Analisador Léxico

Trabalho de Tradutores

Wladimir Gramacho - 15/0048718

Profª. Cláudia Nalon

Setembro de 2019

1 Objetivos

O curso de Tradutores ministrado na Universidade de Brasília tem como objetivo o estudo dos componentes de um tradutor assim como a implementação destes[1]. Sendo assim, o trabalho desta matéria consiste na construção de um tradutor para uma linguagem de programação simplificada e de propósito específico. O desenvolvimento do projeto se dará em 6 etapas diferentes, sendo elas: Escolha do Tema, Analisador Léxico, Analisador Sintático, Analisador Semântico, Gerador de Código Intermediário e Apresentação do Trabalho. Este relatório se trata da apresentação da segunda fase: Analisador Léxico.

2 Desenvolvimento

A construção do analisador léxico se dá em etapas. Primeiramente, as palavras-chave definidas na gramática inicial (apresentada na Escolha do Tema) são utilizadas para criar as primeiras expressões regulares. Após isso, foi feita a análise dos *tokens* auxiliares (pontuação, parênteses, chaves, etc). Por motivos de depuração e tratamento de erros, são expostas as sequências de *tokens* lidas, além de erros léxicos encontrados, discriminados a linha e a coluna do erro. Por fim, é construída uma tabela de símbolos com os identificadores (variáveis e funções) obtidos.

2.1 Funções adicionadas

Além das regras e expressões regulares, o analisador léxico precisou de mais funcionalidades para atingir os objetivos dessa entrega. Foi implementada uma estrutura em lista encadeada e operações de inserção e busca para a construção da tabela de símbolos. A escolha por essa implementação simplificada se deu pelo fato de que parte do funcionamento do analisador léxico será implementado pelo analisador sintático, ou seja, ainda haverá mudanças no algoritmo que são inerentes ao processo de desenvolvimento. Houve também uma função para a impressão da tabela de símbolos.

[1] https://matriculaweb.unb.br/graduacao/disciplina.aspx?cod=116459

Para tratamento de erros, foi criada uma função que é chamada ao encontrar um *token* inválido. Tal função imprime para o usuário o *token* encontrado e a linha e coluna da ocorrência no arquivo de entrada. Foram também necessárias sub-rotinas auxiliares para impressão colorida de texto e controle da linha e coluna lidas.

2.2 Dificuldades

Além das dificuldades inerentes ao desenvolvimento em uma plataforma que o autor não domina (Flex), um obstáculo foi entender como seria feita a transferência da sequência de *tokens* para o analisador sintático. De forma a simplificar a construção dessa etapa, decidiu-se fazer a implementação da tabela de símbolos em lista encadeada, visto que a concepção da tabela só terminará após o processo de análise sintática.

3 Funcionamento

Análise léxica é a primeira fase de um compilador, na qual um analisador léxico lê o fluxo de carácteres do código fonte e os agrupa em sequências de lexemas [1]. Um lexema é uma unidade básica de significado para uma linguagem [2]. Dessa forma, o analisador léxico utiliza uma gramática para identificar lexemas da linguagem. Caso identifique algo que não faça parte da linguagem, o analisador deve informar a localização da ocorrência.

3.1 Gramática

A gramática implementada no analisador léxico sofreu algumas mudanças em relação à apresentada na Escolha do Tema. As mudanças realizadas estão destacadas em azul.

1. *program* → *declaration-list*
2. *declaration-list* → *declaration-list declaration* | *declaration*
3. *declaration* → *var-declaration* | *fun-declaration* | *comment*
4. *comment* → *//.**
5. *var-declaration* → *type* **ID ;** | *type* **ID [NUM] ;**
6. *type* → **int** | **string** | **void**
7. *fun-declaration* → *type* **ID (** *params* **)** *compound-stmt*
8. *params* → *param-list* | **void**
9. *param-list* → *param-list* **,** *param* | *param*
10. *param* → *type* **ID** | *type* **ID [ ]**
11. *compound-stmt* → **{** *local-declarations stmt-list* **}**

12. *local-declarations* → *local-declarations var-declaration* | ε
13. *stmt-list* → *stmt-list stmt* | ε
14. *stmt* → *expression-stmt* | *conditional-stmt* | *iteration-stmt* | *return-stmt*
15. *expression-stmt* → *expression* **;**
16. *conditional-stmt* → **if** (*expression*) *compound-stmt* |
    if (*expression*) *compound-stmt* **else** *compound-stmt*
17. *iteration-stmt* → **while** (*expression*) *compound-stmt*
18. *return-stmt* → **return** *expression* **;** | **return ;**
19. *expression* → *var* = *expression* | *simple-expression*
20. *var* → **ID** | **ID** [*expression*]
21. *simple-expression* → *op-expression relop op-expression* | *op-expression*
22. *relop* → <= | < | > | >= | == | **!=**
23. *op-expression* → *op-expression addop term* | *term*
24. *addop* → + | **-**
25. *term* → *term mulop factor* | *factor*
26. *mulop* → ***** | /
27. *factor* → (*expression*) | *var* | *call* | **“** *string* **”** | **NUM**
28. *call* → **ID** (*args*)
29. *string* → *string* **STRING** | *string* **#{** *expression* **}** | ε
30. *args* → *arg-list* | ε
31. *arg-list* → *arg-list* **,** *expression* | *expression*

ID = *letter* (*letter*|*digit*)*

NUM = *digit digit**

STRING = (\\.|[^#{}\\])*

letter = *a* | ... | *z* | *A* | ... | *Z*

digit = 0 | ... | 9

whitespace = \n | | | \t

A primeira mudança foi o acréscimo de comentários de uma linha. Comentários são observações legíveis para humanos e ignoradas por máquinas que auxiliam o código do programa [3]. Para identificar tais comentários, procuramos o padrão "//" e ignoramos todos os carácteres subsequentes até encontrar uma troca de linha ($\backslash n$).

Outra mudança necessária foi a expressão regular para identificação de *strings*. Com essa expressão regular, fazem parte da *string* todos os carácteres dentro das aspas que não forem aspas nem "#{}".

3.2 Tratamento de Erros

Ao identificar lexemas que não pertencem à linguagem, o analisador implementado imprime o lexema lido e a localização (linha e coluna) da ocorrência, de forma que o programador possa situar o erro e corrigi-lo. Após isso, o analisador se recupera e continua a ler o fluxo de carácteres.

4 Arquivos de teste

Junto com o analisador léxico e o arquivo de diretivas para execução (*makefile*), os arquivos compactados enviados na submissão dessa etapa contêm arquivos de teste do analisador. Há dois arquivos de teste com entradas válidas e dois arquivos com carácteres inválidos. É necessário ler o README para entender as opções de compilação. Para executar o analisador léxico e testar todos os arquivos, basta rodar o comando *make flex* no terminal.

Referências

[1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. Compilers, principles, techniques. *Addison wesley*, 7(8):9, 1986.

[2] David Crystal. *The Cambridge encyclopedia of the English language*. Ernst Klett Sprachen, 2004.

[3] Grubb Penny et al. *Software maintenance: concepts and practice*. World Scientific, 2003.