

Universidad de Alcalá.

Grado Ingeniería Informática.

Asignatura: Procesadores del Lenguaje.

Profesor: José Luis Cuadrado García.

Autor: Luis Ángel Parada

DNI: 50502347M

Memoria PECL1

Analizador Léxico

Un analizador léxico lee los caracteres de la entrada y los agrupa en “objetos token”. Junto con un símbolo de terminal que se utiliza para las decisiones de análisis sintáctico, un objeto token lleva información adicional en forma de valores de atributos.

En un compilador, el analizador léxico lee los caracteres del programa fuente, los agrupa en unidades con significado léxico llamadas lexemas, y produce como salida tokens que representan estos lexemas.

Un token consiste en dos componentes: el nombre del token y un valor de atributo. Los nombres de los tokens, son símbolos abstractos que utiliza el analizador sintáctico para su análisis. A menudo, a estos tokens les llamamos terminales, ya que aparecen como símbolos terminales en la gramática para un lenguaje de programación. El valor de atributo, si está presente, es un apuntador a la tabla de símbolos que contiene información adicional acerca del token. Esta información adicional, no forma parte de la gramática, por lo que a menudo, en nuestra explicación sobre el análisis sintáctico, nos referimos a los tokens y los terminales como sinónimos.

Para reconocer estos token, se utilizan expresiones regulares. En esta práctica hemos utilizado el generador de analizadores léxicos JFLEX basado en Java, el cual utiliza las siguientes reglas léxicas:

- $a|b$ Unión: Es una expresión regular que encuentra todas las entradas que sean válidas para a ó b .

- **ab Concatenación:** Es una expresión regular que encuentra todas las entradas que sean válidas para a seguida de b.
- **a* Cerradura de Kleene:** Es una expresión regular que encuentra todas las entradas que sean válidas para cero o más repeticiones de a.
- **a+ Iteración:** Es una expresión regular que encuentra todas las entradas que sean válidas para cero o más repeticiones de a.
- **a? Opción:** Es una expresión regular que encuentra todas las entradas que sean válidas para cero o una ocurrencia de a.
- **!a Negación:** Es una expresión regular que encuentra todas las entradas que sean válidas para cualquier expresión diferente de a.
- **a{n} Repetición:** Es una expresión regular que encuentra todas las entradas que sean válidas para exactamente n repeticiones de a.
- **a {n}{m}:** Es una expresión regular que encuentra todas las entradas que sean válidas para entre n y m repeticiones de a.
- **a:** Es una expresión regular que encuentra todas las entradas que coincidan exactamente con a.
- **~a upto o hasta** encuentra todo el texto que le sigue a "a" incluyendo la primera ocurrencia de "a" es equivalente a $!([^\wedge]^* a [^\wedge]^*) a$.

Utilicé como opciones del JFLEX

Para la realización de esta PECL1, se utilizaron las siguientes opciones de JFLEX:

- **%unicode** usar unicode para scanear.
- **%line** contador de líneas.
- **%columna** contador de columnas.
- **%ignorase** insensitivo a mayúsculas y minúsculas.
- **%class** para crear una clase con un nombre en particular por el analizador.

MACROS y Expresiones Regulares

En la especificación léxica de la práctica, tenemos unos requisitos que cumplir y se demostrarán y explicarán de ahora en adelante.

Utilicé un total de 6 Macros para los requisitos básicos como: identificadores, funciones, números, espacio, salto de línea, tabulador, operadores, comentarios, signos de puntuación.

Las Palabras Reservadas siempre presentan un problema, ya que pueden confundirse por identificadores porque comparten el mismo patrón de cadenas de caracteres. Para su tratamiento, utilicé el tratamiento de ellas de manera **implícita**. Esto quiere decir, que todas las palabras fueron tratadas como **identificadores** y luego comparadas en un switch. Si el token es una palabra clave, se aumentará el contador palabrasreservadas en 1. Si es una palabra reservada como TRUE o FALSE también se aumentará el contador booleanos en 1. Si no es una palabra clave, tiene que ser un identificador y se aumentará el contador en 1. A continuación el macro para el tratamiento de identificador.

Identificador = [a-zA-Z]([a-zA-Z0-9_]*)

Este macro se ejecutará siempre que el token comience por una letra minúscula o mayúscula y sea precedido o no por un carácter o más caracteres numéricos.

Acción: aumentar en uno el contador palabrasreservadas.

Funciones = "\$[a-zA-Z][a-zA-Z0-9_]*

Este macro se ejecutará siempre que el token comience por un signo de "\$" seguido de una letra minúscula o mayúscula y sea precedido o no por un carácter o más caracteres numéricos.

Acción: aumentar el contador de funciones en uno y agregar en una lista el nombre de la misma. Se verificará si ya contiene el nombre de la misma para no agregar repetidos.

Números = ("-"|"")?[0-9]+

Este macro se ejecutará siempre que el token comience o no por un signo de "+" o "-" y sea precedido por 1 o más números en el rango de 0 hasta 9.

Acción: aumentar el contador de numero en uno.

Salto = \r\n\r\n

Este macro se ejecutará al detectar estos caracteres especiales de salto de línea, acarreo o juntos, no realizará ninguna acción.

Acción: Ninguna.

Comentarios = "/*" ~[\r\n]

Este macro se ejecutará al encontrar dos "/*" juntos y cualquier texto entre "/*" y salto de línea y acarreo. Estos nos garantiza que los comentarios sean unilineales y que lo que se escriba en esa línea pueda ser cualquier cosa sin que lo empareje con otra regla

Acción: Ninguna.

Espacio = {Salto} | [\t\f]

Este macro se ejecutará al detectar estos caracteres especiales de salto de línea, acarreo o juntos, como también los de tabulación.

Acción: Ninguna.

Operadores = "+|-|=|\->

Este macro se ejecutará al detectar cualquiera de los operadores válidos en el lenguaje uno a uno y sin repeticiones.

Acción: Ninguna.

Puntuacion = ",|";|"(|)"

Este macro se ejecutará al detectar cualquiera de los signos de puntuación válidos en el lenguaje, uno a uno y sin repeticiones.

Acción: Ninguna.

Tratamiento de Errores

Para el tratamiento de errores, utilicé 12 expresiones regulares. Cuando un token haga match con alguna de ellas, se indicará la posición del token que presentó el error, indicando qué tipo de error es: operadores o puntuación, identificadores, funciones erróneas, como también indicando la línea, la columna y el token erróneo.

Siempre que aparezca un error en el fichero, este será mostrado mientras se va analizando, más no detendrá la ejecución del analizador léxico.

Errores por operadores no validos

(<>)+

En esta expresión regular se captura cualquier operador de la forma "<>" .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(>=)+

En esta expresión regular se captura cualquier operador de la forma ">=" .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(<=)+

En esta expresión regular se captura cualquier operador de la forma "<=" .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(+=)+

En esta expresión regular se captura cualquier operador de la forma "+=" .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(\+\+)+

En esta expresión regular se captura cualquier operador de la forma “++” .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(\-\-)+

En esta expresión regular se captura cualquier operador de la forma “--” .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(\=\=)+

En esta expresión regular se captura cualquier operador de la forma “==” .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

(\-\=)+

En esta expresión regular se captura cualquier operador de la forma “-+” .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

”)”“(“

En esta expresión regular se captura cualquier operador de la forma “)(” .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

\-[\<\+\-\)\(]+

En esta expresión regular se captura cualquier operador de la forma “-<”, “-+”, “-”, “,”, “-” .

Acción: invocar al procedimiento Error pasándole por parámetro la opción 2.

Errores por identificador o funciones

[!\"'#\$%&\'*\\W.:<|=>\\?\\@\\[\\|\\]\\^_`\\{\\|\\}~0-9]+[a-zA-Z][a-zA-Z0-9_]*

En esta esta expresión regular se captura un identificador erróneo, tomando como error 1 o más apariciones del conjunto de símbolos y números no definidos en el lenguaje, y esto seguido por la expresión regular de identificador.

Acción: invocar al procedimiento Error pasándole por parámetro la opción 0.

"\$"[!\"'#\$%&\'*\\W.:<|=>\\?\\@\\[\\|\\]\\^_`\\{\\|\\}~0-9]+[a-zA-Z][a-zA-Z0-9_]*

En esta esta expresión regular se captura una función errónea, tomando como error 1 o más apariciones del conjunto de símbolos y números no definidos en el lenguaje, y esto seguido por la expresión regular de función tomando en cuenta que tiene que empezar por el signo de "\$".

Otro tipo de errores

Si no hace match con ninguna de las expresiones regulares ya mencionadas se detectara error y se imprimirá por pantalla. utilizando el “.”

Pruebas y Verificaciones

Para verificar que todas las expresiones regulares antes mencionadas funcionan correctamente, hacemos la ejecución del analizador léxico YYlex, este tiene incorporado un procedimiento llamado reporte el cual se ejecutara al llegar EOF del archivo mandado por parámetro. Este procedimiento nos muestra las estadísticas del archivo leído mostrando:

- Número de apariciones de Palabras Reservadas
- Cantidad de apariciones de Números
- Número de apariciones de Booleanos
- Número de apariciones de Identificadores
- Número de apariciones de Funciones y el nombre de las mismas

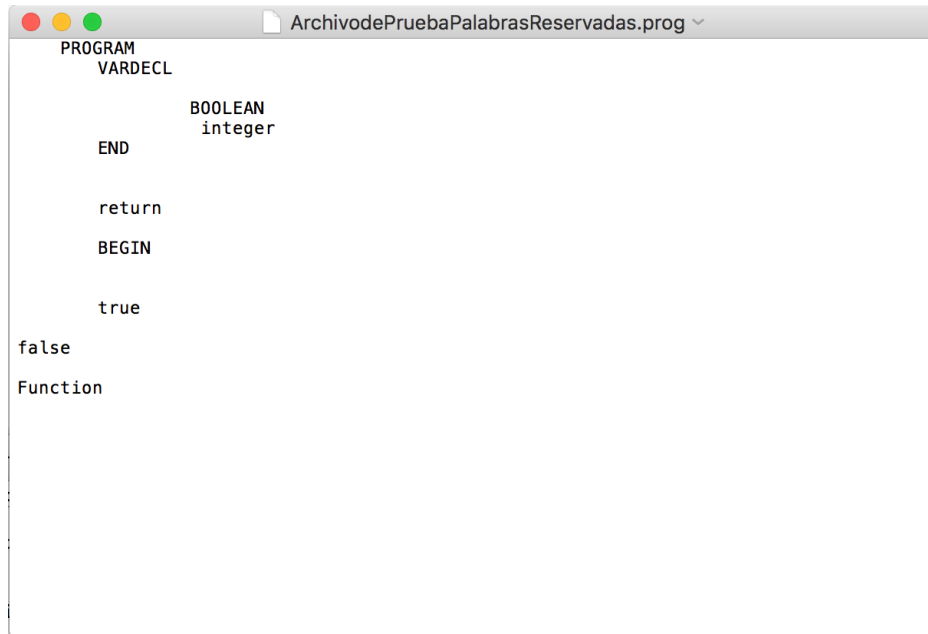
Para estas pruebas solo se tiene que ejecutar el programa, ya que gracias a la facilidad de ECLIPSE de pasar parámetros todos los archivos son enviados como parámetros en la tabla se muestra el valor de la i y a que Archivo de prueba pertenece

Tabla de Pruebas

Tipo de Prueba	Número a colocar en argv[i]
Palabras Reservadas	0
Identificadores	1
Funciones	2
Números	3
Comentarios	4
Errores	5
Extensión	6
Texto	7

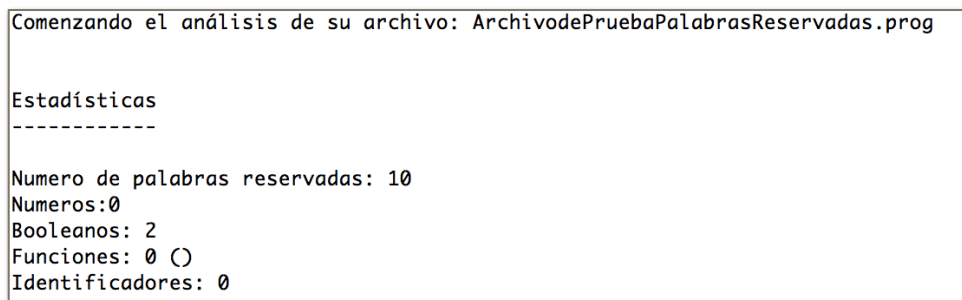
Prueba de Palabras Reservadas

El archivo de texto ArchivodePalabrasReservadas.prog contiene estos datos:



```
PROGRAM
VARDECL
    BOOLEAN
    integer
END
return
BEGIN
true
false
Function
```

La salida esperada para este archivo seria 10 palabras reservadas y 2 boleados, al ejecutar el analizador léxico nos verifica esto.



```
Comenzando el análisis de su archivo: ArchivodePruebaPalabrasReservadas.prog

Estadísticas
-----

Numero de palabras reservadas: 10
Numeros: 0
Booleanos: 2
Funciones: 0 ()
Identificadores: 0
```

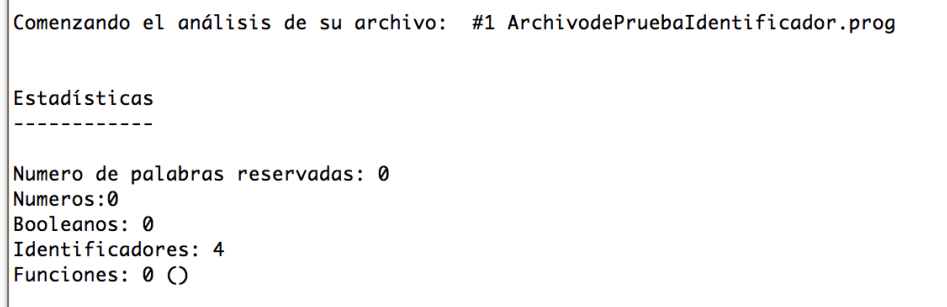
Prueba para Identificadores

El archivo de texto ArchivodePruebaldentificador.prog contiene estos datos:



```
id1 id2
A51 A1
```

La salida esperada de un texto como este seria 4 identificadores validos, al ejecutar el analizador nos verifica esto



```
Comenzando el análisis de su archivo: #1 ArchivodePruebaIdentificador.prog

Estadísticas
-----

Numero de palabras reservadas: 0
Numeros: 0
Booleanos: 0
Identificadores: 4
Funciones: 0 ()
```

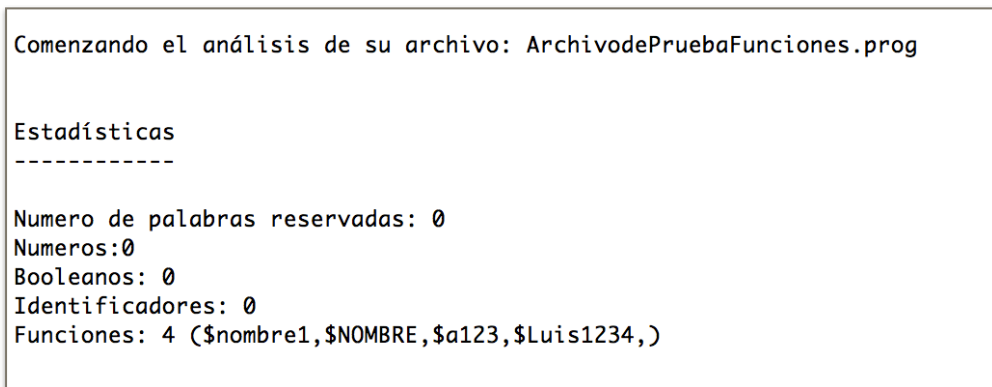
Prueba para Funciones

El archivo ArchivodePruebaFunciones.prog contiene:



```
ArchivodePruebaFunciones.prog
|
$nombre1      $NOMBRE $a123
$Luis1234
```

La salida esperada para este archivo seria 4 funciones y los nombres de las mismas literalmente iguales, al ejecutar el analizador nos verifica esto



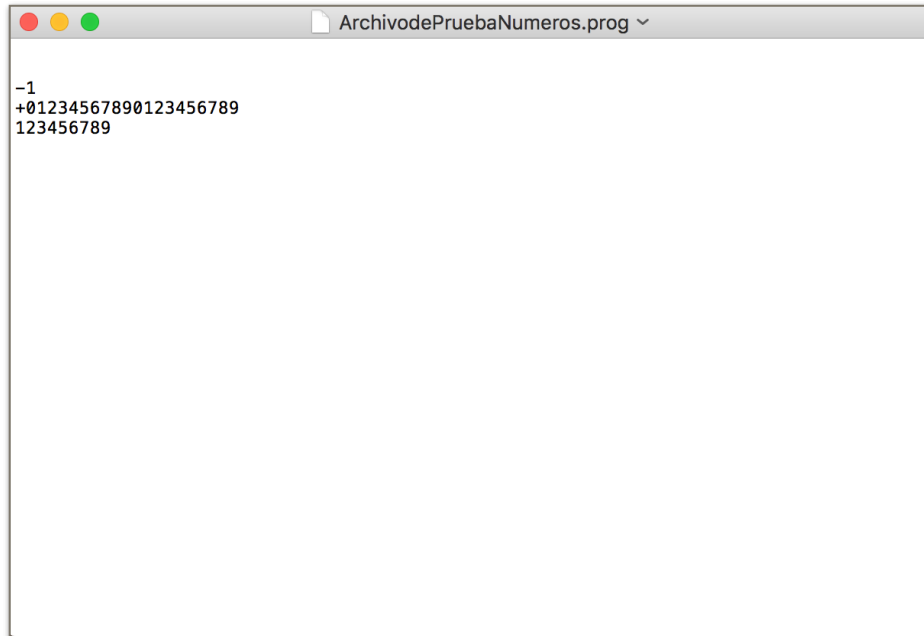
```
Comenzando el análisis de su archivo: ArchivodePruebaFunciones.prog

Estadísticas
-----

Numero de palabras reservadas: 0
Numeros: 0
Booleanos: 0
Identificadores: 0
Funciones: 4 ($nombre1,$NOMBRE,$a123,$Luis1234,)
```

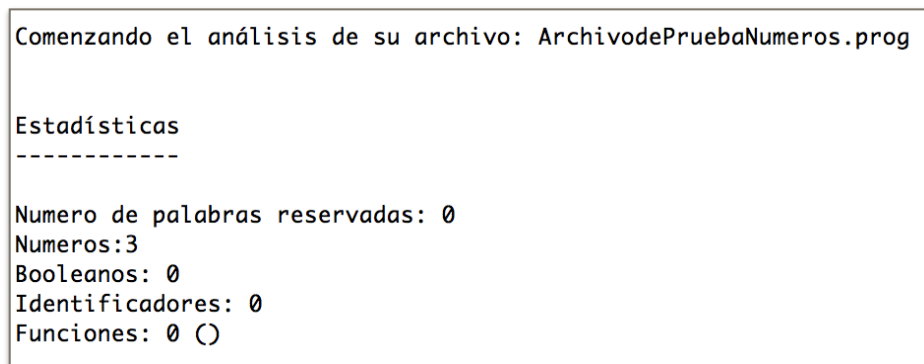
Prueba de Números

El archivo ArchivodePruebaNumeros.prog contiene:



```
-1
+01234567890123456789
123456789
```

La salida esperada para este archivo seria 3 números validos al ejecutar el analizador nos verifica esto:



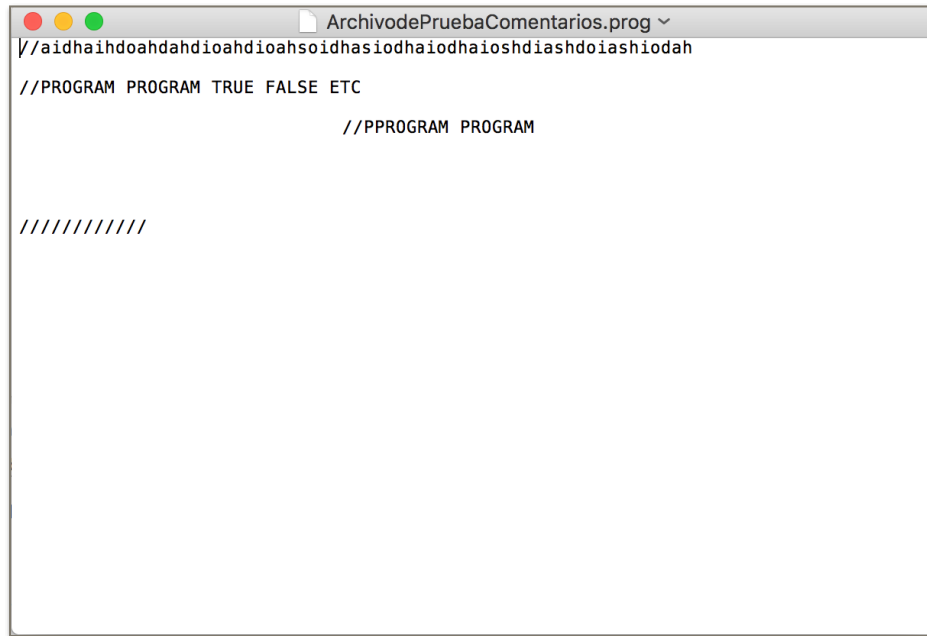
```
Comenzando el análisis de su archivo: ArchivodePruebaNumeros.prog

Estadísticas
-----

Numero de palabras reservadas: 0
Numeros:3
Booleanos: 0
Identificadores: 0
Funciones: 0 ()
```

Prueba de Comentarios

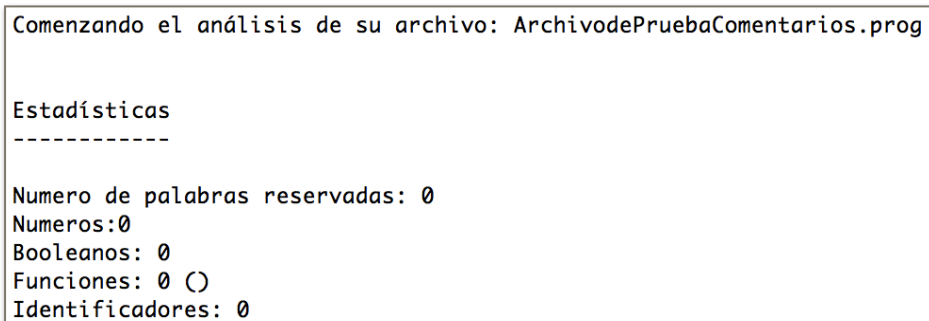
El archivo ArchivoPruebaComentarios.prog contiene:



```
//aidhaihdoahdahdioahdioahsoidhasiodhaiodhaioashdiashdoishiodah
//PROGRAM PROGRAM TRUE FALSE ETC
//PPROGRAM PROGRAM

//////////
```

La salida esperada para este archivo seria que nos muestra 0 en todas las estadísticas y ningún tipo de error al ejecutar el analizador nos verifica esto.



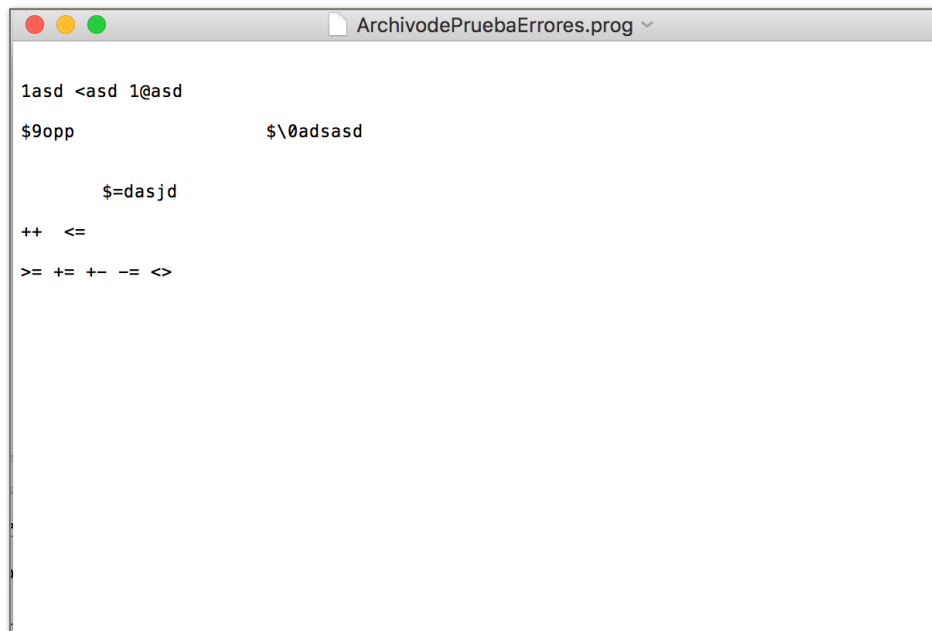
```
Comenzando el análisis de su archivo: ArchivodePruebaComentarios.prog

Estadísticas
-----

Numero de palabras reservadas: 0
Numeros:0
Booleanos: 0
Funciones: 0 ()
Identificadores: 0
```

Prueba de Errores

El archivo ArchivodePruebaErrores.prog contiene:



```
1asd <asd 1@asd
$9opp                $\0adsasd

    $=dasjd
++  <=
>= += +- -= <>
```

La salida esperada para este archivo seria 13 notificaciones de errores indicando posición en el fichero, tipo y su valor al ejecutar el analizado nos verifica esto.

```
Comenzando el análisis de su archivo: ArchivodePruebaErrores.prog

Error en la línea: 3 columna: 1 Identificador Erroneo: '1asd'
Error en la línea: 3 columna: 6 Identificador Erroneo: '<asd'
Error en la línea: 3 columna: 11 Identificador Erroneo: '1@asd'
Error en la línea: 5 columna: 1 Funcion Erroneo: '$9opp'
Error en la línea: 5 columna: 9 Funcion Erroneo: '$\0adsasd'
Error en la línea: 8 columna: 2 Funcion Erroneo: '$=dasjd'
Error en la línea: 10 columna: 1 Operador o Puntuacion erronea: '++'
Error en la línea: 10 columna: 5 Operador o Puntuacion erronea: '<='
Error en la línea: 12 columna: 1 Operador o Puntuacion erronea: '>='
Error en la línea: 12 columna: 4 Operador o Puntuacion erronea: '+= '
Error en la línea: 12 columna: 10 Operador o Puntuacion erronea: '-='
Error en la línea: 12 columna: 13 Operador o Puntuacion erronea: '<>'
```

Prueba de Extension de Archivo

Para la verificación de que solo sea posible ser analizado ficheros con la extensión “.prog” utilice la clase File la cual contiene una serie de métodos por el API de java muy útiles.

Los dos métodos utilizados fueron getName() que devuelve un String con el nombre del archivo pasado por parámetro a la aplicación, obtenemos el tamaño de este String luego comparamos si existe “.prog” a partir de 5 caracteres antes del final del mismo. Si este es verdadero se dispone a analizar el archivo en cambio si el valor es falso mostrara un mensaje por consola de “ Su Archivo: “nombredelarchivo” no es compatible “

```
public static void main (String argv[])throws java.io.IOException {
    for(int i=0; i<argv.length ; i++){
        File fis=new File(argv[i]);
        String nombre= fis.getName();
        int tamaño = nombre.length();

        if (nombre.startsWith(".prog", tamaño-5))
        {
            FileInputStream realFile =new FileInputStream(fis.getAbsolutePath());

            System.out.println();
            System.out.println( "Comenzando el análisis de su archivo: "+ nombre);
            System.out.println();

            Analizador yy = new Analizador(realFile);
            while (yy.yylex() != -1) ;
            reporte();

        }
        else{
            System.out.println();
            System.out.println( "Comenzando el análisis de su archivo: "+ nombre);

            System.out.println( "Su Archivo: " +nombre+ " no es compatible");
        }
    }
}
```


Comprobamos esto pasándole el fichero ArchivodePruebaExtension.txt, la salida esperada para este fichero debe ser “ Su Archivo: “nombredelarchivo” no es compatible” al ejecutar el analizado comprobamos esto

```
Su Archivo: ArchivodePruebaExtension.txt no es compatible
```

Prueba de Texto

En esta prueba introducimos el texto de ejemplo proporcionado en el enunciado de la PECL1, el resultado que esperamos es exactamente el mismo que del enunciado.

Texto PECL1:

```
1  PROGRAM
2  VARDECL
3      id,id2,id3  -> INTEGER;
4      id4,id5,id6 -> BOOLEAN;
5      cont, lasd  -> integer;
6  END;
7
8  function $nombre1 a1 (boolean), b1 (integer)
9      id3 = 4 + b1;
10     return true;
11
12     function $nombre2 a2
13     return 4;
14
15     function $nombre3
16     return true;
17
18     BEGIN
19         //fichero de prueba
20         id3 = (6 + 6) - $nombre2(2);
21         id=$nombre2(4);
22         id4= $nombre1(false,2);
23         id7 =$nombre2(id);
24     END;
```

La salida del analizador realizado

```
Comenzando el análisis de su archivo: ArchivodePruebaTexto.prog

Error en la línea: 5 columna: 10 Identificador Erroneo: '1asd'
Error en la línea: 5 columna: 15 Operador o Puntuacion erronea: '-<'

Estadísticas
-----

Numero de palabras reservadas: 19
Numeros:7
Booleanos: 3
Funciones: 3 ($nombre1,$nombre2,$nombre3,)
Identificadores: 17
```