Analisador Léxico
Trabalho Prático de Tradutores

Gabriel Crespo de Souza[1 [14/0139982]]

Universidade de Brasília, Distrito Federal, Brasil
Departamento de Ciência da Computação
cic@unb.br
https://cic.unb.br/

Resumo Esse projeto tem como principal objetivo a construção de um analisador léxico para um dado subconjunto da linguagem C, apresentando sua nova primitiva e suas principais características.

Keywords: Compilação · Símbolos · Léxico · Análise · Tokens.

1 Objetivos

A disciplina de Tradutores tem como finalidade fornecer o entendimento sobre o funcionamento dos diferentes tipos de tradutores, seus componentes, bem como as principais formas de implementá-los. Dado isso, o projeto principal da disciplina consiste na implementação de um compilador. Essa implementação se dará em etapas e esse documento apresentará o desenvolvimento da etapa inicial, a análise léxica.

1.1 Linguagem

A nível de simplificação, um subconjunto da linguagem C foi escolhido, juntamente com uma nova primitiva que irá implementa e permite a manipulação de conceitos relacionados à listas. Nativamente, esses recursos não fazem parte da linguagem C, portanto, a sua inclusão visa facilitar o uso dessa estrutura de dados e suas principais operações. Listas são estruturas muito importantes devido sua flexibilidade em relação ao espaço e eficiência em relação ao tempo.

2 Desenvolvimento

O desenvolvimento do Analisador Léxico também se dá em fases. Inicialmente, as palavras-chave escolhidas para compor a gramática servem como base para a construção das primeiras expressões regulares. Logo após, é feita a análise dos símbolos auxiliares. Por fim, são apresentadas as sequências de *tokens* geradas, os erros encontrados, bem como a linha e coluna que foram encontrados.

2.1 Funcionalidades Adicionadas

Somente a nível de reaproveitamento de código é que se fez necessária a implementação de três funções, uma responsável por imprimir os *tokens* formatados em tela e as outras responsáveis por imprimir possíveis erros léxicos encontrados. Duas variáveis inteiras também foram adicionadas no escopo de declarações a fim de controlar os números de linha e coluna que estão sendo analisadas. Por fim, condições para remoção de comentários e duas variáveis inteiras que controlam suas linhas e colunas. Regras e expressões regulares também foram criadas.

3 Funcionamento

O Analisador Léxico lê um conjunto de caracteres de um programa fonte e os agrupa em sequências signficativas de caracteres chamadas *lexemas* e para cada *lexema* ele produz como saída *tokens*[1]. O Analisador Léxico irá basear-se na gramática para detectar os *lexemas* da linguagem, depois apresentar os tokens formados em tela e a linha em que foram encontrados. E ao localizar os símbolos que não pertencem à linguagem, o analisador deverá relatar o problema e o indicar o seu local de ocorrência, sem que o problema encontrado interrompa o fluxo de execução do analisador. Qualquer tipo de comentário será desconsiderado.

3.1 Instruções de compilação e execução

O projeto foi estruturado em diretórios, de modo que os arquivos principais estivessem separados dos auxiliares. Na raiz do projeto, */14_0139982* há quatro pastas: */src*, */lib*, */doc* e */tests*, bem como os arquivos *makefile* e *README.md*. Em *src* está o código do Analisador Léxico, *lexer.l*. Em *tests* estão quatro arquivos de testes, *correct_1.c*, *correct_2.c*, *incorrect_1.c* e *incorrect_2.c*. Na pasta *doc* se encontra o ralatório do projeto. Enquanto a pasta *lib* se encontra vazia, no momento. Para que a compilação e a execução do Analisador Léxico se dêem com sucesso, será necessário seguir os seguintes passos estando no diretório raiz:

1. Executar o comando *make* e os seguintes arquivos serão gerados:
   - *lex.yy.c*;
   - *tradutor (arquivo executável)*;
2. Agora, basta chamar o código objeto juntamente com um único arquivo de teste presente em */tests* como parâmetro por exemplo:
   - *./tradutor tests/correct_1.c.*;

4 Teste

Dos quatro arquivos disponibilizados para teste, dois apresentam erros léxicos, pois contém símbolos que não pertencem à definição da linguagem. O arquivo *incorrect_1.c* contém o símbolo $ na linha 2 e coluna 14 e o símbolo # na linha 33 e colunas 19 e 20, além de haver um comentário aberto mas não fechado na linha 23 e coluna 4. Já o arquivo *incorrect_2.c* contém o símbolo @ nas linhas 19 e 24 e coluna 11 e o símbolo # na linha 43 e coluna 47. Os arquivos *correct_1.c* e *correct_2.c* não deverão apresentar erro léxico algum.

Referências

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

A Gramática

1. **program** → *declarations*
2. **declarations** → *declarations declaration* | *declaration*
3. **declaration** → *type identifier*
4. **type** → *int* | *float* | *list*
5. **identifier** → *variable* | *function*
6. **variable** → *type identifier_ list*
7. **function** → *type identifier parameters* | *type identifier*
8. **parameters** → *parameters_ list* | ϵ
9. **parameters_list** → *parameters_ list* | *param*
10. **param** → *type identifier*
11. **blocks** → *blocks block* | *block*
12. **block** → *expression* | *conditional* | *iteration* | *return*
13. **expression** → *expression operator expression* | *operand operator operand*
14. **condicional** → *if (expression) block* | *else (expression) block*
15. **iteration** → *for (expression; expression; expression;) block*
16. **return** → *return expression* | *return*
17. **operation** → *relational* | *logic* | *arithmetic* | *list* | *input-output*
18. **relational** → *<* |*>* | *<=* | *>=* |*==* |*!=*
19. **logic** → *//* |*!* | *&&*
20. **arithmetic** → *+* |*-* | *** | */*
21. **list** → *assignment* |*:* | *?* | *!* | *%* | *»* | *«*
22. **input-output** → *read* |*write* | *writeln*
23. **operand** → *digit*
24. **digit** → *0* | *...* | *9*
25. **alphabet** → *a* | *...* | *z* | *A* | *...* | *Z*
26. **alphanumeric** → *alphabet* | *digit*
27. **identifier** → *alphabet(alphanumeric)*
28. **special character** → *\s* | *\t* | *\n*