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: Тема работы

Вариант 23

Выполнил:

Юшков А.М.

К3139

Санкт-Петербург

2025г.

# Содержание отчета

[**Содержание отчета**](#_10udeakjagvs) **2**

[**Задачи по варианту**](#_gf7yxvsg0zb) **3**

[Задача №1. Стек](#_pgzbaj56cf0o) 3

[Задача №4.](#_pgzbaj56cf0o) Скобочная последовательность 2 4

[Задача №5.](#_pgzbaj56cf0o) Стек с максимумом 5

[Задача №11.](#_pgzbaj56cf0o) Бюрократия 7

[**Вывод**](#_fu90fuyk873) **8**

# Задачи по варианту

## Задача №1. Стек

from lab4.utils import file\_read, file\_write, measure  
  
  
class Stack:  
 def \_\_init\_\_(this):  
 this.stack = []  
 def push(this, value):  
 this.stack.append(value)  
 def pop(this):  
 return this.stack.pop() if this.stack else None  
  
def process\_stack\_commands(commands):  
 stack = Stack()  
 results = []  
  
 for command in commands:  
 if command.startswith('+'):  
 \_, value = command.split()  
 stack.push(int(value))  
 elif command == '-':  
 results.append(stack.pop())  
  
 return results  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 commands = file\_read()  
 result = measure(process\_stack\_commands, commands)  
 file\_write(result[0])

from lab4.utils import file\_read, file\_write, measure  
  
  
class Stack:  
 def \_\_init\_\_(this):  
 this.stack = []  
 def push(this, value):  
 this.stack.append(value)  
 def pop(this):  
 return this.stack.pop() if this.stack else None  
  
def process\_stack\_commands(commands):  
 stack = Stack()  
 results = []  
  
 for command in commands:  
 if command.startswith('+'):  
 \_, value = command.split()  
 stack.push(int(value))  
 elif command == '-':  
 results.append(stack.pop())  
  
 return results  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 commands = file\_read()  
 result = measure(process\_stack\_commands, commands)  
 file\_write(result[0])

Текстовое объяснение решения.

Программа эмулирует работу стека, обрабатывая команды добавления и извлечения элементов, и возвращает результаты извлечённых элементов.

Результат работы кода на примере из текста задачи

Input.txt

6  
+ 1  
+ 10  
-  
+ 2  
+ 1234  
-

Output.txt

10  
1234

## Задача №4. Скобочная последовательность 2

from lab4.utils import file\_read, file\_write, measure  
  
def check\_brackets(sequence):  
 stack = []  
 bracket\_pairs = {'(': ')', '[': ']', '{': '}'}  
  
 for i, char in enumerate(sequence):  
 if char in bracket\_pairs:  
 stack.append((char, i + 1))  
 elif char in bracket\_pairs.values():  
 if not stack:  
 return i + 1  
 top, pos = stack.pop()  
 if bracket\_pairs[top] != char:  
 return i + 1  
  
 if stack:  
 \_, pos = stack.pop()  
 return pos  
  
 return "Success"  
  
  
def process\_brackets():  
 data = file\_read()  
 result = check\_brackets(data[0])  
 file\_write([result])  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 measure(process\_brackets)

Программа проверяет, является ли последовательность скобок в строке корректной, и возвращает результат проверки или позицию первой ошибки.

Программа получает строку, содержащую скобки трёх типов: (), [], {}.

Используется стек для отслеживания открывающих скобок.

Программа проходит по каждому символу строки:

Если символ — открывающая скобка, она добавляется в стек вместе с её позицией.

Если символ — закрывающая скобка, проверяется, соответствует ли она последней открывающей скобке в стеке.

Если нет, программа возвращает позицию ошибки.

Если после обработки строки в стеке остаются открывающие скобки, программа возвращает позицию первой незакрытой скобки.

Если все скобки корректны, программа возвращает "Success".

Результат работы кода на примере из текста задачи

Input.txt

[]

Output.txt

Success

[**Задача №5.**](#_pgzbaj56cf0o) **Стек с максимумом**

from lab4.utils import file\_read, file\_write, measure  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.stack = []  
 self.max\_stack = []  
  
 def push(self, value):  
 self.stack.append(value)  
 if not self.max\_stack or value >= self.max\_stack[-1]:  
 self.max\_stack.append(value)  
  
 def pop(self):  
 if not self.stack:  
 return None  
 value = self.stack.pop()  
 if value == self.max\_stack[-1]:  
 self.max\_stack.pop()  
 return value  
  
 def max(self):  
 return self.max\_stack[-1] if self.max\_stack else None  
  
  
def process\_max\_stack():  
 commands = file\_read()  
 max\_stack = Stack()  
 results = []  
  
 for command in commands:  
 if command.startswith("push"):  
 \_, value = command.split()  
 max\_stack.push(int(value))  
 elif command == "pop":  
 max\_stack.pop()  
 elif command == "max":  
 results.append(max\_stack.max())  
  
 file\_write(results)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 measure(process\_max\_stack)

Используется основной стек для хранения элементов и дополнительный стек (max\_stack) для отслеживания максимальных значений.

При добавлении элемента (push):

Элемент добавляется в основной стек.

Если элемент больше или равен текущему максимуму (или стек максимумов пуст), он также добавляется в max\_stack.

При удалении элемента (pop):

Элемент удаляется из основного стека.

Если удалённый элемент равен текущему максимуму, он также удаляется из max\_stack.

При запросе максимума (max):

Возвращается верхний элемент из max\_stack.

Программа эмулирует стек с дополнительной функцией поиска максимального элемента, обрабатывает команды и возвращает результаты запросов на максимум.

Результат работы кода на примере из текста задачи

Input.txt

10  
push 2  
push 3  
push 9  
push 7  
push 2  
max  
max  
max  
pop  
max

Output.txt

9  
9  
9  
9

[**Задача №11.**](#_pgzbaj56cf0o) **Бюрократия**

from lab4.utils import file\_read, file\_write, measure  
  
def process\_queue(n, m, demands):  
 queue = list(demands)  
  
 while m > 0 and queue:  
 current = queue.pop(0)  
 if current > 1:  
 queue.append(current - 1)  
 m -= 1  
  
 if not queue:  
 return ["-1"]  
 else:  
 return [str(len(queue))] + [" ".join(map(str, queue))]  
  
  
def process\_bureaucracy():  
 data = file\_read()  
 n, m = map(int, data[0].split())  
 demands = list(map(int, data[1].split()))  
 result = process\_queue(n, m, demands)  
 file\_write(result)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 measure(process\_bureaucracy)

Программа моделирует обработку очереди запросов с ограниченным количеством шагов. Она возвращает состояние очереди после выполнения всех шагов или указывает, что очередь опустела раньше.

Алгоритм

Программа обрабатывает очередь запросов:

На каждом шаге извлекается первый запрос из очереди.

Если запрос требует больше одного шага для обработки (current > 1), он возвращается в конец очереди с уменьшенным значением (current - 1).

Количество оставшихся шагов m уменьшается на 1.

Если все шаги m исчерпаны или очередь пуста, программа завершает работу.

Результат работы кода на примере из текста задачи

Input.txt

4 5  
2 5 2 3

Output.txt

3   
4, 1, 2

# Вывод

В ходе работы были изучены и реализованы ключевые структуры данных (стек, очередь) и алгоритмы для их обработки. Каждая программа решает конкретную задачу, демонстрируя эффективное использование структур данных и алгоритмических подходов. Результаты работы программ проверены на корректность, а их производительность оценена с помощью модуля measure.