Analisador Léxico - Trabalho Prático

João Lucas Azevedo Yamin Rodrigues da Cunha - 17/0013731

Universidade de Brasília
Brasília - DF, CEP 70910-900
jlyaminc@gmail.com

1 Motivação

A disciplina de Tradutores é a última da cadeia obrigatória do curso de Ciência da Computação. Ela aborda, em sua ementa, conteúdos de diversos períodos do curso. Por isso, a implementação do analisador léxico, bem como os demais componentes do projeto da disciplina a serem implementados futuramente, permite a assimilação da união do aprendizado acumulado ao longo do curso.

Para este projeto será utilizada a linguagem C-IPL, um subconjunto da linguagem C para tratamento de listas. Dentro da computação, a lista é uma das estruturas de dados mais utilizadas. Possui diversas aplicações práticas dentro e fora da matemática, além de servir como base para a implementação de outras estruturas de dados.

2 Descrição da Análise Léxica

Para montagem do analisador léxico foi utilizada a ferramenta de geração de analisador léxico Flex [V.] [ALSU06]. Por meio dela, redigiu-se um arquivo com extensão .l que descreve como deve ser feita a geração do analisador. Nele foram adicionadas diversas capturas de caracteres por meio de estruturas *regex*. Para cada sequência capturada e reconhecida como uma regra léxica, é impresso na tela a linha e coluna correspondente [Cai], bem como quaisquer erros que divergem destas regras. Na última seção do arquivo, foi adicionada uma função `main` que abriga a lógica para abertura, análise e fechamento do arquivo.

No anexo, a tabela da seção C apresenta a descrição dos *tokens* e a gramática na seção B. Na seção A, consta a possível implementação da estrutura de *tokens* (1.1) e da estrutura dos elementos da tabela de símbolos (1.2), que será construída como uma lista encadeada.

3 Arquivos de Teste

Dentro do diretório `tests/`, foram criados dois arquivos de teste corretos lexicamente e dois arquivos de teste contendo erros léxicos, respectivamente :

1. `correct_test_01.c`;
2. `correct_test_02.c`, .

E dois arquivos de teste contendo erros léxicos:

1. `incorrect_test_01.c`:
   - Erro na linha 1, coluna 7; e e
   - Erro na linha 3, coluna 1.
2. `incorrect_test_02.c`:
   - Erro na linha 5, coluna 9; e
   - Erro na linha 9, coluna 11.

4 Compilação e Organização

O código referente a este projeto foi desenvolvido e executado em um sistema com as seguintes características:

- Sistema Operacional: Manjaro 21.10;
- Kernel: 5.10 LTS;
- gcc: 11.1.0;
- ld: 2.36.1;
- flex: 2.6.4;
- make: 4.3.

Para execução o analisador léxico, o comando

```
$ make tradutor
```

compilará o código e gerará um executável de nome 'tradutor'. Com isto, execute utilizando

```
$ ./tradutor <caminho_para_o_arquivo>
```

Como dito acima, os testes se encontram no diretório `./tests/`. Caso deseje executar diretamente um arquivo de teste, é possível executar uma das variações do seguinte comando:

```
$ make <correct1|correct2|incorrect1|incorrect2>
```

Referências

[ALSU06] A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2 edition, 2006.

[Cai] A. Cainikovs. StackOverflow - Resposta a: stdlib and colored output in C. https://stackoverflow.com/a/3219471/12140684. Acessado por último em 07 Ago 2021.

[Jon] D. Jones. The New C Standard. http://www.coding-guidelines.com/cbook/cbook1_1.pdf. Acessado por último em 18 Ago 2021.

[Pol] B. Pollack. BNF Grammar for C-Minus. http://www.csci-snc.com/ExamplesX/C-Syntax.pdf. Acessado por último em 10 Ago 2021.

[V.] Nachiappan V. USING LEX. https://silcnitc.github.io/lex.html. Acessado por último em 10 Ago 2021.

A Estruturas

```
typedef struct {
 char token_type[20];
 char content[100];
 int line;
 int column;
} Token;
```

Listing 1.1. Estrutura do Token

```
typedef struct {
 char identifier[35];
 int type;
 // 0 int, 1 float, 2 int list, 3 float list
 int is_function;
 // 0 is variable, 1 is function
 char params[127][35]
} Symbol;
```

Listing 1.2. Estrutura da Tabela de Símbolos [Jon]

B Gramática

A Gramática a seguir foi gerada utilizando como base a gramática C-Minus [Pol]. Os rótulos em letras todas maiúsculas são equivalentes aos apresentados na tabela C.

1. *program* → *declarationList*
2. *declarationList* → *declarationList declaration* | *declaration*
3. *declaration* → *variableDeclaration* | *functionDeclaration*
4. *variableDeclaration* → *typeSpecifier* **ID ;** | *typeSpecifier* **list ID ;**
5. *typeSpecifier* → **INT** | **FLOAT**
6. *functionDeclaration* → *typeSpecifier* **ID (** *params* **)** *compoundStmt* | *typeSpecifier* **list ID (** *params* **)** *compoundStmt*
7. *params* → *paramList* | ϵ
8. *paramList* → *paramList* **,** *param* | *param*
9. *param* → *typeSpecifier* **ID** | *typeSpecifier* **list ID**
10. *compoundStmt* → **{** *localDeclarations statementList* **}**
11. *localDeclarations* → *localDeclarations variableDeclaration* | ϵ
12. *statmentList* → *statementList statement* | ϵ
13. *statement* → *expressionStmt* | *compoundStmt* | *conditionalStmt* | *loopStmt* | *returnStmt*
14. *expressionStmt* → *expression* **;** | **;**
15. *conditionalStmt* → **if (** *expression* **)** *statement* | **if (** *expression* **)** *statement* **else** *statment*
16. *loopStmt* → **for (** *expression* **;** *simpleExpression* **;** *expression* **)** *statement*

17. *returnStmt* → **return** *expression* **;**
18. *expression* → *variable* **=** *expression* | *simpleExpression*
19. *var* → **ID**
20. *simpleExpression* → *addExpression* **OP_RELAT** *addExpression* | *addExpression* | *unaryExpression* | *listExpression*
21. *addExpression* → *addExpression addOperator term* | *term*
22. *unaryExpression* → **?***factor* | **!***factor* | **%***factor*
23. *listExpression* → *factor* **CONSTRUCTOR** *factor* | *factor* **FUNC_LIST** *factor*
24. *term* → *term mulOperator factor* | *factor*
25. *addOperator* → **+** | **-**
26. *mulOperator* → ***** | **/**
27. *factor* → **(** *expression* **)** | *var* | *call* | **FLOAT** | **INT** | **NIL**
28. *call* → **ID** **(** *args* **)** | **INPUT** **(** *args* **)** | **OUTPUT** **(** *args* **)**
29. *args* → *argList* | ϵ
30. *argList* → *argList* **,** *expression* | *expression*

C Tokens e Lexemas

Rótulo do Token	Padrão do Lexema (RegEx)	Lexema de Exemplo
ID	[_a-zA-Z][_a-zA-Z0-9]*	num
DIGIT	[0-9]	88
FLOAT	(-)?{DIGIT}*.{DIGIT}+	-402.3
INT	(-)?{DIGIT}+	25
OP_ARITH	[+*/-]	+
OP_LOGIC	(&&)|(||)	&&
OP_RELAT	(<)|(<=)|(>)|(>=)|(==)|(!=)	>=
OP_ASSIG	(=)	=
OP_EXCL	(!)	!
OP_LIST	(?)	?
DESTRUCTOR	(%)	%
FUNC_LIST	(>>)|(<<)	>>
CONSTRUCTOR	(:)	:
TYPE	(int)|(float)|(list)	list
NIL	(NIL)	NIL
IF	(if)	if
ELSE	(else)	else
FOR	(for)	for
RETURN	(return)	return
INPUT	(read)	read
OUTPUT	(write)|(writeln)	writeln
SEMICOLON	(;)	;
COMMA	(,)	,
CURLYB	[{}]	{
PARENTHESIS	[()]	(
STRING	(".*")|('.*')	"string"

Tabela 1. Tokens e lexemas de exemplo