

Analizador Léxico

Trabalho Prático de Tradutores

Gabriel Crespo de Souza¹ - 14/0139982

Universidade de Brasília, Distrito Federal, Brasil
Departamento de Ciência da Computação
`cic@unb.br`
<https://cic.unb.br/>

1 Objetivos

A disciplina de Tradutores tem como finalidade fornecer o entendimento sobre o funcionamento dos diferentes tipos de tradutores, seus componentes, bem como as principais formas de implementá-los. Dado isso, o projeto principal da disciplina consiste na implementação de um compilador. Essa implementação se dará em etapas e esse documento apresentará o desenvolvimento da etapa inicial, a análise léxica.

Ao nível de simplificação, um subconjunto da linguagem C foi escolhido, a C-IPL[Nal], juntamente com uma nova primitiva que implementa e permite a manipulação de conceitos relacionados à listas. Nativamente, esses recursos não fazem parte da linguagem C, portanto, a sua inclusão visa facilitar o uso dessa estrutura de dados e suas principais operações. Listas são estruturas muito importantes devido sua flexibilidade em relação ao espaço e eficiência em relação ao tempo.

2 Desenvolvimento

O desenvolvimento do Analizador Léxico também se dá em fases. Inicialmente, as palavras-chave escolhidas para compor a gramática servem como base para a construção das primeiras expressões regulares. Logo após, é feita a análise dos símbolos auxiliares. Por fim, são apresentadas as sequências de *tokens* geradas, os erros encontrados, bem como a linha e coluna que foram encontrados.

Somente ao nível de reaproveitamento de código é que se fez necessária a implementação de duas funções, uma responsável por imprimir os *tokens* formatados em tela e a outra responsável por imprimir possíveis erros léxicos encontrados. Duas variáveis inteiras também foram adicionadas no escopo de declarações, a fim de controlar os números de linha e coluna que estão sendo analisadas. Condições para remoção de comentários foram adicionadas. Regras e expressões regulares também foram criadas.

3 Funcionamento

O Analisador Léxico lê um conjunto de caracteres de um programa fonte e os agrupa em sequências significativas de caracteres chamadas *lexemas* e para cada *lexema* ele produz como saída *tokens* [ALSU07]. O Analisador Léxico irá basear-se na gramática para detectar os *lexemas* da linguagem, depois apresentar os tokens formados em tela e a linha em que foram encontrados. E ao localizar os símbolos que não pertencem à linguagem, o analisador deverá relatar o problema e o indicar o seu local de ocorrência, sem que o problema encontrado interrompa o fluxo de execução do analisador. Qualquer tipo de comentário será desconsiderado.

O projeto foi estruturado em diretórios, de modo que os arquivos principais estivessem separados dos auxiliares. Na raiz do projeto, */14_0139982*, há quatro pastas: */src*, */lib*, */doc* e */tests*, bem como os arquivos *makefile* e *README.md*. Em *src* está o código do Analisador Léxico, *lexer.l*. Em *tests* estão quatro arquivos de testes, *correct_1.c*, *correct_2.c*, *incorrect_1.c* e *incorrect_2.c*. Na pasta *doc* se encontra o relatório do projeto. Enquanto a pasta *lib* se encontra vazia, no momento. Para que a compilação e a execução do Analisador Léxico se deem com sucesso, será necessário seguir os seguintes passos estando no diretório raiz:

1. Executar o comando *make* e os seguintes arquivos serão gerados:
 - *lex.yy.c*;
 - *tradutor* (arquivo executável);
2. Agora, basta chamar o código objeto juntamente com um único arquivo de teste presente em */tests* como parâmetro por exemplo:
 - *./tradutor tests/correct_1.c*;
3. Por fim, os *tokens* serão apresentados em tela, bem como os erros, caso existam.

4 Teste

Dos quatro arquivos disponibilizados para teste, dois apresentam erros léxicos, pois contém símbolos que não pertencem à definição da linguagem. O arquivo *incorrect_1.c* contém o símbolo \$ na linha 2 e coluna 14 e o símbolo # na linha 32 e colunas 19 e 20. Já o arquivo *incorrect_2.c* contém o símbolo @ nas linhas 19 e 24, coluna 11 e o símbolo # na linha 43 e coluna 47. Os arquivos *correct_1.c* e *correct_2.c* não deverão apresentar erro léxico algum.

Referências

- [ALSU07] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, Tools*. Pearson/Addison Wesley, 2nd edition, 2007.
- [Nal] Claudia Nalon. Trabalho prático - descrição da linguagem. <https://aprender3.unb.br/mod/page/view.php?id=464034>. Acessado pela última vez em 11/08/2021.

$\langle \textit{arithmetic_expression} \rangle$	$::= \langle \textit{arithmetic_expression} \rangle \langle \textit{arithmetic_operators} \rangle \langle \textit{arithmetic_expression} \rangle$ $\langle \textit{values} \rangle$
$\langle \textit{return} \rangle$	$::= \text{'return'} \langle \textit{values} \rangle \text{';'}$
$\langle \textit{io_expression} \rangle$	$::= \text{'read'} \text{'('} \langle \textit{identifier} \rangle \text{'')';}$ $\text{'write'} \text{'('} \langle \textit{string} \rangle \text{' } \langle \textit{logical_expression} \rangle \text{'')';}$ $\text{'writeln'} \text{'('} \langle \textit{string} \rangle \text{' } \langle \textit{logical_expression} \rangle \text{'')';}$
$\langle \textit{values} \rangle$	$::= \langle \textit{constant} \rangle \text{' } \langle \textit{identifier} \rangle$
$\langle \textit{variable_declaration} \rangle$	$::= \langle \textit{type} \rangle \langle \textit{identifier} \rangle \text{';'}$
$\langle \textit{constant} \rangle$	$::= \langle \textit{integer} \rangle \text{' } \langle \textit{float} \rangle \text{' } \langle \textit{nil} \rangle$
$\langle \textit{type} \rangle$	$::= \langle \textit{simple_type} \rangle \text{' } \langle \textit{list_type} \rangle$
$\langle \textit{assign_operator} \rangle$	$::= \text{'='}$
$\langle \textit{arithmetic_operators} \rangle$	$::= \text{'+' '-' '*' '/'}$
$\langle \textit{comparison_operators} \rangle$	$::= \text{'>' '<' '>=' '<=' '==' '!='}$
$\langle \textit{identifier} \rangle$	$::= (\langle \textit{letter} \rangle)(\langle \textit{letter} \rangle \langle \textit{digit} \rangle \text{'_'})^*$
$\langle \textit{nil} \rangle$	$::= \text{NIL}$
$\langle \textit{string} \rangle$	$::= \text{'"} \langle \textit{digit} \rangle \text{' } \langle \textit{letter} \rangle \text{''}$
$\langle \textit{float} \rangle$	$::= \langle \textit{digit} \rangle^* \text{'.'} \langle \textit{digit} \rangle^*$
$\langle \textit{integer} \rangle$	$::= \langle \textit{digit} \rangle^+$
$\langle \textit{list_type} \rangle$	$::= \langle \textit{simple_type} \rangle \text{ list}$
$\langle \textit{simple_type} \rangle$	$::= \text{int float}$
$\langle \textit{letter} \rangle$	$::= [\text{a-zA-Z}]$
$\langle \textit{digit} \rangle$	$::= [0-9]$
$\langle \textit{special_characters} \rangle$	$::= \text{'\s' '\t' '\n'}$

Token	Expressão
letter	[a-zA-Z]
digit	[0-9]
alphanumeric	{letter} {digit}
identifier	{letter}+({alphanumeric} _)*
integer	digit>*
float	[0-9]*(\.[0-9]+)
char	(\'(. \a \b \f \n \r \t \v \\\\ \\\' \\\" \\\\?)\')
string	\"[^\"]*\" \'[^\']*\'
main	\"main\"
int_type	\"int\"
float_type	\"float\"
list_type	\"list\"
const_nil	\"NIL\"
if	\"if\"
else	\"else\"
for	\"for\"
return	\"return\"
read	\"read\"
write	\"write\"
writeln	\"writeln\"
sum_op	\"+\"
sub_op	\"_\"
mult_op	\"*\"
div_op	\"/\"
list_header	\"?\"
list_constructor	\".\"
list_tail	\"%\"
list_map	\">>\"
list_filter	\"<<\"
exc_op	\"!\"
or_op	\" \"
and_op	\"&&\"
equal_op	\"==\"
diff_op	\"!=\"
grt_op	\">\"
lst_op	\"<\"
grt_eq_op	\">=\"
lst_eq_op	\"<=\"
assign_op	\"=\"
r_paren	\"(\"
l_paren)\"
r_brack	{\"
l_brack	}\"
semi	\".\"
comma	\",\"
reffer	\"&\"