# Калькулятор

# Пример 1 -> класс CalculatorStake

Сложные вычислительные задачи обычно требуют больших объемов вычислений, поэтому к разработчикам языков программирования было предъявлено требование: максимально приблизить форму записи математических выражений в коде программы к естествен­ному языку математики.

Одну из первых областей системного програм­мирования составили исследования способов трансляции математических выражений.

В результате наибольшее распространение получил метод трансляции при помощи обратной польской записи, которую предложил польский математик Я. Лукашевич.

Существует 2 основных алгоритма получения обратной польской записи-

- получение из стека

- получение из дерева

**Мы рассматриваем алгоритм получения записи из стека.**

Получение обратной польской записи с использованием стека может осуществляться весьма просто на основе алгоритма, предложенного Дейкстрой, который ввел понятие стекового приоритета операций:

## Промежуточное действие – расстановка приоритетов (веса)

Для того, что бы понимать, какой знак имеет приоритет над каким – нужно создать map, где мы и укажем эти приоритеты

## Действие 1 – собрать строку

Смысл в том, что нам нужно правильно складывать из строки цифры и знаки в разные стеки (всего 2 стека)

Суть алгоритма – мы обходим сроку с лева направо. Числа переносятся в стек numbers, а знаки переносятся в стек symbols на основе следующих правил

1. Если встречаем цифру – помещаем в numbers
2. Если мы встречаем знак, и стек символов пустой – помещаем его в стек symbols
3. Если мы встречаем знак и стек символов не пустой – то проверяем «вес» этого знака, и последнего знака в стеке. Логика следующая. Если вес текущего элемента меньше или равен весу последнего элемента в стеке – то переносим знаки в стек numbers до тех пор, пока в стеке symbol не встретим знак, приоритет которого ниже, чем у нашего текущего элемента
4. Если встречаем открывающуюся скобку – помещаем ее в стек symbols
5. Пока не встретили закрывающуюся скобку – продолжаем проверять вес приходящих знаков с последним в стеке по схеме описанной выше.
6. Если встречаем закрывающую скобку – то мы должны перенести все символы из стека symbols в стек numbers по той же схеме. Но переносить будем до момента, пока не встретим открывающуюся скобку.
7. Если перенося знаки «между скобками» мы встретили открывающуюся скобку – то просто удаляем ее из стека !!! Никуда не переносим. Так же игнорируем закрывающуюся скобку и идем дальше по фору.
8. Если мы пришли к концу строки – а стек symbols не пустой – то мы пошагово просто переносим данные из symbols в стек numbers

### Пример.

200+5\*3-2/1

Приоритеты -> {

“+”, “- “: 1

“\*”, “/” :2

}

Шаг десятый – строка закончилась. Проверяем стек символов – в нем есть данные. Нужно их перенести в стек номеров.

Numbers = [200, 5, 3, \*, +, 2, 1, /, -]

Это и есть строка, которую мф хотели получить!!!

Таким образом мы получим из **выражения *a*+*b*\**c*+(*d*\**e*+*f*)\**g***

Мы получи

|  |  |
| --- | --- |
| **Шаг первый** – получили цифру – сложили ее в стек numbers | Numbers = {200}  Symbols = {} |
| **Шаг второй –** получили +. Проверяем. Стек символов – он пустой. Просто ставим в него плюс | Numbers = {200}  Symbols = {+} |
| **Шаг третий –** получили число 5 – добавляем его в номера | Numbers = {200, 5}  Symbols = {+} |
| **Шаг четвертый** – получили знак \* его вес равен 2. Проверяем стек символов – он не пустой. Достаем последний символ из стека, проверяем его вес – 1. Он меньше, чем вес пришедшего символа – значит просто складываем символ \* в стек. | Numbers = {200, 5}  Symbols = {+, \*} |
| **Шаг пятый** – получаем 3 – кладем ее в numbers | Numbers = {200, 5, 3}  Symbols = {+, \*} |
| **Шаг шестой** - получаем - . Проверяем последний символ в стеке символов – там умножить. У знака умножить приоритет равен 2, а у минуса он равен 1. Это означает что нужно достать знак умножить из стека знаков и переложить их в стек чисел  Проверяем дальше – у минуса и у плюса одинаковый вес – значит тоже переносим.  Проверяем дальше – стек символов пустой. Значит просто кладем в него минус. | Numbers = {200, 5, 3, \*}  Symbols = {+}  Numbers = {200, 5, 3, \*, +}  Symbols = {}  Numbers = {200, 5, 3, \*, +}  Symbols = {-} |
| Шаг седьмой – получаем 2 – кладем в номера | Numbers = {200, 5, 3, \*, +, 2}  Symbols = {-} |
| Шаг восьмой – получаем / проверяем с минусом. У минуса меньше вес, значит просто добавляем деление в стек. | Numbers = {200, 5, 3, \*, +, 2}  Symbols = {-, /} |
| Шаг девятый – получаем единицу. Это число – ее складываем в стек номеров. | Numbers = {200, 5, 3, \*, +, 2, 1}  Symbols = {-, /} |
| Шаг десятый – строка закончилась. Проверяем стек символов – в нем есть данные. Нужно их перенести в стек номеров. | Numbers = [200, 5, 3, \*, +, 2, 1, /, -]  Это и есть та строка, которую мы искали!!! |

запись постфиксного выражения (или польского выражения обратной зависимости) – > ***abc*\*+*de*\**f*+*g*\*+**

А из выражения 200+5\*3-2/1 получаем стек [200, 5, 3, \*, +, 2, 1, /, -]

## Действие 2 – посчитать

Теперь, когда мы собрали строку – нужно построить алгоритм ее считывания.

Смысл в том, что у нас есть стек numbers. В стеке лежат след данные ->

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Шаг1 – проверяем – число, идем дальше  И так пока не встретим знак  Встретили знак – берем 2 предыдущих числа и делаем операцию используя знак | После первого шага перемножаем числа и удаляем знак который использовали  Получаем  Шаг 2 – идем снизу опять – пока не встретим знак. Встретили – повторяем ту же операцию | Шаг3 | Шаг4 | Шаг5 | Шаг6 | рез |
| - |  |  |  |  |  |  |
| / |  |  |  |  |  |  |
| 1 | - |  |  |  |  |  |
| 2 | / |  |  |  |  |  |
| + | 1 | - |  |  |  |  |
| \* | 2 | / |  |  |  |  |
| 3 | + | 1 | - | - |  |  |
| 5 | 5\*3=15 | 2 | 2/1 | 2 |  |  |
| 200 | 200 | 200+15 | 215 | 215 | 215-2 | 213 |

Когда в стеке numbers остается одно число – это и есть результат.

# Пример 2 -> класс Calculator

Этот калькулятор простой – он не учитывает вариант скобочек.

Его логика так же основана на стеке, но стек собирается по другому.  
В момент когда мы получаем знак более высокого приоритета (в нашем случае это знак умножения или деления), мы берем из левой коллекции последнее число, проводим операцию со следующим числом, и сохраняем уже результат.

Таким образом мы избавляемся от знаков приоритета (то есть от знаков умножения и деления) и по факту получаем обычное выражение для калькулятора, в котором производятся все операции степ бай степ.

## Пример

То есть для выражения "200+5\*3-2/1"

Мы создаем две коллекции. Коллекцию символов и коллекцию чисел и разбираем стринг –

Numbers = ["200", "5", "3", "2", "1"]

Symb = ["", "+", "\*", "-", "/"]

Создаём третью коллекцию – сам стек

И сразу же кладем в нее первое число из numbers

Result = [“200”]

Начинаем проверки

|  |  |
| --- | --- |
| Проходим массив символов – получаем плюс. Это не знак умножения или деления – поэтому просто кладем в массив результата этот плюс и следующий элемент | Result = [“200”, “+”, “5”] |
| Берем следующий элемент из массива символов – это знак умножить. То есть нам нужно взять из массива результата число 5 и умножить его на следующее число – число 3. И результат положить в массив результатов (удалив из него 5) | Result = [“200”, “+”, “15”] |
| Получаем следующий знак – это минус.  Кладем в массив результата минус и двойку | Result = [“200”, “+”, “15”, “-“, “2”] |
| Следующий элемент – знак деления. Это приоритетный знак – следовательно достаем последнее число из массива результатов, делим его на следующее число массива номеров и получаем | Result = [“200”, “+”, “15”, “-“, “2”] |
| Массив знаков закончился. Теперь мы получили коллекцию, в которой присутствуют только знаки плюс и минус, и результат которой можно получить просто обойдя ее и сделав все вычисления | “200 + 15 – 2” |