



Unidad III Análisis Léxico

M.C. Juan Carlos Olivares Rojas

Agenda

- 3.1 Introducción a los Autómatas finitos y expresiones regulares.
- 3.2 Analizador de léxico.
- 3.3 Manejo de localidades temporales de memoria (buffers).
- 3.4 Creación de tablas de símbolos.
- 3.5 Manejo de errores léxicos.
- 3.6 Generadores de código léxico: Lex y Flex.



3.1 Introducción a los Autómatas finitos y expresiones regulares

- ¿Qué es un autómata? Es un modelo matemático que sirve para determinar si una cadena pertenece a un lenguaje no.
- $M = (Q, \Sigma, \delta, F)$
 - Q = conjunto de estados
 - Σ = alfabeto de entrada
 - δ = funciones de transición
 - F = conjunto de estados de aceptación



Autómatas finitos

- La característica que tienen los autómatas finitos es que solo existe una función de transición definida para un símbolo de entrada. Esto elimina ambigüedades
- Una expresión regular es una forma abreviada de representar lenguajes:
 - a Representa el lenguaje de las letras a
 - a^* Representa el lenguaje que tiene 0 hasta n a 's



Autómatas

- Oprel $\rightarrow <|>|<=>|=|=|<>$
- Un solo autómatata varios sub_autómatas
- Programar un autómatata:
- Identificador \rightarrow letra (letra|digito|gb)*



Expresión Regular

- Generar la expresión regular para identificar correos electrónicos válidos.
- Formato:
- jcolivar@itmorelia.edu.mx
- id@dominio



Expresiones Regulares y una gramática

- Una gramática sirve para generar cadenas de un determinado lenguaje pero también sirve para generarla.
- Existe una relación uno a uno entre Lenguajes, Autómatas, Expresiones regulares y gramáticas.



Gramática de un if

prop \rightarrow if expr then prop
| if expr then prop else prop
| ϵ

expr \rightarrow termino oprel termino
| termino

termino \rightarrow id
| num



Gramática de un if

- $\text{oprel} \rightarrow < |> | = | < = | < > | > = |$
- $\text{id} \rightarrow \text{letra} (\text{letra} \mid \text{digito})^*$
- $\text{num} \rightarrow \text{digito}^+ (. \text{digito}^+)? (\text{E} (+ \mid -)? \text{digito}^+)?$
- $\text{eb} \rightarrow \text{delim}^+$
- $\text{delim} \rightarrow \text{blanco} \mid \text{tab} \mid \text{linenueva}$



3.2 Análizador Léxico

- Primera fase de la compilación
- Leer caracteres de entrada y generar como salida una secuencia de componentes léxicos
- Eliminar espacios en blanco
- Eliminar comentarios
- Proporcionar información acerca de errores léxicos



Componentes léxicos, lexema y patrones

ID	Componente léxico	Lexema	Patrón
201	IF	if	if
202	OP_RELACIONAL	<, >, !=, ==	<, >, !=, ==
203	IDENTIFICADOR	var, s1, suma, prom	letra (letra dígito gb)*
204	ENTERO	2, 23, 5124	(dígito)+



Análisis Léxico

- El análisis lineal, se llama léxico o exploración.
- $Posicion := inicial + velocidad * 60$
- Posicion: identificador
- := símbolo de asignación
- Inicial: identificador



Análisis Léxico

- +: signo de suma
- Velocidad: identificador
- *: signo de multiplicación
- 60: numero

- Se elimina todos los espacios en blancos (espacios, tabuladores, salto de línea, etc.)

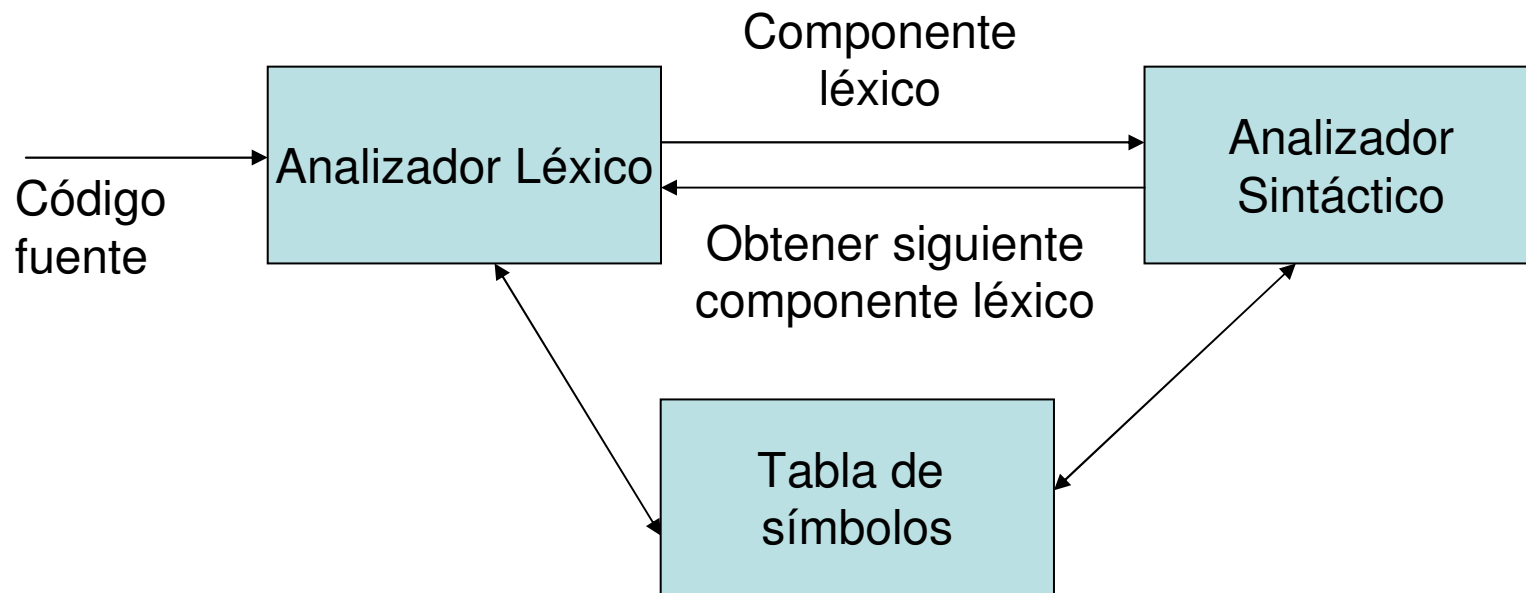


Análisis Léxico

- Reconocedores de identificadores y palabras clave
- La tabla de símbolo debe insertar y buscar componentes léxicos
- La tabla de símbolos es utilizada por el analizador sintáctico y por otras fases del proceso de traducción



Función del analizador léxico



Análisis léxico

- Un patrón es una regla que describe el conjunto de lexemas que puede representar a un conjunto léxico
- Los componentes léxicos se tratan como terminales de la gramática del lenguaje fuente
- La devolución de un componente léxico se hace a través de un número entero



Especificación de componentes léxicos

- Expresiones regulares (patrón)
- Cada patrón concuerda con una serie de cadenas
- Las expresiones regulares dan el nombre al conjunto de cadenas con que concuerdan



Expresiones regulares

- Se construyen a partir de otras expresiones regulares más simples
- Cada expresión regular r , representa un lenguaje $L(r)$
- Letra $a \cup b \cup c \cup \dots \cup z$
- Dígito $1 \cup 2 \cup 3 \cup \dots \cup 0$
- Identificador $\text{letra}(\text{letra} \cup \text{dígito})^*$



Definiciones regulares

- Dan nombres a las expresiones regulares
- Nos permiten referenciarlas recursivamente
- Dígito $\rightarrow 1|2|3|4|5|6|7|8|9|0$
- Entero $\rightarrow \text{dígito}^+$
- Decimal $\rightarrow \text{.dígito}^+ | \text{.dígito}^+E(+|-|\epsilon)\text{dígito}^+$
- Real $\rightarrow \text{entero} (\text{decimal} | \epsilon)$



Abreviaturas

- * cero o más casos
- + uno o más casos
- [a-zA-Z] mayúsculas y minúsculas
- [0-9] dígitos
- ? Cero o un caso



Abreviaturas

- Dígito \rightarrow [0-9]
- Entero \rightarrow dígito+
- Decimal \rightarrow .dígito+exponente?
- Exponente \rightarrow (E|e) (+|-)?dígito+
- Real \rightarrow entero decimal?



3.3 Manejo de localidades temporales de memoria (buffers)

- La forma más fácil de leer un programa es carácter por carácter pero es ineficiente.
- La forma más eficiente es realizar una copia a la memoria de todo el código fuente. Pero esto en la gran mayoría de las ocasiones es impráctico por las dimensiones de los programas. Para solucionar este problema se sugiere utilizar buffers



Manejo de buffers

- Existen muchas formas de dividir el trabajo, pero siempre se deberá llevar dos punteros, uno al carácter actual y otro al inicial del lexema.
- El manejo de buffers es esencial para realizar el análisis de grandes programas de mejor manera



3.4 Creación de tablas de símbolos

- En general el proceso de análisis léxico puede describirse simplemente como el reconocimiento de caracteres de un lenguaje para generar una tabla de símbolos.
- El primer paso consiste en crear un escáner, el cual se encarga de verificar que no existan caracteres no presentes en el lenguaje.



Tabla de símbolos

- La tabla de símbolos va a guardar cada palabra analizada, la va identificar como un lexema y le va asociar un identificador numérico para posteriormente utilizarlo.
- La tabla de símbolos debe estar en memoria para realizar un análisis rápido.



3.5 Manejo de errores léxicos

- Son pocos los errores que se pueden detectar al hacer análisis léxico
- `fi (a == f(x)) //Error de sintaxis`
- Pero puede existir algún error si ninguno de los patrones con cuerda con el prefijo de entrada



Técnicas de recuperación de errores

- Borrar un carácter extraño
- Insertar un carácter que falta
- Reemplazar un carácter incorrecto por otro correcto
- Intercambiar dos caracteres adyacentes



Técnicas para realizar analizadores léxicos

- Utilizar un analizador léxico como FLEX. El generador se encarga de manejar buffers
- Escribir el analizador en un lenguaje de alto nivel haciendo uso de la E/S del lenguaje
- Escribir el lenguaje ensamblador y manejar explícitamente la E/S



Análisis Léxico en XML

- El análisis léxico en documentos XML lo realiza cualquier herramienta o API que utilice XML, ya que debemos cerciorarnos que el lenguaje esté bien formado.
- Si el lenguaje no cumple con las reglas de construcción de documentos XML, falla el proceso.
- Realizar análisis léxico de XML en Java o C#



3.6 Generadores de código léxico: Lex y Flex

- FLEX es la versión de software libre del popular generador de analizadores léxicos LEX para sistemas *NIX, genera código C aunque existen otras herramientas que generan código en otros lenguajes
- Analizador.lex → flex → lex.yy.c → gcc → Programa ejecutable analizador
- \$gcc lex.yy.c -o analizador -fl



Programa Lex

%{

Definiciones globales 'C'

}%

Definiciones flex

%%

Acciones

%%

Código 'C' auxiliar



Definiciones regulares en flex

- %% Separadores de secciones
- Def → expresión
- Acciones
- {def} {código 'C' asociado}
- “@” {código 'C' asociado}



Programa que reconoce flotantes

```
%{  
    #include <stdio.h>  
    Int ocurrencias;  
}%  
Digito[0-9]  
Punto      [\.]  
Exp  [eE]  
Signo[\+|-]  
Digitos    {digito}+  
Decimal    {punto} {digitos}({exp}{signo}{digitos})?  
Flotante   {digitos}{decimal}?
```



Programa que reconoce flotantes

```
%%
```

```
{flotante} { printf("flotante encontrado\n");  
            ocurrencias++; }
```

```
“@” { printf("Arroba\n"); }
```

```
. { printf("Inválido: %s\n", yytext); }
```

```
%%
```



Programa que reconoce flotantes

```
main(int argc, char *argv[])
{
    FILE *f;
    F = fopen(argv[1], "r");
    yyin = f;
    while(yylex());
    printf("%d flotantes encontrados\n", ocurrencias);
    fclose(f);
}
```



Programa para reconocer direcciones IP

Digito [0-9]

Punto [\.]

IP {digito}+{punto}{digito}+{punto}{digito}+
{punto}{digito}+

%%

```
{IP} { strcpy(aux, yytext);  
      strcat(aux, ".");  
      for(i =0 ; i<4; i++)  
      {
```



Programa para reconocer direcciones IP

```
cadnum = strtok(aux, ".");  
if(atoi (cadnum)>255)  
{  
    printf("Error\n");  
    break;  
}  
if(i==4)  
    printf("Dirección IP: %s\n", yytext);  
}
```



¿Preguntas?

