UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS

Seminario de Solución de Problemas de Traductores de Lenguajes II

Reporte de practica

Nombre del alumno:	Casillas Núñez Fernando
Profesor:	López Franco Michel Emanuel
Titulo de la practica:	Analizador Léxico"
Fecha:	26/05/2020

Introducción

En esta actividad, el alumno tiene que realizar un programa en interfaz grafica, el cual implemente un análisis léxico mediante un flujo de entrada de datos (un archivo), mostrando los datos as como un desglose del lenguaje y los caracteres correspondientes al mismo que se leen.

Metodología

Para cumplir con los requerimientos que marca la práctica, hice una aplicación de ventana en C++ usando QT IDE. El programa funciona de la siguiente manera:

Una vez que se presiona el botón CORRER" dentro del programa, se lee un archivo que se encuentra en el directorio raíz del programa, pasa su contenido dentro de una caja de texto visible en la ventana, mientras que a su lado establece un contador de palabras. Abajo de todo, se muestra una tabla donde se agrupan 8 conjuntos de tipos de palabras/tokens que for-man el léxico del lenguaje C. El programa lee carácter por carácter todo el flujo de datos que recibe del archivo, buscando y validando para identificar símbolos/delimitadores u operadores aritméticos, si encuentra alguno de es-tos, el carácter en cuestión es movido a un arreglo diferente para poder ser utilizado después. Las palabras que sobran son revisadas de nuevo mediante varias verificaciones que nos dice si lo que se lee son identificadores, palabras reservadas, tipos de datos, etc. Lo anterior mencionado se maneja entera-mente en la clase "Léxico" del programa, aunque la programación principal ocurre en el método constructor del objeto de dicha clase, el cual se muestra a continuación: Si se quiere un mejor análisis de la clase, se puede veri car el código fuente anexo a este reporte.

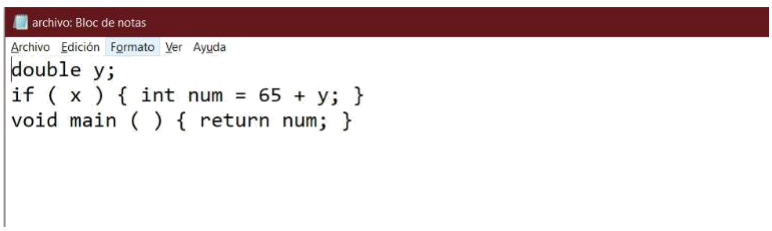
```
L e x i c o : : L e x i c o ( char  s [ ] )
f
      char     p = s ;
      char  b u f f e r [MAX1 ] ;

       int idxW = 0 ,  idxS = 0 ,  idxA = 0 ;

       for ( int  i =0;  i <MAX1;           i ++)
       f
             TokenReser [ i ]          = "";
             TokenTipoDato [ i ]          = "";
             TokenOpRel [ i ]          = "";
             TokenOpLog [ i ]          = "";
             TokenError [ i ]          = "";
             b u f f e r [ i ] =  ' n0 ' ;
       g

       while (  p !=       ' n0 ' )
       f
```

```
        if((p!=' ')&&(p!='nn'))
        {
            if((p==';')||(p=='"')||(p=='(')||(p==')')||
(p=='#')||(p=='f')||(p=='g')||(p==':')||(p==','))
            {
                tokenS[idxS] = p;
                idxS++;
            }
            else if((p=='+')||(p==' ')||(p==' ')||(p=='/')||
            (p=='='))
            {
                tokenOA[idxA] = p;
                idxA++;
            }
            else
            {
                buffer[idxW] = p;
                auxW[idxW] = p;
                idxW++;

            }

        }
        else if((p==' ')||(p==','))

        {
            if(verificarTipo(buffer)==true)
            {
                TokenTipoDato[aux5] = buffer;
                aux5++;
            }
            else if(verificarOpRel(buffer)==true)
            {
                TokenOpRel[aux7] = buffer;
                aux7++;
            }
            else if(verificarOpLog(buffer)==true)
            {
                TokenOpLog[aux6] = buffer;
                aux6++;
            }
            else if(verificarRes(buffer)==true)
            {
                TokenReser[aux1] = buffer;
                aux1++;
            }
            else if(verificarId(buffer)==true)
            {
                TokenIden[aux2] = buffer;
                aux2++;
            }

            else if(verificarNum(auxW)==true)
            {
                TokenNum[aux3] = auxW;
                aux3++;
            }
```

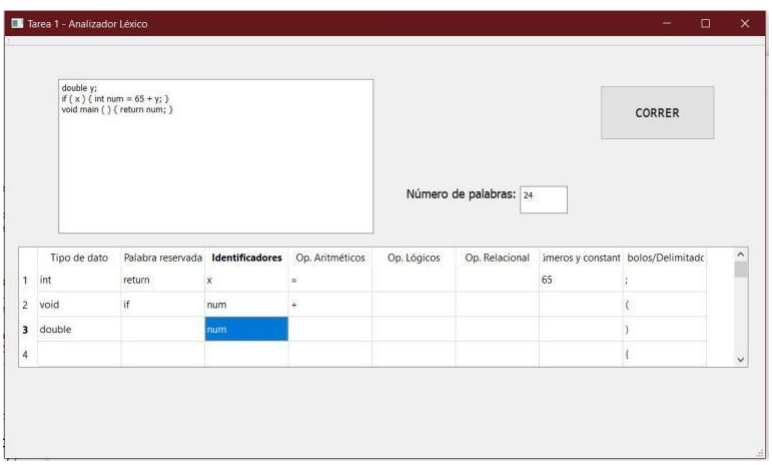
```
                for ( int      i =0;  i <MAX1;     i ++)
                f
                        b u f f e r [ i ] =  ' n0 ' ;
                        auxW [ i ] =  ' n0 ' ;
                g
                idxW = 0 ;
        g
        p++;
 g

 // r e v e r i f i c a n d o
if ( v e r i f i c a r T i p o ( b u f f e r ) == t r u e )
 f
        TokenTipoDato [ aux5 ] = b u f f e r ;
        aux5++;
 g
 e l s e i f ( v e r i f i c a r O p R e l ( b u f f e r ) == t r u e )
 f
        TokenOpRel [ aux7 ] = b u f f e r ;
        aux7++;
 g
 e l s e i f ( v e r i f i c a r O p L o g ( b u f f e r ) == t r u e )
 f
        TokenOpLog [ aux6 ] = b u f f e r ;
        aux6++;
 g
 e l s e i f ( v e r i f i c a r R e s ( b u f f e r ) == t r u e )
 f
        TokenReser [ aux1 ] = b u f f e r ;
        aux1++;
 g
 e l s e i f ( v e r i f i c a r I d ( b u f f e r ) == t r u e )
 f
                TokenIden [ aux2 ] = b u f f e r ;
                aux2++;
 g
 e l s e i f ( v e r i f i c a r N u m (auxW)== t r u e )
 f
        TokenNum [ aux3 ] = auxW ;
        aux3++;
  g // s e termina e l p r o c e s a m i e n t o de l a cadena de e n t r a d a


g
```

Resultados

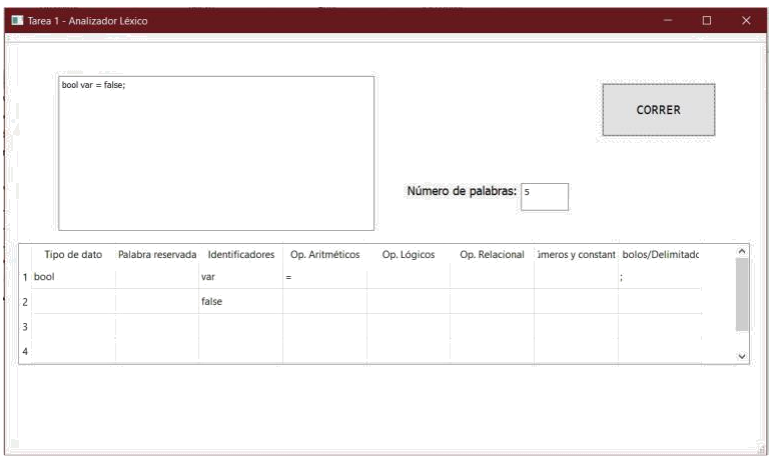
Ahora pasaremos a ver si pasa la prueba nuestro programa. Correremos el programa usando el siguiente archivo, con el contenido que se muestra:



Corremos el programa para ver que se lea y se muestre correctamente.



Probamos nuevamente:



Conclusiones

El análisis léxico es la primera parte del compilador que se va a formar a lo largo del semestre, siendo el léxico una parte muy importante del mismo, ya que gracias a él se pueden leer los caracteres de entrada y salida de nuestro lenguaje, que después pasaran a ser procesados como componentes léxicos en el análisis sintáctico que se verá más adelante.

Referencias

1.- ANALISIS LEXICO. Galeon. Sitio web: http://www.galeon.com/shock/tareas.html

2.- Compiler Design. GeeksforGeeks. Sitio web: https://www.geeksforgeeks.org/compiler-lexical-analysis/