



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 II - DOCUMENTAÇÃO

O analisador sintático descendente desenvolvido neste trabalho foi feito com base no analisador léxico desenvolvido para o primeiro trabalho prático. Ao longo deste documento temos uma breve descrição de cada uma das categorias aqui mencionadas.

1.0 A Linguagem

A linguagem que este analisador interpreta é 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.

A seguir apresentamos a gramática livre de contexto equivalente à gramática definida pelo Torben no livro após os processos de remoção de ambiguidade, eliminação de recursão a esquerda e fatoração de algumas regras.

1. $S \rightarrow \text{Program } \$$

2. $\text{Program} \rightarrow \text{Funs}$

programa

3. $\text{Funs} \rightarrow \text{Fun Funs}'$

4. $\text{Funs}' \rightarrow$

lista de funções

5. $\text{Funs}' \rightarrow \text{Funs}$

6. $\text{Fun} \rightarrow \text{TypeId } (\text{TypeIds}) = \text{Exp}$
funções

declaração de

7. $\text{TypeId} \rightarrow \text{int } \textit{id}$

tipo inteiro

8. $\text{TypeId} \rightarrow \text{bool } \textit{id}$

tipo booleano

9. $\text{TypeIds} \rightarrow \text{TypeId TypeIds}'$

10. $\text{TypeIds}' \rightarrow$

lista de parâmetros

11. $\text{TypeIds}' \rightarrow , \text{TypeIds}$

12. $\text{Exp} \rightarrow \text{let } \textit{id} = \text{Exp} \text{ in } \text{Exp}$

expressão de declaração

13. $\text{Exp} \rightarrow \text{if } \text{Exp} \text{ then } \text{Exp} \text{ else } \text{Exp}$

expressão condicional

14. $\text{Exp} \rightarrow A \text{ Exp}'$

15. $\text{Exp}' \rightarrow = A$

16. $\text{Exp}' \rightarrow$

17. $A \rightarrow T A'$

18. $A' \rightarrow + T A'$

19. $A' \rightarrow$

20. $T \rightarrow \textit{id } T'$

21. $T' \rightarrow (\text{Exps})$

22. $T' \rightarrow$

23. $T \rightarrow \textit{num}$

24. $\text{Exps} \rightarrow \text{Exp Exps}'$

25. $\text{Exps}' \rightarrow$

26. $\text{Exps}' \rightarrow , \text{Exps}$

Um programa para ser escrito na linguagem descrita deve ser composto por uma sequência de declarações de funções.

2.0 Aspectos Léxicos

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 hífen 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”

3.0 Testes Analisador Léxico

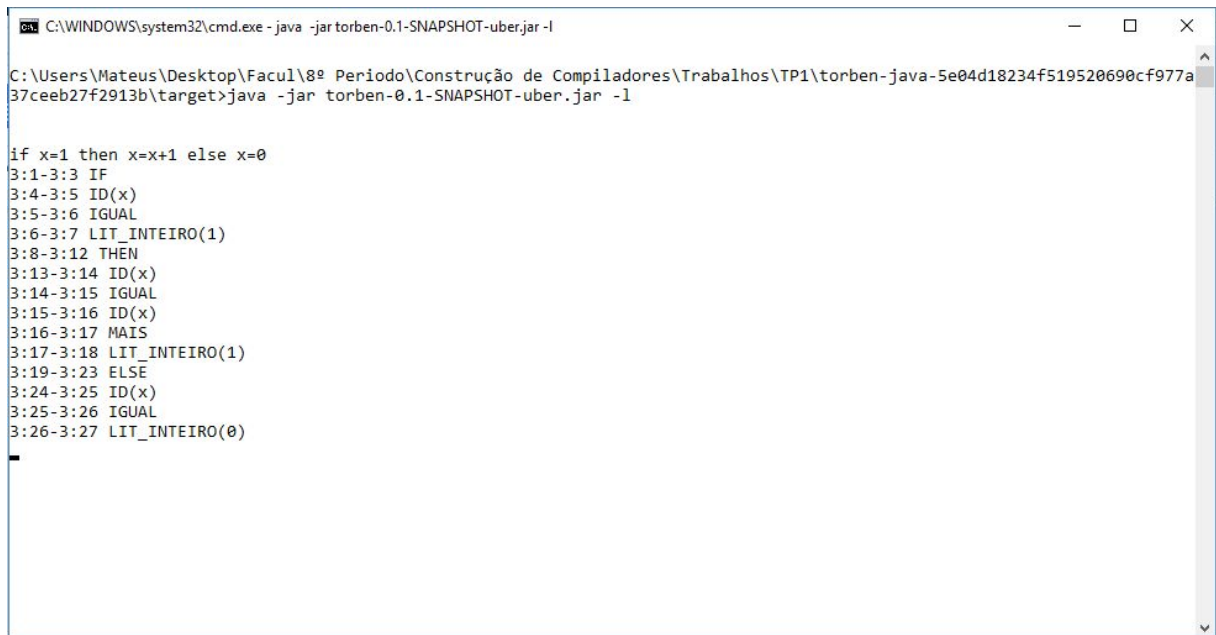
Foram realizados testes automatizados separados para cada uma das categorias descritas anteriormente, levando em conta execuções bem sucedidas e execuções nas quais são identificados erros. A seguir temos como exemplo o teste automatizado realizado para a categoria de comentários de blocos e blocos aninhados.

```
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 Período\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)
```

4.0 Analisador sintático descendente

Para a construção do analisador sintático descendente sabemos que é necessário, após o processamento da gramática, calcularmos as tabelas NULLABLE, FIRST, FOLLOW e em seguida a tabela LL(1). Sendo assim, a seguir temos as tabelas.

4.0.1 Tabelas *nullable*, *first* e *follow*

NT	Nullable	First	Follow
S	F	int bool	
Program	F	int bool	\$
Funs	F	int bool	\$
Funs'	V	int bool	\$
Fun	F	int bool	int bool \$
Typeld	F	int bool	(,)
Typelds	F	int bool)
Typelds'	V	,)
Exp	F	let if id num	in then else , int bool \$)
Exp'	V	=	in then else , int bool \$)
A	F	id num	= in then else , int bool \$)
A'	V	+	= in then else , int bool \$)
T	F	id num	+ = in then else , int bool \$)
T'	V	(+ = in then else , int bool \$)
Exps	F	let if id num)
Exps'	V	,)

4.0.2 Tabela *LL(1)*

NT	let	id	num	if	int	boo l	in	the n	else	()	+	=	,	\$
S					1	1									
Program					2	2									
Funs					3	3									
Funs'					5	5									4
Fun					6	6									
Typeld					7	8									
Typelds					9	9									
Typelds'											10			11	
Exp	12	14	14	13											
Exp'					16	16	16	16	16		16		15	16	16
A		17	17												
A'					19	19	19	19	19		19	18	19	19	19
T		20	23												
T'					22	22	22	22	22	21	22	22	22	22	22
Exps	24	24	24	24											
Exps'											25			26	