Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

Лабораторная работа  
«Методы Поиска»

Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

**Бинарный поиск**

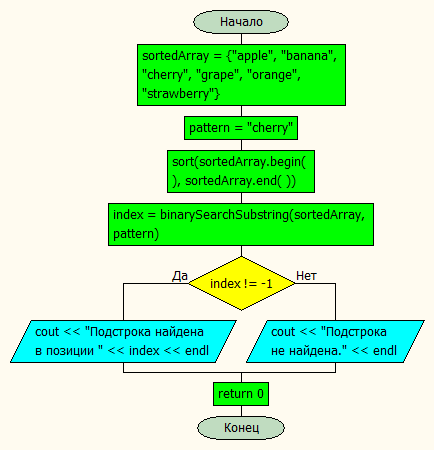
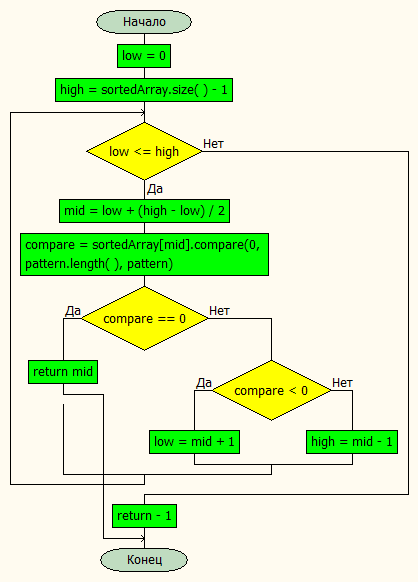
**Постановка задачи:**

Найти подстроку в строке используя следующие поиски: «бинарный», «линейный» и «интерполяционный».

**Анализ задачи:**

* **Описание метода:** Бинарный поиск — алгоритм поиска элемента в отсортированном массиве путем деления массива пополам и последующего сравнения искомого элемента с элементом в середине массива.
* **Подходящий тип массива:** Для бинарного поиска массив должен быть отсортирован в порядке возрастания или убывания.
* **Содержание массива:** Бинарный поиск подходит для любых типов данных, которые можно сравнивать между собой.
* **Преимущества:** Эффективность (временная сложность O(log n)) и простота реализации.
* **Ограничения и осложнения:** Требуется отсортированный массив; может потребоваться дополнительная память для хранения массива.

**Блок-схема:**



**Код на языке с++:**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

int binarySearchSubstring(const vector<string>& sortedArray, const string& pattern) {

int low = 0;

int high = sortedArray.size() - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

int compare = sortedArray[mid].compare(0, pattern.length(), pattern);

if (compare == 0) // подстрока найдена

return mid;

else if (compare < 0) // текущая строка меньше подстроки, идем вправо

low = mid + 1;

else // текущая строка больше подстроки, идем влево

high = mid - 1;

}

return -1; // подстрока не найдена

}

int main() {

vector<string> sortedArray = { "apple", "banana", "cherry", "grape", "orange", "strawberry" };

string pattern = "cherry";

sort(sortedArray.begin(), sortedArray.end()); // сортируем массив строк

int index = binarySearchSubstring(sortedArray, pattern);

if (index != -1) {

cout << "Подстрока найдена в позиции " << index << endl;

}

else {

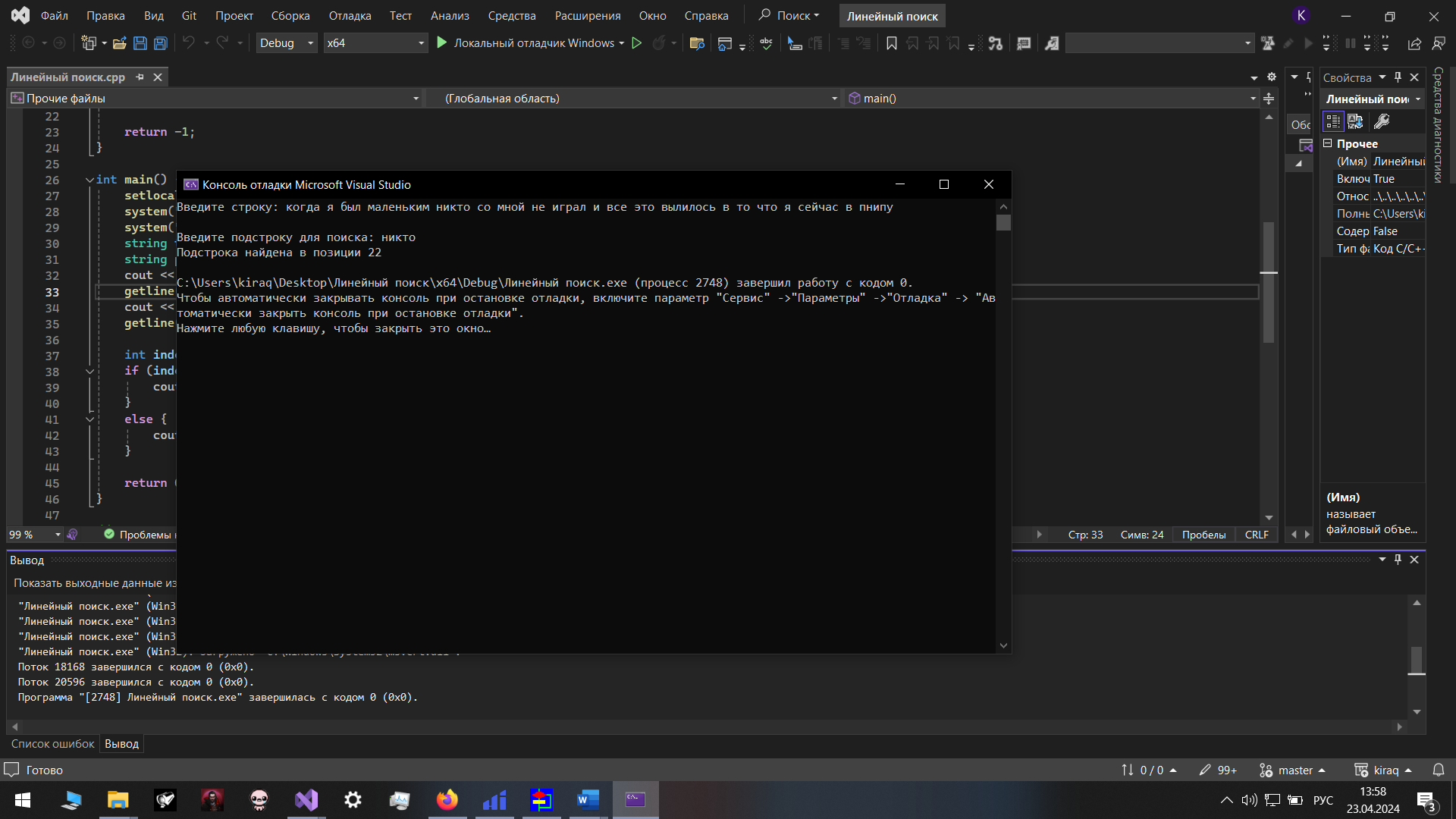
cout << "Подстрока не найдена." << endl;

}

return 0;

}

**Пример работы кода:**



**Вывод:**

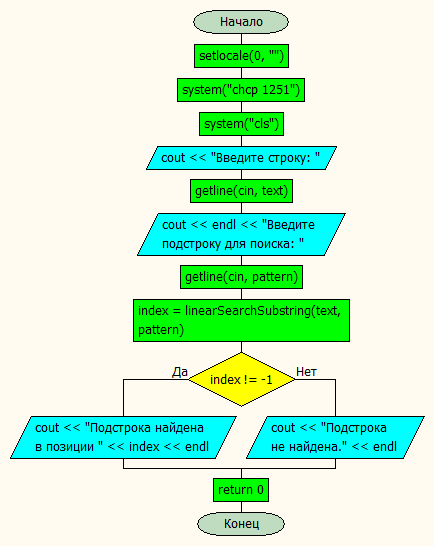
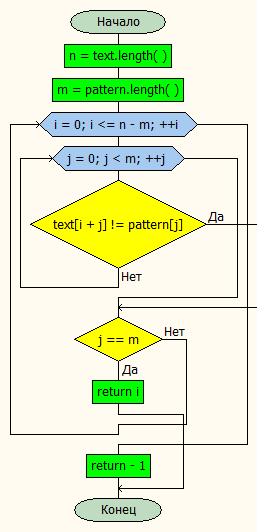
Код отработал как задумано, алгоритм работает исправно

**Линейный поиск**

**Анализ задачи:**

1. **Описание метода:** Линейный поиск — простой алгоритм поиска элемента в массиве путем последовательного просмотра каждого элемента массива до тех пор, пока не будет найден искомый элемент или не будет достигнут конец массива.
2. **Подходящий тип массива:** Для линейного поиска не требуется специальный тип массива. Массив может быть неупорядоченным или упорядоченным.
3. **Содержание массива:** Линейный поиск подходит для любых типов данных, которые можно сравнивать между собой.
4. **Преимущества:** Простота реализации и невысокие требования к массиву. Не требует предварительной сортировки массива.
5. **Ограничения и осложнения:** Линейный поиск имеет временную сложность O(n), что делает его менее эффективным на больших массивах по сравнению с более сложными методами, такими как бинарный поиск.

**Блок-схема:**

****

**Код на языке с++:**

#include <iostream>

#include <string>

using namespace std;

int linearSearchSubstring(const string& text, const string& pattern) {

int n = text.length();

int m = pattern.length();

for (int i = 0; i <= n - m; ++i) {

int j;

for (j = 0; j < m; ++j) {

if (text[i + j] != pattern[j])

break;

}

if (j == m)

return i;

}

return -1;

}

int main() {

setlocale(0, "");

system("chcp 1251");

system("cls");

string text;

string pattern;

cout << "Введите строку: ";

getline(cin, text);

cout << endl << "Введите подстроку для поиска: ";

getline(cin, pattern);

int index = linearSearchSubstring(text, pattern);

if (index != -1) {

cout << "Подстрока найдена в позиции " << index << endl;

}

else {

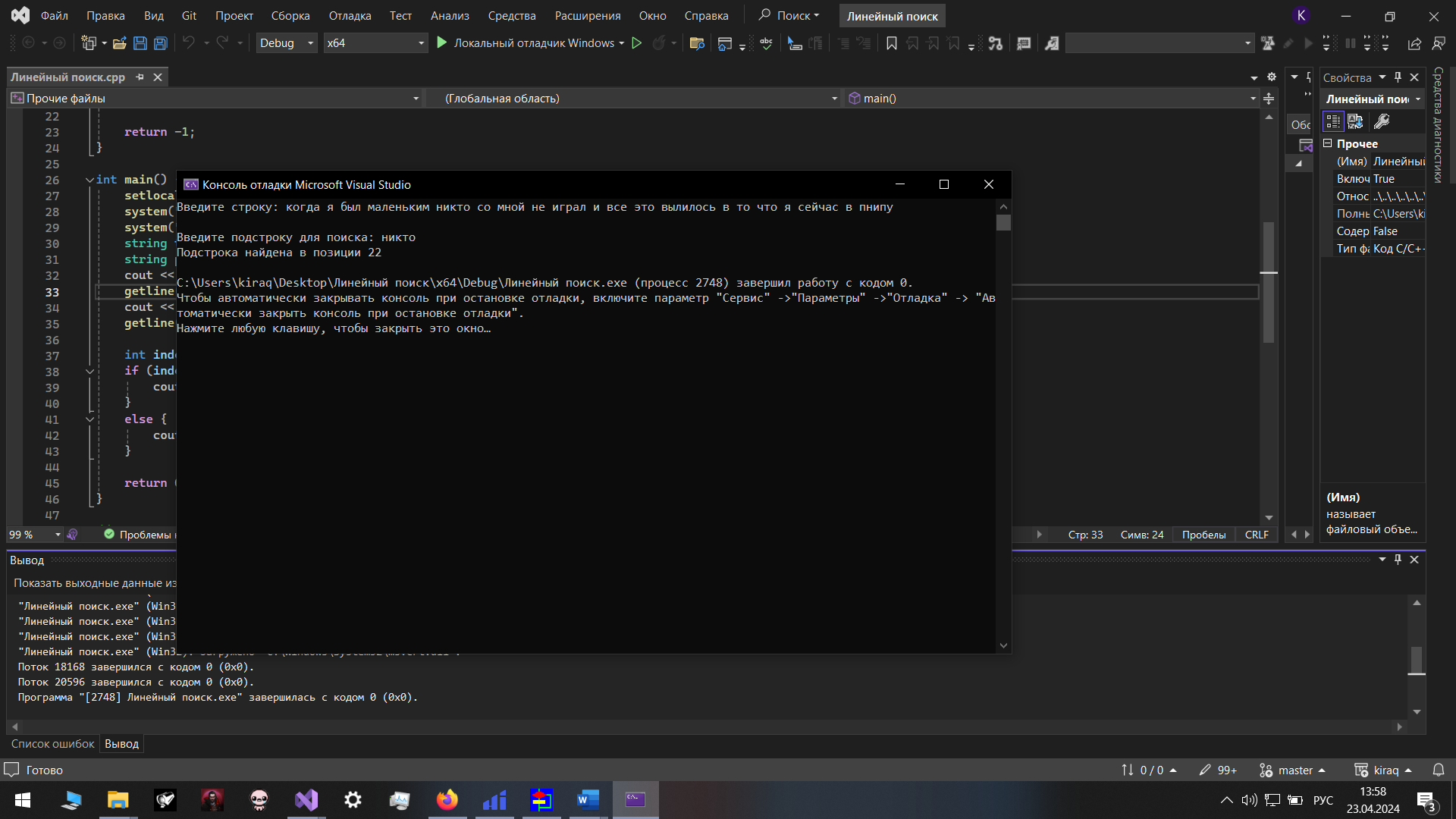
cout << "Подстрока не найдена." << endl;

}

return 0;

}

**Пример работы кода:**



**Вывод:**

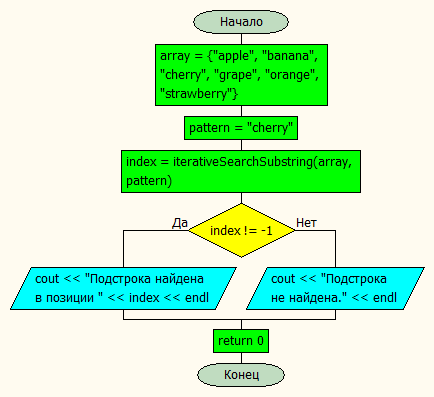
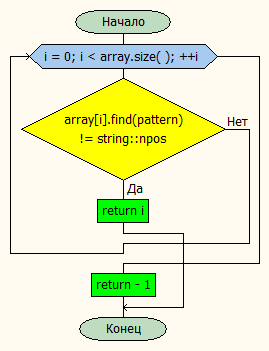
Код отработал как задумано, алгоритм работает исправно

**Интерполяционный поиск**

**Анализ задачи:**

1. **Описание метода:** Интерполяционный поиск — это алгоритм поиска элемента в упорядоченном массиве, который использует формулу интерполяции для приблизительного определения местоположения искомого элемента.
2. **Подходящий тип массива:** Для интерполяционного поиска требуется упорядоченный массив, который может быть отсортирован как в порядке возрастания, так и в порядке убывания.
3. **Содержание массива:** Интерполяционный поиск подходит для любых типов данных, которые можно сравнивать между собой.
4. **Преимущества:** Более эффективен, чем линейный поиск, особенно на больших отсортированных массивах с равномерно распределенными значениями.
5. **Ограничения и осложнения:** Интерполяционный поиск требует равномерно распределенных значений в массиве для достижения оптимальной производительности. На неравномерно распределенных данных его эффективность может снижаться до O(n).

**Блок-схема:**

****

**Код на языке с++:**

#include <iostream>

#include <string>

using namespace std;

int linearSearchSubstring(const string& text, const string& pattern) {

int n = text.length();

int m = pattern.length();

for (int i = 0; i <= n - m; ++i) {

int j;

for (j = 0; j < m; ++j) {

if (text[i + j] != pattern[j])

break;

}

if (j == m)

return i;

}

return -1;

}

int main() {

setlocale(0, "");

system("chcp 1251");

system("cls");

string text;

string pattern;

cout << "Введите строку: ";

getline(cin, text);

cout << endl << "Введите подстроку для поиска: ";

getline(cin, pattern);

int index = linearSearchSubstring(text, pattern);

if (index != -1) {

cout << "Подстрока найдена в позиции " << index << endl;

}

else {

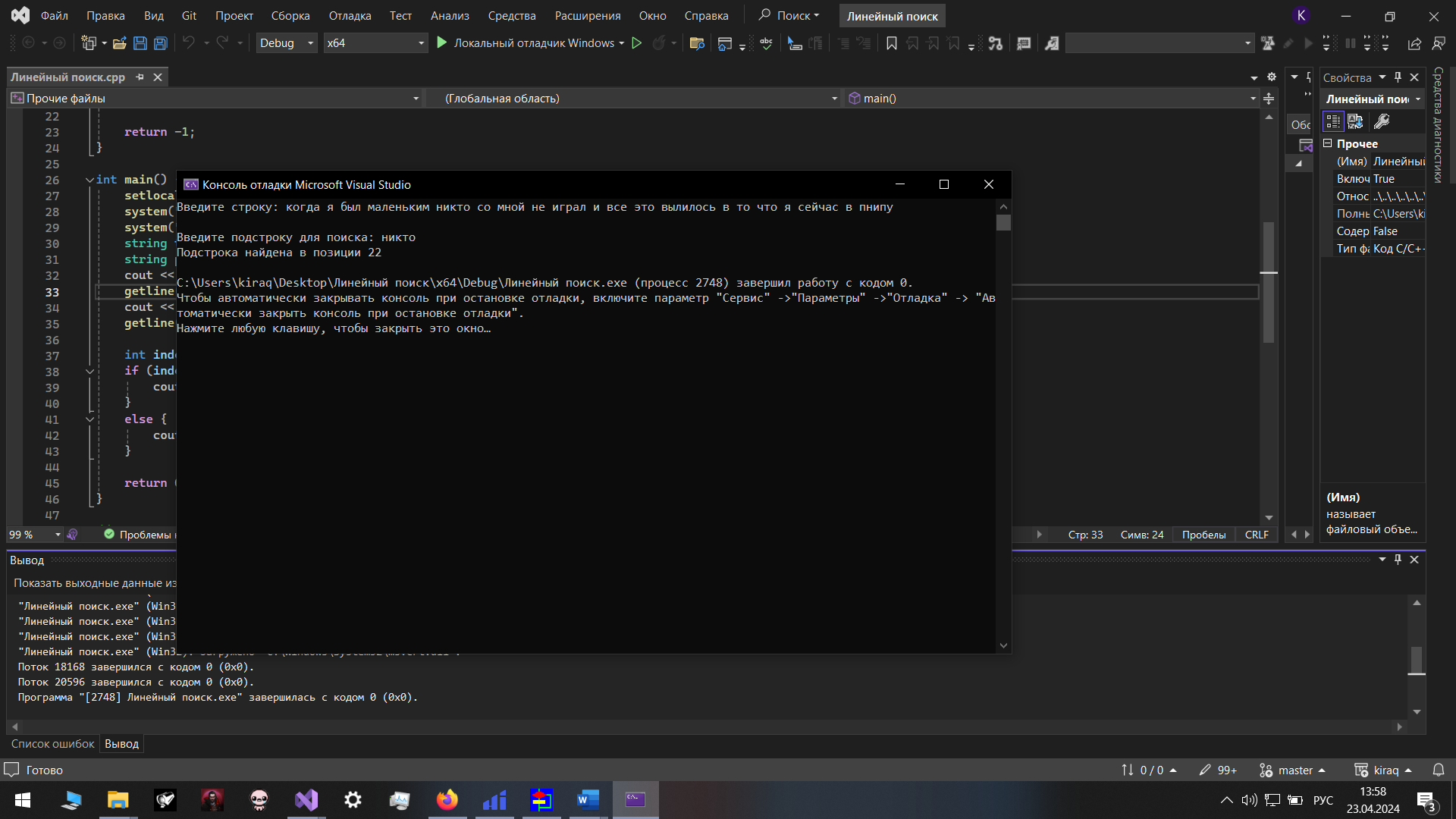
cout << "Подстрока не найдена." << endl;

}

return 0;

}

**Пример работы кода:**



**Вывод:**

Код отработал как задумано, алгоритм работает исправно