TRABALHO PRÁTICO - ANÁLISE LÉXICA

Bruno Marcos Pinheiro da Silva
201565552AC

1. INTRODUÇÃO

Este documento visa descrever as principais características da implementação do Analisador Léxico para a linguagem "*lang*", proposta na disciplina de Teoria dos Compiladores.

O analisador léxico foi desenvolvido com o auxílio da ferramenta JFlex[1]. Para isto, as expressões regulares que descrevem cada um dos tokens da linguagem foram desenvolvidas e acopladas à ferramenta, que gera automaticamente o analisador. Além disso, estruturas extras foram desenvolvidas para interagir com o analisador, que serão discutidas nas seções seguintes.

2. ARQUIVOS

O trabalho consiste de 4 arquivos:

- Token.java
- Scanner.java
- TOKEN_TYPE.java
- lexicalDef.jflex

3. COMO EXECUTAR

Para usar o programa, basta executar as seguintes linhas no terminal de comando aberto na pasta do projeto:

- Se o .jar do JFlex estiver na pasta do projeto:

```
java -jar jflex-full-1.8.2.jar .\lexicalDef.jflex
```

- Se o JFlex estiver configurado no terminal:

```
jflex .\lexicalDef.jflex
```

- Após executar o jflex, compilar os arquivos java:

```
javac *.java
```

- Para executar o programa:

```
java Scanner "caminho para o arquivo de entrada"
```

Um arquivo `compile.bash` será incluído no projeto para executar esses comandos automaticamente, considerando a existência do JFlex na variável de caminho reconhecida pelo terminal.

[1] https://www.jflex.de/

4. DECISÕES DE PROJETO

4.1. Tokens

No total foram definidas 41 classes de tokens diferentes, de acordo com a definição léxica da linguagem. Estes tokens são expostos a seguir:

ID: token para qualquer tipo de identificador que não seja uma das palavras reservadas. Consiste em uma sequência de letras, dígitos e sobrescritos que começa com uma letra. Este ID também identifica *nomes de Tipo*, que começam com letra maiúscula, e sua diferenciação será feita nas etapas posteriores com o analisador sintático/semântico.

```
[A-Za-z]([A-Za-z]|[0-9]|"_")*
```

LITERAL_INT: uma sequência de um ou mais dígitos.

```
[0-9][0-9]*
```

LITERAL_CHAR: um único caractere entre aspas simples. Para sua identificação, considera-se todos os caracteres especiais (que precisam de estarem acompanhados de "\" para seu reconhecimento), assim como qualquer outro caractere que não seja um destes especiais (char vazio '' não é reconhecido).

```
\'(\\(r|n|t|b|\'|\"|\\)|[^\'\"\\])\'
```

LITERAL_BOOL: true ou false.

```
(true|false)
```

LITERAL_FLOAT: zero ou mais dígitos seguido de ponto e um ou mais dígitos.

```
[0-9]*\.[0-9]+
```

LITERAL_NULL: a expressão null

```
null
```

Os próximos tokens representam somente símbolos e palavras reservadas, e suas expressões regulares são somente os símbolos correspondentes a cada um.

EQ: =
EQEQ: ==
LESS: <
GREATER: >
NOTEQ: !=
PLUS: +
MINUS: -
MULT: *
DIV: /
MOD: %
AND: &&
NOT: !

OPEN_ROUND: `(`
CLOSE_ROUND: `)`
OPEN_SQUARE: `[`
CLOSE_SQUARE:`]`
OPEN_CURLY: `{`
CLOSE_CURLY: `}`
SEMI_COLON: `;`
COLON: `:`
COLONCOLON: `::`
COMMA: `,`
DOT: `.`
IF: `if`
ELSE: `else`
DATA: `data`
ITERATE: `iterate`
READ: `read`
PRINT: `read`
RETURN: `return`
NEW: `new`
TYPE_INT: `Int`
TYPE_CHAR: `Char`
TYPE_BOOL: `Bool`
TYPE_FLOAT: `Float`

Comentários de linha: são reconhecidos a partir do reconhecimento de "--" e continuam até a primeira quebra de linha. Para isso as seguintes expressões regulares foram usadas:
Início do comentário: `--`
Fim do comentário: `[\n\r]`
Conteúdo do comentário: `[^\n\r]+`

Comentários de múltiplas linhas: são reconhecidos a partir de "{-" e se extendem até o primeiro "-}." Neste caso, foi considerada qualquer número de "-" seguido de "}" a fim de evitar a parada prematura em casos específicos
Início do comentário: `{-`
Fim do comentário: `\-+\}`
Conteúdo do comentário: `([^\-]|\-+([^\-\}]))+`

Quebras de linha e espaços: são ignorados através da seguinte expressão regular:
`(\r|\n|\r\n)|[ \t\b]`

5. JFlex

A implementação utilizando JFlex é direta e requer poucos detalhes. Basicamente três estados foram definidos:

- YYINITIAL: estado inicial padrão do JFlex. Processa cada um dos tokens definidos na linguagem.
- LINE_COMMENT: estado de comentário de linha. Interrompe o processamento dos tokens até o fim do comentário.
- MULTILNE_COMMENT: estado de comentário de múltiplas linhas. Interrompe o processamento dos tokens até o fim do comentário.

6. ESTRUTURAS DE DADOS

Para a integração com o JFlex e devida impressão dos lexemas, foram definidas algumas estruturas extras, baseadas na implementação exemplificada em sala de aula.

6.1. Token

Classe utilizada para armazenar as informações de cada token identificado. Armazena a linha e coluna em que o token foi encontrado, a classe do Token, seu lexema e/ou sua representação como objeto (por exemplo, um int já é salvo como um número). Nesta classe uma função especial para a impressão dos tokens e lexemas foi implementada.

6.2. Scanner

Classe responsável por utilizar o analisador léxico criado pelo JFlex e imprimir cada um dos elementos identificados no arquivo de entrada.

6.3. TOKEN_TYPE

Uma enumeração das classes de tokens definidas para facilitar a implementação.