Программирование на языке Python

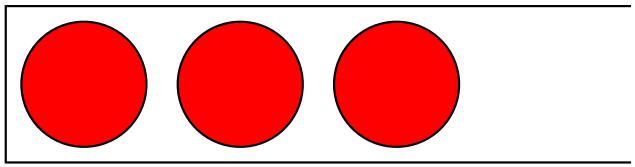
Структуры данных. Стек, очередь.

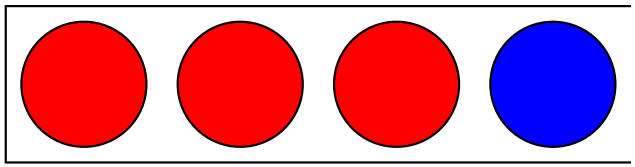
Алексей Сорокин

спецкурс, ОТИПЛ МГУ,
осенний семестр 2017–2018 учебного года
7 ноября 2017 г.

Стеки

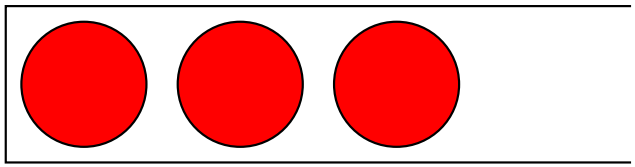
- Стек — структура данных, поддерживающая две основных операциии: добавление элемента в конец (push) и его удаление (pop).

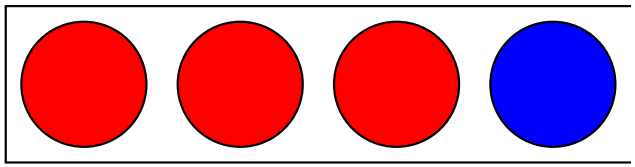




Стеки

- Стек — структура данных, поддерживающая две основных операциии: добавление элемента в конец (push) и его удаление (pop).

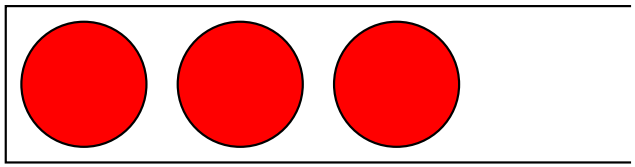


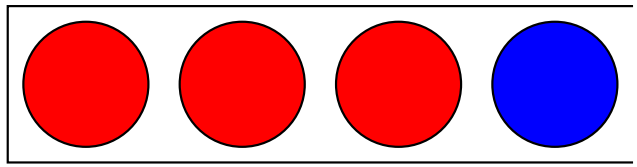


- Стек легко реализуется с помощью списка (append() — добавление, pop() — удаление).

Стеки

- Стек — структура данных, поддерживающая две основных операциии: добавление элемента в конец (push) и его удаление (pop).





- Стек легко реализуется с помощью списка (append() — добавление, pop() — удаление).
- Основные применения стека:
  - Организация вызовов функций.
  - Проверка на корректность (правильные скобочные последовательности, арифметические выражения), синтаксический анализ (контекстно-свободные грамматики).
  - Обход графов в глубину.

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

$$\{()[]\}([])$$

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

$$\{()[]\}([])$$

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

$$\{[]\}([])$$

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

$$\{\}([])$$

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

$$\{\}()$$

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.

$$()$$

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.
- Эквивалентное определение:
  - Пустая последовательность — правильная скобочная.

Правильные скобочные последовательности

- Правильная скобочная последовательность — такая последовательность из открывающих и закрывающих скобок, что её можно вычёркиванием стоящих рядом откраывающих и закрывающих скобок сократить до пустой.
- Эквивалентное определение:
  - Пустая последовательность — правильная скобочная.
  - Если w_1 правильная скобочная последовательность, а $(_i,)_i$ — открывающая и закрывающая скобка одного типа, то $(_i w).{}_i$ — правильная скобочная последовательность.
  - Если w_1 и w_2 — правильные скобочные последовательности, то $w_1 w_2$ — тоже.

Алгоритм проверки скобочной последовательности

- Задача: дана последовательность w, требуется проверить, является ли она правильной скобочной, при этом по последовательности можно двигаться только вперёд.

Алгоритм проверки скобочной последовательности

- Задача: дана последовательность w, требуется проверить, является ли она правильной скобочной, при этом по последовательности можно двигаться только вперёд.
- Будем читать последовательность посимвольно, помещая часть её символов в стек. Вначале работы стек пуст.

Алгоритм проверки скобочной последовательности

- Задача: дана последовательность w, требуется проверить, является ли она правильной скобочной, при этом по последовательности можно двигаться только вперёд.
- Будем читать последовательность посимвольно, помещая часть её символов в стек. Вначале работы стек пуст.
- Если очередной символ — открывающая скобка $($, то поместим её в стек.

Алгоритм проверки скобочной последовательности

- Задача: дана последовательность w, требуется проверить, является ли она правильной скобочной, при этом по последовательности можно двигаться только вперёд.
- Будем читать последовательность посимвольно, помещая часть её символов в стек. Вначале работы стек пуст.
- Если очередной символ — открывающая скобка $(_i$, то поместим её в стек.
- Если закрывающая скобка $)_i$, то в случае, если на вершине стека лежит не $(_i$, сообщим об ошибке.
- Иначе удалим со стека верхний символ.

Алгоритм проверки скобочной последовательности

- Задача: дана последовательность w, требуется проверить, является ли она правильной скобочной, при этом по последовательности можно двигаться только вперёд.
- Будем читать последовательность посимвольно, помещая часть её символов в стек. Вначале работы стек пуст.
- Если очередной символ — открывающая скобка $(_i$, то поместим её в стек.
- Если закрывающая скобка $)_i$, то в случае, если на вершине стека лежит не $(_i$, сообщим об ошибке.
- Иначе удалим со стека верхний символ.
- В конце работы проверим, является ли стек пустым. Если да, то последовательность правильная.

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

$$\{()[]\}([])$$

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) {

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) {(

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) {

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) {[

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) {

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([])

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) (

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) ([

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) (

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.

{()[]}([]) В конце работы стек пуст.

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность {()[]}([]) является правильной скобочной.
- Почему этот алгоритм корректен?

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность $\{()[]\}([])$ является правильной скобочной.
- Почему этот алгоритм корректен?
- Назовём нормальной формой $nf(w)$ строки w результат сокращения в ней всевозможных пар открывающих и закрывающих скобок.
- Например, $nf([]](\{()[]\}[()) =](([$.

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность $\{()[]\}([])$ является правильной скобочной.
- Почему этот алгоритм корректен?
- Назовём нормальной формой $nf(w)$ строки w результат сокращения в ней всевозможных пар открывающих и закрывающих скобок.
- Например, $nf([]](\{()[]\}[()) =](([$.
- У каждой строки существует только одна нормальная форма.

Алгоритм проверки скобочной последовательности

- Проверим, что последовательность $\{()[]\}([])$ является правильной скобочной.
- Почему этот алгоритм корректен?
- Назовём нормальной формой $nf(w)$ строки w результат сокращения в ней всевозможных пар открывающих и закрывающих скобок.
- Например, $nf([]](\{()[]\}[()) =](([$.
- У каждой строки существует только одна нормальная форма.
- В стеке всегда лежит нормальная форма уже прочитанной строки.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы ++ и -- в языке C++.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы $++$ и -- в языке $\mathrm{C}++$.
- В постфиксной нотации можно записывать и арифметические формулы.
- Например, $(3+4*2)-(7*5)$ преобразуется в $3\,4\,2\,*\,+7\,5\,*\,-$.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы $++$ и -- в языке $\mathrm{C}++$.
- В постфиксной нотации можно записывать и арифметические формулы.
- Например, $(3+4*2)-(7*5)$ преобразуется в $3\,4\,2\,*\,+7\,5\,*\,-$.
- Алгоритм вычисления выражения в постфиксной нотации:
  - Строка читается посимвольно слева направо.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы $++$ и -- в языке $\mathrm{C}++$.
- В постфиксной нотации можно записывать и арифметические формулы.
- Например, $(3+4*2)-(7*5)$ преобразуется в $3\,4\,2\,*\,+7\,5\,*\,-$.
- Алгоритм вычисления выражения в постфиксной нотации:
  - Строка читается посимвольно слева направо.
  - Если текущий символ — константа или переменная, поместить её значение в стек.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы $++$ и -- в языке $\mathrm{C}++$.
- В постфиксной нотации можно записывать и арифметические формулы.
- Например, $(3+4*2)-(7*5)$ преобразуется в $3\,4\,2\,*\,+7\,5\,*\,-$.
- Алгоритм вычисления выражения в постфиксной нотации:
  - Строка читается посимвольно слева направо.
  - Если текущий символ — константа или переменная, поместить её значение в стек.
  - Если текущий символ — знак операции op с k аргументами — снять со стека верхние k чисел $x_1, \ldots, x_k$. Если на стеке меньшее число элементов — выдать ошибку.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы $++$ и -- в языке $\mathrm{C}++$.
- В постфиксной нотации можно записывать и арифметические формулы.
- Например, $(3+4*2)-(7*5)$ преобразуется в $3\,4\,2\,*\,+7\,5\,*\,-$.
- Алгоритм вычисления выражения в постфиксной нотации:
  - Строка читается посимвольно слева направо.
  - Если текущий символ — константа или переменная, поместить её значение в стек.
  - Если текущий символ — знак операции op с k аргументами — снять со стека верхние k чисел $x_1, \ldots, x_k$. Если на стеке меньшее число элементов — выдать ошибку.
  - Вместо них поместить на стек $op(x_1, \ldots, x_k)$.

Постфиксная нотация

- Постфиксная нотация — способ записи, при котором операторы ставятся позади операндов.
- Например, операторы $++$ и -- в языке $\mathrm{C}++$.
- В постфиксной нотации можно записывать и арифметические формулы.
- Например, $(3+4*2)-(7*5)$ преобразуется в $3\,4\,2\,*\,+7\,5\,*\,-$.
- Алгоритм вычисления выражения в постфиксной нотации:
  - Строка читается посимвольно слева направо.
  - Если текущий символ — константа или переменная, поместить её значение в стек.
  - Если текущий символ — знак операции op с k аргументами — снять со стека верхние k чисел $x_1, \ldots, x_k$. Если на стеке меньшее число элементов — выдать ошибку.
  - Вместо них поместить на стек $op(x_1, \ldots, x_k)$.
  - Если в конце работы в стеке один элемент, вернуть его значение, иначе сообщить об ошибке.

Другие применения стека

- При вычислении арифметического выражения в постфиксной нотации оно параллельно проверяется на корректность.

Другие применения стека

- При вычислении арифметического выражения в постфиксной нотации оно параллельно проверяется на корректность.
- Также стек можно использовать для проверки выводимости в контекстно-свободной грамматике.

$$
\begin{array}{rcl}
S & \to & NP\ VP \\
NP & \to & A\ NP \\
NP & \to & N \\
VP & \to & VP\ ADV \\
VP & \to & VT\ NP
\end{array}
\qquad
\begin{array}{rcl}
NP & \to & Mary \\
A & \to & white \\
N & \to & cats \\
ADV & \to & greatly \\
VT & \to & likes
\end{array}
$$

Другие применения стека

- При вычислении арифметического выражения в постфиксной нотации оно параллельно проверяется на корректность.
- Также стек можно использовать для проверки выводимости в контекстно-свободной грамматике.

$$
\begin{array}{rcl}
S & \to & NP\ VP \\
NP & \to & A\ NP \\
NP & \to & N \\
VP & \to & VP\ ADV \\
VP & \to & VT\ NP
\end{array}
\qquad
\begin{array}{rcl}
NP & \to & Mary \\
A & \to & white \\
N & \to & cats \\
ADV & \to & greatly \\
VT & \to & likes
\end{array}
$$

- $[_S [_{NP} Mary]_{NP} [_{VP} [_{VP} [_{VT} likes]_{VT} [_{NP} [_A white]_A [_{NP} [_N cats]_N]_{NP}]_{NP}]_{VP} [_{ADV} greatly]_{ADV}]_{VP}]_S$.

Другие применения стека

- При вычислении арифметического выражения в постфиксной нотации оно параллельно проверяется на корректность.
- Также стек можно использовать для проверки выводимости в контекстно-свободной грамматике.

$$
\begin{array}{rcl}
S & \to & NP\ VP \\
NP & \to & A\ NP \\
NP & \to & N \\
VP & \to & VP\ ADV \\
VP & \to & VT\ NP
\end{array}
\qquad
\begin{array}{rcl}
NP & \to & Mary \\
A & \to & white \\
N & \to & cats \\
ADV & \to & greatly \\
VT & \to & likes
\end{array}
$$

- $[_S [_{NP} Mary]_{NP} [_{VP} [_{VP} [_{VT} likes]_{VT} [_{NP} [_A white]_A [_{NP} [_N cats]_N]_{NP}]_{NP}]_{VP} [_{ADV} greatly]_{ADV}]_{VP}]_S$.
- Проверка выводимости — проверка последовательности на корректность.

Другие применения стека

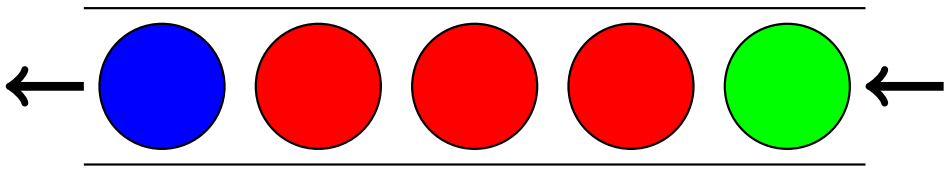
- При вычислении арифметического выражения в постфиксной нотации оно параллельно проверяется на корректность.
- Также стек можно использовать для проверки выводимости в контекстно-свободной грамматике.

$$
\begin{array}{rcl@{\qquad}rcl}
S & \to & NP\ VP & NP & \to & Mary \\
NP & \to & A\ NP & A & \to & white \\
NP & \to & N & N & \to & cats \\
VP & \to & VP\ ADV & ADV & \to & greatly \\
VP & \to & VT\ NP & VT & \to & likes
\end{array}
$$

- $[_S [_{NP} Mary]_{NP} [_{VP} [_{VP} [_{VT} likes]_{VT} [_{NP} [_A white]_A [_{NP} [_N cats]_N]_{NP}]_{NP}]_{VP} [_{ADV} greatly]_{ADV}]_{VP}]_S$.
- Проверка выводимости — проверка последовательности на корректность.
- Так же можно обрабатывать и грамматики зависимостей...

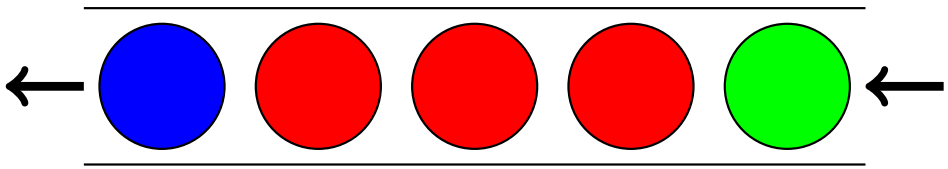
Очередь

- Очередь — структура данных, поддерживающая две операции: добавление в конец (Enqueue) и удаление из начала (Dequeue).



Очередь

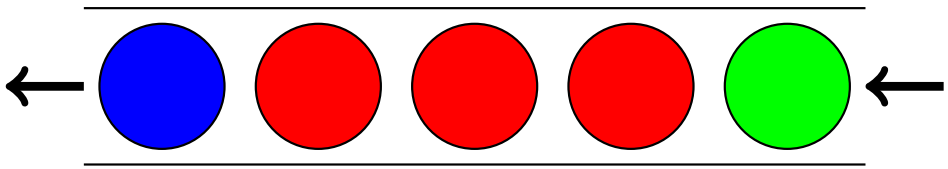
- Очередь — структура данных, поддерживающая две операции: добавление в конец (Enqueue) и удаление из начала (Dequeue).



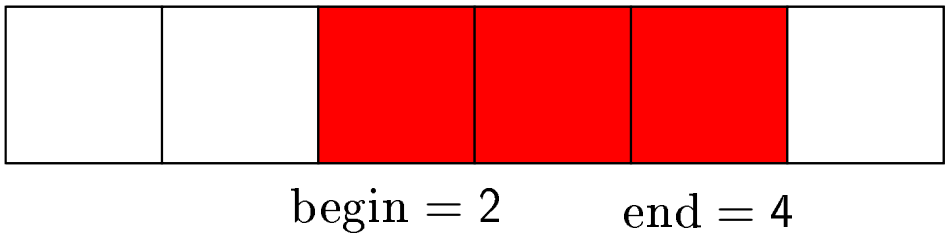
- Простейшая реализация: список, удаление из начала — $\mathrm{pop}(0)$, добавление в конец — append. Недостаток: удаление из начала — дорогая операция.

Очередь

- Очередь — структура данных, поддерживающая две операции: добавление в конец (Enqueue) и удаление из начала (Dequeue).



- Простейшая реализация: список, удаление из начала — $\mathrm{pop}(0)$, добавление в конец — append. Недостаток: удаление из начала — дорогая операция.
- Если заранее известен максимальный размер очереди n, то можно использовать циклическую очередь на списке длины n. Используются два указателя begin и end.



Очередь

- Циклическая очередь:

Очередь

- Циклическая очередь:
  - Инициализация: $\text{data} = [\text{None}] * \text{n},\ \text{start}, \text{end} = 0, 0$.

Очередь

- Циклическая очередь:
  - Инициализация: $\mathrm{data} = [\mathrm{None}] * \mathrm{n},\ \mathrm{start}, \mathrm{end} = 0, 0.$
  - Добавление в конец элемента x:
    $\mathrm{end} = (\mathrm{end} + 1)\%\mathrm{n},\ \mathrm{data}[\mathrm{end}] = x.$

Очередь

- Циклическая очередь:
  - Инициализация: $\mathrm{data} = [\mathrm{None}] * \mathrm{n},\ \mathrm{start}, \mathrm{end} = 0, 0$.
  - Добавление в конец элемента x:
    $\mathrm{end} = (\mathrm{end} + 1)\%\mathrm{n},\ \mathrm{data}[\mathrm{end}] = x$.
  - Извлечение элемента из очереди x:
    $\mathrm{start} = (\mathrm{start} + 1)\%\mathrm{n},\ \mathrm{return}\ \mathrm{data}[\mathrm{start}]$.

Очередь

- Циклическая очередь:
  - Инициализация: $\mathrm{data} = [\mathrm{None}] * \mathrm{n},\ \mathrm{start}, \mathrm{end} = 0, 0$.
  - Добавление в конец элемента x:
    $\mathrm{end} = (\mathrm{end} + 1)\%\mathrm{n},\ \mathrm{data}[\mathrm{end}] = x$.
  - Извлечение элемента из очереди x:
    $\mathrm{start} = (\mathrm{start} + 1)\%\mathrm{n},\ \mathrm{return}\ \mathrm{data}[\mathrm{start}]$.
  - Извлечение производится только при $\mathrm{start} != \mathrm{end}$.

Очередь

- Циклическая очередь:
  - Инициализация: $\mathrm{data} = [\mathrm{None}] * \mathrm{n},\ \mathrm{start}, \mathrm{end} = 0, 0$.
  - Добавление в конец элемента x:
    $\mathrm{end} = (\mathrm{end} + 1)\%\mathrm{n},\ \mathrm{data}[\mathrm{end}] = x$.
  - Извлечение элемента из очереди x:
    $\mathrm{start} = (\mathrm{start} + 1)\%\mathrm{n},\ \mathrm{return}\ \mathrm{data}[\mathrm{start}]$.
  - Извлечение производится только при $\mathrm{start} != \mathrm{end}$.
- Ещё можно поддерживать очередь на двух стеках, удаляя элементы из первого и добавляя во второй.
  - Добавление всегда производится на правый стек.

Очередь

- Циклическая очередь:
  - Инициализация: $\mathrm{data} = [\mathrm{None}] * \mathrm{n},\ \mathrm{start}, \mathrm{end} = 0, 0$.
  - Добавление в конец элемента x:
    $\mathrm{end} = (\mathrm{end} + 1)\%\mathrm{n},\ \mathrm{data}[\mathrm{end}] = x$.
  - Извлечение элемента из очереди x:
    $\mathrm{start} = (\mathrm{start} + 1)\%\mathrm{n},\ \mathrm{return}\ \mathrm{data}[\mathrm{start}]$.
  - Извлечение производится только при $\mathrm{start} != \mathrm{end}$.
- Ещё можно поддерживать очередь на двух стеках, удаляя элементы из первого и добавляя во второй.
  - Добавление всегда производится на правый стек.
  - Удаление производится с левого стека. Если там нет ни одного не элемента, то все элементы с правого стека поочерёдно перекладываются на левый стек (при этом их порядок изменится на противоположный). После этого осуществляется удаление вершины левого стека.