**Обратная польская запись**

**Обра́тная по́льская нота́ция (ОПН)** — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как *обратная польская запись*, *обратная бесскобочная запись (ОБЗ)*, *постфиксная нотация*, *бесскобочная символика Лукасевича*, *польская инверсная запись*, *ПОЛИЗ*.

**Стековой машиной** называется алгоритм, проводящий вычисления по обратной польской записи.

Описание

Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

* Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.
* Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
* Результатом вычисления выражения становится результат последней вычисленной операции.

Например, рассмотрим вычисление выражения ***7 2 3 \* -*** (эквивалентное выражение в инфиксной нотации: *7-2\*3*).

1. Первый по порядку знак операции — «\*», поэтому первой выполняется операция умножения над операндами 2 и 3 (они стоят последними перед знаком). Выражение при этом преобразуется к виду 7 6 - (результат умножения — 6, — заменяет тройку «2 3 \*»).
2. Второй знак операции — «-». Выполняется операция вычитания над операндами 7 и 6.
3. Вычисление закончено. Результат последней операции равен 1, это и есть результат вычисления выражения.

Очевидное расширение обратной польской записи на унарные, тернарные и операции с любым другим количеством операндов: при использовании знаков таких операций в вычислении выражения операция применяется к соответствующему числу последних встретившихся операндов.

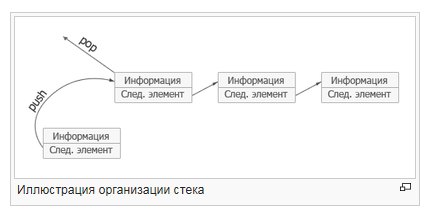
Особенности обратной польской записи следующие:

* Порядок выполнения операций однозначно задаётся порядком следования знаков операций в выражении, поэтому отпадает необходимость использования скобок и введения приоритетов и ассоциативности операций.
* В отличие от инфиксной записи, невозможно использовать одни и те же знаки для записи унарных и бинарных операций. Так, в инфиксной записи выражение *5 \* (-3 + 8)* использует знак «минус» как символ унарной операции (изменение знака числа), а выражение *(10 - 15) \* 3* применяет этот же знак для обозначения бинарной операции (вычитание). Конкретная операция определяется тем, в какой позиции находится знак. Обратная польская запись не позволяет этого: запись *5 3 - 8 + \** (условный аналог первого выражения) будет интерпретирована как ошибочная, поскольку невозможно определить, что «минус» после 5 и 3 обозначает не вычитание; в результате будет сделана попытка вычислить сначала *5 - 3*, затем *2 + 8*, после чего выяснится, что для операции умножения не хватает операндов. Чтобы всё же записать это выражение, придётся либо переформулировать его, либо ввести для операции изменения знака отдельное обозначение, например, «±»: *5 3 ± 8 + \*.*
* Так же, как и в инфиксной нотации, в ОПН одно и то же вычисление может быть записано в нескольких разных вариантах. Например, выражение *(10 - 15) \* 3* в ОПН можно записать как *10 15 - 3 \*,* а можно — как *3 10 15 - \**
* Из-за отсутствия скобок обратная польская запись короче инфиксной. За этот счёт при вычислениях на калькуляторах повышается скорость работы оператора (уменьшается количество нажимаемых клавиш), а в программируемых устройствах сокращается объём тех частей программы, которые описывают вычисления. Последнее может быть немаловажно для портативных и встроенных вычислительных устройств, имеющих жёсткие ограничения на объём памяти.

**Вычисление на стеке.**

**Стек** (англ. *stack* — стопка; читается *стэк*) – абстрактный тип данных, представляющий собой список элементов, организованных по принципу [*LIFO*](https://ru.wikipedia.org/wiki/LIFO) (англ. *last in — first out*, «последним пришёл — первым вышел»).

Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.



В JavaScript для более удобной работы с концом массива существуют специальные методы.

**pop**

Удаляет последний элемент из массива и возвращает его:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удалили "Груша"

alert( fruits ); // Яблоко, Апельсин

**push**

Добавляет элемент в конец массива:

var fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

Вызов fruits.push(...) равнозначен fruits[fruits.length] = ....

Общий порядок.

Автоматизация вычисления выражений в обратной польской нотации основана на использовании стека. Алгоритм вычисления для стековой машины элементарен:

1. Обработка входного символа
   * Если на вход подан операнд, он помещается на вершину стека.
   * Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.
2. Если входной набор символов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

Реализация стековой машины, как программная, так и аппаратная, чрезвычайно проста и может быть очень эффективной. Обратная польская запись совершенно унифицирована — она принципиально одинаково записывает унарные, бинарные, тернарные и любые другие операции, а также обращения к функциям, что позволяет не усложнять конструкцию вычислительных устройств при расширении набора поддерживаемых операций. Это и послужило причиной использования обратной польской записи в некоторых научных и программируемых микрокалькуляторах.

Пример вычисления выражений

Инфиксное выражение ( 1 + 2 ) × 4 + 3 в ОПН может быть записано так: *1 2 + 4 × 3 +*

Вычисление производится слева направо, ввод интерпретируется как указано в приведённой ниже таблице (указано состояние стека после выполнения операции, вершина стека выделена красным цветом):

|  |  |  |
| --- | --- | --- |
| **Ввод** | **Операция** | **Стек** |
| 1 | поместить в стек | 1 |
| 2 | поместить в стек | 1, 2 |
| + | сложение | 3 |
| 4 | поместить в стек | 3, 4 |
| \* | умножение | 12 |
| 3 | поместить в стек | 12, 3 |
| + | сложение | 15 |

Результат, 15, в конце вычислений находится на вершине стека.

Клавиша «Ввод» (обозначаемая на калькуляторах как «Enter» или символом «↑») выполняет роль разделителя операндов, когда два операнда непосредственно следуют друг за другом. Если за операндом следует знак операции, то её нажатие не требуется, это сокращает количество действий, которые нужно выполнить для получения результата.

Преобразование из инфиксной нотации.

Эдсгер Дейкстра изобрёл алгоритм для преобразования выражений из инфиксной нотации в ОПН. Алгоритм получил название «сортировочная станция», за сходство его операций с происходящим на железнодорожных сортировочных станциях. Инфиксная нотация — это форма математических записей, которую использует большинство людей (например, *3 + 4 или 3 + 4 \* (2 - 1)*). Как и алгоритм вычисления ОПН, алгоритм сортировочной станции основан на стеке. В преобразовании участвуют две текстовых переменных: входная и выходная строки. В процессе преобразования используется стек, хранящий ещё не добавленные к выходной строке операторы. Преобразующая программа читает входную строку последовательно символ за символом (символ — это не обязательно буква), выполняет на каждом шаге некоторые действия в зависимости от того, какой символ был прочитан.

Простой пример.

Вход: 3 + 4

Добавим 3 к выходной строке (если прочитано число, то оно сразу добавляется к выходной строке).

Помещаем + (или его Идентификатор) в стек операторов.

Добавим 4 к выходной строке.

Мы прочитали всё выражение, теперь выталкиваем все оставшиеся в стеке операторы в выходную строку. В нашем примере в стеке содержится только +.

Выходная строка: 3 4 +

В данном примере проявляются некоторые правила: все числа переносятся в выходную строку сразу после прочтения; когда выражение прочитано полностью, все оставшиеся в стеке операторы выталкиваются в выходную строку.

Алгоритм

* Пока есть ещё символы для чтения:
* Читаем очередной символ.
* Если символ является числом или постфиксной функцией (например, — факториал), добавляем его к выходной строке.
* Если символ является символом префиксной функцией (например, sin), помещаем его в стек.
* Если символ является открывающей скобкой, помещаем его в стек.
* Если символ является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется. Если стек закончился раньше, чем мы встретили открывающую скобку, это означает, что в выражении либо неверно поставлен разделитель, либо не согласованы скобки.

Если существуют разные виды скобок, появление непарной скобки также свидетельствует об ошибке. Если какие-то скобки одновременно являются функциями (например, [x] – целая часть), добавляем к выходной строке символ этой функции.

* Если символ является оператором ***о1***, тогда:

1) пока на вершине стека префиксная функция…

… ИЛИ операция на вершине стека приоритетнее ***o1***

… ИЛИ операция на вершине стека [левоассоциативная](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)" \o "Ассоциативность (программирование)) с приоритетом как у ***o1***

… выталкиваем верхний элемент стека в выходную строку;

2) помещаем оператор ***o1*** в стек.

* Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку. В стеке должны были остаться только символы операторов; если это не так, значит в выражении не согласованы скобки.

Сложный пример.

Приоритеты:

• ^ высокий

• \* / средний

• + - низкий

• ( ) самый низкий

Вход: 3 + 4 \* 2 / (1 - 5)^2

Читаем «3»

Добавим «3» к выходной строке

Выход: 3

Читаем «+»

Кладём «+» в стек

Выход: 3

Стек: +

Читаем «4»

Добавим «4» к выходной строке

Выход: 3 4

Стек: +

Читаем «\*»

Кладём «\*» в стек

Выход: 3 4

Стек: + \*

Читаем «2»

Добавим «2» к выходной строке

Выход: 3 4 2

Стек: + \*

Читаем «/»

Выталкиваем «\*» из стека в выходную строку, кладём «/» в стек

Выход: 3 4 2 \*

Стек: + /

Читаем «(»

Кладём «(» в стек

Выход: 3 4 2 \*

Стек: + / (

Читаем «1»

Добавим «1» к выходной строке

Выход: 3 4 2 \* 1

Стек: + / (

Читаем «-»

Кладём «-» в стек

Выход: 3 4 2 \* 1

Стек: + / ( -

Читаем «5»

Добавим «5» к выходной строке

Выход: 3 4 2 \* 1 5

Стек: + / ( -

Читаем «)»

Выталкиваем «-» из стека в выходную строку, выталкиваем «(»

Выход: 3 4 2 \* 1 5 -

Стек: + /

Читаем «^»

Кладём «^» в стек

Выход: 3 4 2 \* 1 5 -

Стек: + / ^

Читаем «2»

Добавим «2» к выходной строке

Выход: 3 4 2 \* 1 5 - 2

Стек: + / ^

Конец выражения

Выталкиваем все элементы из стека в строку

Выход: 3 4 2 \* 1 5 - 2 ^ / +

Еще пример.

**(6+10-4)/(1+1\*2)+1**  
  
Читаем "(".  
Добавляем ( в стек операций.  
Выход: пусто.  
Стек операций: (.  
  
Читаем «6»  
Добавляем «6» к выходной строке.  
Выход: 6.  
Стек операций: (.  
  
Читаем "+"  
Добавляем «+» в стек операций.  
Выход: 6.  
Стек операций: (, +.  
  
Читаем «10»  
Добавляем «10» к выходной строке.  
Выход: 6, 10.  
Стек операций: (, +.  
  
Читаем "-"  
*Так как текущий знак (-) имеет равный приоритет перед последним знаком в стеке (+) мы всё равно выталкиваем знак из стека в операций в выходную строку, а текущий добавляем в стек.*  
Выход: 6, 10, +.  
Стек операций: (, -.  
  
Читаем «4»  
Добавляем «4» к выходной строке.  
Выход: 6, 10, +, 4.  
Стек операций: (, -.  
  
Читаем ")"  
*Снова скобка, но теперь уже закрывающая. Здесь необходимо вытолкать все знаки из стека в массив до первой открывающей скобки. От обеих скобок нам попросту нужно избавиться.*  
Выталкиваем "-" в выходную строку. Избавляемся от скобок.  
Выход: 6,10, +, 4, -.  
Стек операций: пусто.  
  
Читаем "/"  
Добавляем в стек.  
Выход: 6,10, +, 4, -.  
Стек операций: /.  
  
Читаем "("  
Добавляем в стек.  
Выход: 6,10, +, 4, -.  
Стек операций: /, (.  
  
Читаем «1»  
Добавляем «1» к выходной строке.  
Выход: 6,10, +, 4, -, 1.  
Стек операций: /, (.  
  
Читаем "+"  
Добавляем в стек.  
Выход: 6,10, +, 4, -, 1.  
Стек операций: /, (, +.  
  
Читаем «1»  
Добавляем «1» к выходной строке.  
Выход: 6,10, +, 4, -, 1, 1.  
Стек операций: /, (, +.  
  
Читаем "\*"  
*Последний символ в стеке операций (+) имеет приоритет ниже, чем текущий знак (\*). Поэтому последний знак из стека мы не трогаем, а просто добавляем как обычно текущий в стек.*  
Добавляем в стек.  
Выход: 6,10, +, 4, -, 1, 1.  
Стек операций: /, (, +,\*.  
  
Читаем «2»  
Добавляем «2» к выходной строке.  
Выход: 6,10, +, 4, -, 1, 1, 2.  
Стек операций: /, (, +,\*.  
  
Читаем ")"  
*Снова закрывающая скобка, делаем все как в прошлый раз.*  
Выталкиваем \* и + в выходную строку. Избавляемся от скобок.  
Выход: 6,10, +, 4, -, 1, 1, 2, \*, +.  
Стек операций: /.  
  
Читаем "+"  
У знака деления приоритет выше. Выталкиваем / в выходную строку. Добавляем + в стек.   
Выход: 6,10, +, 4, -, 1, 1, 2, \*, +, /.  
Стек операций: +.  
  
Читаем «1»  
Добавляем «1» к выходной строке.  
Выход: 6,10, +, 4, -, 1, 1, 2, \*, +, /, 1.  
Стек операций: +.  
  
Выражение закончено. Снова выталкиваем всё из стека операций в выходную строку.  
Выход: 6,10, +, 4, -, 1, 1, 2, \*, +, /, 1, +.

Считаем.

Решение

1) {6,10, +, 4, -, 1, 1, 2, \*, +, /, 1, +} {Пусто}  
2) {10, +, 4, -, 1, 1, 2, \*, +, /, 1, +} {6}  
3) {+, 4, -, 1, 1, 2, \*, +, /, 1, +} {6,10}  
4) {4, -, 1, 1, 2, \*, +, /, 1, +} {16}  
5) {-, 1, 1, 2, \*, +, /, 1, +} {16,4}  
6) {1, 1, 2, \*, +, /, 1, +} {12}  
7) {1, 2, \*, +, /, 1, +} {12, 1}  
8) {2, \*, +, /, 1, +} {12, 1, 1}  
9) {\*, +, /, 1, +} {12, 1, 1, 2}  
10) {+, /, 1, +} {12, 1, 2}  
11) {/, 1, +} {12, 3}  
12) {1, +} {4}  
13) {+} {4, 1}  
13) {} {5}

Итог: (6+10-4)/(1+1\*2)+1=5

**Постановка задачи**

* 1. Перевести арифметическое выражение в обратную польскую запись, показать.
  2. Вычислить выражение.

Например, вычислите выражение 6/4\*5/2+(8+2\*5)/(1+3\*2-4)-5+(9/(1+2)+1)\*3;

((7-6.35)/6.5+9.9)/((1.2/36+1.2/0.25-21/16)/(169/24)) (ответ. 20); ((13.75+9+1/6)\*1.2)/((10.3-8-1/2)\*(5/9))+((6.8-3-3/5)\*(35/6))/((3+2/3-3-1/6)\*56)-27-1/6 (ответ. 1)

Дополнение.

Функция eval(code) позволяет выполнить код, переданный ей в виде строки.

Этот код будет выполнен в *текущей области видимости*.

Вызов eval возвращает последнее вычисленное выражение, например:

alert( eval('1+1') ); // 2