

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | I | | | Группа | 414 |

Отчёт по контрольной работе № 1

Вариант № 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 414 |  |  |  | Кондиляброва  Вероника Данииловна |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

Содержание

[1 Постановка задачи 3](#_Toc111738132)

[2 Исходные данные 3](#_Toc111738133)

[3 Особые ситуации 3](#_Toc111738134)

[4 Математические методы и алгоритмы решения задач 3](#_Toc111738135)

[5 Форматы представления данных 4](#_Toc111738136)

[6 Структура программы 5](#_Toc111738137)

[7 Описание хода выполнения 6](#_Toc111738138)

[8 Блок-схема алгоритма решения задачи 7](#_Toc111738139)

[9 Результаты работы программы 11](#_Toc111738140)

[10 Исходный код полученного программного решения 17](#_Toc111738141)

## 1 Постановка задачи

Разработать программу для составления списка клиентов. Необходимо разработать класс для указанной предметной области. Доступ к данным реализовать с помощью методов Set, Get, Show.

Клиент: фамилия, имя, отчество, адрес, телефон, номер карты, номер банковского счета. Создать массив объектов. Реализовать возможность получения: – списка клиентов в алфавитном порядке, – списка клиентов, номер карты которых находится в заданном интервале.

## 2 Исходные данные

В качестве исходных данных программа использует количество клиентов и их данные: фамилия, имя, отчество, адрес, телефон, номер карты, номер банковского счета для составления и сортировки списка. Ввод с клавиатуры и с файла.

## 3 Особые ситуации

1 Пользователь может ввести число абитуриентов меньше одного, устранено с помощью функции GetInt, осуществляющей проверку ввода;

2 Пользователь вместо фамилии, имя, отчества и адреса может ввести цифры, устранено с помощью функции GetString, осуществляющей проверку ввода на символы;

3 Отсутствие ожидаемых программой файлов устранено методом open. Если программа не находит файл с введенным пользователем именем, этот метод сам его создает.

## 4 Математические методы и алгоритмы решения задач

Алгоритмы для сортировок

Сортировка по алфавиту пузырьковым методом с помощью двойного цикла for:

1. Прохождение по всему массиву;
2. Сравнивание между собой пар соседних ячеек;
3. Если при сравнении оказывается, что значение ячейки i больше, чем значение ячейки i+1, то мы меняем значения этих ячеек местами

Сортировка по номерам карт с помощью двух функций с циклом for:

1. Первая функция считает и возвращает количество нужных клиентов;
2. Вторая функция выделяет память под новый массив и заполняет его данными подходящих клиентов из старого массива

Алгоритм работы пользовательского интерфейса

1. Два цикла while внутренний и внешний отвечают за выход из блока ввода данных и за выход из программы соответственно;
2. Оператор множественного выбора switch – case осуществляет остальные действия в работе с пользователем

# 5 Форматы представления данных

Таблица 1 – Переменные

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| array\_client | Client\* | Массив клиентов |
| copy\_for\_sort\_numb\_card | Client\* | Скопированный массив клиентов для сортировки |
| copy\_for\_sort\_name | Client\* | Скопированный массив клиентов для сортировки |
| exit | bool | Пользовательский выбор выхода из меню |
| user\_choice | int | Пользовательский выбор пункта меню |
| low\_limit, top\_limit | int | Границы интервала обработки клиентов по номерам карт |
| count\_client | int | Пользовательский ввод количества клиентов |

Класс Client

Таблица 2 – Поля

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Тип |
| Surename | private | string |
| Name | private | string |
| Second\_name | private | string |
| City | private | string |
| Street | private | string |
| House | private | int |
| Number\_card | private | int |
| Number\_bank\_account | private | int |
| Phone\_numb | private | int |

Таблица 3 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| set | public | Ввод данных о клиенте |
| get | public | Передает каждое поле |
| show | public | Вывод всех данных о клиенте |

## 6 Структура программы

Таблица 4 – Структура программы

|  |  |  |
| --- | --- | --- |
| Модуль | Функция | Описание |
| Class | set | Ввод данных о клиенте |
|  | get | Передает каждое поле |
|  | show | Вывод всех данных о клиенте |
| Correct\_input | GetString | Проверка ввода букв |
|  | GetInt | Проверка ввода количества клиентов и в общем чисел |
| Function | CreateArray | Создаёт массив данных о клиенте |
|  | SortByName | Сортирует массив по алфавиту |
|  | SortByNumbercard | Сортирует по номерам карт клиентов |
|  | FoundCountSortCard | Возвращает количество подходящих клиентов для сортировки по номерам карт клиентов |
|  | InFile | Считывает данные клиентов с файла |
| Menu | MenuMain | Осуществляет работу пользовательского интерфейса |
|  | InputMenu | Создана для подменю ввода данных клиентов |
|  | MenuGet | Создана для подменю получения данных |
|  | MenuSave | Создана для подменю сохранения данных |
| Main | main | Вызывает функцию MenuMain, с которой начинается выполнение программы |
| Test | tests | Тестирует работу программы |
|  | Test\_comparison | Тестирует работу программы |

## 7 Описание хода выполнения

При выполнении контрольной работы я:

* Освоила работу с Visual Studio на языке С++;
* Изучила, что такое классы, поля, методы;
* Научилась разделять программу на модули;
* Освоила работу с файлами с помощью объектов ifstream, ofstream;

Решенные возникшие проблемы:

* Как создать класс;
* Как структурировать программу;
* Работа с типом данных unsigned long long int (проверка на ввод) – поменяно на int;
* Возвращение массива из функции – пользование указателем на массив;
* При возвращении указателя на массив его размер остаётся неизвестным – создание отдельной функции для нахождения размера;

Непонятные моменты:

* Как получить доступ к массиву объектов, который находится в отдельной в функции в блоке case или if-else?

## 

## 8 Блок-схема алгоритма решения задачи

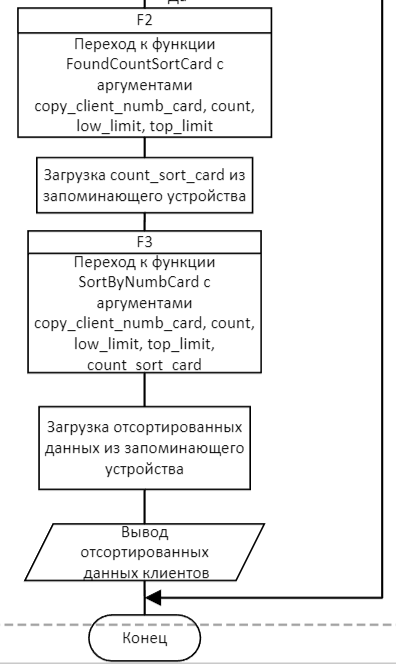
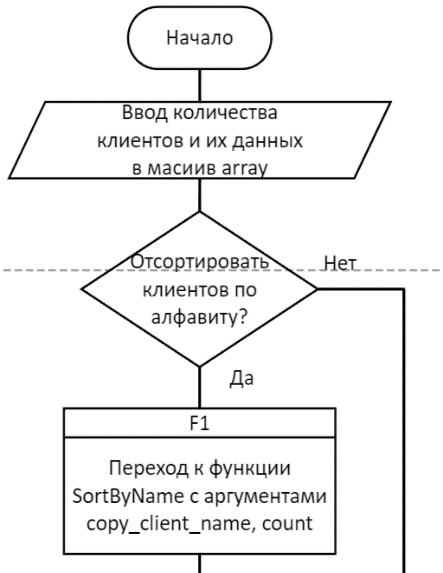
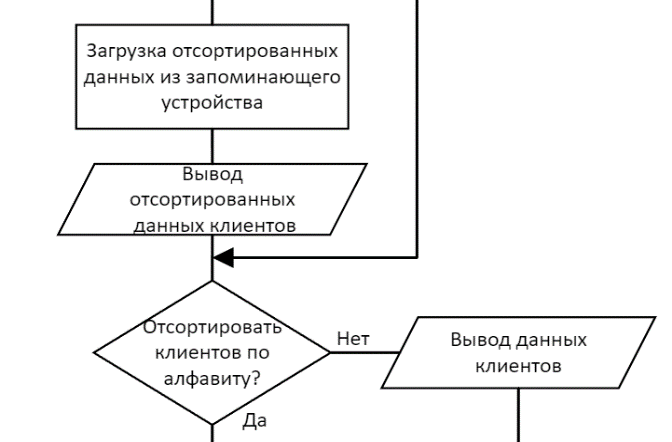
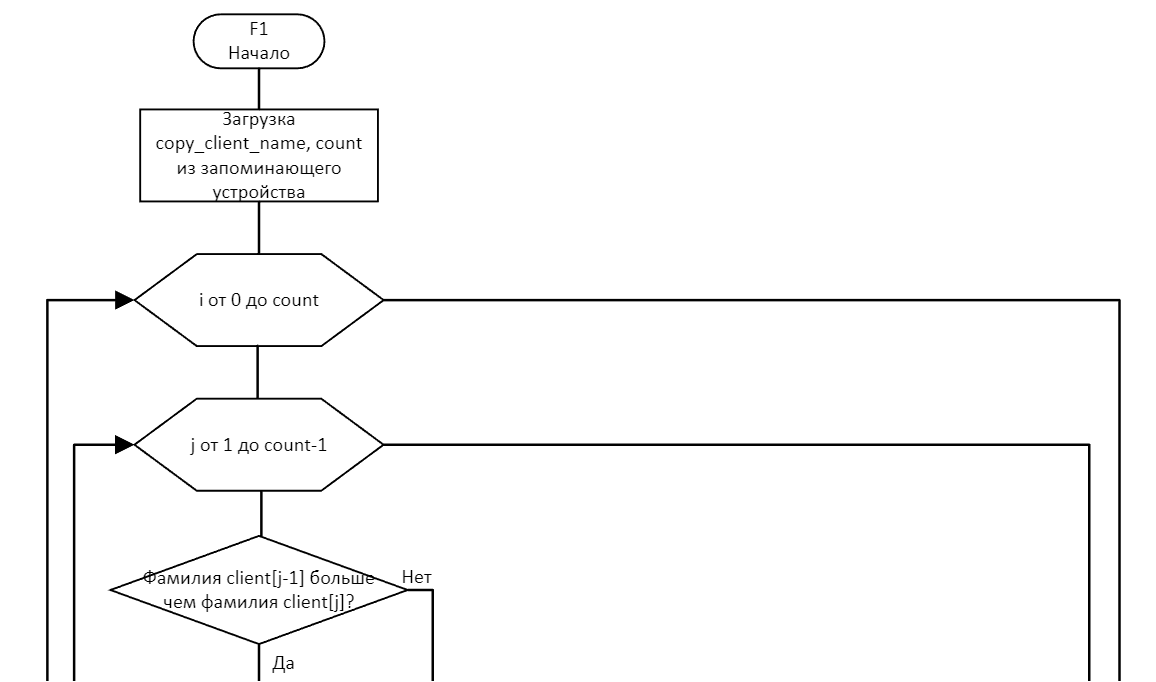


Рисунок 1 – Блок-схема алгоритма меню



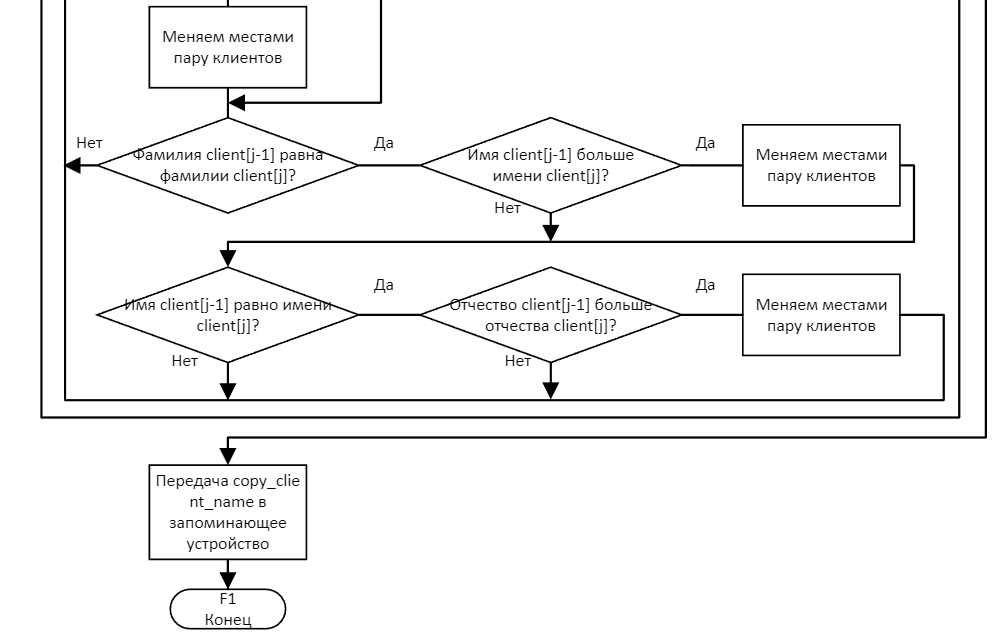


Рисунок 2 – Блок-схема алгоритма сортировки по алфавиту

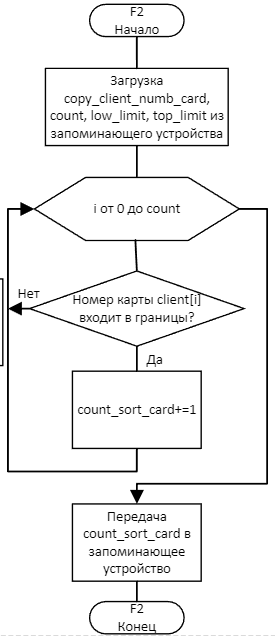


Рисунок 3 – Блок-схема вспомогательной функции для сортировки по номерам карт

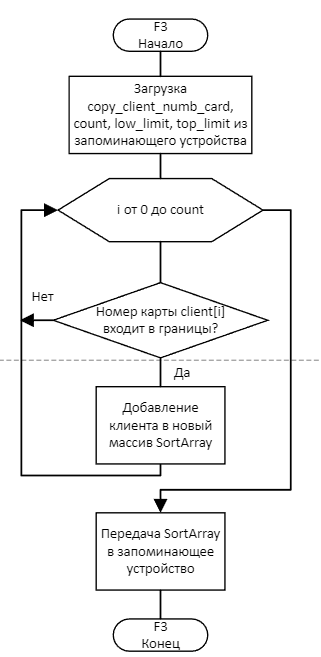


Рисунок 4 – Блок-схема алгоритма сортировки по номерам карт

## 9 Результаты работы программы

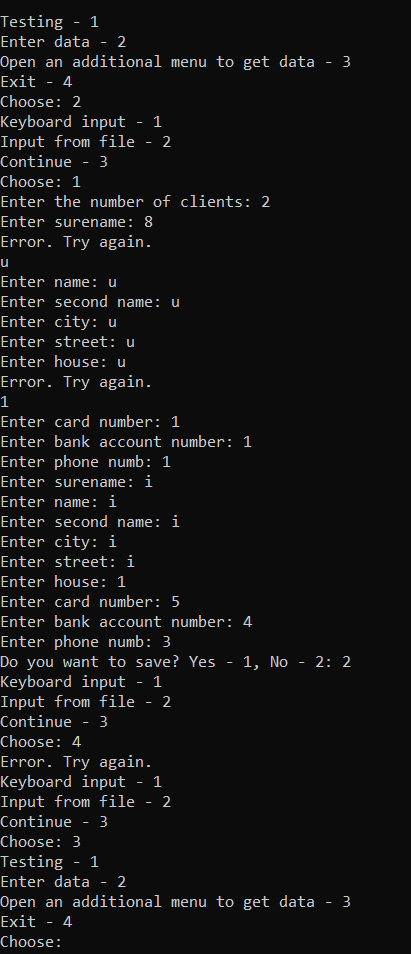


Рисунок 5 – Скриншот ввода данных

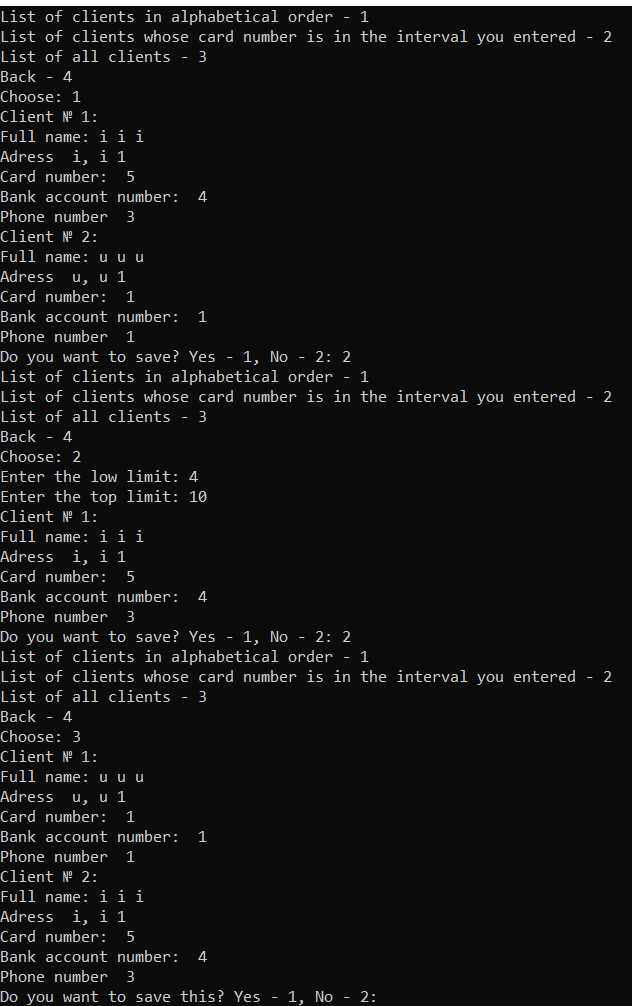


Рисунок 6 – Скриншот сортировки

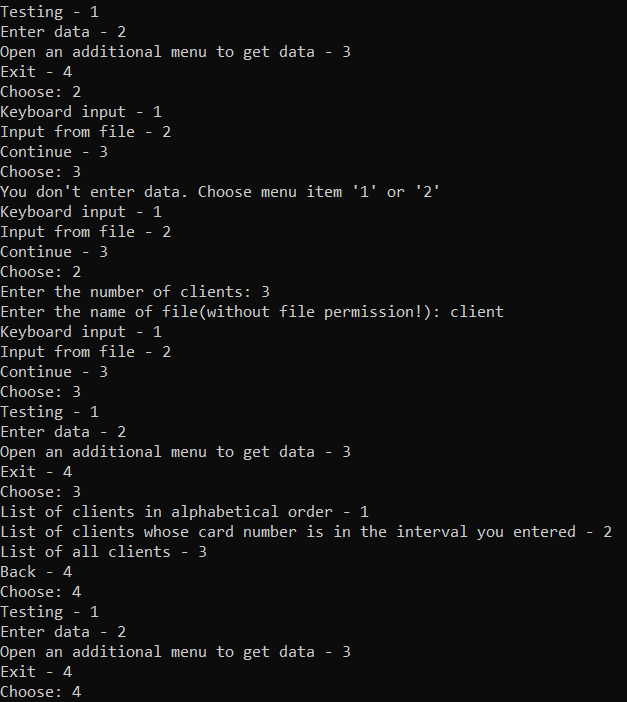


Рисунок 7 – Скриншот проверки работы меню

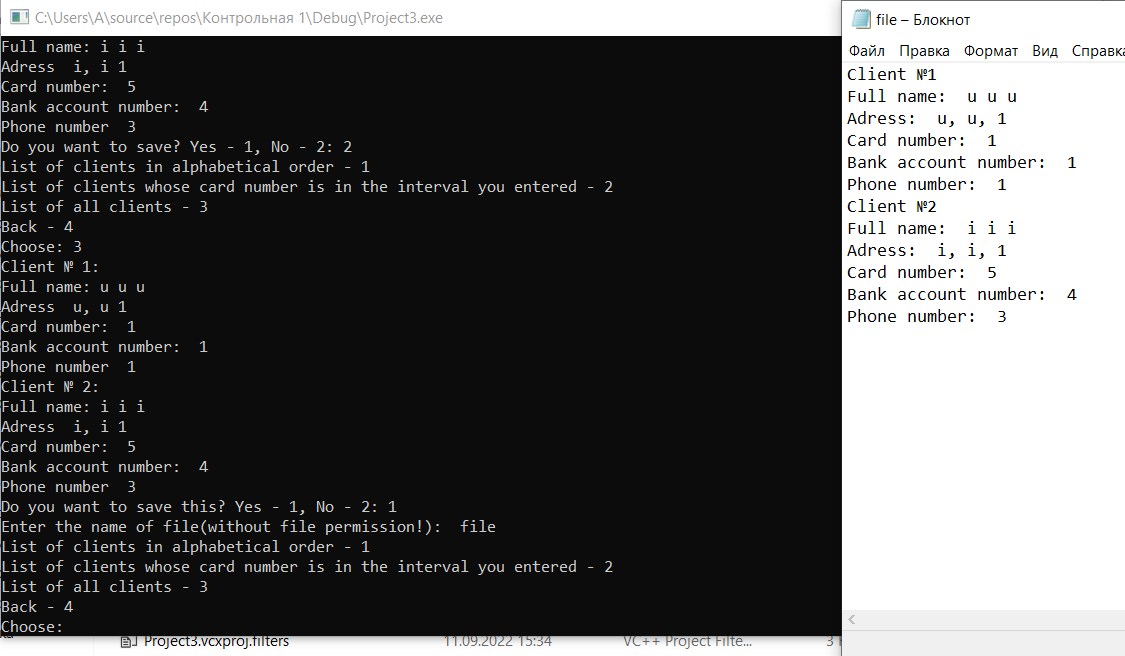


Рисунок 8 – Скриншот сохранения всех данных

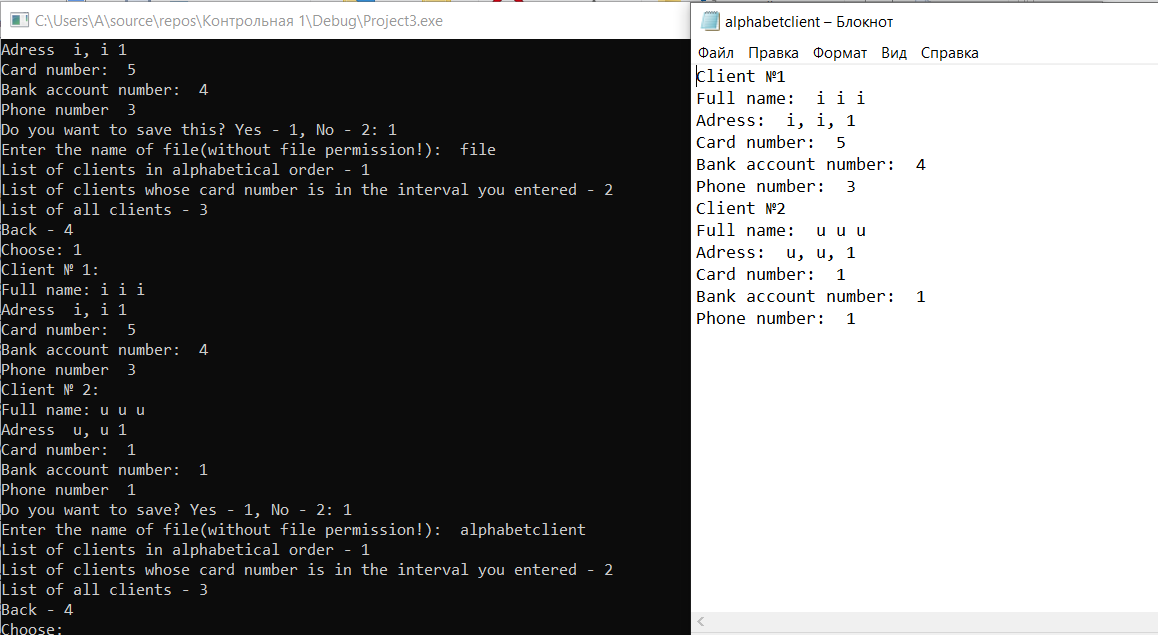


Рисунок 9 – Скриншот сохранения данных отсортированных по алфавиту

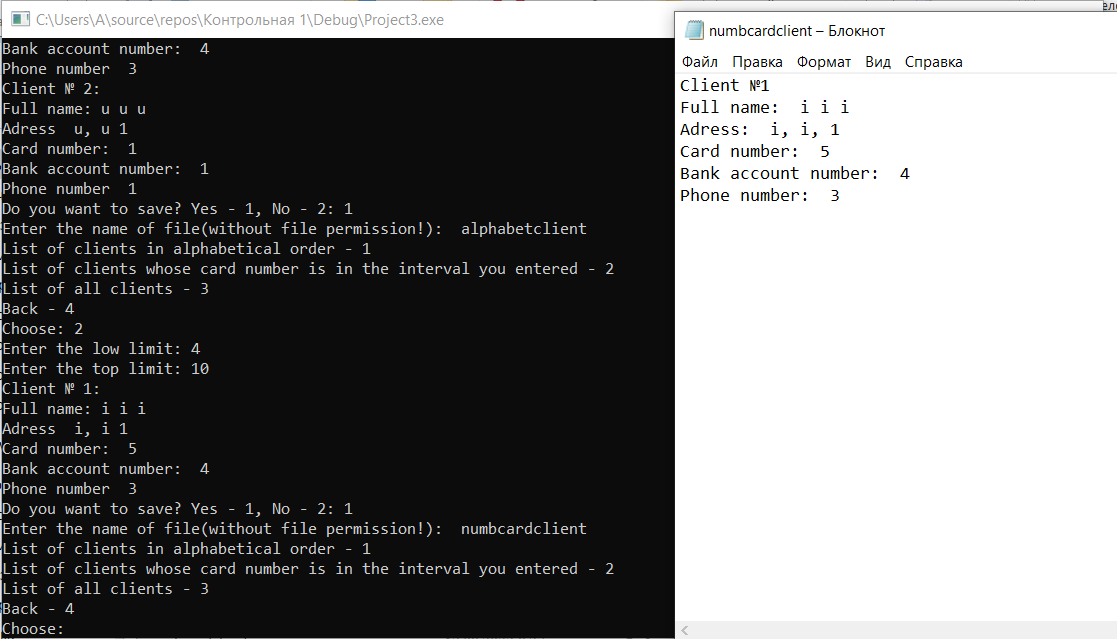


Рисунок 10 – Скриншот сохранения данных, отсортированных по номерам карт

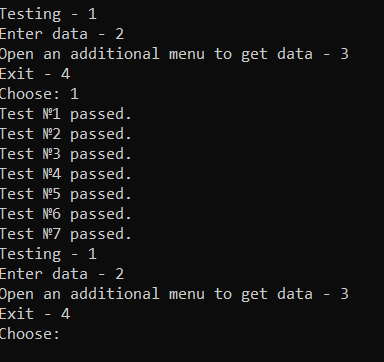


Рисунок 11 – Скриншот успешного тестирования

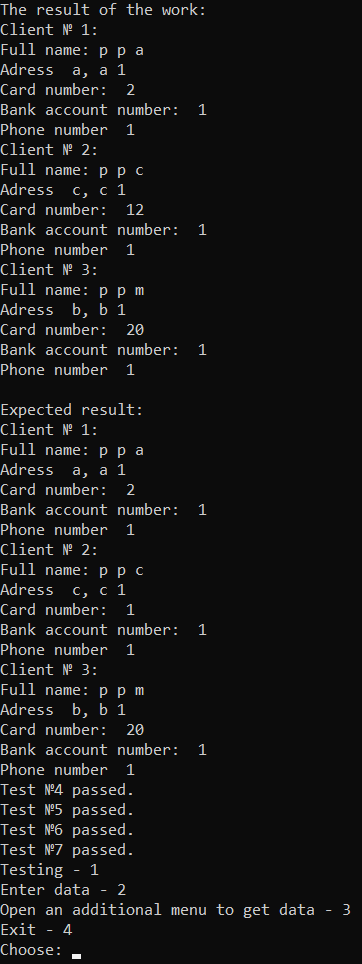
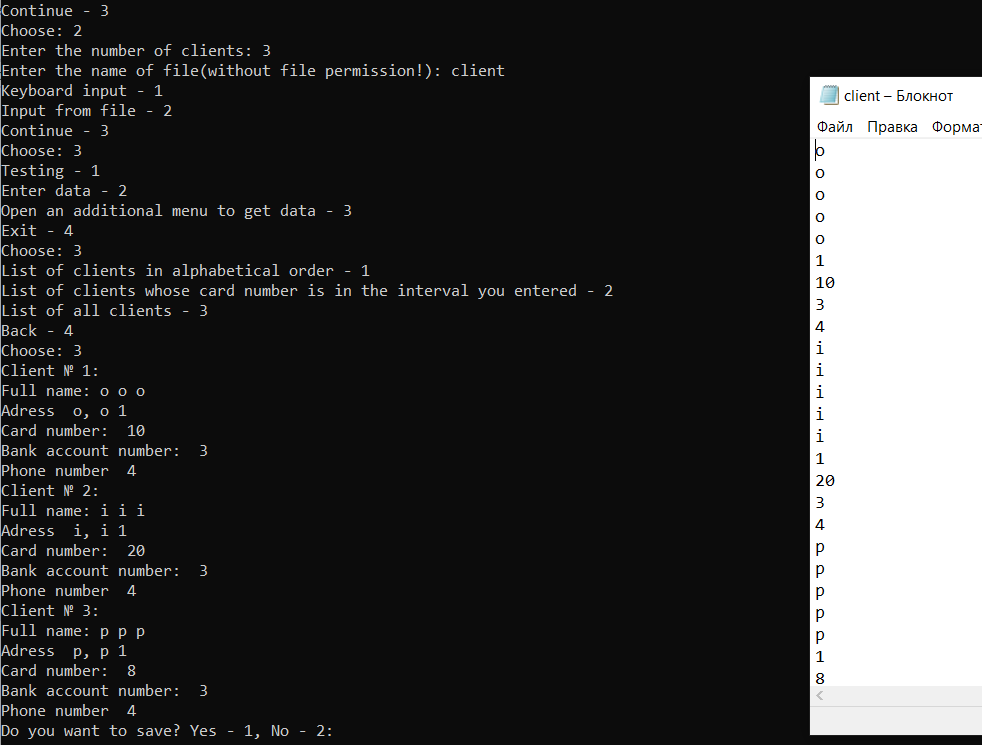


Рисунок 12 – Скриншот не пройденного тестирования



Скриншот 13 – Ввод данных из файла

Изображение выглядит как текст

Автоматически созданное описание

Скриншот 14 – Скриншот работы меню сохранения в файл

Изображение выглядит как текст

Автоматически созданное описание

Скриншот 15 – Скриншот сохранения в существующий файл

Изображение выглядит как текст

Автоматически созданное описание

Скриншот 16 – Скриншот сохранения в новый файл

# 10 Исходный код полученного программного решения

//Main.cpp

#include "Menu.h"

#include<iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "ru");

cout << "Kondilyabrova Veronika Daniilovna\nGroup 414" << endl;

cout << "Control work №1\nVariant 10\n\nDevelop a program for compiling a list of clients. \nIt is necessary to develop a class for the specified subject area. \nTo implement data access using the Set, Get, Show methods.\nClient: last name, first name, patronymic, address, phone number, card number, bank account number. \nCreate an array of objects. To realize the possibility of obtaining: a list of clients in alphabetical order, \na list of clients whose card number is in a given interval.\n" << endl;

MenuMain();

return 0;

}

//Class.h

#pragma once

#include<iostream>

#include <string>

using namespace std;

//Класс, описывающий клиента

class Client {

private:

string Surename = "";

string Name = "";

string Second\_name = "";

string City = "";

string Street = "";

int House = 0;

int Number\_card = 0;

int Numb\_bank\_account = 0;

int Phone\_numb = 0;

public:

Client();

Client(string Surename, string Name, string Second\_name, string City, string Street, int House, //

int Number\_card, int Numb\_bank\_account, int Phone\_numb);

string getSurename();

void setSurename(string Surename);

string getName();

void setName(string Name);

string getSecondName();

void setSecondName(string Second\_name);

string getCity();

void setCity(string City);

string getStreet();

void setStreet(string Street);

int getHouse();

void setHouse(int House);

int getNumberCard();

void setNumberCard(int Number\_card);

int getNumbBankAccount();

void setNumbBankAccount(int Numb\_bank\_account);

int getPhoneNumb();

void setPhoneNumb(int Phone\_numb);

void Show(int i);

};

//Class.cpp

#include "Class.h"

Client::Client() {

};

Client::Client(string Surename, string Name, string Second\_name,//

string City, string Street, int House, int Number\_card, //

int Numb\_bank\_account, int Phone\_numb) {

this->Surename = Surename;

this->Name = Name;

this->Second\_name = Second\_name;

this->City = City;

this->Street = Street;

this->House = House;

this->Number\_card = Number\_card;

this->Numb\_bank\_account = Numb\_bank\_account;

this->Phone\_numb = Phone\_numb;

}

//Методы класса

string Client::getSurename()

{

return Surename;

}

void Client::setSurename(string surename)

{

this->Surename =surename;

}

string Client::getName()

{

return Name;

}

void Client::setName(string name)

{

this->Name = name;

}

string Client::getSecondName()

{

return Second\_name;

}

void Client::setSecondName(string second\_name)

{

this->Second\_name = second\_name;

}

string Client::getCity()

{

return City;

}

void Client::setCity(string city)

{

this->City = city;

}

string Client::getStreet()

{

return Street;

}

void Client::setStreet(string street)

{

this->Street = street;

}

int Client::getHouse()

{

return House;

}

void Client::setHouse(int house)

{

this->House = house;

}

int Client::getNumberCard()

{

return Number\_card;

}

void Client::setNumberCard(int number\_card)

{

this->Number\_card = number\_card;

}

int Client::getNumbBankAccount()

{

return Numb\_bank\_account;

}

void Client::setNumbBankAccount(int numb\_bank\_account)

{

this->Numb\_bank\_account = numb\_bank\_account;

}

int Client::getPhoneNumb()

{

return Phone\_numb;

}

void Client::setPhoneNumb(int phone\_numb)

{

this->Phone\_numb = phone\_numb;

}

void Client::Show(int i) {

cout<< "Client № "<< i+1 << ":\n" << "Full name: "<< getSurename() <<" "<< getName() <<" "<< getSecondName() << "\nAdress " << getCity() << ", " << getStreet() << //

" " << getHouse() << "\nCard number: " << getNumberCard() << "\nBank account number: " << getNumbBankAccount() << "\nPhone number " << getPhoneNumb() << endl;

}

//Function.cpp

#include"Function.h"

//Создание массива клиентов

Client\* CreateArray(int count) {

Client\* client = new Client[count];

string Str = "";

int numb = 0;

int numbint = 0;

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

cout << "Enter surename: ";

Str = GetString();

client[i].setSurename(Str);

cout << "Enter name: ";

Str = GetString();

client[i].setName(Str);

cout << "Enter second name: ";

Str = GetString();

client[i].setSecondName(Str);

cout << "Enter city: ";

Str = GetString();

client[i].setCity(Str);

cout << "Enter street: ";

Str = GetString();

client[i].setStreet(Str);

cout << "Enter house: ";

numbint = GetInt();

client[i].setHouse(numbint);

cout << "Enter card number: ";

numb = GetInt();

client[i].setNumberCard(numb);

cout << "Enter bank account number: ";

numb = GetInt();

client[i].setNumbBankAccount(numb);

cout << "Enter phone numb: ";

numb = GetInt();

client[i].setPhoneNumb(numb);

}

return client;

}

//Запись массива клиентов из файла

Client\* FromFile(int count) {

Client\* client = new Client[count];

cout << "Enter the name of file(without file permission!): ";

string name\_file = "";

cin >> name\_file;

ifstream file;

file.open(name\_file);

int i = 0;

int k = 0;

//if (!(file >> count)) {

// cout << "Wrong number of clients" << endl;

// file.close();

//}

if (!file.is\_open()) {

cout << "File opening error" << endl;

}

else {

while (!file.eof()) {

k++;

if (k > count) {

cout << "Wrong number of clients" << endl;

file.close();

break;

}

string surename = "";

string name = "";

string second\_name = "";

string city = "";

string street = "";

int house{};

int number\_card{};

int numb\_bank\_account{};

int phone\_numb{};

file >> surename;

file >> name;

file >> second\_name;

file >> city;

file >> street;

file >> house;

file >> number\_card;

file >> numb\_bank\_account;

file >> phone\_numb;

client[i] = Client(surename, name, second\_name, city, street, house, number\_card, numb\_bank\_account, phone\_numb);

i++;

}

if (k == count || k < count) {

cout << "Download completed" << endl;

}

}

file.close();

return client;

}

//Сортировка по алфавиту

Client\* SortByName(Client \*copy\_client\_name, int count) {

Client tmp;

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

for (int j = 1; j < count; j++) {

if (copy\_client\_name[j - 1].getSurename() > copy\_client\_name[j].getSurename()) {

tmp = copy\_client\_name[j - 1];

copy\_client\_name[j - 1] = copy\_client\_name[j];

copy\_client\_name[j] = tmp;

}

if (copy\_client\_name[j - 1].getSurename() == copy\_client\_name[j].getSurename()) {

if (copy\_client\_name[j - 1].getName() > copy\_client\_name[j].getName()) {

tmp = copy\_client\_name[j - 1];

copy\_client\_name[j - 1] = copy\_client\_name[j];

copy\_client\_name[j] = tmp;

}

if (copy\_client\_name[j - 1].getName() == copy\_client\_name[j].getName()) {

if (copy\_client\_name[j - 1].getSecondName() > copy\_client\_name[j].getSecondName()) {

tmp = copy\_client\_name[j - 1];

copy\_client\_name[j - 1] = copy\_client\_name[j];

copy\_client\_name[j] = tmp;

}

}

}

}

}

return copy\_client\_name;

}

//Нахождение числа клиентов с номерами карт, входящими в заданный интервал

int FoundCountSortCard(Client \*copy\_client, int count, int low\_lim, int top\_lim) {

int count\_sort\_card = {};

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

if (copy\_client[i].getNumberCard() >= low\_lim && copy\_client[i].getNumberCard() <= top\_lim)

count\_sort\_card += 1;

}

return count\_sort\_card;

}

//Сортировка по номерам карт

Client\* SortByNumbercard(Client \*copy\_client, int count, int low\_lim, int top\_lim, int c\_sort\_card) {

Client\* SortArray = new Client[c\_sort\_card];

int tmp = c\_sort\_card;

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

if (copy\_client[i].getNumberCard() >= low\_lim && copy\_client[i].getNumberCard() <= top\_lim) {

SortArray[c\_sort\_card - tmp].setName(copy\_client[i].getName());

SortArray[c\_sort\_card - tmp].setSurename(copy\_client[i].getSurename());

SortArray[c\_sort\_card - tmp].setSecondName(copy\_client[i].getSecondName());

SortArray[c\_sort\_card - tmp].setCity(copy\_client[i].getCity());

SortArray[c\_sort\_card - tmp].setStreet(copy\_client[i].getStreet());

SortArray[c\_sort\_card - tmp].setHouse(copy\_client[i].getHouse());

SortArray[c\_sort\_card - tmp].setNumbBankAccount(copy\_client[i].getNumbBankAccount());

SortArray[c\_sort\_card - tmp].setNumberCard(copy\_client[i].getNumberCard());

SortArray[c\_sort\_card - tmp].setPhoneNumb(copy\_client[i].getPhoneNumb());

tmp--;

}

}

return SortArray;

}

//Проверка ввел ли пользователь данные

bool CheckExistanceData(Client\* client, int count) {

bool flag = true;

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

if (client[i].getName() == "" && client[i].getSurename() == "" && client[i].getSecondName() == "" && client[i].getCity() == "" && client[i].getStreet() == "" && client[i].getHouse() == 0 && client[i].getNumbBankAccount() == 0 && client[i].getNumberCard() == 0 && client[i].getPhoneNumb() == 0) {

flag = false;

}

}

return flag;

}

//Сохранение данных в файл

void InFile(Client\* client, int count) {

int tryAnotherFile{};

ofstream fout;

string name\_file{};

cout << "Enter the name of file(without file permission!): ";

cin >> name\_file;

if (ifstream(name\_file)) {

cout << "The file exists." << endl;

cout << "Overwrite existing file - 0\nEnter again - 1\nChoose: ";

tryAnotherFile = GetBool();

}

fout.open(name\_file);

// ф-ция open, если не находит файл с таким именем, сама его создает(app не затирает предыдущие данные).

if ((!fout.is\_open())) {

cout << "Fail opening error";

}

else {

if (tryAnotherFile == 0) {

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

fout << "Client №" << i + 1 << "\n" << "Full name: " << client[i].getSurename() << " " << client[i].getName() << " " << client[i].getSecondName() //

<< "\n" << "Adress: " << client[i].getCity() << ", " << client[i].getStreet() << ", " << client[i].getHouse() << " " << "\n" << "Card number: " //

<< client[i].getNumberCard() << "\n" << "Bank account number: " << client[i].getNumbBankAccount() << "\n" << "Phone number: " << client[i].getPhoneNumb() << "\n";

}

}

if (tryAnotherFile == 1) {

InFile(client, count);

}

}

fout.close();

}

//Function.h

#pragma once

#include<fstream>

#include"Class.h"

#include "Correct\_input.h"

using namespace std;

Client\* CreateArray(int count);

Client\* FromFile(int count);

Client\* SortByName(Client \*client, int count);

int FoundCountSortCard(Client \*copy\_client, int count, int low\_lim, int top\_lim);

Client\* SortByNumbercard(Client \*copy\_client, int count, int low\_lim, int top\_lim, int c\_sort\_card);

bool CheckExistanceData(Client\* client, int count);

void InFile(Client\* client, int count);

//Menu.cpp

#include<fstream>

#include"Menu.h"

int low\_limit = {};

int top\_limit = {};

int count\_sort\_card = {};

//Основное меню

void MenuMain() {

int Count = {};

Client\* array = nullptr;

bool exit = false;

while (exit == false) {

cout << "Testing - 1\nEnter data - 2\nOpen an additional menu to get data - 3\nExit - 4\nChoose: ";

int user\_choice = GetInt();

switch (user\_choice) {

case(Testing): {

Test();

}break;

case(Input\_Menu): {

array = InputMenu(Count);

cout << "Do you want to save this? Yes - 1, No - 2: ";

int save\_choice = GetSaveChoice();

if (save\_choice == 1) {

InFile(array, Count);

}

if (save\_choice == 2) {

break;

}

}break;

case(Get\_menu): {

bool flag = CheckExistanceData(array, Count);

if (flag == true) {

MenuGet(array, Count);

}

else {

cout << "You dont enter data. Choose menu item '2'" << endl;

}

}break;

case(Exit): {

exit = true;

}break;

default: {

cout << "Error. Try again." << endl;

}break;

}

}

delete[] array;

array = nullptr;

}

//Меню ввода данных

Client\* InputMenu(int &count) {

bool exit = false;

Client\* client = nullptr;

while (exit == false) {

int user\_choice{};

cout << "Keyboard input - 1\nInput from file - 2\nContinue - 3\nChoose: ";

user\_choice = GetInt();

switch (user\_choice) {

case(Input): {

cout << "Enter the number of clients: ";

count = GetInt();

client = CreateArray(count);

}break;

case(Of\_file): {

cout << "Enter the number of clients: ";

count = GetInt();

client = FromFile(count);

}break;

case(Continue): {

bool flag = CheckExistanceData(client, count);

if (flag == true) {

exit = true;

}

else {

cout << "You don't enter data" << endl;

}

}break;

default: {

cout << "Error. Try again." << endl;

}break;

}

}

return client;

}

//Меню вывода данных

void MenuGet(Client \*client, int count) {

int user\_choice = 0;

bool exit = false;

while (exit == false) {

cout << "List of clients in alphabetical order - 1\nList of clients whose card number is in the interval you entered - 2\nList of all clients - 3\nBack - 4\nChoose: ";

user\_choice = GetInt();

switch (user\_choice) {

case(Sort\_by\_alphabet): {

Client\* copy\_client\_name = new Client[count];

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

copy\_client\_name[i] = client[i];

}

copy\_client\_name = SortByName(copy\_client\_name, count);

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

copy\_client\_name[i].Show(i);

}

cout << "Do you want to save? Yes - 1, No - 2: ";

int save\_choice = GetSaveChoice();

if (save\_choice == 1) {

InFile(copy\_client\_name, count);

}

if (save\_choice == 2) {

break;

}

delete[] copy\_client\_name;

copy\_client\_name = nullptr;

}break;

case(Sort\_by\_numbercard): {

cout << "Enter the low limit: ";

cin >> low\_limit;

cout << "Enter the top limit: ";

cin >> top\_limit;

count\_sort\_card = FoundCountSortCard(client, count, low\_limit, top\_limit);

if (count\_sort\_card > 0) {

Client\* copy\_for\_sort\_numb\_card = new Client[count\_sort\_card];

copy\_for\_sort\_numb\_card = SortByNumbercard(client, count, low\_limit, top\_limit, count\_sort\_card);

for (int i = 0; i < count\_sort\_card; i++) {

copy\_for\_sort\_numb\_card[i].Show(i);

}

cout << "Do you want to save this? Yes - 1, No - 2: ";

int save\_choice = GetSaveChoice();

if (save\_choice == 1) {

InFile(copy\_for\_sort\_numb\_card, count\_sort\_card);

}

if (save\_choice == 2) {

break;

}

delete[]copy\_for\_sort\_numb\_card;

copy\_for\_sort\_numb\_card = nullptr;

}

else {

cout << "No clients" << endl;

}

}break;

case(Not\_sort): {

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

client[i].Show(i);

}

cout << "Do you want to save this? Yes - 1, No - 2: ";

int save\_choice = GetSaveChoice();

if (save\_choice == 1) {

InFile(client, count);

}

if (save\_choice == 2) {

break;

}

}break;

case(Back): {

exit = true;

}break;

default: {

cout << "Error. Try again." << endl;

}

}

}

}

//Menu.h

#pragma once

#include"Function.h"

#include"tests.h"

enum MainMenu {

Testing=1,

Input\_Menu,

Get\_menu,

Exit,

};

enum Input\_data\_menu {

Input=1,

Of\_file,

Continue,

};

enum Get\_data\_menu {

Sort\_by\_alphabet=1,

Sort\_by\_numbercard,

Not\_sort,

Back,

};

void MenuMain();

Client\* InputMenu(int &count);

void MenuGet(Client \*client, int count);

//Correct\_input.cpp

#include <iostream>

#include<string>

#include "Correct\_input.h"

using namespace std;

//Проверка ввода строки

string GetString() {

int count = 0;

int flag = 0;

string Enter;

while (flag != 1) {

cin >> Enter;

char Symbol;

int len = (int)Enter.length();

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

Symbol = Enter[i];

if (!((Symbol >= 'A' && Symbol <= 'Z') || (Symbol >= 'a' && Symbol <= 'z'))) {

count += 1;

}

}

if (count > 0) {

cout << "Error. Try again." << endl;

cin.clear();

cin.sync();

cin.ignore(INT\_MAX, '\n');

count = 0;

}

else {

flag = 1;

}

}

return Enter;

}

//Проверка ввода чисел

int GetInt() {

int input;

cin >> input;

while (!(input >= 1)) {

cout << "Error. Try again." << endl;

cin.clear();

cin.sync();

cin.ignore(INT\_MAX, '\n');

cin >> input;

}

return input;

}

//Проверка ввода выбора сохранения

int GetSaveChoice() {

int input;

cin >> input;

while (!(input >= 1 && input <= 2)) {

cout << "Error. Try again." << endl;

cin.clear();

cin.sync();

cin.ignore(INT\_MAX, '\n');

cin >> input;

}

return input;

}

//Проверка ввода выбора сохранения

bool GetBool() {

bool input;

cin >> input;

while (!(input == 0 || input ==1)) {

cout << "Error. Try again." << endl;

cin.clear();

cin.sync();

cin.ignore(INT\_MAX, '\n');

cin >> input;

}

return input;

}

//Correct\_input.h

#pragma once

#include <iostream>

#include<string>

using namespace std;

string GetString();

int GetInt();

int GetSaveChoice();

//Test\_comparison.cpp

#include "Test\_comparison.h"

#include"Function.h"

//Сравнительный тест сортировки по имени

void TestComparison(Client\* Work\_client, Client\* Result\_client, int size, int number) {

if ((Work\_client[0].getSurename() == Result\_client[0].getSurename()) && (Work\_client[0].getName() == Result\_client[0].getName())

&& (Work\_client[0].getSecondName() == Result\_client[0].getSecondName()) && (Work\_client[0].getCity() == Result\_client[0].getCity())

&& (Work\_client[0].getStreet() == Result\_client[0].getStreet()) && (Work\_client[0].getHouse() == Result\_client[0].getHouse())

&& (Work\_client[0].getNumberCard() == Result\_client[0].getNumberCard()) && (Work\_client[0].getNumbBankAccount() == Result\_client[0].getNumbBankAccount())

&& (Work\_client[0].getPhoneNumb() == Result\_client[0].getPhoneNumb()) && (Work\_client[1].getSurename() == Result\_client[1].getSurename()) && (Work\_client[1].getName() == Result\_client[1].getName())

&& (Work\_client[1].getSecondName() == Result\_client[1].getSecondName()) && (Work\_client[1].getCity() == Result\_client[1].getCity())

&& (Work\_client[1].getStreet() == Result\_client[1].getStreet()) && (Work\_client[1].getHouse() == Result\_client[1].getHouse())

&& (Work\_client[1].getNumberCard() == Result\_client[1].getNumberCard()) && (Work\_client[1].getNumbBankAccount() == Result\_client[1].getNumbBankAccount())

&& (Work\_client[1].getPhoneNumb() == Result\_client[1].getPhoneNumb()) && (Work\_client[2].getSurename() == Result\_client[2].getSurename()) && (Work\_client[2].getName() == Result\_client[2].getName())

&& (Work\_client[2].getSecondName() == Result\_client[2].getSecondName()) && (Work\_client[2].getCity() == Result\_client[2].getCity())

&& (Work\_client[2].getStreet() == Result\_client[2].getStreet()) && (Work\_client[2].getHouse() == Result\_client[2].getHouse())

&& (Work\_client[2].getNumberCard() == Result\_client[2].getNumberCard()) && (Work\_client[2].getNumbBankAccount() == Result\_client[2].getNumbBankAccount())

&& (Work\_client[2].getPhoneNumb() == Result\_client[2].getPhoneNumb()))

{

cout << "Test №" << number << " passed." << endl;

}

else {

cout << endl << "Test №" << number << " failed." << endl << endl;

cout << "The result of the work:" << endl;

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

Work\_client[i].Show(i);

}

cout << endl << "Expected result:" << endl;

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

Result\_client[i].Show(i);

}

}

}

//Сравнительный тест сортировки по номерам карт

void TestComparisonByCount(Client\* Work\_client, Client\* Result\_client, int Result\_size, int Work\_size, int number) {

if (Work\_size == Result\_size) {

bool flag = true;

for (int i = 0; i < Result\_size; i++) {

if ((Work\_client[i].getSurename() != Result\_client[i].getSurename()) || (Work\_client[i].getName() != Result\_client[i].getName())

|| (Work\_client[i].getSecondName() != Result\_client[i].getSecondName()) || (Work\_client[i].getCity() != Result\_client[i].getCity())

|| (Work\_client[i].getStreet() != Result\_client[i].getStreet()) || (Work\_client[i].getHouse() != Result\_client[i].getHouse())

|| (Work\_client[i].getNumberCard() != Result\_client[i].getNumberCard()) || (Work\_client[i].getNumbBankAccount() != Result\_client[i].getNumbBankAccount())

|| (Work\_client[i].getPhoneNumb() != Result\_client[i].getPhoneNumb()))

{

flag = false;

}

}

if (flag == true) {

cout << "Test №" << number << " passed." << endl;

}

else {

cout << endl << "Test №" << number << " failed." << endl << endl;

cout << "The result of the work:" << endl;

for (int i = 0; i < Work\_size; i++) {

Work\_client[i].Show(i);

}

cout << endl << "Expected result:" << endl;

for (int i = 0; i < Result\_size; i++) {

Result\_client[i].Show(i);

}

}

}

if (Work\_size != Result\_size) {

cout << endl << "Test №" << number << " failed." << endl << endl;

cout << "The result of the work:" << endl;

for (int i = 0; i < Work\_size; i++) {

Work\_client[i].Show(i);

}

cout << endl << "Expected result:" << endl;

for (int i = 0; i < Result\_size; i++) {

Result\_client[i].Show(i);

}

}

}

//Test\_comparison.h

#pragma once

#include"Class.h"

void TestComparison(Client\* Work\_client, Client\* Result\_client, int size, int number);

void TestComparisonByCount(Client\* Work\_client, Client\* Result\_client, int Result\_size, int Work\_size, int number);

//tests.cpp

#include"tests.h"

void Test() {

int Test\_count = 0;

int i = 1;

Client Test1[3] = { {"c", "c", "c", "c", "c", 1, 12, 1, 1}, {"a", "a", "a", "a", "a", 1, 2, 1, 1}, {"b", "b", "b", "b", "b", 1, 20, 1, 1} };

Client ResultTest1[3] = { {"a", "a", "a", "a", "a", 1, 2, 1, 1}, {"b", "b", "b", "b", "b", 1, 20, 1, 1}, {"c", "c", "c", "c", "c", 1, 12, 1, 1} };

\*Test1 = \*SortByName(Test1, 3);

TestComparison(Test1, ResultTest1, 3, i);

i += 1;

Client Test2[3] = { {"g", "d", "c", "c", "c", 1, 12, 1, 1}, {"g", "s", "a", "a", "a", 1, 2, 1, 1}, {"g", "a", "b", "b", "b", 1, 20, 1, 1} };

Client ResultTest2[3] = { {"g", "a", "b", "b", "b", 1, 20, 1, 1}, {"g", "d", "c", "c", "c", 1, 12, 1, 1}, {"g", "s", "a", "a", "a", 1, 2, 1, 1} };

\*Test2 = \*SortByName(Test2, 3);

TestComparison(Test2, ResultTest2, 3, i);

i += 1;

Client Test3[3] = { {"p", "p", "c", "c", "c", 1, 12, 1, 1}, {"p", "p", "a", "a", "a", 1, 2, 1, 1}, {"p", "p", "m", "b", "b", 1, 20, 1, 1} };

Client ResultTest3[3] = { {"p", "p", "a", "a", "a", 1, 2, 1, 1}, {"p", "p", "c", "c", "c", 1, 12, 1, 1}, {"p", "p", "m", "b", "b", 1, 20, 1, 1} };

\*Test3 = \*SortByName(Test3, 3);

TestComparison(Test3, ResultTest3, 3, i);

i += 1;

Client Test4[3] = { {"p", "p", "p", "c", "c", 1, 12, 1, 1}, {"p", "p", "p", "a", "a", 1, 2, 1, 1}, {"p", "p", "p", "b", "b", 1, 20, 1, 1} };

Client ResultTest4[3] = { {"p", "p", "p", "c", "c", 1, 12, 1, 1}, {"p", "p", "p", "a", "a", 1, 2, 1, 1}, {"p", "p", "p", "b", "b", 1, 20, 1, 1} };

\*Test4 = \*SortByName(Test4, 3);

TestComparison(Test4, ResultTest4, 3, i);

i += 1;

Client Test5[3] = { {"p", "p", "p", "c", "c", 1, 12, 1, 1}, {"p", "p", "p", "a", "a", 1, 2, 1, 1}, {"p", "p", "p", "b", "b", 1, 20, 1, 1} };

Client ResultTest5[2] = { {"p", "p", "p", "c", "c", 1, 12, 1, 1}, {"p", "p", "p", "b", "b", 1, 20, 1, 1} };

int Result\_test\_count1 = 2;

Test\_count = FoundCountSortCard(Test5, 3, 10, 25);

Client\* test5 = new Client[Test\_count];

test5 = SortByNumbercard(Test5, 3, 10, 25, Test\_count);

TestComparisonByCount(test5, ResultTest5, Result\_test\_count1, Test\_count, i);

i += 1;

Client Test6[3] = { {"p", "p", "p", "c", "c", 1, 12, 1, 1}, {"p", "p", "p", "a", "a", 1, 20, 1, 1}, {"p", "p", "p", "b", "b", 1, 2, 1, 1} };

Client ResultTest6[1] = { {"p", "p", "p", "b", "b", 1, 2, 1, 1} };

int Result\_test\_count2 = 1;

Test\_count = FoundCountSortCard(Test6, 3, 1, 5);

Client\* test6 = new Client[Test\_count];

test6 = SortByNumbercard(Test6, 3, 1, 5, Test\_count);

TestComparisonByCount(test6, ResultTest6, Result\_test\_count2, Test\_count, i);

i += 1;

Client Test7[3] = { {"p", "p", "p", "c", "c", 1, 12, 1, 1}, {"p", "p", "p", "a", "a", 1, 2, 1, 1}, {"p", "p", "p", "b", "b", 1, 20, 1, 1} };

int Result\_test\_count3 = 0;

Test\_count = FoundCountSortCard(Test7, 3, 25, 50);

if (Result\_test\_count3 == Test\_count) {

cout << "Test №" << i << " passed." << endl;

}

i += 1;

}

//tests.h

#pragma once

#include "Menu.h"

#include<fstream>

#include"Test\_comparison.h"

void Test();