Алгоритъмът [“Shunting-yard”](http://en.wikipedia.org/wiki/Shunting-yard_algorithm) се използва за подготовка на математически израз, във вид удобен за използване от алгоритъма [“Reversed Polish Notation”](http://en.wikipedia.org/wiki/Reverse_Polish_notation). Тоест ако въведем в конзолата математически израз под формата на текст “Shunting-yard” го преобразува, след което “Reversed Polish Notation” може да го използва и извежда резултат. Например **string** **input** = 3 + 4 \* 2 / ( 1 – 5 ) ^ 2 ^ 3, след като бъде обработен с “Shunting-yard” алгоритъма би трябвало да изглежда така: 3 4 2 \* 1 5 − 2 3 ^ ^ / + .

Изразът може да съдържа оператори (“+”, “-”, “\*”, “/”), функции (“^”, “ln”, “sqrt”, “sin”, “cos” и др) и реални числа (“1″, “2″, “3″, “-4″, “-10″, “10.465″, “-15,23″ и др).  
“Shunting-yard” може да прочита входа направо докато е във формата на стринг, или ако предварително сме отделили елементите му в списък. За да премината елементите на стринга, или така наречените **токени** ( от англ. “token” – “символ”) трябва да се инициализират предварително стек и опашка.

**Алгоритъмът “Shunting-yard” в отделни стъпки:**

Докато има токени за четене:

* Прочитаме токена
* Ако токенът е число го нареждаме в **опашката**.
* Ако токенът е функция го слагаме в **стека**.
* Ако токенът е символ за разделяне във функциите (например запетая):
  + Докато токенът на върха на стека не стане лява скоба, изваждаме операторите от стека и слагаме в опашката.  Ако не се достигне до лява скоба, то или запетаята или скобите не са правилно разположени
* Ако токенът е оператор, **o1**, то:
  + докато има друг оператор **o2**, на върха на стека, и приоритетът на **o1**е равен или по-малък от този на **o2**, изваждаме **o2** от стека и го слагаме в опашката.
  + слагаме **o1** върху стека.
* Ако токенът е лява скоба. то я слагаме в стека.
* Ако токенът е дясна скоба:
  + Докато токенът най-отгоре в стека не стане лява скоба, изваждаме оператори от стека и ги слагаме в опашката.
* Изваждаме лявата скоба от стека, но не я слагаме в опашката.
* Ако токенът най-отгоре в стека е функция, изваждаме го и го слагаме в опашката.
* Ако стека се изпразни, без да се достигне до лява скоба, то има неправилно разположени скоби.

Когато няма повече токени за четене:

* Докато все още има оператори в стека:
  + Ако токенът най-отгоре в стека е скоба, то има неправилно разположени скоби.
  + Изваждаме оператора от стека и го слагаме в опашката.

Изход

След изпълнението на алгоритъм разполагаме с готова опашка, която можем да използваме в “Reversed Polish” алгоритъма. За целта отново трябва да инициализираме стек.

**Алгоритъмът “Reversed Polish Notation” в отделни стъпки:**

Докато има токени в опашката

* Четем токена.
* Ако токенът е число
  + Слагаме го в стека.
* Ако токенът е оператор или функция.
  + Приема се, че функцията или оператора използва **n** на брой аргумента.
  + Ако има по-малко от **n**числа
    - **(Error)** Не са въведени достатъчен брой числа в израза.
  + В противен случай, измъкваме и взимаме стойността на горните**n** елемента от стека.
  + Извършваме операцията или функцията с горните стойности, като аргументи.
  + Ако има резултат го връщаме в стека.
* Ако има само една стойност в стека
  + Стойността е резултатът на израза.
* Ако има повече стойности в стека
  + **(Error)** Въведени са прекалено много стойности в израза.