Министерство образования и науки Российской Федерации

Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №12**

Дисциплина: «Основы теории алгоритмов и структуры данных»

Тема: Алгоритмы поиска. Работа со списками. Метод интерполяционного поиска

Вариант 18

Выполнил:

Студент группы РИС-20-1б

Тараканов Д. М.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

Пермь

2021 год

# **Цель работы**

Получить практические навыки работы с методами поисков.

# **Постановка задачи**

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов.
2. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать метод линейного поиска и интерполяционного поиска.

(18 вариант)

1. Данные: ФИО, дата\_рождения, №телефона.
2. Ключ: №телефона.

# **Анализ задачи**

1. Определить какие действия необходимо выполнить для решения задачи:

* Разработать структуру Directory которая хранит в себе данные (ФИО, дата\_рождения, №телефона).

struct Directory {

string sure\_name;

string name;

string middle\_name;

string birth\_data;

unsigned long long int phone\_number;

};

* Разработать функцию чтения данных из файла и занесения их в структуру.

int ReadFile(Directory book[], ifstream& file) {

string CurrentLine;

int index = 0;

while (!file.eof()) {

getline(file, CurrentLine);

book[index].sure\_name = CurrentLine;

getline(file, CurrentLine);

book[index].name = CurrentLine;

getline(file, CurrentLine);

book[index].middle\_name = CurrentLine;

getline(file, CurrentLine);

book[index].birth\_data = CurrentLine;

getline(file, CurrentLine);

book[index].phone\_number = stoll(CurrentLine);

getline(file, CurrentLine);

index++;

}

return index;

};

* Разработать меню переключения между методами поиска.

void SwitchSearchMetod(int quantity, Directory book[]) {

int MethodNumber = -1, index = 0;

unsigned long long int key;

while ((MethodNumber < 0) || (MethodNumber > 1)) {

cout << "\nВведите номер метода поиска\n\n0 - метод линейного поиска\n1 - метод интерполяционного поиска\n\nВаш выбор: ";

cin >> MethodNumber;

switch (MethodNumber) {

case (0):

cout << "\nВы выбрали метод линейного поиска\n";

cout << "\nВведите номер искомого телефона: ";

cin >> key;

LineSearch(book, quantity, key);

break;

case (1):

cout << "\nОтсортированый массив данных:\n";

SortDirectory(book, quantity);

cout << endl;

WriteDirectory(book, quantity);

cout << endl;

cout << "Вы выбрали метод интерполяционного поиска\n";

cout << "\nВведите номер искомого телефона: ";

cin >> key;

InterpolSearch(book, quantity, key);

break;

default: cout << "\nВведён не номер метода поиска!\n";

break;

}

}

};

* Разработать функции методов сортировок Линейного и Интерполяционного поиска. Для метода интерполяционного поиска реализовать алгоритм сортировки структуры данных.

void LineSearch(Directory book[], int N, unsigned long long int key) {

bool otvet = false;

for (int i = 0; i < N; i++) {

if (book[i].phone\_number == key) {

cout << "\nРезультат поиска:\n" << book[i].sure\_name << " " << book[i].name << " " << book[i].middle\_name << " " << book[i].birth\_data << " " << book[i].phone\_number << endl;

otvet = true;

}

}

if (otvet == false) {

cout << "\nПоиск не принес результатов!\nТакого номера нет в массиве!\n";

}

};

void InterpolSearch(Directory book[], int N, unsigned long long int key) {

int mid, left = 0, right = N - 1;

bool otvet = false;

while ((book[left].phone\_number <= key) && (book[right].phone\_number >= key) && (otvet == false)) {

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

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

left = mid + 1;

}

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

right = mid - 1;

}

else {

cout << "\nРезультат поиска:\n" << book[mid].sure\_name << " " << book[mid].name << " " << book[mid].middle\_name << " " << book[mid].birth\_data << " " << book[mid].phone\_number << endl;

otvet = true;

}

}

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

cout << "\nРезультат поиска:\n" << book[left].sure\_name << " " << book[left].name << " " << book[left].middle\_name << " " << book[left].birth\_data << " " << book[left].phone\_number << endl;

}

else if (otvet == false){

cout << "\nПоиск не принес результатов!\nТакого номера нет в массиве!\n";

}

};

void SortDirectory(Directory book[], int N) {

Directory replace;

for (int i = 0; i < N; i++) {

for (int j = i + 1; j < N; j++) {

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

replace = book[i];

book[i] = book[j];

book[j] = replace;

}

}

}

};

1. С какими типами данных действие надо сделать, в каком виде эти данные будут представлены:

* Для хранения ФИО в структуре реализованы переменные name, sure\_name, middle\_name типа string.

string sure\_name;

string name;

string middle\_name;

* Для хранения даты рождения в структуре реализована переменная birth\_data типа string.

string birth\_data;

* Для хранения № телефона в структуре реализованна переменная phone number типа unsigned long long int.

unsigned long long int phone\_number;

* Для хранения данных реализован массив book на 100 элементов типа Directory.

Directory book[100];

* Для хранения ключа поиска реализована переменная key типа unsigned long long int.

unsigned long long int key;

* Для чтения данных из файла реализована переменная inputfile типа ifstream

ifstream inputfile("F1.txt");

* Для подсчета кол-ва элементов в массиве реализована переменная quantity типа int.

int quantity = ReadFile(book, inputfile);

1. Реализация операций ввода и вывода данных:

* Ввод данных в структуру реализован через чтение строк из файла.

while (!file.eof()) {

getline(file, CurrentLine);

book[index].sure\_name = CurrentLine;

getline(file, CurrentLine);

book[index].name = CurrentLine;

getline(file, CurrentLine);

book[index].middle\_name = CurrentLine;

getline(file, CurrentLine);

book[index].birth\_data = CurrentLine;

getline(file, CurrentLine);

book[index].phone\_number = stoll(CurrentLine);

getline(file, CurrentLine);

index++;

}

* Вывод данных в консоль реализован при помощи оператора cout.

for (int i = 0; i < N; i++) {

cout << book[i].sure\_name << " " << book[i].name << " " << book[i].middle\_name << " " << book[i].birth\_data << " " << book[i].phone\_number << endl;

}

1. Поставленные задачи будут решены при помощи следующих действий:

* Программа считывает из файла данные и подсчитывает кол-во введённых элементов в массив book.

int quantity = ReadFile(book, inputfile);

int ReadFile(Directory book[], ifstream& file) {

string CurrentLine;

int index = 0;

while (!file.eof()) {

getline(file, CurrentLine);

book[index].sure\_name = CurrentLine;

getline(file, CurrentLine);

book[index].name = CurrentLine;

getline(file, CurrentLine);

book[index].middle\_name = CurrentLine;

getline(file, CurrentLine);

book[index].birth\_data = CurrentLine;

getline(file, CurrentLine);

book[index].phone\_number = stoll(CurrentLine);

getline(file, CurrentLine);

index++;

}

return index;

};

* Программа запрашивает у пользователя номер метода поиска, а затем запрашивает на ввод ключ поиска. При выборе интерполяционного поиска происходит сортировка массива данных.

while ((MethodNumber < 0) || (MethodNumber > 4)) {

cout << "\nВведите номер метода поиска\n\n0 - метод линейного поиска\n1 - метод интерполяционного поиска\n\nВаш выбор: ";

cin >> MethodNumber;

switch (MethodNumber) {

case (0):

cout << "\nВы выбрали метод линейного поиска\n";

cout << "\nВведите номер искомого телефона: ";

cin >> key;

LineSearch(book, quantity, key);

break;

case (1):

cout << "\nОтсортированый массив данных:\n";

SortDirectory(book, quantity);

cout << endl;

WriteDirectory(book, quantity);

cout << endl;

cout << "Вы выбрали метод интерполяционного поиска\n";

cout << "\nВведите номер искомого телефона: ";

cin >> key;

InterpolSearch(book, quantity, key);

break;

default: cout << "\nВведён не номер метода поиска!\n";

break;

}

}

* В случаи выбора линейного поиска происходит сравнение № телефона каждого элемента массива с ключом при помощи оператора for. При совпадении ключа с № телефона программа выводит в консоль элемент массива совпадающий с ключом.

bool otvet = false;

for (int i = 0; i < N; i++) {

if (book[i].phone\_number == key) {

cout << "\nРезультат поиска:\n" << book[i].sure\_name << " " << book[i].name << " " << book[i].middle\_name << " " << book[i].birth\_data << " " << book[i].phone\_number << endl;

otvet = true;

}

}

if (otvet == false) {

cout << "\nПоиск не принес результатов!\nТакого номера нет в массиве!\n";

}

* В случаи выбора интерполяционного поиска в массиве вычисляется примерная область нахождения совпадения № телефона с ключом. Если № телефона центра данной области совпадает с ключом, то программа выводит в консоль элемент массива совпадающий с ключом. Иначе происходит повторное вычисление области нахождения совпадения № телефона с ключом.

bool otvet = false;

while ((book[left].phone\_number <= key) && (book[right].phone\_number >= key) && (otvet == false)) {

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

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

left = mid + 1;

}

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

right = mid - 1;

}

else {

cout << "\nРезультат поиска:\n" << book[mid].sure\_name << " " << book[mid].name << " " << book[mid].middle\_name << " " << book[mid].birth\_data << " " << book[mid].phone\_number << endl;

otvet = true;

}

}

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

cout << "\nРезультат поиска:\n" << book[left].sure\_name << " " << book[left].name << " " << book[left].middle\_name << " " << book[left].birth\_data << " " << book[left].phone\_number << endl;

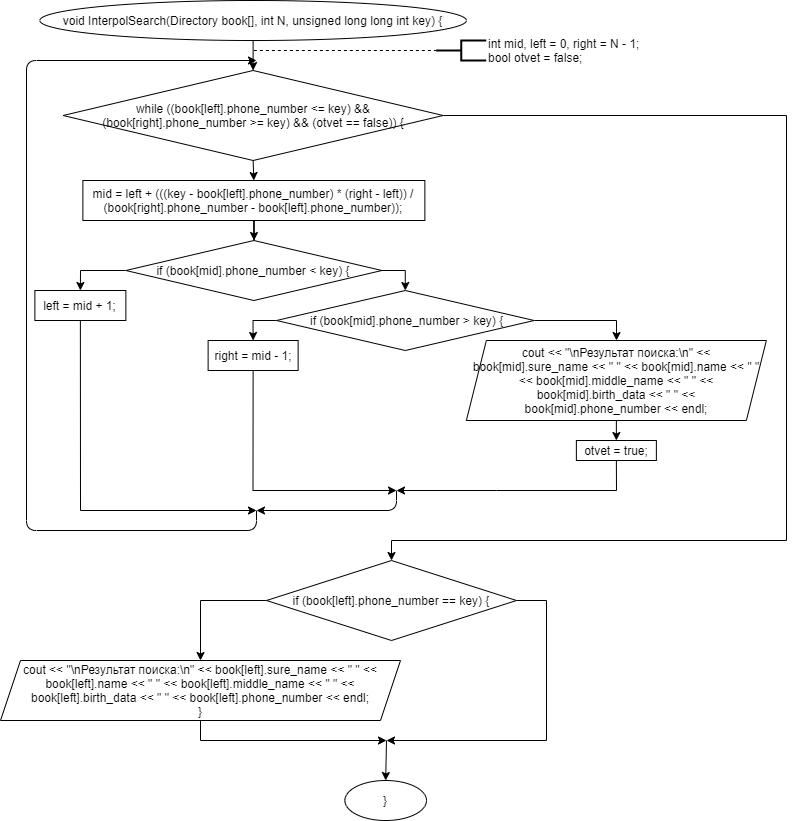
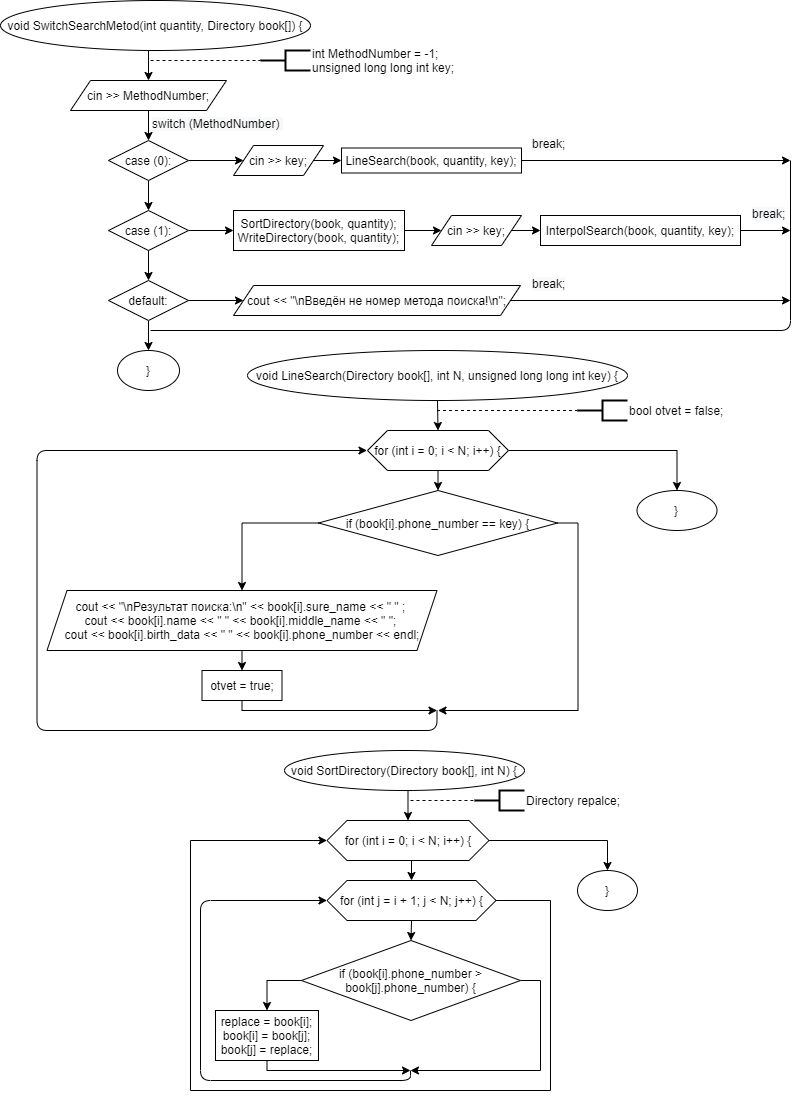
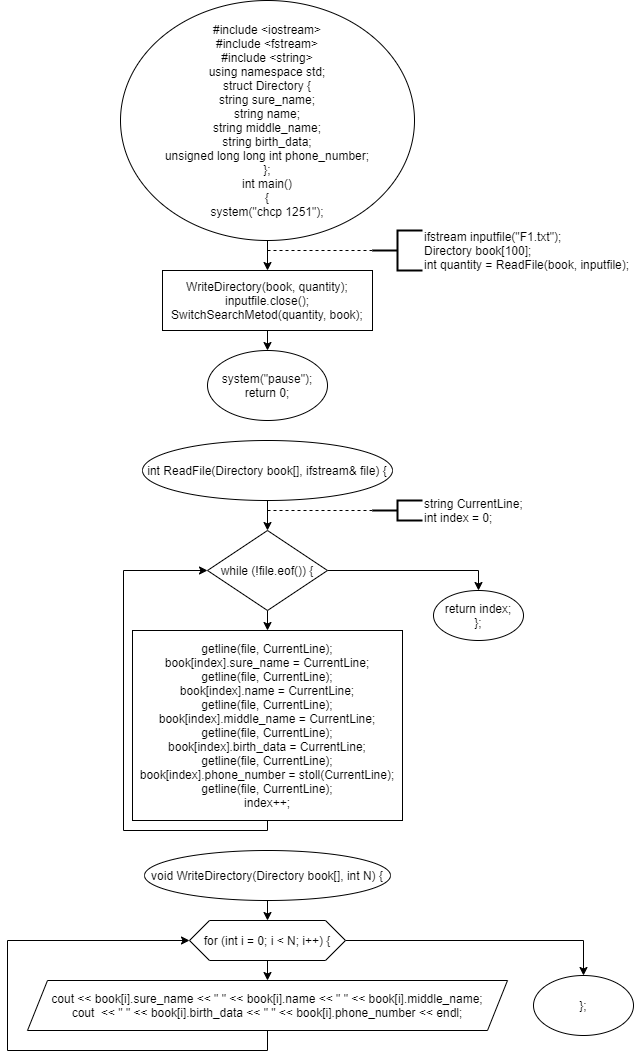
}

else if (otvet == false){

cout << "\nПоиск не принес результатов!\nТакого номера нет в массиве!\n";

}

**Блок схема**



# **Код программы на языке C++**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Directory {

string sure\_name;

string name;

string middle\_name;

string birth\_data;

unsigned long long int phone\_number;

};

int ReadFile(Directory book[], ifstream& file) {

string CurrentLine;

int index = 0;

while (!file.eof()) {

getline(file, CurrentLine);

book[index].sure\_name = CurrentLine;

getline(file, CurrentLine);

book[index].name = CurrentLine;

getline(file, CurrentLine);

book[index].middle\_name = CurrentLine;

getline(file, CurrentLine);

book[index].birth\_data = CurrentLine;

getline(file, CurrentLine);

book[index].phone\_number = stoll(CurrentLine);

getline(file, CurrentLine);

index++;

}

return index;

};

void WriteDirectory(Directory book[], int N) {

for (int i = 0; i < N; i++) {

cout << book[i].sure\_name << " " << book[i].name << " " << book[i].middle\_name << " " << book[i].birth\_data << " " << book[i].phone\_number << endl;

}

};

void LineSearch(Directory book[], int N, unsigned long long int key) {

bool otvet = false;

for (int i = 0; i < N; i++) {

if (book[i].phone\_number == key) {

cout << "\nРезультат поиска:\n" << book[i].sure\_name << " " << book[i].name << " " << book[i].middle\_name << " " << book[i].birth\_data << " " << book[i].phone\_number << endl;

otvet = true;

}

}

if (otvet == false) {

cout << "\nПоиск не принес результатов!\nТакого номера нет в массиве!\n";

}

};

void SortDirectory(Directory book[], int N) {

Directory replace;

for (int i = 0; i < N; i++) {

for (int j = i + 1; j < N; j++) {

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

replace = book[i];

book[i] = book[j];

book[j] = replace;

}

}

}

};

void InterpolSearch(Directory book[], int N, unsigned long long int key) {

int mid, left = 0, right = N - 1;

bool otvet = false;

while ((book[left].phone\_number <= key) && (book[right].phone\_number >= key) && (otvet == false)) {

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

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

left = mid + 1;

}

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

right = mid - 1;

}

else {

cout << "\nРезультат поиска:\n" << book[mid].sure\_name << " " << book[mid].name << " " << book[mid].middle\_name << " " << book[mid].birth\_data << " " << book[mid].phone\_number << endl;

otvet = true;

}

}

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

cout << "\nРезультат поиска:\n" << book[left].sure\_name << " " << book[left].name << " " << book[left].middle\_name << " " << book[left].birth\_data << " " << book[left].phone\_number << endl;

}

else if (otvet == false){

cout << "\nПоиск не принес результатов!\nТакого номера нет в массиве!\n";

}

};

void SwitchSearchMetod(int quantity, Directory book[]) {

int MethodNumber = -1, index = 0;

unsigned long long int key;

while ((MethodNumber < 0) || (MethodNumber > 1)) {

cout << "\nВведите номер метода поиска\n\n0 - метод линейного поиска\n1 - метод интерполяционного поиска\n\nВаш выбор: ";

cin >> MethodNumber;

switch (MethodNumber) {

case (0):

cout << "\nВы выбрали метод линейного поиска\n";

cout << "\nВведите номер искомого телефона: ";

cin >> key;

LineSearch(book, quantity, key);

break;

case (1):

cout << "\nОтсортированый массив данных:\n";

SortDirectory(book, quantity);

cout << endl;

WriteDirectory(book, quantity);

cout << endl;

cout << "Вы выбрали метод интерполяционного поиска\n";

cout << "\nВведите номер искомого телефона: ";

cin >> key;

InterpolSearch(book, quantity, key);

break;

default: cout << "\nВведён не номер метода поиска!\n";

break;

}

}

};

int main()

{

system("chcp 1251");

ifstream inputfile("F1.txt");

if (inputfile.is\_open()) {

Directory book[100];

int quantity = ReadFile(book, inputfile);

cout << "\nМассив данных:\n\n";

WriteDirectory(book, quantity);

inputfile.close();

SwitchSearchMetod(quantity, book);

}

else {

cout << "Ошибка! Не удалось открыть файл F1.txt.\n";

}

system("pause");

return 0;

}

# **Скриншоты тестов**

