Universidade Estadual de Londrina

DC Departamento de Computação

5COP093 - Lista de Exercícios 11

Considere a gramática a seguir, bem como sua tabela de análise sintática LL(1):

$S \to E\$$
$E \to TE'$
$E' \to +TE'$
$E' \to$
$T \to FT'$
$T' \to *FT'$
$T' \to$
$F \to (E)$
$F \to id$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to$	$E' \to$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to$	$T' \to *FT'$		$T' \to$	$T' \to$
F	$F \to id$			$F \to (E)$		
S	$S \to E\$$			$S \to E\$$		

Implemente um analisador sintático LL(1) descendente recursivo para a gramática apresentada com base em sua respectiva tabela de análise. O símbolo terminal **id** que aparece na gramática corresponde a um identificador. Para a análise léxica, considere que identificadores válidos obedecem a seguinte expressão regular:

`[a-z][a-z0-9]`*

As cadeias de entrada do analisador sintático estarão armazenadas em um arquivo, sendo que haverá uma cadeia por linha. Espaços em branco e quebras de linha devem ser removidas pelo analizador léxico sem acusar erro. *Tokens* considerados inválidos devem ser indicados como erro pelo analisador léxico e a análise sintática da cadeia em questão é finalizada, continuando então na próxima cadeia do arquivo.

O arquivo contendo as cadeias deve ser lido da entrada padrão e a saída do programa deve ser impressa na saída padrão. Supondo que o arquivo de entrada se chame `cadeias.txt` e que o programa executável se chame `compilador`, a execução será da seguinte forma:

```
./compilador < cadeias.txt
```

Para cada cadeia aceita, o programa deve imprimir a mensagem:

```
CADEIA ACEITA
```

Se a cadeia contiver um erro léxico, o programa deve imprimir uma mensagem em que o *token* que causou o erro seja impresso. Se por exemplo o caractere `@`, que não corresponde a nenhum *token*, estiver presente em uma cadeia, a mensagem a ser impressa é:

```
ERRO LEXICO: @
```

Quando alguma cadeia apresentar um erro sintático, o programa deve imprimir uma mensagem em que mostra o *token* que causou o erro bem como a lista de *tokens* esperados. Suponha que um *token* do tipo `id` tenha causado um erro sintático e o programa esperava pelos *tokens* `+` ou `*`; a mensagem a ser impressa é:

```
ERRO SINTATICO EM: id ESPERADO: +, *
```

Observe que quando o erro for causado por um *token* do tipo `id`, a mensagem de erro não irá imprimir o identificador, mas sim o tipo do *token*. Desta forma se um identificador chamado `nomequalquer` causar um erro sintático, na mensagem de erro irá aparecer `id`. Ao apresentar os *tokens* esperados, utilize a ordem apresenta na tabela de análise LL(1).

Como exemplo de entrada completa, considere que um arquivo chamado `cadeias.txt` contém 6 linhas, indicando então que serão 6 cadeias a serem analisadas, as quais estão apresentadas a seguir:

```
chuchu + abobrinha * abacate666 $
variavel * @ + # identificador_999 $
programa * abc ($
(i*j+k*l)
abc def $
id$
```

A saída gerada pelo programa deve ser:

```
CADEIA ACEITA
ERRO LEXICO: @
ERRO SINTATICO EM: ( ESPERADO: +, *, ), $
ERRO SINTATICO EM: ESPERADO: +, *, ), $
ERRO SINTATICO EM: id ESPERADO: +, *, ), $
CADEIA ACEITA
```

Observação: Na última linha impressa como saída do analisador sintático LL(1), não deve haver quebra de linha extra, isto é, a saída do programa deve possuir a mesma quantidade de linhas que recebeu como entrada. Uma linha extra, mesmo que vazia, irá implicar que o programa gerou uma saída incorreta. Uma linha vazia no arquivo de entrada, implica que o programa irá receber uma cadeia vazia para analisar.

Considere agora a seguinte entrada composta por 3 linhas:

```
id $ @

id $ )
```

A saída gerada esperada é:

```
ERRO LEXICO: @
ERRO SINTATICO EM: ESPERADO: id, (
CADEIA ACEITA
```

Observe que uma linha que contiver um erro léxico não sofre análise sintática, mesmo que a cadeia anterior ao erro léxico esteja correta, como ocorre na linha 1. Observe também que na linha 3 não existe erro léxico e que todos os *tokens* que aparecem depois de uma cadeia válida são ignorados sem apresentar erro.