

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«Санкт-Петербургский государственный морской технический университет»  
Факультет цифровых промышленных технологий  
Кафедра Киберфизических систем

**Лабораторная работа**

На тему: Алгоритм подбора знаков в выражении

Выполнил

Студент 3 курса 20321 группа

Маршалов М.С.

Преподаватель

Минин М. С.

Г. Санкт-Петербург

2024

## **Цель работы**

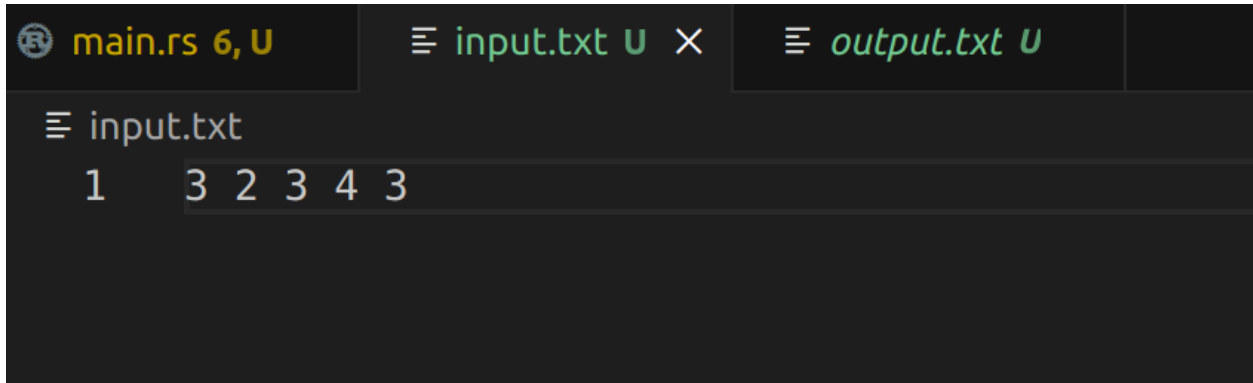
При работе над данной лабораторной работой будут затронуты следующие вопросы:

1. Работа с файлами.
2. Работа с несколькими функциями, а не только с `main`.
3. Реализации рекурсивной функции.

## Ход работы

Для поиска решения используется стандартный алгоритм "разделяй и властвуй" и метод подбора. В функции `find_expression`, принимающей `numbers` вектор чисел из которых нужно составить выражение, `target` значение к которому должно прийти выражение, `index` указывающее на индекс в обрабатываемого числа в массиве `numbers`, `current_sum` текущая сумма, `result` массив хранящий уже подставленные знаки. В начале функции проходит проверка на базовый случай, а точнее случай когда это последний элемент. Если элемент не последний мы вызываем эту же функцию рекурсивно, пытаюсь подставить `+`. Пройдя по всем значениям и вставляю знак сложения, мы можем получить правильный результат и вывести его, а так же можем получить не правильный результат, тогда мы вернёмся на один вызов обратно, и вместо последнего знака сложения, подставим знак вычитания. И так пока мы не дойдём до правильного результата. Теперь рассмотрим функцию `main`. Вначале мы сохраняем аргумент переданный в командной строке. Далее мы извлекая путь к файлу, считываем его по заданному пути. Считав файл, мы разбиваем его на части, где `N` это количество чисел, `numbers` массив с числами, `S` это результат который должен получиться. Создаём массив `result`, который хранит расставленные знаки. Далее мы подставляем результат в строку `output`, чередуя со знаком. В конце мы создаём текстовый файл `output.txt` с результатом

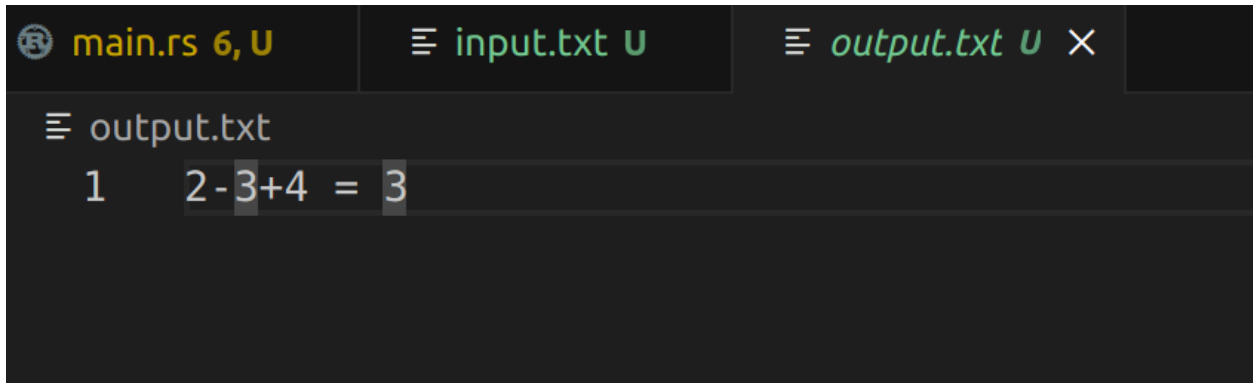
## Результат работы



The screenshot shows a code editor with three tabs: 'main.rs 6, U', 'input.txt U', and 'output.txt U'. The 'input.txt' tab is active, displaying the text '1 3 2 3 4 3' on the first line.

```
input.txt
1 3 2 3 4 3
```

Рисунок 1 – Текстовый файл с входными данными.



The screenshot shows the same code editor with the 'output.txt' tab active. It displays the text '1 2-3+4 = 3' on the first line, where the numbers 2, 3, and 4 are highlighted with a light gray background.

```
output.txt
1 2-3+4 = 3
```

Рисунок 2 – Созданный текстовый файл с результатом.

## **Вывод:**

В процессе выполнения данной лабораторной работы мы получили ценный опыт в трёх ключевых областях программирования:

Работа с файлами: Вы научитесь читать из файлов и записывать в файлы, что является фундаментальным навыком для многих программных приложений.

Использование нескольких функций: Разработка программы с использованием не только главной функции (main), но и дополнительных функций, улучшит структурирование кода и его читаемость.

Реализация рекурсивной функции: Вы разработаете рекурсивную функцию, что поможет вам глубже понять принципы работы стека вызовов и рекурсии в программировании.

## Листинг

```
use std::fs;

fn main() {

    let file_path = &String::from("input.txt");

    let contents = fs::read_to_string(file_path).expect("Не удалось
прочитать файл");

    let mut parts = contents.split_whitespace();

    let N: usize = parts.next().expect("Ожидалось значение
N").parse().expect("Ошибка парсинга N");

    let mut numbers: Vec<i64> = Vec::with_capacity(N);

    for _ in 0..N {

        let num: i64 = parts.next().expect("Ожидалось
число").parse().expect("Ошибка парсинга числа");

        numbers.push(num);

    }

    let S: i64 = parts.next().expect("Ожидалось значение
S").parse().expect("Ошибка парсинга S");

    let mut result = vec![' '; N - 1];

    if find_expression(&numbers, S, 0, 0, &mut result) {
```

```

    let mut output = String::new();

    output.push_str(&format!("{}", numbers[0]));

    for i in 1..N {

        output.push(result[i - 1]);

        output.push_str(&format!("{}", numbers[i]));

    }

    output.push_str(&format!(" = {}", S));

    fs::write("output.txt", output).expect("Не удалось записать в
файл");

    } else {

        fs::write("output.txt", "no solution").expect("Не удалось записать
в файл");

    }

}

fn find_expression(

    numbers: &[i64],

    S: i64,

    index: usize,

    current_sum: i64,

    result: &mut Vec<char>,

) -> bool {

    if index == numbers.len() - 1 {

        if current_sum + numbers[index] == S {

            if index > 0 {

```

```

        result[index - 1] = '+';

    }

    true

} else if current_sum - numbers[index] == S {

    if index > 0 {

        result[index - 1] = '-';

    }

    true

} else {

    false

}

} else {

    if find_expression(numbers, S, index + 1, current_sum +
numbers[index], result) {

        if index > 0 {

            result[index - 1] = '+';

        }

        return true;

    }

    if find_expression(numbers, S, index + 1, current_sum -
numbers[index], result) {

        if index > 0 {

            result[index - 1] = '-';

        }

    }

}

```



```
        return true;

    }

    false

}

}
```