САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4

по курсу «Алгоритмы и структуры данных»

Тема: Стек, очередь, связанный список.

Вариант 1

Выполнила:

Аксянова А.Р

К3140

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc184227238)

[Задачи 3](#_Toc184227239)

[Задача №1. Стек 3](#_Toc184227240)

[Задача №3. Скобочная последовательность 5](#_Toc184227241)

[Задача №6. Очередь с минимумом 7](#_Toc184227242)

[Задача №7 . Максимум в движущейся последовательности. 9](#_Toc184227243)

[Задача №8. Постфиксная запись 11](#_Toc184227244)

[Задача №13 . Стек 13](#_Toc184227245)

[Результат работы всех задач 14](#_Toc184227246)

[Вывод 14](#_Toc184227247)

# Задачи

## Задача №1. Стек

1. Текст задачи

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ *N*”, либо “–”. Команда “+ *N*”означает добавление в стек числа *N*, по модулю не превышающего 109. Команда “–”означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 106 элементов.

**Формат входного файла (input.txt).** В первой строке входного файла содержится *M* (1 ≤ *M* ≤ 106) – число команд. Каждая последующая строка исходного файла содержит ровно одну команду.

**Формат выходного файла (output.txt).** Выведите числа, которые удаляются из стека с помощью команды “–”, по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.

1. Листинг кода

import time  
import random  
import os  
from lab4.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 1  
  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.stack = []  
  
 def push(self, value):  
 self.stack.append(value)  
  
 def pop(self):  
 return self.stack.pop()  
  
 def is\_empty(self):  
 return len(self.stack) == 0  
  
  
def process\_stack\_commands(commands):  
 stack = Stack()  
 output = []  
  
 for command in commands:  
 command = command.strip()  
 if command.startswith('+'):  
 \_, value = command.split()  
 stack.push(int(value))  
 elif command == '-':  
 if not stack.is\_empty():  
 output.append(stack.pop())  
 return output  
  
  
def task1():  
 m, commands = inp(PATH\_INPUT,0,'task1')  
 res = str(process\_stack\_commands(commands))  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

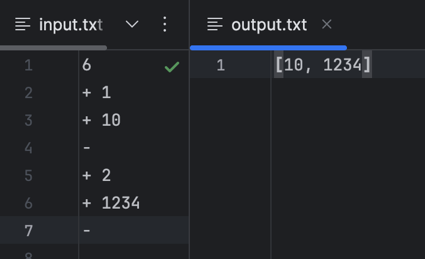
1. Текстовое объяснение решения.

стек организован с помощью класса Stack с тремя методами:

* \_\_init\_\_: стек как пустой список
* push: добавляет элемент в стек
* pop: извлекает последний добавленный элемент из стека
* is\_empty: проверяет, пуст ли стек, возвращая True, если стек пуст, и False в противном случае.

Функция process\_stack\_commands принимает список команд и выполняет обработку каждой команды. Если команда начинается с символа +, добавляет числа в стек, при этом из строки извлекается число и оно добавляется в стек с помощью метода push. Если команда равна -, из стека извлекается элемент с помощью метода pop), и этот элемент добавляется в список результатов.

1. Результат работы кода на примерах:



\*Результат работы кода (скрин терминала) так же будет приложен ниже

1. Вывод по задаче:

В результате выполнения задачи я научилась реализовывать работу стека.

## Задача №3. Скобочная последовательность

1. Текст задачи

Последовательность *A*, состоящую из символов из множества «(», «)», «[» и «]», назовем ***правильной скобочной последовательностью***, если выполняется одно из следующих утверждений:

* *A* – пустая последовательность;
* первыйсимволпоследовательности*A*–это«(»,ивэтойпоследовательности существует такой символ «)», что последовательность можно представить как *A* = (*B*)*C*, где *B* и *C* – правильные скобочные последовательности;
* первыйсимволпоследовательности*A*–это«[»,ивэтойпоследовательности существует такой символ «]», что последовательность можно представить как *A* = (*B*)*C*, где *B* и *C* – правильные скобочные последовательности.

Так, например, последовательности «(())» и «()[]» являются правильными скобочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

* **Формат входного файла (input.txt).** Первая строка входного файла содержит число *N* (1 ≤ *N* ≤ 500) – число скобочных последовательностей, которые необходимо проверить. Каждая из следующих *N* строк содержит скобочную последовательность длиной от 1 до 104 включительно. В каждой из последовательностей присутствуют только скобки указанных выше видов.
* **Форматвыходногофайла(output.txt).**Длякаждойстрокивходногофайла (кроме первой, в которой записано число таких строк) выведите в выходной файл «YES», если соответствующая последовательность является правильной скобочной последовательностью, или «NO», если не является.

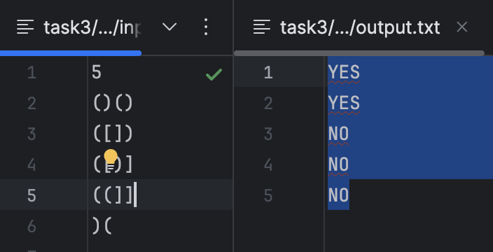
1. Листинг кода

import time  
import random  
import os  
from lab4.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 3  
  
def is\_valid\_sequence(s):  
 stack = []  
 matching\_bracket = {')': '(', ']': '['}  
 for char in s:  
 if char in "([":  
 stack.append(char)  
 elif char in ")]":  
 if stack and stack[-1] == matching\_bracket[char]:  
 stack.pop()  
 else:  
 return "NO"  
 return "YES" if not stack else "NO"  
  
def task1():  
 n, s = inp(PATH\_INPUT, 0, 'task1')  
 res = [is\_valid\_sequence(seq) for seq in s]  
 outp(PATH\_OUTPUT, "\n".join(res))  
 print(caption(task\_numb, "\n".join(res)))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption( task\_numb, time))

1. Текстовое объяснение решения.

is\_valid\_sequence(s)принимает строку и проверяет, является ли последовательность скобок в этой строке правильной. Функция steck проходит по каждому символу строки. Если символ — это открывающая скобка, она добавляется в стек. Если символ — закрывающая скобка то проверяется, соответствует ли она последней открывшейся скобке Если условие соблюдается, последняя открывающая скобка удаляется из стека. Если же нет, функция сразу возвращает "NO". В конце, если стек пуст функция возвращает "YES", иначе — "NO".

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась реализовывать проверку правильности последовательности скобок с использованием стека.

## Задача №6. Очередь с минимумом

1. Текст задачи

Реализуйте работу очереди. В дополнение к стандартным операциям очереди, необходимо также отвечать на запрос о минимальном элементе из тех, которые сейчас находится в очереди. Для каждой операции запроса минимального элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда – это либо «+ *N*», либо «–», либо «?». Команда «+ *N*» означает добавление в очередь числа *N*, по модулю не превышающего 109. Команда «–» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

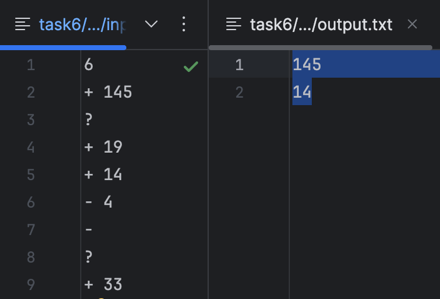
* **Формат входного файла (input.txt).** В первой строке содержится *M* (1 ≤ *M* ≤ 106) – число команд. В последующих строках содержатся команды, по одной в каждой строке.
* **Формат выходного файла (output.txt).** Для каждой операции поиска минимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Гарантируется, что операций извлечения или поиска минимума для пустой очереди не производится.

1. Листинг кода

import time  
import random  
import os  
from lab4.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 6  
  
  
def process\_commands(commands):  
 queue = []  
 min\_queue = []  
 results = []  
  
 for command in commands:  
 if command[0] == '+':  
 n = int(command[1:])  
 queue.append(n)  
 while min\_queue and min\_queue[-1] > n:  
 min\_queue.pop()  
 min\_queue.append(n)  
  
 elif command == '-':  
 removed = queue.pop(0)  
 if removed == min\_queue[0]:  
 min\_queue.pop(0)  
  
 elif command == '?':  
 results.append(str(min\_queue[0]))  
 return results  
  
  
def task1():  
 m, commands = inp(PATH\_INPUT, 0, 'task6')  
 res = process\_commands(commands)  
 outp(PATH\_OUTPUT, "\n".join(res))  
 print(caption(task\_numb, "\n".join(res)))  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption( task\_numb, time))

1. Текстовое объяснение решения.

process\_commands обрабатывает список команд. В этой функции используется две структуры данных: обычная очередь queue, которая реализуется как список, и дополнительная очередь min\_queue, которая отслеживает минимальные элементы в очереди для быстрого получения минимального значения. Число, следующее за символом +, добавляется в очередь, а также добавляется в очередь min\_queue, но перед этим из нее удаляются все элементы, которые больше добавляемого, чтобы в min\_queue оставались только элементы в порядке возрастания. Если команда равна -, это означает удаление элемента из очереди. Первый элемент удаляется из очереди, и если он совпадает с первым элементом в min\_queue, то этот элемент также удаляется из min\_queue, так как он больше не является минимальным. Результатом выполнения ? будет минимальный элемент, который всегда находится в первом месте очереди min\_queue.

1. Результат работы кода на примерах из текста задачи:
2. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм очереди и поиска минимального элемента в очереди.

## Задача №7 . Максимум в движущейся последовательности.

1. Текст задачи

Задан массив из *n* целых чисел - *a*1*,...,an* и число *m < n*, нужно найти максимум среди последовательности ("окна") {*ai,...,ai*+*m*−1} для каждого значения 1 ≤ *i* ≤ *n* − *m* +1. Простой алгоритм решения этой задачи за *O*(*nm*) сканирует каждое "окно"отдельно.

Ваша цель - алгоритм за *O*(*n*).

• **Формат входного файла (input.txt).** В первой строке содержится целое число*n*(1 ≤ *n* ≤ 105)–количествочиселвисходноммассиве,втораястрока содержит *n* целых чисел *a*1*,...,an* этого массива, разделенных пробелом

(0 ≤ *ai* ≤ 105). В третьей строке - целое число *m* - ширина "окна"(1 ≤ *m* ≤ *n*).

* **Формат выходного файла (output.txt).** Нужно вывести max*ai,...,ai*+*m*−1 для каждого 1 ≤ *i* ≤ *n* − *m* +1.

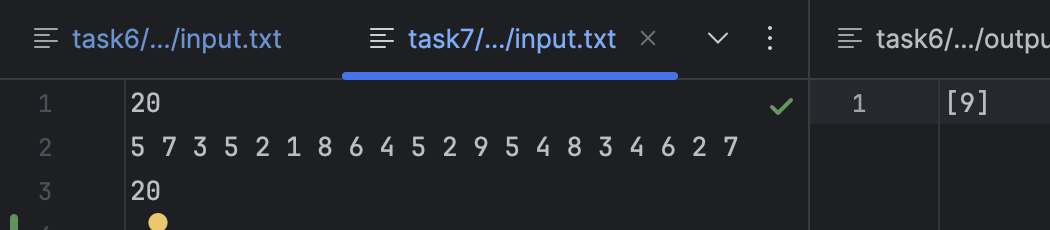
1. Листинг кода

import time  
import random  
import os  
from lab4.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 7  
  
def moving\_equence\_maximum(n, s, m):  
 q = []  
 result = []  
  
 for i in range(n):  
 if q and q[0] < i - m + 1:  
 q.pop(0)  
 while q and s[q[-1]] < s[i]:  
 q.pop()  
 q.append(i)  
 if i >= m - 1:  
 result.append(s[q[0]])  
 return result  
  
def task1():  
 n, s, m = inp(PATH\_INPUT, 0, 'task7')  
 res = str(moving\_equence\_maximum(n, s, m))  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption( task\_numb, time))

1. Текстовое объяснение решения.

moving\_equence\_maximum возвращает список максимальных элементов для каждой последовательности. В процессе работы функции используется очередь q, которая хранит индексы элементов, чтобы эффективно находить максимальные элементы в последовательности. Каждый элемент в очереди соответствует индексу в последовательности, и порядок индексов в очереди обеспечивает правильный доступ к максимальным элементам. Для каждого элемента последовательности проверяется, нужно ли удалить индексы из очереди, которые уже не принадлежат текущему отрезку. Затем, пока очередь не пуста и последний элемент в очереди меньше текущего элемента, из очереди удаляются все элементы, которые меньше текущего. Индекс текущего элемента добавляется в очередь. После того как индексы обработаны, если текущая последовательность имеет достаточную длину, то в список result добавляется максимальный элемент текущего, который всегда находится на первом месте в очереди. После обработки всех элементов последовательности функция возвращает список максимальных значений для каждого куска.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я написала алгоритм поиска максимальных элементов в движущейся последовательности

## Задача №8. Постфиксная запись

1. Текст задачи

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как A B +. Запись B C + D \* обозначает привычное нам (B + C) \* D, а запись A B C + D \* + означает A + (B + C) \* D. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

Дано выражение в обратной польской записи. Определите его значение. • **Формат входного файла (input.txt).** В первой строке входного файла дано число *N* (1 ≤ *n* ≤ 106) – число элементов выражения. Во второй строке содержится выражение в постфиксной записи, состоящее из *N* элементов. В выражении могут содержаться неотрицательные однозначные числа и операции +, -, \*. Каждые два соседних элемента выражения разделены ровно одним пробелом.

* **Формат выходного файла (output.txt).** Необходимо вывести значение записанного выражения. Гарантируется, что результат выражения, а также результаты всех промежуточных вычислений, по модулю будут меньше, чем

231.

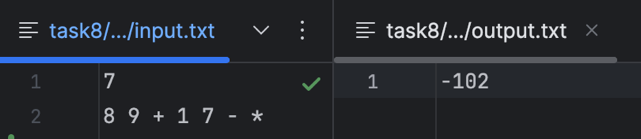
1. Листинг кода

import time  
import random  
import os  
from lab4.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 8  
  
def postfix(n, s):  
 stack = []  
 if isinstance(s, list):  
 s = ' '.join(s)  
 els = s.split()  
  
 for el in els:  
 if el.isdigit():  
 stack.append(int(el))  
 else:  
 b = stack.pop()  
 a = stack.pop()  
  
 if el == '+':  
 stack.append(a + b)  
 elif el == '-':  
 stack.append(a - b)  
 elif el == '\*':  
 stack.append(a \* b)  
  
 return stack[0]  
  
def task1():  
 n, s = inp(PATH\_INPUT, 0, 'task8')  
 res = str(postfix(n, s))  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption( task\_numb, time))

1. Текстовое объяснение решения.

Для выполнения вычислений используется стек, в нем Каждый элемент в выражении перебирается по очереди. Если элемент является числом он добавляется в стек.Если элемент является оператором , из стека извлекаются два последних числа. Сначала извлекается правый b, затем левый a, после чего выполняется соответствующая операция.Результат операции добавляется обратно в стек.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм постфиксной записи.

## Задача №13 . Стек

1. Текст задачи

Реализуйте стек на основе связного списка с функциями isEmpty, push, pop и вывода данных.

1. Листинг кода

class Node:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.next = None  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.top = None  
  
 def isEmpty(self):  
 return self.top is None  
  
 def push(self, value):  
 new\_node = Node(value)  
 new\_node.next = self.top  
 self.top = new\_node  
  
 def pop(self):  
 if self.isEmpty():  
 raise IndexError("pop from empty stack")  
 value = self.top.value  
 self.top = self.top.next  
 return value  
  
 def printstack(self):  
 if self.isEmpty():  
 print('Stack is Empty')  
 return  
 curr = self.top  
 print("Stack is:", end=' ')  
 while curr is not None:  
 print(curr.value, end=' ')  
 curr = curr.next  
 print()

n, m, A, B = inp(PATH\_INPUT,0 ,'task6')  
 res = str(sorting\_integers(A, B))  
 print("LAB3 Task6 answer:", res)  
 outp(PATH\_OUTPUT, res)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 print(f"LAB3 Task6 Time: {time.perf\_counter() - start}")

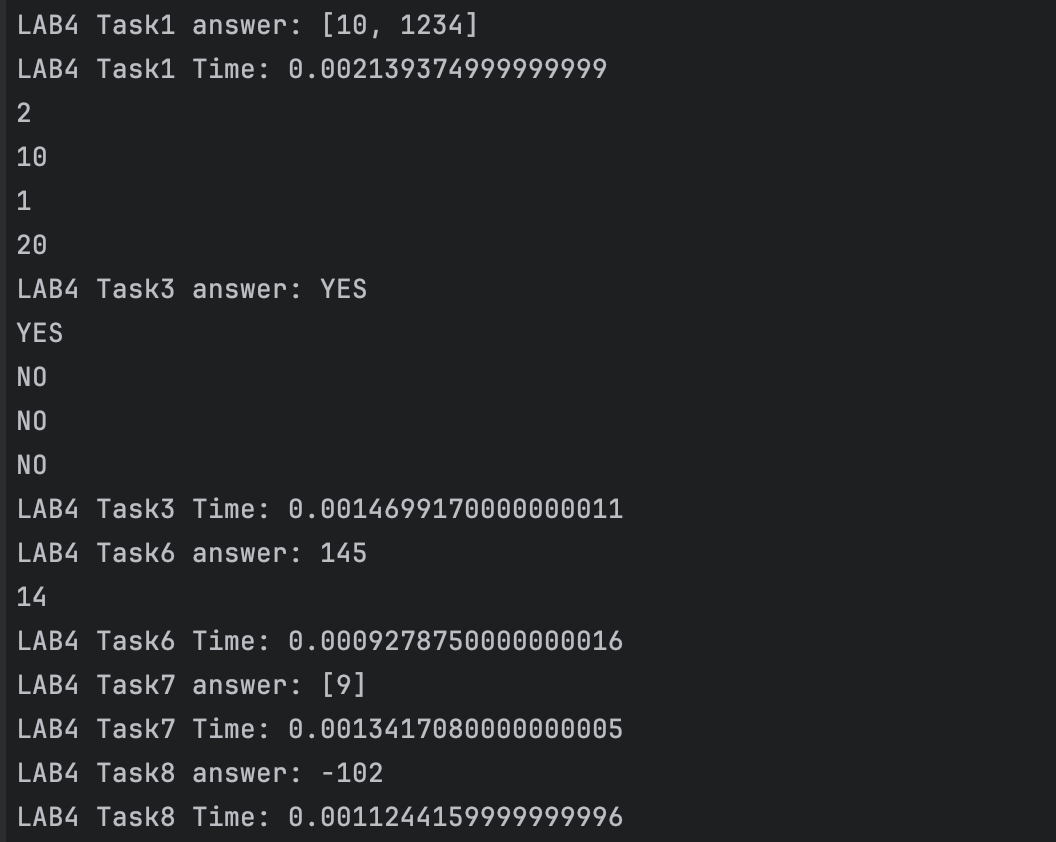
1. Текстовое объяснение решения.

Stack управляет операциями добавления и удаления элементов с помощью связного списка. Метод isEmpty проверяет, пуст ли стек, путем проверки, указывает ли top на None. Метод push добавляет новый элемент в стек. Метод pop извлекает элемент из стека. Перед тем как извлечь элемент, метод проверяет, пуст ли стек, с помощью метода isEmpty. Если стек пуст, возникает ошибка. Если в стеке есть элементы, то извлекается значение верхнего элемента, и верхний элемент стека удаляется. При этом указатель top обновляется. Метод printstack выводит все элементы стека на экран.

1. Вывод по задаче:

В результате выполнения задачи я научилась писать стек

## Результат работы всех задач



# Вывод

В результате лабораторной я написала стеки и очереди.