

Analizador Léxico

João Victor Cabral de Melo¹

Universidade de Brasília

Abstract. This project is to show what we've done and learned in the course of compilers and the grammar of experimental language C-IPL.

Keywords: Compilers · Grammar · Lexical Analyser · Flex

1 Introdução

1.1 Motivação

A maior motivação para a implementação de um tradutor é o entendimento por trás das abstrações do compilador facilitando assim a percepção do aluno enquanto as linguagens de programação.

1.2 Nova Primitiva

A motivação para entrada da nova primitiva *list* é a possibilidade de uma pequena abordagem do paradigma funcional. A importância se dá em problemas onde se precisa de uma estrutura de dados com um tamanho não definido previamente. Facilitando esse conjunto específico de problemas pela não implementação da estrutura de dados manualmente.

2 Análise Léxica

A função do *scanner* do compilador é identificar *tokens* na linguagem e coloca-los na tabelas de símbolos para a avaliação do sintático. Neste projeto foi utilizado o **flex** para tal função, além disso foram feitas tais modificações no arquivo como as variáveis para contagem de linhas, colunas e erros além do tratamento de caso em que não consiga abrir o arquivo a ser escaneado, para ser input de usuário os *tokens* a serem lido. A tabela de símbolos será implementada como uma lista ligada sendo cada valor dessa lista um:

```
struct node {  
    char *Value;  
    char *TOKEN;  
    node *next;  
};
```

3 Testes

No diretório de testes possui dos arquivos com erros léxicos **error.c** e **error1.c**, sendo o segundo com uma *string* que não fechou aspas, e dois arquivos sem erros léxicos **success.c** e **success1.c**. Sendo o primeiro erro do arquivo **error.c** e o segundo erro do arquivo **error1.c**.

```
Error: $ Unidentified token at line: 3 and column: 15
Error: " Unidentified token at line: 2 and column: 8
```

4 Instruções

Para se compilar o analisador léxico utiliza-se os seguintes comandos:

```
lex lexico.l
cc -Wall lex.yy.c -lfl -o tradutor
./tradutor ../test/error.c
./tradutor ../test/error1.c
./tradutor ../test/success.c
./tradutor ../test/success1.c
```

Caso de algum problema com a compilação do arquivo **lex.yy.c** descomente o seguinte código:

```
/*%option noyywrap*/
```

E tente compilar novamente sem a *flag* **-lfl** dos comandos acima.

References

1. AV Aho, R Sethi, JD Ullman: Compilers: Principles, Techniques, and Tools. Pearson Education. Segunda Edição, Agosto 2006.
2. Levine J: Flex and Bison. O'Reilly. Primeira Edição, Agosto 2009.
3. Flex Manual: <https://westes.github.io/flex/manual/>. Acessado pela última vez: 09/08/2020 – 16:24

Appendices

Appendix A

Tabelas de Símbolos

Token	Informal Description	Sample Lexemes
ID	qualquer letra com '.' seguido de um número	jujuba_doce
LIST	qualquer tipo do list	list
TYPES	tipos int e float	int float
NULL	constante para listas vazias	NIL
STRING	qualquer caracter dentro de " "	"chocolate"
ASSIGN	Um único caractere =	=
KEYWORD	Palavras como if else for e return	if else for return
OUTPUT	Palavras da saída de dados da linguagem	write writeln
INPUT	Palavra de entrada de dados	read
CONSTRUCTOR	Palavras para construtores de lista	rvalue lvalue
INFIX	Caracter para fazer construtor uma operação infixa :	
FUNCTION	Caracteres para as função de map e filter	>><<
COMMENT	Qualquer comentário em linha com // ou comentário multilinha com /**/	//pudim /* pudim é bom /
+ - - * - /	Operações aritmeticas	+ - * /
— — &&	Operações Lógicas	&& —
>—<—>=—<=—==—!=	Operação de comparação	<<= >>= == !=
{ }	Escopo	{ }
()	Expressão	()
;	Separador de Expressão	;
?	Operação head da lista	?
!	Operação tail da lista ou negação	!
%	Operação tail destrutiva da lista	%
123	Número inteiro	123
.5	Número decimal	.5

Appendix B

Gramática

- $\langle \text{program} \rangle ::= \langle \text{paramlist} \rangle$ (1)
- $\langle \text{paramlist} \rangle ::= \langle \text{param} \rangle \langle \text{paramlist} \rangle \mid \langle \text{param} \rangle$ (2)
- $\langle \text{param} \rangle ::= \langle \text{variableParam} \rangle \mid \langle \text{functionParam} \rangle$ (3)
- $\langle \text{variableParam} \rangle ::= \langle \text{type} \rangle \langle \text{id} \rangle \langle \text{d} \rangle \mid \langle \text{type} \rangle \langle \text{listType} \rangle \langle \text{id} \rangle \langle \text{d} \rangle$ (4)
- $\langle \text{vFunction} \rangle ::= \langle \text{type} \rangle \langle \text{id} \rangle \mid \langle \text{type} \rangle \langle \text{listType} \rangle \langle \text{id} \rangle$ (5)
- $\langle \text{pList} \rangle ::= \langle \text{vFunction} \rangle \langle \text{v} \rangle \langle \text{pList} \rangle \mid \langle \text{vFunction} \rangle$ (6)
- $\langle \text{functionParam} \rangle ::= \langle \text{type} \rangle \langle \text{id} \rangle \langle \text{p} \rangle \langle \text{pList} \rangle \langle \text{p} \rangle \langle \text{s} \rangle$ (7)
- $\langle \text{expression} \rangle ::= \langle \text{arithmExpression} \rangle \mid \langle \text{logicalExpression} \rangle \mid$ (8)
- $\langle \text{comparisonExpression} \rangle \mid$ (9)
- $\langle \text{expression} \rangle \langle \text{d} \rangle \langle \text{expression} \rangle$ (10)
- $\langle \text{inputExpression} \rangle ::= \langle \text{input} \rangle \langle \text{p} \rangle \langle \text{p} \rangle \langle \text{d} \rangle$ (11)
- $\langle \text{outputExpression} \rangle ::= \langle \text{output} \rangle \langle \text{p} \rangle \langle \text{string} \rangle \langle \text{p} \rangle \langle \text{d} \rangle$ (12)
- $\langle \text{arithmExpression} \rangle ::= \langle \text{number} \rangle \langle \text{arOp} \rangle \langle \text{number} \rangle$ (13)
- $\langle \text{logicalExpression} \rangle ::= \langle \text{loOp} \rangle \langle \text{id} \rangle \mid \langle \text{id} \rangle \langle \text{loOp} \rangle \langle \text{id} \rangle$ (14)
- $\langle \text{comparisonExpression} \rangle ::= \langle \text{id} \rangle \langle \text{coOp} \rangle \langle \text{id} \rangle$ (15)
- $\langle \text{listExpression} \rangle ::= (\langle \text{head} \rangle \mid \langle \text{tail} \rangle \mid \langle \text{dTail} \rangle) \langle \text{id} \rangle \mid$ (16)
- $\langle \text{id} \rangle \langle \text{infix} \rangle \langle \text{id} \rangle \mid$ (17)
- $\langle \text{cosnstructor} \rangle \langle \text{id} \rangle \mid$ (18)
- $\langle \text{id} \rangle \langle \text{function} \rangle \langle \text{id} \rangle$ (19)
- $\langle \text{assignExpression} \rangle ::= \langle \text{id} \rangle \langle \text{assign} \rangle \langle \text{id} \rangle \mid$ (20)
- $\langle \text{id} \rangle \langle \text{assign} \rangle \langle \text{number} \rangle$ (21)
- $\langle \text{keywordExpression} \rangle ::= \langle \text{keyword} \rangle \langle \text{p} \rangle \langle \text{expression} \rangle \langle \text{p} \rangle \langle \text{s} \rangle$ (22)
- $\langle \text{arOp} \rangle ::= + \mid - \mid * \mid /$ (23)
- $\langle \text{loOp} \rangle ::= ! \mid \&\& \mid ||$ (24)
- $\langle \text{coOp} \rangle ::= > \mid < \mid >= \mid <= \mid != \mid ==$ (25)
- $\langle \text{s} \rangle ::= \{ \mid \}$ (26)
- $\langle \text{p} \rangle ::= (\mid)$ (27)
- $\langle \text{d} \rangle ::= ;$ (28)
- $\langle \text{head} \rangle ::= ?$ (29)
- $\langle \text{tail} \rangle ::= !$ (30)

$\langle \text{dTail} \rangle ::= \% \quad (31)$

$\langle \text{type} \rangle ::= \text{int} \mid \text{float} \quad (32)$

$\langle \text{listType} \rangle ::= \text{list} \quad (33)$

$\langle v \rangle ::= , \quad (34)$

$\langle \text{null} \rangle ::= \text{NIL} \quad (35)$

$\langle \text{assign} \rangle ::= = \quad (36)$

$\langle \text{keyword} \rangle ::= \text{for} \mid \text{if} \mid \text{else} \mid \text{return} \quad (37)$

$\langle \text{output} \rangle ::= \text{writeln} \mid \text{write} \quad (38)$

$\langle \text{input} \rangle ::= \text{read} \quad (39)$

$\langle \text{constructor} \rangle ::= \text{rvalue} \mid \text{lvalue} \quad (40)$

$\langle \text{infix} \rangle ::= : \quad (41)$

$\langle \text{function} \rangle ::= >> \mid << \quad (42)$