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.

Выполнил:
Волжева М.И.
К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №2. Очередь

Текст задачи:
Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

- Формат входного файла (input.txt). В первой строке содержится M ($1 \leq M \leq 10^6$) – число команд. В последующих строках содержатся команды, по одной в каждой строке..
- Формат выходного файла (output.txt). Выведите числа, которые удаляются из очереди с помощью команды «–», по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из очереди. Гарантируется, что извлечения из пустой очереди не производится.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

Input.txt	4 + 1 + 10 - -
Output.txt	1 10

Листинг кода:

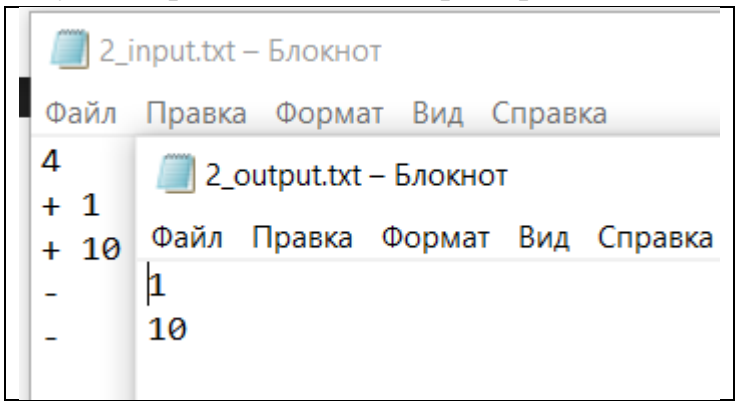
```
import time
import os, psutil


def add_element(queue, val):
    queue.append(val)


def remove_element(queue):
    delete = queue.pop(0)
    return delete


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("2_input.txt")
m = open("2_output.txt", "w")
```

```
queue = []
count = int(f.readline())
for i in range(count):
    string = f.readline()
    elements = list(map(str, string.split()))
    if elements[0] == "+":
        add_element(queue, elements[1])
    if elements[0] == "-":
        m.write(str(remove_element(queue))+"\n")


f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Элемент добавляется в конец массива, функцией append. Элемент удаляется из начала функцией pop(0).

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример из задачи	0.011281299999999994 second	13.61328125 mb

Вывод по задаче:.Мы научились реализовывать тип данных очередь.

Задача №6. Очередь с минимумом

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 10^9.
Команда «–» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди..

- Формат ввода или входного файла (input.txt). В первой строке содержится M ($1 \leq M \leq 10^6$) – число команд. В последующих строках содержатся команды, по одной в каждой строке.
- Формат выхода или 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.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

Input.txt	7 + 1 ? + 10 ? - ? -
Output.txt	1 1 10

Листинг кода:

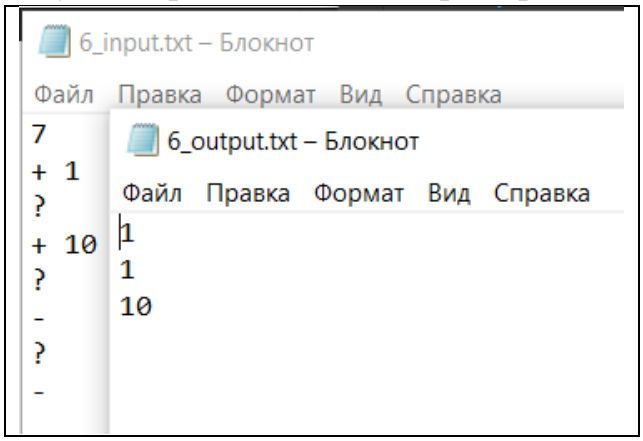
```
import time
import os, psutil
```

```
def add_element(queue, val):
    queue.append(val)


def remove_element(queue):
    delete = queue.pop(0)
    return delete


def find_min_element(queue):
    return min(queue)


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("6_input.txt")
m = open("6_output.txt", "w")

queue = []
count = int(f.readline())
for i in range(count):
    string = f.readline()
    elements = list(map(str, string.split()))
    if elements[0] == "+":
        add_element(queue, int(elements[1]))
    if elements[0] == "-":
        remove_element(queue)
    if elements[0] == "?":
        m.write(str(find_min_element(queue))+"\n")


f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Элемент добавляется в конец массива, функцией append. Элемент удаляется из начала функцией pop(0). Поиск в очереди осуществляется функцией min().

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример из задачи	0.0019382000000000001 second	13.48828125 mb

Вывод по задаче:. Мы научились реализовывать тип данных очередь и искать в очереди минимальный элемент.

Задача №3. Скобочная последовательность. Версия 1

Текст задачи:
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 ($1 \leq N \leq 500$) – число скобочных последовательностей, которые необходимо проверить. Каждая из следующих N строк содержит скобочную последовательность длиной от 1 до 10^4 включительно. В каждой из последовательностей присутствуют только скобки указанных выше видов.
- Формат выходного файла (output.txt Для каждой строки входного файла (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.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

Input.txt	5 ()() ([]) ([)] (([]])(
Output.txt	YES YES NO NO NO

Листинг кода

```
import time
import os, psutil


class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)
```

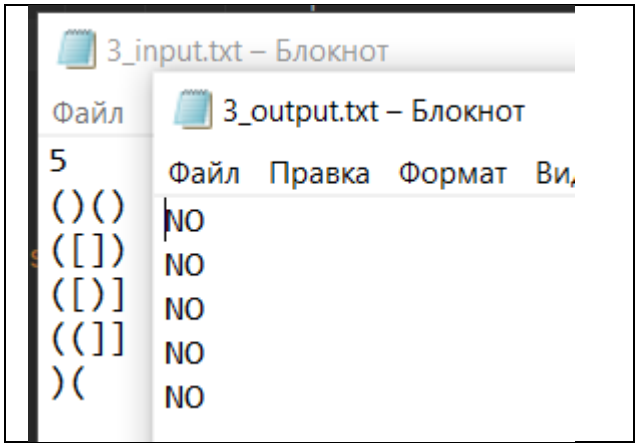
```
    def pop(self):
        if len(self.stack) == 0:
            return None
        removed = self.stack.pop()
        return removed

    def len(self):
        lenth = len(self.stack)
        return lenth

def pravilo(line: str):
    symbols = Stack()
    pravda = True
    for i in line:
        if i in "([":
            symbols.push(i)
        if i in ")]":
            if i == ")":
                if symbols.pop() != "(":
                    pravda = False
                    break
            if i == "]":
                if symbols.pop() != "[":
                    pravda = False
                    break

    if pravda and symbols.len():
        return "YES"
    else:
        return "NO"


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("3_input.txt")
m = open("3_output.txt", "w")

num = int(f.readline())
for i in range(num):
    line = f.readline()
    m.write(pravilo(line) + "\n")


f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Сначала был реализован класс стэк с некоторыми функциями для решение нашей задачи. Далее задача была разбита на N подзадач, для каждой вызывается функция pravilo(). Если в строчке находится какая-то открывающаяся скобка, то она добавляется в стэк. Если обнаруживается закрывающаяся скобка, то из стэка удаляется элемент и если он не

совпадает по форме с обнаруженной скобкой, то строчка не правильная и проверка прекращается. Если в момент основной проверки сбоев не было и к концу стэк пустой, то выводится -“YES”, иначе - “NO”

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример из задачи	0.0020649999999999974 second	13.5234375 mb

Вывод по задаче: Я научилась реализовывать класс данных стэк и определить правильность скобочной последовательности.

Задача №8. Постфиксная запись

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 ($1 \le n \le 10^6$) – 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 2^{31}
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

Input.txt	7 8 9 + 1 7 - *
Output.txt	-102

Листинг кода

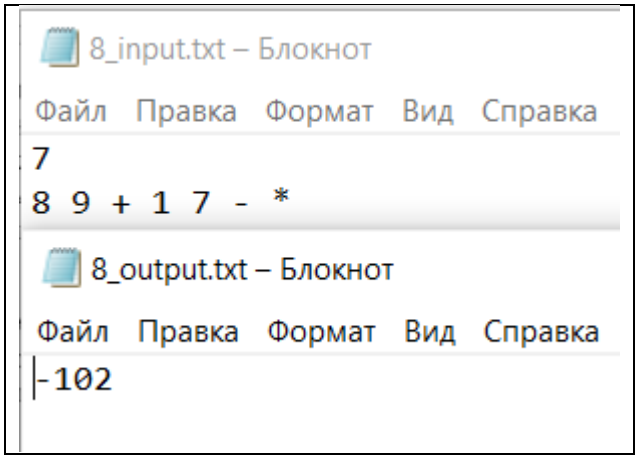
```
import time
import os, psutil


class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if len(self.stack) == 0:
            return None
        removed = self.stack.pop()
        return removed
```

```
    def len(self):
        lenth = len(self.stack)
        return lenth


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("8_input.txt")
m = open("8_output.txt", "w")

a = int(f.readline())
string = f.readline()
elements = list(map(str, string.split()))
num = Stack()
res = 0
for i in elements:
    if i in "+*-":
        if i == "-":
            num1 = int(num.pop())
            num2 = int(num.pop())
            res = num2 - num1
            num.push(res)
        if i == "+":
            res = int(num.pop()) + int(num.pop())
            num.push(res)
        if i == "*":
            res = int(num.pop()) * int(num.pop())
            num.push(res)
    else:
        num.push(i)

m.write(str(res))
f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Мы проходимся по строчке и сохраняем все значения(числа) в стэк, если мы находим знак математической операции, то мы вытаскиваем значения из стэка и записываем в него результат, полученный в результате математической операции.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример из задачи	0.0018680999999999975 second	13.56640625 mb

Вывод по задаче: Я узнала, что такое постфиксная запись, научилась читать и считать её.

Вывод

Я вспомнила как работать с классами, научилась реализовывать через классы такие типы данных, как очередь и стэк. Также научилась на практике использовать эти типы данных.