Implementação do Analisador Léxico

Gabriel Fonseca*

Fevereiro 2021

Resumo

Este relatório descreve a implementação de um analisador léxico parte de projeto maior de um tradutor. O tradutor está sendo construído para um subconjunto da linguagem C adicionado de conjuntos bem como suas operações usais. Estão descritos nesse documento a estrátegia de implementação desse analisador léxico bem como exemplos para teste.

Palavras-chave: tradutores. analisador léxico. flex.

1 Introdução

No escopo de um tradutor, o analisador léxico é responsável por receber como entrada uma cadeia de caractéres e agrupa-las em uma sequência de lexemas. É função também do analisador léxico interagir com a tabela de símbolos, encontrando lexemas do tipo Indentificador e inserindo-os na tabela. Cada lexema é mapeado para um valor correspondente o qual é utilizado para gerar-se tokens. Sendo esses últimos a saída de um analisador léxico e também a entrada para o próxima etapa de tradução: Análise Sintática.

O analisador descrito neste relatório é limitado à receber a entrada, imprimir em tela os tokens válidos obtidos e listar todas as cadeias de caractéres que não puderam ser agrupadas em lexemas. Além disso, será descrita a gramática contruída para o projeto maior do tradutor bem como exemplos para execução do analisador sintático.

2 A linguagem à ser traduzida

A linguagem à ser traduzida trata-se de um subconjunto da linguagem C adicionado dos tipos *set* e *elem*. Os novos tipos emulam os conjuntos matémáticos e devem implementar operações de pertinência, adição e remoção de elementos, seleção de elementos do conjunto e iteração dos mesmos.

*Departamento de Ciência da Computação, Universidade de Brasília, Brasília, DF, Brasil. nunesgrf@gmail.com

3 Analisador Léxico

O analisador léxico foi gerado com aúxilio da ferramenta Flex. Esta ferramenta gera código-fontes para o reconhecimento de padrões léxicos em uma cadeia de caractéres com base em regras fornecidas ao mesmo. As regras definidas para este analisador podem ser encontradas no arquivo `src/main.lex`.

Descritas todas as regras e expressões regulares, basta delegar ao analisador gerado pelo flex a leitura de caractere à caractere em busca de casa-los no maior padrão definido pelas regras. Isto é feito por meio da função `yylex()`, que deve ser invocada enquanto houver caracteres à serem lidos. No momento que um lexema é encontrado, então este é impresso em tela para ciência do usuário.

Além disso, foi implementado dentre as regras os comentários. Então basta utilizar os lexamas `/*` seguido de `*/` ou até mesmo `//` que tais caracteres do escopo serão ignorados pelo *analisador sintático* assim como são ignorados em tradutores para as mais usuais linguagens de programação da modernidade.

3.1 Tratamento de Erros

Uma vez que o analisador léxico não consegue casar padrão de parte da cadeia de entrada com nenhuma regra, o tratamento de erro é invocado. Uma vez que ainda não existe um erro, um objeto é criado para agrupar todos os caractéres relacionados à aquele erro. Após isso, este objeto é salvo em uma lista de erros.

Ao fim, esta lista de erros é iterada e cada objeto é passa por uma formatação para ser exibido em tela. O usuário então pode comparar a saída de erros com a cadeia de entrada.

3.2 Testes

Para testes, estão disponíveis quatro arquivos-exemplos. Dos quais, dois não devem disparar qualquer tratamento de erro dentro do analisador e dois que devem disparar alguns erros.

O arquivo `samples/correct_sample_1.sample` possui apenas declarações de variáveis e operações simples. As variáveis devem ser exibidas como identificadores e os demais lexemas devem ser exibidos conforme as regras definidas na construção do analisador.

O arquivo `samples/correct_sample_2.sample` possui declarações de variáveis dos tipos *set* e *elem*, parte gramática da linguagem. Bem como operações utilizando essas variáveis.

O arquivo `samples/incorrect_sample_1.sample` possui padrões que não podem ser construídos pela gramática linguagem. Todos os padrões devem ser tratados e exibidos em tela.

O arquivo `sample/incorrect_sample_2.sample` possui padrões sintáticos incorretos que não devem ser tratados como erros pela análise léxica e também possui padrões léxicos incorretos que devem sim ser reportados para o usuário.

4 Conclusão

A construção do analisador léxico é simplificada pela ferramenta Flex, o trabalho fica limitado à definir as regras e da-las como entrada para a ferramenta. Dado isso, toda a parte de identificação dos lexemas fica abstraída.

Adicionalmente à isso, o tratamento de erros também teve que ser implementado, mas com auxílio dos contructos do Flex, muito foi abstraído. Dessa forma, o resultado final é um analisador léxico simplificado que pode ser usado para gerar a entrada para o próximo passo da tradução.

5 Gramática da linguagem

⟨*program*⟩ ::= ⟨*declaration-list*⟩

⟨*declaration-list*⟩ ::= ⟨*declaration-list*⟩ ⟨*declaration*⟩ | ⟨*declaration*⟩

⟨*statement*⟩ ::= ⟨*scope*⟩
| ⟨*var-declaration*⟩
| ⟨*assignment*⟩
| ⟨*print*⟩
| ⟨*scan*⟩
| ⟨*expression*⟩ ‘`;`’
| ⟨*condition*⟩
| ⟨*iteration*⟩
| ⟨*return*⟩

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

⟨*var-declaration*⟩ ::= ⟨*type*⟩ ⟨*identifier*⟩ ‘`;`’
| ⟨*type*⟩ ⟨*identifier*⟩ ‘`[`’ ⟨*integer*⟩ ‘`]`’ ‘`;`’
| ⟨*type*⟩ ⟨*identifier*⟩ ‘`{`’ ‘`}`’ ‘`;`’

⟨*function-declaration*⟩ ::= ⟨*type*⟩ ⟨*identifier*⟩ ‘`(`’ ⟨*parameters*⟩ ‘`)`’ ⟨*scope*⟩

⟨*parameters*⟩ ::= ⟨*parameters*⟩ ‘`,`’ ⟨*parameter*⟩ | ⟨*parameter*⟩ | ε

⟨*parameter*⟩ ::= ⟨*type*⟩ ⟨*identifier*⟩

⟨*scope*⟩ ::= ‘`{`’ ⟨*statement-list*⟩ ‘`}`’

⟨*statement-list*⟩ ::= ⟨*statement-list*⟩ ⟨*statement*⟩ | ε

⟨*print*⟩ ::= ⟨*print-word*⟩ ⟨*expression*⟩ ‘`;`’

⟨*scan*⟩ ::= ⟨*scan-word*⟩ ⟨*identifier*⟩ ‘`;`’

⟨*print-word*⟩ ::= ‘`print`’

⟨*scan-word*⟩ ::= ‘`scan`’

⟨*condition*⟩ ::= ‘`if`’ ‘`(`’ ⟨*expression*⟩ ‘`)`’ ⟨*statement*⟩ ⟨*condition-mid*⟩

⟨*condition-mid*⟩ ::= ‘`else if ‘(`’ ⟨*expression*⟩ ‘`)`’ ⟨*statement*⟩ ⟨*condition-mid*⟩
| ⟨*condition-end*⟩

⟨*condition-end*⟩ ::= ‘`else`’ ⟨*statement*⟩ | ε

⟨*iteration*⟩ ::= ‘`while`’ ‘`(`’ ⟨*expression*⟩ ‘`)`’ ⟨*statement*⟩
| ‘`for`’ ‘`(`’ ⟨*expression*⟩? ‘`;`’ ⟨*expression*⟩? ‘`;`’ ⟨*expression*⟩? ‘`)`’ ⟨*statement*⟩

⟨*return*⟩ ::= ‘`return`’ ⟨*expression*⟩? ‘`;`’

⟨*assignment*⟩ ::= ⟨*identifier*⟩ ⟨*assignment-op*⟩ ⟨*expression*⟩ '`;`'

⟨*expression*⟩ ::= ⟨*and-expression*⟩

⟨*and-expression*⟩ ::= ⟨*or-expression*⟩
| ⟨*and-expression*⟩ '`&&`' ⟨*or-expression*⟩

⟨*or-expression*⟩ ::= ⟨*bw-or-expression*⟩
| ⟨*or-expression*⟩ '`||`'

⟨*eq-expression*⟩ ::= ⟨*relational-expression*⟩
| ⟨*eq-expression*⟩ '`==`' ⟨*rel-expression*⟩
| ⟨*eq-expression*⟩ '`!=`' ⟨*rel-expression*⟩

⟨*rel-expression*⟩ ::= ⟨*rel-expression*⟩ ⟨*rel-op*⟩

⟨*add-expression*⟩ ::= ⟨*mult-expression*⟩
| ⟨*add-expression*⟩ '`+`' ⟨*mult-expression*⟩
| ⟨*add-expression*⟩ '`-`' ⟨*mult-expression*⟩

⟨*mult-expression*⟩ ::= ⟨*cast-expression*⟩
| ⟨*mult-expression*⟩ ⟨*mul-op*⟩ ⟨*cast-expression*⟩

⟨*cast-expression*⟩ ::= ⟨*unary-expression*⟩
| '`(`' ⟨*type*⟩ '`)`' ⟨*cast-expression*⟩

⟨*unary-expression*⟩ ::= ⟨*postfix-expression*⟩
| ⟨*unary-op*⟩ ⟨*cast-expression*⟩
| sizeof ⟨*cast-expression*⟩

⟨*postfix-expression*⟩ ::= ⟨*primary-expression*⟩
| ⟨*postfix-expression*⟩ [⟨*expression*⟩]
| ⟨*postfix-expression*⟩ (⟨*param-values*⟩)

⟨*param-values*⟩ ::= ⟨*param-values*⟩ '`,`' ⟨*expression*⟩ | ⟨*expression*⟩ | ε

⟨*primary-expression*⟩ ::= ⟨*identifier*⟩
| ⟨*constant*⟩
| ⟨*string*⟩
| '`(`' ⟨*expression*⟩ '`)`'

⟨*constant*⟩ ::= ⟨*integer*⟩
| ''' ⟨*symbol*⟩ '''
| ⟨*integer*⟩ '`.`' ⟨*integer*⟩

⟨*integer*⟩ ::= ⟨*digit*⟩+

⟨*identifier*⟩ ::= ⟨*letter*⟩ { ⟨*letter*⟩ | ⟨*digit*⟩ | '`_`' }*

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

⟨*symbol*⟩ ::= Printables ASCII chars.

⟨*letter*⟩ ::= '`a`' | '`b`' | ... | '`z`' | '`A`' | ... | '`Z`'

⟨*digit*⟩ ::= '`0`' | '`1`' | '`2`' | ... | '`9`'

⟨*assignment-op*⟩ ::= '`=`' | '`+=`' | '`-=`' | '`*=`' | '`/=`'

⟨*unary-op*⟩ ::= '`+`' | '`-`' | '`!`'

⟨*rel-op*⟩ ::= '`<`' | '`>`' | '`<=`' | '`>=`' | '`in`'

⟨*mul-op*⟩ ::= '`*`' | '`/`'