

Федеральное государственное автономное образовательное
учреждение высшего образования
Университет ИТМО

Факультет инфокоммуникационных технологий

Алгоритмы и структуры данных:

**Отчёт по лабораторной работе №4: Стек, очередь,
связанный список**

Выполнил:
Бочкарь Артём Артёмович

Группа: **K32392**

Преподаватели:
Артамонова В. Е.

Санкт-Петербург 2023 г.

Задача №1: Стек

В первом задании нужно реализовать работу стека с командами «+N» и «-N». В первой строке входного файла содержится M ($1 \leq M \leq 10^5$) – число команд. Каждая последующая строка исходного файла содержит ровно одну команду. В выходном файле нужно вывести числа, которые удаляются из стека с помощью команды «-», по одному в каждой строке:

```
import time

input = open("input.txt")
output = open("output.txt", 'w')
n = int(input.readline())
stack = list()
k = 0
while k != n:
    a = input.readline()
    if a[0] == '+':
        stack.append(a[2:])
        if len(stack) > n:
            output.write("Queue is full")
            break

    if a[0] == '-':
        output.write(stack[-1])
        k += 1

start = time.perf_counter()
print('Time spent: %s seconds' % (time.perf_counter() - start))
```

Тесты:

input.txt	
1	6
2	+ 1
3	+ 10
4	-
5	+ 2
6	+ 1234
7	-
8	
9	
10	
output.txt	
1	10
2	1234

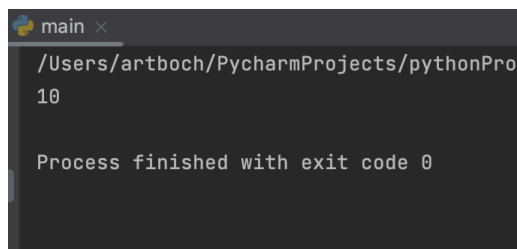
Задача №4: Скобочная последовательность. Версия 2

В четвертом задании от нас требовалось определить правильность скобочной последовательности и вывести “Success”, если всё правильно или вывести отсчитываемый от 1 индекс первой несовпадающей закрывающей скобки, а если нет несовпадающих закрывающих скобок, вывести отсчитываемый от 1 индекс первой открывающей скобки, не имеющей закрывающей:

```
def brack(text: str):
    stack = []
    for pos, element in enumerate(text, 1):
        if element in "({[" :
            stack.append(element)
        if element in ")]}" :
            left = stack.pop()
            if (
                (left == "(" and element != ")")
                or (left == "{" and element != "}")
                or (left == "[" and element != "]")
            ):
                return pos
    if len(stack) != 0:
        return text.index(stack[0]) + 1
    return "SUCCESS"

print(brack("foo(bar[i];"))
```

Тесты:



```
main x
/Users/artboch/PycharmProjects/pythonPro
10
Process finished with exit code 0
```

Задача №5: Стек с максимумом

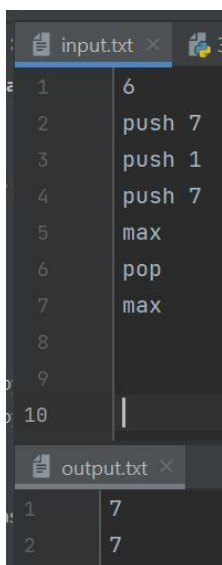
В 5 задании надо реализовать стек с максимумом, который поддерживает команды Push(), Pop() и Max(). В первой строке входного файла содержится n ($1 \leq n \leq 400000$) – число команд. Последующие n строк исходного файла содержит ровно одну команду: push V , pop или max. $0 \leq V \leq 10^5$. Во входном файле нужно вывести (в отдельной строке) максимальное значение стека после каждого запроса Max:

```
import time

input = open("input.txt")
output = open("output.txt", 'w')
n = int(input.readline())
stack = list()
k = 0
while k != n:
    a = input.readline()
    a = a.split()
    if a[0] == 'push':
        stack.append(a[1])
    elif a[0] == 'max':
        max = 0
        for i in stack:
            if int(i) > max:
                max = int(i)
        # output.write(str(max(stack)) + '\n')
        output.write(str(max) + '\n')
    elif a[0] == 'pop':
        del stack[-1]
    k += 1

start = time.perf_counter()
print('Time spent: %s seconds' % (time.perf_counter() - start))
```

Тесты:



Задача №9: Поликлиника

В 9 задании надо реализовать очередь в поликлинику с командами “+ i” (к очереди присоединяется пациент и встаёт в её конец), “* i” (пациент i встаёт в середину очереди) и “-” (первый пациент в очереди заходит к врачу). В первой строке входного файла записано одно целое число n ($1 \leq n \leq 10^5$) - число запросов к вашей программе, а в следующих - команды, описанные выше. Для каждого запроса третьего типа в отдельной строке нужно вывести в выходной файл номер пациента, который должен зайти к шаманам:

```
import time

input = open("input.txt")
output = open("output.txt", 'w')
n = int(input.readline())
stack = list()
k = 0
while k != n:
    a = input.readline()
    a = a.split()
    if a[0] == '+':
        stack.append(a[1])
    elif a[0] == '-':
        output.write(str(stack[0]) + '\n')
        del stack[0]
    elif a[0] == '*':
        if len(stack) % 2 == 0:
            midpoint = len(stack)//2
            stack = stack[0:midpoint] + [a[1]] + stack[midpoint:]
        else:
            midpoint = len(stack)//2+1
            stack = stack[0:midpoint] + [a[1]] + stack[midpoint:]
    k += 1

start = time.perf_counter()
print('Time spent: %s seconds' % (time.perf_counter() - start))
```

Тесты:

input.txt	
1	10
2	+ 1
3	+ 2
4	* 3
5	-
6	-
7	+ 4
8	* 5
9	-
10	-
11	-
12	-
13	

output.txt	
1	1
2	3
3	2
4	5

Задача №11: Бюрократия

В 11 задаче нужно реализовать очередь за справками. Утром в очередь встают n человек, i -й посетитель хочет получить a_i справок. За один прием можно получить только одну справку, поэтому если после приема посетителю нужны еще справки, он встает в конец очереди. За время приема министерство успевает выдать m справок. Остальным придется ждать следующего приемного дня. Ваша задача - сказать, сколько еще справок хочет получить каждый из оставшихся в очереди посетитель в тот момент, когда прием закончится. Если все к этому моменту разойдутся, выведите -1:

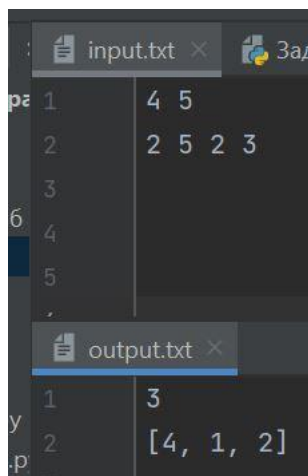
```
import time

input = open("input.txt")
output = open("output.txt", 'w')
n, k = map(int, input.readline().split())
stack = list(map(int, input.readline().split()))
f = 0
while f != k:
    stack[0] -= 1
    if stack[0] == 0:
        del stack[0]
        n -= 1
    else:
        stack.append(stack[0])
        del stack[0]
    f += 1

if len(stack) == 0:
    output.write(str(n))
else:
    output.write(str(n) + '\n')
    output.write(str(stack))

start = time.perf_counter()
print('Time spent: %s seconds' % (time.perf_counter() - start))
```

Тесты:



Задача №13.1: Реализация стека, очереди и связанных списков

В первом пункте нужно реализовать стек на основе связанного списка с функциями isEmpty, push, pop и вывода данных:

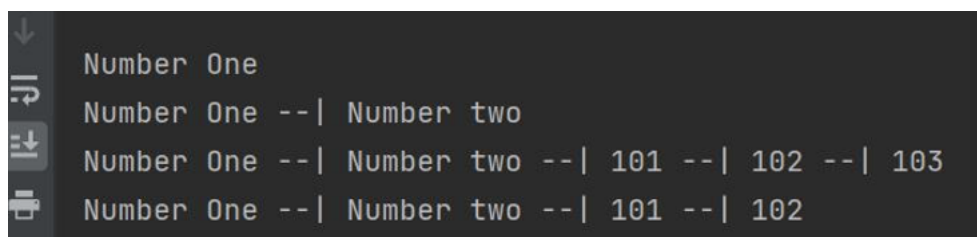
```
class Stack:
    def __init__(self):
        self.stack = []
    def is_empty(self):
        return len(self.stack) == 0
    def push(self, data):
        self.stack.append(data)
    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            raise Exception("Stack is empty")
    def __repr__(self):
        return " --| ".join(map(str, self.stack))
if __name__ == "__main__":
    test = Stack()
    print(test)
    test.push("Number One")
    print(test)

    test.push("Number two")
    print(test)

    test.push(101)
    test.push(102)
    test.push(103)
    print(test)

    test.pop()
    print(test)
```

Тесты:



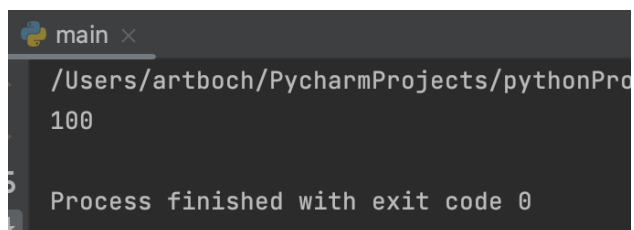
```
Number One
Number One --| Number two
Number One --| Number two --| 101 --| 102 --| 103
Number One --| Number two --| 101 --| 102
```

Задача №13.2: Реализация стека, очереди и связанных списков

Во втором пункте надо реализовать очередь на основе связанного списка с функциями Enqueue, Dequeue с проверкой на переполнение и опустошение очереди:

```
class Queue:
    def __init__(self, max=None):
        self.stack = []
        self.max = max
    def enqueue(self, item):
        if len(self.stack) != self.max:
            self.stack.append(item)
        else:
            raise Exception("Queue is full")
    def dequeue(self):
        if len(self.stack) != 0:
            return self.stack.pop(0)
        else:
            raise Exception("Queue is empty")
if __name__ == '__main__':
    test = Queue(3)
    test.enqueue(100)
    test.enqueue(101)
    test.enqueue(102)
    print(test.dequeue())
```

Тесты:



```
main x
/Users/artboch/PycharmProjects/pythonPro
100
Process finished with exit code 0
```