

Grammar for recursive descent compiler

$E \rightarrow T \{+T\}^* \mid T \{-T\}^*$
 $T \rightarrow P \{ \times P \}^* \mid P \{ / P \}^*$
 $P \rightarrow a \mid b \mid c \mid d \mid e \mid (E)$

Procedure *match(X)* checks that the next token is X and then reads next input.

```
E() procedure {  
    T();  
    while tok == {+ | -}  
        {match(tok); T()}; }  
T() procedure {  
    P();  
    while tok == {× | /}  
        {match(tok); P()}; }  
P() procedure {  
    if tok == {a|b|c|d|e}  
        then match(tok)  
    else { match('(');  
          E();  
          match(''); } }
```

The code on the left is the pseudo code for a recursive descent parser that recognizes arithmetic expressions.

Rewrite the 3 procedures in either Java or C and add statements to output both the input string and the Postfix for the given input string. That is,

For input: a+b+c*d,
the output should be: a+b+c*d ab+cd*+
For input a*b*(c+d)+e,
the output should be: a*b*(c+d)+e ab*cd+*e+

TURN IN on March 1 a listing of your program and sample output showing that your program works as specified.

If this takes more than one hour to get the basic program written and one hour to add the output statements to output Postfix, you need to go back and review recursive descent. If you understand the lectures on LL(1) and recursive descent, the total time to do this should be one hour or less.