Обратен полски (reversed Polish) или постфиксен (postfix) запис (notation)

При него в линеен списък са наредени операнди и операции, като операциите следват операндите си. Полският запис за израза **R = A-(B+C*(D+E))**, например, изглежда така: ABCDE+*+-. Макар да изглежда странно, този запис е изключително удобен за пресмятане на израза.

Достатъчно е да си представим, че имаме стек (LIFO) и процедираме така: срещнем ли променлива, слагаме стойността ѝ в стека, срещнем ли операция - изваждаме необходимия брой операнди от стека, извършваме действието и слагаме отново резултата в стека. Ако изразът е верен, в края на работата в стека ще остане точно едно число - стойността на израза. Както забелязваме, при изчислението не са нужни никакви проверки - алгоритъмът е линеен и елементарен за описване. Тази идеология на работа с изрази се оправдава и от съществуването на линеен алгоритъм за превръщане на обикновен (инфиксен) запис в обратен полски. При организиран празен стек Stack и празен изход P, преобразуването извършваме по следните правила:

1. Прочетен операнд веднага прехвърляме в изхода.
2. Операторите и отварящите скоби задължително минават през стека.
3. Срещнем ли затваряща скоба, изваждаме от стека и прехвърляме в изхода, докато извадим отваряща скоба (която, естествено, не записваме в изхода).
4. При срещането на друг символ ("+","*","(" и т.н.), изваждаме от стека и прехвърляме в изхода дотогава, докато не срещнем нещо с по-нисък приоритет. Едно изключение - никога не премахваме "(" от стека, освен при правило 3 (обработката на ")"). (За тази цел бихме могли да дадем на отварящата скоба най-нисък приоритет.) Като приключим с изваждането от стека, записваме в стека текущия оператор.
5. Най-после, като стигнем края на входа R, прехвърляме всичко, записано в стека, към изхода.

Ето алгоритъмът, приложен върху нашето R:

```
------------------------------------------------------------
        R             Правило    Stack             P
------------------------------------------------------------
 a+b*c+(d*e+f)*g        1                    a
 ^
 a+b*c+(d*e+f)*g        2        +           a
  ^
 a+b*c+(d*e+f)*g        1        +           ab
   ^
 a+b*c+(d*e+f)*g        4        +*          ab
    ^
 a+b*c+(d*e+f)*g        1        +*          abc
     ^
 a+b*c+(d*e+f)*g        4         +(новият!) abc*+
      ^
 a+b*c+(d*e+f)*g        4         +(         abc*+
       ^
 a+b*c+(d*e+f)*g        1         +(         abc*+d
        ^
 a+b*c+(d*e+f)*g        4         +(*        abc*+d
         ^
 a+b*c+(d*e+f)*g        1         +(*        abc*+de
          ^
 a+b*c+(d*e+f)*g        4         +(+        abc*+de*
           ^
 a+b*c+(d*e+f)*g        1         +(+        abc*+de*f
            ^
 a+b*c+(d*e+f)*g        3          +         abc*+de*f+
             ^
 a+b*c+(d*e+f)*g        4         +*         abc*+de*f+
              ^
 a+b*c+(d*e+f)*g        1         +*         abc*+de*f+g
               ^
 a+b*c+(d*e+f)*g        5                    abc*+de*f+g*+
                ^
------------------------------------------------------------
```