Metody programowania 2020/2021
Konwersje: ONP ⟺ INF

Opis

Napisz program w Javie, który będzie realizował następujące operacje:

1. Podczas analizy wejścia usuwa znaki, które nie mogą występować w zadanych wyrażeniach, takie jak spacje, przecinki itp. oraz sprawdza poprawność składniową wyrażeń.
   Przy czym można założyć, że po usunięciu zbędnych symboli wyrażenia wejściowe są poprawne w postaci **INF**- jeśli są akceptowane przez podany poniżej automat skończony, oraz są poprawne w postaci **ONP** - są jeśli są obliczalne (dają wynik).
2. Konwertuje wyrażenia arytmetyczne i instrukcje przypisania z notacji INF do ONP.
3. Konwertuje wyrażenia arytmetyczne i instrukcje przypisania z ONP do notacji INF, zawierającej minimalną liczbę nawiasów, gwarantującą taką kolejność obliczeń jak w wyrażeniu ONP.

Instrukcja przypisania ma postać: ***wyrażenie_arytm1 = wyrażenie_arytm2***.

Wyrażenia arytmetyczne mogą zawierać jedynie:

a. nawiasy: (,) - tylko w notacji INF
b. operandy: małe litery alfabetu angielskiego
c. operatory:

operator	priorytet	łączność	opis operatora
( )	najwyższy	lewostronna	nawiasy
!, ~		prawostronna	negacja , - unarny
^		prawostronna	potęgowania
*, /, %		lewostronna	multiplikatywny
+, -		lewostronna	addytywny
<, >		lewostronna	relacje < i >
?		lewostronna	relacja równości
&		lewostronna	koniunkcja
I		lewostronna	alternatywa
=	najniższy	prawostronna	przypisania

Poprawność wyrażeń arytmetycznych w postaci infiksowej (INF)

Badanie poprawności wyrażeń arytmetycznych po usunięciu zbędnych znaków składa się z dwóch kroków:

1. Sprawdzenie, czy wyrażenie jest akceptowane przez poniższy automat skończony, który jest szczególnym przypadkiem Maszyny Turinga (WdI):

   A=(Q, T, δ, q0, F), gdzie Q={q0, q1, q2,q3} – zbiór stanów

   T = { zm, op1, op2, (,) } – alfabet symboli, które mogą wystąpić w wyrażeniu, przy czym:

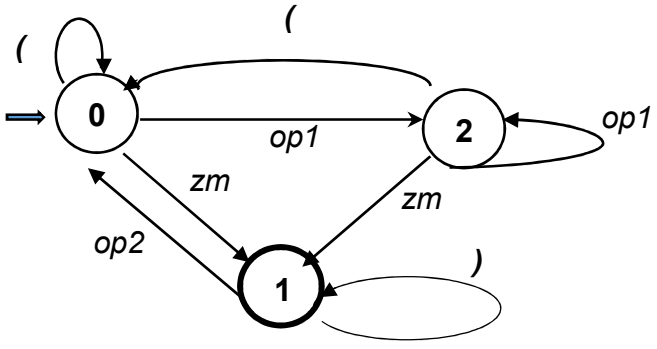
   zm - operand - zmienna (pojedyncza litera),
   op1 - operatory jednoargumentowe { ~ , !)
   op2 - operatory dwuargumentowe { ^, *, /, %, +, -, <, >, ?, &, |, =}
   ( ) - nawiasy

   δ - funkcja przejścia automatu, którą definiuje poniższy graf

   q0 - stan początkowy, F = {q1} – zbiór stanów końcowych

(a) W stanie '0' - automat rozpoczyna analizę wyrażenia w postaci INF od pierwszego symbolu wyrażenia.

(b) Jeśli automat po przeczytaniu wyrażenia znajdzie się w stanie q1 wówczas akceptuje wyrażenie, czyli stwierdza, że jest poprawne.



Przykłady błędnych wyrażeń: INF: a~+b , INF: a+b~ , INF: ()a+b INF: (a+b)+() , INF: ~()a

2. Sprawdzenie, czy nie występują w wyrażeniu następujące przypadki:
   a. niesparowane nawiasy lub ich zła kolejność, np.) (a (
   b. niezgodność liczby operatorów i operandów, np. a+b*

Ze względu na konieczną efektywność analizy, kroki 1 i 2 powinny być wykonywane jednocześnie (w jednej pętli)

Wejście

Dane do programu wczytywane są ze standardowego wejścia zgodnie z poniższą specyfikacją. Pierwsza linia wejścia zawiera liczbę całkowitą *z*, oznaczającą liczbę linii zawierających wyrażenia arytmetyczne lub instrukcje przypisania, których opisy występują kolejno po sobie.

Każda linia zawiera co najmniej 6 znaków i nie przekracza 256 znaków, może mieć jedną z dwóch postaci:

INF: wyrażenie arytmetyczne lub instrukcja przypisania, zapisane w notacji infiksowej,

ONP: wyrażenie arytmetyczne lub instrukcja przypisania, zapisane w notacji ONP

Przy czym wyrażenia i instrukcje przypisania mogą zawierać dowolne znaki. Program najpierw usuwa znaki niewystępujące w wyrażeniach arytmetycznych lub w instrukcjach przypisania, w tym spacje oraz sprawdza poprawność wyrażeń.

Wyjście

- Wyrażenie poprzedzone na wejściu napisem "**`INF: `**" musi być na wyjściu poprzedzone napisem "**`ONP: `**" i analogicznie wyrażenie poprzedzone na wejściu napisem "**`ONP: `**" musi być na wyjściu poprzedzone napisem "**`INF: `**". W przypadku błędnego wyrażenia, na wyjściu, zamiast skonwertowanego wyrażenia pojawi napis **`error`**.
- W przypadku konwersji wyrażenia w ONP do w INF, wyrażenie w INF musi zawierać minimalną liczbę nawiasów, gwarantującą podczas obliczania taką kolejność operacji (uwzględniając typ łączności i priorytety operatorów) jak w wyrażeniu ONP, np. **`ONP: xabc**=`** zostanie przekształcone do **`INF: x=a*(b*c)`**
- W przypadku wyrażeń w notacji **`INF`**, np. **`INF: ( a,+ b)/..[c3`** , program pozostawia jedynie: **`(a+b)/c`**, pozostałe znaki, w tym spacje – odrzuca, dodatkowo sprawdza poprawność wyrażenia, po czym dokonuje konwersji, wypisując na wyjściu: **`ONP: ab+c/.`**
- W przypadku wyrażeń w notacji **`ONP`**, np. **`ONP: ( a,b,.).c;-,*`** program pozostawia jedynie: **`abc-*,`** dodatkowo sprawdza, czy wyrażenie jest poprawne, po czym dokonuje konwersji, wypisując na wyjściu: **`INF: a*(b-c)`**.
- Wszystkie elementy wyrażeń na wyjściu są poprzedzone pojedynczą spacją.

Wymagania implementacyjne

1. Ogólnie jak w poprzednich programach, w szczególności jedynym możliwym importem jest import skanera wczytywania z klawiatury. Tym samym klasę stosu należy zaimplementować samodzielnie.
2. Na końcu kodu przesyłanego submitu proszę dopisać w formie komentarza dane własne wejściowe, zawierające po 10 przykładowych wyrażeń w INF i ONP i otrzymane wyniki.

Przykład danych

wejście:	wyjście:
10	
ONP: xabc**=	INF: x = a * (b * c)
ONP: ab+a~a-+	INF: a + b + (~ a - a)
INF: x=~~a+b*c	ONP: x a ~ ~ b c * + =
INF: t=~a<x<~b	ONP: t a ~ x < b ~ < =
INF: (a,+ b)/..[c3	ONP: a b + c /
ONP: (a,b,.).c;-,*	INF: a * (b - c)
ONP: abc++def++g+++	INF: error
INF: x=a=b=c^d^e	ONP: x a b c d e ^ ^ = = =
INF: (r+y)=a=(b+c)+d	ONP: r y + a b c + d + = =
INF: x=!(c>a & c<b)	ONP: x c a > c b < & ! =