

Федеральное государственное автономное образовательное учреждение
высшего образования «Пермский национальный исследовательский
политехнический университет»

Лабораторная работа
«Методы поиска»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-36

Акбашева Софья Руслановна

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Методы поиска

Цель: знакомство с простыми методами поиска, в числе которых: линейный поиск, бинарный поиск, интерполяционный поиск, прямой поиск подстроки.

Постановка задачи: реализовать поиск заданного элемента, с методов поиска: линейный, бинарный, интерполяционный, прямой поиск подстроки.

Анализ задачи

1) В программе есть несколько циклов do-while. Первый цикл используется для выбора способа поиска: линейный, бинарный, интерполяционный или прямой поиск подстроки. Второй цикл используется для ввода размера массива, а третий цикл - для выбора способа создания массива: вручную или случайно.

2) Если элемент не найден, выводится сообщение "Такого элемента нет!". Если элемент найден, выводится сообщение "Номер элемента - " и номер найденного элемента.

3) Для некоторых методов поиска необходим отсортированный массив, поэтому необходимо реализовать функцию сортировки. В начале функции Quick_Sort проверяется, что индекс start_Index меньше индекса pivot_Index. Если это условие выполняется, то начинается процесс сортировки. Затем в цикле for происходит проход по элементам вектора от индекса current_Index до индекса pivot_Index. Если текущий элемент меньше или равен элементу на индексе pivot_Index, то происходит обмен элементов местами. Это делается для того, чтобы переместить все меньшие элементы влево от опорного элемента, а все большие элементы - вправо. После этого происходит обмен элементов на индексах current_Index и pivot_Index. Это делается для того, чтобы опорный элемент оказался на своем месте после сортировки. Затем вызываются две функции Quick_Sort(vect, start_Index, current_Index - 1) и Quick_Sort(vect, current_Index + 1, pivot_Index). Эти функции вызывают быструю сортировку для двух подмассивов, которые были разделены опорным элементом. Таким образом, алгоритм быстрой

сортировки разделяет исходный массив на два подмассива, сортирует каждый из них рекурсивно, а затем объединяет их.

4) Линейный поиск. В начале функции `linear_search(vector<int>& vect, int element)` инициализируется переменная `index_el` значением -1. Это делается для того, чтобы указать, что элемент еще не найден. Затем начинается цикл `for`, который проходит по всем элементам вектора. В каждой итерации цикла проверяется, равен ли текущий элемент искомому элементу. Если это так, то переменная `index_el` присваивается значению индекса текущего элемента. Цикл продолжается до тех пор, пока не будут проверены все элементы вектора или пока не будет найден искомый элемент. Если элемент не найден, то переменная `index_el` остается равной -1. В конце функции возвращается значение переменной `index_el`. Если элемент был найден, то возвращается его индекс, в противном случае возвращается -1.

5) Бинарный поиск. В начале функции `Binary_search(vector<int>& vect, int key_el)` инициализируются переменные `index_el`, `middle_index`, `the_left_border` и `the_right_border`. Переменная `index_el` инициализируется значением -1, что указывает на то, что элемент еще не найден. Переменные `the_left_border` и `the_right_border` инициализируются значениями 0 и `vect.size() - 1` соответственно. Это начальные границы поиска. Затем начинается цикл `while`, который продолжается до тех пор, пока не будет найден искомый элемент или пока не будут проверены все элементы массива. В каждой итерации цикла вычисляется индекс среднего элемента относительно текущих границ (`the_left_border` и `the_right_border`). Затем проверяется, равен ли средний элемент искомому элементу. Если это так, то переменная `index_el` присваивается значению индекса среднего элемента. Если средний элемент меньше искомого элемента, то левая граница поиска увеличивается на 1. Если средний элемент больше искомого элемента, то правая граница поиска уменьшается на 1. Цикл продолжается до тех пор, пока не будет найден искомый элемент или пока не будут проверены все элементы массива. Если элемент не найден, то переменная `index_el` остается

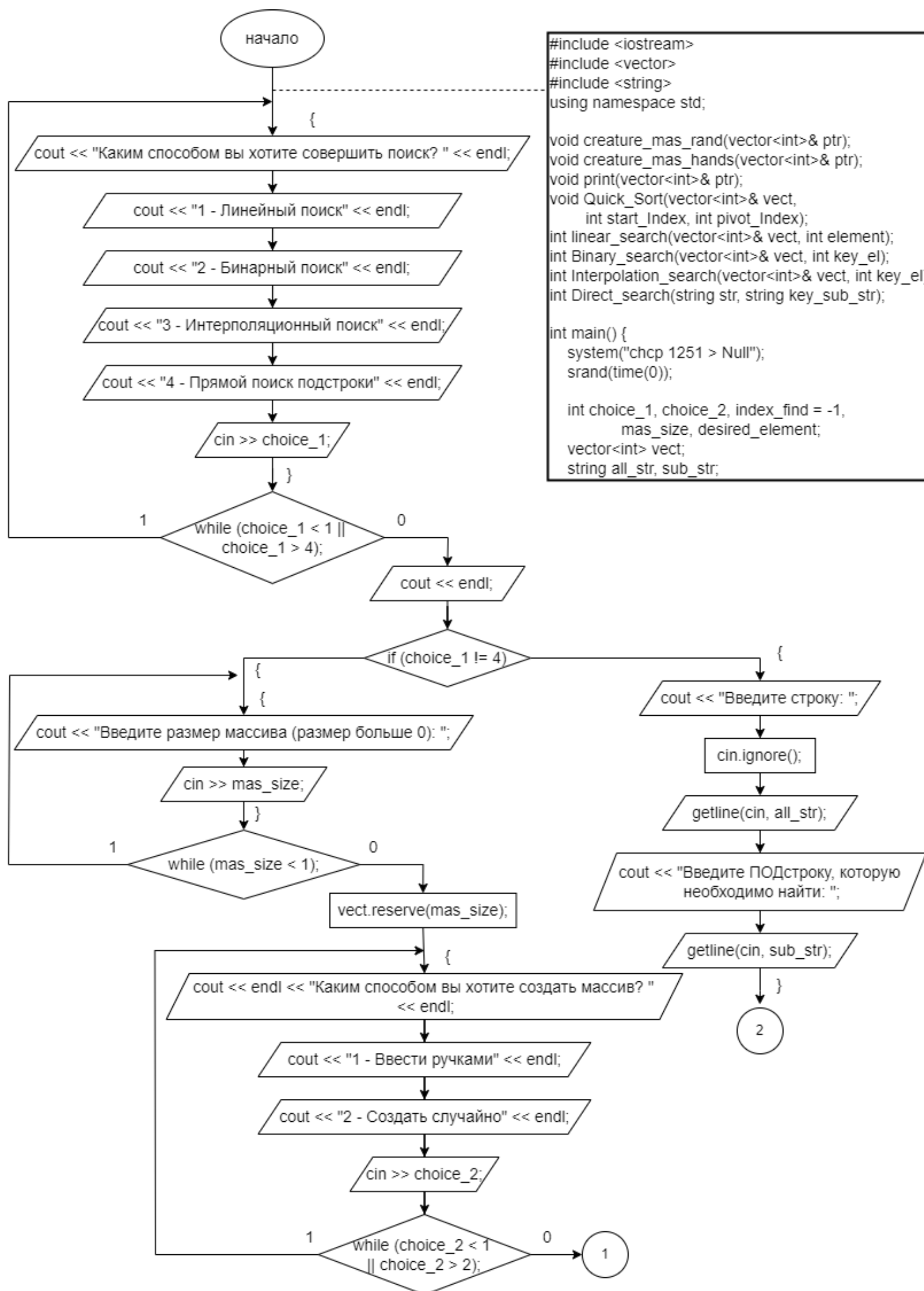
равной -1. В конце функции возвращается значение переменной `index_el`. Если элемент был найден, то возвращается его индекс, в противном случае возвращается -1.

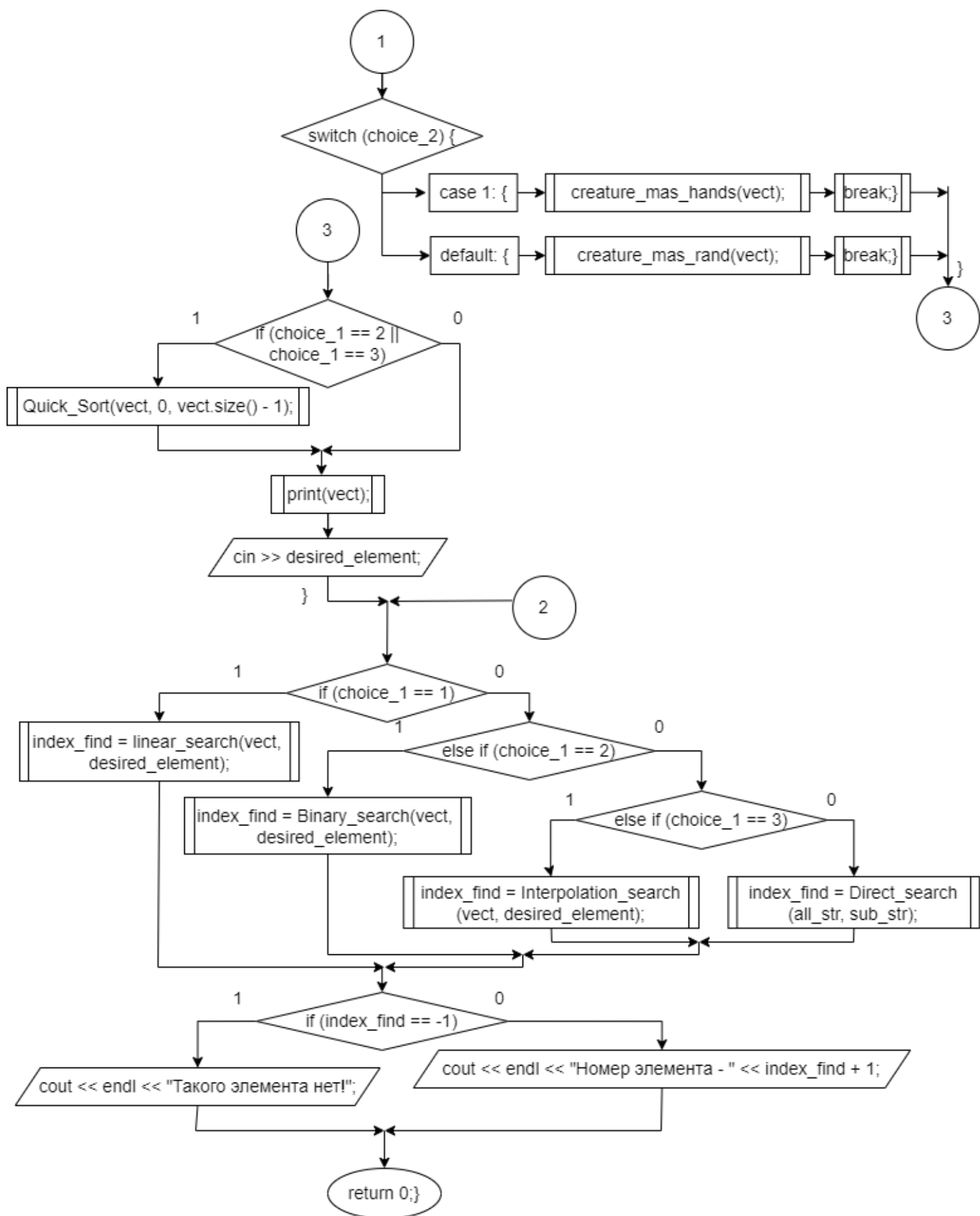
6) Интерполяционный поиск. В начале функции `Interpolation_search(vector<int>& vect, int key_el)` инициализируются переменные `index_el`, `middle_index`, `the_left_border` и `the_right_border`. Переменная `index_el` инициализируется значением -1, что указывает на то, что элемент еще не найден. Переменные `the_left_border` и `the_right_border` инициализируются значениями 0 и `vect.size() - 1` соответственно. Это начальные границы поиска. Затем начинается цикл `while`, который продолжается до тех пор, пока не будет найден искомый элемент или пока не будут проверены все элементы массива. В каждой итерации цикла вычисляется индекс среднего элемента относительно текущих границ (`the_left_border` и `the_right_border`). Затем проверяется, равен ли средний элемент искомому элементу. Если это так, то переменная `index_el` присваивается значению индекса среднего элемента. Если средний элемент меньше искомого элемента, то левая граница поиска увеличивается на 1. Если средний элемент больше искомого элемента, то правая граница поиска уменьшается на 1. Цикл продолжается до тех пор, пока не будет найден искомый элемент или пока не будут проверены все элементы массива. Если элемент не найден, то переменная `index_el` остается равной -1. В конце функции возвращается значение переменной `index_el`. Если элемент был найден, то возвращается его индекс, в противном случае возвращается -1.

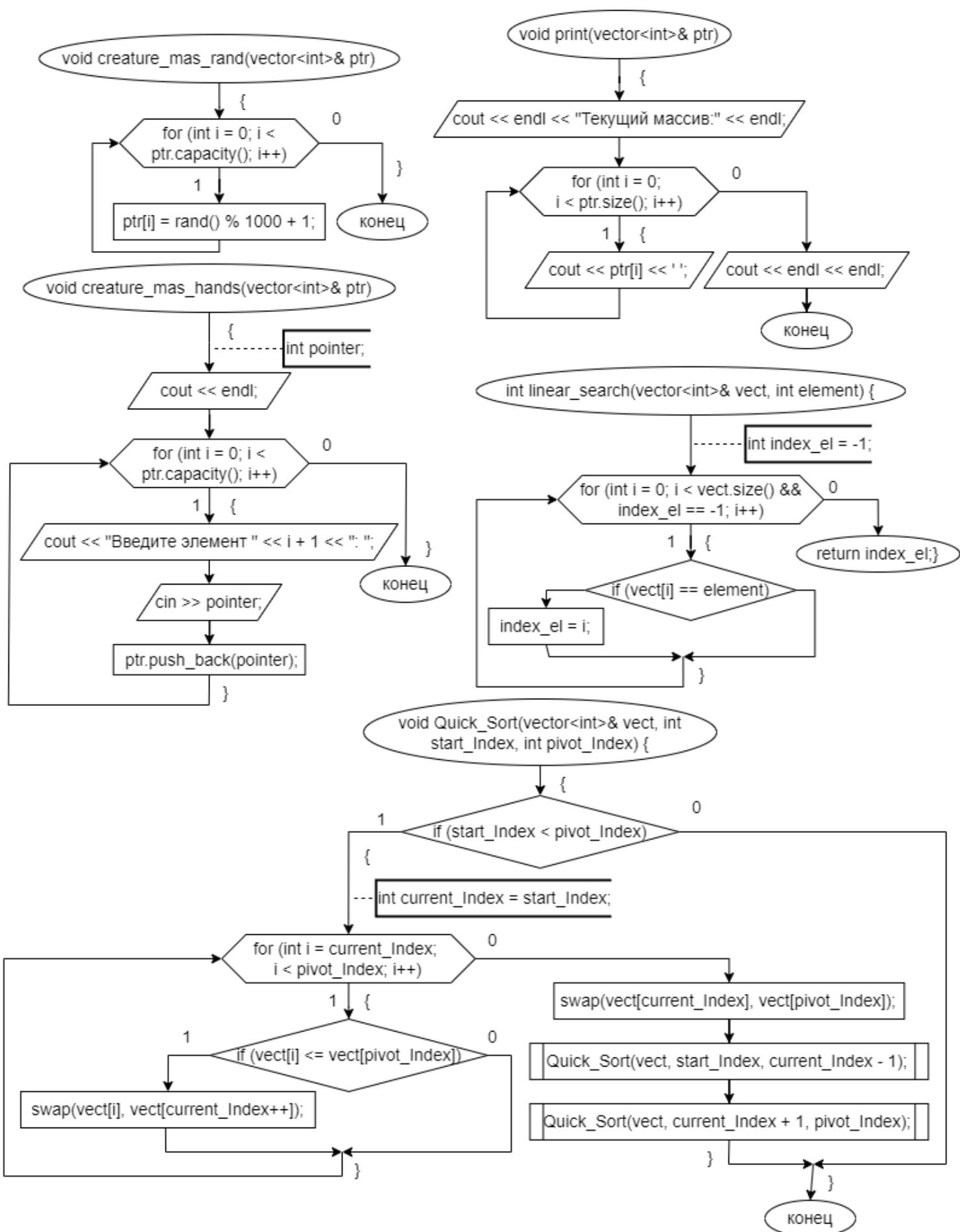
7) Прямой поиск подстроки. В начале функции `Direct_search(string str, string key_sub_str)` инициализируются переменные `index_el` и `len_sub_str`. Переменная `index_el` инициализируется значением -1, что указывает на то, что подстрока еще не найдена. Переменная `len_sub_str` инициализируется значением длины подстроки. Затем начинается цикл `for`, который проходит по всем символам строки. В каждой итерации цикла проверяется, совпадает ли текущий символ строки с первым символом подстроки. Если это так, то

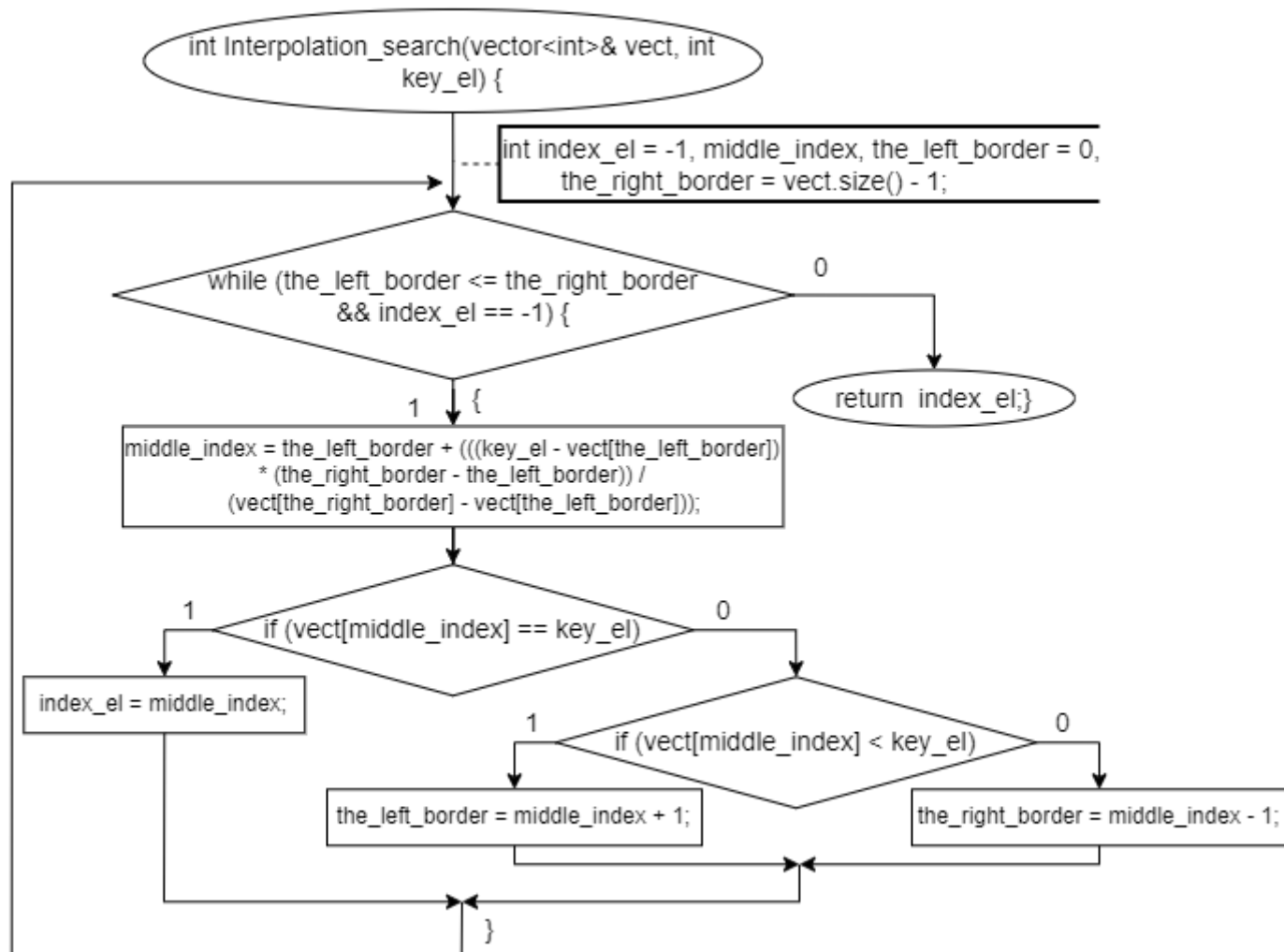
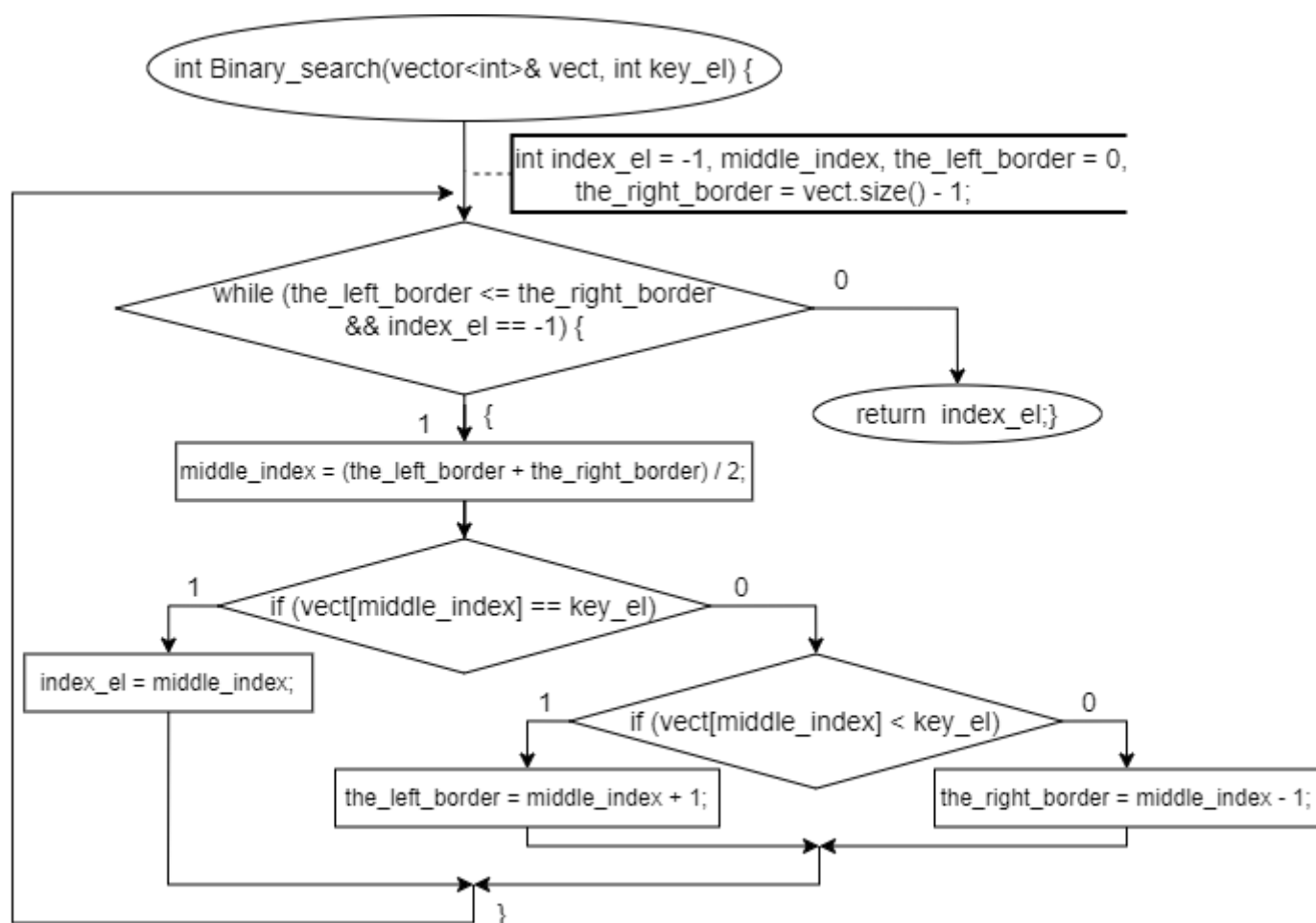
начинается поиск совпадения подстроки в строке. Затем начинается цикл `while`, который проходит по всем символам подстроки. В каждой итерации цикла проверяется, совпадает ли текущий символ подстроки с текущим символом строки. Если все символы подстроки совпадают с символами строки, то переменная `index_el` присваивается значению индекса первого символа подстроки. Цикл продолжается до тех пор, пока не будет найдена подстрока или пока не будут проверены все символы строки. Если подстрока не найдена, то переменная `index_el` остается равной `-1`. В конце функции возвращается значение переменной `index_el`. Если подстрока была найдена, то возвращается ее индекс, в противном случае возвращается `-1`.

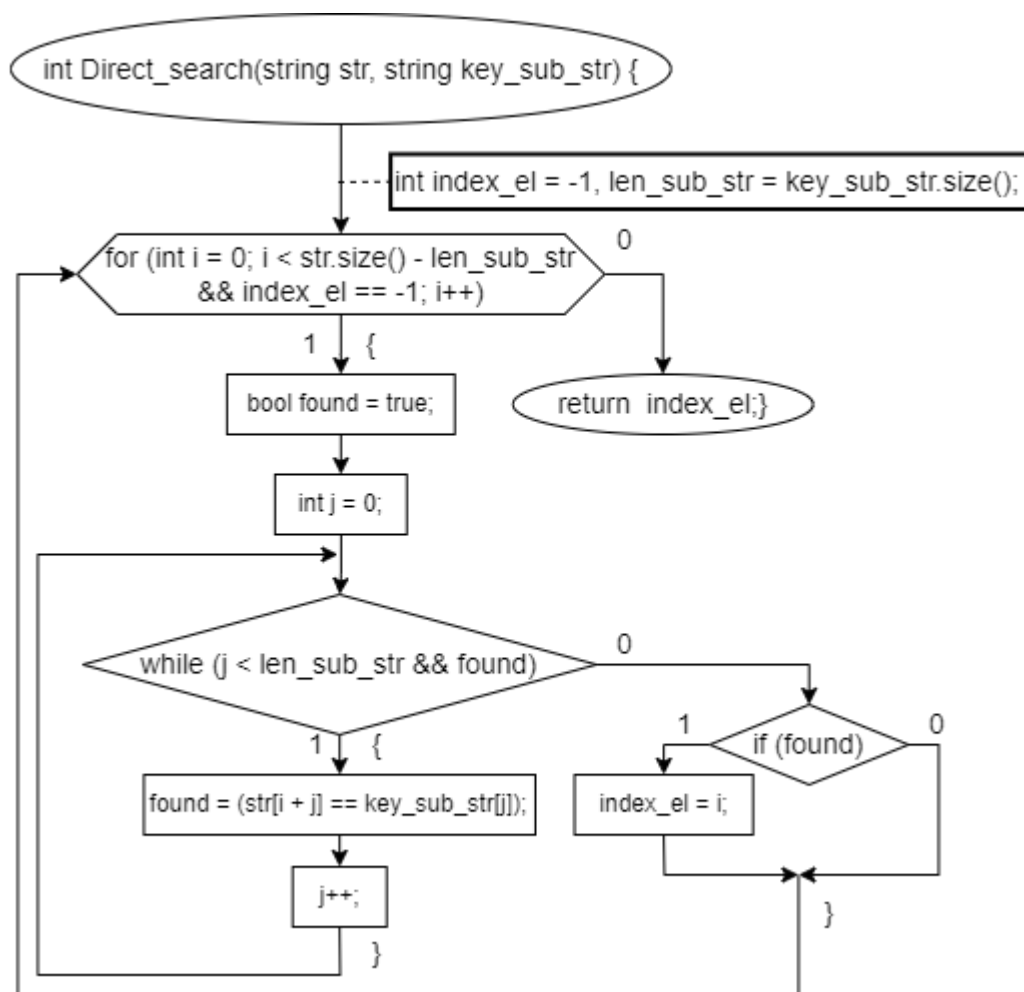
Блок схема











Код

```

#include <iostream>
#include <vector>
#include <string>
using namespace std;

void creature_mas_rand(vector<int>& ptr); //создаю массив через датчик случайных чисел
void creature_mas_hands(vector<int>& ptr); //создание массива с клавиатуры
void print(vector<int>& ptr); //вывод текущего массива
void Quick_Sort(vector<int>& vect, int start_Index, int pivot_Index); //быстрая сортировка
int linear_search(vector<int>& vect, int element); //линейный поиск
int Binary_search(vector<int>& vect, int key_el); //бинарный поиск
int Interpolation_search(vector<int>& vect, int key_el); //интерполяционный поиск
int Direct_search(string str, string key_sub_str); //прямой поиск подстроки в строке

int main() {
    system("chcp 1251 > Null");
    srand(time(0));

    int choice_1, choice_2, index_find = -1, mas_size, desired_element;
    vector<int> vect;
    string all_str, sub_str;

    do { //выбор поиска
        cout << "Каким способом вы хотите совершить поиск? " << endl;
        cout << "1 - Линейный поиск" << endl;
        cout << "2 - Бинарный поиск" << endl;
        cout << "3 - Интерполяционный поиск" << endl;
    } while (choice_1 < 1 || choice_1 > 3);

    if (choice_1 == 1) {
        linear_search(vect, desired_element);
    } else if (choice_1 == 2) {
        Binary_search(vect, desired_element);
    } else if (choice_1 == 3) {
        Interpolation_search(vect, desired_element);
    }

    Direct_search(all_str, sub_str);
}

```

```

    cout << "4 - Прямой поиск подстроки" << endl;
    cin >> choice_1;
} while (choice_1 < 1 || choice_1 > 4);

cout << endl;

if (choice_1 != 4) {
    do {
        cout << "Введите размер массива (размер больше 0): ";
        cin >> mas_size; //ввод размера массива
    } while (mas_size < 1);

    vect.reserve(mas_size); //резервирую память

    do {
        cout << endl << "Каким способом вы хотите создать массив? " << endl;
        cout << "1 - Ввести ручками" << endl;
        cout << "2 - Создать случайно" << endl;
        cin >> choice_2;
    } while (choice_2 < 1 || choice_2 > 2);

    switch (choice_2) { //выбор способа создания массива
    case 1: {
        creature_mas_hands(vect);
        break;
    }
    default: {
        creature_mas_rand(vect); //создание массива случайно
        break;
    }
    }

    if (choice_1 == 2 || choice_1 == 3) { //если надо отсортировать
        Quick_Sort(vect, 0, vect.size() - 1);
    }

    print(vect);
    cout << "Введите элемент, который необходимо найти: ";
    cin >> desired_element;
}
else {
    cout << "Введите строку: ";
    cin.ignore();
    getline(cin, all_str);
    cout << "Введите ПОДстроку, которую необходимо найти: ";
    getline(cin, sub_str);
}

if (choice_1 == 1) { //линейный поиск
    index_find = linear_search(vect, desired_element);
}
else if (choice_1 == 2) { //бинарный поиск
    index_find = Binary_search(vect, desired_element);
}
else if (choice_1 == 3) { //интерполяционный поиск
    index_find = Interpolation_search(vect, desired_element);
}
else { //прямой поиск подстроки
    index_find = Direct_search(all_str, sub_str);
}

if (index_find == -1) {

```

```

        cout << endl << "Такого элемента нет!";
    }
    else {
        cout << endl << "Номер элемента - " << index_find + 1;
    }

    return 0;
}

void creature_mas_rand(vector<int>& ptr) { //создаю массив через датчик случайных чисел
    for (int i = 0; i < ptr.capacity(); i++) {
        ptr.push_back(rand() % 1000 + 1); //случайное число от 1 до 1000
    }
}

void creature_mas_hands(vector<int>& ptr) { //создание массива с клавиатуры
    cout << endl;
    int pointer;
    for (int i = 0; i < ptr.capacity(); i++) {
        cout << "Введите элемент " << i + 1 << ": ";
        cin >> pointer;
        ptr.push_back(pointer);
    }
}

void print(vector<int>& ptr) { //вывод текущего массива
    cout << endl << "Текущий массив:" << endl;
    for (int i = 0; i < ptr.size(); i++) { //прохожу по массиву
        cout << ptr[i] << ' ';
    }
    cout << endl << endl;
}

void Quick_Sort(vector<int>& vect, int start_Index, int pivot_Index) { //быстрая сортировка
    if (start_Index < pivot_Index) {
        int current_Index = start_Index;
        for (int i = current_Index; i < pivot_Index; i++) { //иду до опорного элемента
            if (vect[i] <= vect[pivot_Index]) {
                swap(vect[i], vect[current_Index++]); //меняю элементы местами
            }
        }
        swap(vect[current_Index], vect[pivot_Index]); //меняю элементы местами
        Quick_Sort(vect, start_Index, current_Index - 1);
        Quick_Sort(vect, current_Index + 1, pivot_Index);
    }
}

int linear_search(vector<int>& vect, int element) { //линейный поиск
    int index_el = -1;
    for (int i = 0; i < vect.size() && index_el == -1; i++) {
        if (vect[i] == element) { //если элемент совпал
            index_el = i;
        }
    }
    return index_el;
}

int Binary_search(vector<int>& vect, int key_el) { //бинарный поиск
    int index_el = -1, middle_index, the_left_border = 0, the_right_border = vect.size() - 1;
    while (the_left_border <= the_right_border && index_el == -1) {
        middle_index = (the_left_border + the_right_border) / 2; //средний элемент относительно текущих границ
        if (vect[middle_index] == key_el) { //если элемент совпал

```

```

        index_el = middle_index;
    }
    else {
        if (vect[middle_index] < key_el) { //если средний элемент меньше ключевого
            the_left_border = middle_index + 1; //меняю левую границу
        }
        else { //если средний элемент больше ключевого
            the_right_border = middle_index - 1; //меняю правую границу
        }
    }
}
return index_el;
}

int Interpolation_search(vector<int>& vect, int key_el) { //интерполяционный поиск
    int index_el = -1, middle_index, the_left_border = 0, the_right_border = vect.size() - 1;
    while (the_left_border <= the_right_border && index_el == -1) {
        middle_index = the_left_border + (((key_el - vect[the_left_border]) * (the_right_border - the_left_border)) /
            (vect[the_right_border] - vect[the_left_border])); //средний элемент относительно текущих границ
        if (vect[middle_index] == key_el) { //если элемент совпал
            index_el = middle_index;
        }
        else {
            if (vect[middle_index] < key_el) { //если средний элемент меньше ключевого
                the_left_border = middle_index + 1; //меняю левую границу
            }
            else { //если средний элемент больше ключевого
                the_right_border = middle_index - 1; //меняю правую границу
            }
        }
    }
}
return index_el;
}

int Direct_search(string str, string key_sub_str) { //прямой поиск подстроки в строке
    int index_el = -1, len_sub_str = key_sub_str.size();
    for (int i = 0; i < str.size() - len_sub_str && index_el == -1; i++) { //иду по строке
        bool found = true; //статус совпадения подстроки и строки
        int j = 0; //длина совпадающей части
        while (j < len_sub_str && found) { //иду по ПОДстроке
            found = (str[i + j] == key_sub_str[j]); //совпадение символа строки и ПОДстроки
            j++; //длина совпадающей части
        }
        if (found) { //если все элементы совпали
            index_el = i; //индекс первого элемента
        }
    }
}
return index_el;
}

```

Результат работы

1) Лине́йный поиск

Каким способом вы хотите совершить поиск?

- 1 - Линейный поиск
- 2 - Бинарный поиск
- 3 - Интерполяционный поиск
- 4 - Прямой поиск подстроки

1

Введите размер массива (размер больше 0): 20

Каким способом вы хотите создать массив?

- 1 - Ввести ручками
- 2 - Создать случайно

2

Текущий массив:

419 404 409 986 692 536 778 61 340 829 578 637 980 740 345 42 900 922 208 432

Введите элемент, который необходимо найти: 778

Номер элемента - 7

2) Бинарный поиск

Каким способом вы хотите совершить поиск?

- 1 - Линейный поиск
- 2 - Бинарный поиск
- 3 - Интерполяционный поиск
- 4 - Прямой поиск подстроки

2

Введите размер массива (размер больше 0): 6

Каким способом вы хотите создать массив?

- 1 - Ввести ручками
- 2 - Создать случайно

1

Введите элемент 1: 5

Введите элемент 2: 67

Введите элемент 3: 876

Введите элемент 4: 987

Введите элемент 5: 1000

Введите элемент 6: 99999

Текущий массив:

5 67 876 987 1000 99999

Введите элемент, который необходимо найти: 99999

Номер элемента - 6

3) Интерполяционный поиск

```

1 Каким способом вы хотите совершить поиск?
2 1 - Линейный поиск
3 2 - Бинарный поиск
4 3 - Интерполяционный поиск
5 4 - Прямой поиск подстроки
6 3
7
8 Введите размер массива (размер больше 0): 8
9
10 Каким способом вы хотите создать массив?
11 1 - Ввести ручками
12 2 - Создать случайно
13 2
14
15 Текущий массив:
16 13 156 168 193 653 801 974 975
17
18 Введите элемент, который необходимо найти: 193
19
20 Номер элемента - 4

```

4) Прямой поиск подстроки

```

1 Каким способом вы хотите совершить поиск?
2 1 - Линейный поиск
3 2 - Бинарный поиск
4 3 - Интерполяционный поиск
5 4 - Прямой поиск подстроки
6 4
7
8 Введите строку: fd aw df aueiuri
9 Введите ПОДстроку, которую необходимо найти: w d
10
11 Номер элемента - 5

```

Вывод

В ходе работы я применила знания о работе различных методов поиска, в числе которых: линейный поиск, бинарный поиск, интерполяционный поиск, прямой поиск подстроки. Мне удалось реализовать поставленную задачу: были созданы все необходимые методы поиска.

GitHub

Ссылка: <https://github.com/SonyAkb/sorting-and-searching.git>

main









1 Branch

0 Tags

Go to file

Add file

Code

<div>  <div>SonyAkb Delete Методы поиска.pdf</div> <div>2007b3f · 2 hours ago</div> <div>40 Commits</div> </div>		
 БМ и КМП.cpp	Смена кодировки	2 hours ago
 ЛР Простые сортировки.pdf	Отчет	last week
 Методы поиска.cpp	Смена кодировки	2 hours ago
 Сортировки простые.cpp	Измерение времени сортировок	last week
 методы поиска блок-схема.drawio.png	Обновление методы поиска блок-схема.drawio.png	7 hours ago
 поиск БМ и КМП.drawio.png	Обновление поиск БМ и КМП.drawio.png	5 hours ago
 сортировки простые блок-схема.drawio.png	Обновление сортировки простые блок-схема.drawio.png	last week