

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГСН | | 09.00.00 | Информатика и вычислительная техника | | |
| Уровень образования | |  | Высшее образование – бакалавриат | | |
| Форма обучения | |  | Очная | | |
| Факультет | |  | Информационных технологий  и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Информационные технологии  и программирование | | |
| Курс | I | | | Группа | 4303 |

Отчёт по контрольной работе № 1

Вариант № 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 4303 |  |  |  | Иванов Глеб Игоревич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверил: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |
|  |  |  |  |  |

# СОДЕРЖАНИЕ

[1 Задание №1 3](#_Toc161745186)

[1.1 Постановка задачи 3](#_Toc161745187)

[1.2 Исходные данные 3](#_Toc161745188)

[1.3 Особых ситуаций 3](#_Toc161745189)

[1.4 Математические методы и алгоритмы решения задачи 4](#_Toc161745190)

[1.5 Блок-схемы алгоритмов программы 5](#_Toc161745191)

[1.6 Форматы представления данных 7](#_Toc161745192)

[1.7 Структуры (модулей, функций) программы 9](#_Toc161745193)

[1.8 Ход выполнения работы 10](#_Toc161745194)

[1.9 Полученные результаты 10](#_Toc161745195)

[1.10 Исходный код программы 15](#_Toc161745196)

# 1 Задание №1

## Постановка задачи

Первая контрольная работа предназначена для приобретения практического опыта в создании простейших классов с использованием языка программирования С++. Необходимо разработать класс для указанной предметной области. Доступ к данным реализовать с помощью методов *Set*, *Get*. Предусмотреть необходимые проверки исходных данных.

Абонент: фамилия, имя, отчество, адрес, номер телефона, время внутригородских разговоров, время междугородних разговоров. Создать массив объектов. Реализовать возможность получения:

– списка абонентов, время внутригородских разговоров которых превышает заданное,

– списка абонентов, воспользовавшихся междугородней связью,

– списка абонентов, выведенных в алфавитном порядке.

## Исходные данные

Назначение данных:

Программа получает на вход фамилия, имя, адрес, номер телефона, время внутригородских разговоров, время междугородних разговоров.

Представление в программе:

фамилия - *second\_name\_,*

имя - *name\_,*

адрес - *address\_,*

телефон - *phone\_number\_,*

внутригородских разговоров - *time\_inner\_city\_calls\_,*

время междугородних разговоров - *time\_long\_distance\_calls\_.*

## 1.3 Особые ситуации

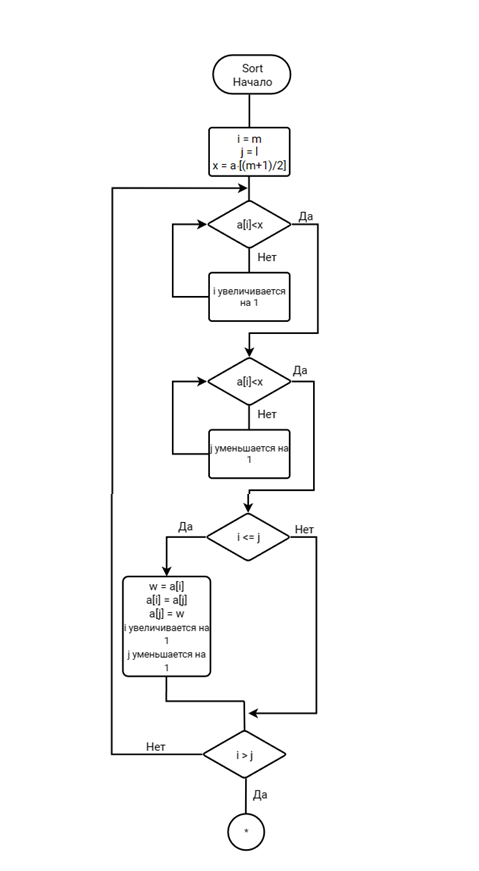
При выполнении задания были встречены ситуации, в которых программа может вести себя некорректно и непредсказуемо. Для решения этой проблемы было принято решение включить в исходный код следующие функции: *Num\_input()* – функция, которая проверяет введённое число на корректность.

Также были обработаны ошибки с некорректно введённым путём к файлу. Программа выдаёт ошибку “*Failed to open the file*” и просит ввести новый путь до тех пор, пока не получит существующий файл.

## Математические методы и алгоритмы решения задачи

В программе используется алгоритм сортировки в алфавитном порядке, который осуществляется с помощью функции *AlphabetOrderSort*. Для этого вызывается стандартная функция сортировки *std::sort*, где объекты *PhoneUser* сравниваются по их фамилиям и именам с помощью пользовательской функции *ValueCmp*. После сортировки данные выводятся или сохраняются в соответствии с заданным вариантом. Сама же функция sort для сортировки использует алгоритм, описанный на рисунке 1. Для описания алгоритма sort было принято решение ввести новые переменные: *m* – первый индекс массива; *l* – конечный индекс массива; *i, j* – переменные для прохода по индексам массива; *x* – переменная для хранения элемента, относительно которого мы будем сортировать; *w* – временная переменна для смены *i*-го элемента с *j*-тым;

## Блок-схемы алгоритмов программы



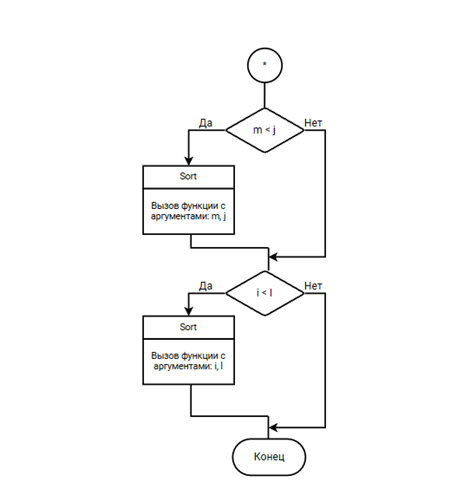


Рисунок 1 – Блок-схема алгоритма Sort

## 1.6 Форматы представления данных

Таблица 1 – Переменные класса PhoneUser

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| second\_name\_ | string | Фамилия клиента |
| name\_ | string | Имя клиента |
| address\_ | string | Адрес клиента |
| phone\_number\_ | int | Номер телефона клиента |
| time\_inner\_city\_calls\_ | int | Время внутригородских разговоров |
| time\_long\_distance\_calls\_ | int | Время междугородних разговоров. |

Таблица 2 – Основные переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| book | PhoneUser\* | Контейнер для сохранения вводимых клиентов |
| command | int | Переменная для выбора изначальной команды |
| save\_book | std::vector<PhoneUser> | Вектор для хранения пользователей для сохранения |
| name, second\_name, address, phone, inner\_time, long\_distance\_time | string, int | Переменные для временного хранения данных пользователя |
| size\_of\_book | int | Переменная для хранения размера массива пользователей |
| option\_of\_filling | int | Переменная для хранения выбора метода заполнения данных пользователей |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| option\_of\_function | int | Переменная для хранения выбора опции функции |
| option\_of\_save | int | Переменная для хранения выбора опции сохранения результатов |
| filename | std::string | Название файла, с которым ведется работа |
| file | std::ifstream/std::ofstream | Поток для чтения/записи данных в файл |
| end\_of\_file | char | Переменная для хранения символа конца файла (EOF) |
| is\_empty | bool | Переменная для проверки, является ли файл пустым |
| result\_book1, result\_book2, result\_book3, result\_book4, result\_book5 | std::vector<PhoneUser> | Векторы, содержащие результаты тестирования |
| test\_book | PhoneUser\* | Указатель на массив пользователей, используемый для тестирования |
| limit | const int | Переменная, определяющая ограничение для теста 3 |

## 1.7 Структуры (модулей, функций) программы

Таблица 3 – Функции, составляющие программу

|  |  |
| --- | --- |
| Имя | Описание |
| Num\_input | Функция для ввода беззнакового целого числа с защитой от некорректного ввода |
| FileIsEmpty | Функция для проверки, пуст ли файл |
| OpenFile | Функция для открытия файла для чтения или записи |
| FileInput | Функция для записи данных пользователей в файл |
| FileFilling | Функция для чтения данных пользователей из файла |
| Test1 | Функция для тестирования первого теста |
| Test2 | Функция для тестирования второго теста |
| Test3 | Функция для тестирования третьего теста |
| Test4 | Функция для тестирования четвертого теста |
| Test5 | Функция для тестирования пятого теста |
| Test | Функция для запуска серии тестов |
| HandFilling | Функция для ручного заполнения данных пользователей |
| Filling | Функция для выбора метода заполнения данных пользователей |
| LongDistanceCallsUser | Функция для обработки данных о пользователях, совершавших долгие звонки |
| InnerCityCallsUserLimit | Функция для обработки данных о пользователях с ограничением для внутригородних звонков |
| ValueCmp | Функция сравнения двух пользователей по фамилии и имени |
| AlphabetOrderSort | Функция для сортировки пользователей по алфавиту |
| AlphabetOrder | Функция для сортировки пользователей по алфавиту с возможностью сохранения результата |
| Function | Основная функция, выполняющая основной сценарий программы |

## 1.8 Ход выполнения работы

Во время выполнения программы в Visual Studio 2022 были освоены базовые навыки работы, такие как: настройка, запуск проекта, отладка процесса выполнения с помощью точек останова и просмотра значений переменных. Также были обнаружены проблемы в нехватке знаний необходимых инструментов. Были решены проблемы с созданием класса и добавлением новых клиентов вектор путем чтения многих форумов. Проблемы ввода и вывода данных при помощи файлов также были решены благодаря многочисленным форумам в интернете. При помощи YouTube были решены проблемы сортировки и вывода данных.

## 1.9 Полученные результаты

На рисунках 2 – 4 показана работа функций, обрабатывающих ошибки, на рисунках 5 – 7 показана работа функций сортировок и тестов.

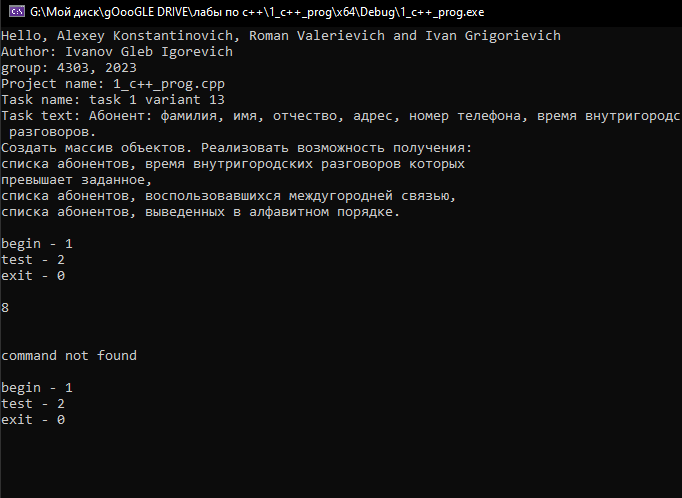


Рисунок 2 – Экранная копия результата работы разработанной программы задания №1

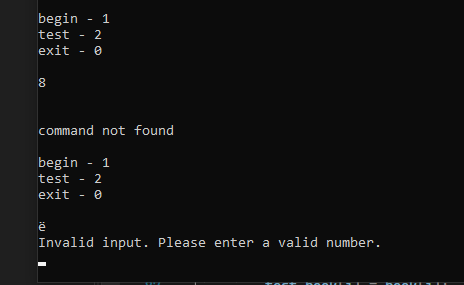


Рисунок 3 – Экранная копия результата работы разработанной программы задания №1

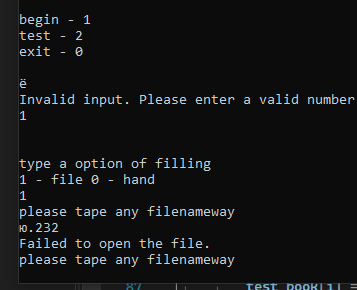


Рисунок 4 – Экранная копия результата работы разработанной программы задания №1

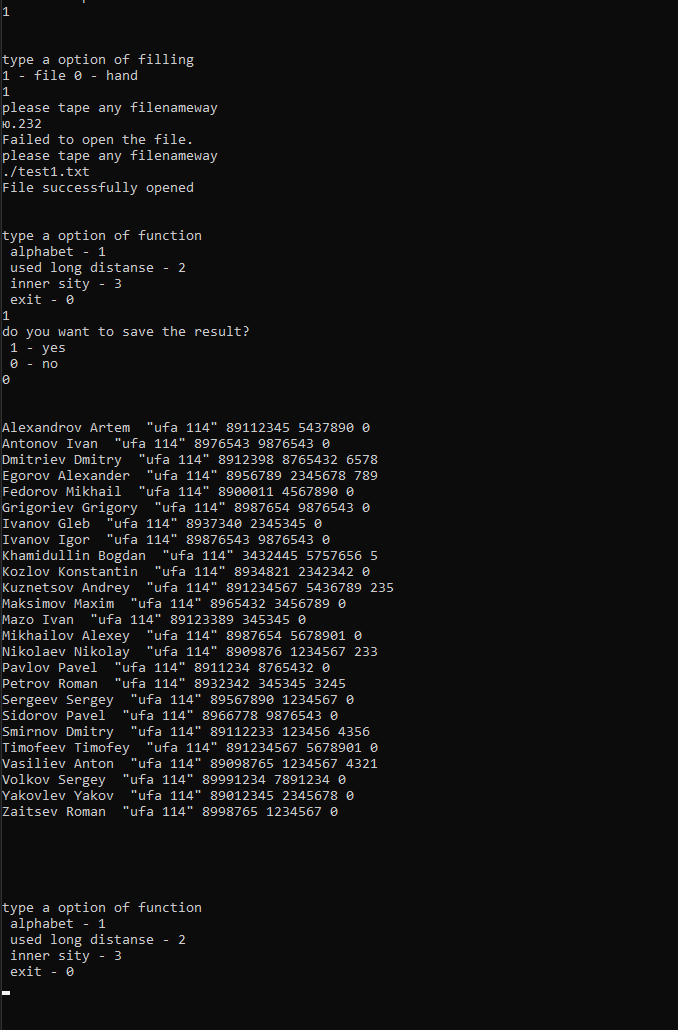


Рисунок 5 – Экранная копия результата работы разработанной программы задания №1

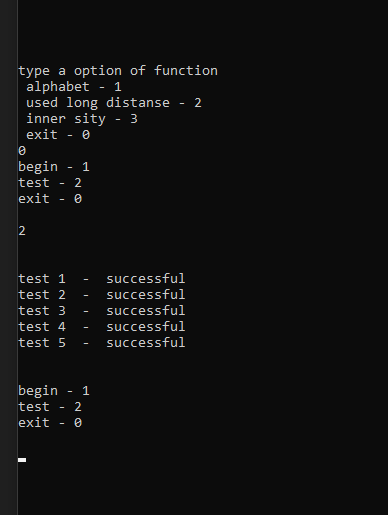


Рисунок 6 – Экранная копия результата работы разработанной программы задания №1

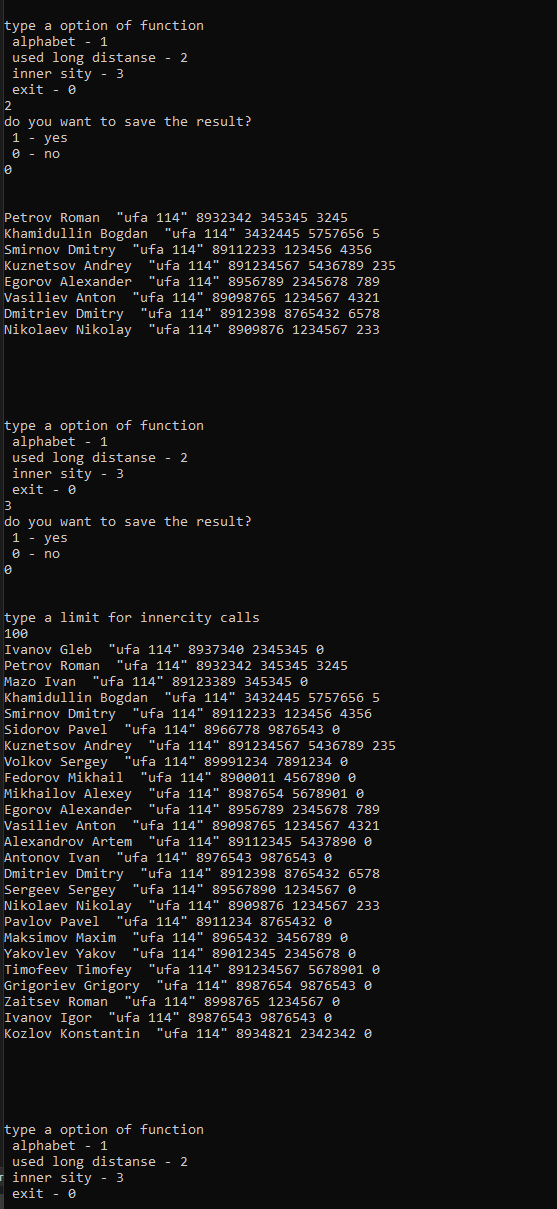


Рисунок 7 – Экранная копия результата работы разработанной программы задания №1

## 1.10 Исходный код программы

#include <iostream>

#include <algorithm>

#include <fstream>

#include <vector>

#include <string>

using std::string;

enum Menu { hand\_filling , file\_filling };

class PhoneUser {

public:

string GetName() {

return name\_;

}

string GetSecondName() {

return second\_name\_;

}

string GetAddress() {

return address\_;

}

int GetPhoneNumber() {

return phone\_number\_;

}

int GetTimeInnerCityCalls() {

return time\_inner\_city\_calls\_;

}

int GetTimeLongDistanceCalls() {

return time\_long\_distance\_calls\_;

}

void SetName(string name) {

name\_ = name;

}

void SetSecondName(string second\_name) {

second\_name\_ = second\_name;

}

void SetAddress() {

while (true)

{

std::string tmp;

std::cin >> tmp;

address\_ = address\_ + " " + tmp;

if (tmp.back() == '\"') break;

}

}

void SetAddress(std::ifstream& file) {

while (true)

{

std::string tmp;

file >> tmp;

address\_ = address\_ + " " + tmp;

if (tmp.back() == '\"') break;

}

}

void SetPhoneNumber(int phone\_number){

phone\_number\_ = phone\_number;

}

void SetTimeInnerCityCalls(int time\_inner\_city\_calls){

time\_inner\_city\_calls\_ = time\_inner\_city\_calls;

}

void SetTimeLongDistanceCalls(int time\_long\_distance\_calls){

time\_long\_distance\_calls\_ = time\_long\_distance\_calls;

}

bool operator==(const PhoneUser& other) const {

return (name\_ == other.name\_ &&

second\_name\_ == other.second\_name\_ &&

address\_ == other.address\_ &&

phone\_number\_ == other.phone\_number\_ &&

time\_inner\_city\_calls\_ == other.time\_inner\_city\_calls\_ &&

time\_long\_distance\_calls\_ == other.time\_long\_distance\_calls\_);

}

private:

string name\_;

string second\_name\_;

string address\_;

int phone\_number\_= 0;

int time\_inner\_city\_calls\_= 0;//внутригород

int time\_long\_distance\_calls\_= 0;//межгород

};

unsigned long long Num\_input() {

unsigned long long input = 0;

bool isError = false;

do {

if (isError) {

std::cout << "Invalid input. Please enter a valid number." << std::endl;

}

if (!(std::cin >> input)) {

isError = true;

std::cin.clear(); // Очищаем флаг ошибки ввода

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Очищаем буфер ввода до символа новой строки

}

else {

isError = false;

}

} while (isError);

return input;

}

bool FileIsEmpty(std::string filename) {

std::ifstream file;

file.open(filename);

char end\_of\_file;

if (file.is\_open()) {

file >> end\_of\_file;

if (end\_of\_file == EOF) return 1;

else return 0;

}

}

bool OpenFile(std::string filename, std::ifstream& file) { // чтение

// Попытаться открыть файл

file.open(filename);

if (file.is\_open()) {

std::cout << "File successfully opened" << "\n";

return 1;

}

else {

std::cout << "Failed to open the file." << "\n";

return 0;

}

}

bool OpenFile(std::string filename, std::ofstream& file) { //запись

bool is\_empty = FileIsEmpty(filename);

// Попытаться открыть файл

file.open(filename, std::ios::app);// для дозаписи

if (file.is\_open() && is\_empty) {

std::cout << "File successfully opened" << "\n";

return 1;

}

if (file.is\_open()&&is\_empty ==0) {

std::cout << "File is not empty, failed to open the file. " << "\n";

return 0;

}

else {

std::cout << "Failed to open the file." << "\n";

return 0;

}

}

void FileInput(std::vector<PhoneUser> \*save\_book) {

std::ofstream file;

while (true) {

std::cout << "please tape any filenameway" << "\n";

std::string filename;

std::cin >> filename;

//открытие файла:

if (OpenFile(filename, file)) {

break;

}

}

// Запись данных в файл

file << save\_book->size()<<"\n";

for ( PhoneUser& user : \*save\_book) {

file << user.GetSecondName() << " " << user.GetName() << " "<< user.GetAddress() << " " << user.GetPhoneNumber() << " " << user.GetTimeInnerCityCalls()

<< " " << user.GetTimeLongDistanceCalls() << "\n";

}

std::cout << "Data successfully written to the file." << std::endl;

// Закрытие файла

file.close();

}

void FileFilling(PhoneUser\*& book, int& size\_of\_book) {

std::ifstream file;

while (true) {

std::cout << "please tape any filenameway" << "\n";

std::string filename;

std::cin >> filename;

//открытие файла:

if (OpenFile(filename, file)) {

break;

}

}

file >> size\_of\_book;

book = new PhoneUser[size\_of\_book];

// Чтение данных из файла

for (int i = 0; i < size\_of\_book; i++) { // заполнение

string name;

string second\_name;

int phone = 0;

int inner\_time = 0;

int long\_distance\_time = 0;

file >> second\_name;

file >> name;

book[i].SetAddress(file);

file.ignore(); // clear input buffer

file >> phone;

file >> inner\_time;

file >> long\_distance\_time;

book[i].SetSecondName(second\_name);

book[i].SetName(name);

book[i].SetPhoneNumber(phone);

book[i].SetTimeInnerCityCalls(inner\_time);

book[i].SetTimeLongDistanceCalls(long\_distance\_time);

}

// Закрытие файла

file.close();

}

void FileFilling(std::vector<PhoneUser>& book,std::string filename) { // для тестов

std::ifstream file;

file.open(filename);

int size\_of\_book = 0;

file >> size\_of\_book;

book.resize(size\_of\_book);

// Чтение данных из файла

for (int i = 0; i < size\_of\_book; i++) { // заполнение

string name;

string second\_name;

int phone = 0;

int inner\_time = 0;

int long\_distance\_time = 0;

file >> second\_name;

file >> name;

book[i].SetAddress(file);

file.ignore(); // clear input buffer

file >> phone;

file >> inner\_time;

file >> long\_distance\_time;

book[i].SetSecondName(second\_name);

book[i].SetName(name);

book[i].SetPhoneNumber(phone);

book[i].SetTimeInnerCityCalls(inner\_time);

book[i].SetTimeLongDistanceCalls(long\_distance\_time);

}

// Закрытие файла

file.close();

}

void Test1(std::vector<PhoneUser>\* save\_book) {

std::vector <PhoneUser> result\_book1;

std::string filename = "./result1.txt";

FileFilling(result\_book1, filename);

bool checker = (\*save\_book == result\_book1);

if (checker) std::cout << "test 1 - successful \n";

else std::cout << "test 1 - faild \n";

}

void Test2(std::vector<PhoneUser>\* save\_book) {

std::vector <PhoneUser> result\_book2;

std::string filename = "./result2.txt";

FileFilling(result\_book2, filename);

bool checker = (\*save\_book == result\_book2);

if (checker) std::cout << "test 2 - successful \n";

else std::cout << "test 2 - faild \n";

}

void Test3(std::vector<PhoneUser>\* save\_book) {

std::vector <PhoneUser> result\_book3;

std::string filename = "./result3.txt";

FileFilling(result\_book3, filename);

bool checker = (\*save\_book == result\_book3);

if (checker) std::cout << "test 3 - successful \n";

else std::cout << "test 3 - faild \n";

}

void Test4(std::vector<PhoneUser>\* save\_book) {

std::vector <PhoneUser> result\_book4;

std::string filename = "./result4.txt";

FileFilling(result\_book4, filename);

bool checker = (\*save\_book == result\_book4);

if (checker) std::cout << "test 4 - successful \n";

else std::cout << "test 4 - faild \n";

}

void Test5(std::vector<PhoneUser>\* save\_book) {

std::vector <PhoneUser> result\_book5;

std::string filename = "./result5.txt";

FileFilling(result\_book5, filename);

bool checker = (\*save\_book == result\_book5);

if (checker) std::cout << "test 5 - successful \n";

else std::cout << "test 5 - faild \n";

}

void LongDistanceCallsUser(PhoneUser\* book, int size\_of\_book, int option\_of\_save) { // вторая функция

std::vector<PhoneUser> save\_book;

for (int i = 0; i < size\_of\_book; i++) { //вывод массива

if (book[i].GetTimeLongDistanceCalls()) {

if (option\_of\_save == 0 || option\_of\_save == 1) {

std::cout << book[i].GetSecondName() << " " << book[i].GetName() << " " << book[i].GetAddress()

<< " " << book[i].GetPhoneNumber() << " " << book[i].GetTimeInnerCityCalls()

<< " " << book[i].GetTimeLongDistanceCalls() << "\n";

}

if (option\_of\_save) {

save\_book.push\_back(book[i]);

}

}

}

if (option\_of\_save == 1) {

FileInput(&save\_book);

}

else if (option\_of\_save == 3) {

Test2(&save\_book);

}

else if (option\_of\_save == 5) {

Test4(&save\_book);

}

}

void InnerСityСallsUserLimit(PhoneUser\* book, int size\_of\_book, int limit, int option\_of\_save) {

std::vector<PhoneUser> save\_book;

for (int i = 0; i < size\_of\_book; i++) { //вывод массива

if (book[i].GetTimeInnerCityCalls() > limit) {

if (option\_of\_save == 0 || option\_of\_save == 1) {

std::cout << book[i].GetSecondName() << " " << book[i].GetName() << " " << book[i].GetAddress()

<< " " << book[i].GetPhoneNumber() << " " << book[i].GetTimeInnerCityCalls()

<< " " << book[i].GetTimeLongDistanceCalls() << "\n";

}

if (option\_of\_save) {

save\_book.push\_back(book[i]);

}

}

}

if (option\_of\_save == 1) {

FileInput(&save\_book);

}

else if (option\_of\_save == 4) {

Test3(&save\_book);

}

else if (option\_of\_save == 6) {

Test5(&save\_book);

}

}

bool ValueCmp(PhoneUser& a, PhoneUser& b)

{

if (a.GetSecondName() == b.GetSecondName()) {

return a.GetName() < b.GetName();

}

return a.GetSecondName() < b.GetSecondName();

}

void AlphabetOrderSort(PhoneUser\* book, int size\_of\_book) {

std::sort(book, book + size\_of\_book, ValueCmp);

}

void AlphabetOrder(PhoneUser\* book, int size\_of\_book, int option\_of\_save) {

AlphabetOrderSort(book, size\_of\_book);

std::vector<PhoneUser> save\_book;

for (int i = 0; i < size\_of\_book; i++) { //вывод массива

if (option\_of\_save == 0 || option\_of\_save == 1) {

std::cout << book[i].GetSecondName() << " " << book[i].GetName() << " " << book[i].GetAddress()

<< " " << book[i].GetPhoneNumber() << " " << book[i].GetTimeInnerCityCalls()

<< " " << book[i].GetTimeLongDistanceCalls() << "\n";

}

if (option\_of\_save) {

save\_book.push\_back(book[i]);

//std::reverse(save\_book.begin(), save\_book.end());

}

}

if (option\_of\_save==1) {

FileInput(&save\_book);

}

else if (option\_of\_save == 2) {

Test1(&save\_book);

}

else if (option\_of\_save == 5) {

LongDistanceCallsUser(book, size\_of\_book, 5);

}

else if (option\_of\_save == 6) {

int limit = 100;

InnerСityСallsUserLimit(book, size\_of\_book, limit, 6);

}

}

//

void HandFilling(PhoneUser\*& book, int& size\_of\_book) {

std::cout << "example: name secondName \"address\" phone innerTime longDistansetime" << "\n";

for (int i = 0; i < size\_of\_book; i++) { // заполнение

string name;

string second\_name;

int phone = 0;

int inner\_time = 0;

int long\_distance\_time = 0;

std::cin >> second\_name;

std::cin >> name;

book[i].SetAddress();

std::cin.ignore(); // clear input buffer

phone = static\_cast<int>(Num\_input());

inner\_time = static\_cast<int>(Num\_input());

long\_distance\_time = static\_cast<int>(Num\_input());

book[i].SetSecondName(second\_name);

book[i].SetName(name);

book[i].SetPhoneNumber(phone);

book[i].SetTimeInnerCityCalls(inner\_time);

book[i].SetTimeLongDistanceCalls(long\_distance\_time);

}

std::cout << "\n" << "\n" << "\n";

}

void Filling(bool option, PhoneUser\*& book, int& size\_of\_book) {

if (option == file\_filling) {

FileFilling(book, size\_of\_book);

}

else {

HandFilling(book, size\_of\_book);

}

}

//

void Function() {

int size\_of\_book = 0;

PhoneUser\* book = nullptr;

std::cout << "type a option of filling " << "\n"

"1 - file 0 - hand" << "\n";// запрос заполнения

int option\_of\_filling = 0;

while (true) {

option\_of\_filling = static\_cast<int>(Num\_input());

if (option\_of\_filling == 1 || option\_of\_filling == 0) {

break;

}

std::cout << "Invalid input. Please enter a valid number." << "\n";

}

if (option\_of\_filling == 0) {

std::cout << "type a size of book of users " << "\n";

size\_of\_book = static\_cast<int>(Num\_input());

book = new PhoneUser[size\_of\_book];

}

Filling(option\_of\_filling, book, size\_of\_book); //вызов функции заполнения

bool exit = 0;

while (!exit) {

// спрашивать что за функция

// потом откуда брать

// потом сохранить ли результаты

int option\_of\_function = 0;

std::cout << "\n\n";

std::cout << "type a option of function\n alphabet - 1\n used long distanse - 2\n inner sity - 3\n exit - 0" << "\n";// запрос функции

while (true) { //проверка на корректность выбранной функции

option\_of\_function = static\_cast<int>(Num\_input());

if (option\_of\_function == 1) {//aлфафит сделать энамом

break;

}

else if (option\_of\_function == 2) {// long distanse users сделать энамом

break;

}

else if (option\_of\_function == 3) {// inner city sf сделать энамом

break;

}

else if (option\_of\_function == 0) {// выход

exit = 1;

break;

}

else {

std::cout << "Invalid input. Please enter a valid number." << "\n";

}

}

if (exit == 1) {

break;

}

std::cout << "do you want to save the result?\n 1 - yes\n 0 - no" << "\n";// запрос сохраниненя данных

int option\_of\_save = 0;

while (true) {

option\_of\_save = static\_cast<int>(Num\_input());

if (option\_of\_save == 1 || option\_of\_save == 0) {

break;

}

std::cout << "Invalid input. Please enter a valid number." << "\n";

}

std::cout << "\n\n";

switch (option\_of\_function) { // выбор функции

case 1:

AlphabetOrder(book, size\_of\_book, option\_of\_save);

break;

case 2:

LongDistanceCallsUser(book, size\_of\_book, option\_of\_save);

break;

case 3:

std::cout << "type a limit for innercity calls" << "\n";

int limit = static\_cast<int>(Num\_input());

InnerСityСallsUserLimit(book, size\_of\_book, limit, option\_of\_save);

break;

}

std::cout << "\n" << "\n" << "\n";

}

delete[] book;

}

//testing

//./tmp

void Test() {

std::vector <PhoneUser> book;

std::string filename = "./test1.txt";

FileFilling(book, filename);

int size\_of\_book = static\_cast<int>(book.size()) ;

PhoneUser\* test\_book = new PhoneUser[size\_of\_book];

for (size\_t i = 0; i < size\_of\_book; i++)

{

test\_book[i] = book[i];

}

//test1 просто алфавит

AlphabetOrder(test\_book, size\_of\_book, 2); //сделать энамом

delete[] test\_book;

test\_book = new PhoneUser[size\_of\_book];

for (size\_t i = 0; i < size\_of\_book; i++)

{

test\_book[i] = book[i];

}

//test2 межгород

LongDistanceCallsUser(test\_book, size\_of\_book, 3);

delete[] test\_book;

test\_book = new PhoneUser[size\_of\_book];

for (size\_t i = 0; i < size\_of\_book; i++)

{

test\_book[i] = book[i];

}

//test 3 внутренние с лимитом

const int limit = 100;

InnerСityСallsUserLimit(test\_book, size\_of\_book, limit, 4);

delete[] test\_book;

test\_book = new PhoneUser[size\_of\_book];

for (size\_t i = 0; i < size\_of\_book; i++)

{

test\_book[i] = book[i];

}

//test 4 межгород с алфавитом

delete[] test\_book;

test\_book = new PhoneUser[size\_of\_book];

for (size\_t i = 0; i < size\_of\_book; i++)

{

test\_book[i] = book[i];

}

AlphabetOrder(test\_book, size\_of\_book, 5);

//test 5 внутренние с алфавитом

delete[] test\_book;

test\_book = new PhoneUser[size\_of\_book];

for (size\_t i = 0; i < size\_of\_book; i++)

{

test\_book[i] = book[i];

}

AlphabetOrder(test\_book, size\_of\_book, 6);

}

//

int main()

{

setlocale(LC\_CTYPE, "Russian");

int command = 0;

// grets

while (true) {

std::cout << "begin - 1\ntest - 2 \nexit - 0\n\n";

command = static\_cast<int>(Num\_input());

std::cout << "\n\n";

if (command == 1) {

Function();

}

else if (command == 2) {

Test();

std::cout << "\n\n";

}

else if (command == 0) {

std::cout << "prog has been finished";

std::cout << "\n\n";

break;

}

else {

std::cout << "command not found" << "\n" << "\n";

}

}

}

/\*

to do list:

1. use enum

2. make russuian

3. make tests

4. unint

5. checking is file empty DONE

C:\Users\Gleb\Desktop\for\_test\test.txt

1

C:\Users\Gleb\Desktop\folder\_fortests\file.txt

1

1

C:\Users\Gleb\Desktop\folder\_fortests\result.txt

test: name secondName "address" phone innerTime longDistansetime

4

1

0

Ivanov Gleb 8937340 2345345 0

Petrov Roman 8932342 345345 3245

Mazo Ivan 89123389 345345 0

Khamidullin Bogdan 3432445 5757656 5

C:\Users\Gleb\Desktop\folder\_fortests\file.txt

25

1

0

Ivanov Gleb "ufa 114" 8937340 2345345 0

Petrov Roman "ufa 114" 8932342 345345 3245

Mazo Ivan "ufa 114" 89123389 345345 0

Khamidullin Bogdan "ufa 114" 3432445 5757656 5

Smirnov Dmitry "ufa 114" 89112233 123456 4356

Sidorov Pavel "ufa 114" 8966778 9876543 0

Kuznetsov Andrey "ufa 114" 891234567 5436789 235

Volkov Sergey "ufa 114" 89991234 7891234 0

Fedorov Mikhail "ufa 114" 8900011 4567890 0

Mikhailov Alexey "ufa 114" 8987654 5678901 0

Egorov Alexander "ufa 114" 8956789 2345678 789

Vasiliev Anton "ufa 114" 89098765 1234567 4321

Alexandrov Artem "ufa 114" 89112345 5437890 0

Antonov Ivan "ufa 114" 8976543 9876543 0

Dmitriev Dmitry "ufa 114" 8912398 8765432 6578

Sergeev Sergey "ufa 114" 89567890 1234567 0

Nikolaev Nikolay "ufa 114" 8909876 1234567 233

Pavlov Pavel "ufa 114" 8911234 8765432 0

Maksimov Maxim "ufa 114" 8965432 3456789 0

Yakovlev Yakov "ufa 114" 89012345 2345678 0

Timofeev Timofey "ufa 114" 891234567 5678901 0

Grigoriev Grigory "ufa 114" 8987654 9876543 0

Zaitsev Roman "ufa 114" 8998765 1234567 0

Ivanov Igor "ufa 114" 89876543 9876543 0

Kozlov Konstantin "ufa 114" 8934821 2342342 0

5

0

Ivanov Gleb "ufa 114" 8937340 2345345 0

Petrov Roman "ufa 114" 8932342 345345 3245

Mazo Ivan "ufa 114" 89123389 345345 0

Khamidullin Bogdan "ufa 114" 3432445 5757656 5

Smirnov Dmitry "ufa 114" 89112233 123456 4356

1

1

C:\Users\Gleb\Desktop\folder\_fortests\file.txt

3

1

0

Ivanov Gleb "ufa 114" 8937340 2345345 0

Petrov Roman "ufa 114" 8932342 345345 3245

Mazo Ivan "ufa 114" 89123389 345345 0

for (int i = 0; i < size\_of\_book; i++) { //вывод массива

std::cout << book[i].GetSecondName() << "\t" << book[i].GetName() << "\n";

}

//AlphabetOrder(book, size\_of\_book);// вызов функции сортировки по алфавиту

LongDistanceCallsUser(book, size\_of\_book);

InnerСityСallsUserLimit(book, size\_of\_book, 345345);

\*/