Zadanie 2. Kalkulator

Zadanie
Napisz program obliczający wartość wyrażenia z nawiasami, np.: dla „(2*(2+4))”, należy zwrócić 12.

Wejście
Program czyta ze standardowego wejścia dane zawierają prawidłowy ciąg znaków: cyfr, operatorów, nawiasów i spacji o maksymalnej długości 1 000 000 znaków. Dane wejściowe są prawidłowe, nie ma konieczności sprawdzania.

Wyjście
Program wypisuje na standardowe wyjście jedną liczbę, która jest wynikiem obliczenia wartości wyrażenia. Jeżeli podczas obliczania wyrażenia dojdzie do dzielenia przez zero, program powinien wypisać: `NaN`

Przykłady
1.
Wejście: `( 2 * ( 2 + 4 ) )`
Wyjście: `12`

2.
Wejście: `( ( ( 5 + 1 ) * ( 2 + 3 ) ) - ( 4 / 2 ) )`
Wyjście: `28`

3.
Wejście: `( ( 5 + 1 ) / ( 8 - ( 2 * 4 ) ) )`
Wyjście: `NaN`

Algorytm
Edsger Dijkstra opracował algorytm obliczania wyrażenia w oparciu o dwa stosy. Uproszczona wersja tego algorytmu jest następująca:

```
Przetwarzamy kolejne tokeny, aż do końca wyrażenia:
     jeżeli napotykamy liczbę, odkładamy ją na stos wartości
     jeżeli napotykamy operator, odkładamy go na stos operatorów
     jeżeli napotykamy nawias otwierający, nie robimy nic
     jeżeli napotykamy nawias zamykający, to:
          zdejmujemy ze stosu operator i dwie wartości
          obliczamy wartość wyrażenia
          odkładamy wynik na stos wartości
Na stosie znajduje się obliczona wartość wyrażenia
```

Przykład działania algorytmu:

krok	wartości	operatory	wyrażenie
1.	{}	{}	(2 * (2 + 4))
2.	{}	{}	2 * (2 + 4))
3.	{2}	{}	* (2 + 4))
4.	{2}	{*}	(2 + 4))
5.	{2}	{*}	2 + 4))
6.	{2, 2}	{*}	+ 4))
7.	{2, 2}	{*, +}	4))
8.	{2, 2, 4}	{*, +}	))
9.	{2, 6}	{*}	)
10.	{12}	{}	

Dodatkowe założenia

Poniższe założenia upraszczają implementację algorytmu:

- Każda operacja jest w nawiasie. Przykłady nieprawidłowych wejść:
  - `2 * 2 + 4`
  - `2 * ( 2 + 4 )`
- Liczby, nawiasy oraz operatory oddzielone są spacjami. Kolejne tokeny można zatem przeczytać przy pomocy instrukcji: `cin >> token;`. Przykłady nieprawidłowych wejść:
  - `( 2 * ( 2+4 ) )`
  - `( 2 * (2 + 4) )`
  - `( 2 *( 2 + 4 ) )`
- Liczby w wyrażeniu są całkowite, nieujemne i jednocyfrowe – czyli od 0 do 9
- Wspierane operacje: +, -, *, / (dzielenie całkowite, jak w języku C++)
- Wynik działania mieści się w zakresie 32-bitowej liczby całkowitej ze znakiem

Wskazówka

Wskazane jest użycie gotowej implementacji stosu, np. z biblioteki STL. Deklaracja: `stack<int> value_stack;`

Punktacja

Za ćwiczenie można otrzymać maksymalnie 20 punktów.

Mile widziane usprawnienie programu, np.: czytanie dowolnej liczby całkowitej (niekoniecznie jednocyfrowej), wsparcie dla liczb zmiennoprzecinkowych (ze zmiennoprzecinkowym dzieleniem). O potencjalnych rozszerzeniach, proszę informować w mailu z rozwiązaniem.

Termin

Termin oddania ćwiczenia: 5.12, godz. 23:59.

Oddanie ćwiczenia

Program należy wysłać za pomocą MS Teams – jeden plik zawierający kod programu. Plik powinien nazywać się `000000_calc_0.cpp`, gdzie `000000` to numer indeksu autora, `0` to numer kolejnej wersji programu. Nagłówek pliku powinien zawierać nazwę ćwiczenia i dane autora zgodnie z opisem w ogólnych zasadach.

Należy dobrze przetestować program przed wysłaniem!