

Analizador Léxico

Heitor de Lima Belém^[160123950]

Universidade de Brasília cic@unb.com

1 Motivação

Este trabalho tem como objetivo a apresentação da análise léxica, que é a primeira fase do processo de compilação de um programa. Para a abordagem prática, com o intuito de fixar os conhecimentos obtidos do livro base da disciplina [ALSU07], foi proposto o desenvolvimento de um analisador léxico para uma nova linguagem, *C-IPL* [Nal], baseada nos princípios da linguagem *C*. O objetivo dessa linguagem é acrescentar uma estrutura de dados não existente em *C*, as listas.

Listas implementam uma coleção organizada de valores, assim como os *arrays*, entretanto, elas possuem operações especiais de acesso, adição e remoção de itens, que podem variar de linguagem para linguagem. Para a *C-IPL* [Nal], foram apresentadas operações de acesso aos elementos através dos operadores ‘?’ e ‘!’, operações de remoção utilizando o operador ‘%’ e de atribuição por meio do ‘:’. Por fim, são descritas funções para operar sobre as listas, sendo elas: filter, representada pelo operador binário ‘<<’ e map, representada por ‘>>’.

2 Descrição da análise léxica

Para implementar o analisador léxico da linguagem proposta, foi utilizado o *software FLEX (Fast Lexical Analyzer)*, uma ferramenta geradora de programas que reconhecem padrões léxicos em texto [Est]. A estrutura de um arquivo reconhecido pelo *FLEX*, que possui a extensão *.l*, é dividida em três partes:

1. Definições

Definições de funções, constantes, variáveis globais e inclusão de bibliotecas. Para este trabalho, foram criadas 3 variáveis globais: *errors_count*, *line_idx* e *column_idx*, que representam, respectivamente, a quantidade de erros obtidos durante a análise e o número da linha e coluna atual.

2. Regras

Aqui são escritas as expressões regulares que vão procurar padrões no arquivo juntamente com as ações a serem tomadas ao encontrar tais padrões.

3. Código

Nesta seção, encontra-se o código da função principal do arquivo com extensão *.l*, é aqui que será colocado o código gerado pelo *FLEX*.

3 Descrição dos arquivos de teste

Os arquivos utilizados para verificar o funcionamento do analisador léxico desenvolvido no trabalho estão no subdiretório */tests*. Nesse diretório, estão arquivos com o prefixo *success_*, que representam os casos em que a análise léxica não identifica nenhum erro durante o processo. Já os arquivos com o prefixo *wrong_* englobam os casos em que o analisador identifica erros léxicos na análise.

Os erros apresentados para cada arquivo estão identificados abaixo:

```
- wrong_ex1.c

Line: 2 | Column: 9 => ERROR: Invalid token '549a'
Line: 3 | Column: 11 => ERROR: Invalid token '1$bas'
Line: 4 | Column: 10 => ERROR: Unexpected character '#'
Line: 6 | Column: 13 => ERROR: Unexpected character '\ '
Line: 9 | Column: 11 => ERROR: Unexpected character '~'

- wrong_ex2.c

Line: 2 | Column: 15 => ERROR: Unexpected character '~'
Line: 8 | Column: 10 => ERROR: Unexpected character '^'
Line: 28 | Column: 17 => ERROR: Unexpected character '['
Line: 28 | Column: 19 => ERROR: Unexpected character ']'
```

4 Compilação e execução do programa.

Requisitos para compilação: software *FLEX* (versão 2.6.4), o compilador *GCC* (versão 11.1.0), GNU Make (versão 4.3) e valgrind (versão 3.17.0). Com isso devidamente instalado, é possível prosseguir para os seguintes passos.

– No diretório raiz do projeto, executar os comandos:

```
$ make
$ ./tradutor ./tests/<nome_do_arquivo>.c
```

Referências

- [ALSU07] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, Tools*. Pearson/Addison Wesley, 2nd edition, 2007.
- [Est] W. Estes. Flex: Fast lexical analyser generator. <https://github.com/westes/flex>. Online; acessado 08 de Agosto de 2021.
- [Gup21] Ajay Gupta. The syntax of c in backus-aur form. <https://tinyurl.com/max5eep>, Online; acessado 08 de Agosto de 2021.
- [Nal] Claudia Nalon. Trabalho prático - descrição da linguagem. <https://aprender3.unb.br/mod/page/view.php?id=464034>. Acessado pela última vez em 10/08/2021.

A Gramática

Gramática que descreve o compilador da linguagem C-IPL [Nal]. A estrutura do [Gup21] foi utilizada como base.

$\langle \text{program} \rangle$	$::= \langle \text{function_def} \rangle$ $\langle \text{function_def} \rangle \langle \text{program} \rangle$
$\langle \text{function_def} \rangle$	$::= \langle \text{type} \rangle \langle \text{identifier} \rangle ' (' \langle \text{function_parameters} \rangle ') ' \langle \text{multi_statements} \rangle$
$\langle \text{function_parameters} \rangle$	$::= \langle \text{type} \rangle \langle \text{identifier} \rangle ' , ' \langle \text{function_parameters} \rangle$ $\langle \text{type} \rangle \langle \text{identifier} \rangle$
$\langle \text{multi_statements} \rangle$	$::= ' \{ ' \langle \text{single_statement} \rangle ' \}$
$\langle \text{single_statement} \rangle$	$::= \langle \text{for} \rangle$ $\langle \text{conditional} \rangle$ $\langle \text{multi_statements} \rangle$ $\langle \text{variable_declaration} \rangle$ $\langle \text{assign_statement} \rangle$ $\langle \text{filter_list_expression} \rangle$ $\langle \text{map_list_expression} \rangle$ $\langle \text{push_list_expression} \rangle$ $\langle \text{single_statement} \rangle$
$\langle \text{for} \rangle$	$::= ' \text{for} ' ' (' \langle \text{assign_expression} \rangle ';' \langle \text{logical_expression} \rangle ';' ' \langle \text{identifier} \rangle \langle \text{unary_operators} \rangle ') ' \langle \text{single_statement} \rangle$ $' \text{for} ' ' (' \langle \text{assign_expression} \rangle ';' \langle \text{logical_expression} \rangle ';' ' \langle \text{assign_expression} \rangle ') ' \langle \text{single_statement} \rangle$
$\langle \text{conditional} \rangle$	$::= ' \text{if} ' ' (' \langle \text{logical_expression} \rangle ') ' \langle \text{single_statement} \rangle ' \text{else} ' \langle \text{single_statement} \rangle$ $' \text{if} ' ' (' \langle \text{logical_expression} \rangle ') ' \langle \text{multi_statements} \rangle ' \text{else} ' \langle \text{single_statement} \rangle$ $' \text{if} ' ' (' \langle \text{logical_expression} \rangle ') ' \langle \text{multi_statements} \rangle ' \text{else} ' \langle \text{multi_statements} \rangle$ $' \text{if} ' ' (' \langle \text{logical_expression} \rangle ') ' \langle \text{single_statement} \rangle ' \text{else} ' \langle \text{multi_statements} \rangle$ $' \text{if} ' ' (' \langle \text{logical_expression} \rangle ') ' \langle \text{single_statement} \rangle$ $' \text{if} ' ' (' \langle \text{logical_expression} \rangle ') ' \langle \text{multi_statements} \rangle$
$\langle \text{filter_list_expression} \rangle$	$::= \langle \text{identifier} \rangle ' = ' \langle \text{identifier} \rangle ' < < ' \langle \text{identifier} \rangle ';' ';$
$\langle \text{map_list_expression} \rangle$	$::= \langle \text{identifier} \rangle ' = ' \langle \text{identifier} \rangle ' > > ' \langle \text{identifier} \rangle ';' ';$
$\langle \text{push_list_expression} \rangle$	$::= \langle \text{identifier} \rangle ' = ' \langle \text{identifier} \rangle ' : ' \langle \text{identifier} \rangle ';' ';$
$\langle \text{assign_expression} \rangle$	$::= \langle \text{identifier} \rangle \langle \text{assign_operators} \rangle \langle \text{values} \rangle ';' ';$ $\langle \text{logical_expression} \rangle ' ' \langle \text{comparison_expression} \rangle$ $\langle \text{comparison_expression} \rangle$
$\langle \text{logical_expression} \rangle$	$::= \langle \text{logical_expression} \rangle ' \& \& ' \langle \text{comparison_expression} \rangle$ $\langle \text{logical_expression} \rangle ' ' \langle \text{comparison_expression} \rangle$ $\langle \text{comparison_expression} \rangle$

$\langle comparison_expression \rangle$	$::= \langle comparison_expression \rangle \langle comparison_operators \rangle$ $\quad \langle arithmetic_expression \rangle$ $\quad \quad \langle arithmetic_expression \rangle$
$\langle arithmetic_expression \rangle$	$::= \langle arithmetic_expression \rangle \langle arithmetic_operators \rangle \langle arithmetic_expression \rangle$ $\quad \quad \langle values \rangle$
$\langle return \rangle$	$::= \text{'return'} \langle values \rangle \text{';'}$
$\langle io_expression \rangle$	$::= \text{'read'}(' \langle identifier \rangle \text{'}) \text{';'}$ $\quad \quad \text{'write'}(' \langle string \rangle \langle logical_expression \rangle \text{'}) \text{';'}$ $\quad \quad \text{'writeln'}(' \langle string \rangle \langle logical_expression \rangle \text{'}) \text{';'}$
$\langle values \rangle$	$::= \langle constant \rangle \langle identifier \rangle$
$\langle variable_declaration \rangle$	$::= \langle type \rangle \langle identifier \rangle \text{';'}$
$\langle constant \rangle$	$::= \langle integer \rangle \langle float \rangle \langle nil \rangle$
$\langle type \rangle$	$::= \langle simple_type \rangle \langle list_type \rangle$
$\langle assign_operators \rangle$	$::= \text{'='} \text{'+='} \text{'-='} \text{'*='} \text{'/='}$
$\langle unary_operators \rangle$	$::= \text{'++'} \text{'--'}$
$\langle arithmetic_operators \rangle$	$::= \text{'+'} \text{'-'} \text{'*'} \text{'/'}$
$\langle comparison_operators \rangle$	$::= \text{'>'} \text{'<'} \text{'>='} \text{'<='} \text{'=='} \text{'!='}$
$\langle identifier \rangle$	$::= (\langle character \rangle \text{'_'})(\langle character \rangle \langle digit \rangle \text{'_'})^*$
$\langle nil \rangle$	$::= \text{NIL}$
$\langle string \rangle$	$::= \text{'\"} .* \text{'\"}$
$\langle float \rangle$	$::= \langle digit \rangle^* \text{'.'} \langle digit \rangle^+$
$\langle integer \rangle$	$::= \langle digit \rangle^+$
$\langle list_type \rangle$	$::= \langle simple_type \rangle \text{ list}$
$\langle simple_type \rangle$	$::= \text{int} \text{float}$
$\langle character \rangle$	$::= [\text{a-zA-Z}]$
$\langle digit \rangle$	$::= [0-9]$

B Definições Regulares

Tokens	Expressões Regulares
digit	[0-9]
character	[a-zA-Z]
integer	{digit}+
float	{digit}*.{digit}+
nil	'NIL'
t_simple	int float
t_list	{t_simple} list
identifier	{character} [_]({character} {digit} [_])*
list_operators	'?' ':' '<<' '>>' '%'
comparison_operators	'>' '<' '>=' '<=' '==' '!='
arithmetic_operators	'+' '-' '*' '/'
unary_operators	'++' '--'
assign_operators	'=' '+=' '- =' '* =' '/ ='
logical_operators	'&&' ' '
reserved_words	'for' 'if' 'else' 'return'
io_operations	'read' 'write' 'writeln'
delimiters	(' ') '{' '}' ';' ','
string	" . * "
exclamation_operator	'!'