


1COP020 - Lista de Exercícios 09

1.  **Exercício Prático:** Considere a gramática a seguir:

$S \rightarrow \text{if } E \text{ then } S \text{ else } S$
 $S \rightarrow \text{begin } S \text{ } L$
 $S \rightarrow \text{print } E$
 $L \rightarrow \text{end}$
 $L \rightarrow ; S \text{ } L$
 $E \rightarrow \text{num} = \text{num}$

Abaixo existe um pseudocódigo que implementa um analisador sintático descendente recursivo para a gramática acima:

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;
```

```
int token = getToken();  
void advance() {token=getToken();}  
void eat(int t) {if (token==t) advance(); else error();}
```

```
void S(){  
    switch(token) {  
        case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;  
        case BEGIN: eat(BEGIN); S(); L(); break;  
        case PRINT: eat(PRINT); E(); break;  
        default: error(); }  
}  
void L(){  
    switch(token) {  
        case END: eat(END); break;  
        case SEMI: eat(SEMI); S(); L(); break;  
        default: error(); }  
}  
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

Implemente o pseudocódigo apresentado em C ou C++, de modo a torná-lo funcional. Observe que será necessário implementar um analisador léxico para os símbolos terminais da gramática. A ferramenta **Flex** ****não**** pode ser utilizada neste exercício. Você deve implementar manualmente o analisador léxico.

As palavras das cadeias de entrada estão associadas da seguinte forma entre o analisador léxico, a gramática e o pseudocódigo:

Analizador Léxico (ERs)	Gramática	Pseudocódigo
if	if	IF
then	then	THEN
else	else	ELSE
begin	begin	BEGIN
end	end	END
print	print	PRINT
;	;	SEMI
[+-]?[0-9]+	num	NUM
=	=	EQ

Observe que a coluna **Analizador Léxico (ERs)** corresponde as expressões regulares que descrevem os *tokens*. A coluna **Gramática** apresenta a forma como os *tokens* são classificados, ou seja o tipo de *token*. A coluna **Pseudocódigo** apresenta a forma como o *token* é identificado dentro do pseudocódigo.

Um exemplo de cadeia de entrada que deve ser reconhecida pela sua implementação é mostrada a seguir:

```
print +666 = 666
```

Na gramática, tal entrada corresponde à cadeia: `print num = num`

Se a cadeia de entrada for aceita, o seu programa deve imprimir a mensagem:

CADEIA ACEITA

Considere agora que o seu programa recebeu a seguinte cadeia de entrada:

```
print = -666 +666
```

Observe que a cadeia não é gerada pela gramática. Desta forma ela deve ser recusada pelo seu programa, o qual deve apresentar a seguinte mensagem:

ERRO SINTATICO EM: = ESPERADO: num

Quando ocorrer um erro sintático na cadeia de entrada, o seu programa deve informar o *token* que causou o erro bem como o *token* esperado. O *token* esperado é determinado através do parâmetro que a função `eat` recebe no pseudocódigo. Nas funções associadas a cada não-terminal, quando for chamada a função de erro, devem-se informar os *tokens* que seriam esperados pela função na seguinte ordem:

```
void S() → if, begin, print
void L() → end, ;
void E() → num
```

Como exemplo, considere a seguinte cadeia de entrada:

```
= -666 +666
```

O seu programa deve gerar a seguinte saída para essa cadeia:

```
ERRO SINTATICO EM: = ESPERADO: if, begin, print
```

Uma outra situação que o seu programa deve acusar é quando um erro sintático ocorre em uma cadeia que em princípio possui todos os *tokens* na posição correta, mas que termina de forma abrupta. Como exemplo, considere a seguinte cadeia de entrada:

```
print 666
```

O seu programa deve gerar a seguinte saída para essa cadeia:

```
ERRO SINTATICO: CADEIA INCOMPLETA
```

De modo a testar o seu programa, utilize as entradas e saídas fornecidas. Utilize o programa `diff` para comparar a sua saída com a saída esperada. Sua saída só pode ser considerada correta quando estiver idêntica à saída esperada.