

Trabalho Prático 1 - Analisador Léxico

João Victor Bohrer Munhoz^[16/0071101]

Universidade de Brasília, Brasília, DF, 70910-900, Brasil
160071101@aluno.unb.br

1 Motivação

Para a matéria de Tradutores da Universidade de Brasília, o trabalho obrigatório consiste em criar um tradutor que consiga transformar um arquivo escrito na linguagem especificada pela docente em um código executável através das ferramentas *Flex* e *Bison*.

A linguagem em questão é a **C-IPL**^[Nal21], que foi projetada para facilitar o tratamento de listas em programas escritos em C. Para isso, foi introduzida uma nova primitiva de dados para listas e operações sobre esta nova primitiva. As demais primitivas e comandos de C têm semântica padrão.

Espera-se que ao implementar o tradutor, o aluno entenda mais profundamente de que maneira os programas são compilados e como certas características de algumas linguagens deixam esse processo mais fácil ou mais difícil.

2 Descrição

Nessa linguagem **C-IPL**, foi criada uma nova primitiva *list* para lidar com listas, uma nova constante **NIL** para indicar o fim de uma lista e novas operações sobre as listas, podendo-se dividi-las em construtores, operações unárias e operações binárias. A explicação detalhada dos operadores se encontra na especificação da linguagem.^[Nal21]

Para realizar a análise léxica, utilizamos em grande parte as operações regulares. Portanto, começamos o arquivo com várias macros definindo algumas expressões regulares que usaremos mais para frente, tais como *DIGIT*, que identifica números, *TYPE*, que identifica os tipos de declaração, *NIL* que identifica a constante **NIL**, *MINUS* que identifica o símbolo de menos, *SUMOP* que identifica o símbolo de soma, *MULOP* que identifica as operações de divisão e multiplicação, *EXCLAM* que identifica o símbolo de exclamação, *LOGOP* que identifica as operações lógicas padrão de C, *RELOP* que identifica as operações relacionais usuais de C, *ASSIGN* que identifica o operador de atribuição, *KEYWORDS* que identifica as estruturas de *loop* especificadas na linguagem, *READWRITE* que identifica as operações de leitura e escrita da linguagem, *LISTOP* que identifica as operações sobre listas também especificadas na linguagem e *ID* que checa por qualquer outro identificador que não os já reservados anteriormente. Checou-se também por comentários e *strings* através de máquinas de estados ao longo do programa.

A partir do momento que identificamos esses lexemas, criamos uma *struct* para gerar o respectivo *token* com: o **tipo** do *token*, com todos os tipos presentes na **tabela** 1, a **posição**, em linha e coluna, em que se encontra o lexema no arquivo e seu **atributo opcional**, o qual pode variar de acordo com qual tipo de *token* estamos lidando.

Para qualquer *token* que seja uma declaração de função, declaração de variável ou declaração de parâmetro, esse *token* será inserido na tabela de símbolos, a qual será muito importante para as partes mais adiante do tradutor. No caso desse trabalho, pretende-se usar uma árvore para representar a tabela de símbolos, onde cada nó é uma lista com o *token*, se o *token* é uma função ou uma variável e de que tipo é essa função ou variável.

Cada nó também representa um escopo, e a cada novo escopo que surja dentro de um desses nós, esse nó gera um filho, o qual é um novo escopo dentro do escopo do pai, desse modo controlando os escopos do programa e checando por possíveis declarações repetidas ou uso de variáveis ou funções inexistentes.

3 Arquivos de Teste

Ao total foram disponibilizados 5 arquivos de teste junto com essa entrega, três deles corretos e dois deles com erros léxicos apontados pelo analisador. Os itens corretos são:

- **Ex_Correct_0.c:** Arquivo de teste fornecido junto com a descrição da linguagem.
- **Ex_Correct_1.c:** Arquivo próprio que testa metade das instruções.
- **Ex_Correct_2.c:** Arquivo próprio que testa a outra metade das instruções.

Os itens incorretos são;

- **Ex_Incorrect_1.c:** Há um ponto extra no float da linha 17 na coluna 16, há um símbolo \$ que não faz parte da linguagem na linha 27 e coluna 17 e há uma *string* aberta mas não fechada na linha 32 e coluna 13.
- **Ex_Incorrect_2.c:** Há um símbolo ^ que não faz parte da linguagem na linha 10 e coluna 33 e há um comentário em bloco não fechado que se estende até o final do arquivo na linha 20 e coluna 5.

4 Instruções de Execução

O programa foi desenvolvido e compilado utilizando as seguintes versões das ferramentas:

- **OS:** **Fedora 34** (Workstation Edition) x86_64
- **Kernel:** 5.13.8
- **Flex:** 2.6.4
- **GCC:** 11.2.1



Para compilar, simplesmente entre na pasta raiz e digite:

```
$ make tradutor_lex
```

Uma vez com o arquivo *tradutor* já na pasta raiz, digite o seguinte comando para rodar os programas no arquivo de teste:

```
$ ./tradutor <caminho para o arquivo>
```

References

- [Nal21] Claudia Nalon. Trabalho prático - descrição da linguagem. <https://aprender3.unb.br/mod/page/view.php?id=464034>, Agosto 2021. Acessado pela última vez em 10/08/2021.

Table 1. Projeto Léxico do trabalho

Token	Expressão Regular
Tipo	int float
Constante Int	[0-9]+
Constante Float	[0-9]+ "." [0-9]+
NIL	(NIL)
Símbolo de Menos	[-]
Operação de Soma	[+]
Operação de Multiplicação	[*/]
Símbolo de Exclamação	[!]
Operação Lógica	(&&) (\\ \\)
Operação Relacional	[<] (<=) [>] (>=) (==) (!=)
Atribuição	[=]
Keyword	if else for return
I/O	read write writeln
String	\"(\\. [^\"])*\"
Operação de Listas	[:?%] (>>) (<<)
ID	[A-Za-z-][A-Za-z0-9-]*
Ponto e vírgula	[;]
Abre parêntesis	[([
Fecha parêntesis)]
Abre chaves	[{
Fecha chaves	}
Vírgula	[,]

1. $\text{program} \rightarrow \text{declList}$
2. $\text{declList} \rightarrow \text{declList decl} \mid \text{decl}$
3. $\text{decl} \rightarrow \text{varDecl} \mid \text{funDecl}$
4. $\text{varDecl} \rightarrow \text{typeList varDecl} \mid \text{ID}$
5. $\text{typeList} \rightarrow \text{type typeList} \mid \text{type}$
6. $\text{type} \rightarrow \text{int} \mid \text{float} \mid \text{list}$
7. $\text{funDecl} \rightarrow \text{typeList ID (params) stmt} \mid \text{ID (params) stmt}$
8. $\text{params} \rightarrow \text{paramList} \mid \epsilon$
9. $\text{paramList} \rightarrow \text{paramTypeList , paramList} \mid \text{paramTypeList}$
10. $\text{paramTypeList} \rightarrow \text{typeList ID}$
11. $\text{stmt} \rightarrow \text{expStmt} \mid \text{compoundStmt} \mid \text{ifStmt} \mid \text{forStmt} \mid \text{returnStmt}$
12. $\text{expStmt} \rightarrow \text{exp} ; \mid ;$
13. $\text{compoundStmt} \rightarrow \{ \text{localDecls stmtList} \}$
14. $\text{localDecls} \rightarrow \text{localDecls varDecl} \mid \epsilon$
15. $\text{stmtList} \rightarrow \text{stmtList stmt} \mid \epsilon$
16. $\text{ifStmt} \rightarrow \text{if simpleExp compoundStmt} \mid \text{if simpleExp compoundStmt elseStmt}$
17. $\text{elseStmt} \rightarrow \text{ifStmt} \mid \text{compoundStmt}$
18. $\text{forStmt} \rightarrow \text{for (simpleExp ; relExp ; simpleExp) compoundStmt}$
19. $\text{returnStmt} \rightarrow \text{return} ; \mid \text{return exp} ;$
20. $\text{exp} \rightarrow \text{ID} = \text{exp} \mid \text{simpleExp}$
21. $\text{simpleExp} \rightarrow \text{simpleExp} \parallel \text{andExp} \mid \text{andExp}$
22. $\text{andExp} \rightarrow \text{andExp} \&\& \text{unaryRelExp} \mid \text{unaryRelExp}$
23. $\text{unaryRelExp} \rightarrow ! \text{unaryRelExp} \mid \text{relExp}$
24. $\text{relExp} \rightarrow \text{sumExp relOp sumExp} \mid \text{sumExp}$
25. $\text{relOp} \rightarrow < \mid <= \mid > \mid >= \mid == \mid !=$
26. $\text{sumExp} \rightarrow \text{sumExp sumOp sumExp} \mid \text{mulExp}$
27. $\text{sumOp} \rightarrow - \mid +$
28. $\text{mulExp} \rightarrow \text{mulExp mulOp mulExp} \mid \text{listExp}$
29. $\text{mulOp} \rightarrow * \mid /$
30. $\text{listExp} \rightarrow \text{listExp listOp listExp} \mid \text{unaryExp}$
31. $\text{listOp} \rightarrow : \mid >> \mid <<$
32. $\text{unaryExp} \rightarrow \text{unaryListOp factor} \mid \text{factor}$
33. $\text{unaryOp} \rightarrow - \mid ? \mid ! \mid \%$
34. $\text{factor} \rightarrow (\text{exp}) \mid \text{call constant} \mid \text{readFunc} \mid \text{writeFunc} \mid \text{ID}$
35. $\text{call} \rightarrow \text{ID (args)}$
36. $\text{readFunc} \rightarrow \text{read (ID)} ;$
37. $\text{writeFunc} \rightarrow \text{writeType (ID)} \mid \text{writeType (constant)}$
38. $\text{writeType} \rightarrow \text{write} \mid \text{writeln}$
39. $\text{args} \rightarrow \text{argList} \mid \epsilon$
40. $\text{argList} \rightarrow \text{argList , exp} \mid \text{exp}$
41. $\text{constant} \rightarrow \text{NUMCONST} \mid \text{FLOATCONST} \mid \text{STRINGCONST} \mid \text{NIL}$