
	<b>Guía No. 3</b>	Introducción a los Compiladores.		Departamento de informática.
	Universidad Don Bosco <b>Facultad de Ingeniería</b> Es cuela de Computación			
	Asignatura		: Compiladores	
	Ciclo		: 01 – 2004	
Lugar de ejecución:		: Centro de Computo		

## I. OBJETIVOS

- ☞ Conocer las características básicas de un Compilador.
- ☞ Analizar e interpretar como un compilador realiza el análisis léxico de un programa.

## II. INTRODUCCIÓN

### Compiladores

Un traductor que transforma textos fuente de lenguajes de alto nivel a lenguajes de bajo nivel se le denomina *compilador*.

El tiempo que se necesita para traducir un lenguaje de alto nivel a lenguaje objeto se denomina *tiempo de compilación*.



Fig. 1

El tiempo que tarda en ejecutarse un programa objeto se denomina *tiempo de ejecución*.



Fig. 2

### Léxico

El léxico de un lenguaje natural está constituido por todas las palabras y símbolos que lo componen. Para un lenguaje de programación la definición también es válida.

En un lenguaje de programación el léxico lo constituyen todos los elementos individuales del lenguaje, denominados frecuentemente en inglés *tokens*. Así son tokens: las palabras reservadas del lenguaje, los símbolos que denotan los distintos tipos de operadores, identificadores (de variables, de funciones, de procedimientos, de tipos, etc.), separadores de sentencias y otros.

### Sintaxis

En lingüística, sintaxis es el estudio de la función que desempeña cada palabra en el entorno de una frase. Mientras que semántica es el estudio del significado de una palabra tanto a nivel individual como en el contexto de una frase.

En los lenguajes de programación, sintaxis es un conjunto de reglas formales que especifican la composición de los programas a base de letras, dígitos y otros caracteres. Por ejemplo, las reglas de sintaxis especifican en C/C++ que cada sentencia o línea de programa debe terminar con un “;”, o que la declaración de tipos debe ir antes que la de variables. (int var;)

### Semántica

Semántica en los lenguajes de programación es el conjunto de reglas que especifican el significado de cualquier sentencia, sintácticamente correcta y escrita en un determinado lenguaje.

Por ejemplo en el lenguaje Pascal la sentencia:

suma:= 27/lado

es sintácticamente correcta, ya que a la izquierda del símbolo de asignación hay un identificador, y a la derecha una expresión. Pero para que sea semánticamente correcta hay que comprobar:

- a) *lado* debe ser compatible con el operador “/” y con el operando 27.
- b) *suma* debe ser un tipo compatible con el resultado de la operación.

### Análisis Léxico de un Compilador -Traductor

Un programa fuente es una serie de símbolos (letras, símbolos, caracteres especiales: +, \*, !). Con estos símbolos se representan las construcciones del lenguaje tales como variables, etiquetas, palabras reservadas, constantes, etc. Es necesario que el compilador o traductor identifique los distintos significados de estas construcciones, que los creadores de lenguajes dan en la definición del lenguaje.

El programa fuente se trata inicialmente con el **analizador léxico** (en inglés *scanner*), con el propósito de agrupar el texto en grupos de caracteres con significado propio llamados *tokens* o *componentes léxicos*, tales como variables, identificadores, palabras reservadas y operadores. Por razones de eficiencia a cada token se le asocia un atributo (o más de uno) que se representa internamente por un código numérico o por un tipo enumerado.

Por ejemplo considere la siguiente sentencia es C/C++:

if sueldo == 1000 sueldo \* 0.25;

El analizador léxico la separa en la siguiente secuencia de tokens:

if sueldo == 1000 sueldo \* 0.25 ;

y les asigna su atributo cuyo significado se ha definido previamente:

Token	Atributo	Observaciones
if	20	Palabra reservada
sueldo	1	Identificador
==	15	Operador de comparación
1000	8	Valor numérico o constante
*	12	Operador Aritmético
0.25	8	Valor numérico o constante
;	27	Separador de sentencias

En general, el **análisis léxico** es un análisis a nivel de caracteres, su misión es reconocer los componentes léxicos o tokens, enviando al analizador sintáctico (en la próxima guía) los tokens y sus atributos.

### III. REQUERIMIENTOS:

#### Materiales y equipo a utilizar

DESCRIPCIÓN	CANTIDAD
Guía de laboratorio	1
Editor de texto	1
Compilador de Visual C++	1
Disco Flexible (1.44 Mb)	1

### IV. PROCEDIMIENTO

Suponga que se desea construir una mini simulación de un compilador, tomando en cuenta nada más el análisis léxico de un programa. El programa fuente será un código escrito en un lenguaje definido por el usuario, denominado “LPTLP”.

Generalmente un compilador toma el programa fuente, lo interpreta y crea un programa objeto (normalmente en lenguaje máquina). Por ahora nos limitaremos a comprender y analizar una de las formas, de como se llevaría a cabo un analizador léxico según las características de un lenguaje.

La definición de los componentes léxicos del lenguaje LPTLP es la siguiente:

- o Identificadores, que sólo son nombres de variables y están compuestos por una única letra minúscula de rango de a – z.
- o Constantes numéricas de un solo dígito, de rango 0 – 9.
- o Operadores: +, -, \*, / y %.
- o Símbolo de asignación: =
- o Paréntesis: ( y ).
- o Separador de sentencias: ; (punto y coma).
- o Indicadores de principio y fin de bloque: { y }.
- o Palabras reservadas, están formadas por una letra mayúscula, las cuales son:  
R (lectura). W (escritura) y M (programa principal).

Observe que en este lenguaje, todos los *tokens* son de un sólo carácter. Además se considera que se tiene un sólo tipo de dato: entero, y que las variables están formadas por una única letra minúscula, y las constantes son de un dígito.

Se asume que para identificar la *sintaxis* de cada sentencia, se conoce que reglas de programa se han de seguir, con solo conocer el token por el que comienza la sentencia.

Digite el siguiente programa (en lenguaje LPTLP) en un archivo de texto, guárdelo como **prueba1.txt** en la unidad C. (Programa Ejemplo).

```
M{
  R a;
  R b;
  c = a+b-2;
  W c;
}
```

Describa lo que hace el programa anterior según las definiciones del lenguaje: \_\_\_\_\_

---

---

---

---

## Analizador léxico

El análisis léxico debe separar el fichero fuente en componentes léxicos o tokens, y enviarlos al analizador sintáctico (en este guía no se detallara el analizador sintáctico). Habitualmente se envían los componentes léxicos y sus atributos. En este caso solo se enviaran los tokens, ya que el atributo va implícito en el token (tan sólo se tiene el tipo de dato entero).

A continuación se muestra la definición de clase **Léxico**, la cual contiene las funciones necesarias para poder implementar un análisis léxico adecuado para el lenguaje LPTLP.

```
#include <iostream.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#define TAM_BUFFER 100

class Lexico
{
    char *nombreFichero; //nombre del fichero fuente de entrada
    FILE *entrada;       //fichero de entrada
    int n1;              // # de línea
    int traza;           //control de línea para dar seguimiento a la ejecución
    char buffer[TAM_BUFFER]; //buffer aux. de caracteres
    int pBuffer;
public:
    Lexico(char *unNombreFichero, int una_traza=0);
    ~Lexico(void);
    char siguienteToken(void);
    void devuelveToken(char token);
    int lineaActual (void) { return n1;};
    int existeTraza(void) {if (traza) return 1; else return 0;}
};

Lexico::Lexico(char *unNombreFichero, int una_traza){
    if ((entrada=fopen(unNombreFichero, "r"))==NULL)
    {
        cout << "No se puede abrir el archivo"<<endl;
        exit(-2);
    }
    if (una_traza) traza=1;
    else traza=0;
    n1 = 1; //se inicializa el contador de líneas
    pBuffer=0; //Se inicializa la posición del buffer
}
Lexico::~~Lexico(){
    fclose(entrada);
}

char Lexico::siguienteToken(void){
    char car;
    while((car=
        ((pBuffer>0) ? buffer[--pBuffer]:getc(entrada))
        )!=EOF)
    {
        if (car==' ') continue;
        if (car=='\n') {++n1; continue;}
        break;
    }
}
```

```

}

if (traza) cout<<"ANALIZADOR LEXICO: Lee el token"<<car<<endl;

switch (car)
{
case 'M':
case 'R':
case 'W': //palabras reservadas
case '=': //asignacion
case '(': //paréntesis
case ')':
case ';': //separadores
case '}':
case '{':
case ':': //fin de programa
case '+': //operadores aritméticos
case '*':
case '-':
case '/':
case '%': return (car);
}

if (islower(car)) return (car); //variables
else if (isdigit(car)) return (car); //constantes
else
{
    cout<<"Error Lexico: TOKEN DESCONOCIDO"<<endl;
    exit(-4);
}
return (car);
}

void Lexico::devuelveToken(char token){
if (pBuffer>TAM_BUFFER)
{
    cout<<"ERROR: Desbordamiento del buffer del analizador lexico"<<endl;
    exit(-5);
}
else
{
    buffer[pBuffer++]=token;
    if (existeTraza())
        cout<<"ANALIZADOR LEXICO: Recibe en buffer el token "<<token<<endl;
}
}
}

```

**Nota:** En la próxima guía se implementara un analizador sintáctico, utilizando esta misma clase como referencia.

### Programa Principal

A continuación se muestra un pequeño programa para observar como es realizado el proceso de análisis léxico por la clase. (SI Ud. desea puede implementar otro tipo de proceso dentro de main).

```

int main()
{

```

```
int traza; //para dar seguimiento a la corrida
char token;

//Objeto para acceder a los miembros de la clase (obj)
Léxico obj("C:\\prueba1.txt", 1); //se sobrecarga el constructor de la clase

if ( obj.existeTraza() )
    cout <<"INICIO DE ANALISIS" <<endl;

while ((token=obj.siguienteToken() )!= '}')
    cout<<token<<endl;

return 0;
}
```

### Actividad

1. Para qué cree que funcionaría la función devuelve token dentro del programa. Analice la clase Lexico.
2. Coloque en el archivo prueba.txt algún carácter no reconocido por el lenguaje LPTLP y vuelva a ejecutar el programa.

## V. INVESTIGACIÓN Y EJERCICIOS COMPLEMENTARIOS

- Investigue como funciona y cómo se crea un árbol sintáctico. (Palabra clave: Árboles y recursividad).