Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Disciplina: Construção de Compiladores – BCC328
Alunos: Mateus Vitor Pereira Lana
Matrícula: 15.1.4340

TRABALHO PRÁTICO I - DOCUMENTAÇÃO

O analisador léxico desenvolvido nesse trabalho, que foi feito com base em uma analisador já existente disponibilizado pelo professor, é aplicado a uma linguagem embasada na gramática "Grammar 4.1 Example language for interpretation" do livro "Introduction to Compiler Design" do Torben. A linguagem é composta basicamente por três sinais de pontuação, dois operadores, literais inteiros, sete palavras reservadas e identificadores. Além dessas características, foram adicionados a linguagem comentários de linha, de bloco e de blocos aninhados, bem como delimitadores de espaços e quebras de linha. Ao longo deste documento temos uma breve descrição de cada uma das categorias aqui mencionadas.

Essa documentação consta basicamente de alterações realizadas em três arquivos do analisador léxico proposto pelo Torben, são eles o **'lexer.jflex'**, o **'parser.cup'**, e o **'LexerTest.java'**. O arquivo de extensão *cup* define todos os tokens da linguagem. O arquivo de extensão *jflex* contém o código que gera o analisador léxico e define seu comportamento. E por último o arquivo de extensão *java* executa os testes para o analisador léxico implementado. É importante citar também o arquivo **'Driver.java'**, que contém a função *main* e é como se fosse o compilador propriamente dito. Além desses, existem outras implementações relevantes mas que não serão abordadas.

1.0 A Linguagem

Os comentários e delimitadores de espaços e quebras de linha são ignorados pelo analisador léxico no momento de análise de uma entrada. São levados em conta os seguintes delimitadores de tabulação e quebra de linha **\t**,**\f**,**\n** e **\r**. Os comentários de linha são indicados por dois hífens contíguos e os comentários de blocos são delimitados por uma barra e um hífen na abertura e um hífen e uma barra no término, veja os exemplos a seguir.

```
Exemplos: "-- quebra de linha\n",
          "-- \ttabulacao",
          "-- comentário de linha",
          "/- comentário de bloco -/".
```

Para permitir comentários de blocos aninhados o analisador léxico possui um contador que incrementa cada vez que há um delimitador de abertura de comentário (“/-”) e decrementa quando há um delimitador de término (“-/”), acusando erro em casos nos quais o valor do contador é diferente de zero. A seguir um pequeno exemplo de comentário com blocos aninhados.

Exemplo: “/- externo /- interno -/ externo-/”.

Os literais inteiros são representados pela seguinte expressão regular **[0-9]+**, que significa que eles são compostos por qualquer número pertencente ao conjunto do intervalo de 0 a 9 e pelo fecho positivo dos mesmos, ou seja, a concatenação de um ou mais elementos deste conjunto. Quando um literal inteiro é lido da entrada, o mesmo é classificado como um token ***LIT_INTEIRO***. A seguir, temos alguns exemplos.

Exemplos: “7”,“15”,“1256”,“182456”.

Os identificadores são representados pela seguinte expressão regular **[a-zA-Z][a-zA-Z0-9_]***, isso quer dizer que a estrutura de um identificador se inicia com uma letra qualquer concatenada em seguida com o fecho de Kleene de um conjunto de letras, números de zero a nove e o símbolo sublinhado(“_”). Ou seja, uma letra maiúscula ou minúscula seguida da concatenação de um ou mais elementos do conjunto mencionado. Os identificadores são rotulados pelo token ***ID*** quando são lidos de uma entrada. Segue alguns exemplos de identificadores.

Exemplos: “x”,“y”,“var1”,“Variavel_1”,“nomeVar2”,“Nome_Var_30”.

Os operadores existentes na linguagem são o mais e o igual(**“+”, “=”**). No momento em que estes sinais são detectados em uma entrada eles são categorizados como os seguintes tokens ***MAIS*** e ***IGUAL***, respectivamente. É possível ver com clareza nos exemplos abaixo.

Exemplos: “x**+**7”,“y**=**15”,“var1 **+** x **=** 10”.

Os sinais de pontuação presentes na linguagem são abre parênteses, fecha parênteses e a vírgula, ou seja **“(” , “)”** e **“,”**. Quando estes sinais são lidos em uma entrada classificamos os mesmos como os tokens ***PARENTESE_ESQ***, ***PARENTESE_DIR*** e ***VIRGULA*** respectivamente. Abaixo temos alguns exemplos de entrada utilizando os sinais.

Exemplos: “var1, var2”; “if **(**x=1**)**”; “if**(**x=y, y=z**)**”.

As palavras reservadas da linguagem são **bool**, **int**, **if**, **then**, **else**, **let** e **in**; que quando são identificadas em uma entrada são catalogadas como os tokens ***BOOLEANO, INTEIRO, IF, THEN, ELSE, LET*** e ***IN*** respectivamente. A seguir temos alguns pequenos exemplos de entrada utilizando as palavras reservadas.

```
Exemplos: "if (x=1) then x=x+1 else x=0"
          "int var1 = 10"
          "let var2 = x in y"
          "bool flag"
```

2.0 Testes

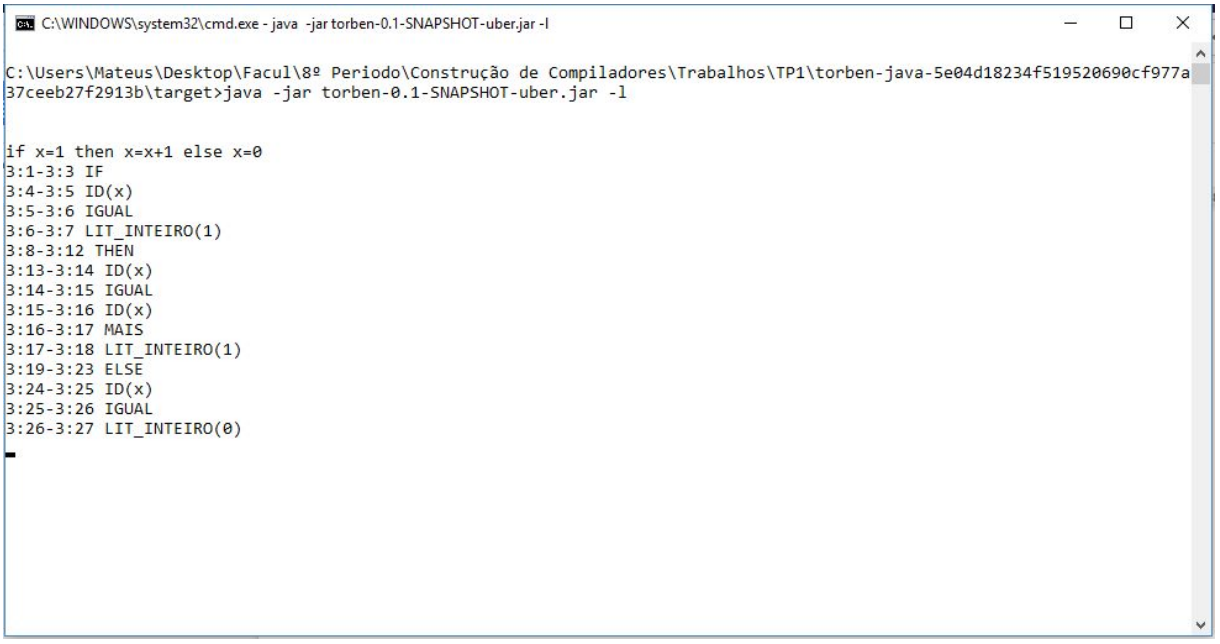
Foram realizados testes automatizados separados para cada uma das categorias descritas anteriormente, levando em conta execuções bem sucedidas e execuções nas quais são identificados erros. A seguir temos como exemplo o teste automatizado realizado para a categoria de comentários de blocos e blocos aninhados.

```
trun("/- a block comment -/", "1:22-1:22 EOF");
trun("/- a\nmultiline\ncomment -/", "3:11-3:11 EOF");
trun("/- begin ----/", "1:15-1:15 EOF");
trun("/- outer /- inner -/ outer -/", "1:30-1:30 EOF");
erun("/- a /- ab /- abc -/ ba", "1:24-1:24 lexical error:
unclosed comment");
```

O método ***trun*** recebe como parâmetros uma **string** de entrada e uma ou mais **strings** de saída(resultados esperados). Ao longo da função é testado se as saídas esperadas para a entrada coincidem com as saídas que foram recebidas como parâmetro.

O método ***erun*** recebe como parâmetros um **string** de entrada e um **string** de saída que representa uma mensagem de erro. Ao longo da função é testado se as mensagens de erro esperadas para a entrada coincidem com as mensagens de erro que foram recebidas como parâmetro.

A seguir temos um exemplo de execução que representa uma utilização de várias características da linguagem ao mesmo tempo.


C:\WINDOWS\system32\cmd.exe - java -jar torben-0.1-SNAPSHOT-uber.jar -l
C:\Users\Mateus\Desktop\Facul\8º Periodo\Construção de Compiladores\Trabalhos\TP1\torben-java-5e04d18234f519520690cf977a37ceeb27f2913b\target>java -jar torben-0.1-SNAPSHOT-uber.jar -l
if x=1 then x=x+1 else x=0
3:1-3:3 IF
3:4-3:5 ID(x)
3:5-3:6 IGUAL
3:6-3:7 LIT_INTEIRO(1)
3:8-3:12 THEN
3:13-3:14 ID(x)
3:14-3:15 IGUAL
3:15-3:16 ID(x)
3:16-3:17 MAIS
3:17-3:18 LIT_INTEIRO(1)
3:19-3:23 ELSE
3:24-3:25 ID(x)
3:25-3:26 IGUAL
3:26-3:27 LIT_INTEIRO(0)