Analisador Léxico

Pedro Henrique Santos Gonzaga - 140030131

Departamento de Ciência da Computção, Universidade de Brasília (UnB) - Fev/2021

1 Motivação e Proposta

Este relatório tem como objetivo apresentar a primeira parte do projeto da disciplina de Tradutores do curso de ciência da computação da Universidade de Brasília. O projeto é dividido em 4 partes e tem como finalidade a construção de um compilador completo. Nesta primeira etapa será apresentado um analisador léxico funcional, assim como a gramática a ser utilizada pelo tradutor no decorrer do semestre. A linguagem foco do projeto é a linguagem C com uma nova primitiva, onde seja possível determinadas operações entre conjuntos. Com esta nova primitiva, o leque de possibilidades para a linguagem se expande, pois, em C puro, para se trabalhar com conjuntos, é necessária a declaração e manipulação de várias estruturas de dados, não havendo um padrão para tal. Com a definição formal desta nova primitiva, suas operações serão padronizadas e facilitadas.

2 Descrição da Análise Léxica

Para o desenvolvimento do analisador léxico, foram utilizadas expressões regulares para identificar os lexemas aceitos pela linguagem, assim como seus tokens auxiliares (pontuação, chaves, etc). As definições de todas as regras podem ser vistas no arquivo *skylang.l*. Quando o analisador é executado e lhe é dado um arquivo de entrada para análise, o mesmo inicia a leitura dos caracteres do arquivo seguindo as regras definidas pelas expressões regulares. Caso um token que não esteja previsto dentro das regras estipuladas seja encontrado, a função *LexicalError()* é chamada e uma mensagem de erro é mostrada ao usuário, explicitando a linha do token inválido.

3 Arquivos de Teste

Juntamento com o código foram disponibilizados dois arquivos de texto para teste. Os arquivos testeCorreto1.txt e testeCorreto2.txt possuem exemplos de códgio que não possuem erros para os lexemas definidos para o projeto. Os arquivos testeErro1.txt e testeErro2.txt apresentam códigos com erros léxicos que serão apontados ao se executar o programa (token ! no arquivo testeErro1.txt e token $ no arquivo testeErro2).

4 Compilação e execução

Para a compilação do analisador léxico, em ambiente Linux, abra o terminal, navegue até a pasta com os arquivos .l e .txt e digite os seguintes comandos:

1 - flex skylang.l

2 - gcc lex.yy.c

Para executar o programa e realizar a análise dos arquivos de teste, digite o seguinte comando, também no terminal:

1- ./a.out

Por último, quando a mensagem "Digite o nome do arquivo a ser analisado" aparecer, digite o nome do arquivo de teste que deverá ser analisado. É necessário digitar a extensão do arquivo também, como exemplificado abaixo:

1 - testeCorreto1.txt

References

1. Alfred, Aho., Monica, Lam., Ravi, Sethi.: Compiladores: Princípios, Técnicas e Ferramentas. 2nd edition. Pearson, 2008
2. Flex Manual, https://westes.github.io/flex/manual/. Last accessed 18 Feb 2021

A Gramática

Foi definida uma gramática inicial a ser utilizada durante o projeto, podendo ser alterada, ou expandida nas próximas etapas do desenvolvimento.

1. *Programa* → *declarationList*
2. *declarationList* → *declarationList declaration* | *declaration*
3. *declaration* → *variable declaration* c *function declaration*
4. *variable declaration* → **int** *ID* | **float** *ID* | **set** *ID* | **elem** *ID*
5. *function declaration* → **int** *ID (params) codeBlock* | **float** *ID (params) codeBlock* | **set** *ID (params) codeBlock* | **elem** *ID (params) codeBlock*
6. *params* → params-list
7. *params-list* → *params-list* | *param*
8. *param* → **int** *ID* | **float** *ID* | **set** *ID* | **elem** *ID*
9. *codeBlock* → *statements* returns
10. *returns* → **return** *exp*
11. *statements* → *statements statement*
12. *statement* → *IfStatement* | *WhileStatement* | *ForStatement* | *DeclarationStatement* | *ForallStatement* | *ExistsStatement* | *AddStatement* | *RemoveStatement* | *WritlnStatement* | *WriteStatement*
13. *DeclarationStatement* → *variable declaration*
14. *WhileStatement* → **while** (exp) codeBlock
15. *IfStatement* → **if** (exp) codeBlock | **if** (exp) codeBlock **else** codeBlock
16. *ForStatement* → **for** (exp) codeBlock
17. *ForallStatement* → **forall** (exp) codeBlock | **forall** (exp)
18. *ExistsStatement* → **exists** (exp) codeBlock | **exists** (exp)
19. *AddStatement* → **add** (exp) codeBlock | *AddStatement* (exp)
20. *RemoveStatement* → **remove** (exp) codeBlock | **remove** (exp)
21. *WriteStatement* → **write** (*"ID"*) codeBlock | **write** (*"ID"*)
22. *WritelnStatement* → **writeln** (*"ID"*) codeBlock | **writeln** (*"ID"*)
23. *exp* → *setExpressions* | *OpExpressions* | RelationalExpressions
24. *setExpressions* → *ID* **in** *ID* | *ID* **in** *ID codeBlock*
25. OpExpressions → *ID* | *ID* = exp | *ID***ID* | *ID* + *ID* | *ID* - *ID* | *ID*/*ID*
26. *RelationalExpressions* → *ID* rel *ID* | *ID* rel exp
27. *rel* → > | < | <= | >= | ! = | == | | &&

ID = letter(letter | digit)*
NUM = digit+
letter = [a-zA-Z]
digit = [0-9]

Palavras reservadas da linguagem: write, writeln, add, exists, for, forall, remove, in, empty, if, else, float, elem, set, return, read.