МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра МОиСИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6.2

«Поиск образца в тексте»
по дисциплине
«Структуры и алгоритмы обработки данных**»**

Выполнил студент группы ИКБО-10-24 Таганов А.А.

Практическую работу выполнил «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

Цель работы: освоить приёмы реализации алгоритмов поиска образца в тексте.

Задание 1: дан текст, состоящий из слов, разделенных знаками препинания. Сформировать массив из слов, которые содержат заданную подстроку.

Для выполнения 1 задания была реализована функция find_words_with_substring. Сначала создается вектор слов, которые будут содержать подстроку и пустая строка, для хранения слов. Мы проходимся по каждой букве в тексте, и проверяем, чтобы она не была знаком препинания или пробелом. Если проверка проходит, то мы добавляем букву в строку. Когда встречается пробел или знак препинания, то мы проверяем, содержит ли слово подстроку, и если содержит, то добавляем его в вектор слов и очищаем строку для хранения слов. В конце мы с помощью цикла возвращаем все слова, содержащие подстроку.

Листинг 1 – Функция поиска слов с подстрокой

```
void find_words_with_substring(string text, string substring) {
    vector<string> words;
    string word;

    for (char c : text) {
        if (!ispunct(c) && c != ' ') {
            word += c;
        } else if (!word.empty()) {
            if (word.find(substring) != string::npos) {
                words.push_back(word);
            }
            word.clear();
        }
    }
    if (!word.empty() && word.find(substring) != string::npos) {
        words.push_back(word);
    }

    cout << "Words with substring '" << substring << "': ";
    for (const auto& w : words) {
        cout << w << " ";
    }
    if (words.empty()) cout << "not found";
    cout << endl;
}
```

На рисунках 1, 2 представлены результаты работы программы.

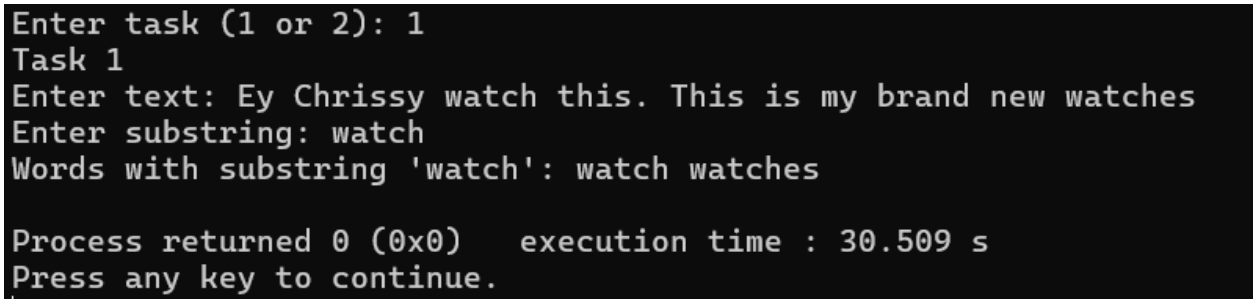

Рис.1 – Успешный поиск

Рис.2 – Безуспешный поиск

Задание 2: назовём строку палиндромом, если она одинаково читается слева направо и справа налево. Примеры палиндромов: "abcba", "55", "q", "xyzzyx". Требуется для заданной строки найти максимальную по длине ее подстроку, являющуюся палиндромом. Реализация алгоритмом Бойера-Мура-Хорспула.

Алгоритм выполняет сравнение шаблона с текстом справа налево, начиная с последнего символа шаблона. При обнаружении несовпадения символов алгоритм не сдвигает шаблон на одну позицию, а вычисляет максимально возможный сдвиг на основе таблицы плохих символов. Перед началом поиска строится таблица сдвигов для каждого символа, которая определяет, на какое расстояние можно переместить шаблон при несовпадении: для символов, присутствующих в шаблоне (кроме последнего), сдвиг равен расстоянию от конца шаблона до последнего вхождения этого символов, для символов, отсутствующих в шаблоне, сдвиг равен полной длине шаблона

Для выполнения этого задания был реализован класс BoyerMooreHorspool. В нем мы создаем хеш-таблицу плохих символов при помощи функции build_table. Сам алгоритм реализован в функции search. В классе есть функция, проверяющая, является ли строка палиндромом, и использующая search.

Листинг 2 – Реализация алгоритма

```
class BoyerMooreHorspool {
private:
    unordered_map<char, int> bad_char_table;
    string pattern;

    void build_table(const string& pat) {
        pattern = pat;
        int m = pattern.length();
        for (int i = 0; i < m - 1; i++) {
            bad_char_table[pattern[i]] = m - 1 - i;
        }
    }

public:
    int search(const string& text) {
        build_table(pattern);
        int n = text.length();
        int m = pattern.length();

        if (m == 0) return 0;
        if (n < m) return -1;

        int i = m - 1;
        while (i < n) {
            int j = m - 1;
            int k = i;

            while (j >= 0 && text[k] == pattern[j]) {
                j--;
                k--;
            }

            if (j < 0) return i - m + 1;

            char bad_char = text[k];
            i += (bad_char_table.find(bad_char) != bad_char_table.end()) ?
                 bad_char_table[bad_char] : m;
        }

        return -1;
    }

    bool is_palindrome(const string& s) {
        string reversed = s;
        reverse(reversed.begin(), reversed.end());
        build_table(s);
        return search(reversed) == 0;
    }
};
```

Сам же поиск максимального палиндрома реализован в функции find_max_palindrome. Алгоритм последовательно перебирает все возможные подстроки, начиная с длины равной исходной строке и уменьшая её на каждом шаге. При обнаружении палиндрома текущей длины поиск немедленно завершается, так как гарантированно найден максимальный палиндром. Это обеспечивает значительное ускорение по сравнению с полным перебором всех подстрок. Для проверки палиндромности применяется модифицированный алгоритм Бойера-Мура-Хорспула, где исходная строка сравнивается со своей обратной версией. Если поиск обнаруживает совпадение с начала обратной строки, это свидетельствует о палиндромности проверяемой подстроки. Такой подход сочетает эффективность алгоритма BMH с гарантированным нахождением оптимального решения.

Листинг 3 – Поиск максимального палиндрома

```
string find_max_palindrome(const string& s) {
    BoyerMooreHorspool bmh;
    string max_palindrome = "";

    for (int length = s.length(); length >= 1; length--) {
        for (int i = 0; i <= s.length() - length; i++) {
            string substr = s.substr(i, length);

            if (bmh.is_palindrome(substr)) {
                if (substr.length() > max_palindrome.length()) {
                    max_palindrome = substr;
                }

                if (max_palindrome.length() == s.length()) {
                    return max_palindrome;
                }
            }
        }

        if (!max_palindrome.empty()) {
            return max_palindrome;
        }
    }

    return string(1, s[0]);
}
```

На рисунках 3, 4 представлены результаты работы программы.

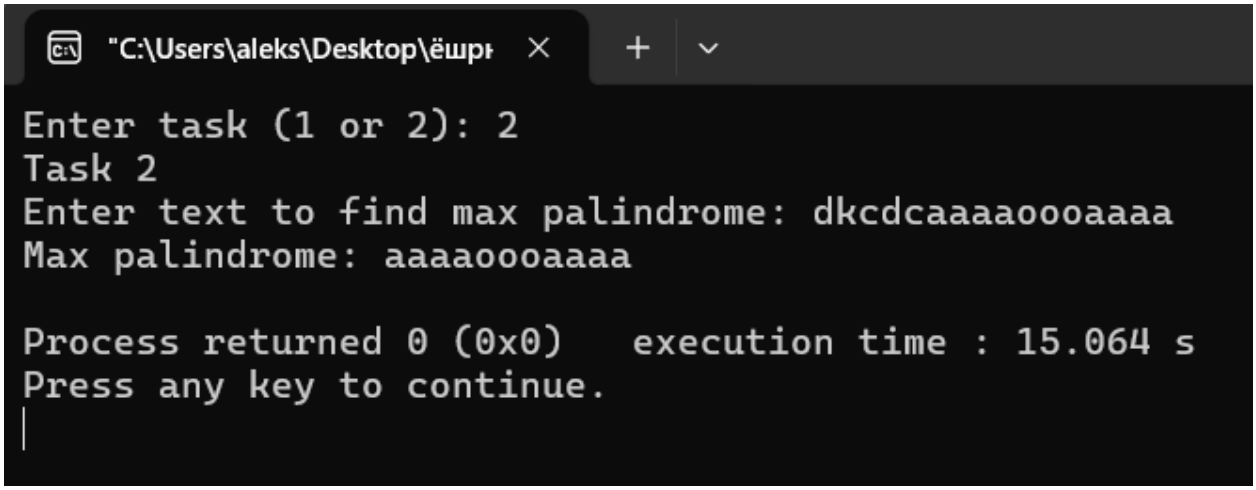

Рис.3 – Успешный поиск

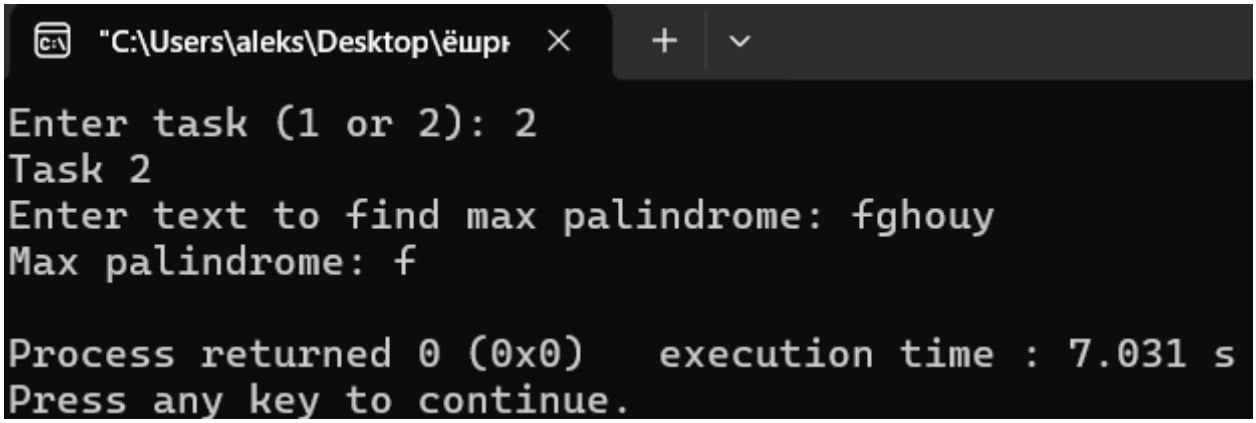

Рис.4 – Безуспешный поиск

Вывод

Поставленные задачи выполнены: реализованы алгоритмы поиска вхождений образа в строку с помощью простого поиска и метода Бойера-Мура-Хорспула. Создана функция поиска максимального палиндрома при помощи алгоритма Бойера-Мура-Хорспула.

Литература.

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 18.10.2025).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс].URL:https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 18.10.2025).