

## Opis

Napisz program w Javie, który będzie realizował następujące operacje:

- Podczas analizy wyrażeń wejściowych program usuwa znaki, które nie mogą występować w zadanych wyrażeniach, takie jak spacje, przecinki lub nawiasy w wyrażeniu ONP oraz sprawdza poprawność składniową wyrażeń. Przy czym zakładamy, że po usunięciu zbędnych symboli wyrażenia wejściowe w postaci INF są poprawne, jeśli są akceptowane przez podany poniżej automat skończony, natomiast wyrażenia w postaci ONP są poprawne - jeśli są obliczalne.
- Konwertuje wyrażenia arytmetyczne i instrukcje przypisania z notacji INF do ONP.
- 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:

- nawiasy: ( , ) - tylko w notacji INF
- operandy: małe litery alfabetu angielskiego
- 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
		lewostronna	alternatywa
=	najniższy	prawostronna	przypisania

### Poprawność wyrażeń arytmetycznych w postaci infiksowej.

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:

$A=(Q, T, \delta, q_0, F)$ , gdzie  $Q=\{q_0, q_1, q_2, q_3\}$  – zbiór stanów

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

zm - operand - zmienna (pojedyncza litera),

op1 - operatory jednoargumentowe  $\{ \sim, ! \}$

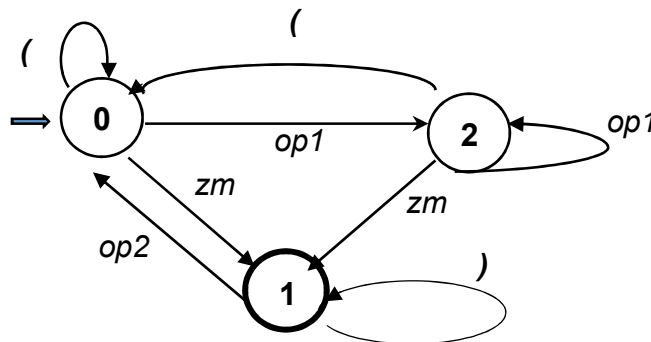
op2 - operatory dwuargumentowe  $\{ ^, *, /, \%, +, -, <, >, ?, \&, |, = \}$

$( )$  - nawiasy

$\delta$  - funkcja przejścia automatu, którą definiuje poniższy graf

$q_0$  - stan początkowy,  $F = \{q_1\}$  – zbiór stanów końcowych


- (a) W stanie  $q_0$  - automat rozpoczyna analizę wyrażenia w postaci INF od pierwszego symbolu wyrażenia.
- (b) Jeśli automat po przeczytaniu wyrażenia znajdzie się w stanie  $q_1$  wówczas akceptuje wyrażenie, czyli jest poprawne.



Przykłady błędnych wyrażeń: INF:  $a\sim+b$ , INF:  $a+b\sim$ , INF:  $()a+b$ , INF:  $(a+b)+()$ , INF:  $\sim()a$

2. Sprawdzenie, czy nie występują w wyrażeniach 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)

	<p style="text-align: center;">Metody programowania 2022/2023 Konwersje: ONP <math>\Leftrightarrow</math> INF</p>	<p style="text-align: center;">P_03</p>
---	---	---

## 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 wyrażen.

## 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 wyrażen 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 wyrażen 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 wyrażen 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 przykładowe wyrażenia w INF i ONP i otrzymane wyniki.
3. Przypominam o komentowaniu aplikacji w formie opisanej w punkcie 3 Regulaminu zaliczania programów na BaCy.

## 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 < & ! =