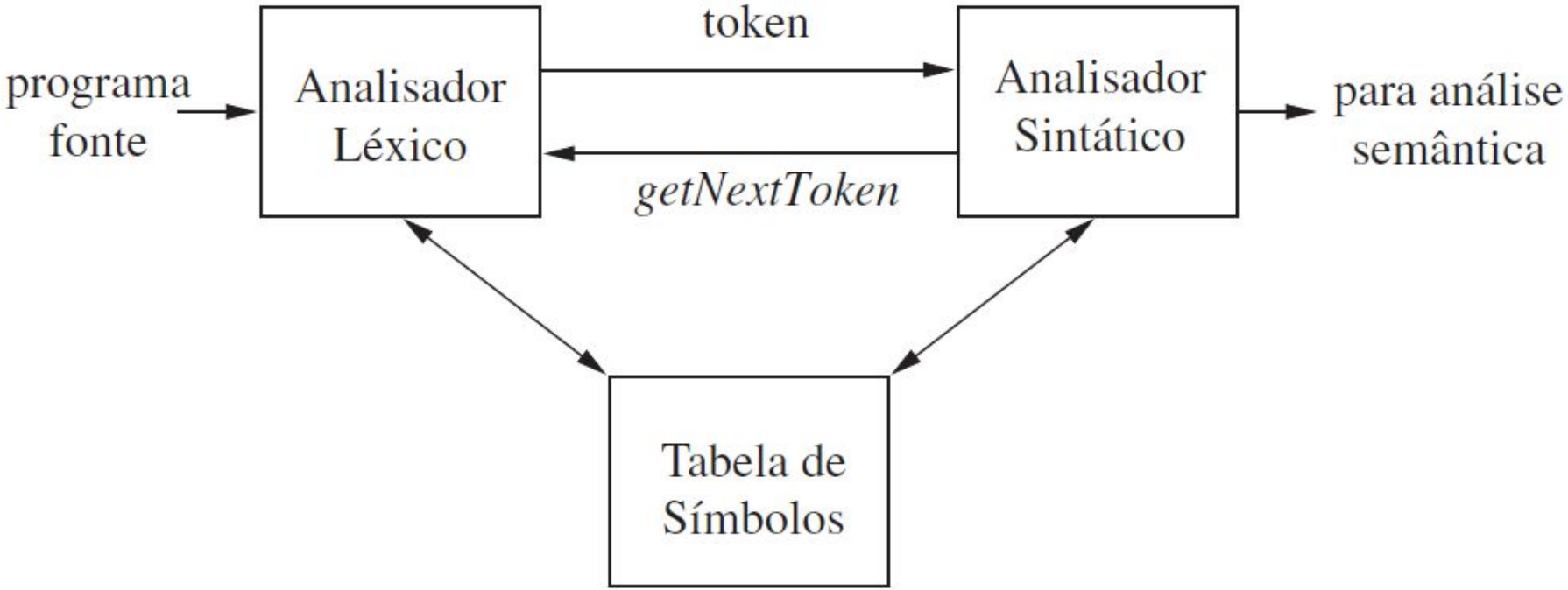
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CEARÁ
CAMPUS ARACATI

Bacharelado em Ciência da Computação

Disciplina: Compiladores
Aula 02 - Criando um analisador Léxico

Professor: Diego Rocha Lima

Analisador Léxico → Analisador Sintático



Tokens X Lexema

TOKEN	DESCRIÇÃO INFORMAL	EXEMPLOS DE LEXEMAS
if	caracteres `i`, `f`	`if`
else	caracteres `e`, `l`, `s`, `e`	`else`
comparison	`<` `or` `>` ou `<=` ou `>=` ou `==` ou `!=`	`<=`, `!=`
id	letra seguida por letras e dígitos	`pi`, `score`, `D2`
number	qualquer constante numérica	`3.14159`, `0`, `6.02e23`
literal	qualquer caractere diferente de `"`, cercado por `"`s	`"core dumped"`

1. Um token para cada palavra-chave. O padrão para uma palavra-chave é o mesmo que a própria palavra-chave.
2. Tokens para os operadores, seja individualmente ou em classes, como o token `comparison`
3. Um token representando todos os identificadores.
4. Um ou mais tokens representando constantes, como números e cadeias literais.
5. Tokens para cada símbolo de pontuação, como parênteses esquerdo e direito, vírgula e ponto-e-vírgula.

Exemplos de Tokens

```
E = M * C ** 2
```

são escritos a seguir como uma seqüência de pares.

< **id**, apontador para entrada da tabela de símbolos de E>
< **assign_op**>
< **id**, apontador para entrada da tabela de símbolos de M>
< **mult_op**>
< **id**, apontador para entrada da tabela de símbolos de C>
< **exp_op**>
<**number**, valor inteiro 2>

Operações

Operação	Definição e notação
União de L e M	$L \cup M = \{s \mid s \text{ está em } L \text{ ou } s \text{ está em } M\}$
Concatenação de L e M	$LM = \{st \mid s \text{ está em } L \text{ e } t \text{ está em } M\}$
Fecho Kleene de L	$L^* = \bigcup_{i=0}^{\infty} L^i$

- A união é a conhecida operação sobre conjuntos.
- A concatenação de linguagens são todas as cadeias formadas a partir de uma cadeia da primeira linguagem e uma cadeia da segunda linguagem, em todas as formas possíveis, e concatenando-as.
- O fecho (Kleene) de uma linguagem L, indicado por L*, é o conjunto de cadeias obtidas concatenando L zero ou mais vezes.

Exemplos

Considere que L seja o conjunto de letras {A, B,..., Z, a, b, z} e D seja o conjunto de dígitos {0, 1,... 9}.

1. L ∪ D é o conjunto de letras e dígitos — estritamente falando, a linguagem com 62 cadeias de tamanho um, cada uma tendo uma letra ou um dígito.
2. LD é o conjunto de 520 cadeias de tamanho dois, cada uma consistindo em uma letra seguida por um dígito.
3. L^3 é o conjunto de todas as cadeias de 3 letras.
4. L* é o conjunto de todas as cadeias de letras, incluindo , a cadeia vazia.
5. 5. L(L ∪ D)* é o conjunto de todas as cadeias de letras e dígitos começando com uma letra.
6. D+ é o conjunto de todas as cadeias de um ou mais dígitos.

Expressões Regulares

Nesta notação, se estabelecermos que *letra_* significa qualquer letra ou o sublinhado, e que *dígito_* significa qualquer dígito, então poderíamos descrever os identificadores da linguagem C por:

$$letra\_\ (letra\_ \mid dígito\ )^*$$

Existem quatro partes na indução, por meio das quais expressões regulares maiores são construídas a partir das menores. Suponha que r e s sejam expressões regulares denotando as linguagens L(r) e L(s), respectivamente.

1. $(r)|(s)$ é uma expressão regular denotando a linguagem $L(r) \cup L(s)$.
2. $(r)(s)$ é uma expressão regular denotando a linguagem $L(r)L(s)$.
3. $(r)^*$ é uma expressão regular denotando $(L(r))^*$.
4. (r) é uma expressão regular denotando $L(r)$. Essa última regra diz que podemos acrescentar pares de parênteses adicionais em torno das expressões sem alterar a linguagem que elas denotam.

Relembrando Leis Algébricas

Lei	Descrição
$r\|s = s\|r$	| é comutativo
$r\|(s\|t) = (r\|s)\|t$	| é associativo
$r(st) = (rs)t$	A concatenação é associativa
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$	A concatenação distribui entre |
$\epsilon r = r\epsilon = r$	ϵ é o elemento identidade para concatenação
$r^* = (r\|\epsilon)^*$	ϵ é garantido em um fechamento
$r^{**} = r^*$	* é igual potência

Construindo definições regulares

Os identificadores de C são cadeias de letras, dígitos e sublinhados. Aqui está uma definição regular para os identificadores da linguagem C. Por convenção, vamos usar itálico para os símbolos definidos em definições regulares.

$$
\begin{aligned}
\textit{letter\_} &\rightarrow \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z} \mid \texttt{\_} \\
\textit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \ldots \mid \texttt{9} \\
\textit{id} &\rightarrow \textit{lettter\_} \, (\, \textit{letter\_} \mid \textit{digit} \,)^{*}
\end{aligned}
$$

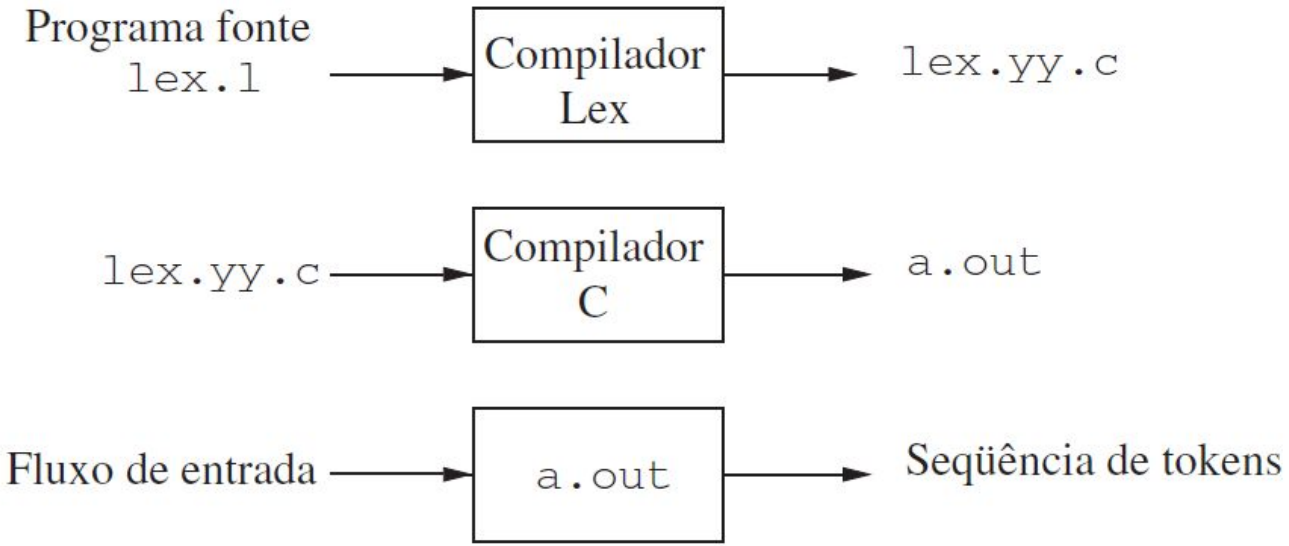
Construindo definições regulares

Números sem sinal (inteiros ou ponto flutuante) são cadeias como 5280, 0.01234, 6.336E4 ou 1.89E-4. A definição regular

$$
\begin{aligned}
\textit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \ldots \mid \texttt{9} \\
\textit{digits} &\rightarrow \textit{digit}\ \textit{digit}^{*} \\
\textit{optionalFraction} &\rightarrow \texttt{.}\ \textit{digits} \mid \epsilon \\
\textit{optionalExponent} &\rightarrow (\texttt{E}\ (\texttt{+} \mid \texttt{-} \mid \epsilon)\ \textit{digits}) \mid \epsilon \\
\textit{number} &\rightarrow \textit{digits}\ \textit{optionalFraction}\ \ \textit{optionalExponent}
\end{aligned}
$$

Utilizando o LEX para criar analisadores léxicos

Lex é um programa de computador que gera analisadores léxicos, que são ferramentas que mapeiam expressões regulares em blocos de código. Ele é usado para escrever programas em linguagem C ou C++. A seguir algumas expressões regulares com o LEX



Expressão	Casa com	Exemplo
c	o único caractere não operador *c*	`a`
c	o caractere *C* literalmente	`\*`
"*s*"	a cadeia *s* literalmente	`"***"`
.	qualquer caractere menos quebra de linha	`a.*b`
^	o início de uma linha	`^abc`
$	o fim de uma linha	`abc$`
[*s*]	qualquer um dos caracteres na cadeia *s*	`[abc]`
[^*s*]	qualquer caractere não presente na cadeia *s*	`[^abc]`
*r**	zero ou mais cadeias casando com *r*	`a*`
r+	uma ou mais cadeias casando com *r*	`a+`
r?	zero ou um *r*	`a?`
r{*m*,*n*}	entre *m* e *n* ocorrências de *r*	`a[1,5]`
r_1r_2	um r_1 seguido por um r_2	`ab`
$r_1 \mid r_2$	um r_1 ou um r_2	`a|b`
(*r*)	o mesmo que *r*	`(a|b)`
r_1/r_2	R_1 quando seguido por r_2	`abc/123`

Exemplos de Tokens com o LEX

digit	→	`[0-9]`
digits	→	*digit*$^{+}$
number	→	*digits* (. *digits*)? (E [+-]? *digits*)?
lettter	→	`[A-Za-z]`
id	→	*letter* (*letter* | *digit*)*
if	→	`if`
then	→	`then`
else	→	`else`
relop	→	`<	>	<=	>=	=	<>`

Valores de atributos

LEXEMAS	NOME DO TOKEN	VALOR DO ATRIBUTO
Qualquer *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Qualquer *id*	**id**	Apontador para entrada de tabela
Qualquer *number*	**number**	Apontador para entrada de tabela
<	**relop**	LT
<=	**relop**	LE
=	**relop**	EQ
<>	**relop**	NE
>	**relop**	GT
>=	**relop**	GE

Estrutura do LEX (arquivo.l)

```
declarações
%%
regras de tradução
%%
funções auxiliares
```

A seção de declarações inclui declarações de variáveis, constantes manifestas (identificadores declarados para significar uma constante, por exemplo, o nome de um token) e definições regulares.

Já as regras de produção tem o seguinte formato:

Padrão {Ação}

A terceira seção contém quaisquer funções adicionais usadas nas ações. Alternativamente, essas funções podem ser compiladas separadamente e carregadas com o analisador léxico

Primeiro programa em LEX (1/3)

```
%{
     /* definições de constantes manifestas
     LT, LE, EQ, NE, GT, GE,
     IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* definições regulares */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
```

Primeiro programa em LEX (2/3)

```
%%

{ws}        {/* nenhuma ação e nenhum retorno */}
if          {return(IF);}
then        {return(THEN);}
else        {return(ELSE);}
{id}        {yylval = (int) installID(); return(ID);}
{number}    {yylval = (int) installNum(); return(NUMBER);}
"<"         {yylval = LT; return(RELOP);}
"<="        {yylval = LE; return(RELOP);}
"="         {yylval = EQ; return(RELOP);}
"<>"        {yylval = NE; return(RELOP);}
">"         {yylval = GT; return(RELOP);}
">="        {yylval = GE; return(RELOP);}

%%
```

Primeiro programa em LEX (3/3)

```
int installID() {/* função para instalar o lexema, cujo
                    primeiro caractere é apontado por yytext,
                    e cujo tamanho é yyleng, na tabela de
                    símbolos, e retorna um apontador para lá */
}

int installNum() {/* semelhante a installID, mas coloca constantes
                     numéricas em uma tabela separada */
}
```

Na seção de função auxiliar, vemos duas dessas funções, installID() e installNum(). Assim como a parte da seção de declaração que aparece entre %{...%}, tudo na seção auxiliar é copiado diretamente para o arquivo lex.yy.c, mas pode ser usado nas ações.

Variáveis padrão do lex

yylval → variável inteira contendo o lexema

yytext → é um apontador para o início do lexema, ou seja, uma string que contém o valor atual, lido e armazenado no buffer

yyleng → é o tamanho do lexema encontrado

Vamos a um exemplo

aula1.l

```
%{
    //definições,funções... (código C)
%}
/*expressões regulares*/
NUM [0-9]+
STR [A-Za-z]+
SIMB [-+*/=]
DESC [ \n\t]
%%
    /*tokens*/
"FIM" {printf("Fim do programa\n");}
{NUM} {printf("Número: %s\n",yytext);}
{SIMB} {printf("Operação: %s\n",yytext);}
{STR} {printf("String: %s\n",yytext);}
{DESC} {printf("Caractere ignorado\n");}
%%
/*definições*/
int main(){
    yyin=fopen("codigo.d","r"); //abre o código fonte
    yylex();    //chama o analizador léxico
    fclose(yyin); //fecha o código fonte
return 0;
}
```

```
all: aula1.l
    clear
    flex -i aula1.l
    gcc lex.yy.c -o aula1 -lfl
    ./aula1
```

Exercício

1 - Criar um analisador léxico utilizando FLEX para reconhecer operações em uma calculadora simples

2 - Dar início ao seu projeto para criação de uma linguagem de programação