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 9

Выполнила:

Коновалова Кира Романовна

К3139

Проверил:

Афанасьев А.В

Санкт-Петербург

2024 г.

# Содержание отчета

**[Задачи по варианту. Задачи по выбору 3](#_Toc14454)**

[Задача №1. Стек 3](#_Toc10056)

[Задача №3. Скобочная последовательность. Версия 1 6](#_Toc15570)

[Задача №5. Стек с максимумом (по выбору) 9](#_Toc10720)

[Задача №6. Очередь с минимумом 13](#_Toc7841)

[Задача 16](#_Toc32208)

[Задача №9. Поликлиника 19](#_Toc5218)

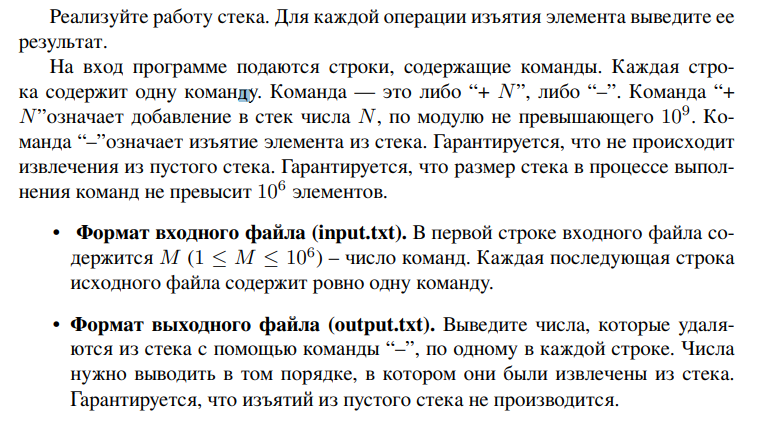
**[Запуск всех задач: 22](#_Toc12030)**

**[Вывод 23](#_Toc5154)**

# Задачи по варианту. Задачи по выбору

## Задача №1. Стек

Текст задачи:



Листинг кода:

import sys  
import os  
  
def process\_stack(commands):  
 stack = []  
 result = []  
  
 for command in commands:  
 if command.startswith('+'):  
 \_, num = command.split()  
 stack.append(int(num))  
 elif command == '-':  
 result.append(stack.pop())  
  
 return result  
  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 num\_commands = int(lines[0].strip())  
 commands = [line.strip() for line in lines[1:num\_commands + 1]]  
  
 # Проверяем массив  
 result = process\_stack(commands)  
  
 # Записываем результат в выходной файл  
 output\_file.write(f"{result}\n")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

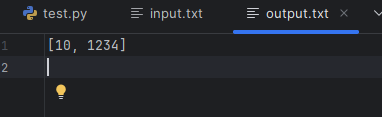
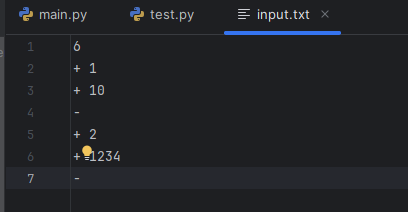
Текстовое объяснение задачи:

Был создан функционал для обработки списка команд. Был создан функционал для обработки списка команд. Для команды “+ N” число добавлялось в стек. Для команды “-” производилось изъятие элемента, и этот элемент добавлялся в список результатов.

Тесты:

import unittest  
from lab4.task1.src.main import process\_stack  
  
  
class TestStack(unittest.TestCase):  
 def test\_case\_example(self):  
 commands = [  
 "6",  
 "+ 1",  
 "+ 10",  
 "-",  
 "+ 2",  
 "+ 1234",  
 "-"  
 ]  
 expected = [10, 1234]  
 self.assertEqual(process\_stack(commands), expected)  
  
  
 def test\_case\_1(self):  
 commands = [  
 "+ 5",  
 "+ 10",  
 "-",  
 "+ 15",  
 "-",  
 "-"  
 ]  
 expected = [10, 15, 5]  
 self.assertEqual(process\_stack(commands), expected)  
  
 def test\_case\_2(self):  
 commands = [  
 "+ 1",  
 "+ 2",  
 "+ 3",  
 "-",  
 "-",  
 "-"  
 ]  
 expected = [3, 2, 1]  
 self.assertEqual(process\_stack(commands), expected)  
  
 def test\_case\_3(self):  
 commands = [  
 "+ 100",  
 "-",  
 "+ 200",  
 "+ 300",  
 "-",  
 "-"  
 ]  
 expected = [100, 300, 200]  
 self.assertEqual(process\_stack(commands), expected)  
  
 def test\_case\_4(self):  
 commands = [  
 "+ 42",  
 "+ 17",  
 "-",  
 "+ 8",  
 "+ 9",  
 "-",  
 "-",  
 "-"  
 ]  
 expected = [17, 9, 8, 42]  
 self.assertEqual(process\_stack(commands), expected)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

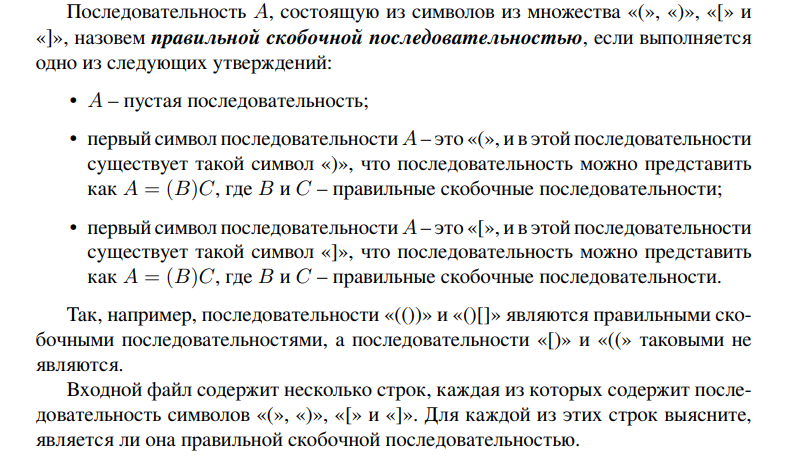


Вывод по задаче:

Реализованный алгоритм обеспечивает быструю обработку команд для стека и правильную работу со входными данными

## **Задача №3. Скобочная последовательность. Версия 1**

Текст задачи:



Листинг кода:

import os  
import sys  
def is\_correct\_sequence(sequence):  
 stack = []  
 matching\_brackets = {')': '(', ']': '['}  
 for char in sequence:  
 if char in '([':  
 stack.append(char)  
 elif char in ')]':  
 if stack and stack[-1] == matching\_brackets[char]:  
 stack.pop()  
 else:  
 return "NO  
 return "YES" if not stack else "NO"  
  
def process\_sequences(sequences):  
 return [is\_correct\_sequence(seq) for seq in sequences]  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 n = int(lines[0].strip()) # Число строк  
 sequences = [line.strip() for line in lines[1:n + 1]]  
 results = process\_sequences(sequences)  
 output\_file.write("\n".join(results) + "\n")  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

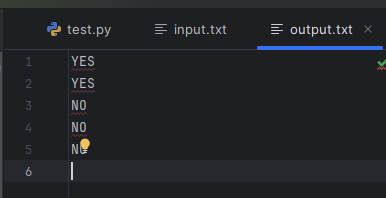
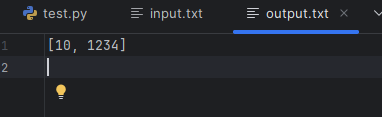
Текстовое объяснение задачи:

Требовалось для нескольких входных строк узнать, являются ли они правильными скобочными последовательностями.Последовательность считается правильной, если она либо пустая, либо каждая открывающая скобка имеет соответствующую закрывающую в правильном порядке. Был реализован алгоритм с использованием стека. ткрывающие скобки добавляются в стек, а при встрече закрывающей проверяется соответствие с верхним элементом стека. Если стек оказывается пустым или порядок нарушен, последовательность считается неправильной. В результате возвращается "YES" или "NO".

Тесты:

import unittest  
from lab4.task3.src.main import is\_correct\_sequence, process\_sequences  
  
class TestBracketSequence(unittest.TestCase):  
 def test\_valid\_sequences(self):  
 self.assertEqual(is\_correct\_sequence("()"), "YES")  
 self.assertEqual(is\_correct\_sequence("[]"), "YES")  
 self.assertEqual(is\_correct\_sequence("()[]"), "YES")  
 self.assertEqual(is\_correct\_sequence("(())"), "YES")  
 self.assertEqual(is\_correct\_sequence("([])"), "YES")  
 self.assertEqual(is\_correct\_sequence("[([])]"), "YES")  
  
 def test\_invalid\_sequences(self):  
 self.assertEqual(is\_correct\_sequence(")("), "NO")  
 self.assertEqual(is\_correct\_sequence("["), "NO")  
 self.assertEqual(is\_correct\_sequence("])"), "NO")  
 self.assertEqual(is\_correct\_sequence("(("), "NO")  
 self.assertEqual(is\_correct\_sequence("([)]"), "NO")  
 self.assertEqual(is\_correct\_sequence("[(])"), "NO")  
  
 def test\_process\_sequences(self):  
 sequences = ["()", "([])", "[([)])", "((()))", "(]", "[()]"]  
 expected = ["YES", "YES", "NO", "YES", "NO", "YES"]  
 self.assertEqual(process\_sequences(sequences), expected)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

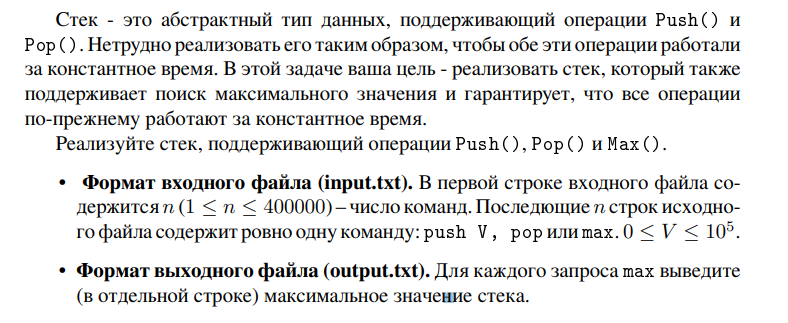


Вывод по задаче:

Алгоритм корректно определяет правильность скобочных последовательностей, используя эффективный подход с использованием стека

## **Задача №5. Стек с максимумом (по выбору)**

Текст задачи:



Листинг кода:

import os  
import sys  
class StackWithMax:  
 def \_\_init\_\_(self):  
 self.stack = []  
 self.max\_stack = []  
  
 def push(self, value):  
 self.stack.append(value)  
 # Если стек с максимумами пуст или текущий элемент >= текущего максимума  
 if not self.max\_stack or value >= self.max\_stack[-1]:  
 self.max\_stack.append(value)  
  
 def pop(self):  
 if not self.stack:  
 return  
 # Если удаляемый элемент равен текущему максимуму, удаляем его и из max\_stack  
 if self.stack[-1] == self.max\_stack[-1]:  
 self.max\_stack.pop()  
 self.stack.pop()  
  
 def max(self):  
 if not self.max\_stack:  
 return None  
 return self.max\_stack[-1]  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 n = int(lines[0].strip()) # Число строк  
 commands = [line.strip() for line in lines[1:n + 1]]  
 stack = StackWithMax()  
 results = []  
 for command in commands:  
 if command.startswith("push"):  
 \_, value = command.split()  
 stack.push(int(value))  
 elif command == "pop":  
 stack.pop()  
 elif command == "max":  
 results.append(stack.max())  
 output\_file.write("\n".join(map(str, results)) + "\n")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

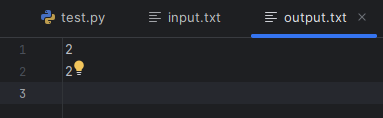
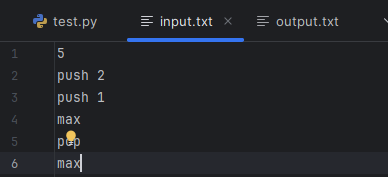
Текстовое объяснение задачи:

Необходимо реализовать стек, поддерживающий операции Push(), Pop() и Max(), где Max() возвращает максимальный элемент стека. Все операции должны выполняться за константное время. Для решения задачи был использован дополнительный стек для хранения текущих максимумов. При добавлении элемента в основной стек обновляется стек максимумов. При удалении элемента из основного стека также проверяется и обновляется стек максимумов. Операция Max() просто возвращает верхний элемент стека максимумов.

Тесты:

import unittest  
from io import StringIO  
from lab4.task5.src.main import StackWithMax, main  
  
class TestStackWithMax(unittest.TestCase):  
 def setUp(self):  
 self.stack = StackWithMax()  
  
 def test\_push\_and\_max(self):  
 self.stack.push(1)  
 self.assertEqual(self.stack.max(), 1)  
 self.stack.push(3)  
 self.assertEqual(self.stack.max(), 3)  
 self.stack.push(2)  
 self.assertEqual(self.stack.max(), 3)  
  
 def test\_pop\_and\_max(self):  
 self.stack.push(1)  
 self.stack.push(3)  
 self.stack.push(2)  
 self.stack.pop()  
 self.assertEqual(self.stack.max(), 3)  
 self.stack.pop()  
 self.assertEqual(self.stack.max(), 1)  
 self.stack.pop()  
 self.assertIsNone(self.stack.max())  
  
 def test\_mixed\_operations(self):  
 self.stack.push(5)  
 self.stack.push(1)  
 self.stack.push(5)  
 self.assertEqual(self.stack.max(), 5)  
 self.stack.pop()  
 self.assertEqual(self.stack.max(), 5)  
 self.stack.pop()  
 self.assertEqual(self.stack.max(), 5)  
 self.stack.pop()  
 self.assertIsNone(self.stack.max())  
  
class TestMainFunction(unittest.TestCase):  
 def run\_main(self, input\_data):  
 output = StringIO()  
 main(input\_data, output)  
 return output.getvalue().strip()  
  
 def test\_main\_example(self):  
 input\_data = """5  
push 1  
push 2  
max  
pop  
max"""  
 expected\_output = "2\n1"  
 self.assertEqual(self.run\_main(input\_data), expected\_output)  
  
 def test\_main\_all\_pop(self):  
 input\_data = """4  
push 10  
push 20  
pop  
pop"""  
 expected\_output = ""  
 self.assertEqual(self.run\_main(input\_data), expected\_output)  
  
 def test\_main\_only\_max(self):  
 input\_data = """3  
push 5  
max  
max"""  
 expected\_output = "5\n5"  
 self.assertEqual(self.run\_main(input\_data), expected\_output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

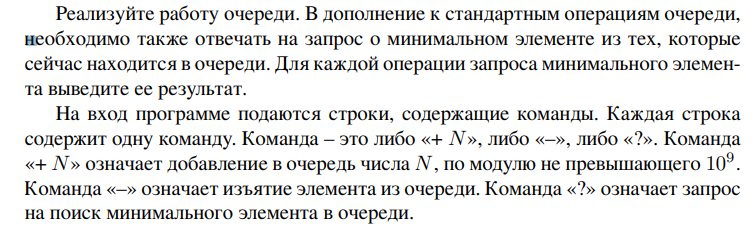


Вывод по задаче:

Алгоритм успешно обеспечивает выполнение всех операций за O(1), благодаря использованию дополнительного стека для хранения максимумов

## **Задача №6. Очередь с минимумом**

Текст задачи:



Листинг кода:

from collections import deque  
import os  
import sys  
  
class MinQueue:  
 def \_\_init\_\_(self):  
 self.queue = deque()  
 self.min\_queue = deque()  
  
 def push(self, value):  
 self.queue.append(value)  
 while self.min\_queue and self.min\_queue[-1] > value:  
 self.min\_queue.pop()  
 self.min\_queue.append(value)  
  
 def pop(self):  
 if self.queue:  
 value = self.queue.popleft()  
 if value == self.min\_queue[0]:  
 self.min\_queue.popleft()  
  
 def get\_min(self):  
 if self.min\_queue:  
 return self.min\_queue[0]  
  
  
def process\_commands(commands):  
 queue = MinQueue()  
 results = []  
  
 for command in commands:  
 if command.startswith("+"):  
 \_, value = command.split()  
 queue.push(int(value))  
 elif command == "-":  
 queue.pop()  
 elif command == "?":  
 results.append(queue.get\_min())  
  
 return results  
  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 m = int(lines[0].strip())  
 commands = [line.strip() for line in lines[1:m + 1]]  
  
 results = process\_commands(commands)  
  
  
 output\_file.write("\n".join(map(str, results)) + "\n")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

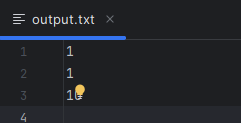
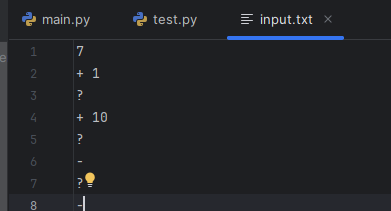
Текстовое объяснение задачи:

Использовалась основная очередь для хранения элементов и дополнительная очередь для отслеживания минимальных значений. При добавлении элемента в очередь обновляется минимальная очередь таким образом, чтобы в ней хранились только актуальные минимумы. Удаление элементов синхронизировано между двумя очередями. Запрос минимума возвращает первый элемент минимальной очереди

Тесты:

import unittest  
from lab4.task6.src.main import process\_commands  
  
  
class TestQueueWithMin(unittest.TestCase):  
 def test\_example(self):  
 commands = ["7", "+ 1", "?", "+ 10", "?", "-", "?", "-"]  
 expected = [1, 1, 10]  
 self.assertEqual(process\_commands(commands), expected)  
  
 def test\_all\_push(self):  
 commands = ["+ 5", "+ 2", "+ 8", "+ 1", "?"]  
 expected = [1]  
 self.assertEqual(process\_commands(commands), expected)  
  
 def test\_alternating\_commands(self):  
 commands = ["+ 10", "?", "+ 5", "?", "-", "?", "-"]  
 expected = [10, 5, 5]  
 self.assertEqual(process\_commands(commands), expected)  
  
 def test\_large\_input(self):  
 commands = ["+ 1"] \* 1000 + ["?"] \* 500 + ["-"] \* 500  
 expected = [1] \* 500  
 self.assertEqual(process\_commands(commands), expected)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

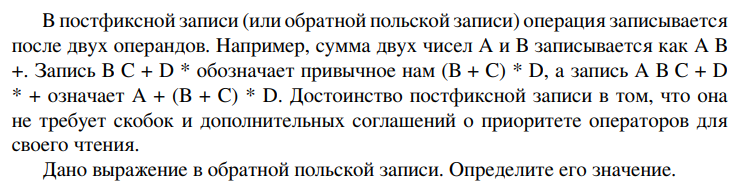


Вывод по задаче:

Алгоритм корректно выполняет все операции с очередью, включая определение минимума за O(1)

Задача **№8. Постфиксная запись**  **(по выбору)**

Текст задачи:



Листинг кода:

import sys  
import os  
  
def evaluate\_postfix(expression):  
 stack = []  
  
 for token in expression:  
 if token.isdigit():  
 stack.append(int(token))  
 else:  
 b = stack.pop()  
 a = stack.pop()  
 if token == '+':  
 stack.append(a + b)  
 elif token == '-':  
 stack.append(a - b)  
 elif token == '\*':  
 stack.append(a \* b)  
  
 return stack[0]  
  
def main(input\_data, output\_file):  
 lines = input\_data.strip().split("\n")  
 n = int(lines[0]) # Читаем количество элементов  
 expression = lines[1].strip().split() # Читаем выражение  
  
 result = evaluate\_postfix(expression)  
 output\_file.write(str(result) + "\n")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

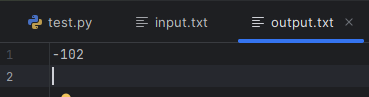
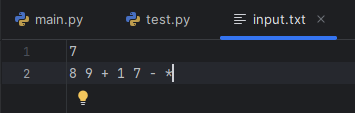
Текстовое объяснение задачи:

Для решения задачи использовался стек. Числа добавлялись в стек, а при встрече оператора выполнялась соответствующая операция над двумя верхними элементами стека. Результат операции помещался обратно в стек. В конце работы в стеке остается единственный элемент — результат выражения

Тесты:

import unittest  
from io import StringIO  
from lab4.task8.src.main import evaluate\_postfix, main  
class TestEvaluatePostfix(unittest.TestCase):  
 def test\_single\_number(self):  
 self.assertEqual(evaluate\_postfix(["5"]), 5)  
 def test\_simple\_addition(self):  
 self.assertEqual(evaluate\_postfix(["2", "3", "+"]), 5)  
  
 def test\_simple\_subtraction(self):  
 self.assertEqual(evaluate\_postfix(["5", "3", "-"]), 2)  
  
 def test\_simple\_multiplication(self):  
 self.assertEqual(evaluate\_postfix(["4", "3", "\*"]), 12)  
  
 def test\_complex\_expression(self):  
 self.assertEqual(evaluate\_postfix(["2", "3", "+", "4", "\*"]), 20) # (2 + 3) \* 4  
 self.assertEqual(evaluate\_postfix(["5", "1", "2", "+", "4", "\*", "+", "3", "-"]), 14) # 5 + ((1 + 2) \* 4) - 3  
  
class TestMainFunction(unittest.TestCase):  
 def run\_main(self, input\_data):  
 output = StringIO()  
 main(input\_data, output)  
 return output.getvalue().strip()  
  
 def test\_main\_example(self):  
 input\_data = """7  
2 3 + 4 \*"""  
 expected\_output = "20"  
 self.assertEqual(self.run\_main(input\_data), expected\_output)  
  
 def test\_main\_large\_expression(self):  
 input\_data = """9  
5 1 2 + 4 \* + 3 -"""  
 expected\_output = "14"  
 self.assertEqual(self.run\_main(input\_data), expected\_output)  
  
 def test\_main\_single\_number(self):  
 input\_data = """1  
5"""  
 expected\_output = "5"  
 self.assertEqual(self.run\_main(input\_data), expected\_output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

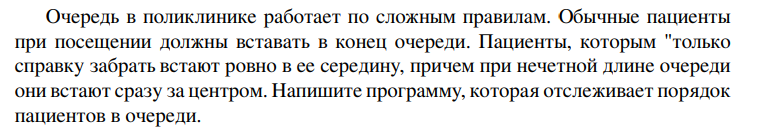


Вывод по задаче:

Алгоритм успешно обрабатывает выражения в постфиксной записи, корректно вычисляя их значение

## **Задача №9. Поликлиника**

Текст задачи:



Листинг кода:

from collections import deque  
import os  
import sys  
  
class PolyclinicQueue:  
 def \_\_init\_\_(self):  
 self.queue = deque()  
 self.middle\_index = 0  
  
 def add\_end(self, patient\_id):  
 self.queue.append(patient\_id)  
  
 def add\_middle(self, patient\_id):  
 self.middle\_index = (len(self.queue) + 1) // 2  
 self.queue.insert(self.middle\_index, patient\_id)  
  
 def serve\_patient(self):  
 return self.queue.popleft()  
  
def process\_requests(commands):  
 queue = PolyclinicQueue()  
 results = []  
  
 for command in commands:  
 if command.startswith("+"):  
 \_, patient\_id = command.split()  
 queue.add\_end(int(patient\_id))  
 elif command.startswith("\*"):  
 \_, patient\_id = command.split()  
 queue.add\_middle(int(patient\_id))  
 elif command == "-":  
 results.append(queue.serve\_patient())  
 return results  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 m = int(lines[0].strip())  
 commands = [line.strip() for line in lines[1:m + 1]]  
 results = process\_requests(commands)  
 output\_file.write("\n".join(map(str, results)) + "\n")  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

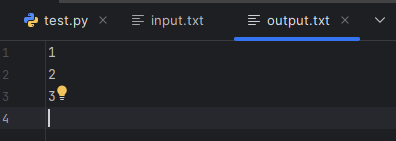
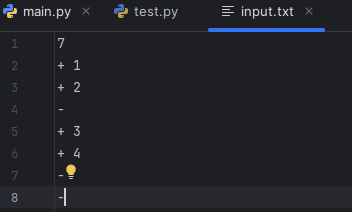
Текстовое объяснение задачи:

Программа моделирует очередь в поликлинике, обрабатывая три типа запросов: добавление пациента в конец очереди, добавление в середину и удаление пациента с начала очереди. Для реализации используется структура данных deque, которая позволяет эффективно добавлять и удалять элементы с концов очереди. Метод add\_middle вставляет пациента в середину очереди, а при нечетной длине — сразу за центром. Обработка команд происходит через функцию process\_requests, а результаты выводятся в файл. Код корректно обрабатывает запросы, при этом вставка в середину очереди может быть медленной при большом числе операций

Тесты:

import unittest  
from lab4.task9.src.main import process\_requests  
class TestPolyclinicQueue(unittest.TestCase):  
 def test\_example1(self):  
 commands = ["7", "+ 1", "+ 2", "-", "+ 3", "+ 4", "-", "-"]  
 expected = [1, 2, 3]  
 self.assertEqual(process\_requests(commands), expected)  
  
 def test\_example2(self):  
 commands = ["10", "+ 1", "+ 2", "\* 3", "-", "+ 4", "\* 5", "-", "-", "-", "-"]  
 expected = [1, 3, 2, 5, 4]  
 self.assertEqual(process\_requests(commands), expected)  
  
 def test\_simple\_case(self):  
 commands = ["+ 1", "+ 2", "\* 3", "-", "-"]  
 expected = [1, 3]  
 self.assertEqual(process\_requests(commands), expected)  
  
 def test\_only\_end\_and\_serve(self):  
 commands = ["+ 10", "+ 20", "+ 30", "-", "-", "-"]  
 expected = [10, 20, 30]  
 self.assertEqual(process\_requests(commands), expected)  
  
 def test\_large\_case(self):  
 commands = ["+ 1"] \* 1000 + ["-"] \* 1000  
 expected = [1] \* 1000  
 self.assertEqual(process\_requests(commands), expected)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

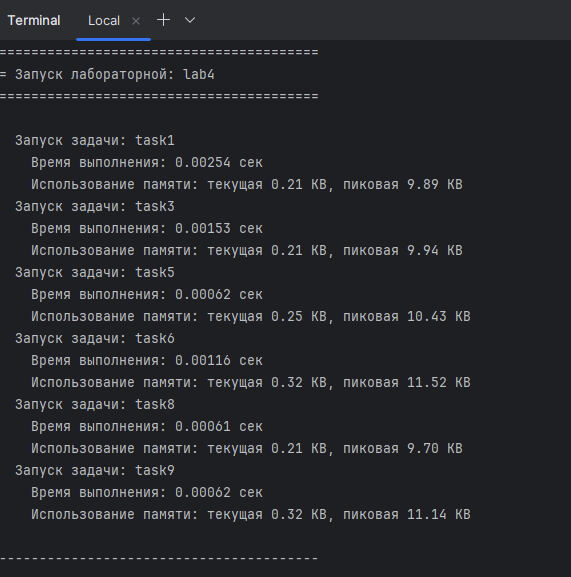
Результат работы кода на примерах из текста задачи:



Вывод по задаче:

Несмотря на эффективность использования deque для операций с концами очереди, вставка в середину приводит к ухудшению производительности при большом числе запросов, что ограничивает масштабируемость решения

# Запуск всех задач:



# Вывод

В рамках лабораторной работы были реализованы алгоритмы, демонстрирующие практическое применение структур данных: стеков, очередей и связанного списка. Каждая задача была решена с использованием эффективных методов, минимизирующих временные и пространственные затраты. Реализация задач позволила глубже понять основные принципы работы с этими структурами данных и их применение в реальных сценариях