

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное образовательное учреждение высшего образования
САНКТ – ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

старший преподаватель		С. Ю. Гуков
_____ должность, уч. степень, звание	_____ подпись, дата	_____ инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Стек с поддержкой максимума

по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	4321		К.А. Лебедев
		_____ подпись, дата	_____ инициалы, фамилия

Санкт-Петербург, 2024

СОДЕРЖАНИЕ

1 Цель работы.....	3
2 Задание.....	4
3 Ход разработки	5
4 Исходный код программы	6
5 Результаты работы программы	11
6 Вывод.....	12

1 Цель работы

Реализовать стек с поддержкой операций push, pop и max.

2 Задание

Написать программу, для обработки запросов к стеку. Необходимо реализовать программу, которая будет считывать количество команд и обрабатывать заранее установленные команды, выводя очередь ответов на запросы. Дополнительные команды: `remove v`, `avg`.

3 Ход разработки

Были реализованы классы стэка и задачи, написана утилита для прочтения чисел из командной строки на языке TypeScript, интегрирован компилятор в JavaScript, который сразу собирает TypeScript проект без предварительной компиляции.

4 Исходный код программы

```
// index.ts

import Task from "../lib/Task";

new Task(); // инициализация задачи

// utils/readline.ts

import readline from "node:readline";

// readline интерфейс

export const readlineInterface = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

// interfaces/interfaces.ts

// Дженерик стэка

export interface IStack<T> {
  value: T[];
  push: (item: T) => void;
  pop: () => T | undefined;
  max: () => number | undefined;
  min: () => number | undefined;
  remove: (item: T) => void;
}

// config/index.ts

// enum с операциями

export enum Operations {
  MAX = "max",
  MIN = "min",
  AVG = "avg",
  PUSH = "push",
  POP = "pop",
  REMOVE = "remove",
}
```

```

// lib/Stack.ts

import { IStack } from "../interfaces/interfaces";

// Класс стэка для работы с массивами дженерного типа

class Stack<T> implements IStack<T> {
    public value: T[]; // значение стэка
    private isNumberStackFlag: boolean; // флаг того, является ли массив
    ЧИСЛОВЫМ

    constructor(arr: T[]) {
        this.value = arr; // создание массива
        this.isNumberStackFlag = true; // флаг того, является ли массив
    ЧИСЛОВЫМ
        this.value.forEach(
            item => (this.isNumberStackFlag && typeof item === "number")
        );
    }

    // push в стэк

    public push(item: T) {
        this.value.push(item);
    }

    // pop из стэка

    public pop() {
        return this.value.pop();
    }

    // максимальное значение стэка

    public max() {
        return this.isNumberStackFlag
            ? Math.max(...(this.value as number[]))
            : undefined;
    }

    // минимальное значение стэка

    public min() {
        return this.isNumberStackFlag

```

```

        ? Math.min(...(this.value as number[]))
        : undefined;
    }

    // удаление значения из массива по value

    public remove(item: T) {
        const index = this.value.indexOf(item);
        if (index !== -1) {
            this.value.splice(index, 1);
        }
    }
}

export default Stack;

// lib/Task.ts

import { Operations } from "../config";
import { readlineInterface } from "../utils/readline";
import Stack from "./Stack";

class Task {
    private res: Stack<number>;
    private resCountCommands: number;
    private sum: number;
    private callbackQueue: (() => void)[];
    private countCommands: number;

    // перечисление свойств

    constructor() {
        this.init(); // инициализация
        this.res = new Stack<number>([]); // создание стека
        this.resCountCommands = 0; // итоговое количество команд
        this.sum = 0; // сумма
        this.callbackQueue = []; // очередь запросов
        this.countCommands = 0; // текущее количество команд
    }

    // метод прочтения команды

    readElement() {
        if (this.countCommands < this.resCountCommands) {

```



```

        readlineInterface.question("", element => {
            this.parse(element); // парсинг команды
            this.countCommands++; // увеличение количества команд
            this.readElement(); // рекурсия
        });
    } else {
        while (this.callbackQueue.length > 0) {
            (this.callbackQueue.shift() as () => void)(); // очистка
очереди команд
        }
        readlineInterface.close(); // закрытие интерфейса readline
    }
}

// парсинг команды

parse(command: string) {
    const [operation, value] = command.split(" "); // деструктуризация
команды

    switch (operation) {
        case Operations.PUSH: // push в стэк с учетом суммы
            this.res.push(Number(value));
            this.sum += Number(value);
            break;
        case Operations.POP: // pop из стэка с учетом суммы
            this.sum -= Number(this.res.pop());
            break;
        case Operations.MIN: // минимальное число в стэке с записью в
очередь запросов
            const min = this.res.min();
            this.callbackQueue.push(() => console.log(min));
            break;
        case Operations.MAX: // максимальное число в стэке с записью в
очередь запросов
            const max = this.res.max();
            this.callbackQueue.push(() => console.log(max));
            break;
        case Operations.AVG: // среднее число в стэке с записью в
очередь запросов, сложность O(N)
            this.callbackQueue.push(() =>
                console.log(this.sum / this.res.value.length)
            );
            break;
        case Operations.REMOVE: // удаления числа из стэка

```

```

        this.res.remove(Number(value));
        break;
      default:
        break;
    }
  }

  // инициализация задачи

  init() {
    readlineInterface.question("", (length: string) => {
      this.resCountCommands = parseInt(length); // чтение количества
команд

      if (isNaN(this.resCountCommands) || this.resCountCommands <=
0) {

        console.log(
          "Пожалуйста, введите корректное положительное число."
        );
        readlineInterface.close();
        return;
      }

      this.readElement(); // вызов рекурсии
    });
  }
}

export default Task;

```

5 Результаты работы программы

Результат работы программы включает в себя вывод очереди команд

```
5  
push 1  
push 2  
max  
pop  
max  
2  
1
```

Рисунок 1 – тест программы №1

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

Рисунок 2 – тест программы №2

6 Вывод

Стек — это самодостаточная структура данных для решения задач в программировании, основанная на принципе LIFO (Last In, First Out), что означает, что последний добавленный элемент будет первым, который будет извлечен. Эта структура данных имеет множество применений как в вычислительной технике, так и в других областях информационных технологий.

В результате выполнения лабораторной работы был изучен принцип работы стека (LIFO), написаны утилитарные функции для обработки входящих команд.