IC-5701
Compiladores e Intérpretes

Profesor:
Ing. Allan Rodríguez Dávila, MGP

Agenda

TEC

- Análisis Léxico
- Expresiones Regulares
- Autómatas Finitos
- Conversión

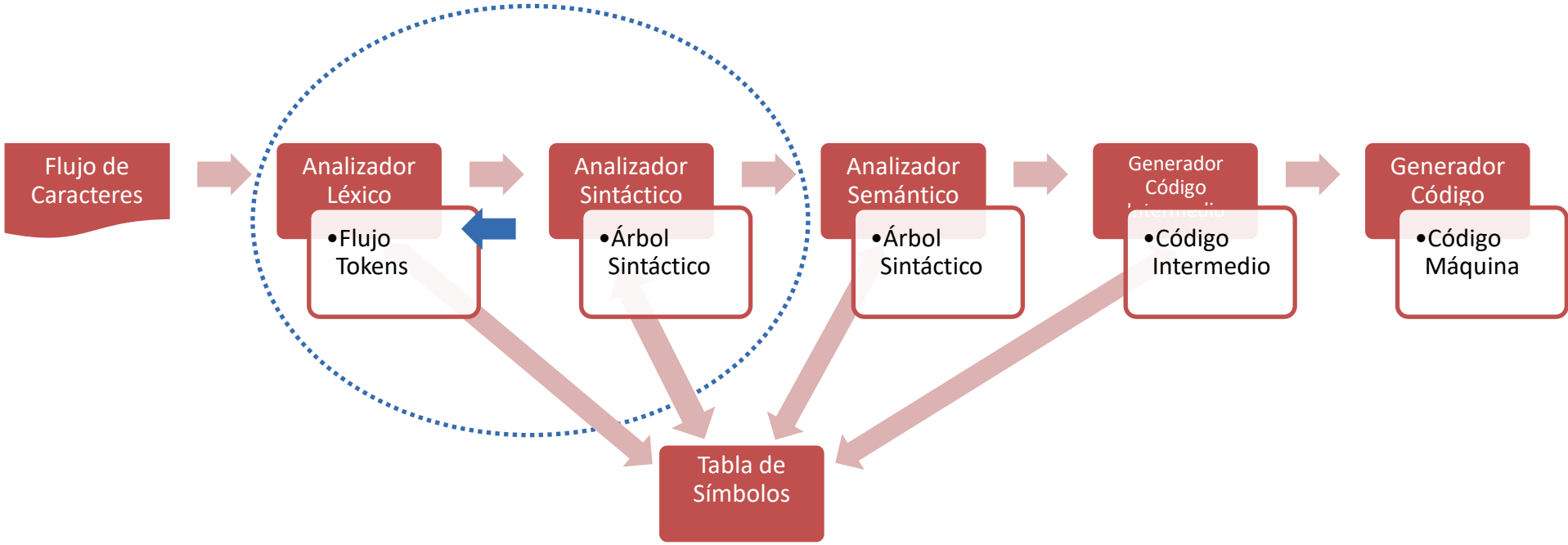
Proceso de compilación

Compilador

Proceso de Compilación

- Lectura (Portafolio #1) – **Mapa Mental**
  - Identificar las principales características de las diferentes fases del proceso de compilación:
    - Análisis léxico
    - Análisis sintáctico
    - Análisis semántico
    - Generación de código intermedio
    - Optimización de código
    - Generación de código
    - Administración de tabla de símbolos

Compilación

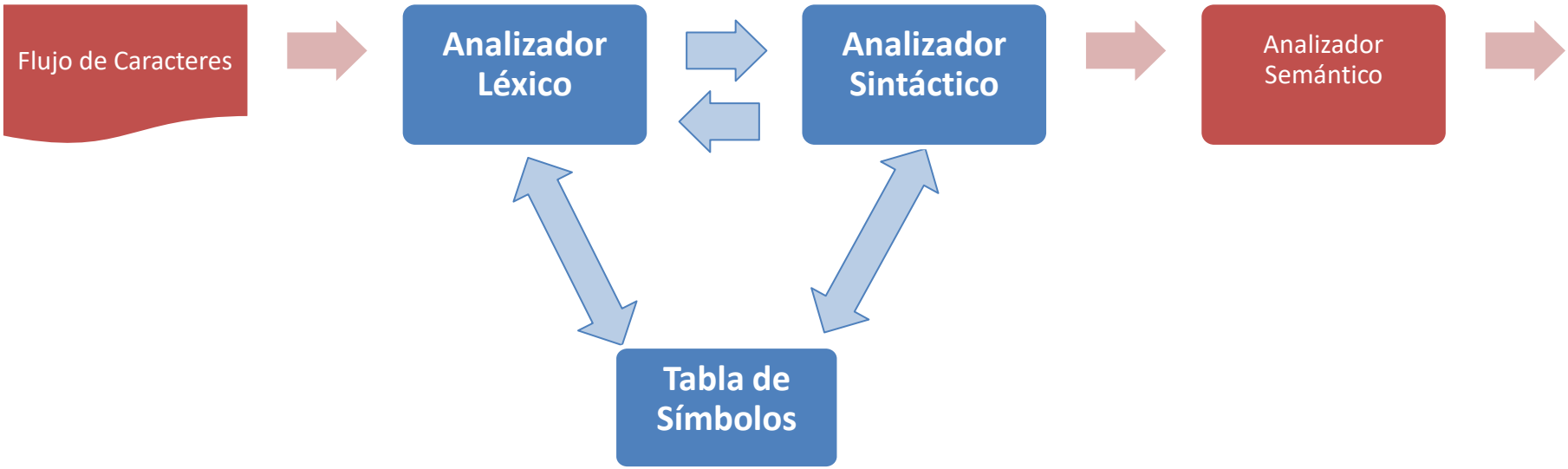


Análisis Léxico

Análisis Léxico

- Primera fase del compilador
- Lee los caracteres del programa fuente
- Envía los tokens al analizador sintáctico
- Interactúa con la tabla de símbolos

Interacción



Análisis Léxico

- Lee el texto origen
- Elimina comentarios y espacios en blanco
- Correlaciona los mensajes de error con el programa fuente.
- Expansión de macros

Procesos

- Escaneo
  - Limpieza de código
- Análisis léxico
  - Secuencia de tokens

Tokens

- Par [nombre,valor-atributo]
- Nombre es un símbolo abstracto que representa un tipo de unidad léxica

```
printf(“Total = %d”,puntuacion);
```

- Tokens:
  - [printf] //tipo id
  - [(] //tipo signo puntuación: ,) ;
  - [“Total = %d”] //tipo literal
  - [id,1] //tipo id

Patrón

- Descripción de la forma de los lexemas
  - `else` → `e, l, s, e`
  - `posicion` → `relación de muchas cadenas`

Lexemas

- Secuencia de caracteres que coinciden con el patrón de un token.

```
printf(“Total = %d”,puntuacion);
```

- Lexemas:
  - `printf`
  - `(`
  - `“Total = %d”`
  - `,`
  - …

Tokens, Patrones y Lexemas

Token	Descripción informal	Lexemas ejemplo
`if`	Caracteres i, f	`if`
`else`	Caracteres e, l, s, e	`else`
`comparacion`	< o > o <= o >= o == o !=	`<=, !=`
`id`	Letra seguida por letras y dígitos	`pi, puntuacion, D2`
`numero`	Cualquier constante numérica	`3.1415, 0, 6.02e23`
`literal`	Cualquier cosa excepto “ rodeada por dos “’s	`“Hola Mundo”`

Atributos de los tokens

- Apuntador a la entrada en la tabla de símbolos del identificador
  - Cadena de caracteres
  - Tipo (token)
  - ¿Qué más?
    - Línea
    - Columna

Bonus…

- Divida el siguiente código en lexemas

```
float cuadradoLimitado(float x)
{
    /*devuelve x al cuadrado, pero nunca más de 100*/
    float limiteInferior = -10.0;
    float limiteSuperior = 10.0;
    return (x<=limiteInferior||x>=limiteSuperior)?100:x*x;
}
```

- Qué lexemas deberían tener valores léxicos asociados?

Bonus…

- Divida el siguiente código en lexemas

```
float cuadradoLimitado(float x)
{
    /*devuelve x al cuadrado, pero nunca más de 100*/
    float limiteInferior = -10.0;
    float limiteSuperior = 10.0;
    return (x<=limiteInferior||x>=limiteSuperior)?100:x*x;
}
```

- Conviértalos a tokens!

Bonus…

- Transcriba el código

```
float cuadradoLimitado(float x)
{
    /*devuelve x al cuadrado, pero nunca más de 100*/
    float limiteInferior = -10.0;
    float limiteSuperior = 10.0;
    return (x<=limiteInferior||x>=limiteSuperior)?100:x*x;
}
```

- Con tokens (Incluir tabla simbolos)

Bonus…

- Divida el siguiente código en lexemas y convíertalos a tokens

```
/*Uso del while con ejemplo*/
 int x; int y;
 x=read();
 y=read();
 while(x!=y){
    if(y<=x){
        y=y-x;
    } else {
        x = x-y;
    }           }
 /*Salida de resultado*/
 write("Resultado: ");
 write(x);
```

Expresiones Regulares

Expresiones Regulares

- Notación para expresar un conjunto de strings de símbolos terminales.

Sintaxis	Descripción
'|'	Alternativas separadas
'*'	Ítem previo aparece zero o muchas veces
'+'	Ítem previo aparece una o muchas veces
'.'	Un carácter cualquiera
'(' y ')'	Paréntesis de agrupamiento

Expresiones Regulares

- Notación para especificar patrones de lexemas.

Sintaxis	Descripción
‘^’	Inicio de Línea
‘$’	Fin de línea
‘\’	Carácter de escape
‘?’	Ítem previo aparece cero o una vez
‘[’ y ‘]’	Cualquier carácter dentro de los paréntesis
‘{’n,m‘}’	Ítem previo aparece de n a m veces

TEC

Expresiones Regulares

Acción	Expresión Regular	Descripción
Vacío	ε	String vacío
Un ítem	t	String de ***t*** sólo
Concatenación	XY	Concatenación de cualquier string generado por ***X*** y cualquier string generado por ***Y***
Alternativa	$X\|Y$	Cualquier string generado ya sea por ***X*** o por ***Y***
Iteración	X^*	Concatenación de cero o más string generados por ***X***
Agrupamiento	(X)	Cualquier string generado por ***X***

TEC

Expresiones Regulares

Expresión Regular	Conjunto Resultante
Sr|Sra	
S(r|ra)	
ps*t	
ba(na)*	
M(r|s)*	

TEC

Expresiones Regulares

Expresión Regular	Conjunto Resultante
Sr|Sra	Sr, Sra
S(r|ra)	Sr, Sra
ps*t	pt, pst, psst, pssst, …
ba(na)*	ba, bana, banana, bananana, …
M(r|s)*	M, Mr, Ms, Mrr, Mrs, Mss, Mrrr, …

Conceptos

- Un alfabeto es un conjunto finito de símbolos y se representa por **Σ**.

  {0,1} alfabeto binario

- **|s|** representa el número de ocurrencias de **s**.

Conceptos

- La cadena vacía es presentada por ε
  - Cadena de longitud 0
  - Es la identidad en la concatenación

    εs = sε = s
- Un lenguaje es cualquier conjunto contable de cadenas de algún alfabeto fijo.

Conceptos

- El prefijo de la cadena **s** es cualquier cadena que se obtiene de eliminar cero o más símbolos del final de **s**.

- El sufijo de la cadena **s** es cualquier cadena que se obtiene de eliminar cero o más símbolos del principio de **s**.

Conceptos

- Una subcadena de **s** se obtiene de eliminar cualquier prefijo y cualquier sufijo de **s**.

- Prefijos, sufijos y subcadenas propios de una cadena **s** son aquellos que no son **ε** ni iguales a la misma cadena.

Conceptos

- Una subsecuencia de **s** es cualquier cadena que se forma mediante la eliminación de cero o más posiciones no necesariamente consecutivas de **s**.

Operaciones de Lenguaje

Operaciones	Definición y notación
Unión L y M	L U M = {s | s está en L o en M}
Concatenación L y M	LM = {st | s está en L y t en M}
Cerradura de Kleene de L	$L^* = \bigcup_{i=0}^{\infty} L^i$
Cerradura positivo de L	$L^+ = \bigcup_{i=1}^{\infty} L^i$

Expresiones Regulares

- Notación utilizada para describir lenguajes por medio de operadores que se aplican a los símbolos del alfabeto.
- Cada expresión regular ***r*** denota un lenguaje ***L(r)***
  - Ejemplo: Identificadores en C
    - `letra_(letra_|digito_)*`

Leyes Algebraicas para REGEX

Operaciones	Definición y notación
r | s = s | r	| es conmutativo
r | (s | t) = (r | s) | t	| es asociativo
r(st) = (rs)t	La concatenación es asociativa
r(s | t) = rs | rt; (s | t)r = sr | tr	La concatenación se distribuye sobre |
εr = rε = r	ε es la identidad para la concatenación
r* =(r | ε)*	ε se garantiza en una cerradura
r** = r*	* es idempotente

Ejemplos

- Escriba las expresiones regulares para los siguientes lenguajes:

{aa, ee, ii, oo, uu, ala, ele, ili, olo, ulu}

{int, bool, float, char, string}

{+, -, *, /, %}

Un identificador

Un número entero

Portafolio #1

- Escriba las expresiones regulares para los siguientes lenguajes:

  {a, ab, abb, abbb, abbbb, …, ac, abc, abbc, abbbc, abbbbc, …}

  {olu, omu, onu, ulo, umo, uno}

  Expresión regular de números pares

  REGEX para los números decimales

Ejemplos

- Escriba las expresiones regulares para los siguientes lenguajes:
  - Expresión regular de números enteros impares
  - Expresión regular para un expresión aritmética binaria (suma, resta, multi, div) de literales enteros

Bibliografía

- Programming Language Processors in Java: compilers and interpreters. Watt, David, Brown, Deryck. Pearson Education. 2000

- Compilers: principles, techniques and tools (2da. ed.). Aho, Alfred. Pearson Education. 2007


TEC
Tecnológico
de Costa Rica