**Universidad CAECE**

**Procesadores de Lenguaje**

Analizador Léxico

Versión 1.1

**Docente:**Amitrano, Sergio

**Integrantes:**Dinardi, Matías (00000/0)  
Maquieira , Guillermo (70519/9)  
Palmieri, Bruno (77086/4)  
Pentreath, Matías (00000/0)  
Zarco, Nicolás (76102/8)

12 de Septiembre de 2012

Contenido

[Contenido 2](#_Toc335174692)

[Historial de Revisiones 3](#_Toc335174693)

[Descipcion del Trabajo Practico 4](#_Toc335174694)

[Autómata 4](#_Toc335174695)

[Implementación 5](#_Toc335174696)

[Casos de Prueba 6](#_Toc335174697)

[Indice 8](#_Toc335174698)

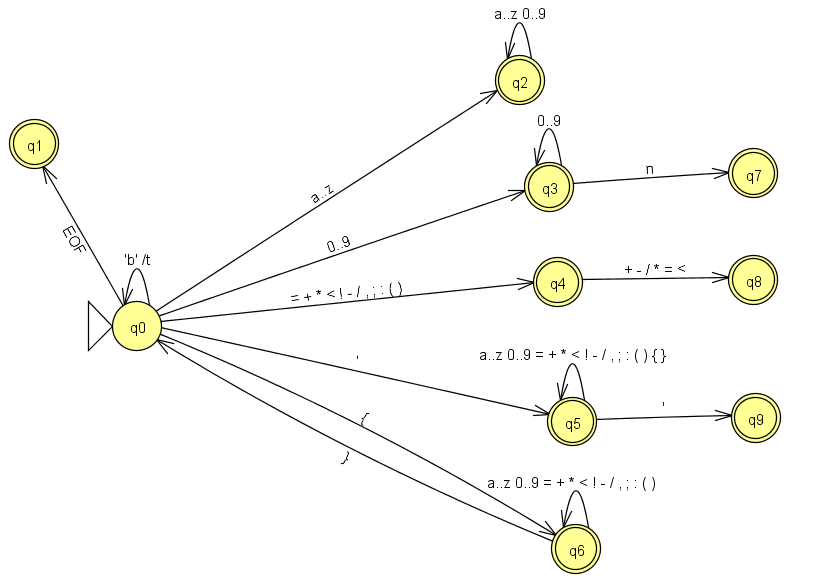
Historial de Revisiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** | **Autor** |
| **08/09/2012** | 1.0 | -Creación del documento.  -Autómata | Zarco Nicolás |
| **11/09/2012** | 1.1 | -Cambio diseño del documento  -Creación de Índice  -Creación de Tabla de Contenidos  -Implementación  -Casos de Prueba | Palmieri Bruno |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Descipcion del Trabajo Practico

Autómata

Se crea un autómata finito no determinístico para modelar los tipos de componentes léxicos a aceptar.



Se utiliza la técnica de “Espiado” (o *peeking*) de próximo carácter, para avaluar que tipo de lexema se está leyendo del archivo, y luego, categorizar el tipo de *Token* a devolver. Los *Token* posibles son “EOF”, “Palabra”, “Entero”, “Natural”, “Operador”, “Cadena” y “Error”.

Implementación

Para la implementación creamos una clase llamada **LexicAnalyzer** la inicializamos con el directorio donde se encuentra el archivo a analizar. Esta clase posee un paramento **getToken()** el cual retornara el siguiente *token* de lexema encontrado en el archivo. Existen 7 tipos de *token* que pueden ser retornados:

* 4 para los diferentes tipos de lexemas (Entero, Natural, Operador, Palabra)
* 1 para el EOF (*End of File* o Fin de Archivo).
* 1 de Error, el cual sera enviado en caso de detectar el mismo dentro de un lexema.
* 1 de Palabra Reservada, en casa de que el lexema corresponda a una palabra reservada el lexema se enviara un token de dicho tipo.

La clase **LexicAnalyzer** al recibir una petición **getToken()** comienza a analizar el archivo, para ello comienza por omitir todos aquellos carácter o secuencia de caracteres que pueden ser omitidas hasta encontrarse con el primer carácter valido para el comienzo de un *token*. Una vez encontrado dicho carácter este identifica el tipo de lexema a tratar (ya sea del tipo Numerico, Cadena, Operador, Palabra o EOF) y se invoca a la clase **LexicHelper** la cual retornará el token con el **ID** correspondiente. En caso de que se detecte algún error a nivel léxico el **LexicHelper** retornara un token de error, esto puede producirse por la aparición de un carácter invalido, una mala declaración numérica o el no cierre de una cadena.

Tanto la clase **LexicAnalyzer** como la clase **LexicHelper** hacen uso de la clase **FileReader** para la lectura del archivo. La misma posee dos parámetros muy utilizados, los cuales son:

**read():** el cual posiciona el puntero en el siguiente carácter del archivo y almacena devuelve el calor de este.

**peek():** el mismo retorna el siguiente carácter que se encuentra en el archivo pero no modifica la posición del puntero.

Casos de Prueba

El siguiente documento fue utilizado para realizar los casos de prueba

|  |
| --- |
| globales  const M : natural = 7n, R : entero = 90;  var N, S : entero;  var A[12] : natural;  fin-globales;  procedimiento PROC1();  var A : entero;  var B : natural;  comenzar  B := M;  while B > 0 do  si par(B) entonces  A := A + 1;  fin-si  mostrarLN 'Visualizacion', B, ' ', A  B := B - 1  fin-mientras  fin-proc;  procedimiento PROC2(ref R : entero);  const T : natural = 67n;  var W11 : entero;  var W12, Q, R : natural;  procedimiento PROC2A();  comenzar  mostrarLN 'Interno A'  fin-proc;  const S : natural = 15n;  procedimiento PROC2B();  comenzar  mostrarLN 'Interno B'  fin-proc;  comenzar  S := (S ++ M) \*\* 2n;  Q := 1n;  leer W11; {lectura de teclado}  W12 := ANATURAL(AW11 \* 2 + S);  while (W12 – 2) <= R + S do  W12 := W12 \* 2;  fin-mientras;  while W12 + M > Y / 2 do  Q := Q \* 2;  W12 := W12 / 2;  si W12 <= R entonces  R := R – W12;  Q := Q + 1;  fin-si;  fin-mientras;  A[1] := Q;  fin-proc;  var T: natural;  funcion INC(N : entero) : entero; adelantado;  funcion FUN1(T: entero, val N2 : entero) : entero;  var N: entero;  comenzar  si T > 0 entonces  N := 45;  sino  N := INC(70);  fin-si;  fin-func N \* 2;  funcion INC(N : entero) : entero;  comenzar  fin-func N + 1;  var H: entero;  procedimiento PRINCIPAL();  var X: entero;  comenzar  PROC1();  S := M + 1;  PROC2(S);  X := FUN1(5, 8);  A[5n] := X;  mostrar A[ANATURAL(S) ++ 1n];  fin-proc; |

Indice

A

Autómata 4

C

Casos de Prueba 6

Contenido 2

D

Descipcion del Trabajo Practico 4

H

Historial de Revisiones 3

I

Implementación 5

Indice 8