

Trabalho Prático 1 - Parte 1: Análise Léxica

Geovanna Menegasse Silva
Vitor Lucio dos Santos Ferreira

1. Introdução

Esse trabalho consiste no desenvolvimento de um analisador léxico para a linguagem TIGER, primeira etapa para o desenvolvimento de um compilador completo. Essa fase é responsável pelo agrupamento de sequências de caracteres em tokens seguindo uma série de regras definidas por expressões regulares. Os tokens gerados serão agrupados em fases gramaticais na próxima etapa do trabalho.

Nas próximas seções apresentaremos a descrição da solução encontrada e as instruções de utilização do analisador léxico criado. Além disso, apresentaremos alguns dos resultados obtidos pela submissão dos códigos fonte feitos na linguagem TIGER, disponibilizados para teste, ao nosso analisador léxico.

2. Descrição da solução

A solução para este trabalho prático foi, inicialmente, definir quais seriam os tokens a serem gerados, utilizando como base a documentação disponibilizada sobre a análise sintática da linguagem TIGER. Criamos um arquivo com extensão “.l” utilizando a linguagem Lex e C contendo a especificação do analisador léxico, definindo tanto as regras a serem seguidas para a criação de tokens distintos para cada símbolo quanto a forma como esses tokens seriam exibidos no terminal.

Para a criação de um executável para nosso analisador léxico que seja capaz de ler uma sequência de caracteres e gerar tokens, foi feita a transformação do “*scanner.l*” no arquivo “*lex.yy.c*” contendo a representação tabular do diagrama de transição construído a partir das expressões regulares. Ao final, criou-se o executável “*a.out*” que representa o scanner do analisador léxico, o qual recebe um código fonte TIGER como entrada e o transforma em uma sequência de tokens definidos pelas regras.

Para as transformações mencionadas e criação do scanner, utilizamos os comandos:

```
flex scanner.l  
cc lex.yy.c -lfl
```

Por fim, para facilitar a compilação e execução do programa, criamos um arquivo makefile que executa esses comandos.

3. Exemplo de teste

Para testarmos nosso analisador léxico, submetemos ao scanner os códigos disponibilizados para teste. Apresentamos aqui apenas o primeiro teste: código para cálculo de fatorial. Utilizando o comando `./a.out < tests/00.txt` submetemo-nos para análise.

```
/* define a recursive function */
let

/* calculate n! */
function nfactor(n: int): int =
  if n = 0
  then 1
  else n * nfactor(n-1)

in
  nfactor(10)
end
```

Entrada

Podemos ver abaixo que nosso scanner ignorou os comentários e identificou corretamente as palavras reservadas da linguagem, símbolos, números e identificadores, atribuindo-os corretamente aos seus diferentes tipos de tokens.

```
Token: 90 Lexema: let
Token: 94 Lexema: function
Token: 41 Lexema: nfactor
Token: 20 Lexema: (
Token: 41 Lexema: n
Token: 61 Lexema: :
Token: 41 Lexema: int
Token: 21 Lexema: )
Token: 61 Lexema: :
Token: 41 Lexema: int
Token: 52 Lexema: =
Token: 70 Lexema: if
Token: 41 Lexema: n
Token: 52 Lexema: =
Token: 40 Lexema: 0
Token: 71 Lexema: then
Token: 40 Lexema: 1
Token: 72 Lexema: else
Token: 41 Lexema: n
Token: 12 Lexema: *
Token: 41 Lexema: nfactor
Token: 20 Lexema: (
Token: 41 Lexema: n
Token: 11 Lexema: -
Token: 40 Lexema: 1
Token: 21 Lexema: )
Token: 91 Lexema: in
Token: 41 Lexema: nfactor
Token: 20 Lexema: (
Token: 40 Lexema: 10
Token: 21 Lexema: )
Token: 92 Lexema: end
```

Saída

4. Suposições sobre as especificações

Fizemos algumas suposições sobre os itens destacados para a implementação e resultados deste trabalho prático:

- Apresentou os resultados do(s) programa(s) fonte(s) submetido(s) com os símbolos reconhecidos pelo Analisador Léxico? [S, N, Mais ou Menos] **(2 pontos)**
- Apresentou a listagem do código fonte submetido ao gerador de análise léxica? [S, N, Mais ou Menos] **(2 pontos)**

Para estes 2 itens, entendemos que os prints dos tokens seriam tanto o resultado esperado quanto a listagem do código fonte submetido ao Analisador Léxico.

- Apresentou o(s) programa(s) fonte(s) submetido(s) ao seu Analisador Léxico? [S, N, Mais ou Menos] **(1 ponto)**

Para este item, entendemos que precisávamos apresentar os programas fonte utilizados para teste no nosso Analisador Léxico e, portanto, colocamos estes códigos fontes na pasta *tests* localizada na raiz deste trabalho prático.

5. Instruções para utilização do analisador

1. Utilizar um sistema operacional Linux.
2. Instalar as ferramentas Flex, Bison e C:

```
sudo apt install flex
```

3. Ir para a pasta root do projeto.
4. Executar o seguinte comando para gerar o scanner:

```
make compile  
make run
```

5. Digitar o código fonte como entrada para ser analisado.
6. Outra alternativa seria colocar o código fonte em um arquivo texto e executar:

```
./a.out < tests/[nome_do_arquivo]
```

7. A saída será no formato:

```
"Token: <numero_do_token> Lexema: <valor_do_lexema>"
```



6. Ferramentas de apoio

Como ferramentas de apoio para o desenvolvimento do analisador léxico, utilizamos as tecnologias Flex e Bison para gerar o código de análise léxica.

7. Conclusão

Com este trabalho, concluímos que a criação de tokens para uma determinada linguagem depende muito do conhecimento sobre a sintaxe da mesma e de suas palavras reservadas, já que precisamos gerar tokens que possam ser usados na fase de análise sintática e que, sem essa orientação, não teríamos como saber quais tipos de tokens precisaríamos gerar. Também concluímos que expressões regulares podem ser extremamente úteis nessa fase do compilador para auxiliar nas transições entre os estados do autômato gerado para o reconhecimento de símbolos.

8. Referências

1.  Part 01: Tutorial on lex/yacc
2.  Part 02: Tutorial on lex/yacc.
3. Aho, Alfred V., Lam Monic S., Sethi, R.; Ullman and Jeffrey D., Compilers Principles, Techniques, & Tools , 2nd Edition, Pearson Addison Wesley, New York, 2007