



**Universidad Nacional Autónoma de
México**

Facultad de Ingeniería



Compiladores

Grupo 1

Analizador Léxico en ANSI C

ALUMNOS:

ARGUETA CORTES JAIRO I.

MENDOZA GAYTAN JOSE TRINIDAD

PROFESORA:

ING. LAURA SANDOVAL M.

Lunes 7 de Septiembre de 2009



INDICE

Análisis.....	2
Estudio preliminar.....	2
Planeación.....	3
Propuesta de servicios.....	3
Diseño.....	4
Expresiones regulares y autómatas.....	4
Unión de autómatas.....	6
Autómata sin transiciones épsilon.....	7
Tabla de transiciones.....	8
Desarrollo.....	9
Definición de tokens.....	9
Técnicas de búsqueda e inserción.....	10
Implementación.....	11
Ejecución de programa lex.c.....	26
Explicación.....	28



Analizador léxico en ANSI C

Análisis.

Estudio preliminar.

Objetivo: elaborar un analizador léxico en ANSI C que reconozca los componentes léxicos pertenecientes a las clases debajo descritas.

Lista de requerimientos:

- El programa deberá solo utilizar instrucciones de ANSI C, independientemente de la distribución de C que se emplee para su elaboración.
- Las clases de los componentes léxicos validos para el analizador léxico son:
 - Clase Descripción
 - 0 constantes enteras (incluyendo octales y hexadecimales).
 - 1 identificadores (según lenguaje C).
 - 2 operadores aritméticos (+,-,%,/).
 - 3 operadores de asignación (según lenguaje C).
 - El número de las clases es inamovible.
- El analizador léxico tendrá como entrada un archivo con las palabras que deberá reconocer. Éste fungirá como programa fuente.
- Como delimitador de un componente léxico será uno varios espacios, tabuladores o saltos de línea, así como el inicio de otro componente léxico.
- Cuando detecte un error léxico, deberá seguir el reconocimiento a partir del siguiente símbolo valido.
- El analizador deberá crear la tabla de símbolos con 2 campos: nombre y tipo.
- Los token's contendrán 2 campos.
 - Campo1: la clase (entero de un byte).
 - Campo2: el valor (de acuerdo a las sig. Tablas).

Operador de asig.		Operador arit.	
Operador	valor	Operador	Valor
=	0	+	0
+=	1	-	1
-=	2	%	2
*=	3	/	3
/=	4		
%=	5		
=	6		
&=	7		
>>=	8		
<<=	9		
^=	10		

El valor para el token de cada identificador es la posición dentro de la tabla de símbolos y de las constantes enteras su valor numérico en base 10.



Como resultado, el analizador léxico deberá mostrar el contenido tanto de la tabla de símbolos como de los tokens.

Los errores que vaya encontrando el analizador léxico, los podrá ir mostrando en pantalla o escribirlos en un archivo.

El programa deberá estar documentado.

Planeación.

Se cuenta con un total de 11 días para la entrega del proyecto por lo que la organización quedara de la siguiente manera:

El programa del analizador léxico se dividirá en tres módulos:

- MOD I: creación del autómata que nos definirá los diferentes componentes léxicos.
- MOD II: implementación de la función para generar tokens.
- MOD III; implementación de la función para la creación de la tabla de símbolos.

Distribución de tiempos:

- Tiempo de análisis y diseño. Tomando en cuenta los requerimientos y alcance del proyecto se realizara en un total de 3 días.
- Tiempo de desarrollo. Será el tiempo restante antes de la entrega del proyecto en este caso será un total de 8 días naturales.

Asignación de labores:

Tomando en cuenta el tiempo con el que se cuenta, así como los recurso y alcance del proyecto se organizara de la siguiente manera:

Desarrollador1.- Argueta Cortes Jairo I. (D1)

Desarrollador2.- Mendoza Gaytán José T. (D2)

- ❖ MOD I.- se realizara en un total de 2 días tomando en cuenta que el grado de dificultad es bajo. Se realizara por D1 y D2
- ❖ MOD II.- se asignaran 3 días para la realización de este modulo. Grado de dificultad de bajo a medio. D1 y D2-
- ❖ MOD III.- se le asignara el tiempo restante de desarrollo en este caso un total de 3 días. Grado de dificultad de medio a algo difícil. D1 y D2.

Propuesta de servicios.

La forma en la que se dará solución a los requerimientos presentados será la siguiente:

Se cuenta con un total de 11 días para la entrega del proyecto por lo que las actividades relacionadas con el desarrollo del proyecto se tendrán que adecuar a este No. De días.

Objetivo.- Que la Ing. Laura Sandoval Montaña obtenga el programa de Analizador léxico en tiempo y forma establecidos por la misma, cumpliendo con todos y cada uno de los requerimientos antes establecidos.

Entregables del proyecto.- se realizara la entrega de:

El análisis, diseño y desarrollo así como la forma en la que el programa se puede implementar.



Se entregara además el código fuente del programa así como un archivo ejecutable del mismo, esto se realizara de manera electrónica mediante el correo electrónico y de forma escrita mediante un documento.

Diseño.

Para poder cumplir con los requerimientos especificados en el presente programa se tendrá que partir desde la creación de las gramáticas hasta la realización de un autómata para su posterior programación, veamos:

Expresiones regulares y autómatas.

Constantes enteras en C:

<entero> = <octa>|<hexa>|<deci>

<octal> = 0<digOn><digO>*

<hexa> = 0(x|X)<dighn><digh>*

<deci> = <dign><dig>*|0

<digOn> = 1|...|7

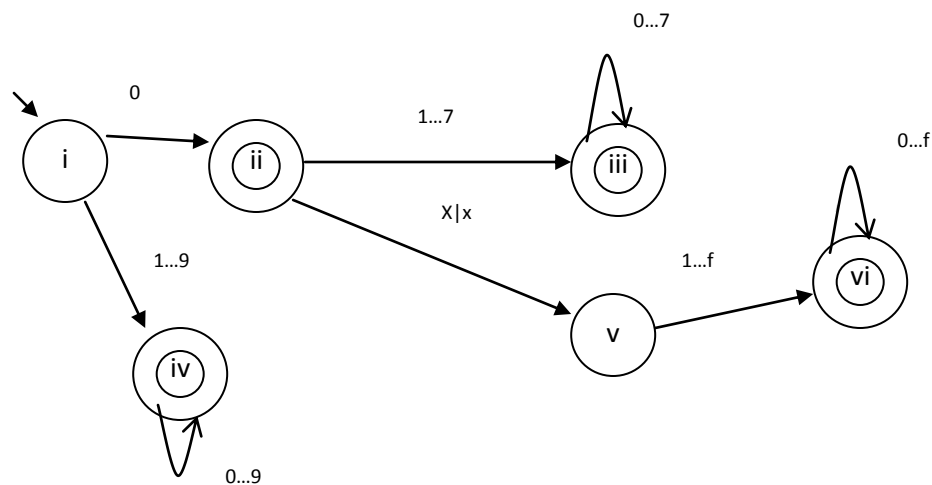
<digO> = 0|<digOn>

<dign> = <digOn>|8|9

<dig> = 0|<dign>

<dighn> = <dign>|A|B...|F|a|b...|f

<digh> = 0|<dighn>



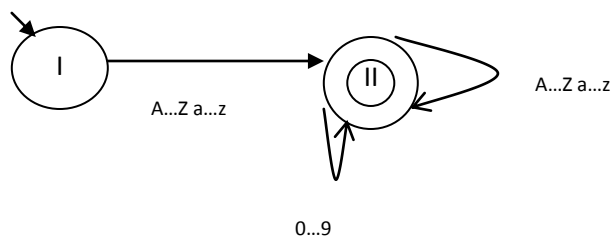


Identificadores según C.

$\langle \text{let} \rangle = _ | A | B | \dots | Z | a | b | \dots | z$

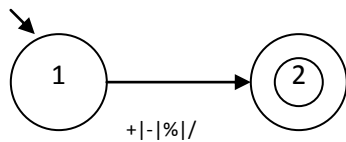
$\langle \text{dig} \rangle = 0 | 1 | \dots | 9$

$\langle \text{ident} \rangle = \langle \text{let} \rangle (\langle \text{let} \rangle | \langle \text{dig} \rangle)^*$



Operadores aritméticos en C.

$\langle \text{opArit} \rangle = + | - | * | \% | /$

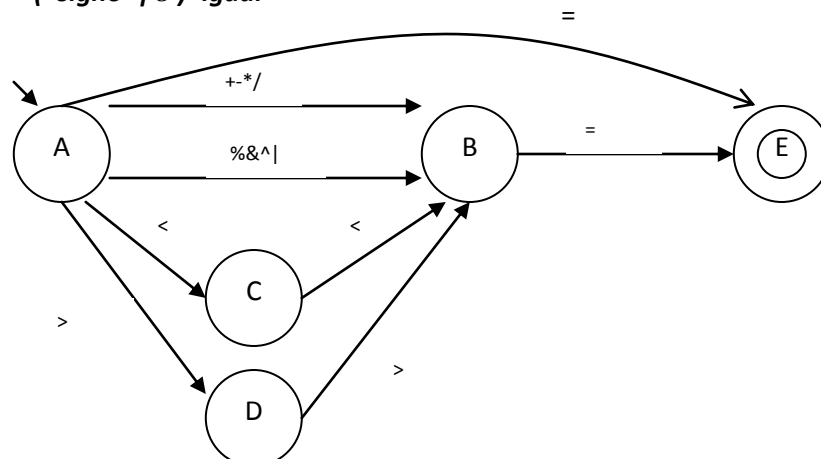


Operadores de asignación según C.

$\langle \text{signo} \rangle = + | - | * | / | \% | \& | | | \langle \langle \rangle \rangle | ^$

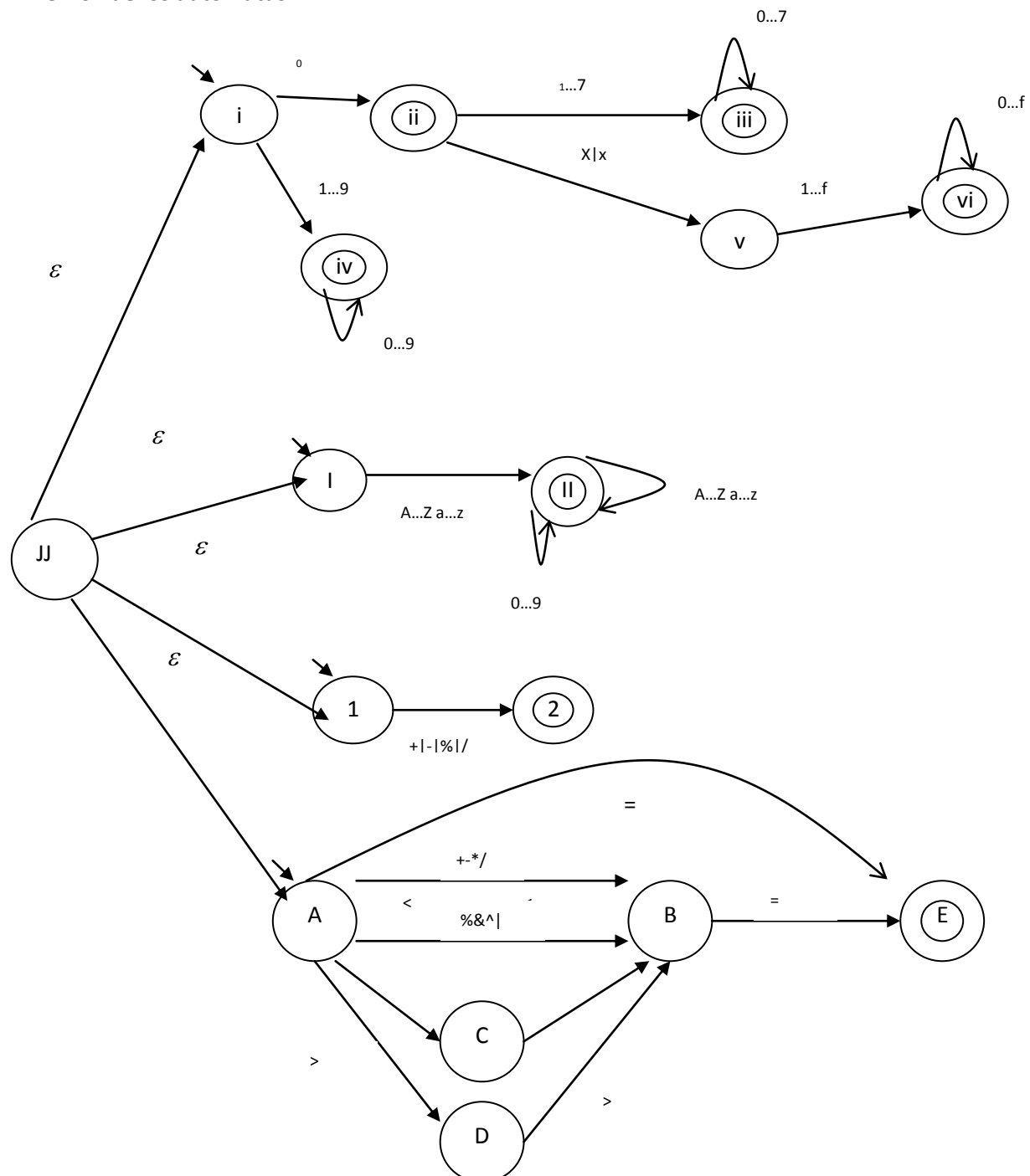
$\langle \text{igual} \rangle = =$

$\langle \text{opAisg} \rangle = (\langle \text{signo} \rangle | \epsilon) \langle \text{igual} \rangle$



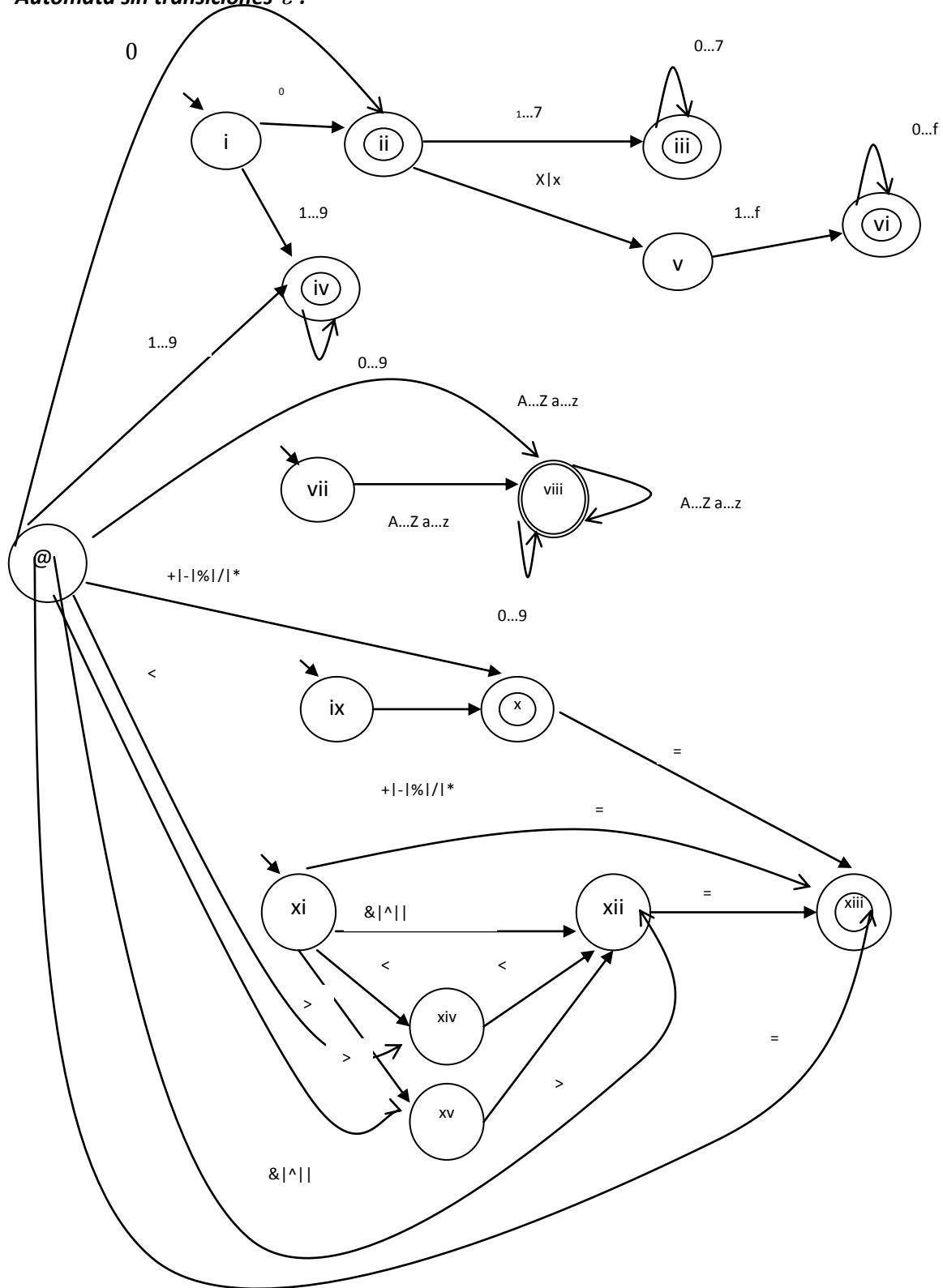


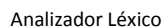
Unión de los autómatas.





Autómata sin transiciones ϵ .





Tablas de transiciones.

[illegible]



Desarrollo.

Definición de la tabla de símbolos

Para trabajar con la tabla de símbolos la definimos de la siguiente manera:

TABLA DE SIMBOLOS		
Clase	Nombre	Valor

Para lo cual trabajamos con una estructura en C. definida de la siguiente manera

```
typedef struct sim {  
    int ClaseS;           //Clase del Símbolo  
    int valorS;           //Valor del símbolo  
    char simbolo[20];     //Símbolo  
    struct sim *siguiente; //Puntero al siguiente símbolo  
} tablaSimbolos;
```

Esta estructura nos permitirá trabajar con listas ligadas, es por eso que trabajamos con `struct sim *siguiente` que es un apuntador al siguiente elemento.

Este tipo de utilización de estructuras de datos nos permitirá optimizar memoria, ya que podríamos haber utilizado una matriz multidimensional de un tamaño fijo, lo que sería difícil de modificar en su tamaño aunque simplificaría mas el código.

Definición de Token's

Cada TOKEN tiene dos campos definidos de la siguiente manera (Clase, Valor)

CLASE 0

Corresponde a constantes enteras incluyendo Octales y Decimales y su valor corresponde a su equivalente en base 10

CLASE 1

Corresponde a Identificadores y su valor corresponde a su posición en la tabla de símbolos. El valor de cada operador lo definimos en C de la siguiente manera

CLASE 2

```
//Estructura para definir Operadores ariteticos {Operador, valor}  
struct opAritmetico {  
    char operador[1]; //Operador  
    int valor;         //Valor  
} operadorArit[4] = {"+", 0}, {"-", 1}, {"%", 2}, {"/", 3};
```

CLASE 3



```
//Estructura para definir Operadores de asignación {operador,
valor}
struct opAsignacion {
    char operador[3]; //Operdor
    int valor;        //Valor
} operadorAsig[11] = {{ "=", 0}, {"+=", 1}, {"-=", 2}, {"*=", 3},
                      {"/=", 4}, {"%=", 5}, {"|=", 6}, {"&=", 7},
                      {">>=", 8}, {"<<=", 9}, {"^=", 10}};
```

Técnicas de búsqueda e inserción

La técnica de búsqueda es lineal para nuestras listas ligadas que viene siendo la tabla de símbolos por lo tanto cuando el autómata reconoce un Identificador, primero lo busca en la tabla de símbolos y si este se encuentra toma el valor correspondiente en la tabla de símbolos y genera en Token.

En caso de que el símbolo no se encuentre en la tabla, lo inserta y le asigna un valor para poder generar el Token correspondiente.

El algoritmo está definido en C de la siguiente manera:

```
void buscaSimbolo(int ClaseB, char *CadenaB) {

    //aux apunta al inicio de la tabla de simbolos
    aux = inicioTabla;
    //Bandera que indica si encuentro el simbolo
    int encontro = 0;

    //Inicializacion de la tabla por primera vez
    if (ContadorS==0)
    {
        insertaSimbolo(ClaseB, CadenaB);
        aux = inicioTabla;
        printf("( %d , %d )\n", aux->ClaseS, aux->valorS);
        encontro=1;
    }
    //Loop que busca un simbolo en la tabla
    while (encontro==0)
    {
        //Si encuentra el simbolo en la tabla
        //imprime clase y valor correspondiente
        if (strcmp(aux->simbolo, CadenaB)==0)
        {
            printf("( %d , %d )\n", aux->ClaseS, aux->valorS);
            encontro=1;
        }
        //Apunta al siguiente elemento de la tabla
        else
            aux = aux->siguiente;
        //Si no encuentra el simbolo lo inserta en la tabla
        if (aux==NULL)
        {

```



```
        insertaSimbolo(ClasaB, CadenaB);  
        aux = inicioTabla;  
        printf("( %d , %d )\n", aux->ClaseS, aux->valorS);  
        encontro=1;  
    }  
}
```

Implementación

Código fuente

```
1  /*  
2  * File:    lex.c  
3  * Descripcion:  
4  * Este es una analizador LEXico escrito en ANSIC  
5  * que reconoce los componentes lexicos  
6  * perteneciente a las clases descritas en el documento  
7  * de especificaciones.  
8  
9  * Autores: Argueta Cortes Jairo I  
10 *          Mendoza Gaytan Jose T.  
11 *  
12 * Fecha de Creacion:29/Ago/2009-4/Sep/2009  
13  
14 * Como utilizar lex.c  
15 * Desde una Terminal de Linux tecleamos:  
16 * 1) gcc lex.c -o gcc  
17 * 2) ./lex "[nombreArchivo].txt  
18  
19 */  
20  
21 //Librerias a utilizar  
22 #include <stdio.h>  
23 #include <stdlib.h>  
24 #include <string.h>  
25  
26 //Declaracion de variables Globales  
27 char car;          //Var. para almacenar caracter de archivo  
28 char caracter[1];  //Var. auxiliar de caracter "car"  
29 char Cadena[20];   //Var. para concatenar caracteres leidos  
30 char Estado = 'A'; //Var. de estados  
31 int EdoAcep = 0;   //Var. de Estadode Acep./Rec.  
32 int Clase;         //Var. del tipo de clase  
33 FILE *codfuente;   //Apuntador a nombre de archivo  
34  
35  
36 //Estructura para definir Operadores aritmeticos  
{operador,valor}  
37 struct opAritmetico {  
38     char operador[1]; //Operador
```



```
39     int valor;           //Valor
40 } operadorArit[4] = {{ "+", 0},{ "-", 1},{ "%", 2},{ "/", 3}};
41
42 //Estructura para definir Operadores de asignacion
{operador,valor}
43 struct opAsignacion {
44     char operador[3]; //Operdor
45     int valor;        //Valor
46 } operadorAsig[11] = {{ "=", 0},{ "+=", 1},{ "-=", 2},{ "*=", 3},
47                      { "/=", 4},{ "%=", 5},{ "|=", 6},{ "&=", 7},
48                      { ">=", 8},{ "<=", 9},{ "^=", 10}};
49
50 //Definicion de Tabla de simbolos
51 //Estructura para trabajar con Listas Ligadas
52 typedef struct sim {
53     int Clases;           //Clase del Simbolo
54     int valorS;           //Valor del simbolo
55     char simbolo[20];     //Simbolo
56     struct sim *siguiente; //Puntero al siguiente simbolo
57 } tablaSimbolos;
58
59 //Inicializacion de punteros a la tabla de Simbolos
60 tablaSimbolos *inicioTabla = NULL, *aux = NULL;
61
62 //Inicializacion del contador de simbolos en la tabla
63 int ContadorS = 0;
64
65 //Declaracion de funciones para trabajar
66 //con el automata
67
68 void automata(char);
69 void error();
70 void generaToken(int, char[]);
71 void limpiaCadena();
72
73 //Declaracion de funciones para trabajar
74 //Con la tabla de simbolos
75 tablaSimbolos *nuevoSimbolo();
76 void insertaSimbolo(int, char*);
77 void buscaSimbolo(int, char*);
78 void errorSim();
79 void imprimeTablaSimbolos();
80
81 //Metodo Main()
82 int main(int argc, char * argv[]) {
83
84     //Apertura del codigo fuente
85     codfuente = fopen(argv[1], "r");
86
87     //Verificador de apertura de archivo
88     if (codfuente == NULL) {
89         printf("Error: el fichero no puede abrirse! \n");
```



```
90     printf("Velva a intentarlo \n");
91 }
92
93 //Ignorar primeros Espacios en blanco y saltos de linea
94 car = fgetc(codfuente);
95 while (car == ' ' || car == '\n') {
96     car = fgetc(codfuente);
97 }
98 printf("\nLISTA DE TOKENS \n\n");
99 //Mientras no se encuentre EOF del archivo
100 while (!feof(codfuente)) {
101
102     //Reinicializacion de Estado de Aceptacion/Rechazo.
103     if (EdoAcep == 1 && car != ' ') {
104         EdoAcep = 0;
105     }
106
107     //Ignorar Espacios en blanco y saltos de linea del
108     //codigo fuente
109     while ((car == ' ' && EdoAcep == 1) || (car == '\n' &&
110     EdoAcep == 0 && Estado == 'A') || (car == ' ' && Estado == 'A')) {
111         car = fgetc(codfuente);
112         EdoAcep = 0;
113     }
114
115     //Llamada a la funcion automata
116     automata(car);
117
118     //Lectura del siguiente caracter dependiendo de su
119     //contexto
120     if (car != ' ' && EdoAcep != 1)
121         car = fgetc(codfuente);
122
123     //Impresion de tabla de simbolos
124     imprimeTablaSimbolos();
125
126     //Cierre de archivo leido
127     fclose(codfuente);
128
129     return 0;
130 }
131
132 //Funcion de error, indica cuando un elemento
133 //no es reconocido por el automata
134 void error() {
135     //Mientras no se encuentre EOF
136     if (!feof(codfuente))
137     {
138         printf("%c", car);
139     }
140 }
```



```
138     printf(" Error: elemento no reconocido!\n");
139     Estado = 'A'; //Inicializa Estado
140     EdoAcep = 0; //Pone en estado de no Aceptacion
141     limpiaCadena(); //borra el caracter leido
142 }
143 }
144
145 //Funcion limpiaCadena
146 void limpiaCadena() {
147     //Limpia cadena
148     int i;
149     for (i = 0; i <= 19; i++)
150         Cadena[i] = NULL;
151 }
152
153 //Funcion automata():
154 //Reconoce las cadenas definidas por
155 //las expresiones regulares.
156 void automata(char car) {
157
158     //Copia el caracter leido a caracter[0]
159     caracter[0] = car;
160
161     //Evalua estado actual
162     switch (Estado) {
163         case 'A':
164
165             //Evalua caracter leido
166             switch (car) {
167                 case '<':
168                     Estado = 'B';
169                     //Concatena cadena: Cadena=Cadena+caracter
170                     strcat(Cadena, caracter);
171                     break;
172
173                 case '>':
174                     Estado = 'C';
175                     strcat(Cadena, caracter);
176                     break;
177
178                 case '^':
179                 case '|':
180                 case '&':
181                     Estado = 'D';
182                     strcat(Cadena, caracter);
183                     break;
184
185                 case '=':
186                     Estado = 'E';
187                     strcat(Cadena, caracter);
188                     break;
189
```



```
190         case '+':
191         case '-':
192         case '*':
193         case '/':
194         case '%':
195             Estado = 'F';
196             strcat(Cadena, caracter);
197             break;
198
199         case '0':
200             Estado = 'G';
201             strcat(Cadena, caracter);
202             break;
203
204         case '1':
205         case '2':
206         case '3':
207         case '4':
208         case '5':
209         case '6':
210         case '7':
211         case '8':
212         case '9':
213             Estado = 'H';
214             strcat(Cadena, caracter);
215             break;
216
217         case '_':
218         case 'a':
219         case 'b':
220         case 'c':
221         case 'd':
222         case 'e':
223         case 'f':
224         case 'g':
225         case 'h':
226         case 'i':
227         case 'j':
228         case 'k':
229         case 'l':
230         case 'm':
231         case 'n':
232         case 'o':
233         case 'p':
234         case 'q':
235         case 'r':
236         case 's':
237         case 't':
238         case 'u':
239         case 'v':
240         case 'w':
241         case 'x':
```




```
242         case 'y':
243         case 'z':
244         case 'A':
245         case 'B':
246         case 'C':
247         case 'D':
248         case 'E':
249         case 'F':
250         case 'G':
251         case 'H':
252         case 'I':
253         case 'J':
254         case 'K':
255         case 'L':
256         case 'M':
257         case 'N':
258         case 'O':
259         case 'P':
260         case 'Q':
261         case 'R':
262         case 'S':
263         case 'T':
264         case 'U':
265         case 'V':
266         case 'W':
267         case 'X':
268         case 'Y':
269         case 'Z':
270             Estado = 'I';
271             strcat(Cadena, caracter);
272             break;
273
274         default:
275             error(); //Estado de no aceptacion o
cadena no valida
276             break;
277     }
278     break;
279     case 'B':
280         switch (car) {
281             case '<':
282                 Estado = 'D';
283                 //Concatena cadena: Cadena=Cadena+caracter
284                 strcat(Cadena, caracter);
285                 break;
286
287             default:
288                 error();
289                 break;
290         }
291         break;
292     case 'C':
```



```
293         switch (car) {
294             case '>':
295                 Estado = 'D';
296                 //Concatena cadena: Cadena=Cadena+caracter
297                 strcat(Cadena, caracter);
298                 break;
299             default:
300                 error();
301                 break;
302         }
303         break;
304     case 'D':
305         switch (car) {
306             case '=':
307                 Estado = 'E';
308                 //Concatena cadena: Cadena=Cadena+caracter
309                 strcat(Cadena, caracter);
310                 break;
311             default:
312                 error();
313                 break;
314         }
315         break;
316     case 'E':
317         Clase = 3;    //Operador de Asignacion
318         Estado = 'A';
319         EdoAcep = 1;
320         generaToken(Clase, Cadena);
321         limpiaCadena();
322         break;
323
324     case 'F':
325         switch (car) {
326             case '=':
327                 Estado = 'E';
328                 //Concatena cadena: Cadena=Cadena+caracter
329                 strcat(Cadena, caracter);
330                 break;
331
332             default:
333                 Clase = 2; //Operador Aritmetico
334                 Estado = 'A';
335                 EdoAcep = 1;
336                 generaToken(Clase, Cadena);
337                 limpiaCadena();
338                 break;
339         }
340         break;
341
342     case 'G':
343         switch (car) {
344             case '1':
```



```
345         case '2':
346         case '3':
347         case '4':
348         case '5':
349         case '6':
350         case '7':
351             Estado = 'J';
352             //Concatena cadena: Cadena=Cadena+caracter
353             strcat(Cadena, caracter);
354             break;
355         case 'x':
356         case 'X':
357             Estado = 'K';
358             strcat(Cadena, caracter);
359             break;
360
361         default:
362             Clase = 0; //Constante entera
363             Estado = 'A';
364             EdoAcep = 1;
365             generaToken(Clase, Cadena);
366             limpiaCadena();
367             break;
368     }
369     break;
370 case 'H':
371     switch (car) {
372         case '0':
373         case '1':
374         case '2':
375         case '3':
376         case '4':
377         case '5':
378         case '6':
379         case '7':
380         case '8':
381         case '9':
382             Estado = 'H';
383             //Concatena cadena: Cadena=Cadena+caracter
384             strcat(Cadena, caracter);
385             break;
386         default:
387             Clase = 0; //Constante entera
388             Estado = 'A';
389             EdoAcep = 1;
390             generaToken(Clase, Cadena);
391             limpiaCadena();
392             break;
393     }
394     break;
395 case 'I':
396     switch (car) {
```



```
397 case '0':
398 case '1':
399 case '2':
400 case '3':
401 case '4':
402 case '5':
403 case '6':
404 case '7':
405 case '8':
406 case '9':
407 case '_':
408 case 'a':
409 case 'b':
410 case 'c':
411 case 'd':
412 case 'e':
413 case 'f':
414 case 'g':
415 case 'h':
416 case 'i':
417 case 'j':
418 case 'k':
419 case 'l':
420 case 'm':
421 case 'n':
422 case 'o':
423 case 'p':
424 case 'q':
425 case 'r':
426 case 's':
427 case 't':
428 case 'u':
429 case 'v':
430 case 'w':
431 case 'x':
432 case 'y':
433 case 'z':
434 case 'A':
435 case 'B':
436 case 'C':
437 case 'D':
438 case 'E':
439 case 'F':
440 case 'G':
441 case 'H':
442 case 'I':
443 case 'J':
444 case 'K':
445 case 'L':
446 case 'M':
447 case 'N':
448 case 'O':
```



```
449         case 'P':
450         case 'Q':
451         case 'R':
452         case 'S':
453         case 'T':
454         case 'U':
455         case 'V':
456         case 'W':
457         case 'X':
458         case 'Y':
459         case 'Z':
460             Estado = 'I';
461             strcat(Cadena, caracter);
462             break;
463         default:
464             Clase = 1; //Identificador
465             Estado = 'A';
466             EdoAcep = 1;
467             generaToken(Clase, Cadena);
468             limpiaCadena();
469             break;
470     }
471     break;
472     case 'J':
473         switch (car) {
474             case '0':
475             case '1':
476             case '2':
477             case '3':
478             case '4':
479             case '5':
480             case '6':
481             case '7':
482                 Estado = 'J';
483                 strcat(Cadena, caracter);
484                 break;
485             default:
486                 Clase = 0; //Constante entera
487                 Estado = 'A';
488                 EdoAcep = 1;
489                 generaToken(Clase, Cadena);
490                 limpiaCadena();
491                 break;
492         }
493     break;
494     case 'K':
495         switch (car) {
496             case '1':
497             case '2':
498             case '3':
499             case '4':
500             case '5':
```



```
501         case '6':
502         case '7':
503         case '8':
504         case '9':
505         case 'a':
506         case 'b':
507         case 'c':
508         case 'd':
509         case 'e':
510         case 'f':
511         case 'A':
512         case 'B':
513         case 'C':
514         case 'D':
515         case 'E':
516         case 'F':
517             Estado = 'L';
518             //Concatena cadena: Cadena=Cadena+caracter
519             strcat(Cadena, caracter);
520             break;
521         default:
522             error();
523             break;
524     }
525     break;
526     case 'L':
527         switch (car) {
528             case '0':
529             case '1':
530             case '2':
531             case '3':
532             case '4':
533             case '5':
534             case '6':
535             case '7':
536             case '8':
537             case '9':
538             case 'a':
539             case 'b':
540             case 'c':
541             case 'd':
542             case 'e':
543             case 'f':
544             case 'A':
545             case 'B':
546             case 'C':
547             case 'D':
548             case 'E':
549             case 'F':
550                 Estado = 'L';
551                 strcat(Cadena, caracter);
552                 break;
```



```
553         default:
554             Clase = 0; //Constante entera
555             Estado = 'A';
556             EdoAcep = 1;
557             generaToken(Clase, Cadena);
558             limpiaCadena();
559             break;
560     }
561     break;
562     default:
563         error();
564         break;
565 }
566 }
567
568 //Funcion generaToken():
569 //Recibe como parametro el valor de la Clase
570 //y la cadena a evaluar.
571 void generaToken(int Clase, char Cadena[]) {
572     //Copia cadena
573     char *CadenaT = Cadena;
574     int ClaseT = Clase;
575
576     //Apuntador para la funcion strtol(Cadena,Apuntador,base)
577     //Permite convertir una cadena a un numero entero
578     char *end;
579
580
581     //Evalua si es Constate Entera y calcula
582     //su valor numerico en base 10
583     if (ClaseT == 0){
584         //Convierte a numeros decimales
585         if(CadenaT[0]!='0'){
586             printf("( %d , %d )\n", ClaseT, atoi(CadenaT));
587         }
588         //Convierte a nueros Hexadecimales
589         else if(CadenaT[1]=='x' || CadenaT[1]=='X'){
590             printf("( %d , %ld )\n", ClaseT,
591                 strtol(CadenaT,&end,16));
592         }
593         //Convierte a numeros Octales
594         else {
595             printf("( %d , %ld )\n", ClaseT,
596                 strtol(CadenaT,&end,8));
597         }
598     }
599     //Evalua si es Identificador
600     else if(ClasseT==1) {
601         //Busca simbolo en la tabla
602         buscaSimbolo(ClasseT, CadenaT);
```



```
602
603     }
604     //Evalua si es Operador Aritmetico
605     else if (ClaseT == 2) {
606         int valor;
607
608         //Busca en la estructura "opAritetico" el valor
609         //del operador leído
610         for (valor = 0; valor <= 3; valor++) {
611             //Compara operadores
612             if (strcmp(CadenaT, operadorArit[valor].operador)
== 0) {
613                 printf("( %d , %d )\n", ClaseT, valor);
614             }
615         }
616     }
617     //Evalua si es Operador de asignacion
618     else if (ClaseT == 3) {
619         int valor;
620         //Busca en la estructura "opAsignacion" el valor
621         //del operador leído
622         for (valor = 0; valor <= 10; valor++) {
623             //Compara operadores
624             if (strcmp(CadenaT, operadorAsig[valor].operador)
== 0) {
625                 printf("( %d , %d )\n", ClaseT, valor);
626             }
627         }
628     }
629 }
630
631 //Funciones para manejar TABLA de Simbolos
632
633 //Funcion que crea un nuevo simbolo
634 //Devuelve un apuntador de tipo tablasimbolos
635 tablaSimbolos *nuevoSimbolo() {
636
637     //Asigna espacio en memoria para nuevos simbolos
638     tablaSimbolos *nuevo = (tablaSimbolos *) malloc(sizeof
(tablaSimbolos));
639
640     //En caso de memoria insuficiente
641     if (!nuevo)
642         errorSim();
643     return nuevo;
644 }
645
646 //Funcion que indica memoria insuficiente
647 void errorSim() {
648     printf("Insuficiente memoria\n");
649     exit(1);

```




```
650 }
651
652 //Funcion que inserta un nuevo simbolo en la tabla
653 void insertaSimbolo(int nClase, char *nCadena) {
654
655     //Define una variable del tipo tablaSimbolos
656     tablaSimbolos *nuevoSim = nuevoSimbolo();
657
658     //Asigna valores
659     nuevoSim->ClaseS = nClase;
660     nuevoSim->valorS = ContadorS;
661     strcpy(nuevoSim->simbolo, nCadena);
662
663     //nuevoSimbolo Apunta al siguiente elemento
664     nuevoSim->siguiente = inicioTabla;
665
666     //inicioTabla apunta al comienzo de la tabla
667     inicioTabla = nuevoSim;
668
669     //Incrementa contador de elementos en la tabla
670     ContadorS = ContadorS + 1;
671
672 }
673
674 //Funcion que busca un simbolo en la tabla.
675 //Recibe como parametros en valor de la clase
676 //y el simbolo a buscar
677 //Si no encuentra el simbolo, lo inserta.
678 //Caso contrario imprime su valor almacenado
679 //en la tabla.
680
681 void buscaSimbolo(int ClaseB, char *CadenaB) {
682
683     //aux apunta al inicio de la tabla de simbolos
684     aux = inicioTabla;
685     //Bandera que indica si encontro el simbolo
686     int encontro = 0;
687
688     //Inicializacion de la tabla por primera vez
689     if(ContadorS==0)
690     {
691         insertaSimbolo(ClasseB, CadenaB);
692         aux = inicioTabla;
693         printf("( %d , %d )\n", aux->ClaseS, aux->valorS);
694         encontro=1;
695     }
696
697     //Loop que busca un simbolo en la tabla
698     while (encontro==0)
699     {
700         //Si encuentra el simbolo en la tabla
701         //imprime clase y valor correspondiente
```



```

702     if (strcmp(aux->simbolo,CadenaB)==0)
703     {
704         printf("( %d , %d )\n", aux->ClaseS, aux->valorS);
705         encontro=1;
706     }
707     //Apunta al siguiente elemento de la tabla
708     else
709     aux = aux->siguiente;
710     //Si no encuentra el simbolo lo inserta en la tabla
711     if (aux==NULL)
712     {
713         insertaSimbolo(ClasaB, CadenaB);
714         aux = inicioTabla;
715         printf("( %d , %d )\n", aux->ClaseS, aux->valorS);
716         encontro=1;
717     }
718 }
719 }
720
721 //Funcion que imprime la tabla se simbolos
722 void imprimeTablaSimbolos() {
723
724     //Define un apuntador de tipo tablaSimbolo
725     tablaSimbolos *aux = NULL;
726
727     //aux apunta al inicio de la tabla de simbolos
728     aux = inicioTabla;
729
730     printf("\n\t\t\t TABLA DE SIMBOLOS");
731     printf("\n-----");
732     printf("\n| Clase | Nombre | Valor |\n");
733     printf("-----");
734
735     //Mientras aux no apunte al final de la tabla
736     while (aux != NULL) {
737         printf("\n| \t\t %d \t %s\t\t %d|", aux->ClaseS, aux->simbolo,
aux->valorS);
738
739         //aux apunta al siguiente simbolo
740         aux = aux->siguiente;
741     }
742
743     printf("\n-----\n\n");
744
745 }
746 }
747

```



Ejecución de Programa lex.c

En una terminal de Linux teclear lo siguiente:

```
boxer@boxer-desktop:~/Escritorio$ gcc lex.c -o lex  
boxer@boxer-desktop:~/Escritorio$ ./lex [nombreArchivo].txt
```

En *[nombreArchivo]* se sustituye por el nombre del archivo fuente, para nuestro caso de ejemplo utilizamos *fuentes.txt*

El archivo fuentes.txt contiene el siguiente texto:

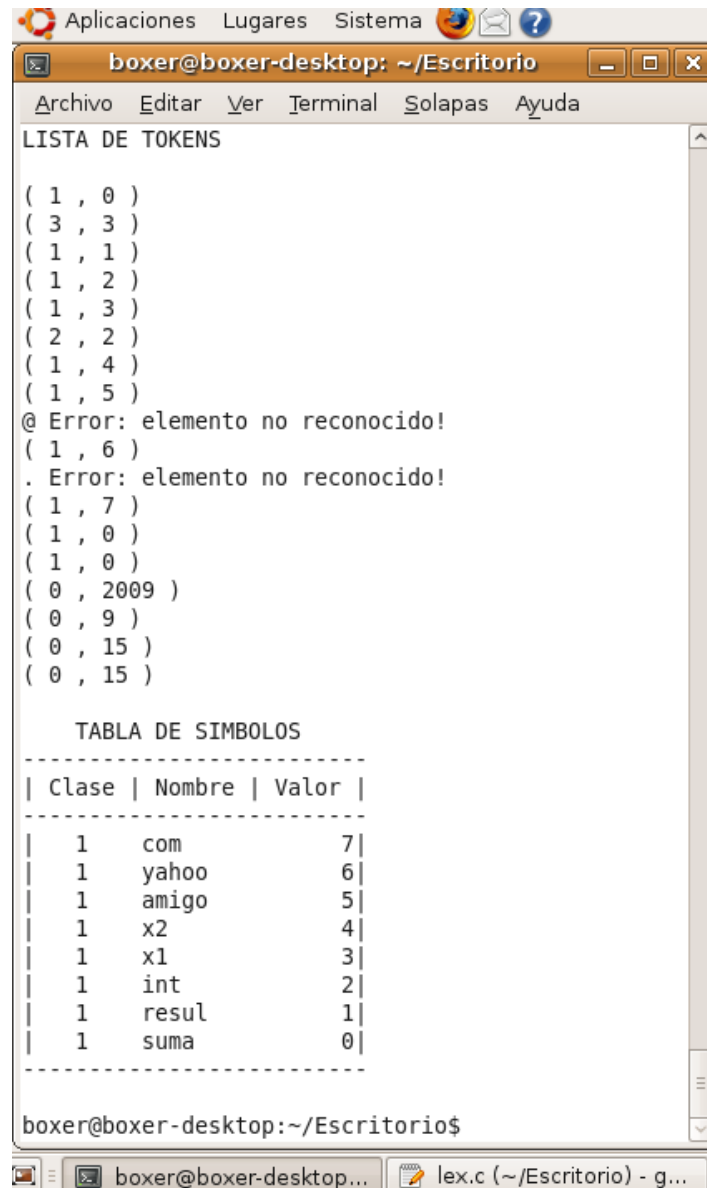
```
suma *=resul  
  
int x1%x2  
  
amigo@yahoo.com
```

Al teclear en consola:

```
boxer@boxer-desktop:~/Escritorio$ gcc lex.c -o lex
```



Observamos lo siguiente:



```
boxer@boxer-desktop: ~/Escritorio
Archivo Editar Ver Terminal Solapas Ayuda
LISTA DE TOKENS
( 1 , 0 )
( 3 , 3 )
( 1 , 1 )
( 1 , 2 )
( 1 , 3 )
( 2 , 2 )
( 1 , 4 )
( 1 , 5 )
@ Error: elemento no reconocido!
( 1 , 6 )
. Error: elemento no reconocido!
( 1 , 7 )
( 1 , 0 )
( 1 , 0 )
( 0 , 2009 )
( 0 , 9 )
( 0 , 15 )
( 0 , 15 )

TABLA DE SIMBOLOS
-----
| Clase | Nombre | Valor |
-----
| 1     | com    | 7     |
| 1     | yahoo  | 6     |
| 1     | amigo  | 5     |
| 1     | x2     | 4     |
| 1     | x1     | 3     |
| 1     | int    | 2     |
| 1     | resul  | 1     |
| 1     | suma   | 0     |
-----

boxer@boxer-desktop:~/Escritorio$
```

Analizando los resultados primero observamos una lista de TOKENS generados por el analizador LEXico, y por ultimo observamos la tabla de símbolos generada por el analizador:



Explicación.

LISTA DE TOKENS

```
( 1 , 0 ) // suma Identificador Clase=1; Valor en la tabla de símbolos=0
( 3 , 3 ) // *= Op. Asignación Clase=3; Valor =3
( 1 , 1 ) // resul Identificador Clase=1; Valor en la tabla de símbolos=1
( 1 , 2 ) // int Identificador Clase=1; Valor en la tabla de símbolos=2
( 1 , 3 ) // x1 Identificador Clase=1; Valor en la tabla de símbolos=3
( 2 , 2 ) // % Op. aritmético Clase=2; Valor =2
( 1 , 4 ) // x2 Identificador Clase=1; Valor en la tabla de símbolos=4
( 1 , 5 ) // amigo Identificador Clase=1; Valor en la tabla de símbolos=5
@ Error: elemento no reconocido!
( 1 , 6 ) // yahoo Identificador Clase=1; Valor en la tabla de símbolos=6
. Error: elemento no reconocido!
( 1 , 7 ) // com Identificador Clase=1; Valor en la tabla de símbolos=7
( 1 , 0 ) // suma Identificador Clase=1; Valor en la tabla de símbolos=0
( 1 , 0 ) // suma Identificador Clase=1; Valor en la tabla de símbolos=0
( 0 , 2009 ) // 2009 Cte. Entera Clase=0; Valor en base10=2009
( 0 , 9 ) // 011 Cte. Entera Clase=0; Valor en base10=11
( 0 , 15 ) // 0xf Cte. Entera Clase=0; Valor en base10=15
( 0 , 15 ) // 0XF Cte. Entera Clase=0; Valor en base10=15
```

TABLA DE SIMBOLOS

	Clase	Nombre	Valor
--	-------	--------	-------

	1	com	7
	1	yahoo	6
	1	amigo	5
	1	x2	4
	1	x1	3
	1	int	2
	1	resul	1
	1	suma	0

boxer@boxer-desktop:~/Escritorio\$