Analisador Léxico

Bruno Cordeiro Mendes

Universidade de Brasília
150007094@aluno.unb.br

Abstract. Neste trabalho é apresentada a especificação do analisador léxico para o subconjunto da linguagem C com suporte a operações entre conjuntos. O analisador léxico será parte do compilador construído ao longo da disciplina de Tradutores ministrada pela professora Cláudia Nalon.

Keywords: Tradutores · Analisador Léxico · Compilador

1 Motivação

Este trabalho tem como principal motivador exercer o conhecimento adquirido na disciplina de Tradudores através da construção de um analisador léxico para uma linguagem fictícia fornecida na descrição do trabalho. O analisador léxico foi construído utilizando como base os conhecimentos adquiridos na leitura do livro-texto [1]. A linguagem em questão utiliza um nova primitiva de dados para conjuntos e foi projetada para facilitar as operações entre os mesmos em programas escritos na linguagem C. Em A está descrita a gramática da linguagem deste trabalho.

2 Descrição da Análise Léxica

O analisador trabalha de forma a identificar os tokens dentro do programa utilizando REGEX. O código flex criado para gerar o analisador teve como referência o tutorial fornecido por [2]. A saída para os tokens obtidos ficou da seguinte forma:

- *KEYWORD: k*, onde k é uma palavra reservada da linguagem
- *ID: i*, onde i é um identificador para nomes de variáveis e nomes de funções
- *INTEGER: i*, onde i é um número inteiro
- *FLOAT: f*, onde f é um número flutuante.
- *STRING: s*, onde s é uma string que se encontra dentro de aspas duplas.
- *CHAR: c*, onde c é um caractere que se encontra dentro de aspas simples.
- *PLUS*, para representar operação aritmética de soma
- *MINUS*, para representar operação aritmética de subtração
- *TIMES*, para representar operação aritmética de multiplicação
- *DIVIDE*, para representar operação aritmética de divisão

- *LESS*, para representar operação relacional de 'menor'
- *LESS_EQUAL*, para representar operação relacional de 'menor igual'
- *GREATER*, para representar operação relacional de 'maior'
- *GREATER_EQUAL*, para representar operação relacional de 'maior igual'
- *LESS_EQUAL*, para representar operação relacional de 'menor igual'
- *CONJUNCTION*, para representar operação lógica de conjunção
- *DISJUNCTION*, para representar operação lógica de disjunção
- *NEGATION*, para representar operação lógica de negação

Além disso foram acrescentados *EQUALS* para comparar se dois valores são iguais; *DIFFERENT* para comparar se dois valores são diferentes; *COMMA* para vírgula; *SEMICOLON* para ponto e vírgula; *OPEN_PAREN* para abertura de parênteses; *CLOSE_PAREN* para fechamento de parênteses; também foi adicionado *OPEN_CURL_BRACKETS* para abertura de chaves; assim como *CLOSE_CURL_BRACKETS* para fechamento de chaves; *ATRIBUICAO* para atribuições; e *NEWLINE* para reconhecimento de quebra de linha. Ao encontrar um caracter não válido o programa informa o erro juntamente com o número da linha seguido da posição (coluna).

3 Descrição dos arquivos de teste

O analisador vem acompanhado de quatro arquivos de teste, sendo eles *ex1* e *ex2* com código correto e *ex3* e *ex4* com código contendo erros. O erro do *ex3* se encontra na linha 2 coluna 4. E o erros do arquivo *ex4* se encontram na linha 11 na primeira coluna e na linha 17 na coluna 3.

4 Instruções para compilação e execução do programa.

O programa se encontra dentro do arquivo **lab1.l**. Para gerar o arquivo **lex.yy.c** execute:

```
flex lab1.l
```

Em seguida compile o arquivo **lex.yy.c** com o comando:

```
gcc lex.yy.c
```

Após compilar o arquivo, execute o programa **a.out**, passando o arquivo de exemplo, através do comando:

```
./a.out < ex1
```

Com isso será gerado os tokens identificados dentro do arquivo de exemplo passado para o programa.

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., USA (2006)
2. Levine, J., John, L.: Flex & Bison. O'Reilly Media, Inc., 1st edn. (2009)

A Gramática da Linguagem

⟨*program*⟩ ::= ⟨*decl_list*⟩

⟨*decl_list*⟩ ::= ⟨*decl_list*⟩ ⟨*decl*⟩ | ⟨*decl*⟩

⟨*decl*⟩ ::= ⟨*var-decl*⟩ | ⟨*function*⟩

⟨*var_decl*⟩ ::= ⟨*type*⟩ ⟨*identifier*⟩ **;**

⟨*function*⟩ ::= ⟨*type*⟩ ⟨*identifier*⟩ (⟨*arg_list*⟩?) ⟨*block*⟩

⟨*arg_list*⟩ ::= ⟨*arg_list*⟩ **,** ⟨*type*⟩ ⟨*identifier*⟩ | ϵ

⟨*type*⟩ ::= **int** | **float** | **elem** | **set**

⟨*block*⟩ ::= **{** ⟨*statement_list*⟩* **}**

⟨*statement_list*⟩ ::= ⟨*statement_list*⟩ ⟨*statement*⟩ | ϵ

⟨*statement*⟩ ::= ⟨*var_decl*⟩
| ⟨*statement_if*⟩
| ⟨*statement_add*⟩
| ⟨*statement_for*⟩
| ⟨*statement_forall*⟩
| ⟨*statement_assign*⟩
| ⟨*statement_return*⟩
| ⟨*statement_read*⟩
| ⟨*statement_write*⟩
| ⟨*statement_writeln*⟩

⟨*statement_if*⟩ ::= **if** (⟨*bool_exp*⟩) ⟨*block*⟩
| **if (** ⟨*bool_exp*⟩ **)** ⟨*statement*⟩ **else** ⟨*statement*⟩

⟨*statement_add*⟩ ::= **add**(⟨*pertinence_exp*⟩)

⟨*statement_remove*⟩ ::= **remove**(⟨*pertinence_exp*⟩)

⟨*statement_for*⟩ ::= **for (**⟨*statement_assign*⟩?**;** ⟨*bool_exp*⟩**;** ⟨*statement_assign*⟩? **)** ⟨*block*⟩

⟨*statement_forall*⟩ ::= **forall** (⟨*pertinence_exp*⟩) ⟨*block*⟩

⟨*statement_assign*⟩ ::= ⟨*identifier*⟩ = ⟨*value*⟩ ; | ⟨*identifier*⟩ = ⟨*identifier*⟩ ;

⟨*statement_return*⟩ ::= **return** ⟨*expr*⟩ ; | **return** ⟨*bool_exp*⟩ **;**

⟨*statement_read*⟩ ::= **read(**⟨*identifier*⟩**)**

⟨*statement_write*⟩ ::= **write(**⟨*exp*⟩**)**

⟨*statement_writeln*⟩ ::= **writeln(**⟨*exp*⟩**)**

⟨*exp*⟩ ::= ⟨*exp*⟩ **+** ⟨*term*⟩ | ⟨*exp*⟩ **-** ⟨*term*⟩ | ⟨*term*⟩

⟨*term*⟩ ::= ⟨*term*⟩ ***** ⟨*factor*⟩ | ⟨*term*⟩ **/** ⟨*factor*⟩ | ⟨*factor*⟩

⟨*bool_exp*⟩ ::= ⟨*factor*⟩ ⟨*logical_op*⟩ ⟨*factor*⟩ | ⟨*factor*⟩ | ⟨*is_set*⟩

⟨*is_set*⟩ ::= **is_set**(⟨*factor*⟩)

⟨*pertinence_exp*⟩ ::= ⟨*factor*⟩ **in** ⟨*factor*⟩

⟨*factor*⟩ ::= ⟨*identifier*⟩ | ⟨*value*⟩

⟨*identifier*⟩ ::= [A-z_][A-z_0-9]*

⟨*float*⟩ ::= -?[0-9]+.[0-9]+

⟨*int*⟩ ::= -?[0-9]+

⟨*string*⟩ ::= `.*`

⟨*char*⟩ ::= '.'|'\\n'|'\\r'|'\\t'

⟨*value*⟩ ::= ⟨*int*⟩ | ⟨*float*⟩ | ⟨*string*⟩ | ⟨*char*⟩ | **EMPTY**