Лабораторная работа 4 - Стек, очередь, связанный список

Цель

Реализовать базовые структуры данных (очередь и стек) и решить задачи на их применение: обработка команд очереди, проверка скобочной последовательности, стек с поддержкой максимума, моделирование очереди в пекарне.

Задания и решения

Задание 2 - Очередь

По входу задается число команд M, дальше идут команды:

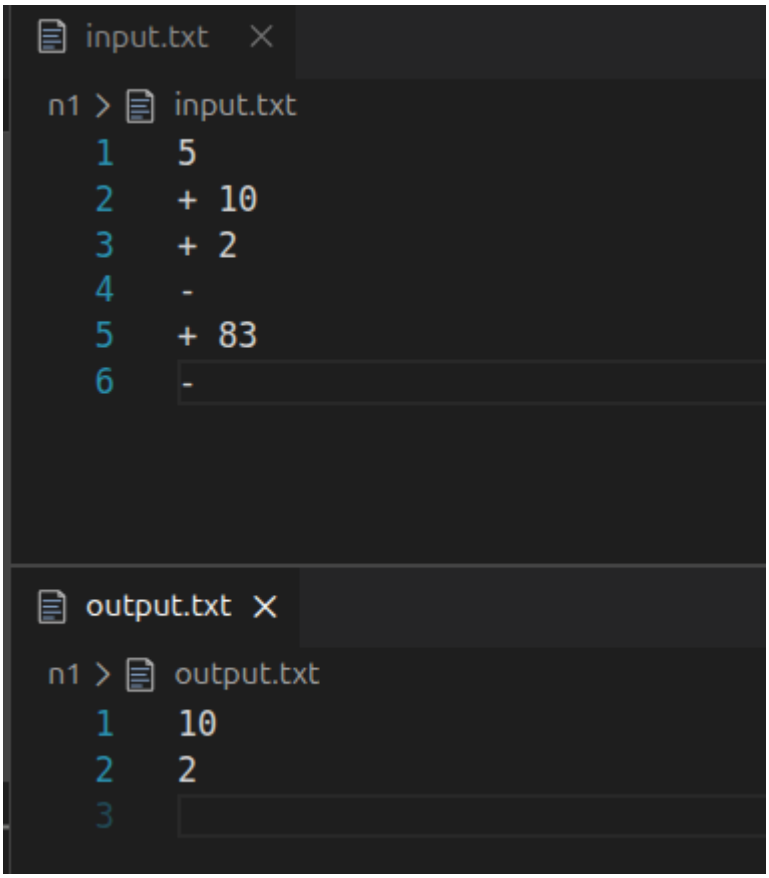
- `+ N` - добавить число N в очередь
- `-` - извлечь элемент и вывести его

Очередь реализована вручную (без библиотек) как кольцевой буфер с индексами `head` и `tail`. Операции добавления и извлечения выполняются за O(1).

```
def main():
    with open("n1/input.txt", "r", encoding="utf-8") as f:
        m = int(f.readline())

        q = [0] * m
        head = 0
        tail = 0
        size = 0

        out_lines = []

        for _ in range(m):
            line = f.readline().strip()

            if line[0] == '+':
                x = int(line[1:].strip())
                q[tail] = x
                tail += 1
                if tail == m:
                    tail = 0
                size += 1
            else:
                # "-"
                x = q[head]
                out_lines.append(str(x) + "\n")
                head += 1
                if head == m:
                    head = 0
```

```
                size -= 1

    with open("n1/output.txt", "w", encoding="utf-8") as out:
        out.write("".join(out_lines))


if __name__ == "__main__":
    main()
```

Проверка на примере (input.txt и output.txt):



Задание 4 - Скобочная последовательность

Дана строка, содержащая символы и скобки типов `()`, `[]`, `{}`. Нужно вывести:

- `Success`, если скобки расставлены правильно
- иначе 1-based индекс первой ошибки
  - в первую очередь ошибка для неправильной закрывающей скобки
  - если неправильных закрывающих нет, то индекс первой открывающей, у которой нет закрывающей

Решение построено на стеке:

- при встрече открывающей скобки кладем в стек пару (скобка, позиция)
- при встрече закрывающей проверяем верх стека
- если стек пуст или тип не совпал - сразу выводим позицию ошибки
- если после прохода стек не пуст - выводим позицию первой оставшейся открывающей

```python
def main():
    with open("n2/input.txt", "r", encoding="utf-8") as f:
        s = f.readline()
        if s is None:
            s = ""
        s = s.rstrip("\n")

    open_set = {'(', '[', '{'}
    match = {')': '(', ']': '[', '}': '{'}

    stack = [] # скобка

    for pos, ch in enumerate(s, start=1):
        if ch in open_set:
            stack.append((ch, pos))
        elif ch in match:
            if not stack:
                with open("т2/output.txt", "w", encoding="utf-8") as out:
                    out.write(str(pos))
                return
            top_ch, top_pos = stack.pop()
            if top_ch != match[ch]:
                with open("n2/output.txt", "w", encoding="utf-8") as out:
                    out.write(str(pos))
                return

    if stack:
        # первая открывающая без закрывающей
        with open("n2/output.txt", "w", encoding="utf-8") as out:
            out.write(str(stack[0][1]))
    else:
        with open("n2/output.txt", "w", encoding="utf-8") as out:
            out.write("Success")


if __name__ == "__main__":
    main()
```

Проверено на нескольких вариантах input-файлов (корректные строки, лишняя закрывающая, неправильный тип закрывающей, лишняя открывающая). Результаты совпадают с ожидаемыми.

Задание 5 - Стек с максимумом

Нужно поддерживать команды:

- `push V`
- `pop`
- `max`

Решение использует два стека:

- основной стек значений
- стек максимумов, где на каждой позиции хранится максимум на текущей глубине

```
def main():
    with open("n3/input.txt", "r", encoding="utf-8") as f,
open("n3/output.txt", "w", encoding="utf-8") as out:
        n_line = f.readline()
        if not n_line:
            return
        n = int(n_line)

        st = []
        mx = []

        buf = []
        FLUSH = 50000

        for _ in range(n):
            line = f.readline().strip()
            if not line:
                continue

            parts = line.split()
            cmd = parts[0]

            if cmd == "push":
                v = int(parts[1])
                st.append(v)
                if not mx:
                    mx.append(v)
                else:
                    mx.append(v if v > mx[-1] else mx[-1])

            elif cmd == "pop":
                st.pop()
                mx.pop()

            else: # "max"
                buf.append(str(mx[-1]) + "\n")
                if len(buf) >= FLUSH:
                    out.write("".join(buf))
                    buf.clear()

        if buf:
            out.write("".join(buf))


if __name__ == "__main__":
    main()
```

Тогда:

- `push` и `pop` работают за O(1)
- `max` тоже за O(1), это просто верх стека максимумов

Задание 10 - Очередь в пекарню

Моделируется работа одного продавца:

- обслуживание одного покупателя занимает 10 минут
- покупатели приходят по времени (в порядке возрастания)
- у каждого покупателя есть степень нетерпения p - сколько людей максимум он готов видеть перед собой в очереди
- если при приходе перед ним людей больше p, он уходит сразу (время ухода равно времени прихода)
- иначе он остается, обслуживается и уходит в момент окончания обслуживания
- если время окончания обслуживания совпало со временем прихода следующего, считаем что обслуживание закончилось раньше, потом пришел следующий

Решение хранит времена окончания обслуживания всех принятых клиентов в очереди и сдвигает указатель, когда обслуживание у кого-то уже закончилось. Это дает работу за линейное время по числу покупателей.

```
SERVICE = 10 # минут на одного

def main():
    with open("n4/input.txt", "r", encoding="utf-8") as f:
        n = int(f.readline())
        customers = []
        for _ in range(n):
            h, m, p = map(int, f.readline().split())
            t = h * 60 + m
            customers.append((t, p))

    end_times = [] # времена окончания обслуживания тех, кто остался
    head = 0

    out_lines = []

    for t, p in customers:
        # убираем всех, кто успел закончить к моменту t
        while head < len(end_times) and end_times[head] <= t:
            head += 1

        if head == len(end_times):
            end_times = []
            head = 0

        in_system = len(end_times) - head

        if in_system > p:
            leave = t
```

```
            else:
                if in_system == 0:
                    start = t
                else:
                    start = end_times[-1]
                leave = start + SERVICE
                end_times.append(leave)

            out_lines.append(f"{leave // 60} {leave % 60}\n")

        with open("n4/output.txt", "w", encoding="utf-8") as out:
            out.write("".join(out_lines))


    if __name__ == "__main__":
        main()
```

Проверено на нескольких вариантах input-файлов, включая случаи:

- очередь пустая
- несколько клиентов уходят из-за нетерпения
- совпадение времени окончания обслуживания и прихода следующего

На всех проверках логика совпала с ожидаемым поведением.