Analisador léxico

Vinícius T M Sugimoto

CiC-UnB

Resumo Este documento apresenta a construção de um analisador léxico para um subconjunto da linguagem C feito com a ferramenta `flex`.

Keywords: Analisador sintático · Flex · C · Tradutores

1 Motivação

Este projeto foi desenvolvido com o objetivo de construir uma linguagem derivada da linguagem **C** para trabalhar com conjutos. Assim, os tipos primitivos `set` e `elem` foram adicionados à linguagem.

2 Análise Léxica

A análise léxica é a primeira etapa para a construção de um tradutor para a linguagem proposta. Com a análise léxica, um programa escrito na linguagem proposta é convertido em uma sequência de *tokens* que podem ser usados nas próximas etapas da construção do traduto, as análises sintática e semântica. Esta análise foi feita usando a ferramenta `flex`, que define um conjunto de "regras" que casam expressões regulares com o programa de entrada e executa comandos específicos para cada regra.

O `flex` foir utilizado para gerar *tokens* de maneira apropriada para cada regra e tentar identificar erros no código na etapa de análise léxica.

3 Testes

Junto aos arquivos deste projeto, que podem ser obtidos seguindo os passos da seção 4 estão quatro arquivos nomeados `subset_sumX.conj`, em que `X` $= 1, 2, 3, 4$ que server como arquivos de entrada teste para o analisador léxico gerado. Os arquivos com final $1, 2$ são arquivos corretos, já os arquivos com final $3, 4$ representam arquivos com erros.

4 Reprodução

4.1 Dependências

Este projeto tem as seguintes dependências:

- **gcc** versão 10.2.0
- **flex** versão 2.6.4

4.2 Execução

Os arquivos deste projeto podem ser encontrados no endereço https://github.com/vinicius-toshiyuki/conjunto.git. Os arquivos podem ser baixados em formato `.zip` pelo site ou o repositório pode ser clonado com o programa `git` no terminal. A seguir serão mostrados os passos a serem seguidos para executar o analisador léxico gerado em um terminal Linux.

```
# Clona o repositório
git clone \
https://github.com/vinicius-toshiyuki/conjunto.git
# Muda para o diretório criado
cd conjunto
# Gera o código C do analisador léxico
flex sintaxe.l
# Compila o analisador léxico
gcc lex.yy.c -lfl
# Para executar use o comando cat
cat subset_sum.conj | ./a.out
```

Referências

UoCa. Berkeley University of California. flex_1. Disponível em http://web.mit.edu/gnu/doc/html/flex_1.html. Acessado em 2021.

UoCb. Berkeley University of California. Lexical analysis with flex, for flex 2.6.3. Disponível em https://www.cs.virginia.edu/~cr4bd/flex-manual/index.html#SEC_Contents. Acessado em 2021.

5 Anexo

A sintaxe da linguagem é composta de expressões e comandos. A seguir são apresentados os padrões reconhecidos pela gramática desta linguagem. Nos padrões a seguir os caracteres [ ] são usados para impôr uma precedência maior na expressão regular interna, [[ ]] são usados para indicar grupos ou intervalos de caracteres em que qualquer um dos caracteres pertencentes pode ser casado, $\star$ é usado como a "*Kleene star*", + é usado para indicar que a expressão regular precedente é obrigatória e pode ser repetida, ? é usado para indicar que a expressão regular precedida é opcional, $\langle\ \rangle$ são usados para indicar um grupo de padrões. Vale ressaltar que espaços em branco não estão inclusos nos padrões descritos a seguir, para fins de simplicidade, mas considere-os válidos onde seriam válidos na linguagem `C`.

5.1 Padrões auxiliares

Os padrões a seguir são utilizados nos outros grupos de padrões e cada um tem sua própria denominação.

1. $-^{?}[[\mathbf{0-9}]]^{+}$
   Um número inteiro. É denominado pelo grupo $\langle\mathbf{INT}\rangle$.
2. $-^{?}[[\mathbf{0-9}]]^{+}.[[\mathbf{0-9}]]^{\star}\mathbf{f}^{?}$ ou $-^{?}[[\mathbf{0-9}]]^{\star}.[[\mathbf{0-9}]]^{+}\mathbf{f}^{?}$ ou $\langle\mathbf{INTEGER}\rangle\mathbf{f}$
   Um número real. É denominado pelo grupo $\langle\mathbf{FLOAT}\rangle$.
3. $\langle\mathbf{INT}\rangle$ ou $\langle\mathbf{FLOAT}\rangle$ ou **EMPTY**
   Uma constante. É denominada pelo grupo $\langle\mathbf{CONST}\rangle$.
4. $[[\mathbf{a-zA-Z\_}]][[\mathbf{a-zA-Z0-9\_}]]^{\star}$
   Um identificador. É denominado pelo grupo $\langle\mathbf{ID}\rangle$.
5. **int** ou **float** ou **set** ou **elem**
   Um tipo de dados. É denomidado pelo grupo $\langle\mathbf{TYPE}\rangle$.
6. $*$ ou / ou % ou + ou $-$
   Um operador aritimético. É denomidado pelo grupo $\langle\mathbf{OPARIT}\rangle$.
7. $<$ ou $>$ ou $<=$ ou $>=$ ou $==$ ou $!=$
   Um operador relacional. É denominado pelo grupo $\langle\mathbf{OPREL}\rangle$.
8. && ou || ou & ou |
   Um operador booleano. É denominado pelo grupo $\langle\mathbf{OPBOOL}\rangle$.
9. $\langle\mathbf{OPARIT}\rangle =$ ou $\& =$ ou $| =$
   Um operador composto. É denominado pelo grupo $\langle\mathbf{OPCOMP}\rangle$.
10. $\langle\mathbf{OPARIT}\rangle$ ou $\langle\mathbf{OPCOMP}\rangle$ ou $\langle\mathbf{OPREL}\rangle$ ou $\langle\mathbf{OPBOOL}\rangle$ ou = ou **in**
    Um operador binário. É denominado pelo grupo $\langle\mathbf{OPBIN}\rangle$.
11. ! ou $-$
    Um operador unário. É denominado pelo grupo $\langle\mathbf{OPUNI}\rangle$.
12. ++ ou $--$
    Um operador unário especial. É denominado pelo grupo $\langle\mathbf{OPSPE}\rangle$.

5.2 Expressões

Os padrões a seguir fazem parte do grupo ⟨**EXP**⟩.

1. ⟨**CONST**⟩
   Uma expressão constante. Seu valor é a própria constante.
2. ⟨**ID**⟩
   Uma expressão identificador. Seu valor é o valor associado ao identificador.
3. ⟨**EXP**⟩⟨**OPBIN**⟩⟨**EXP**⟩
   Uma expressão com operação binária. Seu valor é o valor da operação sobre os valores das duas expressões internas.
4. ⟨**OPUNI**⟩⟨**EXP**⟩ ou ⟨**OPSPE**⟩⟨**EXP**⟩
   Uma expressão com operação unária. Seu valor é o valor da operação sobre o valor da expressão interna.
5. ⟨**EXP**⟩⟨**OPSPE**⟩
   Uma expressão com operação unária especial. Seu valor é o valor da expressão interna.
6. ⟨**ID**⟩([⟨**EXP**⟩[, ⟨**EXP**⟩]*]$^?$)
   Uma expressão chamada de função. Seu valor é o valor de retorno da função computada sobre seus argumentos.

5.3 Comandos

Os padrões a seguir fazem parte do grupo ⟨**CMD**⟩.

1. {⟨**CMD**⟩*}
   Um bloco é um comando. Um bloco é seguido de uma sequência opcional de comandos.
2. ⟨**EXP**⟩$^?$;
   Um comando vazio. Um comando vazio pode ser precedido de uma expressão.
3. **if**(⟨**EXP**⟩)⟨**CMD**⟩
   Um comando `if`. Um comando `if` é seguido de uma expressão em parênteses e de um comando.
4. **else**⟨**CMD**⟩
   Um comando `else`. Um comando `else` é seguido de um comando. Na etapa de análise léxica não há checagem se um comando `else` é precedido de um `if`.
5. **while**(⟨**EXP**⟩)⟨**CMD**⟩
   Um comando `while`. Um comando `while` é seguido de uma expressão em parênteses e de um comando.
6. **forall**(⟨**EXP**⟩)⟨**CMD**⟩
   Um comando `forall`. Um comando `forall` é seguido de uma expressão em parênteses e de um comando. Na etapa de análise léxica não há checagem se a expressão é uma expressão "`in`".
7. **for**(⟨**EXP**⟩$^?$; ⟨**EXP**⟩$^?$; ⟨**EXP**⟩$^?$)⟨**CMD**⟩
   Um comando `for`. Um comando `for` é seguido de três expressões opcionais separadas por ponto-e-vírgula em parênteses e de um comando.

8. ⟨**TYPE**⟩⟨**ID**⟩[, ⟨**ID**⟩]*;
   Uma comando de declaração de variável. Uma declaração é composta de um tipo seguido de um identificador seguido de uma lista de "vírgula e identificador" opcional e de um ponto-e-vírgula.
9. ⟨**TYPE**⟩⟨**ID**⟩([⟨**TYPE**⟩⟨**ID**⟩[, ⟨**TYPE**⟩⟨**ID**⟩]*]$^{?}$){⟨**CMD**⟩}
   Uma comando de declaração de função. Uma declaração é composta de um tipo seguido de um identificador seguido de uma lista opcional de parâmetros no formato "tipo e identificador" em parênteses e de um comando bloco.
10. **return**⟨**EXP**⟩
    Um comando de retorno. É composto da palavra reservada `return` seguido de uma expressão e um ponto-e-vírgula.