Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №12, 14**

Дисциплина: «Информатика»

Тема: Алгоритмы поиска. Работа со списками. Метод интерполяционного поиска. Методы поиска: метод Кнутта-Мориса Пратта, метод Бойера-Мура

Вариант 18

Выполнил:

Студент группы РИС-20-1б

Кузнецов Михаил Сергеевиич

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Цель работы**

Изучить алгоритмы поиска данных и реализовать их в программе.

**Постановка задачи**

1) Необходимо разработать программу, которая создаёт массивы данных, выполняет поиск по этим массивам и выдаёт результат.

2) Взаимодействие с пользователем производится через меню.

3) В зависимости от команд пользователя программа выполняет поиск данных и выводит результат

**Анализ задачи**

1. Определить какие действия необходимо выполнить для решения задачи.

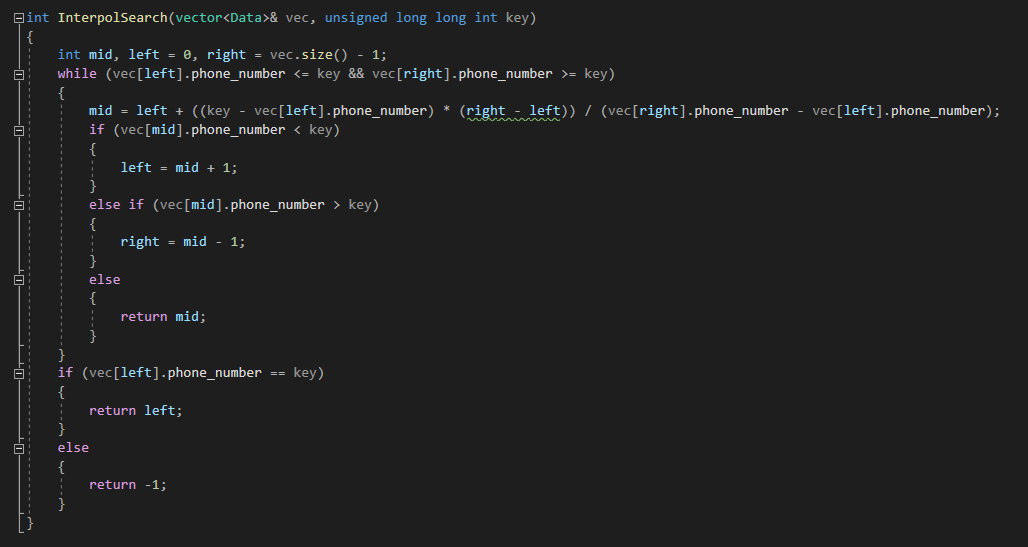
* Добавление и удаление элементов.
* Поиск данных
* Чтение из файла
* Запись в файл
* Вывод в консоль

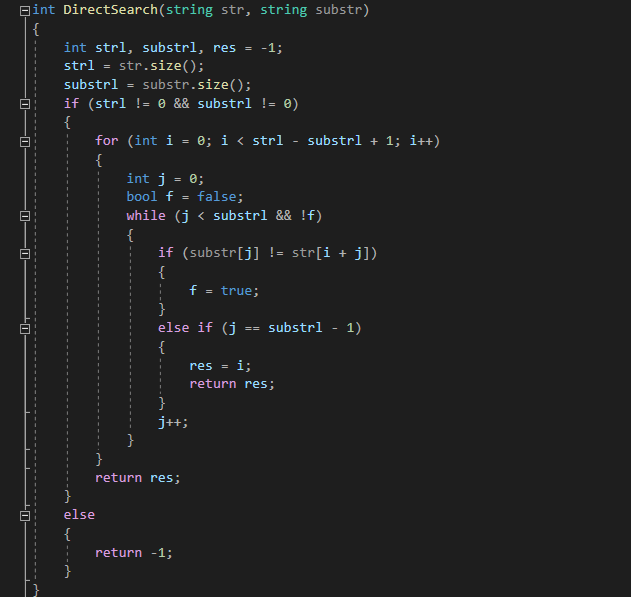
2) Информация из файла хранится в векторе

  
3) Считывание данных производится через цикл while:

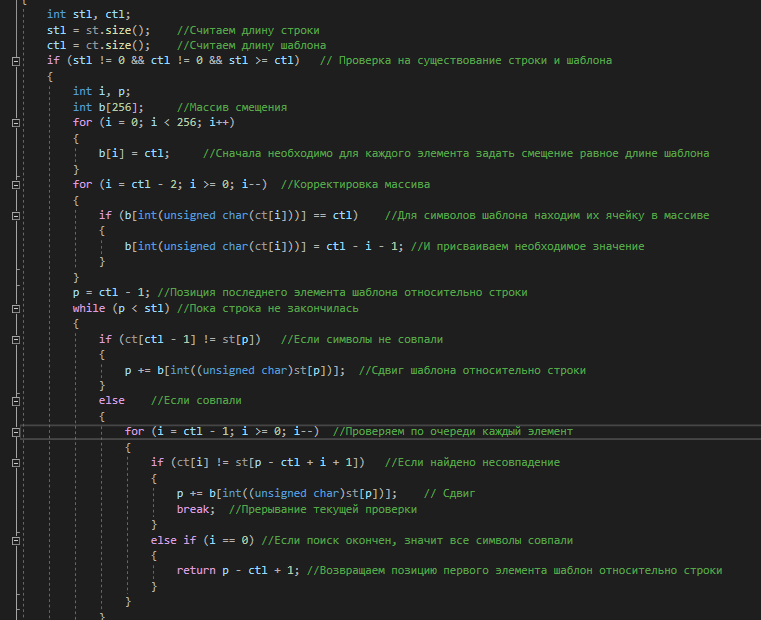


4) Интерполяционный поиск:

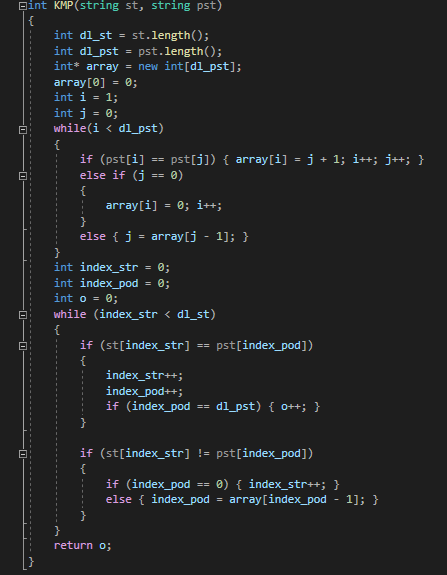
5) Прямой поиск:



6) Алгоритм Боера-Мура



7) Алгоритм Кнута-Морриса-Пратта:



**Код**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

struct Data

{

string fio;

string date;

unsigned long long int phone\_number = 0;

};

template <typename T>

void get\_value(T& value)

{

bool is\_correct = false;

do

{

cin >> value;

if (cin.fail())

{

cout << "\nIncorrect value" << endl << endl;

cin.clear();

cin.ignore(32767, '\n');

}

else

{

is\_correct = true;

cin.ignore(32767, '\n');

}

} while (!is\_correct);

}

void read\_file(fstream& file, vector<Data>& vec)

{

string currentLine;

Data new\_elem;

while (!file.eof())

{

getline(file, currentLine);

new\_elem.fio = currentLine;

getline(file, currentLine);

new\_elem.date = currentLine;

getline(file, currentLine);

new\_elem.phone\_number = stoull(currentLine);

getline(file, currentLine);

vec.push\_back(new\_elem);

}

}

void print(vector<Data>& vec)

{

for (unsigned int i = 0; i < vec.size(); i++)

{

cout << "Element № " << i + 1 << endl;

cout << "FIO: " << vec[i].fio << endl;

cout << "Date: " << vec[i].date << endl;

cout << "Phone: " << vec[i].phone\_number << endl << endl;

}

}

void print\_struc(Data& d)

{

cout << "FIO: " << d.fio << endl;

cout << "Date: " << d.date << endl;

cout << "Phone: " << d.phone\_number << endl << endl;

}

void print\_oper()

{

cout << "\nChoose the required action:\n";

cout << "0 - Find a person by phone number\n";

cout << "1 - Find a person by name\n";

cout << "2 - Add new person\n";

cout << "3 - Delete information about person\n";

cout << "4 - Save current array in file\n";

cout << "5 - Print array\n";

cout << "6 - Exit\n\n";

}

void sort\_field(vector<Data> &vec)

{

Data replace;

int size = vec.size();

for (int i = 0; i < size; i++)

{

for (int j = i + 1; j < size; j++)

{

if (vec[i].phone\_number > vec[j].phone\_number)

{

replace = vec[i];

vec[i] = vec[j];

vec[j] = replace;

}

}

}

}

int InterpolSearch(vector<Data>& vec, unsigned long long int key)

{

int mid, left = 0, right = vec.size() - 1;

while (vec[left].phone\_number <= key && vec[right].phone\_number >= key)

{

mid = left + ((key - vec[left].phone\_number) \* (right - left)) / (vec[right].phone\_number - vec[left].phone\_number);

if (vec[mid].phone\_number < key)

{

left = mid + 1;

}

else if (vec[mid].phone\_number > key)

{

right = mid - 1;

}

else

{

return mid;

}

}

if (vec[left].phone\_number == key)

{

return left;

}

else

{

return -1;

}

}

int DirectSearch(string str, string substr)

{

int strl, substrl, res = -1;

strl = str.size();

substrl = substr.size();

if (strl != 0 && substrl != 0)

{

for (int i = 0; i < strl - substrl + 1; i++)

{

int j = 0;

bool f = false;

while (j < substrl && !f)

{

if (substr[j] != str[i + j])

{

f = true;

}

else if (j == substrl - 1)

{

res = i;

return res;

}

j++;

}

}

return res;

}

else

{

return -1;

}

}

int BMura(string st, string ct)

{

int stl, ctl;

stl = st.size(); //Считаем длину строки

ctl = ct.size(); //Считаем длину шаблона

if (stl != 0 && ctl != 0 && stl >= ctl) // Проверка на существование строки и шаблона

{

int i, p;

int b[256]; //Массив смещения

for (i = 0; i < 256; i++)

{

b[i] = ctl; //Сначала необходимо для каждого элемента задать смещение равное длине шаблона

}

for (i = ctl - 2; i >= 0; i--) //Корректировка массива

{

if (b[int(unsigned char(ct[i]))] == ctl) //Для символов шаблона находим их ячейку в массиве

{

b[int(unsigned char(ct[i]))] = ctl - i - 1; //И присваиваем необходимое значение

}

}

p = ctl - 1; //Позиция последнего элемента шаблона относительно строки

while (p < stl) //Пока строка не закончилась

{

if (ct[ctl - 1] != st[p]) //Если символы не совпали

{

p += b[int((unsigned char)st[p])]; //Сдвиг шаблона относительно строки

}

else //Если совпали

{

for (i = ctl - 1; i >= 0; i--) //Проверяем по очереди каждый элемент

{

if (ct[i] != st[p - ctl + i + 1]) //Если найдено несовпадение

{

p += b[int((unsigned char)st[p])]; // Сдвиг

break; //Прерывание текущей проверки

}

else if (i == 0) //Если поиск окончен, значит все символы совпали

{

return p - ctl + 1; //Возвращаем позицию первого элемента шаблон относительно строки

}

}

}

}

}

return -1; //Если шаблона нет в строке или переданы некорректные данные возвращаем -1

}

int KMP(string st, string pst)

{

int dl\_st = st.length();

int dl\_pst = pst.length();

int\* array = new int[dl\_pst];

array[0] = 0;

int i = 1;

int j = 0;

while(i < dl\_pst)

{

if (pst[i] == pst[j]) { array[i] = j + 1; i++; j++; }

else if (j == 0)

{

array[i] = 0; i++;

}

else { j = array[j - 1]; }

}

int index\_str = 0;

int index\_pod = 0;

int o = 0;

while (index\_str < dl\_st)

{

if (st[index\_str] == pst[index\_pod])

{

index\_str++;

index\_pod++;

if (index\_pod == dl\_pst) { o++; }

}

if (st[index\_str] != pst[index\_pod])

{

if (index\_pod == 0) { index\_str++; }

else { index\_pod = array[index\_pod - 1]; }

}

}

return o;

}

void find\_num(vector<Data> vec)

{

unsigned long long int number;

cout << "\n>Enter phone number\n";

get\_value(number);

// Line search

cout << "\nLine search method:\n";

vector<Data> line\_out;

for (unsigned int i = 0; i < vec.size(); i++)

{

if (vec[i].phone\_number == number)

{

line\_out.push\_back(vec[i]);

}

}

if (line\_out.size() > 0)

{

print(line\_out);

}

else

{

cout << "This phone number was not found\n\n";

}

//Interpolation method

cout << "\nInterpolation method:\n";

vector<Data> interp\_out;

interp\_out = vec;

sort\_field(interp\_out);

int result = InterpolSearch(interp\_out, number);

if (result != -1)

{

print\_struc(interp\_out[result]);

}

else

{

cout << "This phone number was not found\n\n";

}

}

void find\_name(vector<Data> vec)

{

cout << "\nEnter name, or surname, or middle name\n";

string name;

getline(cin, name);

int size = vec.size();

vector<Data> out;

// Direct search method

cout << "\nDirect search method:\n";

for (int i = 0; i < size; i++)

{

if (DirectSearch(vec[i].fio, name) != -1)

{

out.push\_back(vec[i]);

}

}

if (out.size() != 0)

{

cout << "\n==Searching results==\n";

print(out);

}

else

{

cout << "This person not found\n\n";

}

out.clear();

// Method Boera-Mura

cout << "\nMethod Boera-Mura:\n";

for (int i = 0; i < size; i++)

{

if (BMura(vec[i].fio, name) != -1)

{

out.push\_back(vec[i]);

}

}

if (out.size() != 0)

{

cout << "\n==Searching results==\n";

print(out);

}

else

{

cout << "This person not found\n\n";

}

out.clear();

// Method KMP

cout << "\nMethod Knuta-Morrisa-Pratta:\n";

for (int i = 0; i < size; i++)

{

if (KMP(vec[i].fio, name) > 0)

{

out.push\_back(vec[i]);

}

}

if (out.size() != 0)

{

cout << "\n==Searching results==\n";

print(out);

}

else

{

cout << "This person not found\n\n";

}

}

void add\_new\_pers(vector<Data>& vec)

{

int pos;

do

{

cout << "Enter position from 1 to " << vec.size() + 1 << endl;

get\_value(pos);

} while (pos < 1 || pos > vec.size() + 1);

Data pers{"Балуев Александр Петрович", "19.01.2002", 89454511238};

if (pos == vec.size() + 1)

{

vec.push\_back(pers);

}

else

{

auto it = vec.begin();

vec.insert(it + pos - 1, pers);

}

print(vec);

}

void delete\_pers(vector<Data>& vec)

{

if (vec.size() != 0){

int type;

int res = -1;

do {

cout << "\n0 - Delete by phone number\n";

cout << "1 - Delete by index\n";

get\_value(type);

} while (type < 0 || type > 1);

switch (type)

{

case (0):

unsigned long long int number;

cout << "\nEnter phone number\n";

get\_value(number);

for (unsigned int i = 0; i < vec.size(); i++)

{

if (vec[i].phone\_number == number)

{

res = i;

}

}

if (res != -1)

{

auto it = vec.begin();

vec.erase(it + res);

}

else

{

cout << "Phone number not found\n";

}

break;

case (1):

int pos;

do

{

cout << "\nEnter position from 1 to " << vec.size() << endl;

get\_value(pos);

} while (pos < 1 || pos > vec.size());

auto it = vec.begin();

vec.erase(it + pos - 1);

break;

}

}

else

{

cout << "\nData array is empty\n\n";

}

}

void write\_in\_file(vector<Data>& vec)

{

remove("OutLab12.txt");

ofstream file("OutLab12.txt");

for (int i = 0; i < vec.size(); i++)

{

file << vec[i].fio << endl;

file << vec[i].date << endl;

file << vec[i].phone\_number << endl;

file << endl;

}

}

int main()

{

system("chcp 1251>null");

fstream input\_file("F1.txt");

if (input\_file.is\_open())

{

vector<Data> vec\_data;

read\_file(input\_file, vec\_data);

print(vec\_data);

bool exit = false;

int oper;

while (!exit) //Main loop

{

print\_oper();

get\_value(oper);

switch (oper)

{

case (0):

find\_num(vec\_data);

break;

case (1):

find\_name(vec\_data);

break;

case (2):

add\_new\_pers(vec\_data);

break;

case (3):

delete\_pers(vec\_data);

break;

case (4):

write\_in\_file(vec\_data);

break;

case (5):

print(vec\_data);

break;

case (6):

exit = true;

break;

default:

cout << "\nIncorrect value\n\n";

break;

}

}

}

else

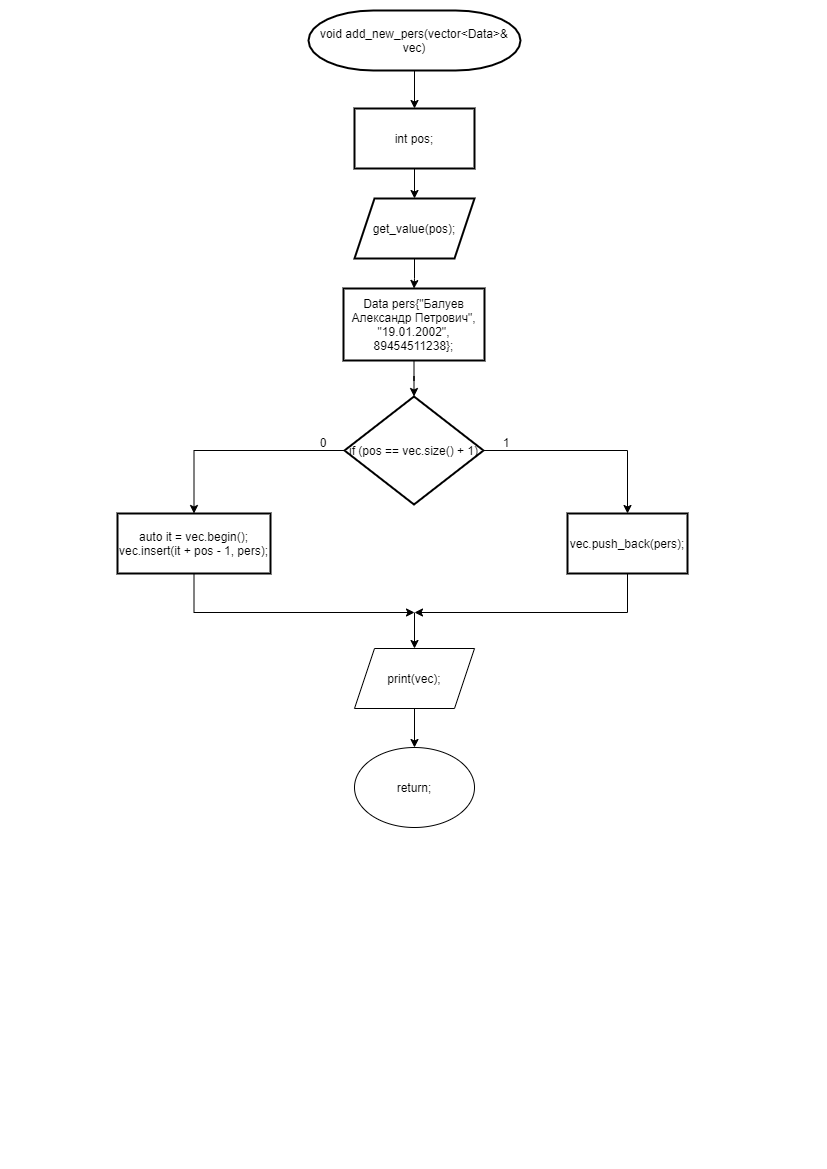
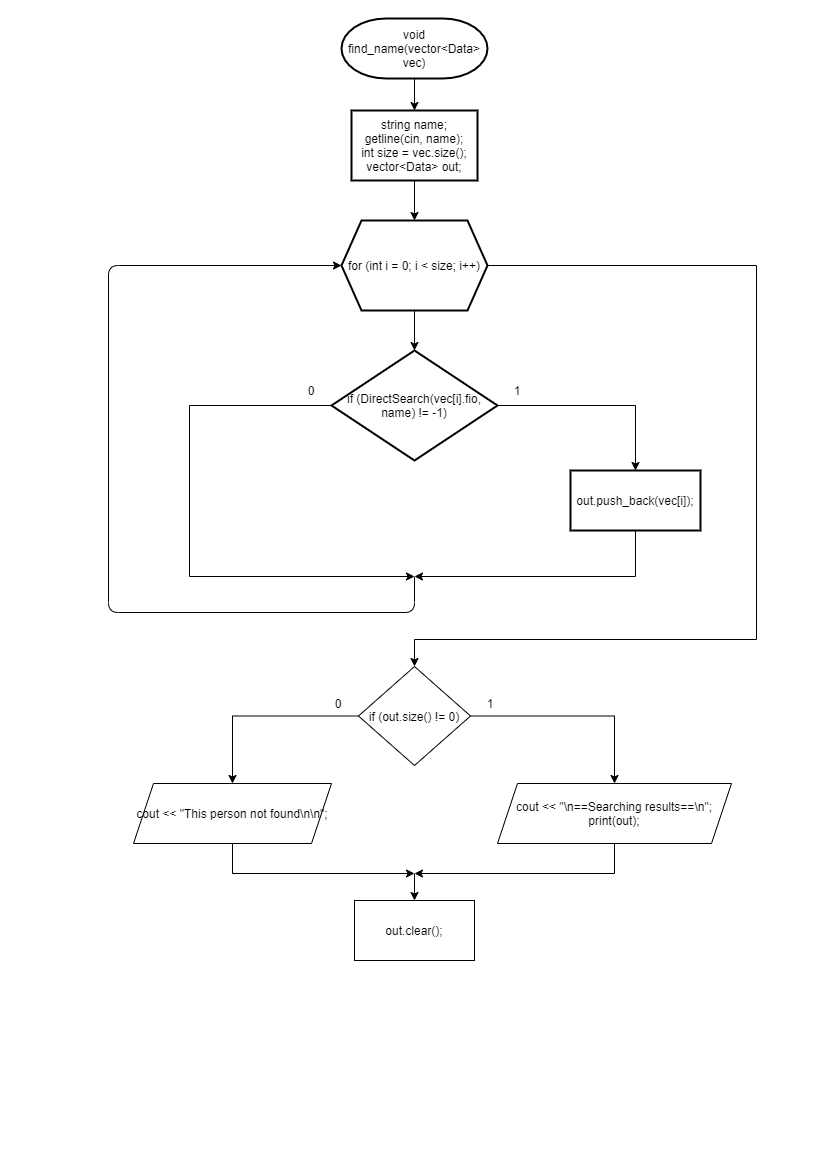
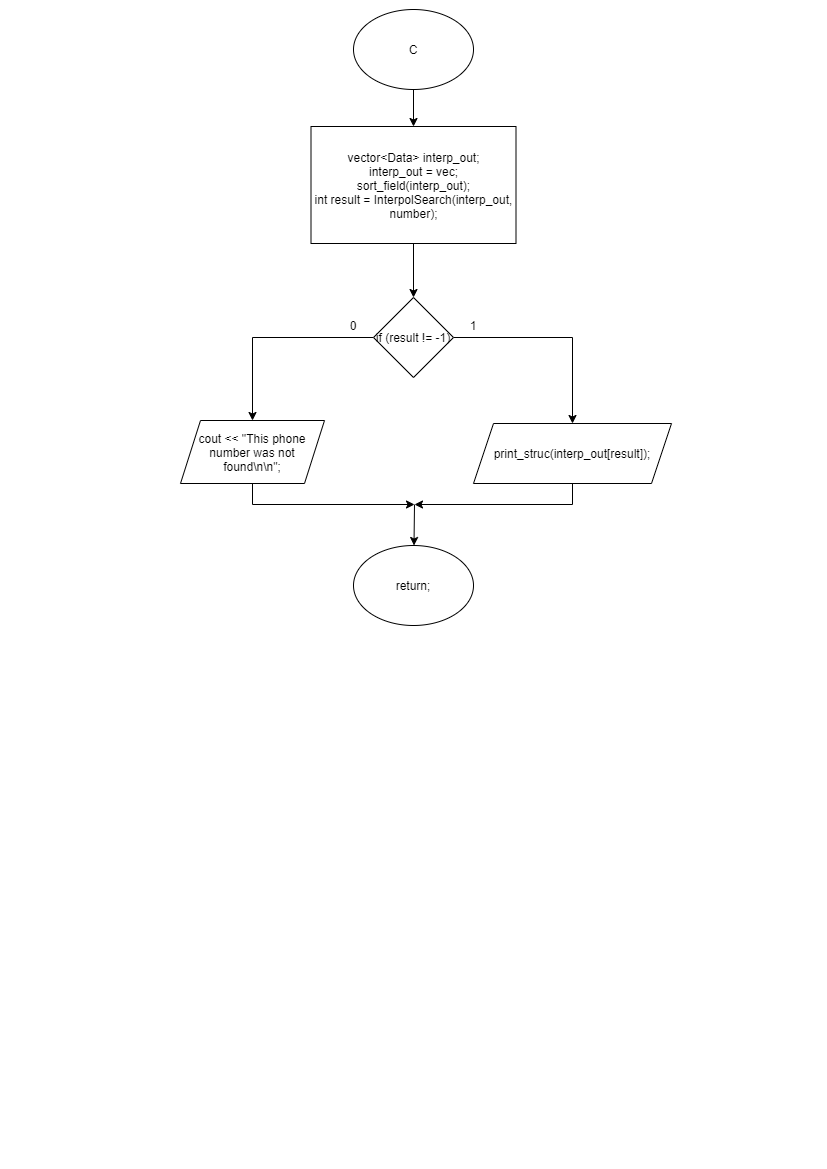
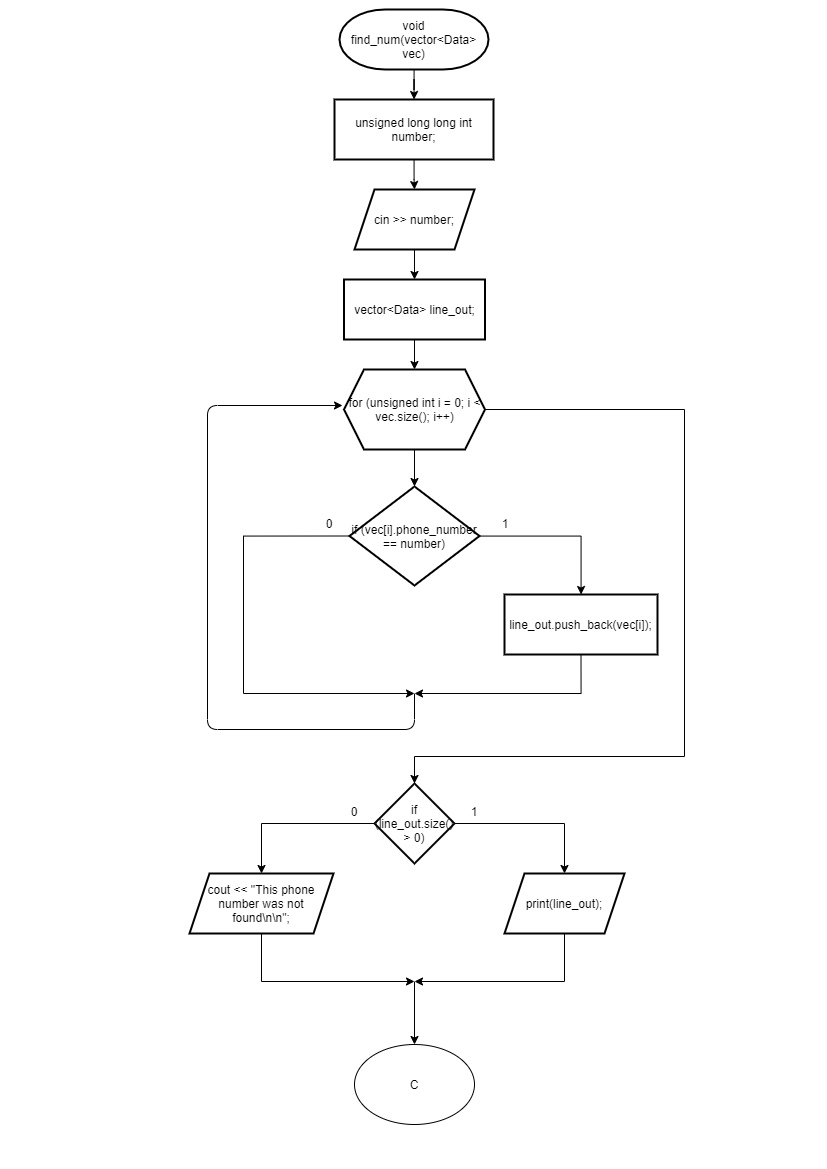
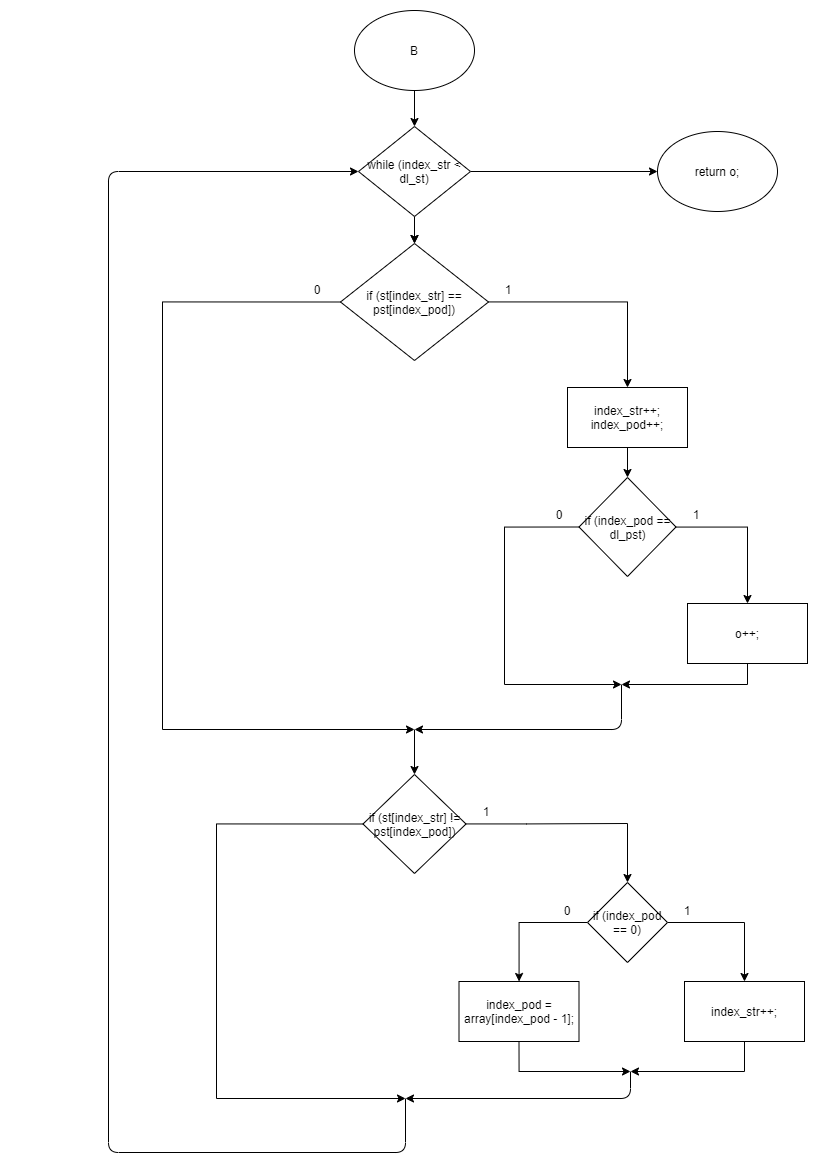
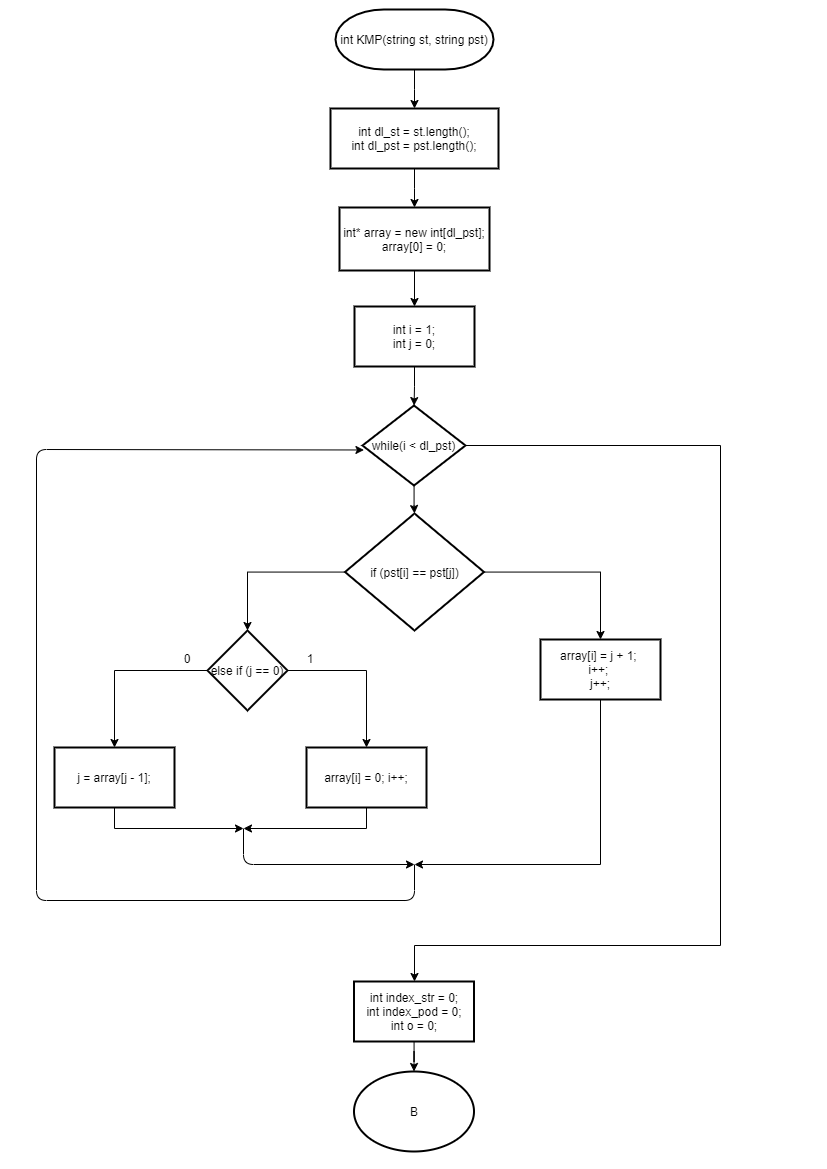
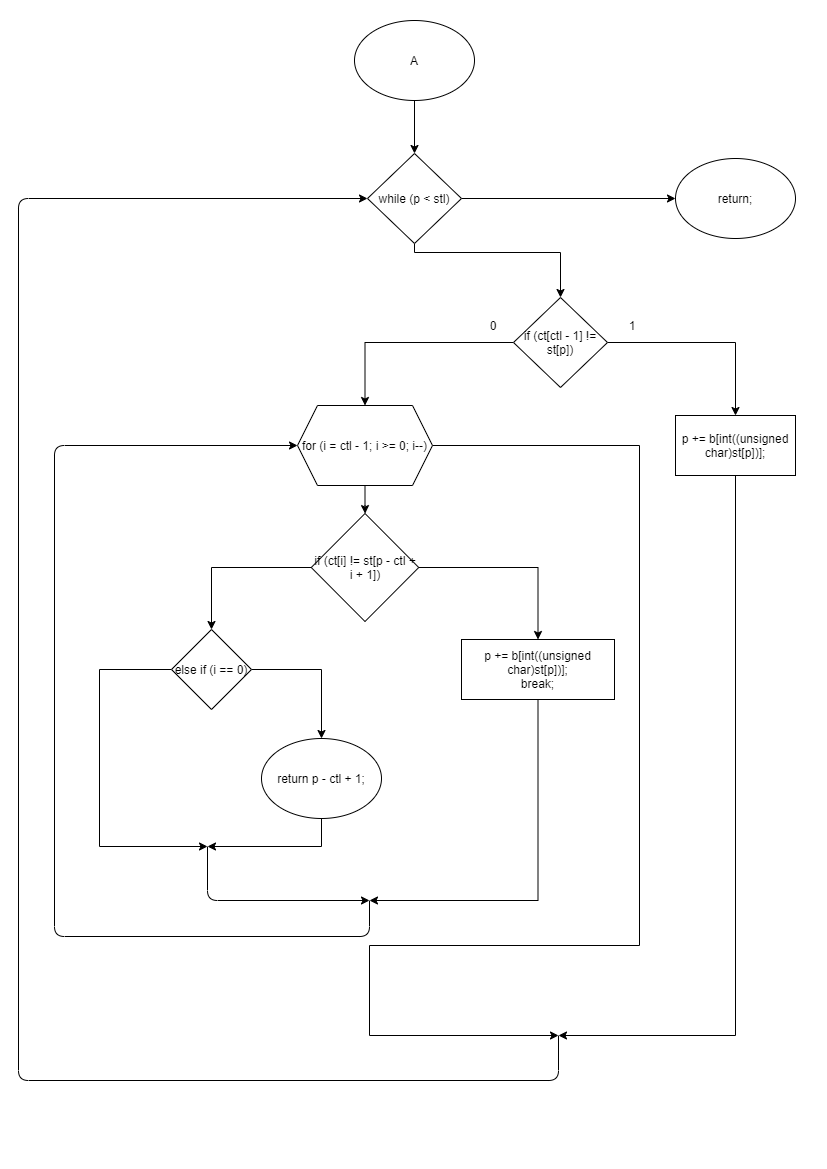
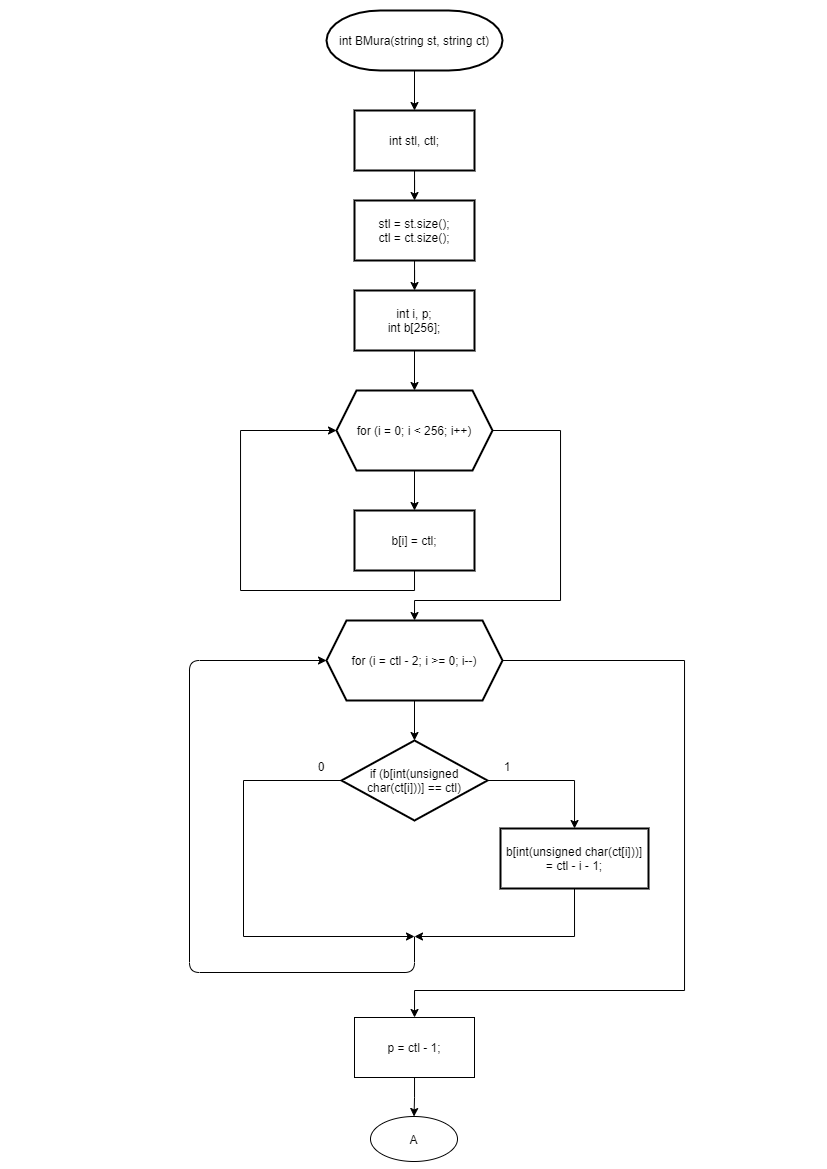
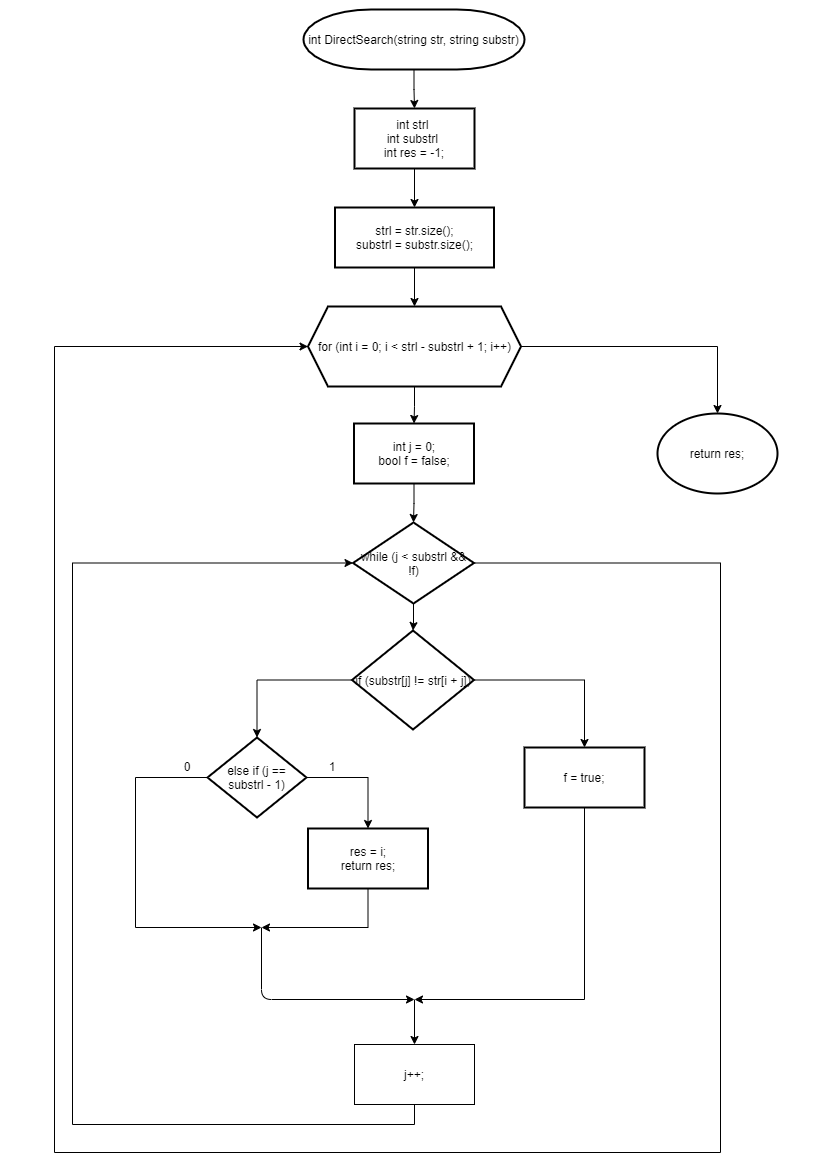
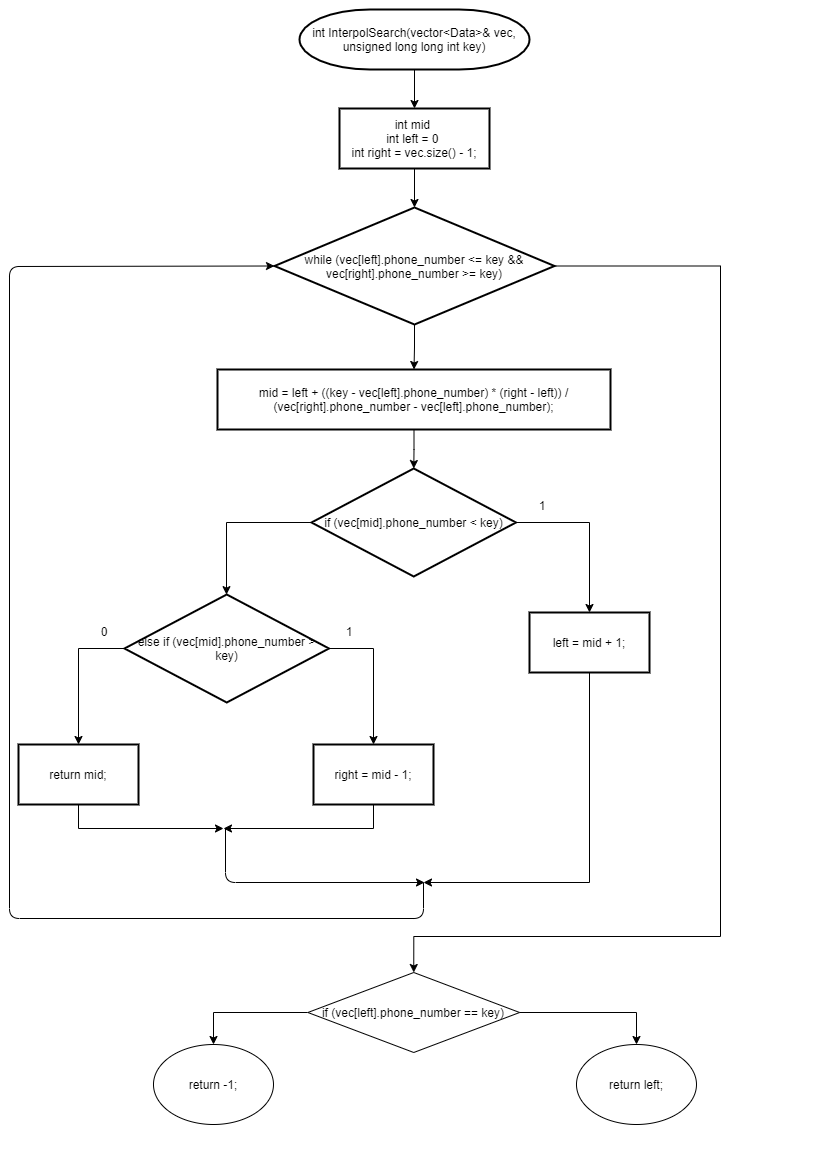
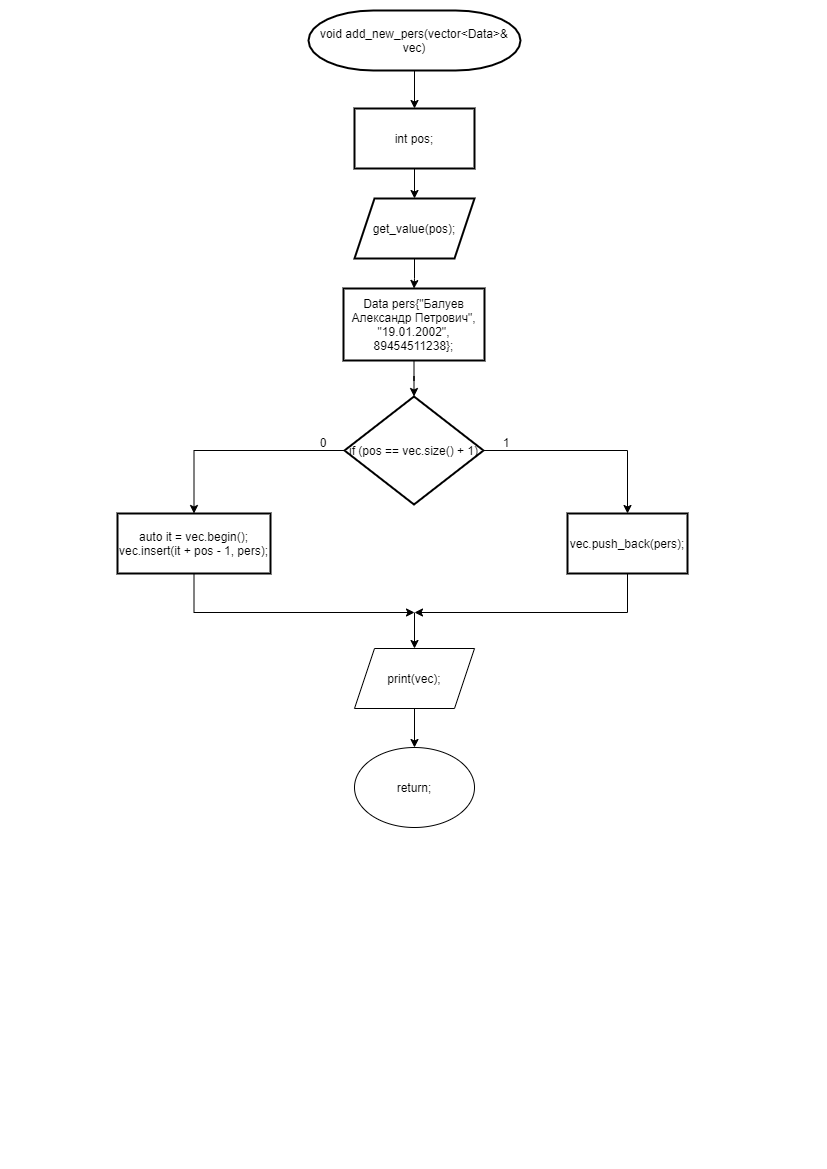
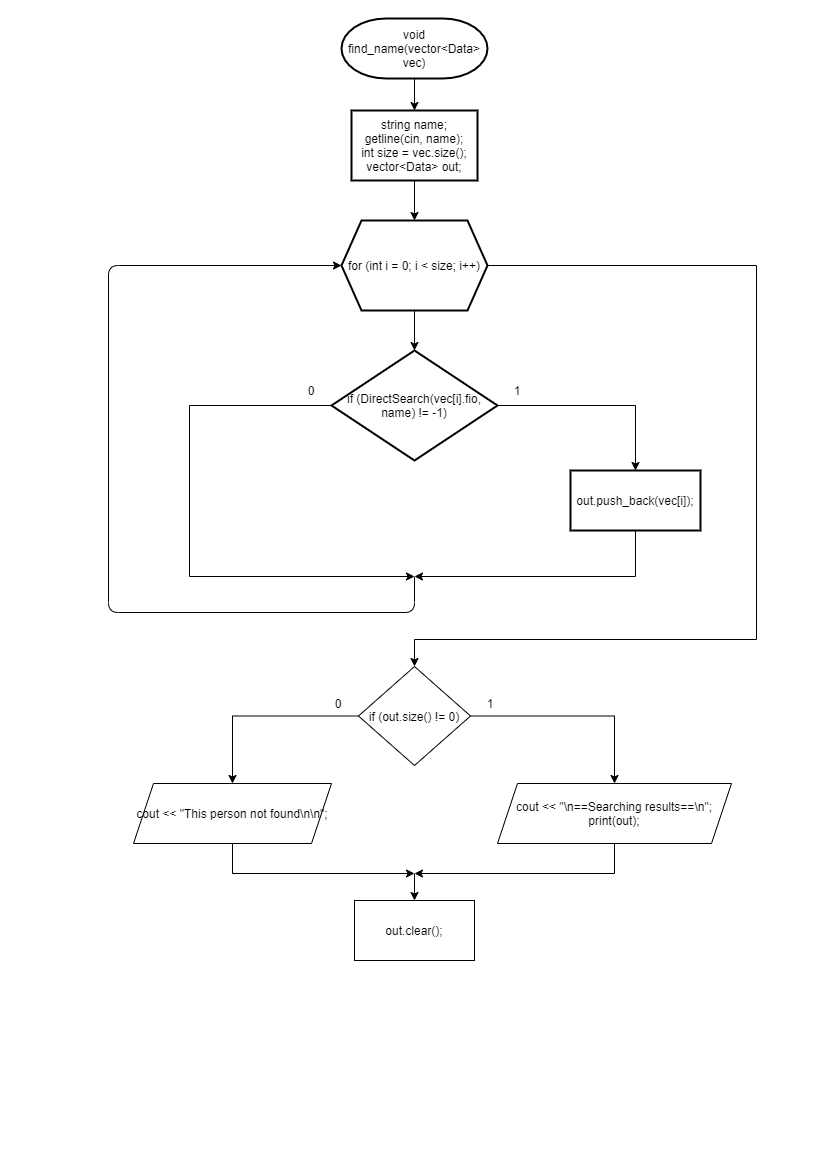
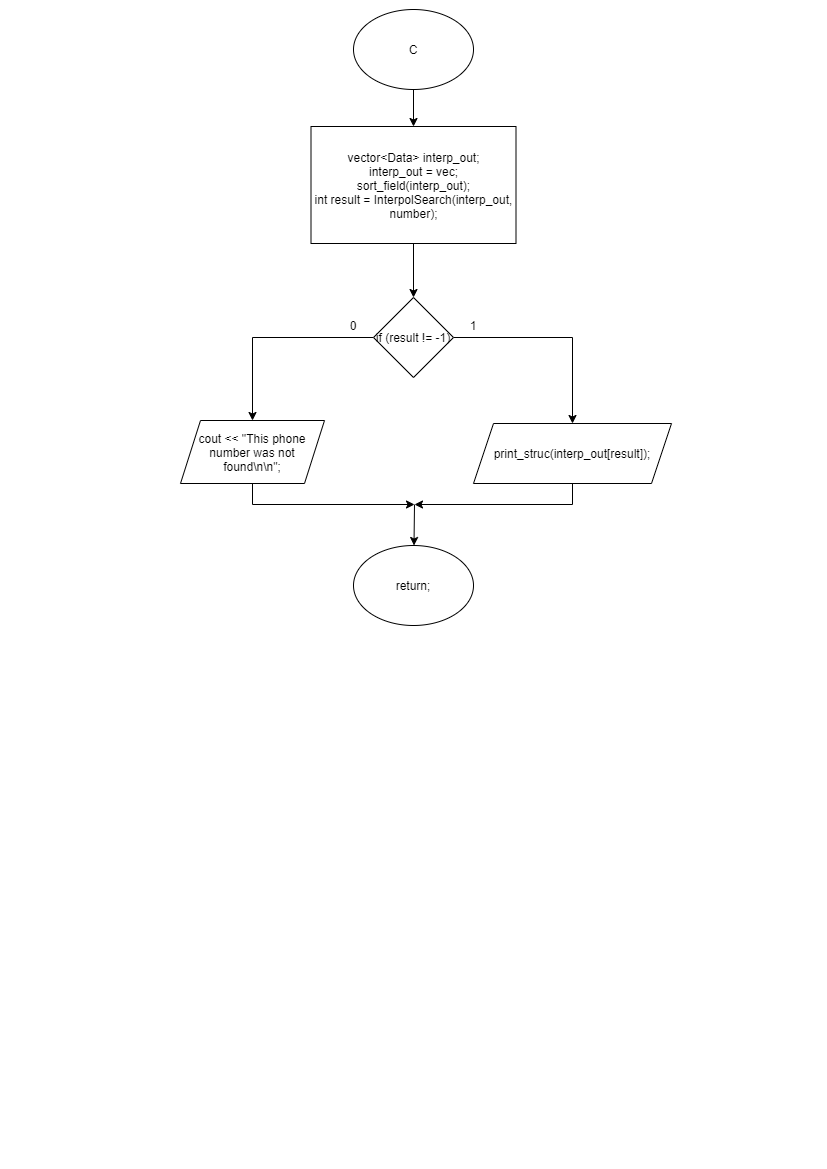
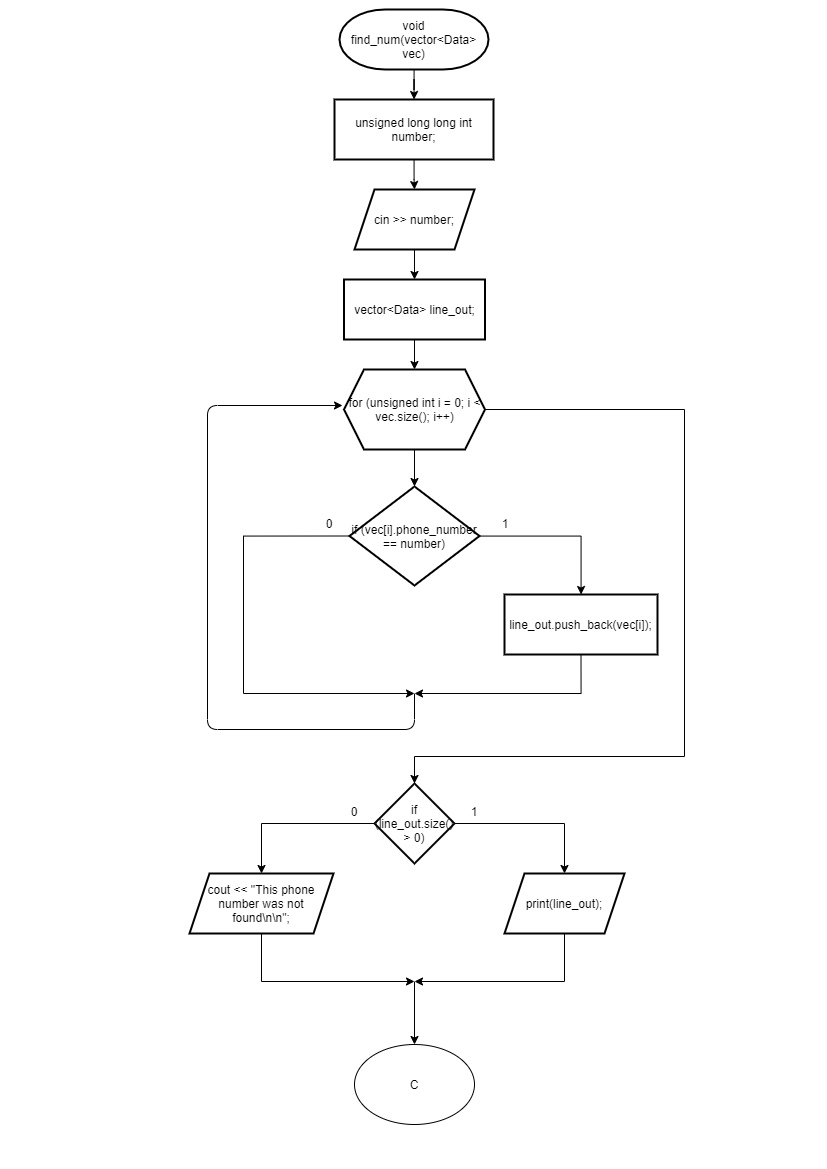
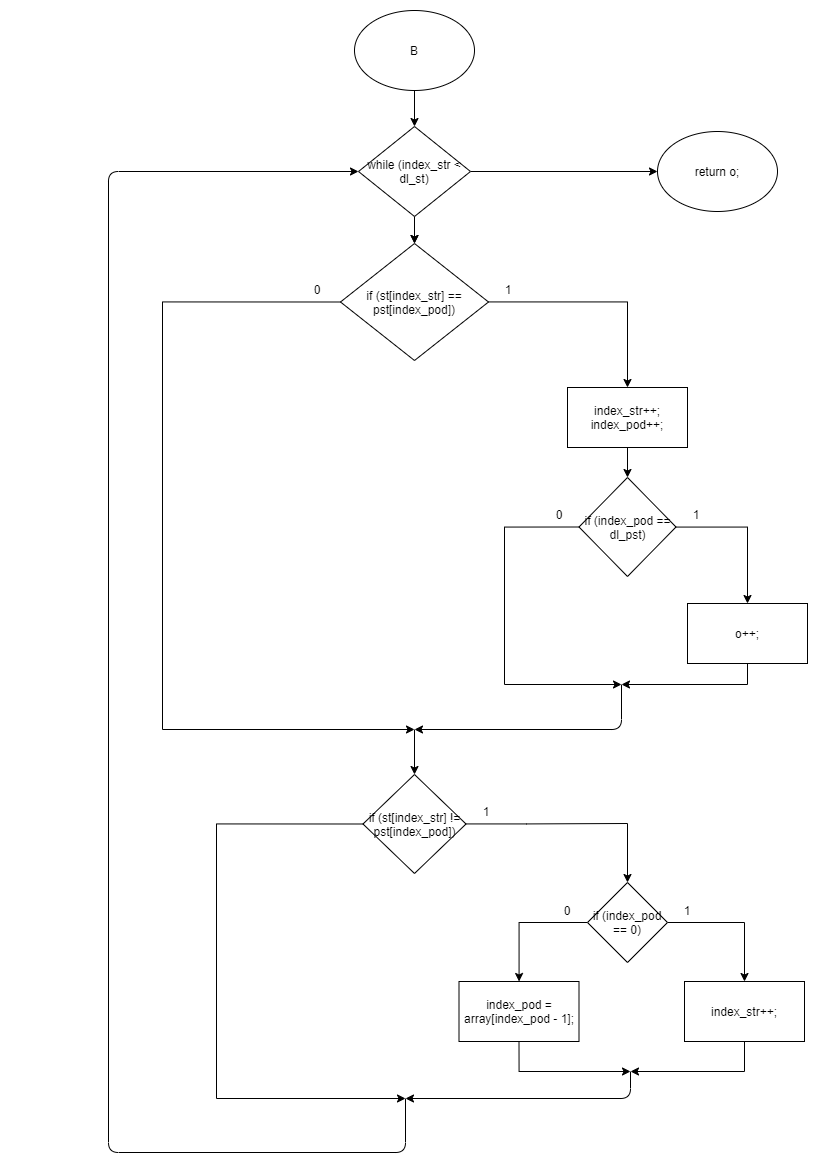
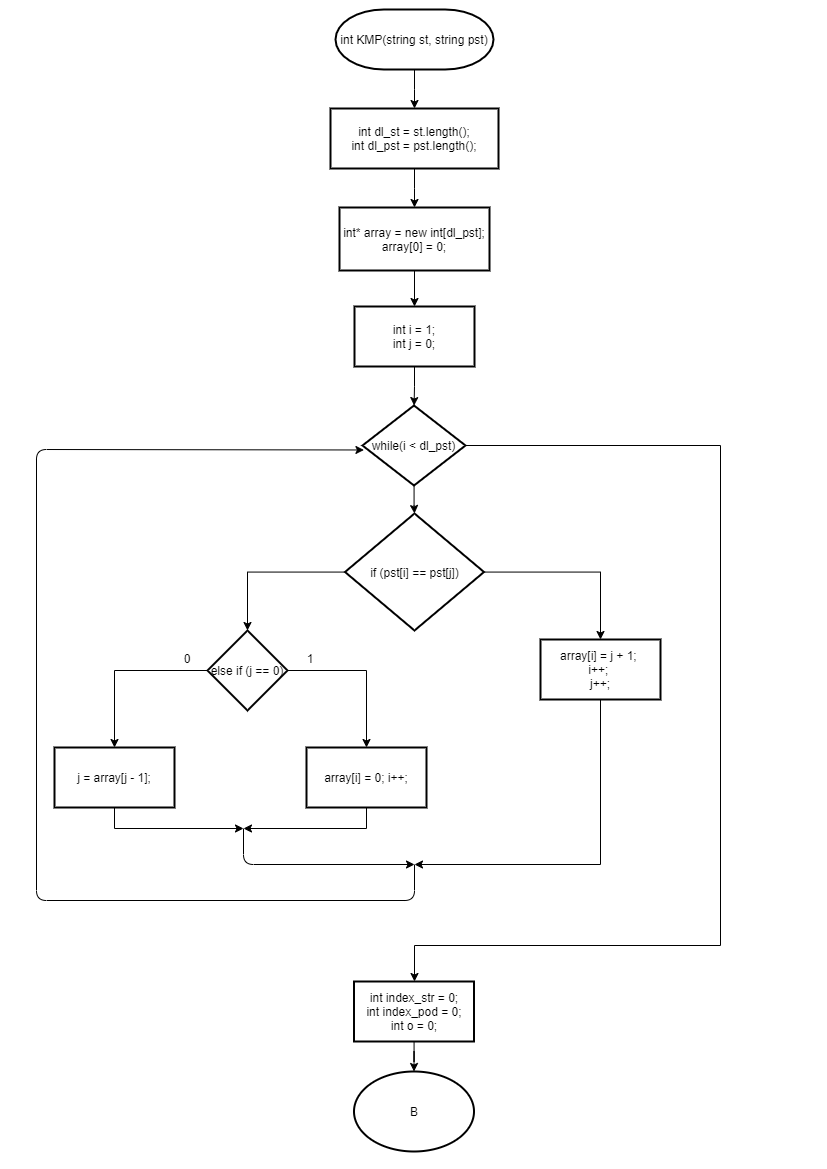
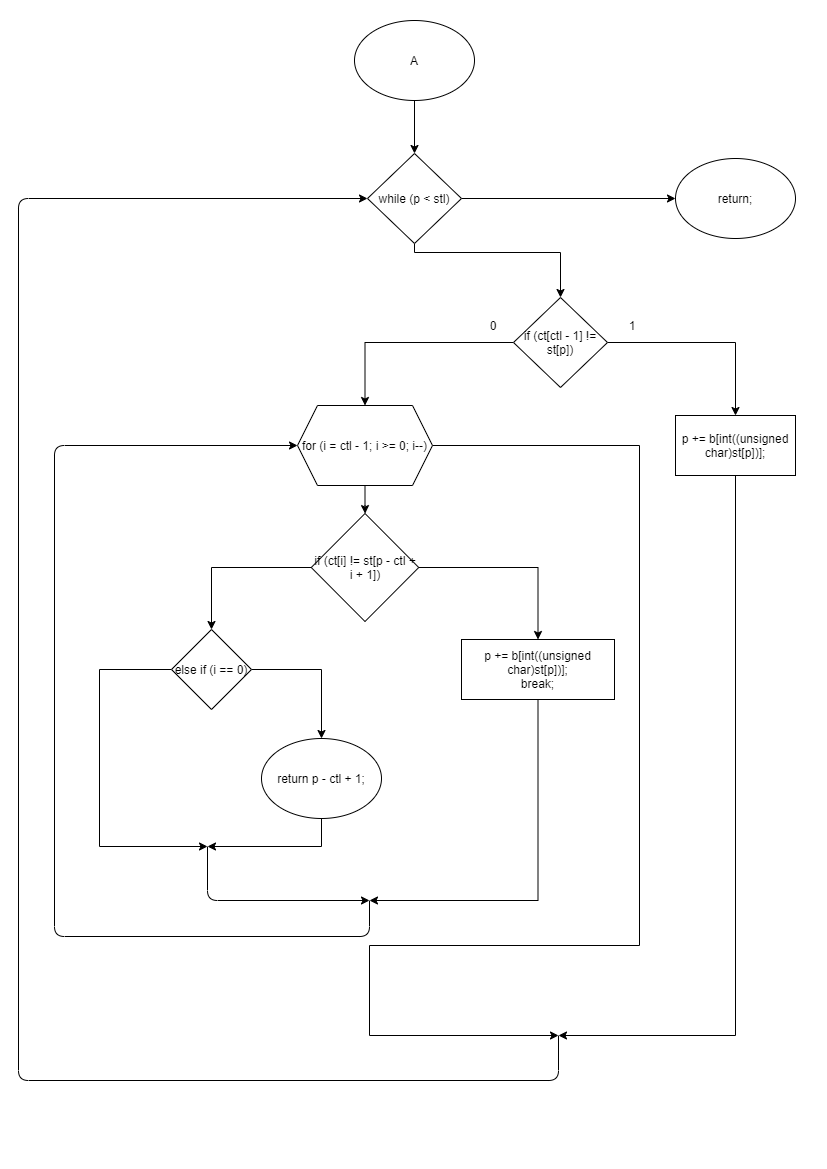
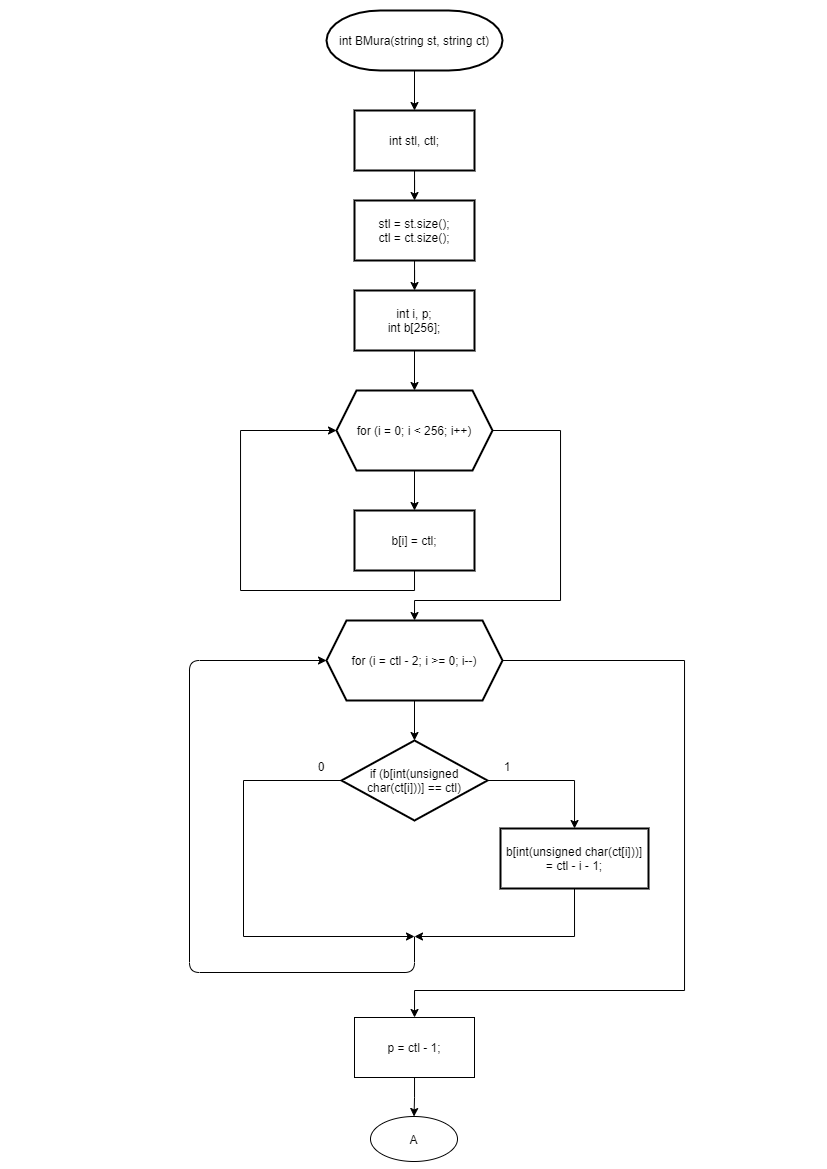
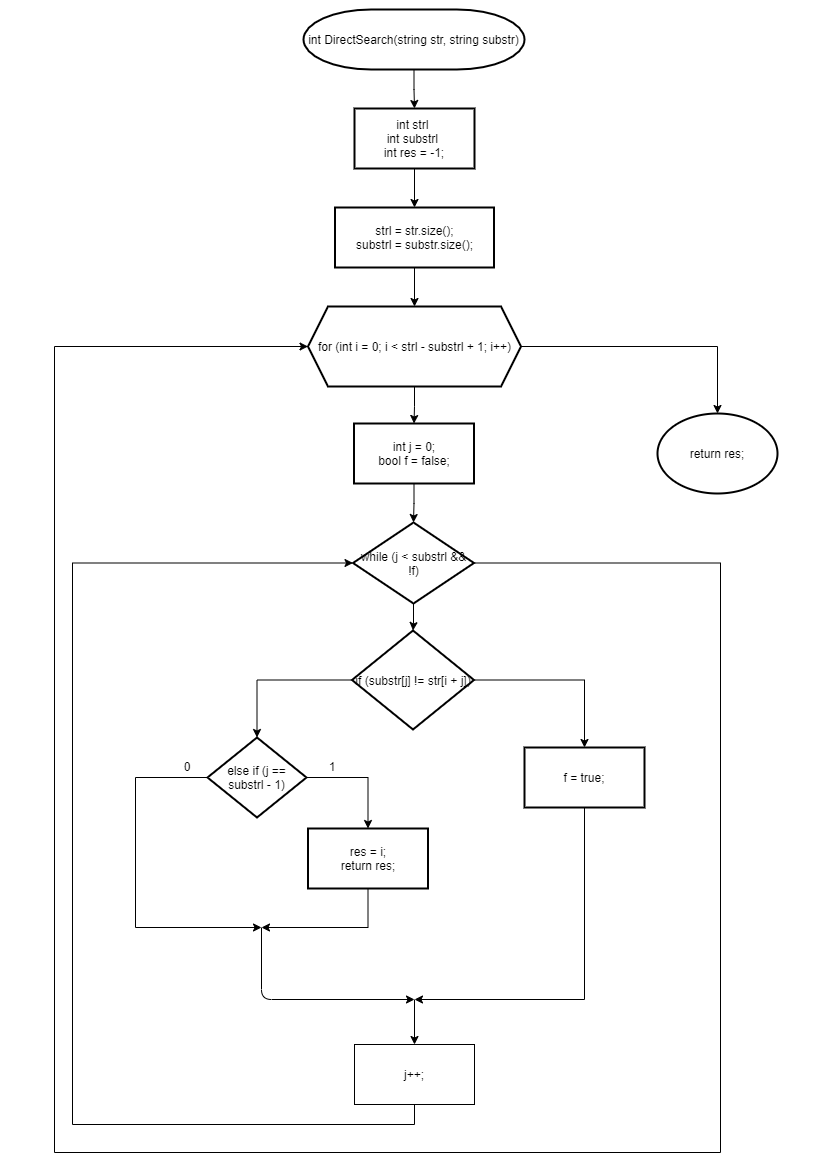
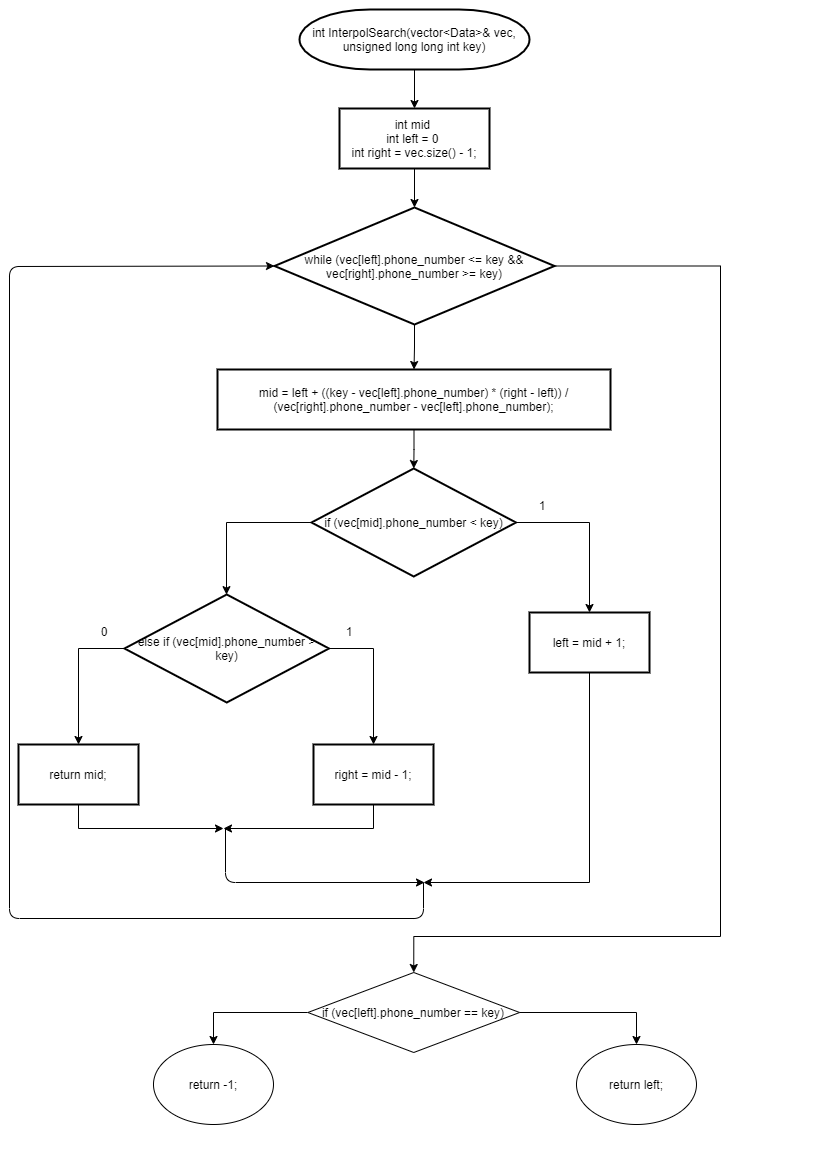
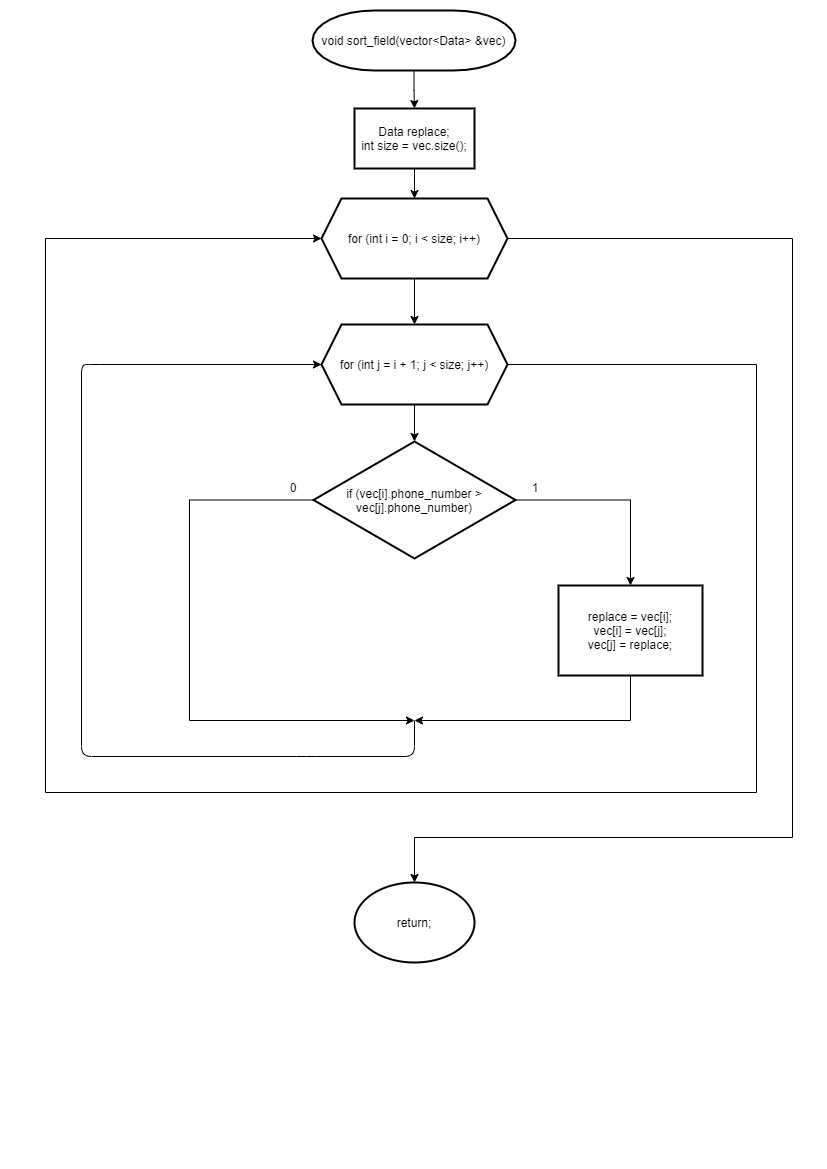
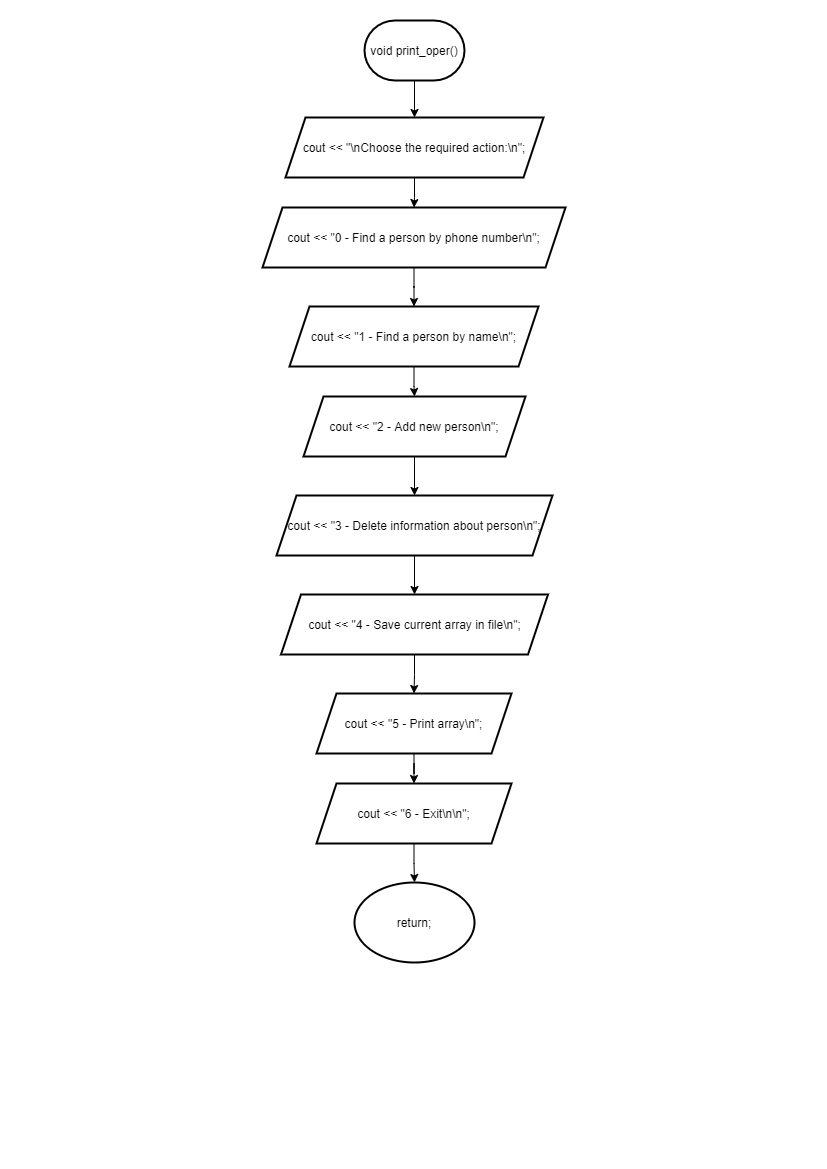
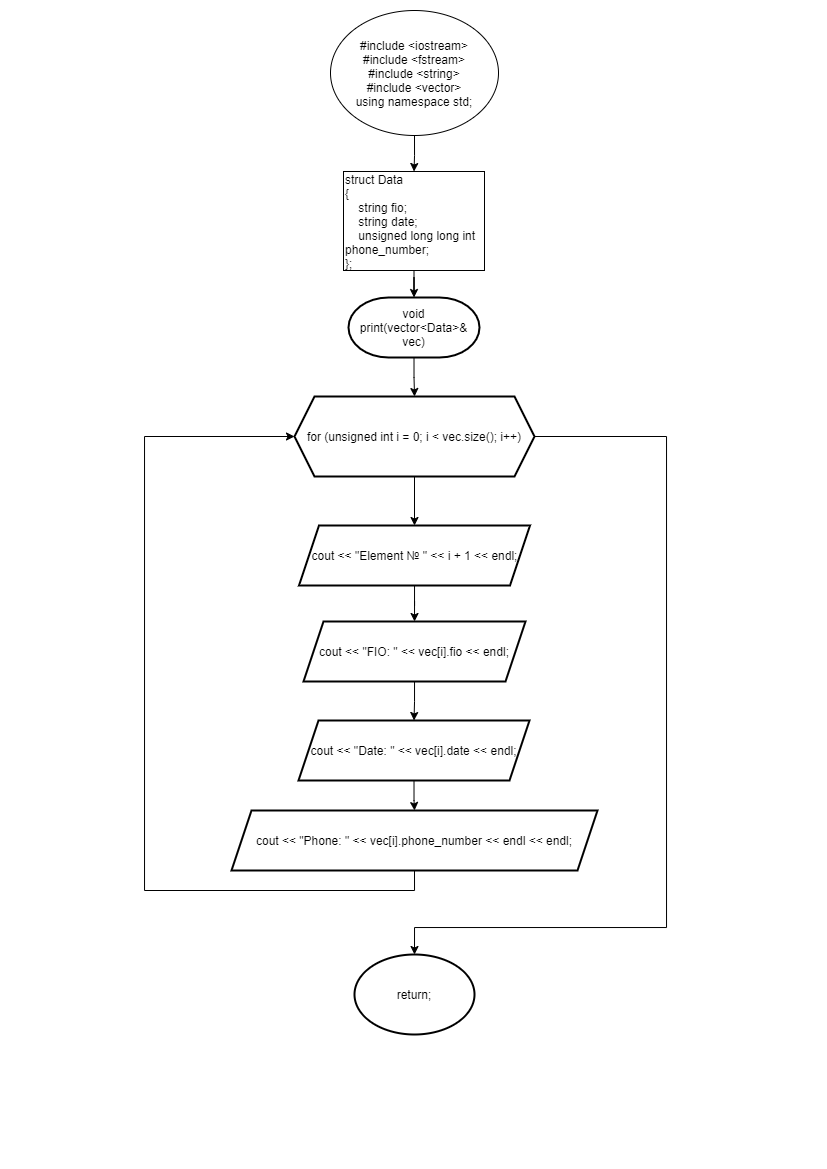
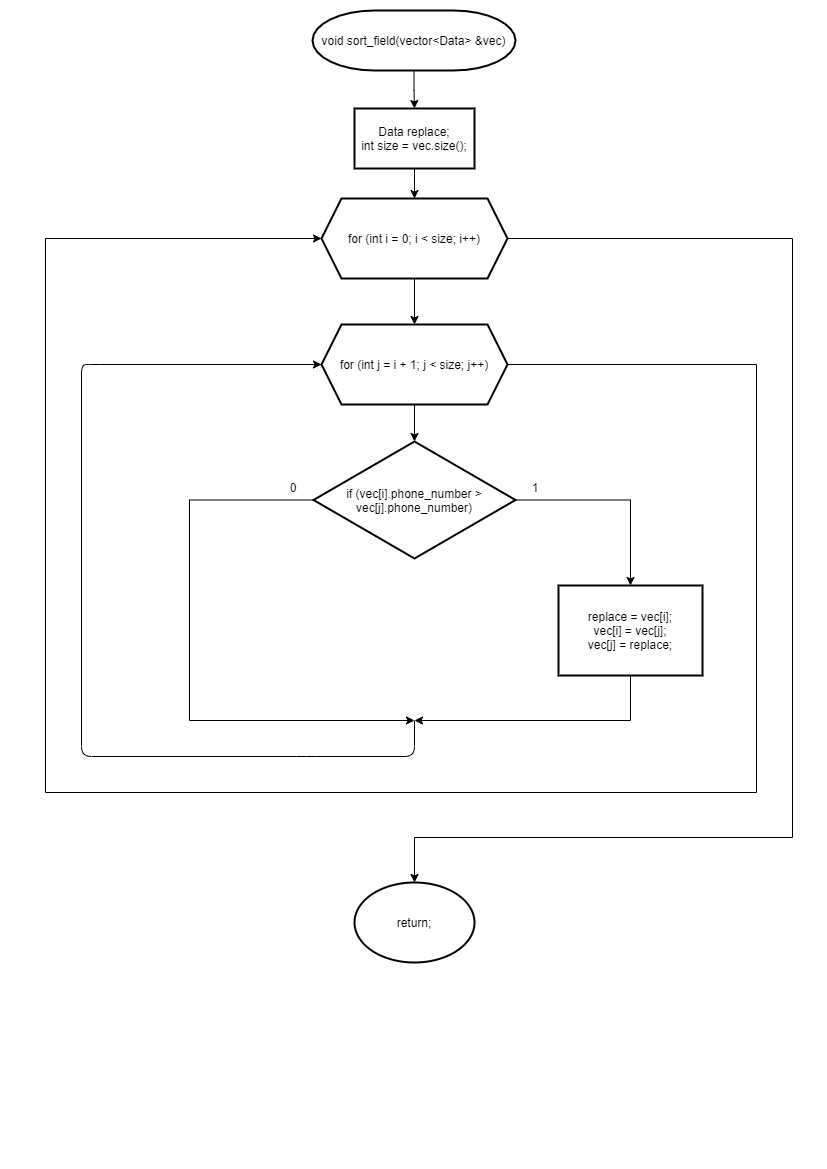
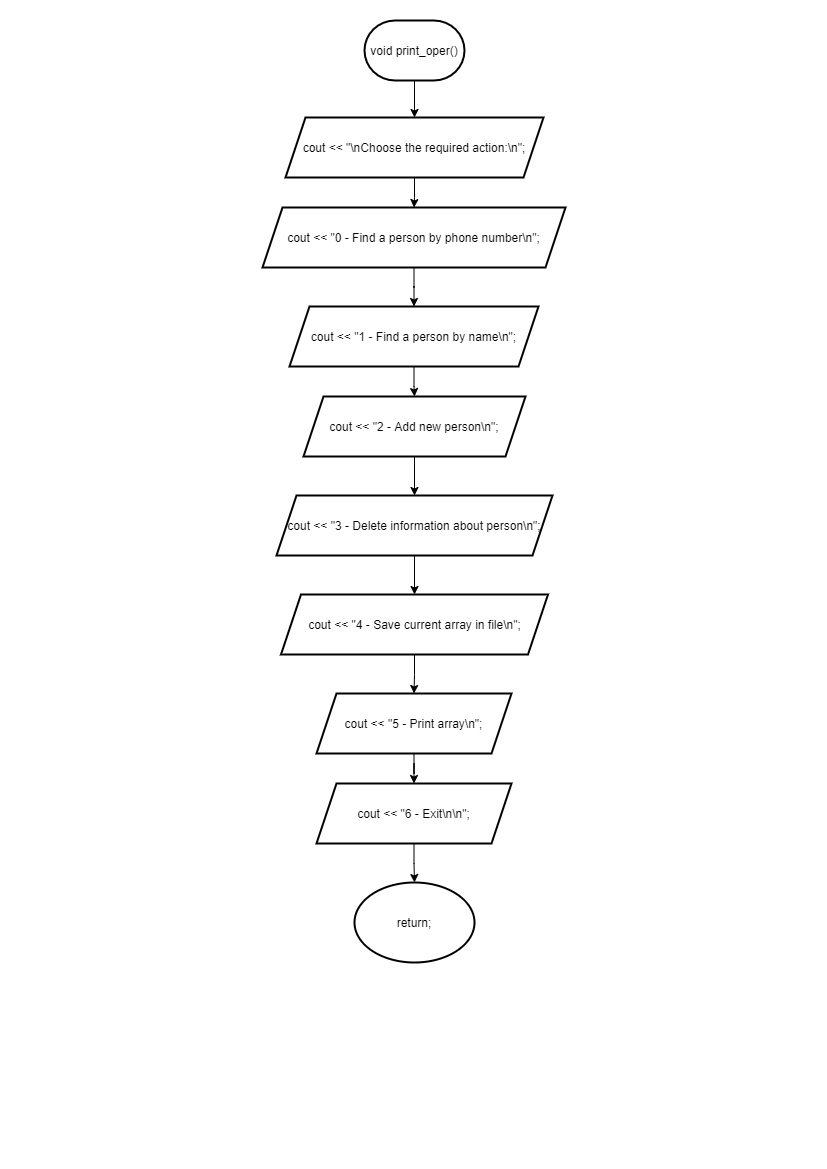
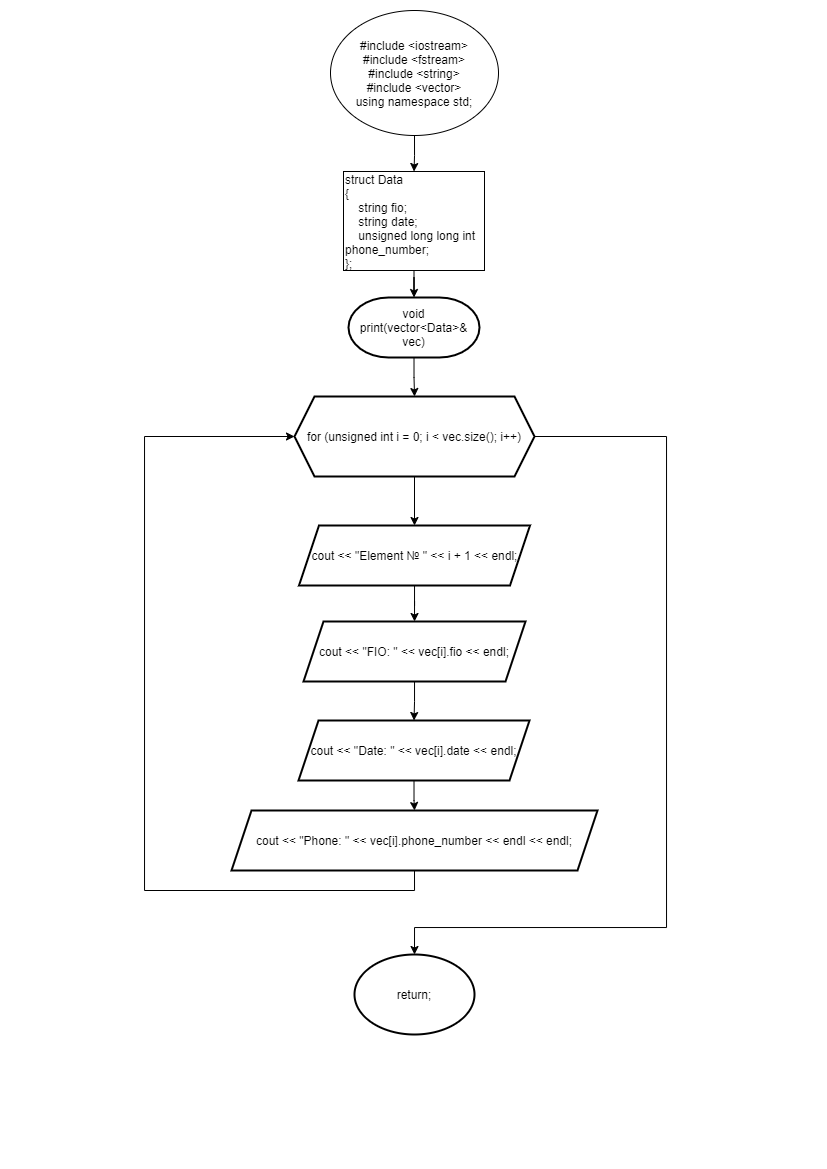
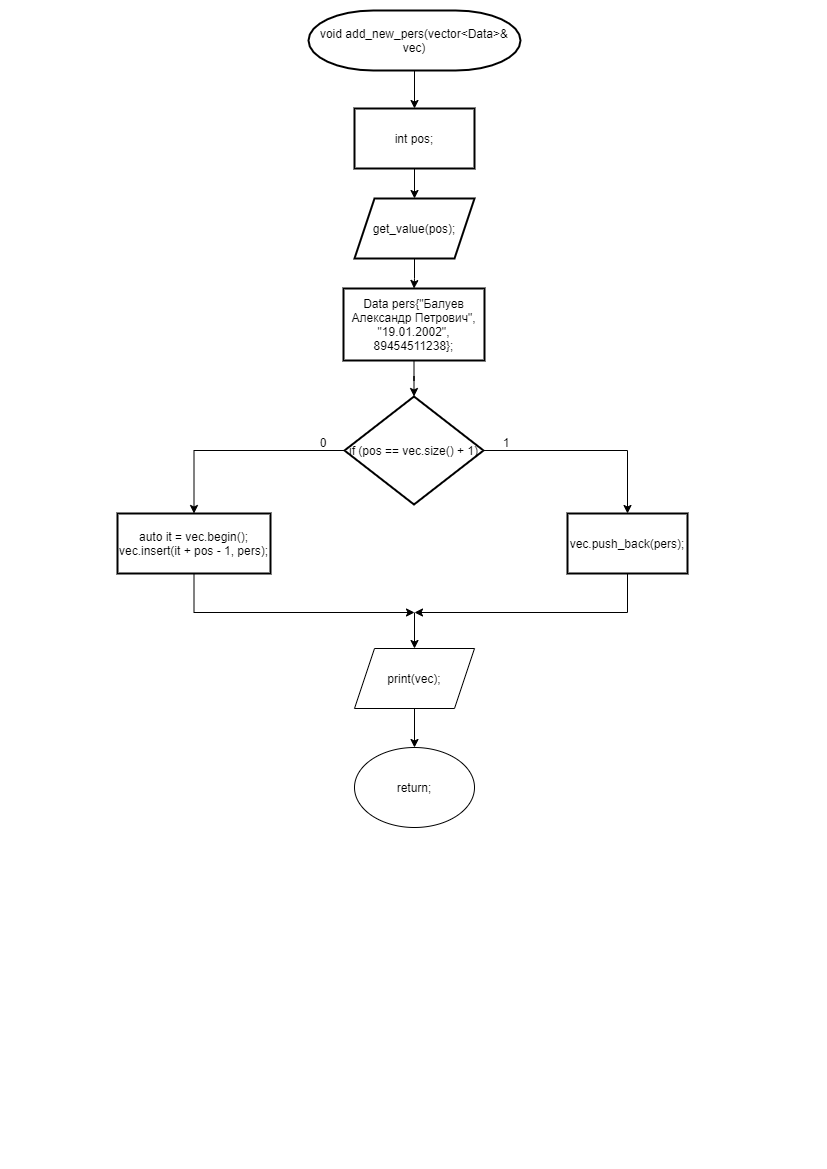
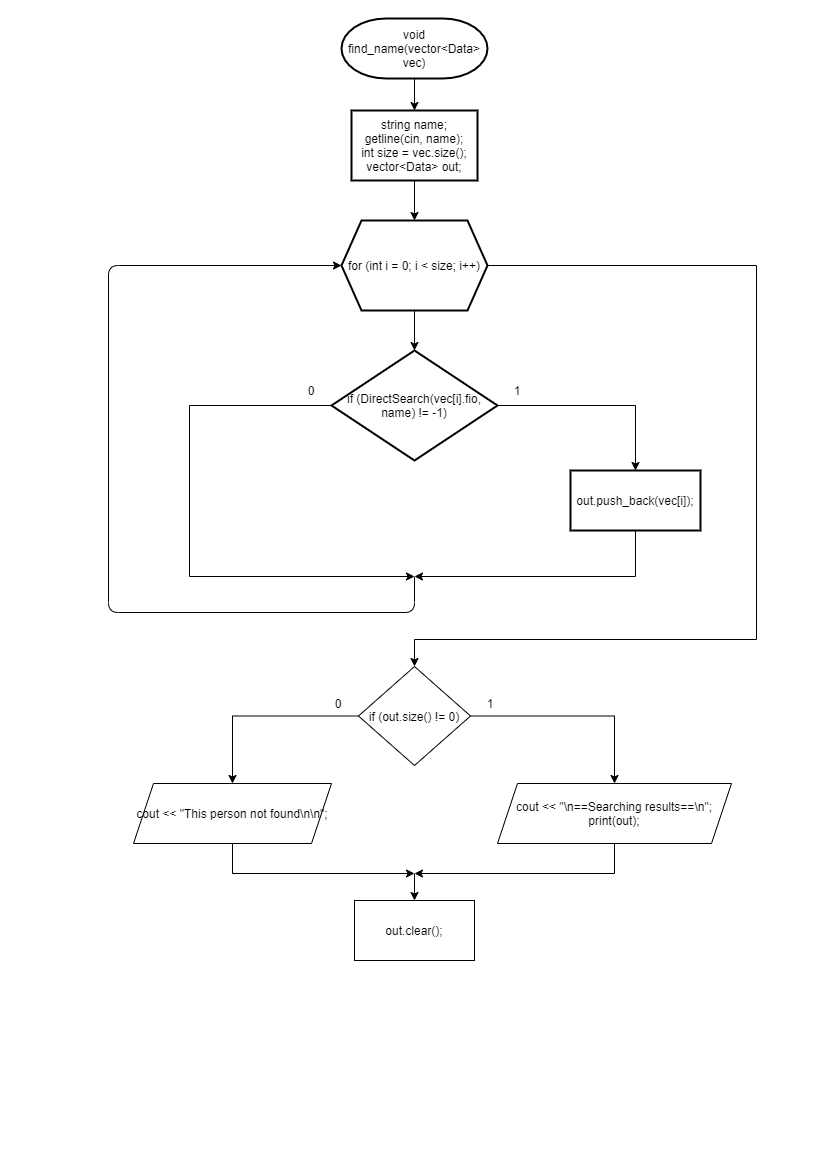
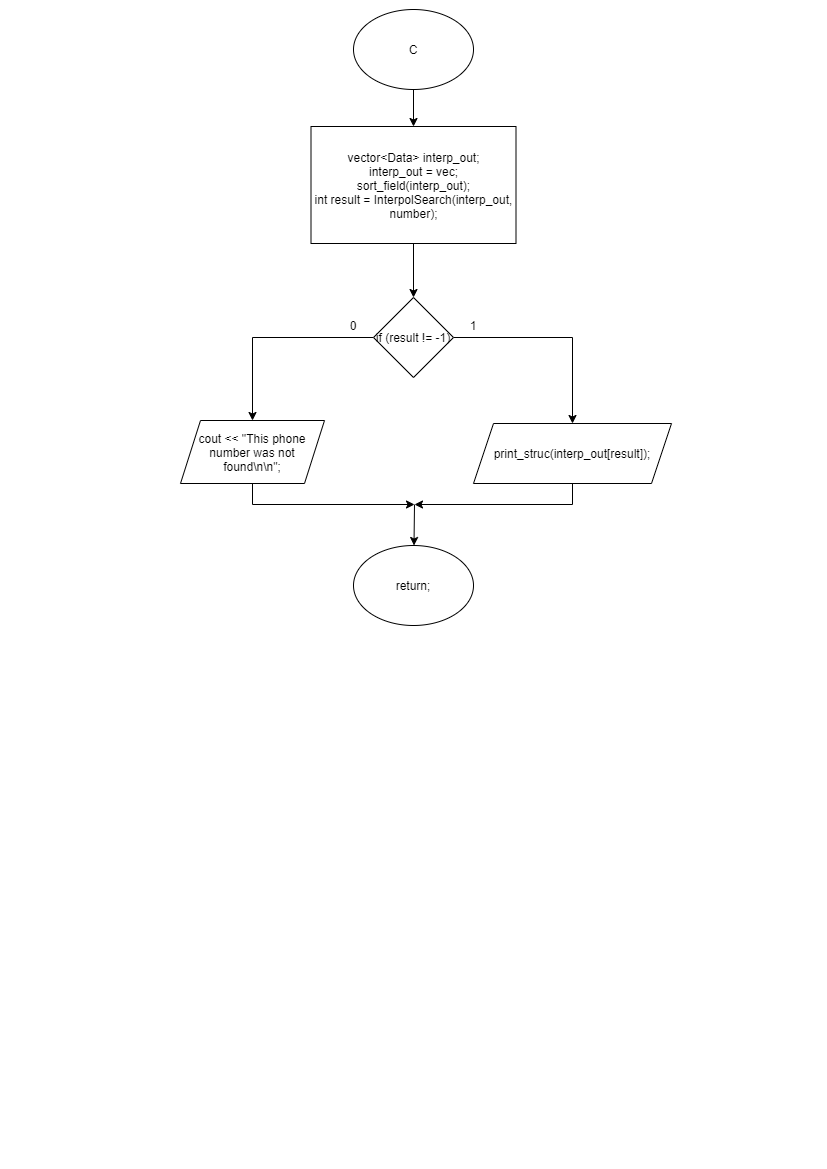
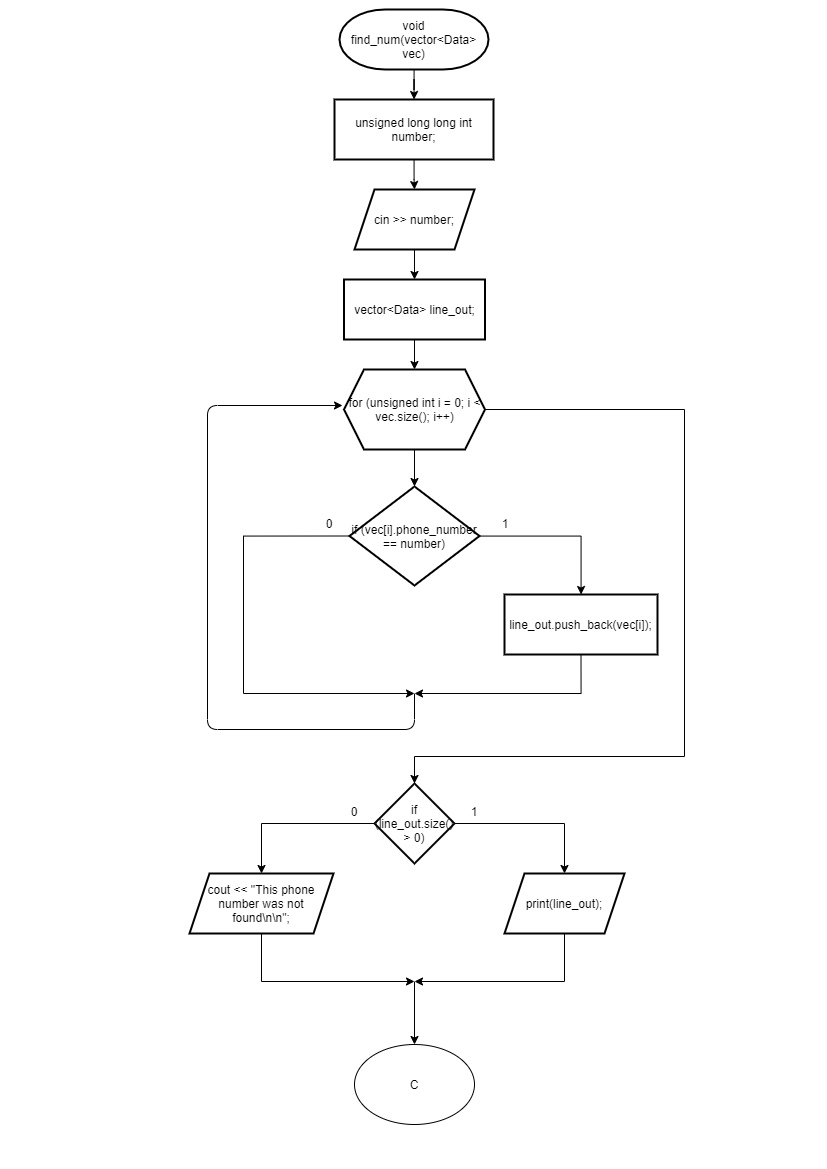
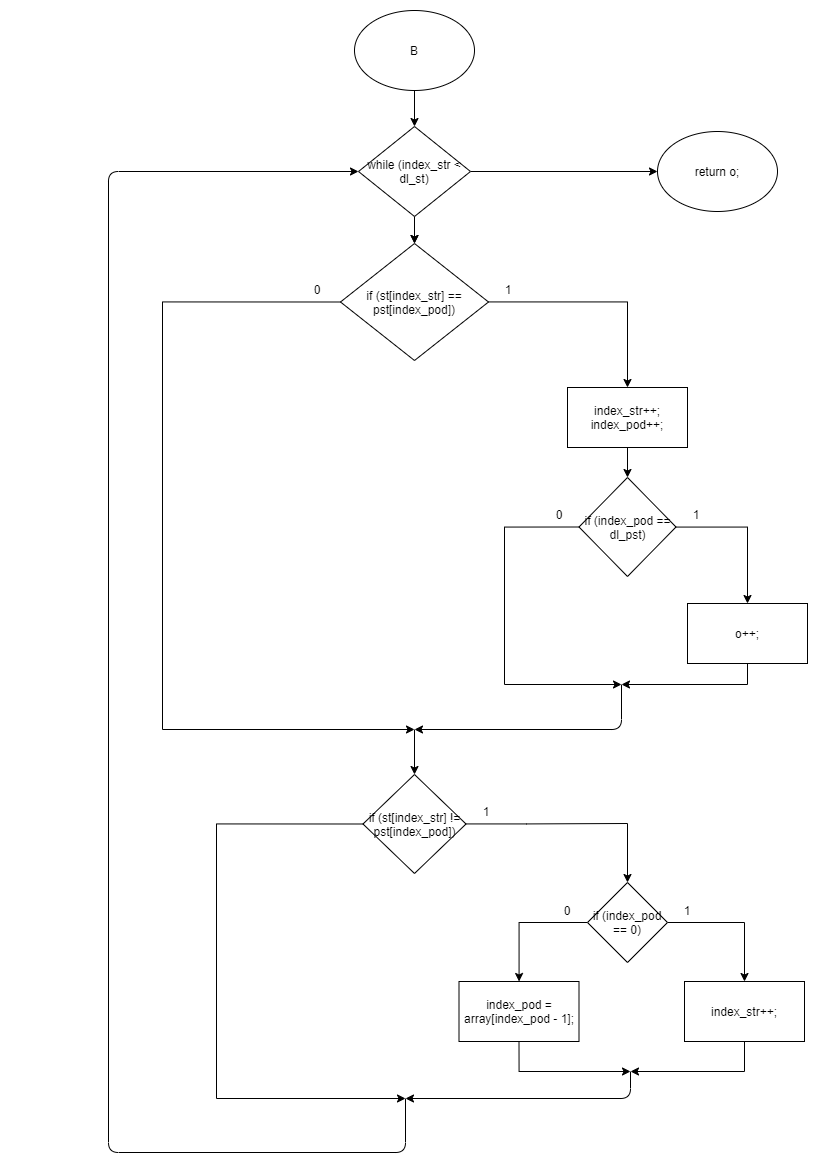
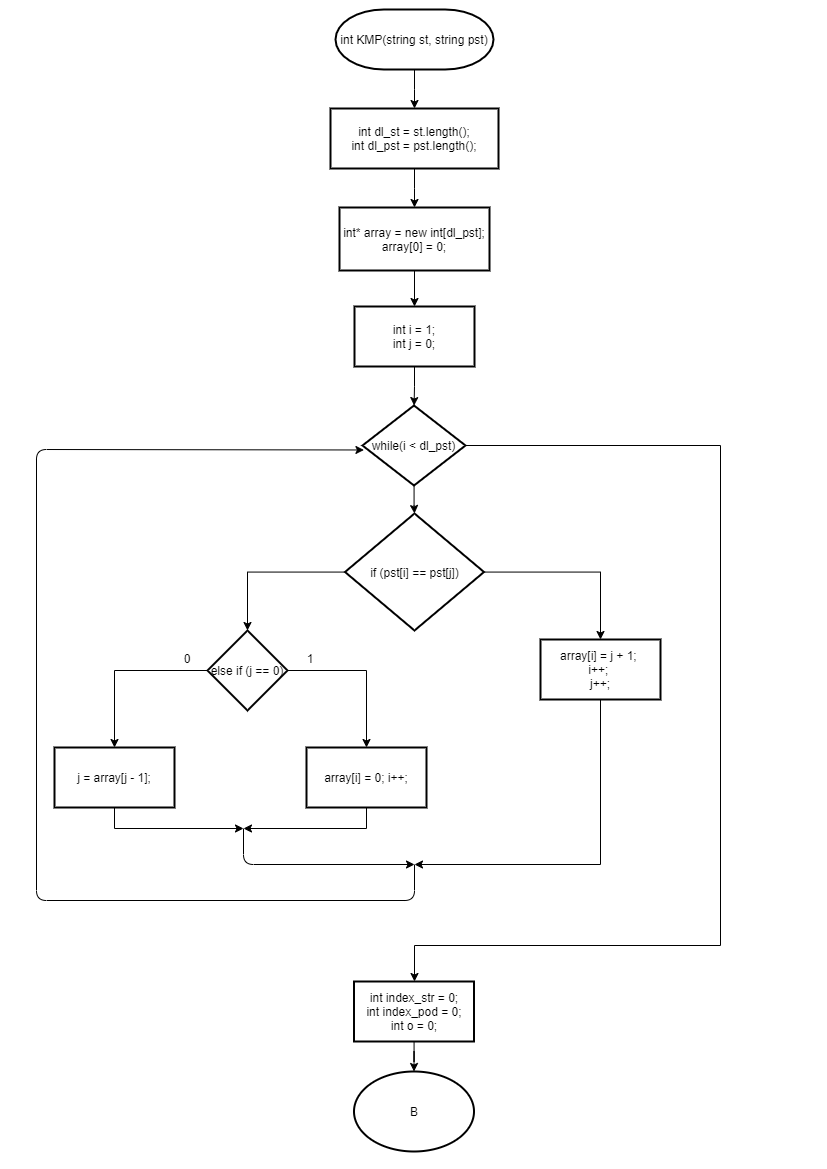
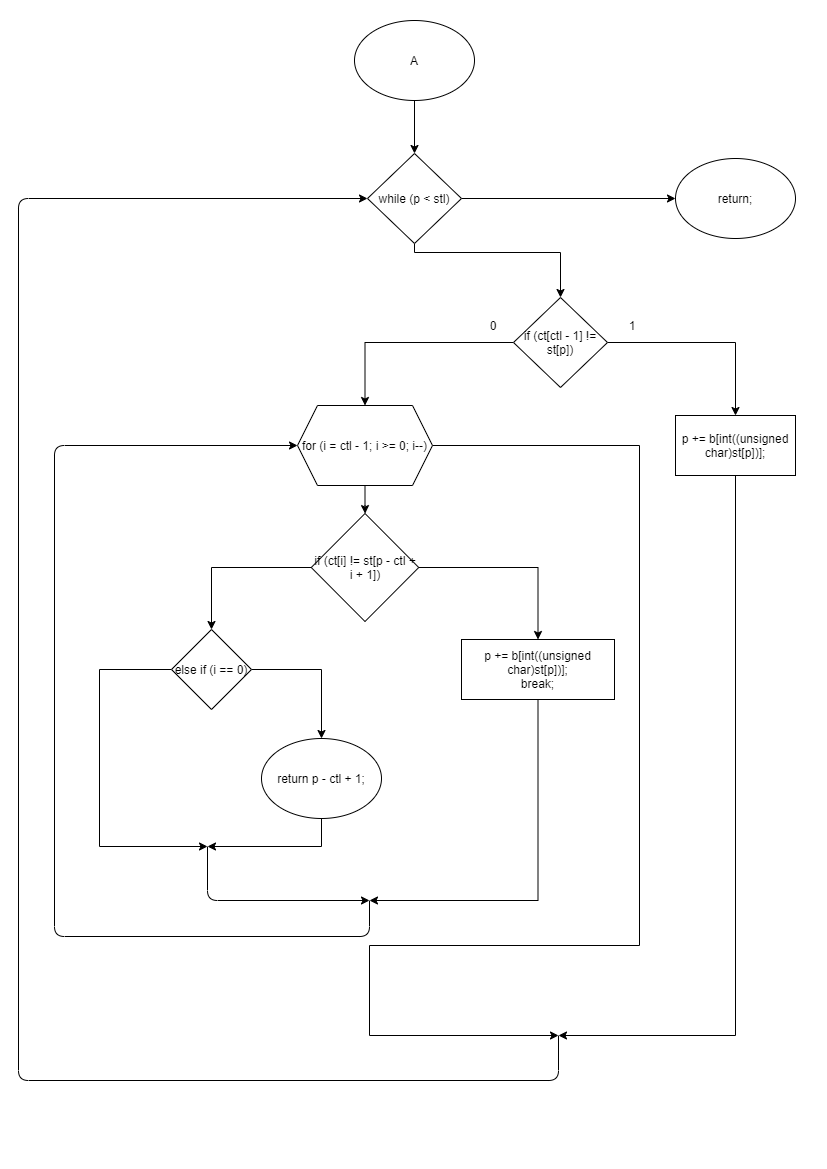
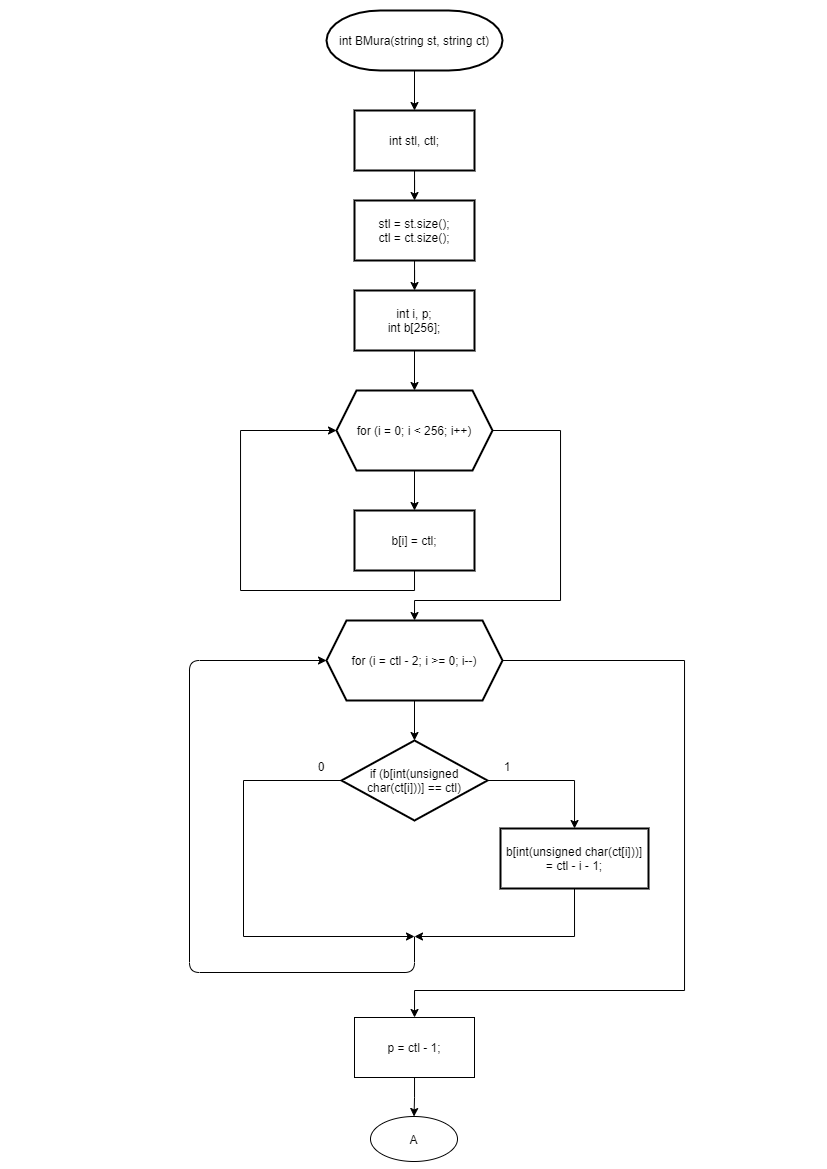
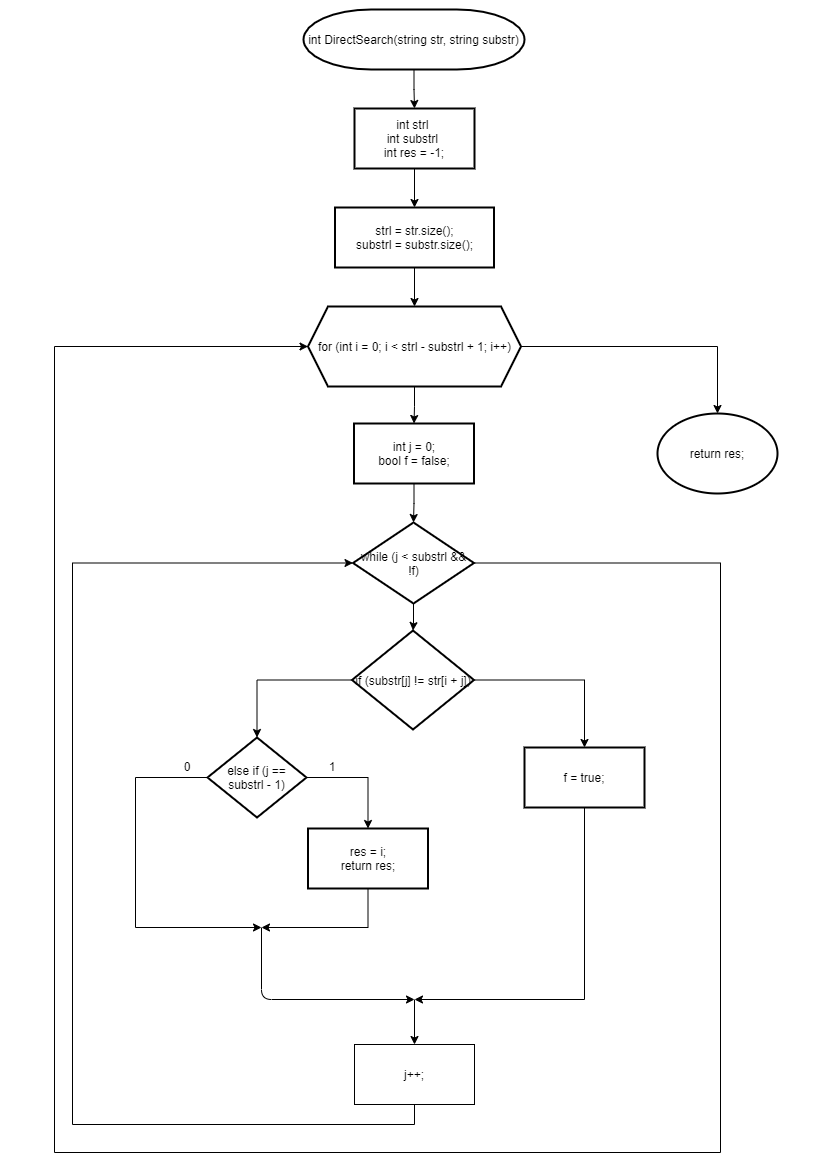
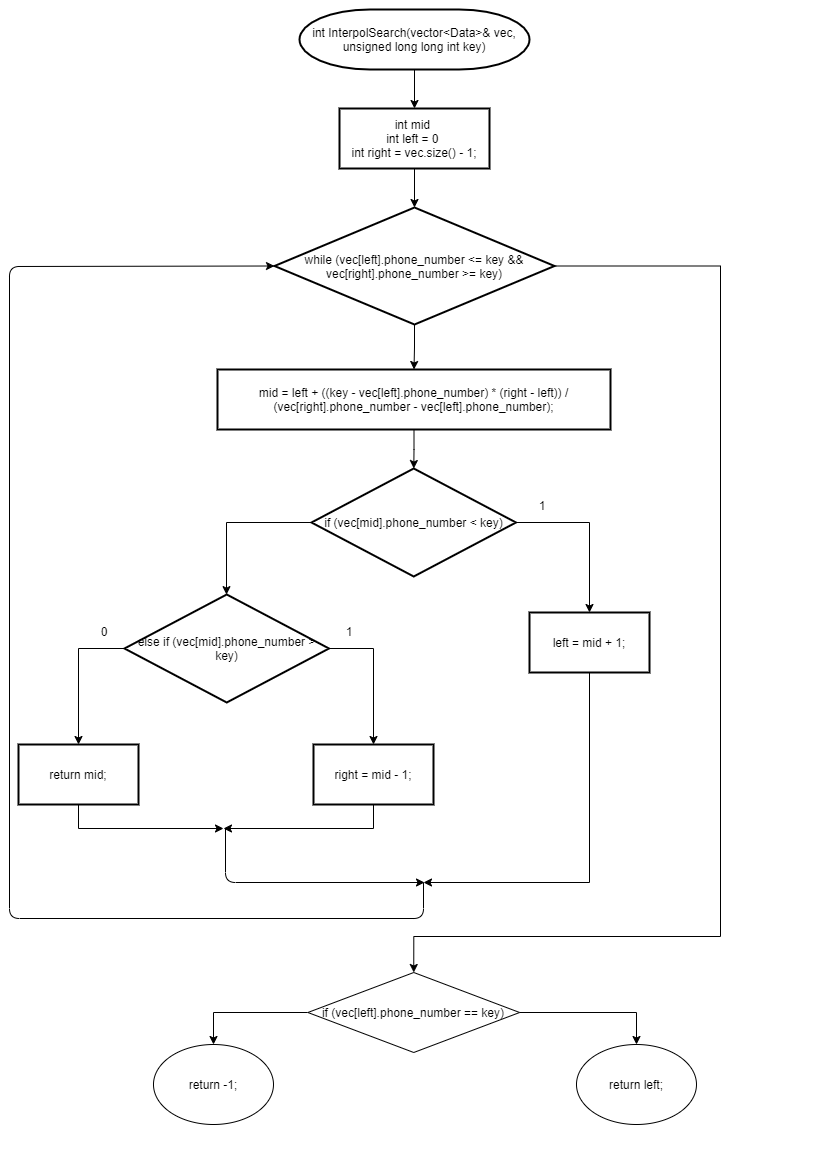
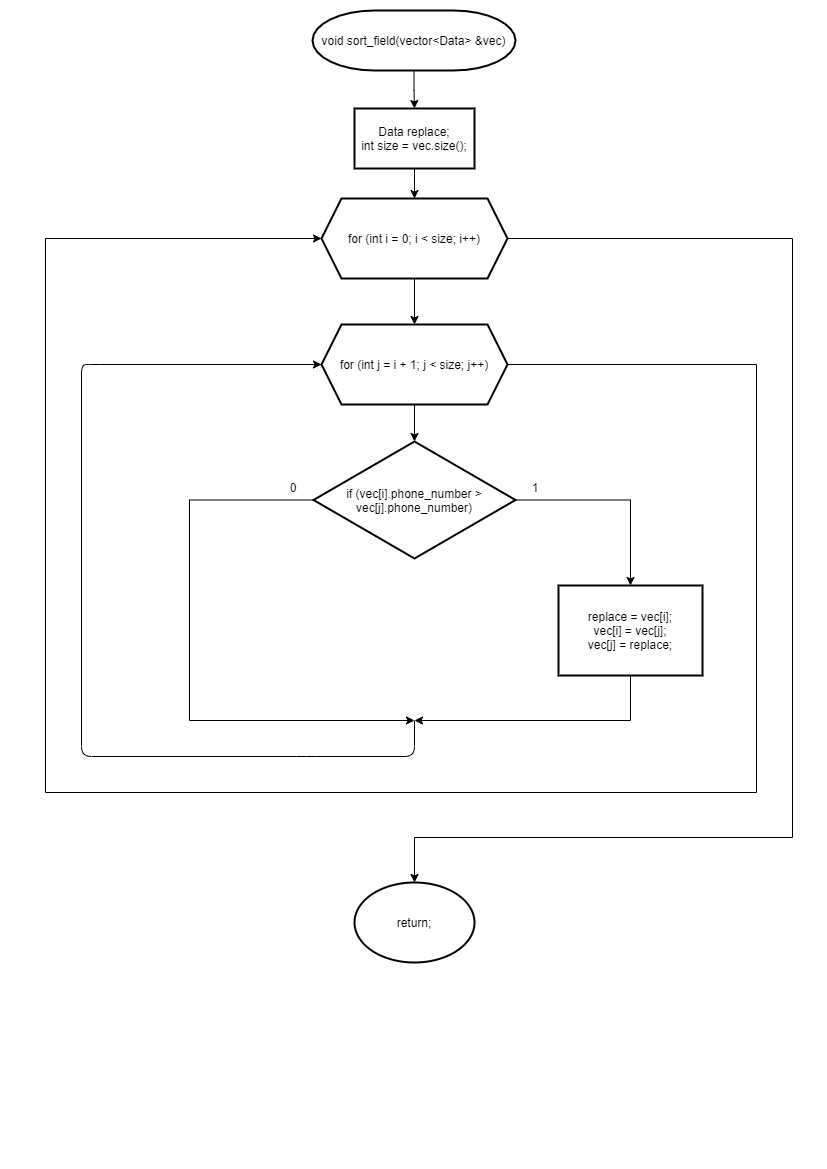
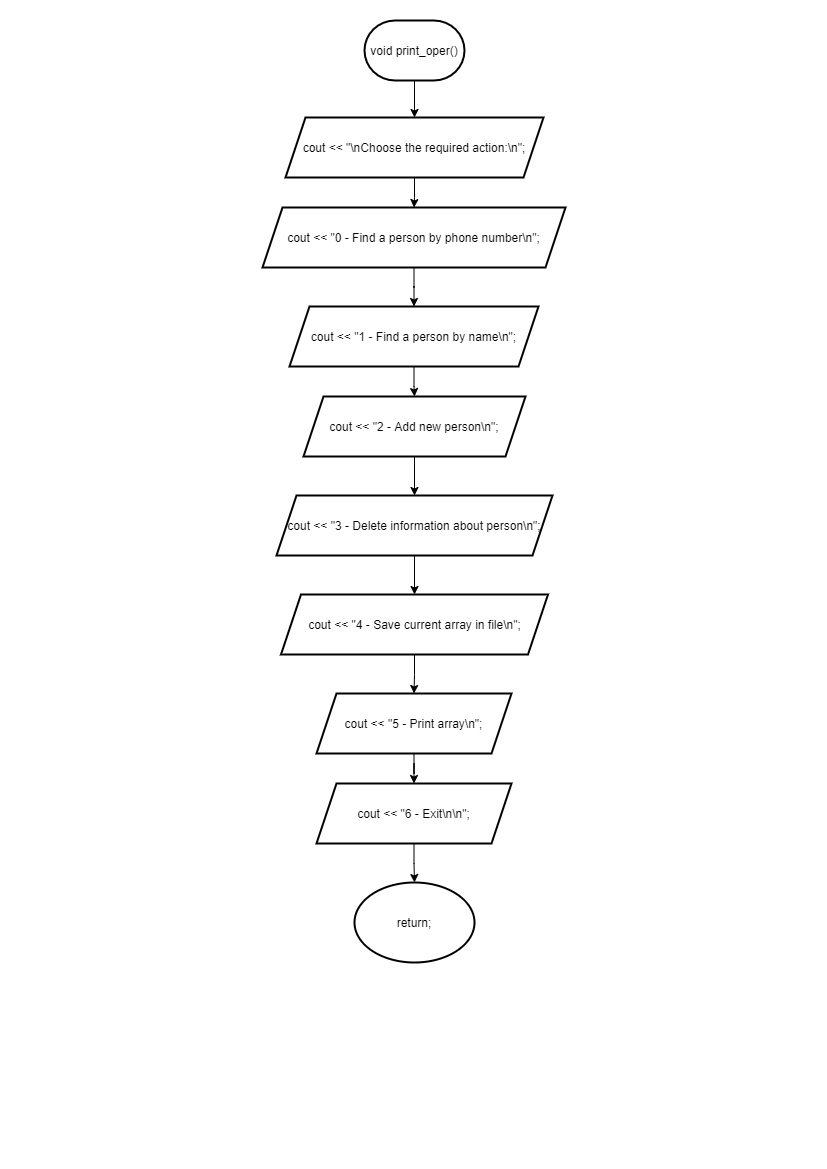
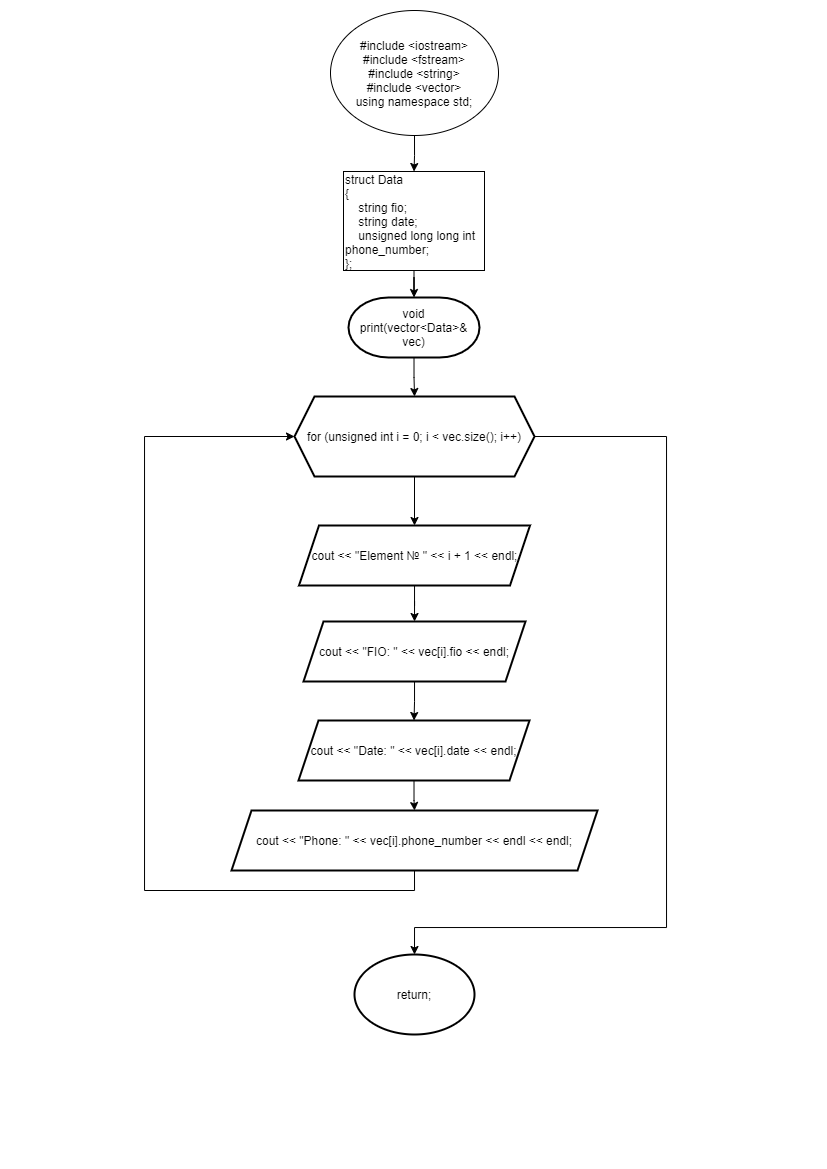
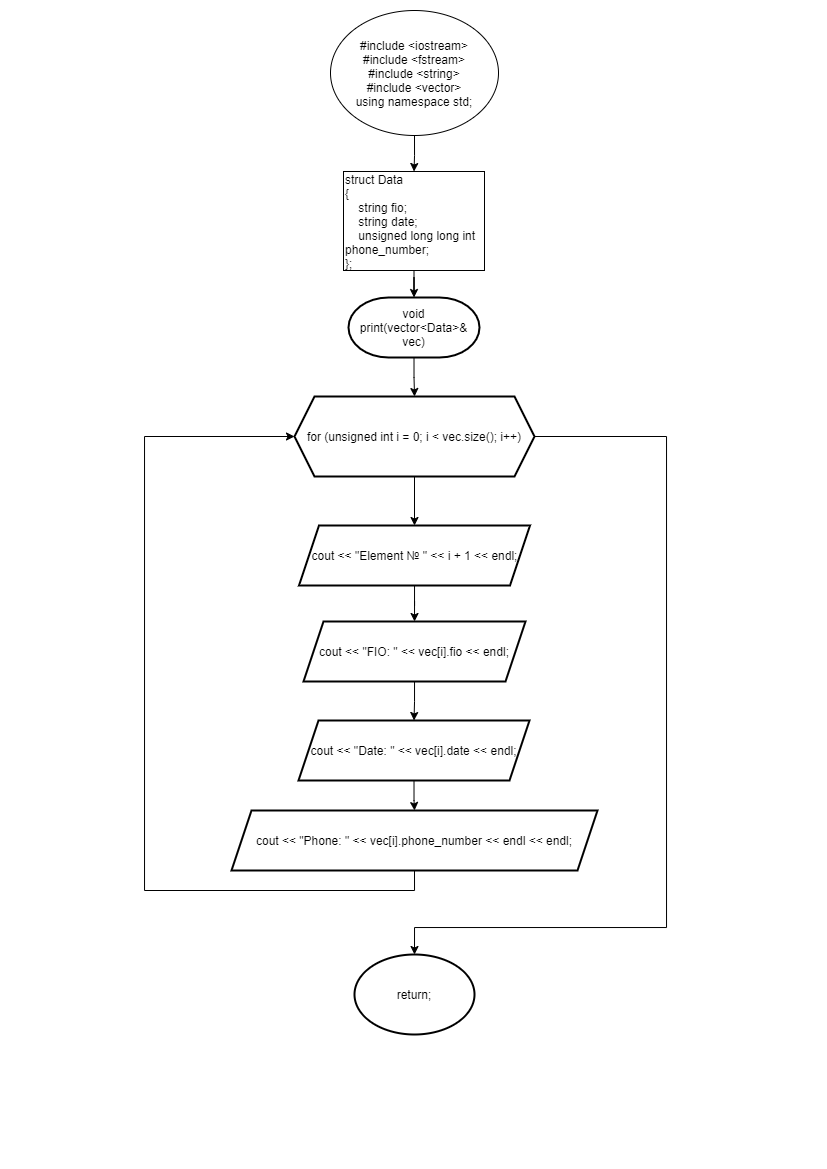
{

cout << "File not found\n";

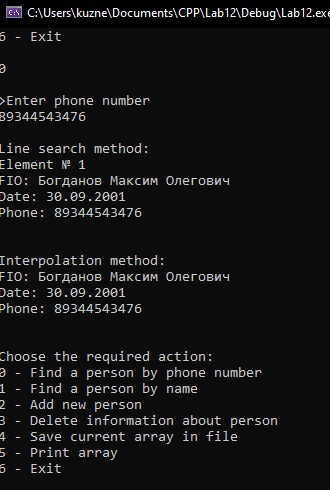
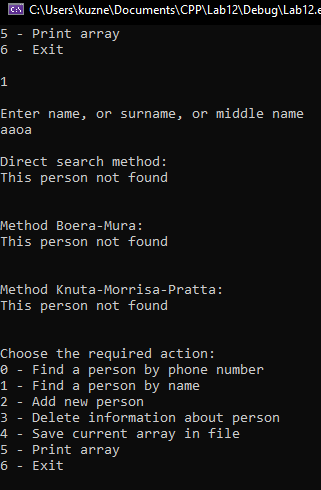
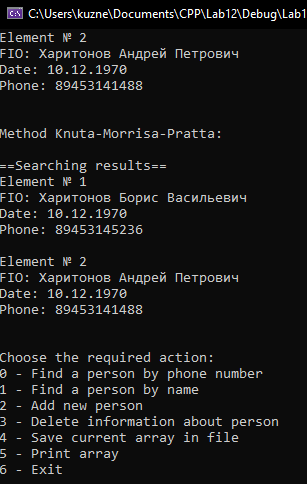
}

}

**Блок-схема**



**Скриншоты тестов**

****