICO TERM )

| (TERM ARITMETICO TERM OPERADOR TERM )

| ((TERM) ARITMETICO TERM )

| ( TERM OPERADOR TERM OPLOG TERM OPERADOR TERM )

IZQ →  **ƛ** | **(**

DER → **ƛ** | **)**

TERM → ENTERO

| REAL

| CADENA

| NOT ID

OPERACIONLOGICA → IZQ OPERACIÓN OPLOG OPERACIÓN DER

**Analizador Léxico:**

%{

                        /\*SECTOR DE LIBRERIAS DE C\*/

#include <stdio.h>

#include <stdlib.h>

*int* linea = 1;

#include "y.tab.h"

extern YYSTYPE yylval;

extern *char* \*yytext;

extern *int* yylineno;

extern *int* errcont;

#define TOKERROR -1

#define OK 0

%}

                        /\*comandos\*/

%option noyywrap

%option yylineno

%option caseless

                        /\*TOKEN OPERADORES RELACIONALES\*/

menorque                <

mayorque                >

menorigual              <=

mayorigual              >=

igual                   =

distinto                <>

dosptos                 :

parentesis              "("|")"

                        /\*TOKEN OPERADORES ARITMETICOS\*/

MAS                     "+"

POR                     "\*"

ASIGNACION              ":="

MENOS                   "-"

DIV                     "\_div"

MODULO                  "\_mod"

DIVENTERO               "/"

POT                     "\*\*"

                        /\*TOKEN OPERADORES LOGICOS\*/

Y                       "\_y"

O                       "\_O"

Negacion                "\_no"

                        /\*TOKEN CARACTERES\*/

espacio       [ \t]+

caract             [a-zA-Z]+

COMILLAS            \"

cadena             \".\*\"

comentario          "#"{espacio}+\*\n+\*{espacio}+\*{caract}+{espacio}+\*\n+\*{espacio}+\*\n+\*"#"|"@"{espacio}\*{caract}+

Id                  [a-zA-Z]+[0-9]\*|[a-zA-Z]+"\_"{digito}+|[a-zA-Z]+"\_"{digito}+"\_"{caract}+|[a-zA-Z]+"\_"{caract}+

findesentencia          ";"

Id2 [a-zA-Z]([a-zA-Z0-9]|[a-zA-Z0-9])\*

                        /\*TOKEN NUMEROS\*/

digito                  [0-9]

numREAL           [0-9]+"."([0-9])\*

%%

                        /\*REGLAS\*/

"'"   {return SIMPLECOMILLA;}

"(" {return PARIZQ;}

")" {return PARDER;}

">="       {return MAYORIGUAL;}

"<="        {return MENORIGUAL;}

"="    {return IGUAL;}

">"      {return MAYOR;}

"<"      {return MENOR;}

"<>"     {return DISTINTO;}

","       {return COMA;}

":"                     {return TOKDOSPUNTOS;}

"+"                  {return TOKMAS;}

"entonces" {return TOKENTONCES;}

"\*"                   {return TOKMULTIPLICAR;}

":="                 {return TOKASIGNACION;}

"-"                  {return TOKMENOS;}

"\_div"               {return TOKDIV;}

"\_mod"                {return TOKMOD;}

"/"                  {return TOKDIVENTERA;}

"\*\*"                  {return TOKPOTENCIA;}

"real"                  {return REAL;}

"entero" {return ENTERO;}

"alfanumerico"          {return ALF;}

"\_" {return GBAJO;}

"caracter" {return CARACTER;}

[0-9]+               {return TOKENTERO;}

"Mientras"           {return TOKMIENTRAS;}

"Si"                  {return TOKSI;}

"Para"               {return TOKPARA;}

"repetir"            {return TOKREPETIR;}

"ambiente"           {return TOKAMBIENTE;}

"\_es"                   {return TOKES;}

"\_y"                  {return TOKAND;}

"\_o"                   {return TOKOR;}

{cadena} {return CADENA;}

"\_no"                 {return TOKNOT;}

"accion"             {return TOKENACCION;}

"escribir"           {return TOKENESCRIBIR;}

"leer"                {return TOKENLEER;}

"proceso"            {return TOKENPROCESO;}

"fin\_si"                {return TOKFINSI;}

"sino"                  {return TOKENSINO;}

"hacer"                 {return TOKENHACER;}

"fin\_mientras"          {return TOKENFINMIENTRAS;}

"hasta"                 {return TOKENHASTA;}

"fin\_para"              {return TOKENFINPARA;}

"fin\_accion"            {return TOKFINACCION;}

"hasta\_que"             {return TOKHASTAQUE;}

{digito}+"."{digito}+ {return TOKREAL;}

{Id}                   {return IDENTIF;}

{comentario}            {return TOKCOMENTARIO;}

";"          {return PCOMA;}

{espacio}               {}

\n {linea++;}

.                       {printf("Error lexico: '%s', linea numero %i\n", yytext, yylineno); errcont++;}

%%

**Analizador Sintáctico:**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

extern int p=0;

#include "y.tab.h"

extern int j=0;

extern *int* linea;

extern int ubi=0;

*int* yylex();

int errcont=0;

extern *char* \*yytext;

extern *int* yylineno;

extern FILE \*yyin;

*void* yyerror(*char* \**s*);

*void* para();

*char* s;

typedef char matriz[50];

matriz variables[50];

matriz tiposdatos[50];

matriz valor[50];

%}

%start sigma

%token SIMPLECOMILLA

%token PARIZQ

%token TOKPARA

%token PARDER

%token MENORIGUAL

%token MAYORIGUAL

%token MENOR

%token MAYOR

%token IGUAL

%token TOKFINSI

%token COMA

%token DISTINTO

%token TOKDOSPUNTOS

%token TOKMAS

%token TOKENTONCES

%token TOKMULTIPLICAR

%token TOKASIGNACION

%token TOKENSINO

%token TOKENPROCESO

%token TOKMENOS

%token TOKDIV

%token TOKMOD

%token TOKDIVENTERA

%token TOKPOTENCIA

%token TOKREPETIR

%token TOKENTERO

%token CARACTER

%token ALFANUMERICO

%token TOKNUM

%token ENTERO

%token TOKENLEER

%token TOKENESCRIBIR

%token TOKENACCION

%token TOKMIENTRAS

%token TOKES

%token PCOMA

%token TOKSI

%token TOKOR

%token REAL

%token ALF

%token TOKENHACER

%token LETRA

%token GBAJO

%token TOKNOT

%token TOKAND

%token TOKENHASTA

%token TOKENFINMIENTRAS

%token TOKENFINPARA

%token TOKFINACCION

%token TOKHASTAQUE

%token TOKREAL

%token TOKCOMENTARIO

%token IDENTIF

%token TOKAMBIENTE

%token CADENA

%%

sigma: TOKENACCION IDENTIF TOKES TOKAMBIENTE comentarios var TOKENPROCESO comentarios sent TOKFINACCION PCOMA comentarios

;

var:

| xx TOKDOSPUNTOS tipo var

;

xx: IDENTIF {controlarVariable();}

;

tipo: ENTERO | REAL | ALF

;

sent:

| TOKSI not condicion TOKENTONCES sent TOKFINSI PCOMA sent

| TOKSI not condicion TOKENTONCES sent TOKENSINO sent TOKFINSI PCOMA sent

| TOKMIENTRAS condicion TOKENHACER sent TOKENFINMIENTRAS PCOMA sent

| TOKREPETIR sent TOKHASTAQUE condicion PCOMA sent

| TOKPARA PARIZQ IDENTIF TOKASIGNACION TOKENTERO PARDER TOKENHASTA TOKENTERO inc TOKENHACER sent TOKENFINPARA PCOMA sent

| TOKPARA PARIZQ IDENTIF TOKASIGNACION TOKENTERO PARDER TOKENHASTA IDENTIF inc TOKENHACER sent TOKENFINPARA PCOMA sent

| TOKENESCRIBIR PARIZQ CADENA PARDER PCOMA sent

| TOKENESCRIBIR PARIZQ SIMPLECOMILLA cad TOKDOSPUNTOS SIMPLECOMILLA dd PARDER PCOMA sent

| TOKENESCRIBIR PARIZQ SIMPLECOMILLA cad SIMPLECOMILLA dd PARDER PCOMA sent

| TOKENESCRIBIR PARIZQ operacion a PARDER PCOMA sent

| TOKENESCRIBIR PARIZQ hh PARDER PCOMA sent

| TOKENESCRIBIR PARIZQ CADENA COMA IDENTIF a PARDER PCOMA sent

| TOKENLEER PARIZQ IDENTIF a PARDER PCOMA sent

| IDENTIF TOKMAS TOKMAS sent

| IDENTIF TOKMENOS TOKMENOS sent

| IDENTIF TOKASIGNACION PARIZQ operacion PARDER PCOMA sent

| IDENTIF TOKASIGNACION kk PCOMA sent

| IDENTIF TOKASIGNACION bb PCOMA sent

;

comentarios:

| TOKCOMENTARIO PCOMA;

dd:

| COMA IDENTIF

| IDENTIF

;

hh: TOKENTERO aritmetico TOKENTERO

| TOKENTERO aritmetico IDENTIF

| IDENTIF aritmetico TOKENTERO

| IDENTIF aritmetico IDENTIF

;

kk: PARIZQ bb aritmetico bb PARDER

| bb aritmetico bb

;

bb: IDENTIF

| TOKENTERO

| TOKREAL

;

inc:

| COMA TOKENTERO;

oplog: TOKAND

| TOKOR

;

cad:

| IDENTIF cad

;

a:

| COMA operacion a

;

not:

| TOKNOT

;

operador: MAYORIGUAL | MENORIGUAL | IGUAL| MAYOR | MENOR | DISTINTO

;

aritmetico: TOKMAS | TOKMENOS | TOKMULTIPLICAR | TOKDIV | TOKMOD | TOKDIVENTERA | TOKPOTENCIA

;

arit2:

| TOKMAS IDENTIF

| TOKMENOS IDENTIF

| TOKMULTIPLICAR IDENTIF

;

condicion: operacionlogica

| operacionlogica oplog condicion

| PARIZQ term operador term PARDER

| PARIZQ term aritmetico term operador TOKENTERO PARDER

;

operacion:

PARIZQ term PARDER

| PARIZQ term aritmetico operacion PARDER

| PARIZQ term aritmetico term PARDER arit2

| PARIZQ term aritmetico term IGUAL term PARDER

| PARIZQ term PARDER aritmetico term PARDER

| PARIZQ term operador term oplog term operador term PARDER

;

operacionlogica: PARIZQ operacion oplog operacion PARDER

;

term: TOKENTERO | TOKREAL | CADENA | not IDENTIF

;

%%

*void* yyerror(*char* \**s*)

{

printf ("\t\tError sintactico: en linea %i: %s", yylineno, yytext);

    errcont++;

}

*void* controlarVariable()

{

int n = 0;

int m = 0;

while(n<=50)

{

if(strcmp(variables[n],yytext)==0)

{

m++;

}

n++;

}

if(m>0)

{

printf(" ERRORES SEMANTICOS:\n ");

printf(" En la linea %i",linea );

printf(" La variable %s",yytext );

printf(" Ya fue declarada\n\n" );

}

else

{

strcpy(variables[p],yytext);

p++;

}

}

*int* main (*int* *argc*, *char* \**argv*[])

{

if (argc == 2)

{

    yyin = fopen (argv[1], "rt");

    if (yyin == NULL)

    {

        printf ("El archivo %s no se puede abrir\n", argv[1]);

        exit (-1);

    }

}

else

{

printf("\t\t\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t\t \* \*\n");

printf("\t\t\t\t \* BIENVENIDO \*\n");

printf("\t\t\t\t \* -ANALIZADOR SINTACTICO- \*\n");

printf("\t\t\t\t \* \*\n");

printf(" \t\t\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n\n\n");

printf("\t\t\t\tIndique el modo para realizar analisis:\n");

printf("\t\t\t\t1) Manual\n");

printf("\t\t\t\t2) Apartir de un archivo\n");

printf("Su respuesta: \n");

*char* rta;

scanf ("%c",&rta);

switch (rta){

case '1':

{

printf("\n\t\tIngrese codigo a analizar por favor:\n");

yyin=stdin;

break;

}

case '2':

{

printf("\n\t\tIngrese el nombre/ruta del archivo que desea analizar por favor: ");

char string[200];

scanf ("%s", string);

yyin=fopen(string,"rt");

if (yyin == NULL)

   {

       printf ("\n\t\tEl archivo %s no se puede abrir\n", string);

       exit (-1);

   }

break;

}

default: return 0;

}

}

yyparse();

printf("\n\t\tEl analisis sintactico finalizo con %i error(es).\n",errcont);

return 0;

}

**Conclusión:**

Al realizar el trabajo tuvimos complicaciones respecto a la recursión de las cadenas, paréntesis, comas y la utilización de espacios en blanco los cuales eran imprescindibles para avanzar en el parser, ya que al probar el analizador nos devolvía error sintáctico por no definir correctamente dichos caracteres.

A medida que se desarrollaba el trabajo, se acentuaron errores acarreados de la gramática en la primera entrega, por lo cual, nos vimos en la necesidad de modificar dicha gramática para el correcto funcionamiento del analizador sintáctico.

Sin embargo, se logró reducir los errores originados anteriormente, alcanzando el correcto funcionamiento del interprete. Detectando la estructura principal del algoritmo, ya que si falta algún elemento, informa cual es.

**La bibliografía utilizada de ayuda:**

http://webdiis.unizar.es/asignaturas/LGA/material\_2004\_2005/Intro\_Flex\_Bison.pdf

https://www.ecured.cu/Flex