COMPILADORES

Luis Ildefonso Gómez Solana
Fernando Arias Porras
Víctor Gómez Aragoneses

ANALIZADOR LÉXICO

Lenguaje a reconocer

El analizador léxico debe reconocer un subconjunto del lenguaje C++. De este subconjunto habrá que hacer especial énfasis en una parte del mismo, ya que no todo lo especificado en el guión de la práctica es requisito obligatorio.

De esta manera el lenguaje que el analizador léxico reconoce es:
Los elementos que componen una estructura de un programa típico en C++, como por ejemplo la función main.

- Comentarios, tanto // como /* ... */
- Constantes enteras, cadenas de caracteres y lógicas.
- Operadores:
  - Aritméticos: + | - | * | / | %
  - Relacionales: == | != | < | > | <= | >=
  - Lógicos: && | || | !
  - [Auto][In|de]crementales: ++ | --
- Operador de asignación y asignación con operación: [op]=
- Operador de clase.
- Operador condicional.
- Identificadores.

Además, y como parte específica de nuestro compilador, tendremos:

- Tipo de datos:
  - Vectores.
- Sentencia:
  - Condicional simple y compuesta.
  - Asignación.
- Instrucciones de Entrada/Salida:
  - cin, cout.

Por otra parte tenemos las palabras reservadas del mismo lenguaje C++:

main	true	false	char	int	bool	if	else	switch	case	cin
return	do	while	for	void	class	public	private	break	default	cout

Implementación del analizador léxico

El analizador léxico es el primer módulo que nos encontramos en el diseño del compilador. Tiene la característica de que ningún otro módulo volverá a acceder de forma alguna al código fuente, por lo que toda la información que nos pueda resultar necesaria en los módulos siguientes deberá ser recogida por dicho módulo.

Para ser más exactos, el analizador léxico leerá carácter a carácter y gracias a la gramática generada por nuestra herramienta podrá llegar a algún estado final. En este caso se devolverá un Token con toda la información que pueda obtener del mismo; este módulo es el encargado de filtrar ciertos elementos que no son necesarios para la compilación, como por ejemplo los comentarios, tabulaciones, espacios en blanco, nulos... De igual manera y cómo hemos dicho anteriormente habrá que recabar la información necesaria para que en caso de error en el código fuente, poder informar al usuario del lugar de dicho problema.

Por último, añadiremos que el analizador léxico ha sido diseñado utilizando una herramienta para Java conocida como ANTLR.

Tipos de tokens

Para poder interactuar con el analizador sintáctico, se ha diseñado la siguiente codificación de los tokens que devolverá el analizador léxico:

1	<PALABRA_RESERVADA, valor>	Palabra reservada
2	<OP_MAS, “+”>	Operador de suma
3	<OP_MENOS, “-”>	Operador de resta
4	<OP_PRODUCTO, “*”>	Operador de multiplicación
5	<OP_DIVISION, “/”>	Operador de división
6	<OP_MODULO, “%”>	Operador modular
7	<OP_IGUAL, “==”>	Operador de comparación
8	<OP_DISTINTO, “!=”>	Operador distinto
9	<OP_MENOR, “<”>	Operador de comparación menor
10	<OP_MAYOR, “>”>	Operador de comparación mayor
11	<OP_MENOR_IGUAL, “<=”>	Operador de comparación menor o igual
12	<OP_MAYOR_IGUAL, “>=”>	Operador de comparación mayor o igual
13	<OP_AND, “&&”>	Operador lógico AND
14	<OP_OR, “||”>	Operador lógico OR
15	<OP_NOT, “!”>	Operador lógico de negación
16	<OP_REFERENCIA, “&”>	Operador de referenciación
17	<OP_MASMAS, “++”>	Operador de incremento
18	<OP_MENOSMENOS, “--”>	Operador de decremento
19	<DOSPUNTOS, “:”>	Dos puntos
20	<INTER, “?”>	Opearador condicional
21	<OP_ASIG, “=”>	Operador de asignación
22	<OP_ASIG_MAS, “+=”>	Operador de asignación con autosuma
23	<OP_ASIG_MENOS, “-=”>	Operador de asignación con autoresta
24	<OP_ASIG_PRODUCTO, “*=”>	Operador de asignación con autoproducto
25	<OP_ASIG_DIVISION, “/=”>	Operador de asignación con autodivisión
26	<OP_ASIG_MODULO, “%=”>	Operador de asignación con automódulo
27	<PUNTO_COMA, “;”>	Punto y coma
28	<COMA, “,”>	Coma
29	<CORCHETE_AB, “[”>	Abre corchete
30	<CORCHETE_CE, “]”>	Cierra corchete
31	<LLAVE_AB, “{”>	Abre llave
32	<LLAVE_CE, “}”>	Cierra llave
33	<PUNTO, “.”>	Operador de acceso a clase
34	<PARENT_AB, “[”>	Abre paréntesis
35	<PARENT_CE, “]”>	Cierra paréntesis
36	<BARRA_VERT, “|”>	Barra vertical
37	<MENOR_MENOR, “<<”>	Menor menor
38	<MAYOR_MAYOR, “>>”>	Mayor mayor
39	<DOSPUNTOS_DOS, “::”>	Doble dos puntos

Gramática:

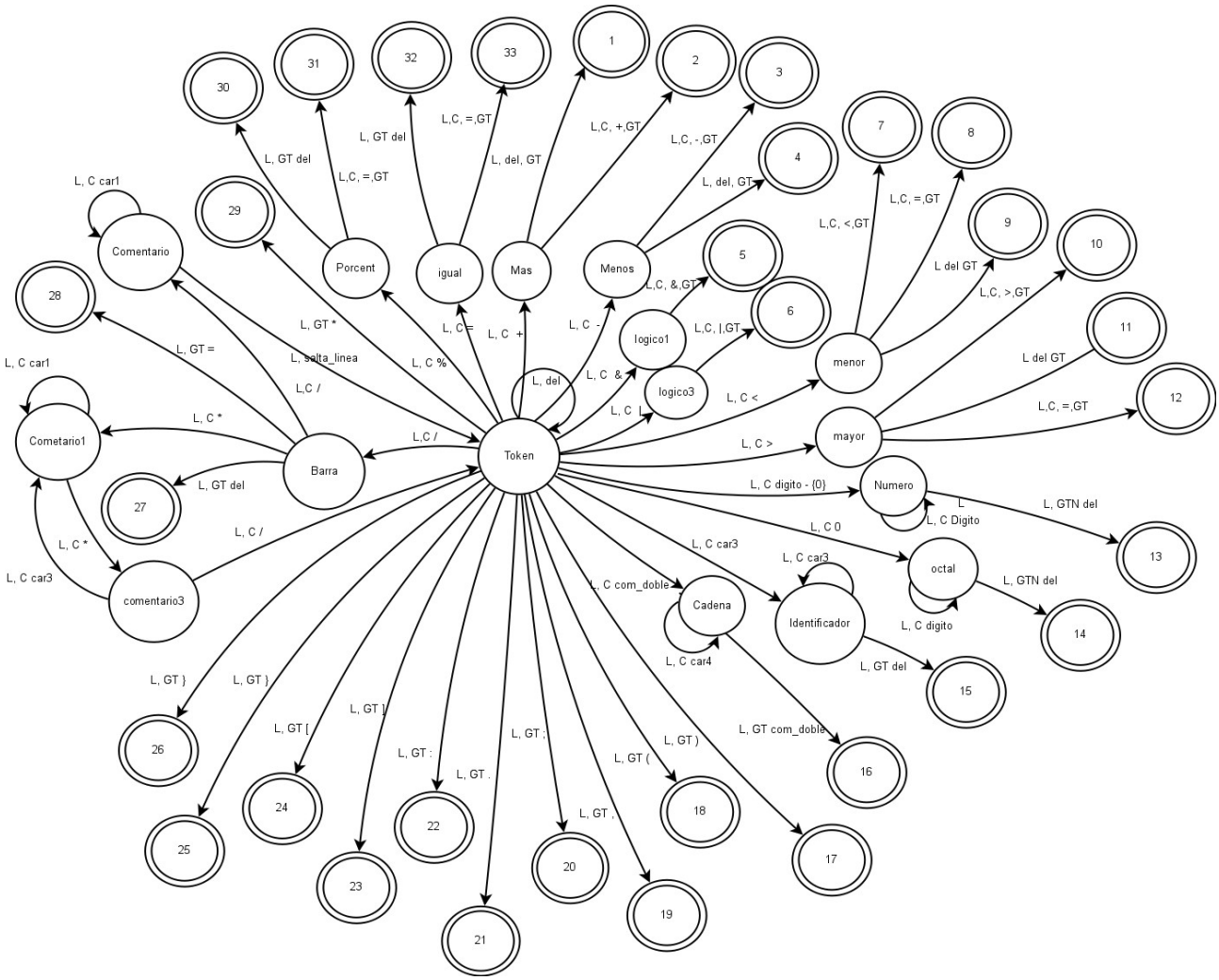
Token => digito0 Numero | 0 Otal|+ Asignacion | - Asignacion | * Asignacion | / Barra | % Asignacion | ! | & Logico1 | | Logico2 | < Menor | > Mayor | = Asignación | car3 Identificador | com_doble Cadena | del Token |, | ;| :|.|(|)| [|]|{|}

Numero := digito Numero | λ
Octal := digito Octal | λ
Asignacion := =| -|+|λ
Barra := = | * Comentario1 | / Barra2|λ
Barra2 := car1 Barra2| salto_linea Token
Comentario1 := letra Comentario1 |digito Comentario1 | car1 Cometario1| *Comentario2
Comentario2 := letra Comentario1 |digito Comentario1 | car2 Cometario1| /
Logico1 := &
Logico2 := |
Menor : = = | < | λ
Mayor : = = | > |λ
Identificador := car3 Identificador | λ
Cadena := car4 Cadena | λ

Donde:

barra_comentario => /
com_doble => "
subrayado => _
del => espacio, \t, \r
salto_linea => \n
letra => cualquier letra del alfabeto
digito => 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
digito0=> digito – {0}
car1 => cualquier_caracter
car3 => cualquier_letra U digito U subrayado
car4 => cualquier caracter - salto_linea – com_dobleAutómata:

Autómata finito determinista



Errores detectados

Error	**Ejemplo**
Cadena mal formada	“hola
Carácter inválido	' # @

Acciones semánticas

L	Leer carácter
C	Concatenar carácter
GTN	Genera token de número Int(lexema)

GT	if (lexema == PAL_RES)
	  GENERA_TOKEN (PAL_RES, lexema)
	else
	  GENERA_TOKEN (ID, valor)

TABLA DE SÍMBOLOS

Diseño

La tabla de símbolos es creada por el primero de los componentes del proceso de compilación, el analizador léxico. En esta tabla se irán creando elementos y actualizando la información de los mismos según se vayan pasando por los diferentes módulos del proceso de compilación.

La tabla de símbolos servirá para almacenar la información dada en el fichero fuente, ya sea de manera global o local (en diferentes tablas) puesto que podemos encontrarnos con varios nombres de variables locales que sean iguales y no debería provocar fallo alguno.

Gracias a la herramienta usada para el desarrollo de la práctica (ANTLR, escrita en Java) el método de almacenamiento ha sido en una tabla hash, lo que nos procurará una mayor velocidad de acceso y agilidad de uso.

La tabla de símbolos principal contendrá los nombres y la información de las variables globales; de los subprogramas almacenaremos los argumentos que recibe y su tipo, junto con una dirección de otra tabla de símbolos creada para dicho subprograma que almacenará la información específica de dicho subprograma.

Con la información almacenada en la tabla de símbolos el compilador podrá generar el código final, ya que conoce toda la información necesaria para ello.

CASOS DE PRUEBA

- Pruebas correctas

Prueba_1.cpp

```
int main(int argc, char *argv[]) {
        cout << "Hola mundo";
        return 0;
}
```

Prueba_2.cpp

```
//Prueba 2

//Probamos las palabras int, return, char, cout
//Identificadores hola, coco
//Tokens ( ) [ ] + ; <<

int hola(char nombre[50])
{
   int coco;
   coco = 5 + 4;
   return coco;
}

int main()
{
   char nombre[50];
   cout << "Cual es tu nombre?: ";
}
```

Prueba_3.cpp

```
//Palabras reservadas void, int, for, cout, cin, return,
//Tokens ( ) [ ] <= = ++ << + ;

void pedir(int matriz[3], int len)
{
   for(int i=0; i<=len; i++)
   {
      cout << "Numero " << i+1 << ":? ";
      cin >> matriz[i];
   }
}

int sumar(int matriz[3], int len)
{
   for(int i=0; i<=len; i++)
      matriz[i++];
   return matriz[3];
}

void mostrar(int matriz[3], int len)
{
   for(int i=0; i<=len; i++)
      cout << matriz[i] << " ";
}

int main()
   return 0;
```

Prueba_4.cpp

```
// Palabras reservadas

int main()
{
   int a, b, c, A, B, C;

   cout << "Introduce lado a: "; cin >> a;
   cout << "Introduce lado b: "; cin >> b;
   cout << "Introduce lado c: "; cin >> c;

   A = a * 2;
   B = b / 4;
   C = c % 3;
   A += 3;
   B -= 1;
   C *= 2

   if(A == 90 || B == 90 || C == 90)
      cout << "coquito";
   if(A < 90 && B < 90 && C < 90)
      cout << "Los 3 son menores que 90";
   if(A > 90 || B > 90 || C > 90)
      cout << "Hay alguno mayor que 90";
}
```

Prueba_5.cpp

```
//Palabras reservadas class, private, public, char, void, cout, int,
return
//Tokens { } * ( ) :: : ; << .
//Identificadores Empleado, m_nombre, ImprimirInfo, main

class Empleado {
   private:
      char* m_nombre;

   public:
      void ImprimirInfo();
};

void Empleado::ImprimirInfo( )
{
  cout << "Nombre: " << m_nombre;
}

int main()
{
   //creacion de un objeto de la clase Empleado
   Empleado empleado12;

   //impresion de los datos
   empleado12.ImprimirInfo();
   return 0;
}
```

Prueba_6.cpp

```
//Prueba 6
//Palabras reservadas int, for, cout, while, main
//Tokens ( ) { } += << -- > =

int coco(int toto)
{
 int a;
 for (a = 50; a >0; a--)
          {
          toto += 2;
          cout << toto
          cout << a
          }
}

int cece(int toto)
{
 int a;
 for (a = 50; a >0; a--)
          {
          toto += 2;
          cout << toto
          cout << a
          }
}

int main(void) {    // ================
  int i;
  while (i > 0)
          {
          cout << "hola";
          i--;
          }
}
```

- **Pruebas incorrectas**

Prueba_1.cpp

```
//Prueba 1 errónea.
//Declarando una variable de tipo cadena de forma errónea

int main(int argc, char *argv[]) {
        char* hola[5];
        hola = "lango ;
}
```

Prueba_2.cpp

```
//Hallar el promedio de n numeros
//Carácter no reconocido '
void main(void)
{int i,cantidad;

cout <<"Ingrese la cantidad de numeros para calcular el
promedio:"  ; cin >'> cantidad;

}
```

Prueba_3.cpp

```
//Prueba 3. Incorrecta. Prueba errores del nivel léxico
//(En la función pedir el parámetro tiene un nombre
//mal formado)
//Palabras reservadas void, int, for, cout, cin, return,
//Tokens ( ) [ ] <= = ++ << + ;

void pedir (int 123foo[3], int len)
{
   for(int i=0; i<=len; i++)
   {
      cout << "Numero " << i+1 << ":? ";
      cin >> matriz[i];
   }
}

int sumar(int matriz[3], int len)
{
   for(int i=0; i<=len; i++)
      matriz[i++];
   return matriz[3];
}

void mostrar(int matriz[3], int len)
{
   for(int i=0; i<=len; i++)
      cout << matriz[i] << " ";
}

int main()

   return 0;
```

Prueba_4.cpp

```
//Prueba 4. Prueba de errores a nivel sintáctico. Falta el ; en
C*= 2
// Palabras reservadas

int main()
{
   int a, b, c, A, B, C;

   cout << "Introduce lado a: "; cin >> a;
   cout << "Introduce lado b: "; cin >> b;
   cout << "Introduce lado c: "; cin >> c;

   A = a * 2;
   B = b / 4;
   C = c % 3
   A += 3;
   B -= 1;
   C *= 2

   if(A == 90 || B == 90 || C == 90)
      cout << "coquito";
   if(A < 90 && B < 90 && C < 90)
      cout << "Los 3 son menores que 90";
   if(A > 90 || B > 90 || C > 90)
      cout << "Hay alguno mayor que 90";
}
```

Prueba_5.cpp

```
//Prueba 5. Prueba de errores a nivel sintáctico.
//Intento de poner un identificador que es palabra reservada
//(línea 25)
//Palabras reservadas class, private, public, char, void, cout, int,
//return
//Tokens { } * ( ) :: : ; << .
//Identificadores Empleado, m_nombre, ImprimirInfo, main,

class Empleado {
   private:
      char* m_nombre;

   public:
      void ImprimirInfo();
};

void Empleado::ImprimirInfo( )
{
  cout << "Nombre: " << m_nombre;
}

int main()
{
   //creacion de un objeto de la clase Empleado
   Empleado if;

   //impresion de los datos
   empleado12.ImprimirInfo();
   return 0;
}
```

Prueba_6.cpp

```
//Prueba 6. Errores a nivel sintáctico. Modificada la línea 31 (Intento
//de hace cout sin el operador <<
//Palabras reservadas int, for, cout, while, main
//Tokens ( ) { } += << -- > =

int coco(int toto)
{
 int a;
 for (a = 50; a >0; a--)
          {
          toto += 2;
          cout << toto
          cout << a
          }
}

int cece(int toto)
{
 int a;
 for (a = 50; a >0; a--)
          {
          toto += 2;
          cout << toto
          cout << a
          }
}

int main(void) {    // ================
  int i;
  while (i > 0)
          {
          cout /*<<*/ "hola";
          i--;
          }
}
```