ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент		В. А. Кузнецов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 2.2

АНАЛИЗ СТРОК, ХЕШИРОВАНИЕ

по курсу:

ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи

Задача: реализовать программную функцию на языке C/C++, выполняющую поставленную задачу. Вариант задания, пример входных и выходных данных представлен в таблице 1. Глобальные параметры использовать запрещено; допустимо использование дополнительных функций.

Таблица 1 – Вариант

N	Текст задания	Вход	Выход
6	**Антиплагиат** Даны две строки S1, S2. Разделителем между словами в строке может являться пробел « », запятая «,» или перенос строки «\n». Найти все повторяющиеся в обеих строках слова без учета регистра символов (“Антиплагиат” = “АнТиплагиаТ”). Для каждого повторяющегося слова определить все индексы его вхождения в строку S1 и строку S2.	S1 : “Get now to get first” S2 : “I will get back to you”	“get” : [0, 11], [7]

2 Схема алгоритма решения

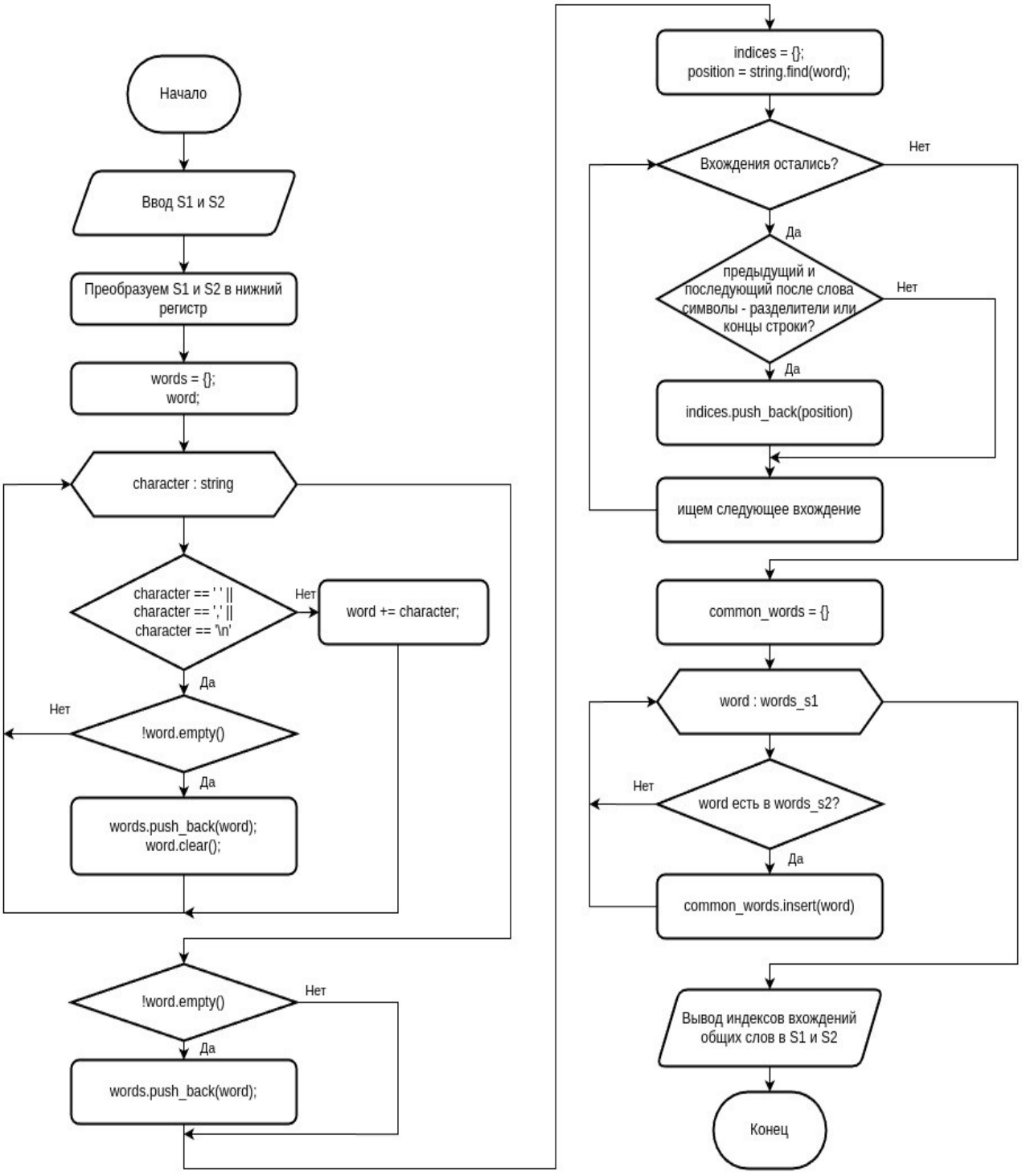

Рисунок 1 – Блок-схема алгоритма

3 Полное описание реализованных функций

3.1 split_words

Функция split_words разделяет строку на слова, используя пробелы, запятые и переносы строк в качестве разделителей. Принимает следующие аргументы:

1. const std::string &string — строка, которую нужно разделить на слова.

Возвращает std::vector<std::string> — вектор строк, каждая из которых является словом из исходной строки. Работа функции происходит следующим образом:

1. Создаётся пустой вектор words и временная строка word.
2. Для каждого символа в строке проверяется, является ли он разделителем (пробел, запятая, перенос строки).
3. Если символ является разделителем и word не пуст, добавляем word в words и очищаем word.
4. Если символ не является разделителем, добавляем его к word.
5. В конце добавляем последнее слово в words, если оно не пустое.

3.2 find_indices

Функция find_indices находит все индексы вхождения слова в строку. Принимает следующие аргументы:

1. const std::string &string — строка, в которой ищем вхождения слова.
2. const std::string &word — слово, индексы вхождения которого ищем.

Возвращает std::vector<int> — вектор индексов, где слово встречается в строке. Работа функции происходит следующим образом

1. Создаётся пустой вектор indices.
2. Используем string.find для поиска первого вхождения слова в строке.
3. Пока находятся вхождения, проверяем, является ли предыдущий и последующий после слова символы разделителями или концом строки.
4. Если условие выполняется, добавляем индекс начала слова в indices.
5. Ищем следующее вхождение, начиная с позиции position + 1.

3.3 transform_to_lower

Функция transform_to_lower преобразует все символы строки в нижний регистр. Принимает следующие аргументы:

1. const std::string &string — строка, которую нужно преобразовать.

Возвращает std::string — строка, преобразованная в нижний регистр. Работа функции происходит следующим образом:

1. Копируем исходную строку в новую строку lower.
2. Используем std::transform для преобразования каждого символа строки в нижний регистр с помощью функции ::tolower.

3.4 main

1. Программа запрашивает ввод строк S1 и S2 от пользователя.
2. Преобразуем S1 и S2 в нижний регистр с помощью transform_to_lower.
3. Разделяем S1 и S2 на слова с помощью функции split_words.
4. Для каждого слова в S1 и S2 находим индексы вхождений в исходные строки, используя функцию find_indices.
5. Находим слова, которые есть в обеих строках, используя unordered_set.
6. Для каждого общего слова выводим индексы их вхождений в S1 и S2.

4 Листинг программы

Листинг 1

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <algorithm>

std::vector<std::string> split_words(const std::string &string) {
    std::vector<std::string> words;
    std::string word;
    for (char character : string) {
        if (character == ' ' || character == ',' || character == '\n') {
            if (!word.empty()) {
                words.push_back(word);
                word.clear();
            }
        } else {
            word += character;
        }
    }
    if (!word.empty()) { // Добавляем последнее слово
        words.push_back(word);
    }
    return words;
}

std::vector<int> find_indices(const std::string &string, const std::string
&word) {
    std::vector<int> indices;
    size_t position = string.find(word); // Находим первое вхождение
    while (position != std::string::npos) {
        char start_character = string[position - 1];
        char end_character = string[position + word.size()];
        // Проверяем начало и окончание слова
        if (end_character == ' ' || end_character == ',' || end_character ==
'\n' || end_character == '\0')
            if (start_character == ' ' || start_character == ',' ||
start_character == '\n' || start_character == '\0')
                indices.push_back((int) position); // Добавляем индекс в
вектор
        position = string.find(word, position + 1); // Ищем следующее
вхождение, начиная с позиции position + 1
    }
    return indices;
}

std::string transform_to_lower(const std::string &string) {
    std::string lower = string;
    std::transform(lower.begin(), lower.end(), lower.begin(), ::tolower);
    return lower;
}

int main() {
    std::string S1;
    std::string S2;
```

```
    std::cout << "S1: ";
    std::getline(std::cin >> std::ws, S1);

    std::cout << "S2: ";
    std::getline(std::cin >> std::ws, S2);

    std::string lower_s1 = transform_to_lower(S1);
    std::string lower_s2 = transform_to_lower(S2);

    std::unordered_map<std::string, std::vector<int>> word_indices_s1;
    std::unordered_map<std::string, std::vector<int>> word_indices_s2;

    std::vector<std::string> words_s1 = split_words(S1);
    std::vector<std::string> words_s2 = split_words(S2);

    // Находим индексы
    for (const std::string &word : words_s1) {
        std::string lower_word = transform_to_lower(word);
        word_indices_s1[lower_word] = find_indices(lower_s1, lower_word);
    }
    for (const std::string &word : words_s2) {
        std::string lower_word = transform_to_lower(word);
        word_indices_s2[lower_word] = find_indices(lower_s2, lower_word);
    }

    // Находим общие слова
    std::unordered_set<std::string> common_words;
    for (const auto &pair : word_indices_s1) {
        if (word_indices_s2.find(pair.first) != word_indices_s2.end()) {
            common_words.insert(pair.first);
        }
    }

    // Выводим индексы вхождений
    for (const std::string &word : common_words) {
        std::vector<int> current_indices_s1 = word_indices_s1[word];
        std::vector<int> current_indices_s2 = word_indices_s2[word];

        std::cout << "\"" << word << "\": ";
        std::cout << "[";
        for (size_t i = 0; i < current_indices_s1.size(); i++) {
            std::cout << current_indices_s1[i];
            if (i < current_indices_s1.size() - 1) std::cout << ", ";
        }
        std::cout << "], [";
        for (size_t i = 0; i < current_indices_s2.size(); i++) {
            std::cout << current_indices_s2[i];
            if (i < current_indices_s2.size() - 1) std::cout << ", ";
        }
        std::cout << "]" << std::endl;
    }

    return 0;
}
```

5 Результаты тестирования программы

```
S1: Get now to get first
S2: I will get back to you
"get": [0, 11], [7]
"to": [8], [16]

Process finished with exit code 0
```

Рисунок 2

```
S1: fInD ThESe wORDS iF yOU cAn
S2: CaN yoU HeLP Me OuT and FIND
"find": [0], [24]
"you": [20], [4]
"can": [24], [0]

Process finished with exit code 0
```

Рисунок 3

```
S1: test,testing,tested 123 abc789
S2: 789 abc,test
"test": [0], [8]

Process finished with exit code 0
```

Рисунок 4

```
S1: lorem ipsum,dolor sit,amet
S2: abcdefg qwerty

Process finished with exit code 0
```

Рисунок 5

```
S1: In ipsa modi et molestiae ipsa quae galisum et ipsa mollitia aut eveniet modi
S2: Modi et eveniet ipsa quae aut et ipsa mollitia aut eveniet modi
"et": [13, 44], [5, 30]
"eveniet": [65], [8, 51]
"ipsa": [3, 26, 47], [16, 33]
"modi": [8, 73], [0, 59]
"quae": [31], [21]
"aut": [61], [26, 47]
"mollitia": [52], [38]

Process finished with exit code 0
```

Рисунок 6