Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Уфимский государственный авиационный технический университет»

Кафедра технической кибернетики

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *%* | Ритмичность сдачи этапов | | | | | | | | | | | | | | | | | |
| 100 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|  | *Учебная неделя* | | | | | | | | | | | | | | | | | |

**Проектирование классов с использованием механизмов наследования, перегрузки операций ввода/вывода и присваивания в языке С++ для обработки файловых данных**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине **«Методы современного программирования»**

|  |
| --- |
| **1304.320007.000 ПЗ** |

(обозначение документа)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ИВТ-321* | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | *Самсонова Е.О.* |  |  |  |
| Консультант | *Хасанов А.Ю.* |  |  |  |
| Принял | *Хасанов А.Ю.* |  |  |  |

Уфа 2022

**Аннотация**

Настоящий курсовой проект посвящен проектированию классов массивов структур в среде С++ с использованием механизмов наследования, перегрузки операций присваивания, операций сравнения, ввода/вывода в файлы и на экран для различных видов обработки информационных записей и содержащей меню, позволяющий выполнить и проверить работу всех методов разработанных классов.

В данной программе использован основной класс, класс поисков и класс перечней, разработанные на основе иерархии классов и использующие перегрузки операций ввода/вывода и методы: добавление/удаление записей, алфавитные и числовые сортировки, создание перечней и их сортировки, поиск по заданным полям и их сортировки.

Программа реализована в Visual Studio.

**Оглавление**

[Введение 4](#_Toc7284501)

[Описание разработанных функций 5](#_Toc7284502)

[Тестирование программы 7](#_Toc7284503)

[Заключение 7](#_Toc7284504)

[Использованная литература 35](#_Toc7284505)

[Код программы 36](#_Toc7284506)

# **Введение**

**Цель работы:** в ходе решения поставленных задач ознакомиться и закрепить основные понятия программирования на языке высокого уровня С++, освоить навыки разработки классов на основе механизма наследования для различных видов обработки данных с использованием перегрузки операций.

**Задание:** спроектировать структуру PERSON, содержащую поля: фамилия и инициалы, район, адрес, телефон.

Спроектировать структуру ABONENT, содержащую поле типа PERSON и дополнительно поля: номер договора, дата заключения, абонентская плата помесячно.

Файл содержит информацию об абонентах кабельного телевидения. Требуется спроектировать класс, позволяющий обрабатывать эти данные. В классе должно быть предусмотрено, как минимум, выполнение следующих видов обработки:

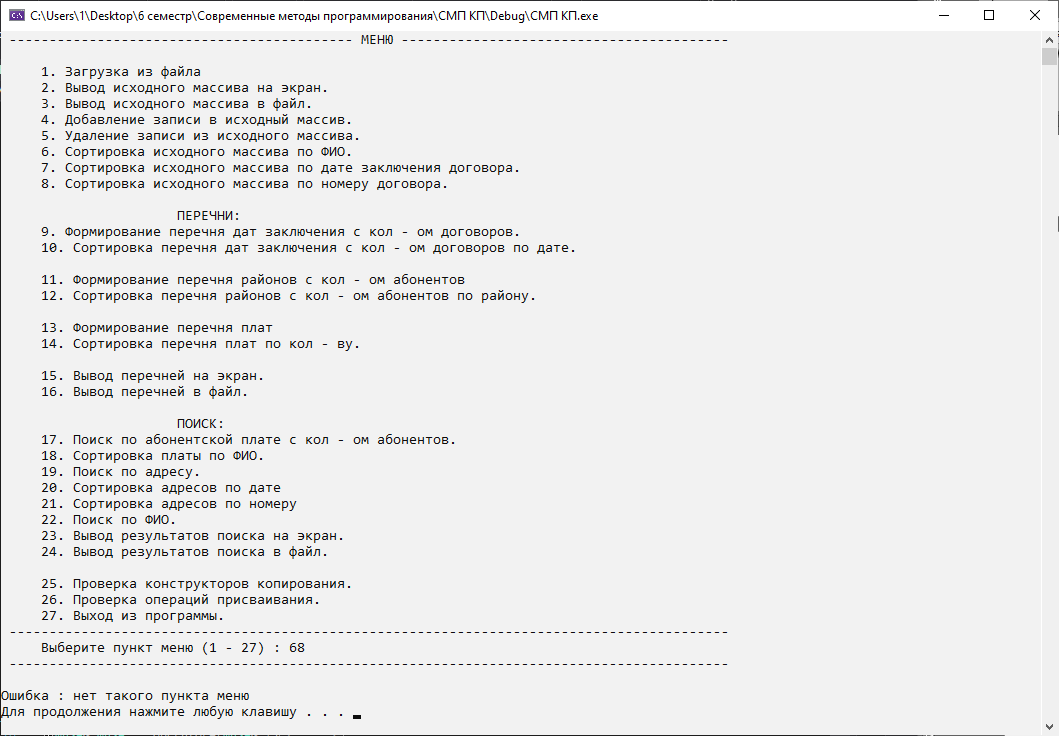
1. Загрузка из файла;
2. Сохранение результатов обработки в файлах;
3. Добавление записи;
4. Удаление записи;
5. Вывод на экран дисплея данных и результатов обработки;
6. Алфавитная сортировка по Ф.И.О.;
7. Сортировка по дате заключения договора;
8. Числовая сортировка по номеру договора;
9. Перечень дат заключения с указанием количества договоров (результат отсортирован по дате заключения);
10. Перечень районов с указанием количества абонентов (результат отсортирован по району);
11. Поиск по абонентской плате (результат отсортирован по Ф.И.О.);
12. Перечень абонентских плат с указанием числа абонентов с такой платой (результат отсортирован по численности);
13. Поиск по адресу (результат отсортирован по дате заключения);
14. Поиск по адресу (результат отсортирован по номеру договора);
15. Поиск по Ф.И.О. информации об абоненте кабельного телевидения.

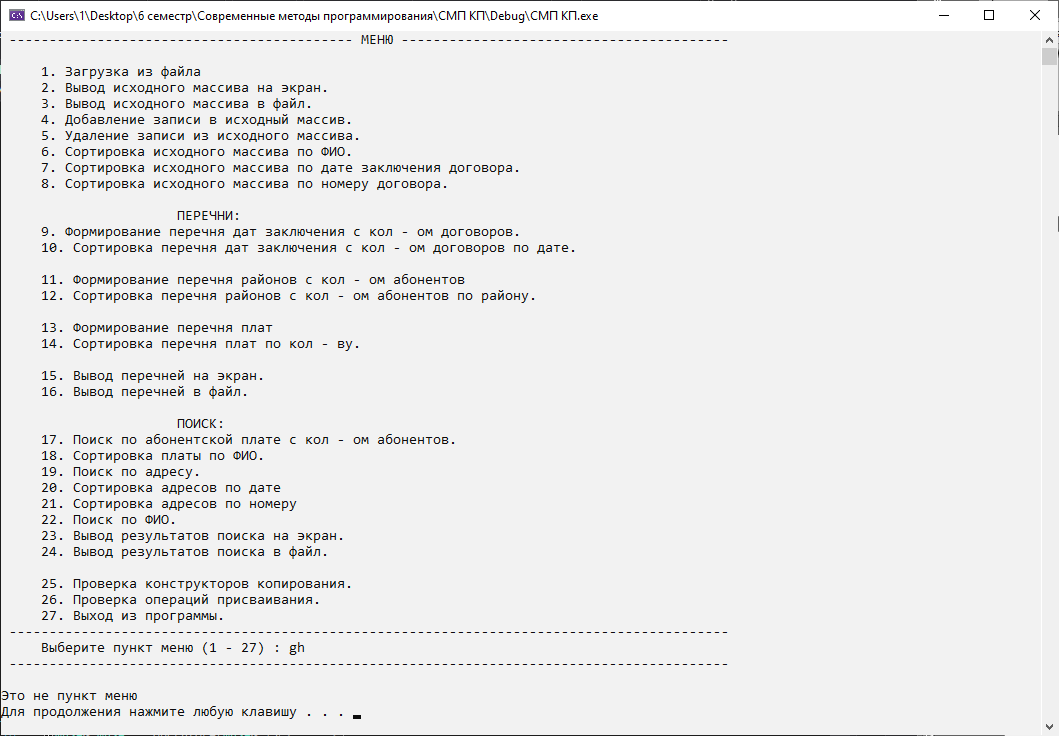
# **Описание разработанных функций**

|  |  |  |
| --- | --- | --- |
| Название функции | Параметры функции | Назначение функции |
| main | - | Главная подпрограмма, содержащая доступ к меню |
| Класс masA | | |
| masA | - | Конструктор, задающий начальное значение |
| masA | masA& z | Конструктор копирования |
| ~masA |  | Деструктор |
| masA & operator= | (masA& z) | Перегруженная операция присваивания |
| friend ostream operator << | (ostream &out, masA& z) | Перегруженная операция вывода на экран |
| friend ofstream operator << | (ofstream &out, masA& z) | Перегруженная операция вывода в файл |
| friend ifstream operator >> | (ifstream &in, masA& z) | Перегруженная операция ввода из файла |
| addAbonent | - | Добавление записи |
| deleteAbonent | - | Удаление записи |
| sortName | - | Сортировка по Ф.И.О. |
| sortDat | - | Сортировка по дате заключения договора |
| sortNumber | - | Сортировка по номеру договора |
| Класс masB | | |
| masB | - | Конструктор, задающий начальное значение |
| masB | (masB& z) | Конструктор копирования |
| ~ masB |  | Деструктор |
| masB& operator = | (masB& z) | Перегруженная операция присваивания |
| friend ostream operator << | (ostream &out, masB& z) | Перегруженная операция вывода на экран |
| friend ofstream operator << | (ofstream &out, masB& z) | Перегруженная операция вывода в файл |
| makePerechRegions | - | Создание перечня районов |
| makePerechDates | - | Создание перечня дат заключения договоров |
| makePerechPlata | - | Создание перечня абонентских плат |
| sortRegions | - | Сортировка перечня районов по району |
| sortDates | - | Сортировка перечня дат заключения по дате заключения |
| sortCount | - | Сортировка перечня абонентских плат по численности |
| Класс masC | | |
| masC | - | Конструктор, задающий начальное значение |
| masC | (masC& z) | Конструктор копирования |
| ~ masC |  | Деструктор |
| masC& operator = | (masC& z) | Перегруженная операция присваивания |
| friend ostream operator << | (ostream &out, masC& z) | Перегруженная операция вывода на экран |
| friend ofstream operator << | (ofstream &out, masC& z) | Перегруженная операция вывода в файл |
| findPlata | - | Поиск по абонентской плате |
| findAddress | - | Поиск по адресу |
| findbyFIO | - | Поиск по Ф.И.О. |
| sortPlataFIO | - | Сортировка массива поиска по абонентской плате по Ф.И.О. |
| sortAddressDate | - | Сортировка массива поиска по адресу по дате заключения договора |
| sortAddressNum | - | Сортировка массива поиска по адресу по номеру договора |

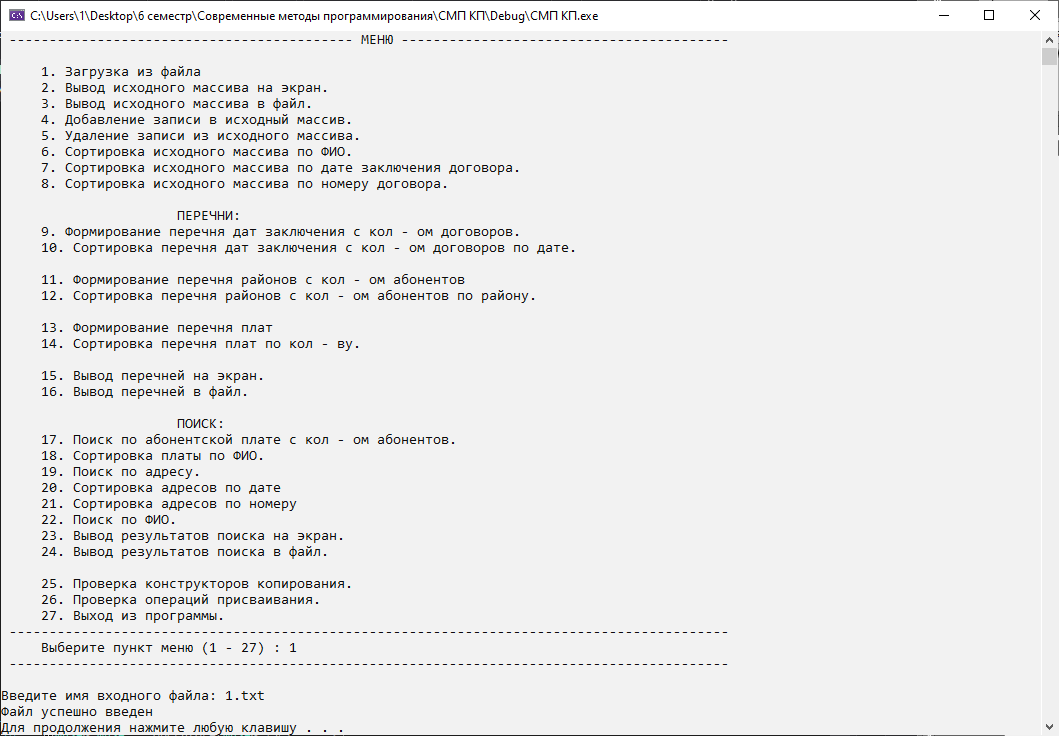
# **Тестирование программы**

**Меню и проверка на ввод пункта меню**

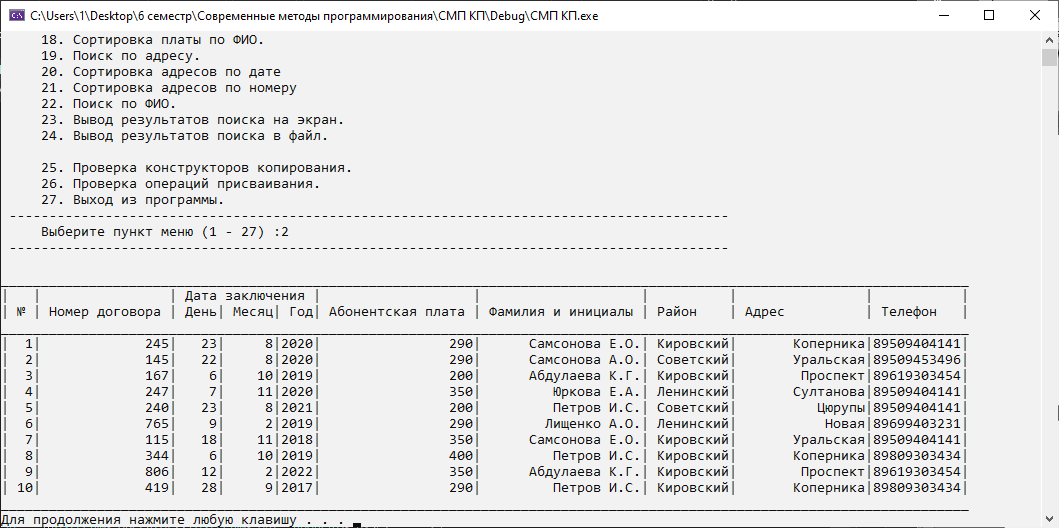




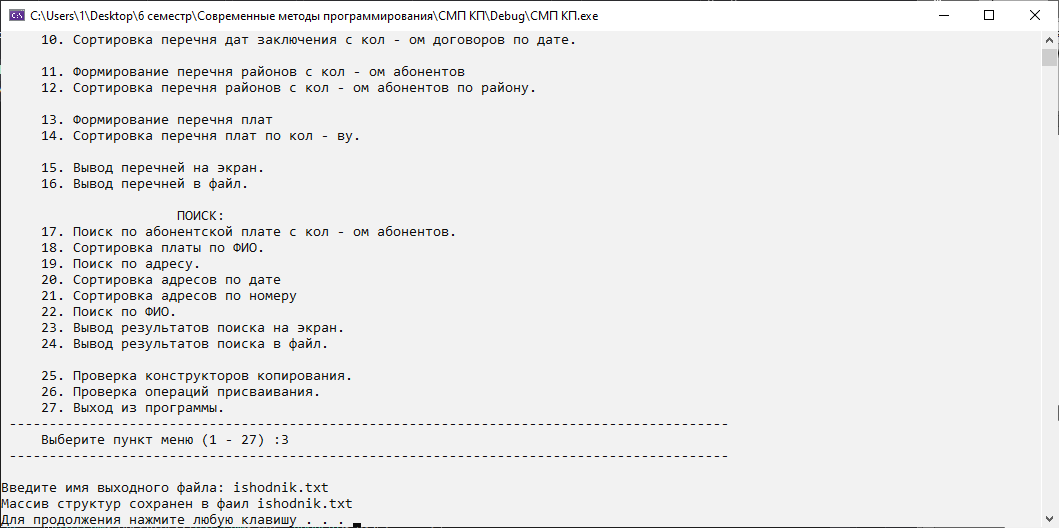
**Загрузка из файла**

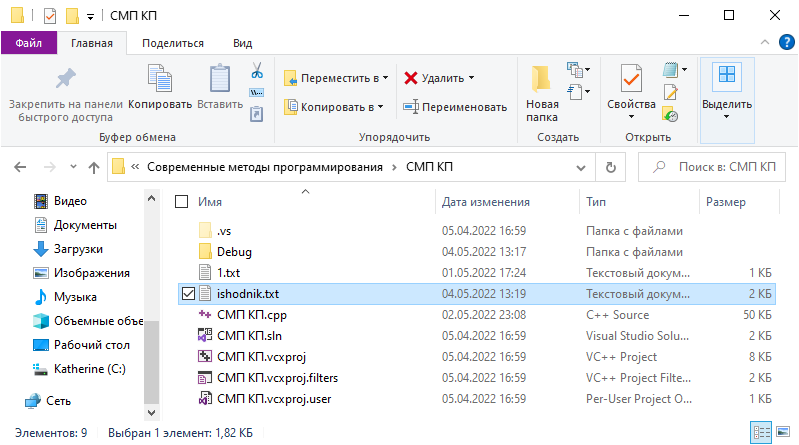


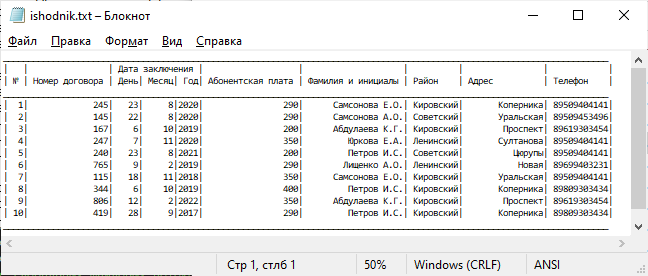
**Вывод исходного массива на экран**



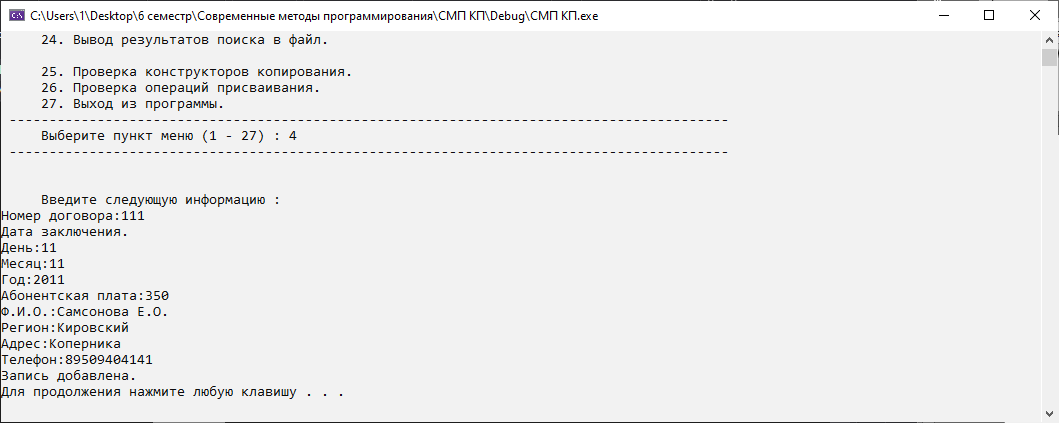
**Вывод исходного массива в файл**

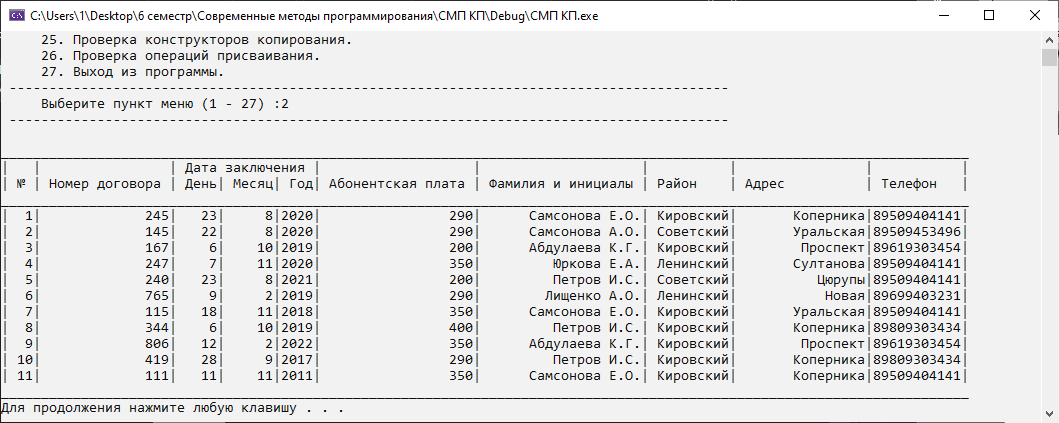




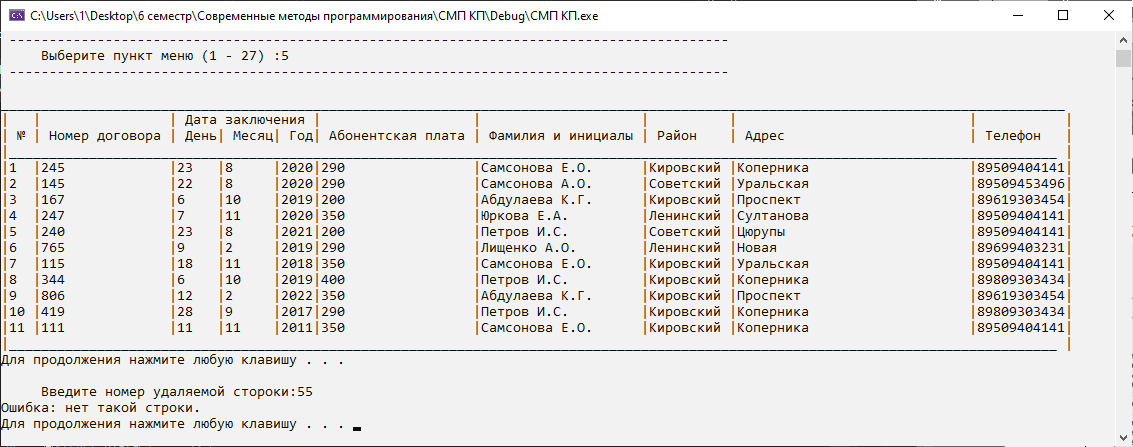


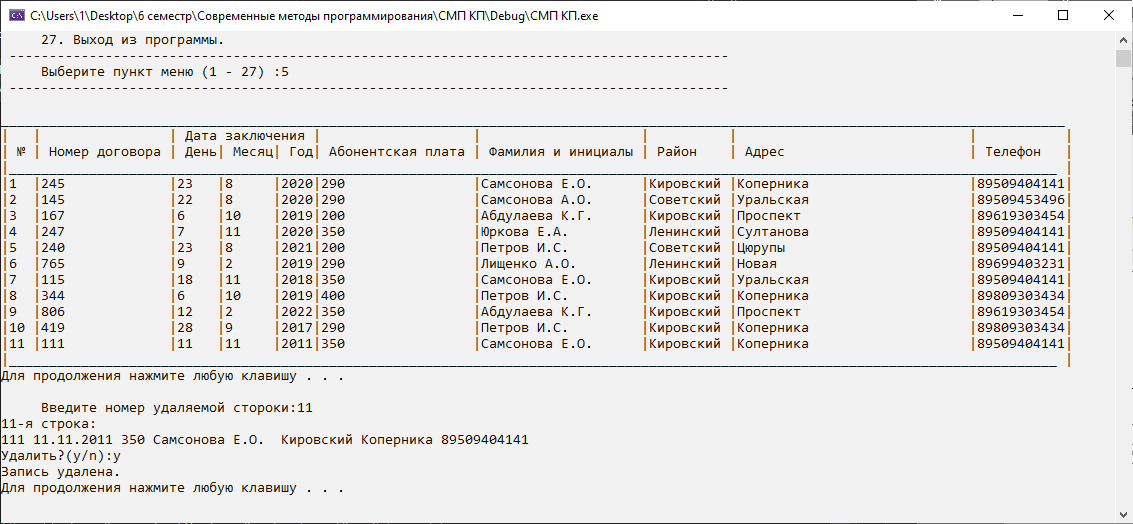
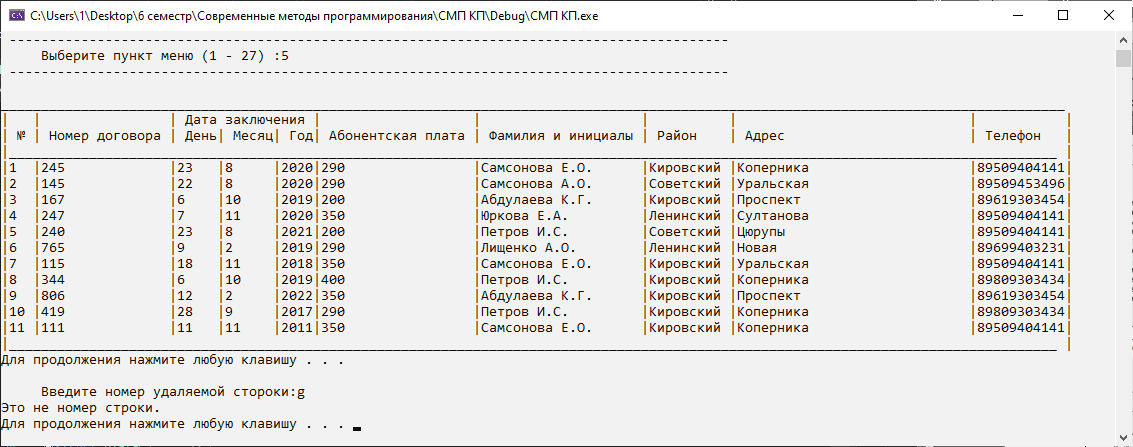
**Добавление записи в исходный массив**



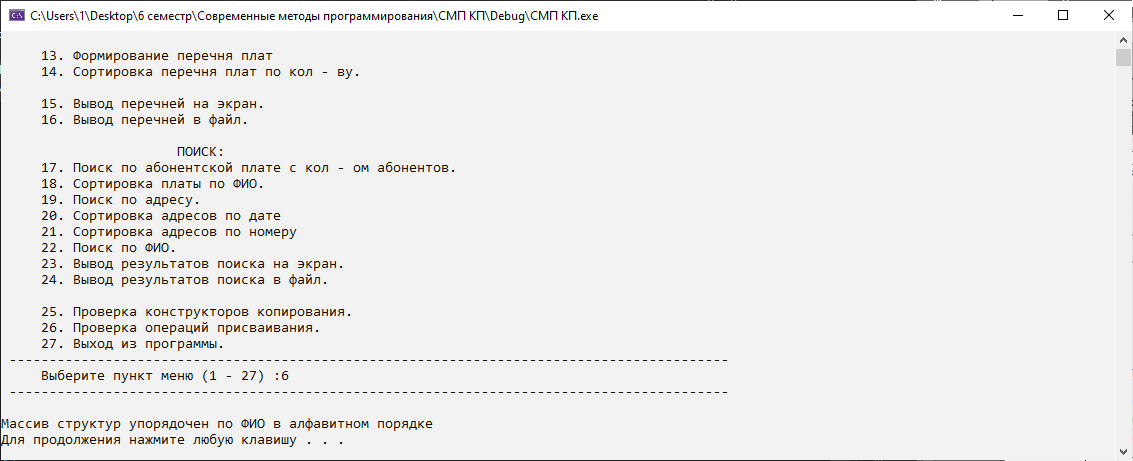


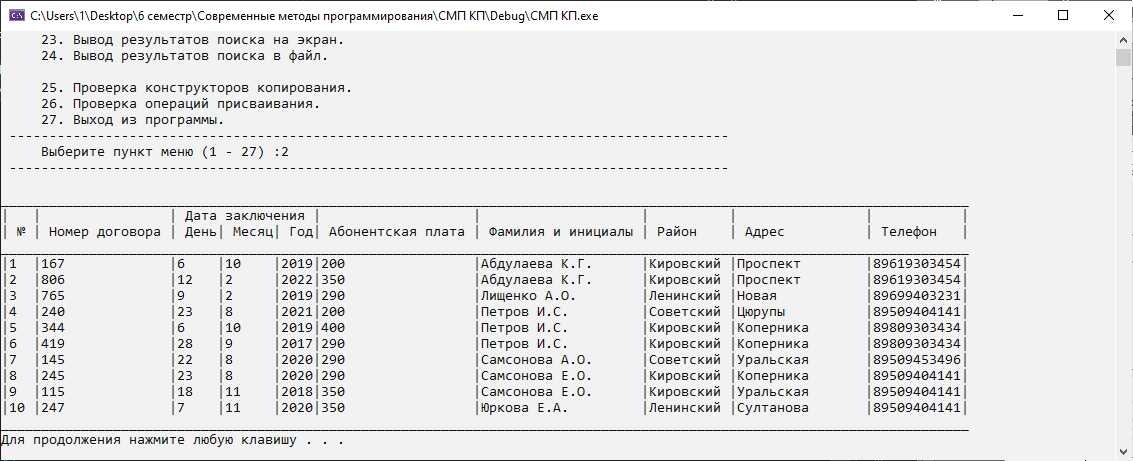
**Удаление записи из исходного массива**

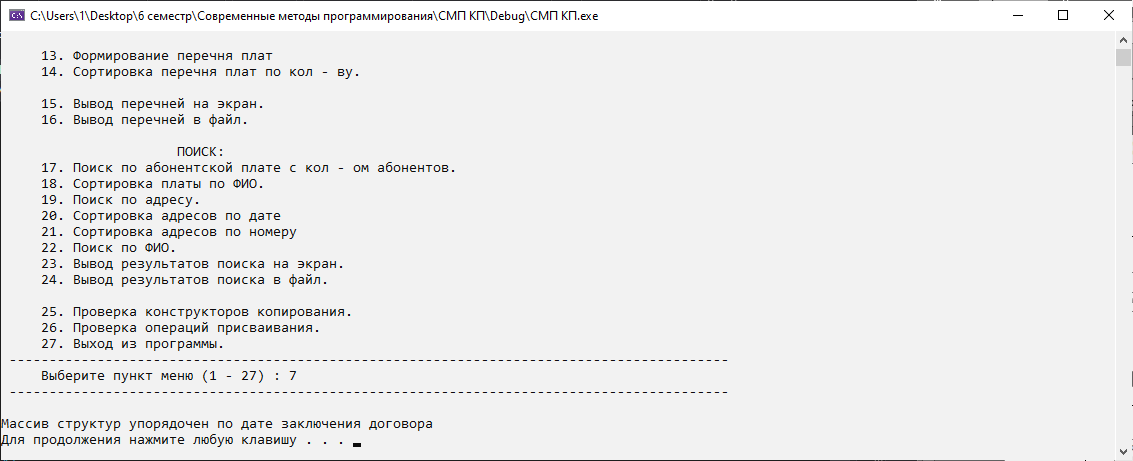


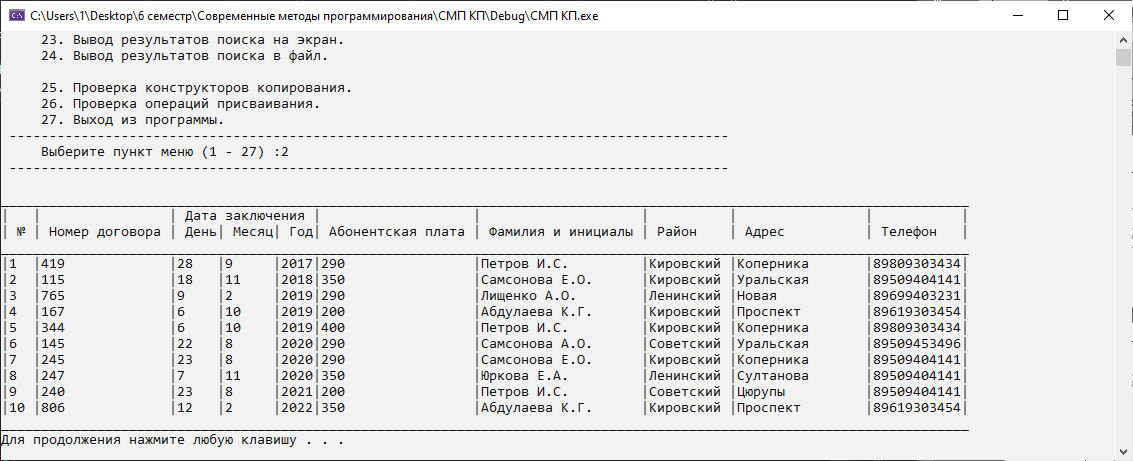


**Сортировка исходного массива по Ф.И.О.**

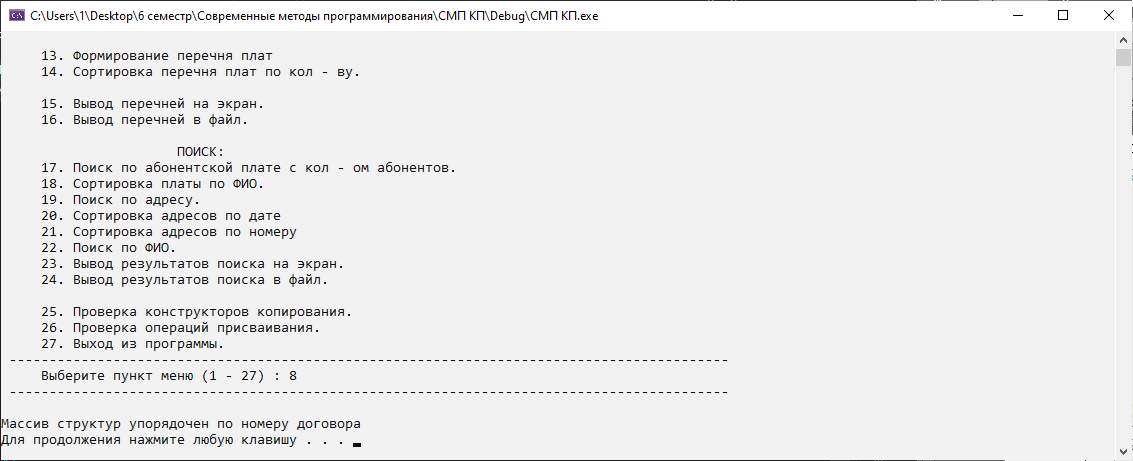


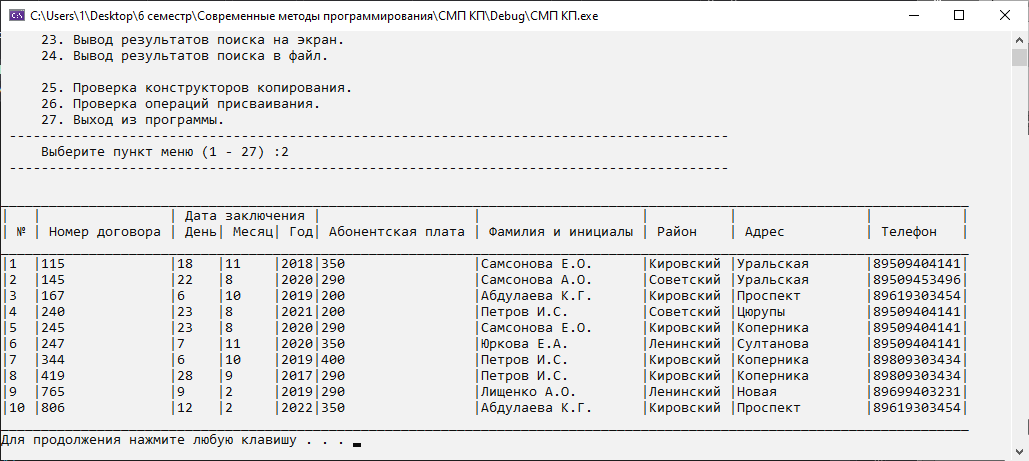
 **Сортировка исходного массива по дате заключения договора**



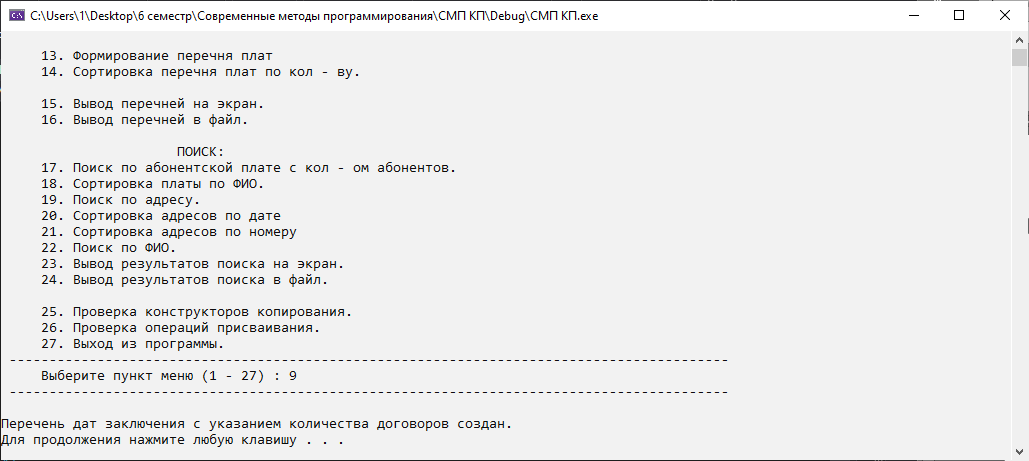


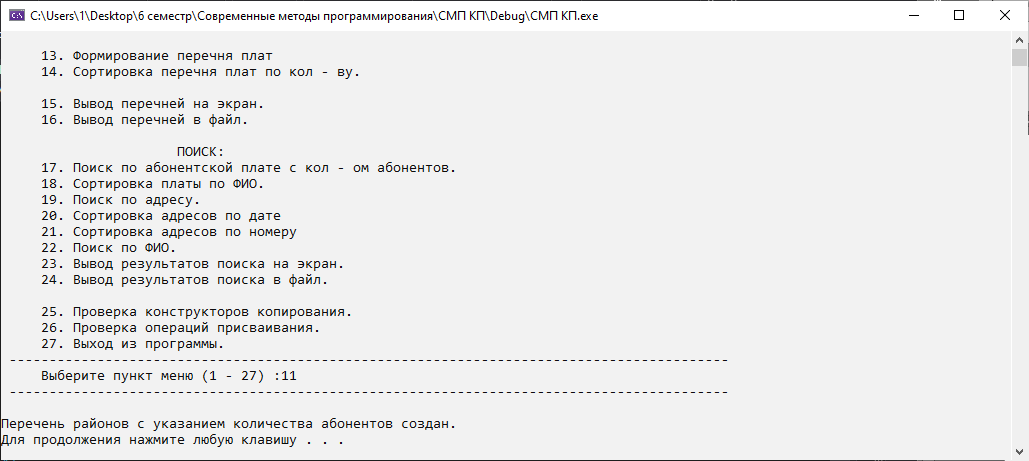
**Сортировка исходного массива по номеру договора**

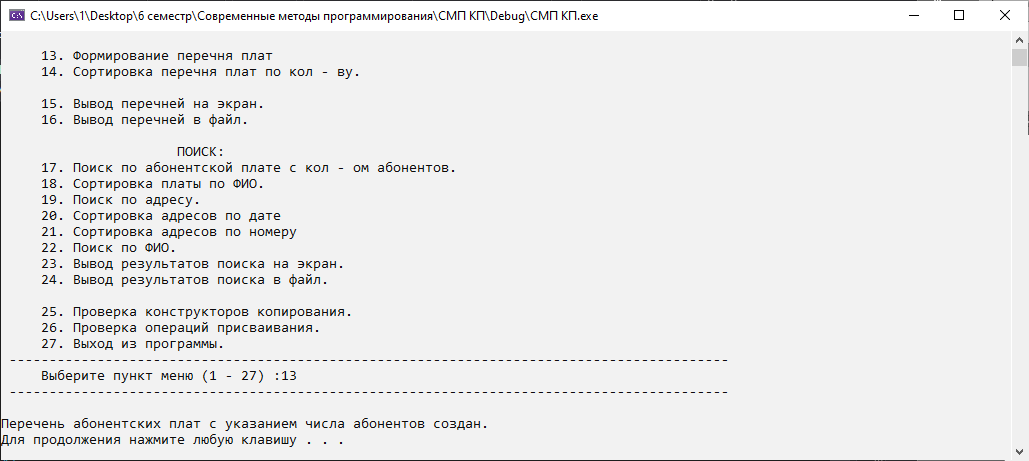




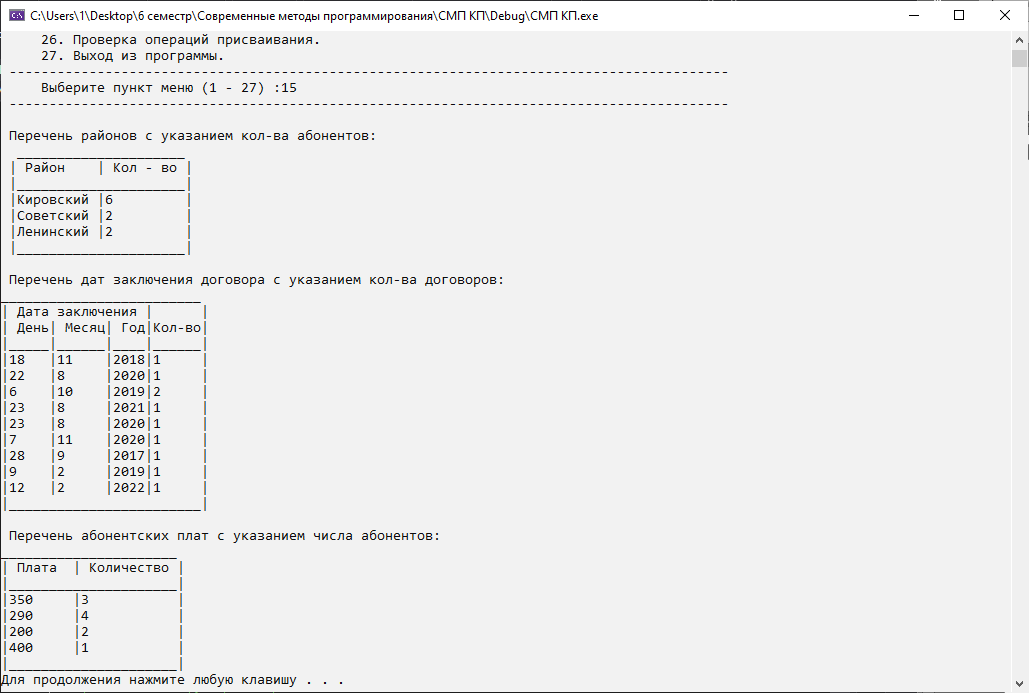
**Формирование перечней**



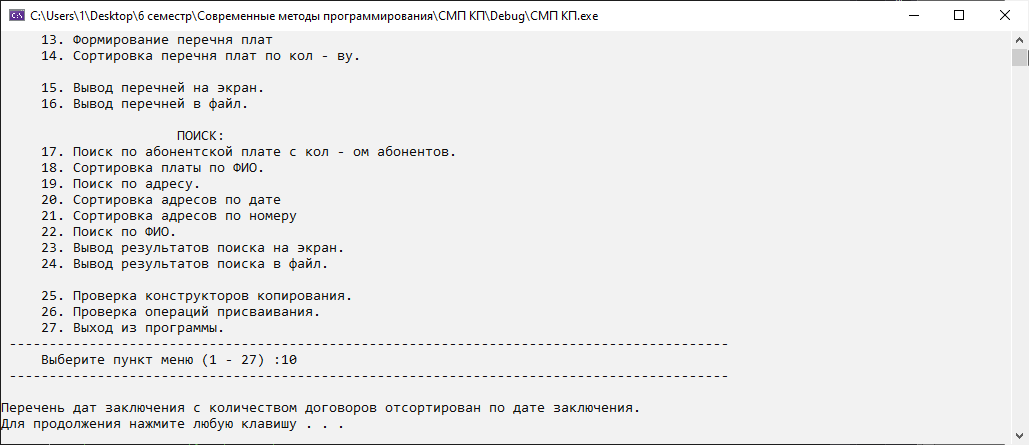


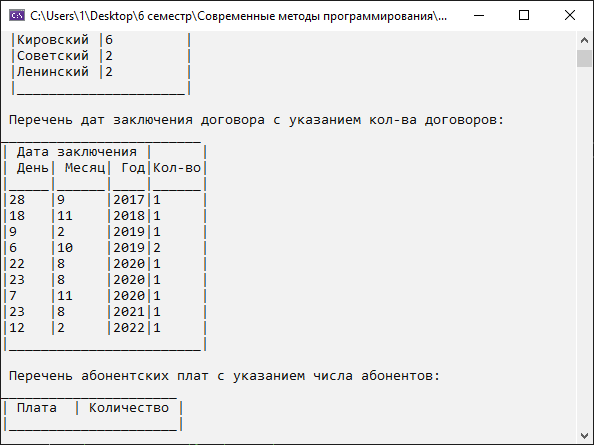


**Вывод перечней на экран**

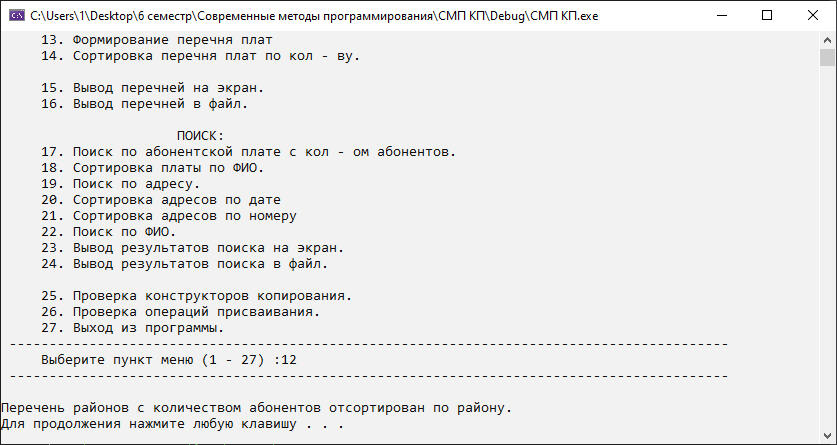


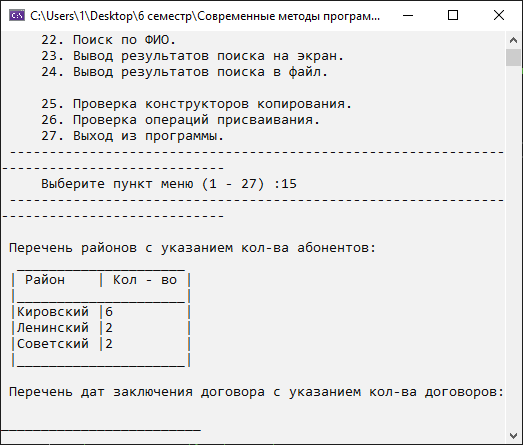
**Сортировка перечня дат заключения с количеством договоров по дате**



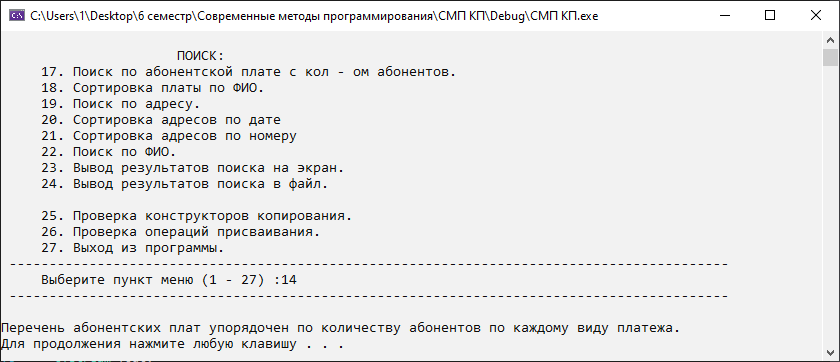


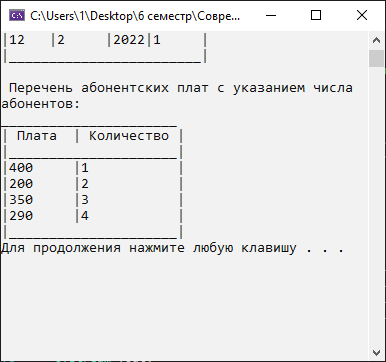
**Сортировка перечня районов с количеством абонентов по району**



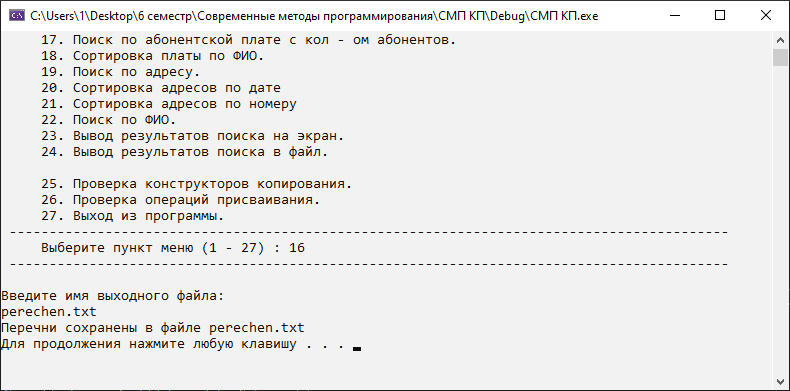


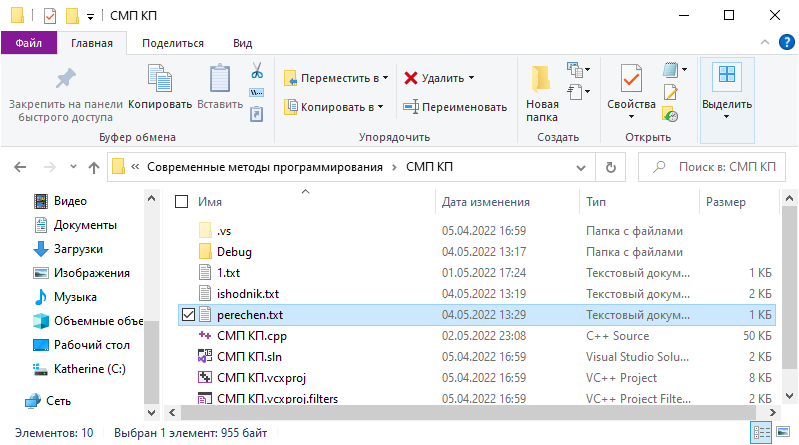
**Сортировка перечня плат по количеству**

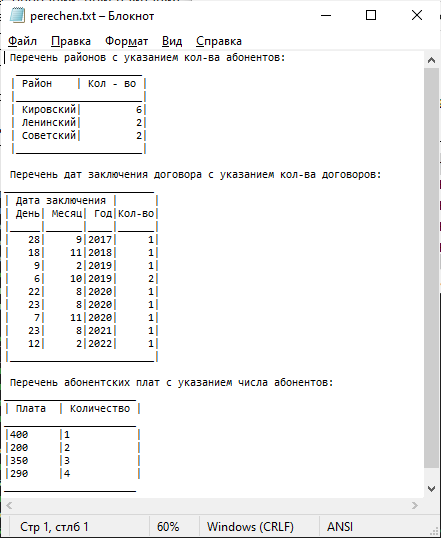




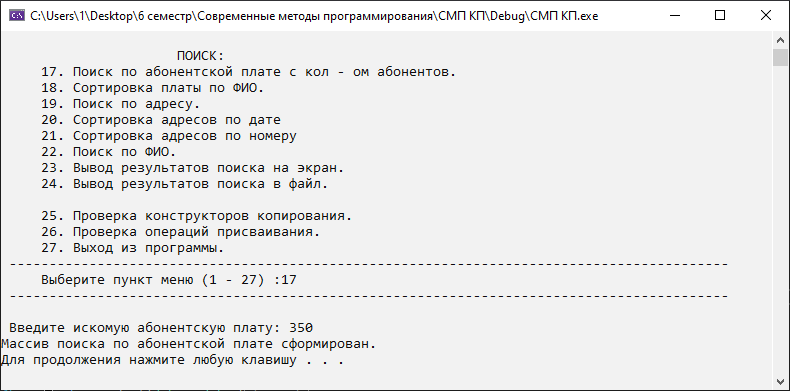
**Вывод перечней в файл**

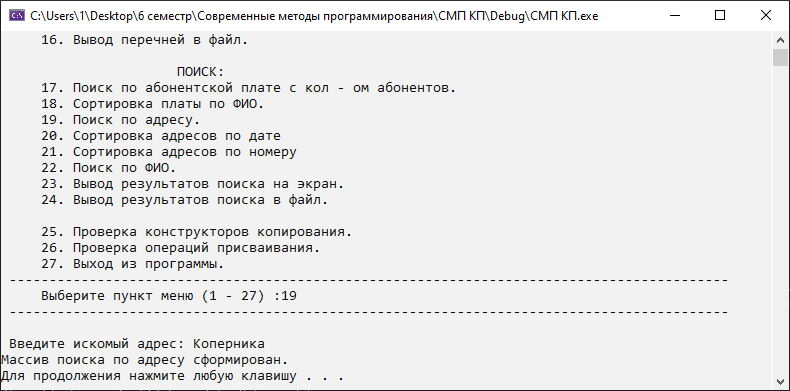


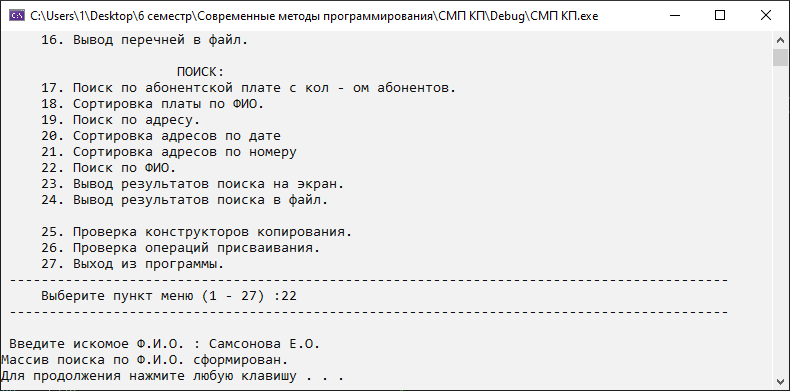




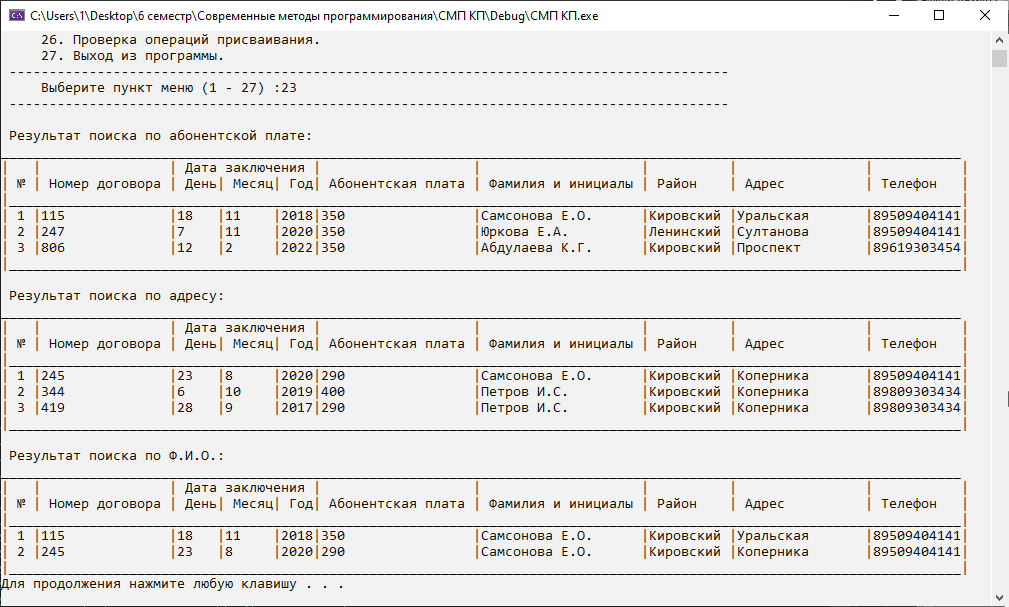
**Выполнение поиска**



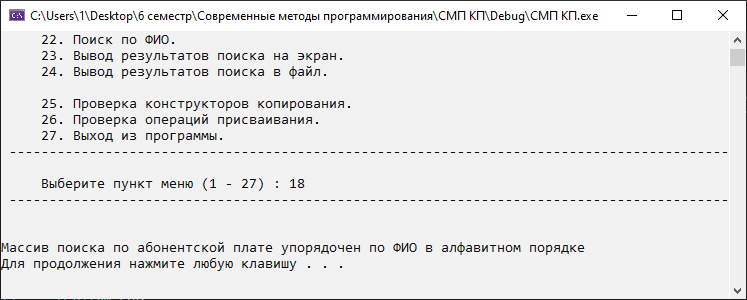


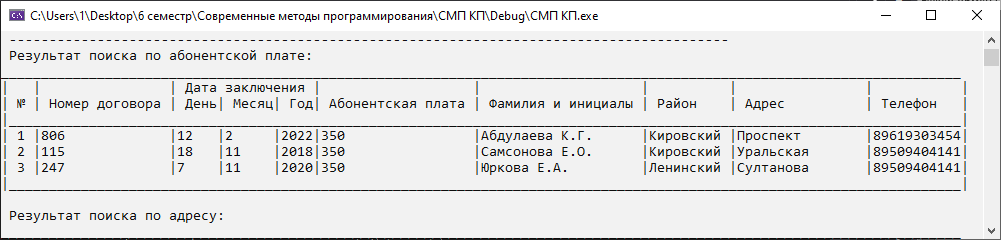


**Вывод результатов поиска на экран**

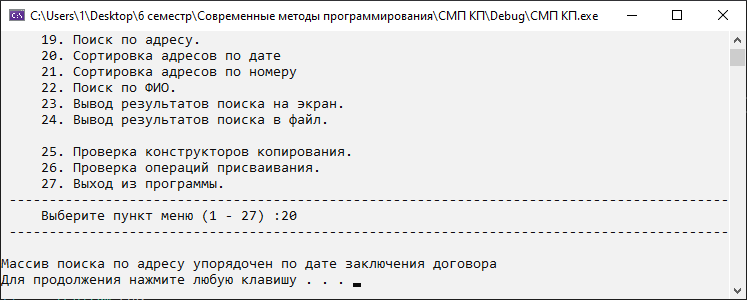


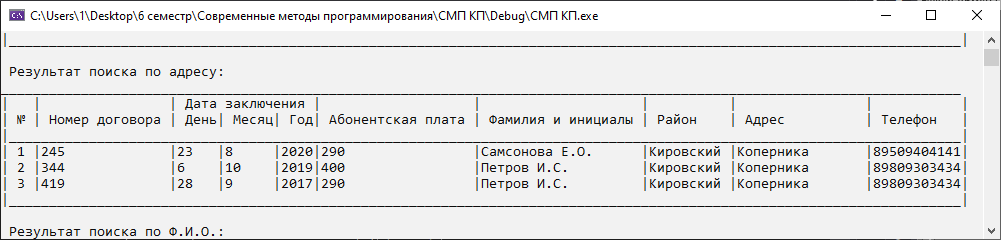
**Сортировка поиска по плате по Ф.И.О.**



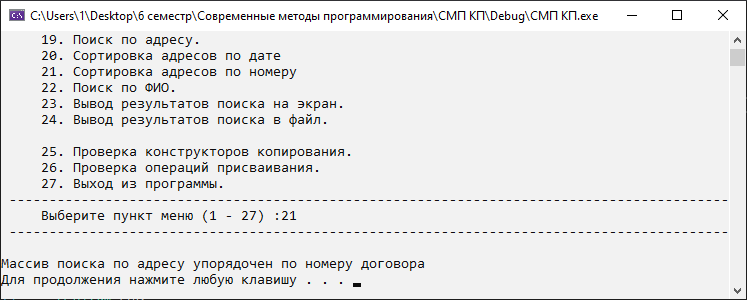


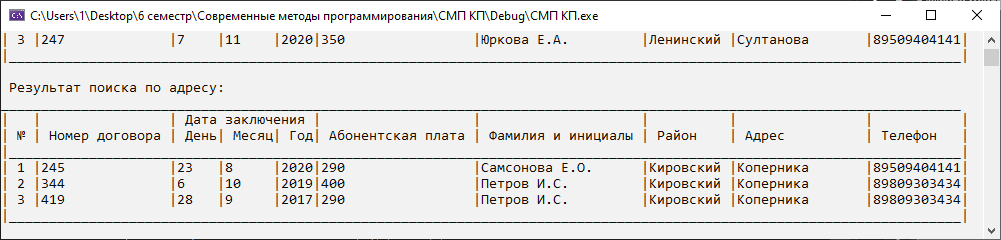
**Сортировка поиска адресов по дате**



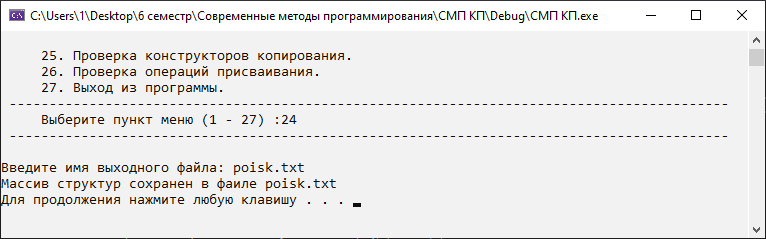


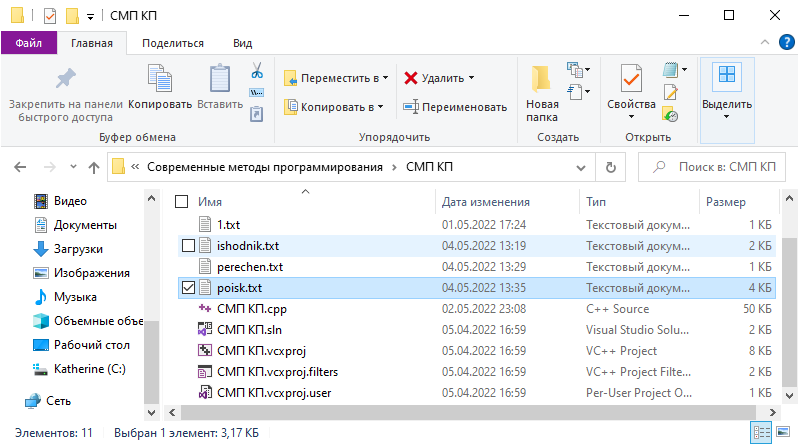
**Сортировка поиска адресов по номеру**

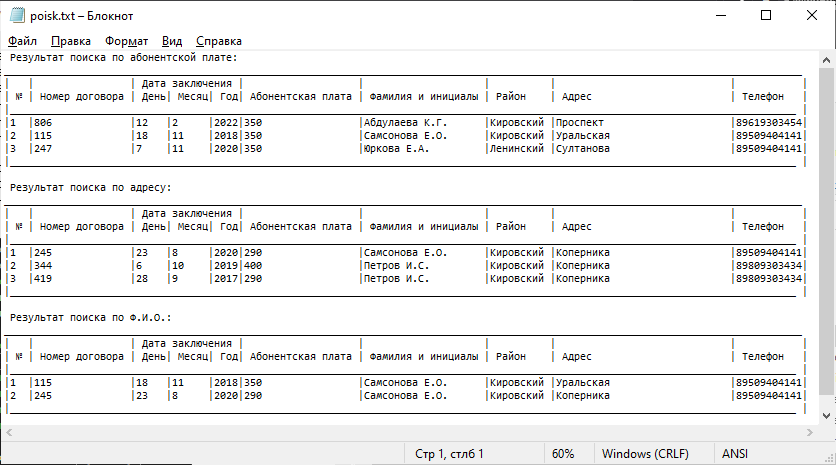




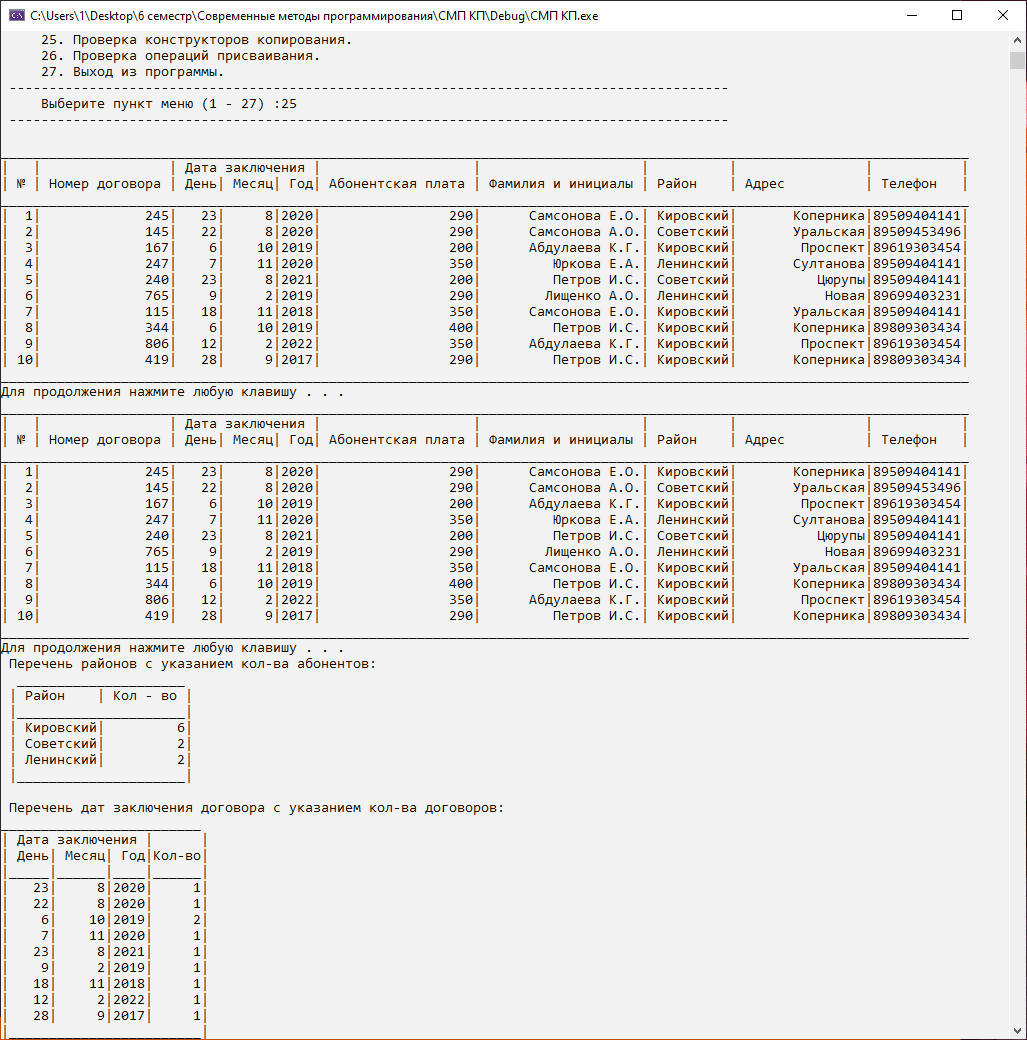
**Вывод результатов поиска в файл**

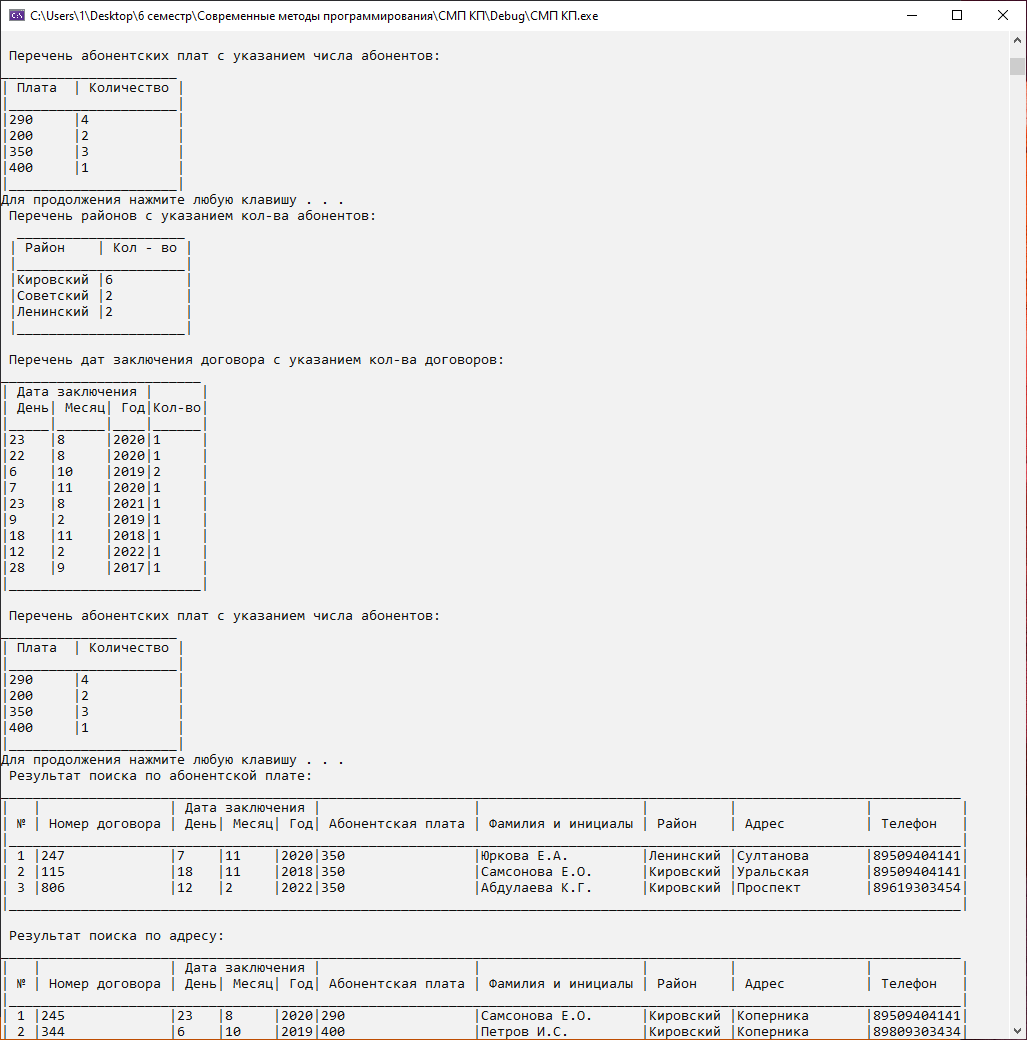


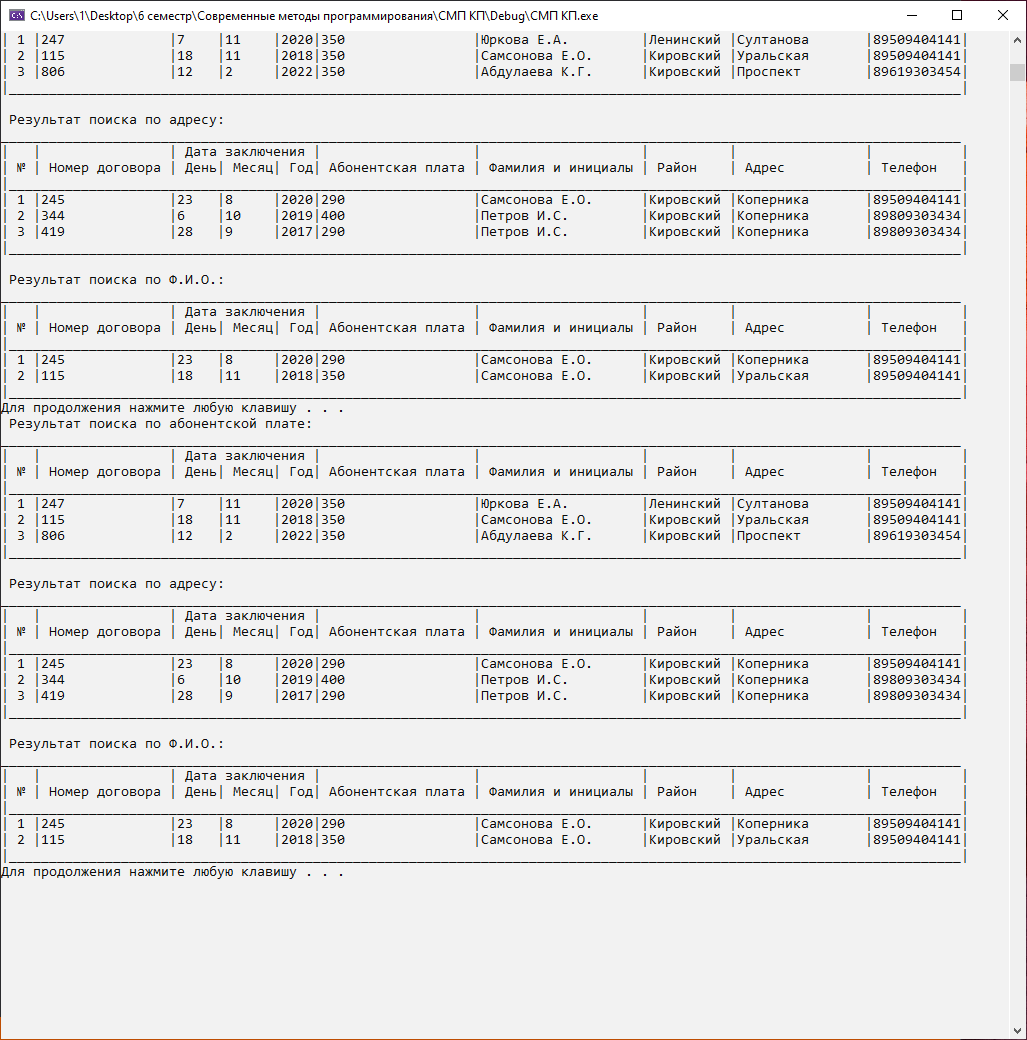




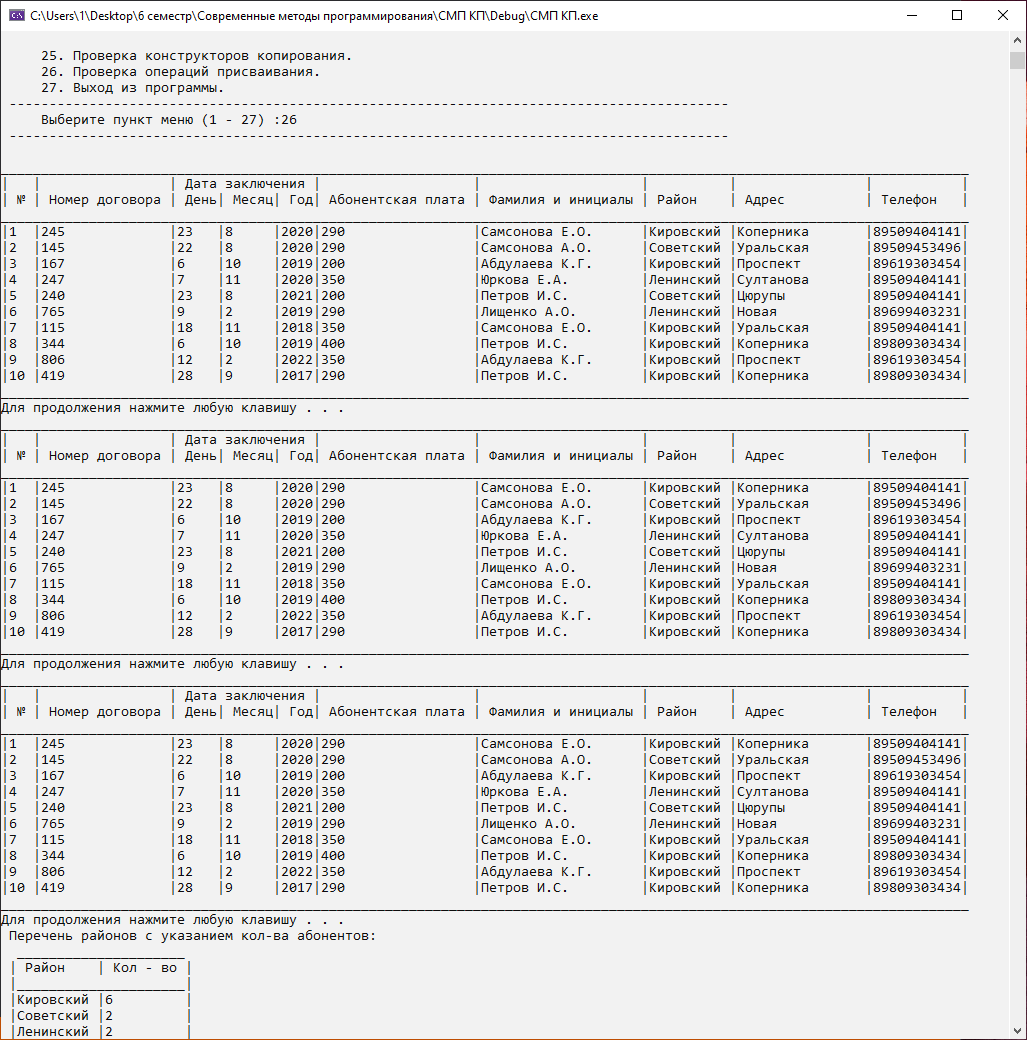
**Проверка конструкторов копирования**

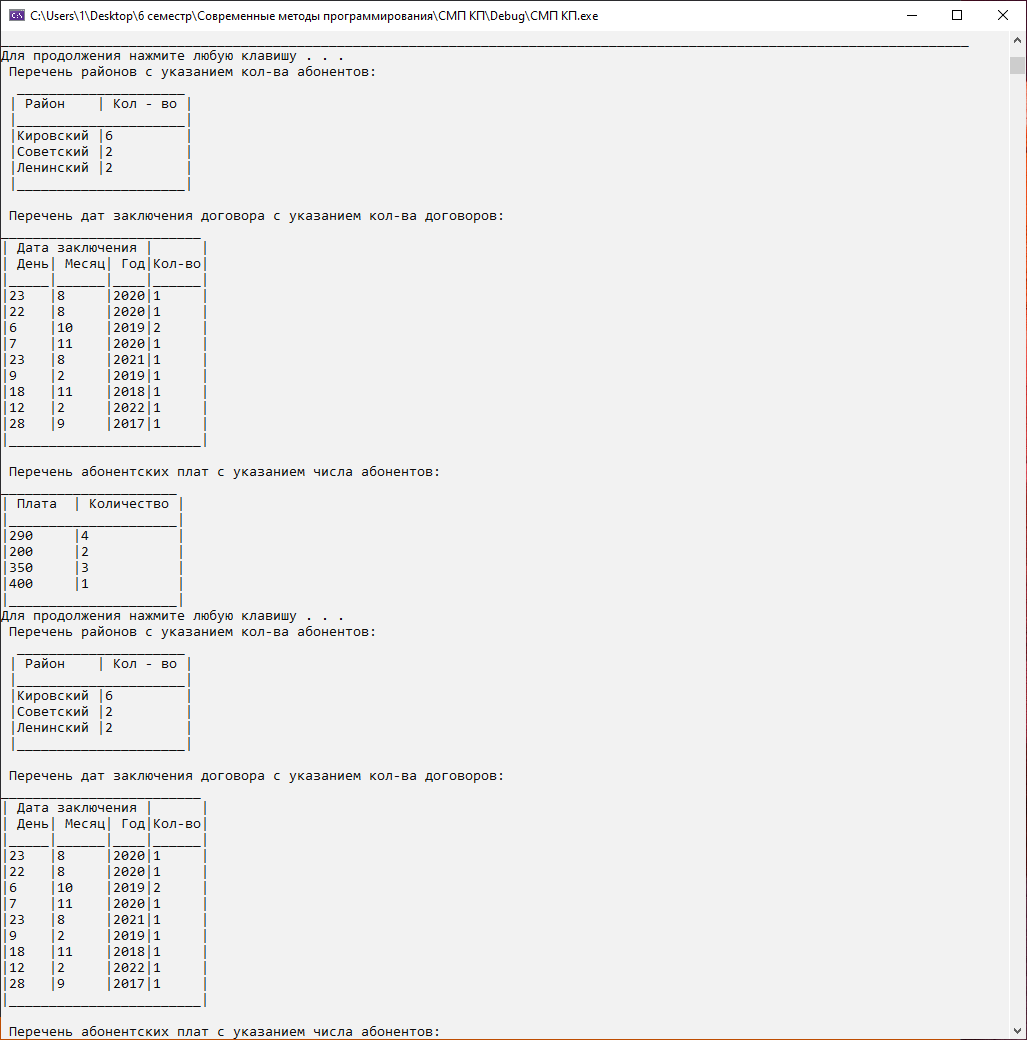


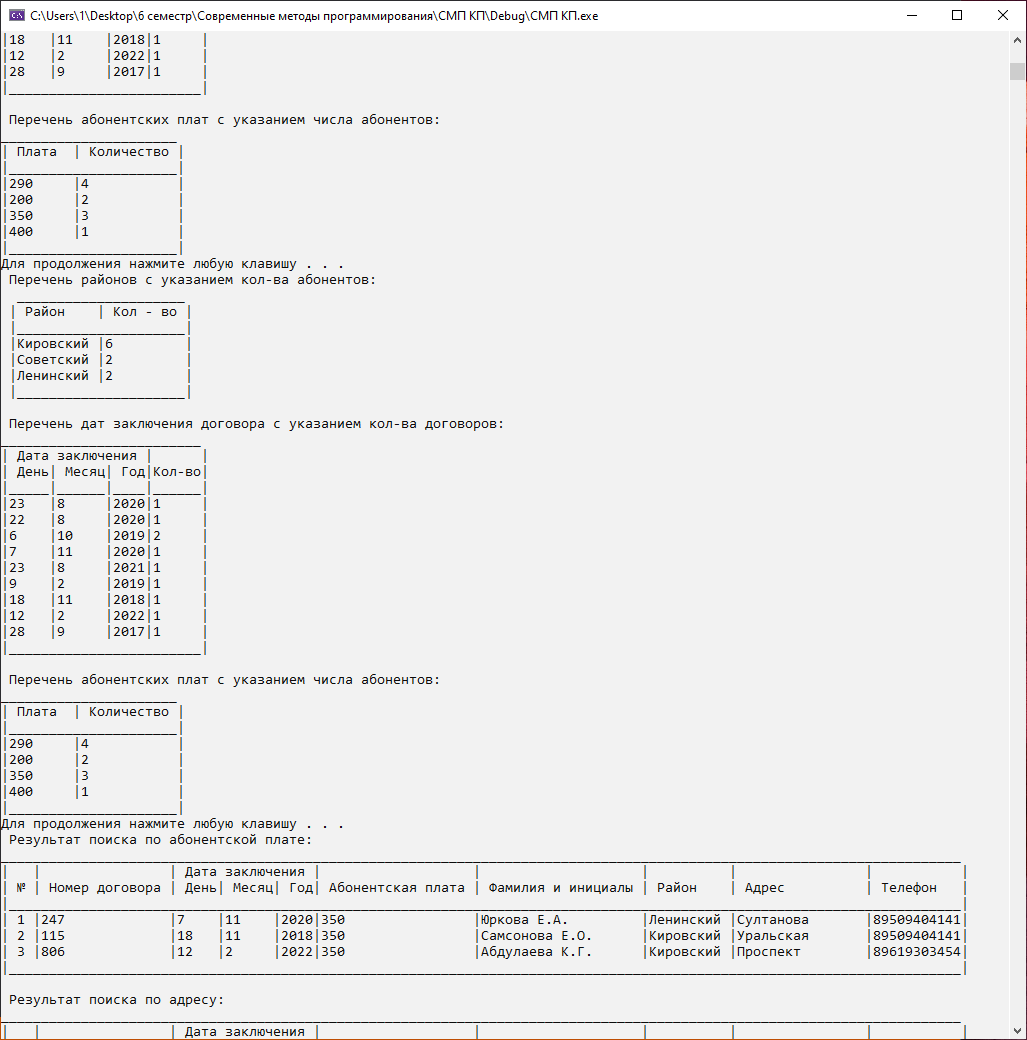


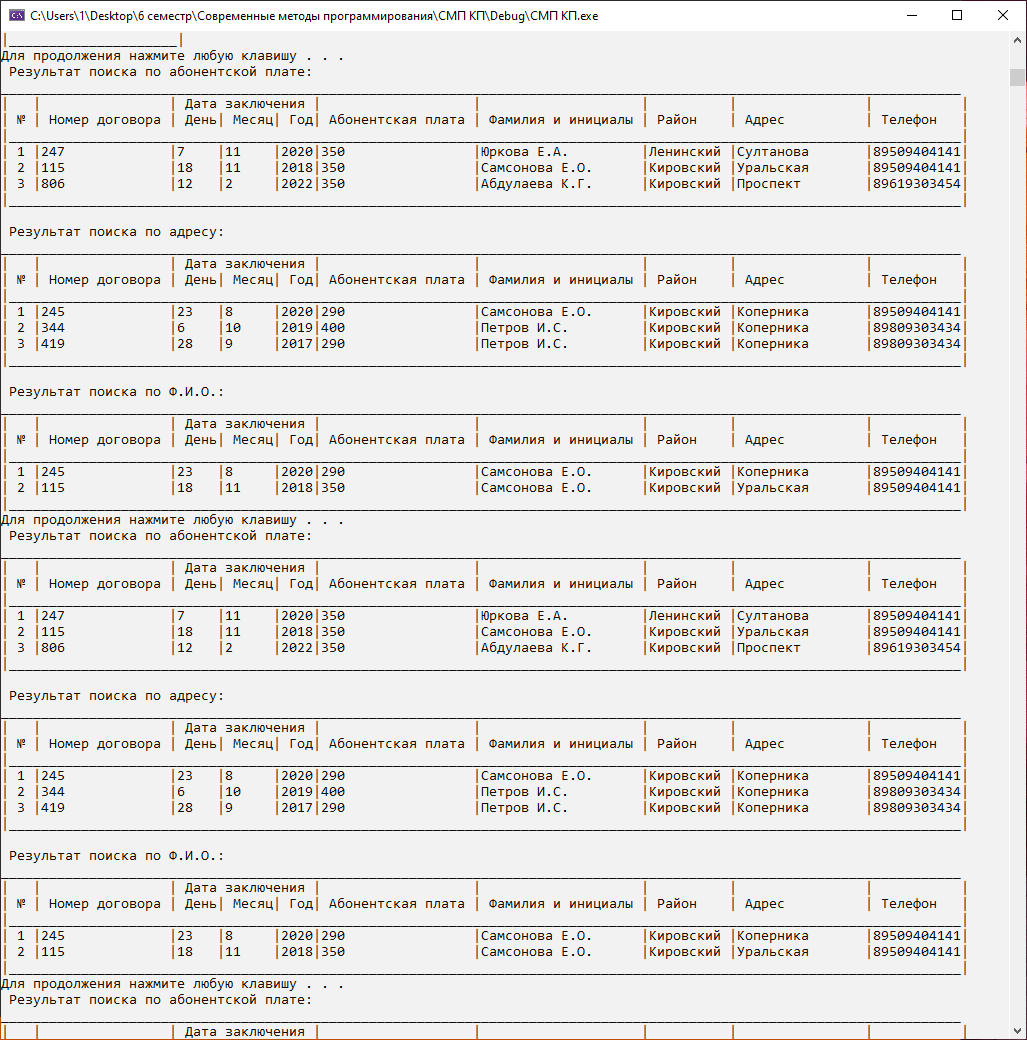


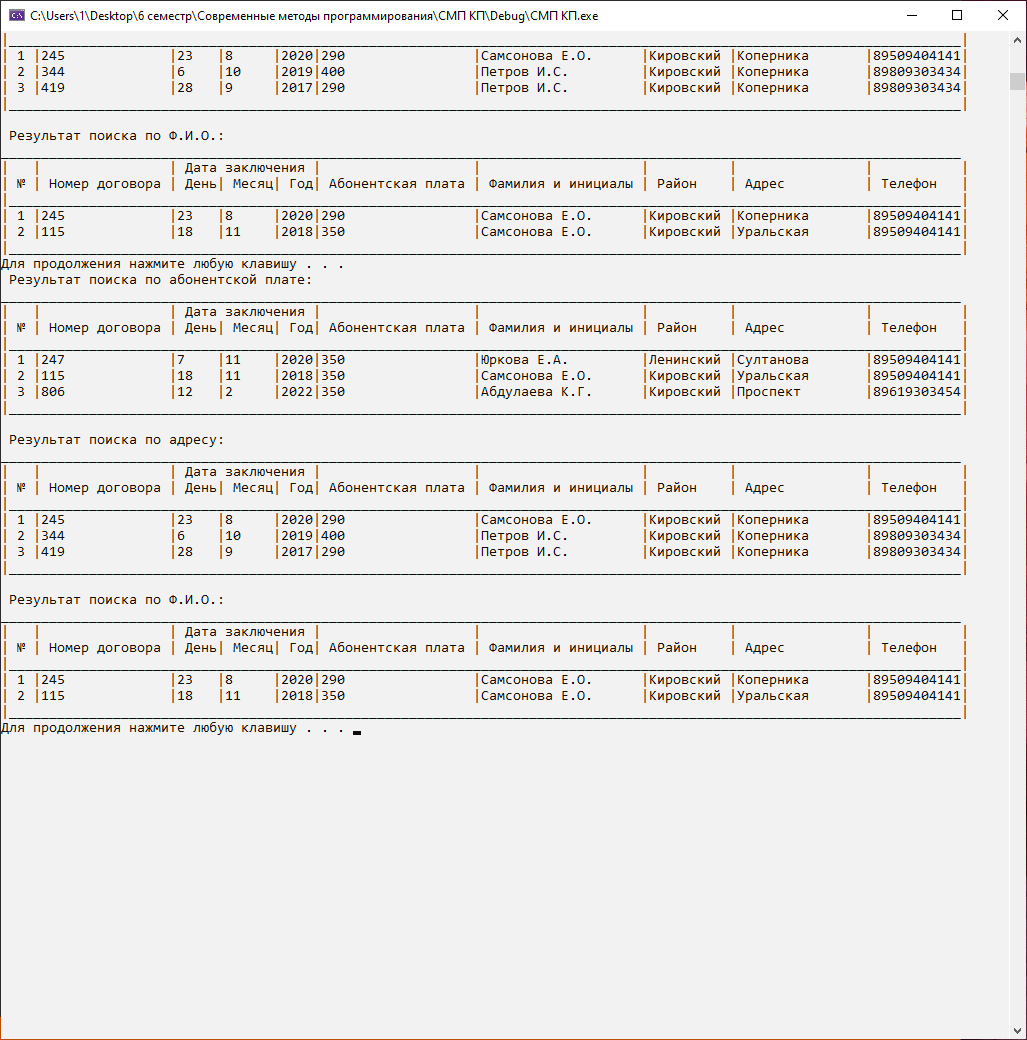
**Проверка операций присваивания**



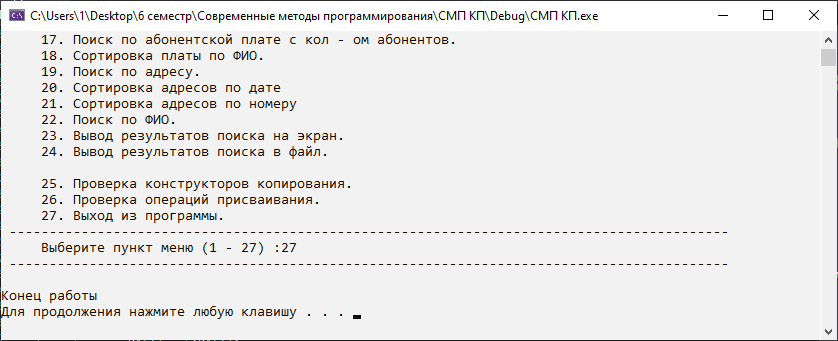








**Выход из программы**



# **Заключение**

В результате выполнения курсового проекта были разработаны классы массивов структур для решения таких задач, как сортировки, различные виды поиска, создание перечней, добавление, удаление записей. Программа содержит меню, позволяющее выполнить и проверить все методы класса на работоспособность. Классы были реализованы с помощью метода наследования, использования перегрузки операций присваивания, операций сравнения, ввода/вывода в файлы и на экран.

Программа написана на языке С++, но может быть переписана и на другом современном языке программирования.

# **Использованная литература**

1. Язык программирования C++. Лекции и упражнения

// Стивен Прата //6-е изд. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2015. - 928 с.

1. С/С++. Структурное программирование. Практикум.

//Т.А. Павловская, Ю.А. Щупак //СпБ: Питер, 2011г.

1. Методы современного программирования // конспект

//А.Ю. Хасанов //УГАТУ, 2022г.

# **Код программы**

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <string>

#include <iomanip>

#include <conio.h>

#include <Windows.h>

#include <sstream>

using namespace std;

class masB;

class masC;

// Вспомогательный структурный тип для дат

struct dat { int d, m, y; };

// Структурный тип для исходного массива и массивов результатов поиска

struct person {

string name;

string region;

string adress;

string phone;

};

struct abonent {

int number;

int plata;

person pers;

dat datzakl;

};

// Структурный тип для перечня дат

struct dates {

dat datzakl;

int nd;

};

// Структурный тип для перечня районов

struct regions {

string region;

int nr;

};

// Структурный тип для перечня плат

struct plata

{

int plata;

int count;

};

//Определение обычных операторных функций

int operator>(dat v, dat w) //Определение перегруженной операции сравнения для структур типа dat

{

if (v.y > w.y) return 1;

if ((v.y == w.y) && (v.m > w.m)) return 1;

if ((v.y == w.y) && (v.m == w.m) && (v.d > w.d)) return 1;

return 0;

}

int operator>(abonent v, abonent w) //Определение перегруженной операции сравнения для структур типа abonent

{

if (v.plata > w.plata) return 1;

if ((v.plata > w.plata) && (v.pers.name > w.pers.name)) return 1;

return 0;

}

ostream& operator<<(ostream& out, abonent& px) //Определение перегруженной операции вывода на экран структуры

{

out << "|" << setw(16) << px.number << "|" << setw(5) << px.datzakl.d << "|" << setw(6) << px.datzakl.m << "|" << setw(4) << px.datzakl.y << "|" << setw(19) << px.plata << "|" << setw(20) << px.pers.name << "|" << setw(10) << px.pers.region << "|" << setw(16) << px.pers.adress << "|" << setw(11) << px.pers.phone << "|\n";

return out;

}

ostream& operator<<(ostream& out, plata& z) //Определение перегруженной операции вывода на экран структуры типа PLATA

{

out << "|" << setw(8) << std::left << z.plata << "|" << setw(12) << std::left << z.count << "|" << endl;

return out;

}

ostream& operator<<(ostream& out, dates& z) //Определение перегруженной операции вывода на экран структуры типа PLATA

{

out << "|" << setw(5) << std::left << z.datzakl.d << setw(6) << z.datzakl.m << "|" << setw(4) << z.datzakl.y << "|" << setw(12) << std::left << z.nd << "|" << endl;

return out;

}

//Объявление глобальных файловых потоков

ifstream fin;

ofstream fоut;

//Объявление и определение базового класса masA

class masA

{ protected:

abonent \* px;

int n;

public:

masA() : px(NULL), n(0) {}

masA(masA& z);

~masA() { if (px != NULL) delete[] px; }

masA& operator=(masA& z);

void addAbonent();

void deleteAbonent();

void sortName();

void sortDat();

void sortNumber();

void outputMas();

friend ostream& operator<<(ostream& out, masA& z);

friend ifstream& operator>>(ifstream& fin, masA& z);

friend ofstream& operator<<(ofstream& fout, masA& z);

};

//Определение методов класса masА

//Определение конструктора копирования класса masА

masA::masA(masA& z) //Конструктор копирования класса masA

{

int i;

n = z.n;

if (n == 0) px = NULL;

else {

px = new abonent[n];

if (px == NULL) {

cout << "нет памяти.\n";

cout << "Конструктор копирования.\n";

system("pause"); exit(1);

}

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

px[i] = z.px[i];

}

} // конец конструктора копирования

//Определение перегруженной операции присваивания класса masА

masA& masA :: operator=(masA& z)

{

int i;

if (this == &z) return \*this;

if (px != NULL) delete[]px;

n = z.n;

if (z.px == NULL) px = NULL;

else {

px = new abonent[n];

if (px == NULL) {

cout << "нет памяти.\n";

cout << "операция присваивания.\n";

system("pause"); return z;

}

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

px[i] = z.px[i];

}

return \*this;

}

void masA::outputMas()

{

int i;

// Вывод заголовка таблицы (массива структур)

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "| | | Дата заключения | | | | | |\n";

cout << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

cout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | " << "\n";

cout << left;

// Вывод строк таблицы

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

cout << "|" << setw(3) << i + 1 << "|" << setw(16) << px[i].number << "|" << setw(5) << px[i].datzakl.d << "|" << setw(6) << px[i].datzakl.m << "|" << setw(4) << px[i].datzakl.y << "|" << setw(19) << px[i].plata << "|" << setw(20) << px[i].pers.name << "|" << setw(10) << px[i].pers.region << "|" << setw(29) << px[i].pers.adress << "|" << setw(11) << px[i].pers.phone << "| \n";

cout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | \n";

system("pause");

}

//Определение метода addAbonent() класса masА

void masA::addAbonent()

{

int i;

abonent t, \* p;

string iniz;

p = new abonent[n + 1]; //выделение памяти для нового массива

if (p == NULL) {

cout << "нет памяти.\n";

cout << "добавить не удается.\n";

system("pause"); return;

}

cout << " \n";

cout << " Введите следующую информацию : \n";

cout << "Номер договора:"; cin >> t.number;

cout << "Дата заключения.\n";

cout << "День:"; cin >> t.datzakl.d;

cout << "Месяц:"; cin >> t.datzakl.m;

cout << "Год:"; cin >> t.datzakl.y;

cout << "Абонентская плата:"; cin >> t.plata;

cout << "Ф.И.О.:"; cin >> t.pers.name >> iniz;

t.pers.name = t.pers.name + " " + iniz;

cout << "Регион:"; cin >> t.pers.region;

cout << "Адрес:"; cin >> t.pers.adress;

cout << "Телефон:"; cin >> t.pers.phone;

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

p[i] = px[i]; //копирования старого массива в новый

p[n] = t; //добавление новой записи в массив структур

n++; //увеличение счетчика количества записей

if (px != NULL) delete[] px; //удаление старого массива

px = p; //адрес нового массива заносится в px

cout << "Запись добавлена.\n";

system("pause");

}

//Определение метода deleteStudent() класса masА

void masA::deleteAbonent()

{

int j, i;

abonent\* p = NULL;

string iniz;

char ch;

outputMas(); //вызов на экран массива структур

cout << " \n";

cout << " Введите номер удаляемой стороки:";

cin >> j;

//Проверки ввода номера строки

if (cin.fail())

{

string s;

cin.clear();

cin >> s;

cout << "Это не номер строки.\n";

system("pause");

return;

}

if (j < 1 || j > n) {

cout << "Ошибка: нет такой строки.\n";

system("pause"); return;

}

j--; // переход от номера удаленной строки к ее индексу

cout << j + 1 << "-я строка:\n";

cout << px[j].number << " " << px[j].datzakl.d << "." << px[j].datzakl.m << "." << px[j].datzakl.y << " " << px[j].plata << " " << px[j].pers.name << " " << iniz << " " << px[j].pers.region << " " << px[j].pers.adress << " " << px[j].pers.phone << endl;

cout << "Удалить?(y/n):"; cin >> ch;

if (ch == 'n') return; // отказ удаления строки

if (ch != 'y') { cout << "Ошибка ответа на вопрос. \n"; system("pause"); return; }

// Выполнение удаления строки

if (n == 1) { delete[] px; px = NULL; n = 0; }

else

{

p = new abonent[n - 1];

if (p == NULL)

{

cout << "Нет памяти.\n ";

cout << "Удалить не удается.\n";

system("pause"); return;

}

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

p[i] = px[i];

for (i = j + 1; i < n; i++)

p[i - 1] = px[i];

delete[] px; // удаление старого массива

px = p; // адрес нового массива заносится в поле px

n--;// уменьшение счетчика количества записей

}

cout << "Запись удалена.\n";

system("pause");

}

//Сортировка массива структур по ФИО (сортировка по строковому полю в алф-ом порядке)

void masA::sortName()

{

int i, fl, nn;

abonent t;

nn = n;

do {

fl = 0; nn--;

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

if (px[i].pers.name > px[i + 1].pers.name)

{

fl = 1; t = px[i];

px[i] = px[i + 1];

px[i + 1] = t;

}

} while (fl == 1);

cout << "Массив структур упорядочен по ФИО в алфавитном порядке\n";

system("pause");

}

//Сортировка массива структур по дате заключения(сортировка по нескольким полям)

void masA::sortDat() // сортировка по дате заключения

{

int fl, i, nn;

abonent t;

nn = n;

do {

fl = 0; nn--;

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

if (px[i].datzakl > px[i + 1].datzakl) // это перегруженная операция “>”

{

fl = 1; t = px[i];

px[i] = px[i + 1];

px[i + 1] = t;

}

} while (fl == 1);

cout << "Массив структур упорядочен по дате заключения договора\n";

system("pause");

}

//Сортировка массива структур по числовому полю

void masA::sortNumber() // по номеру договора

{

int i, fl, nn;

abonent t;

nn = n; // вводим копию размера массива

do {

fl = 0; nn--;

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

if (px[i].number > px[i + 1].number)

{

fl = 1; t = px[i];

px[i] = px[i + 1];

px[i + 1] = t;

}

} while (fl == 1);

cout << "Массив структур упорядочен по номеру договора\n";

system("pause");

}

//Вывод массива структур на экран

//Определение перегруженной операции вывода на экран объекта класса masA

ostream& operator<<(ostream& out, masA& z)

{

int i;

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << "| | | Дата заключения | | | | | |\n";

out << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

for (i = 0; i < z.n; i++)

out << "|" << setw(3) << i + 1 << z.px[i];

// out<<z.px[i] это перегруженная операция вывода для структуры

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

system("pause");

return out;

}

//Определение дружественной функции ввода из файла operator>>(ifstream& fin, masA& z) класса masА

ifstream& operator>>(ifstream& fin, masA& z)

{

string file;

string iniz;

abonent t;

cout << "Введите имя входного файла: ";

cin >> file;

fin.open(file.c\_str());

// Защита от неправильного ввода имени файла

if (fin.fail()) {

cout << file << " Файл не открывается\n";

system("pause"); return fin;

}

z.n = 0;

if (z.px != NULL) { delete[]z.px; z.px = NULL; }//очистка массива

while (1)

{

fin >> t.number >> t.datzakl.d >> t.datzakl.m >> t.datzakl.y >> t.plata >> t.pers.name >> iniz >>t.pers.region >> t.pers.adress >> t.pers.phone;

if (fin.fail()) break;z.n++;

}

fin.close();

z.px = new abonent[z.n];

if (z.px == NULL) {

cout << "Нет памяти.\n"; fin.close();

cout << "Ввести фаил не удается.\n";

system("pause"); z.n = 0; return fin;

}

fin.close();

fin.open(file.c\_str());

if (fin.fail()) {

cout << file << " Файл повторно не открывается\n";

system("pause"); z.n = 0; return fin;

}

for (int i = 0; i < z.n; i++)

{

fin >> z.px[i].number >> z.px[i].datzakl.d >> z.px[i].datzakl.m >> z.px[i].datzakl.y >> z.px[i].plata >> z.px[i].pers.name >> iniz >> z.px[i].pers.region >> z.px[i].pers.adress >> z.px[i].pers.phone;

z.px[i].pers.name = z.px[i].pers.name + " " + iniz;

}

fin.close();

cout << "Файл успешно введен " << endl;

system("pause");

return fin;

} // Конец определения перегруженной операции ввода из файла

//Определение дружественной функции вывода в файл operator<<(ofstream& fout, masA& z) класса masА

ofstream& operator<<(ofstream& fout, masA& z)

{

string file;

int i;

cout << "Введите имя выходного файла: "; cin >> file;

fout.open(file.c\_str());

if (fout.fail()) {

cout << file << "не создается.\n";

system("pause"); return fout;

}

// Вывод заголовка

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << "| | | Дата заключения | | | | | |\n";

fout << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

// Вывод остальных строк таблицы

for (i = 0; i < z.n; i++)

fout << "|" << setw(3) << i + 1 << "|" << setw(16) << z.px[i].number << "|" << setw(5) << z.px[i].datzakl.d << "|" << setw(6) << z.px[i].datzakl.m << "|" << setw(4) << z.px[i].datzakl.y << "|" << setw(19) << z.px[i].plata << "|" << setw(20) << z.px[i].pers.name << "|" << setw(10) << z.px[i].pers.region << "|" << setw(16) << z.px[i].pers.adress << "|" << setw(12) << z.px[i].pers.phone << "| \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout.close();

cout << "Массив структур сохранен в фаил " << file <<"\n";

system("pause");

return fout;

} // Конец определения перегруженной операции вывода в файл

//Объявление и определение производного класса masВ

class masB : public masA

{

protected:

regions\* py;

int k;

dates\* py2;

int k2;

plata\* mas;

int k3;

public:

masB() : py(NULL), k(0), py2(NULL), k2(0), mas(NULL), k3(0) {}

masB(masB& z);

~masB() {

if (py != NULL) delete[] py;

if (py2 != NULL) delete[] py2;

if (mas != NULL) delete[] mas;

}

masB& operator=(masB& z);

void sortRegions();

void sortDates();

void sortCount();

void makePerechRegions();

void makePerechDates();

void makePerechPlata();

friend ostream& operator<<(ostream& out, masB& z);

friend ofstream& operator<<(ofstream& fout, masB& z);

};

//Определение методов класса masB

//Определение конструктора копирования класса masB

masB::masB(masB& z) : masA(z)

{

int i;

k = z.k;

if (z.py == NULL) py = NULL;

else {

py = new regions[k];

if (py == NULL) {

cout << "Нет памяти.\n";

cout << "Конструктор копирования перечня районов.\n";

system("pause"); exit(1);

}

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

py[i] = z.py[i];

}

k2 = z.k2;

if (z.py2 == NULL) py2 = NULL;

else {

py2 = new dates[k2];

if (py2 == NULL) {

cout << "Нет памяти.\n";

cout << "Конструктор копирования перечня дат заключения договора.\n";

system("pause"); exit(1);

}

for (i = 0; i < k2; i++)

py2[i] = z.py2[i];

}

k3 = z.k3;

if (z.mas == NULL) mas = NULL;

else {

mas = new plata[k3];

if (mas == NULL) {

cout << "Нет памяти.\n";

cout << "Конструктор копирования перечня абонентских плат.\n";

system("pause"); exit(1);

}

for (i = 0; i < k3; i++)

mas[i] = z.mas[i];

}

} // Конец определения конструктора копирования класса masB

//Определение перегруженной операции присваивания класса masB

masB& masB :: operator=(masB& z)

{

int i;

if (this == &z) return \*this;

// Вызов перегруженной операции присваивания класса masА для

masA :: operator= (z); // присваивания родительской части объекта z

if (py != NULL) delete[] py;

k = z.k;

if (z.py == NULL) py = NULL;

else {

py = new regions[k];

if (py == NULL) {

cout << "Нет памяти при присваивании. \n";

system("pause"); return z;

}

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

py[i] = z.py[i];

}

if (py2 != NULL) delete[] py2;

k2 = z.k2;

if (z.py2 == NULL) py2 = NULL;

else {

py2 = new dates[k2];

if (py2 == NULL) {

cout << "Нет памяти при присваивании. \n";

system("pause"); return z;

}

for (i = 0; i < k2; i++)

py2[i] = z.py2[i];

}

if (mas != NULL) delete[] mas;

k3 = z.k3;

if (z.mas == NULL) mas = NULL;

else {

mas = new plata[k3];

if (mas == NULL) {

cout << "Нет памяти при присваивании. \n";

system("pause"); return z;

}

for (i = 0; i < k3; i++)

mas[i] = z.mas[i];

}

return \*this;

}// Конец определения перезагрузки операции присваивания класса masB

//Определение метода makePerechRegons() класса masB

void masB::makePerechRegions() // Создание перечня районов

{

int i, j, fl;

regions\* y;

y = new regions[n];

if (y == NULL) {

cout << " Нет памяти.\n";

system("pause"); return;

}

k = 0;

if (py != NULL) delete[] py;

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

{

fl = 0;

for (j = 0; j < k; j++)

if (px[i].pers.region == y[j].region)

{

fl = 1; y[j].nr++;

}

if (fl == 0) {

y[k].region = px[i].pers.region;

y[k].nr = 1;

k++;

}

}

py = new regions[k];

if (py == NULL) {

cout << " Нет памяти для перечня.\n";

system("pause"); k = 0; delete[] y; return;

}

for (j = 0; j < k; j++)

py[j] = y[j];

delete[] y;

cout << "Перечень районов с указанием количества абонентов создан.\n";

system("pause");

} // Конец определения метода формирования перечня групп

//Определение метода makePerechDates() класса masB

void masB::makePerechDates() // Создание перечня годов рождения

{

int i, j, fl;

dates\* y2;

y2 = new dates[n];

if (y2 == NULL) {

cout << " Нет памяти.\n";

system("pause"); return;

}

k2 = 0;

if (py2 != NULL) delete[] py2;

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

{

fl = 0;

for (j = 0; j < k2; j++)

if (px[i].datzakl.d == y2[j].datzakl.d && px[i].datzakl.m == y2[j].datzakl.m && px[i].datzakl.y == y2[j].datzakl.y)

{

fl = 1; y2[j].nd++;

}

if (fl == 0) {

y2[k2].datzakl = px[i].datzakl;

y2[k2].nd = 1;

k2++;

}

}

py2 = new dates[k2];

if (py2 == NULL) {

cout << " Нет памяти для перечня.\n";

system("pause"); k2 = 0; delete[] y2; return;

}

for (j = 0; j < k2; j++)

py2[j] = y2[j];

delete[] y2;

cout << "Перечень дат заключения с указанием количества договоров создан.\n";

system("pause");

}

//Определение метода makePerechPlata() класса masB

void masB::makePerechPlata()

{

int i, j, fl;

plata\* y3;

y3 = new plata[n];

if (y3 == NULL) {

cout << "Нет памяти.\n";

system("pause"); return;

}

k3 = 0;

if (mas != NULL) delete[] mas;

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

{

fl = 0;

for (j = 0; j < k3; j++)

if (px[i].plata == y3[j].plata)

{

fl = 1; y3[j].count++;

}

if (fl == 0) {

y3[k3].plata = px[i].plata;

y3[k3].count = 1;

k3++;

}

}

mas = new plata[k3];

if (mas == NULL) {

cout << "Нет памяти для перечня.\n";

system("pause"); k3 = 0; delete[] y3; return;

}

for (j = 0; j < k3; j++)

mas[j] = y3[j];

delete[] y3;

cout << "Перечень абонентских плат с указанием числа абонентов создан.\n";

system("pause");

}

//Определение метода sortRegions() класса masB

void masB::sortRegions()

{

int fl, i, kk;

regions t;

kk = k;

do {

fl = 0; kk--;

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

if (py[i].region > py[i + 1].region)

{

fl = 1; t = py[i];

py[i] = py[i + 1];

py[i + 1] = t;

}

} while (fl == 1);

cout << "Перечень районов с количеством абонентов отсортирован по району.\n";

system("pause");

}//Конец определения метода сортировки перечня групп по названию групп

//Определение метода sortDates() класса masB

void masB::sortDates()

{

int fl, i, kk;

dates t;

kk = k2;

do {

fl = 0; kk--;

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

if (py2[i].datzakl >py2[i + 1].datzakl)

{

fl = 1; t = py2[i];

py2[i] = py2[i + 1];

py2[i + 1] = t;

}

} while (fl == 1);

cout << "Перечень дат заключения с количеством договоров отсортирован по дате заключения.\n";

system("pause");

} // Конец определения метода сортировки перечня годов рождения по году

//Определение метода sortPlata() класса masB

void masB::sortCount()

{

int i, fl, kk;

plata t;

kk = k3;

do {

fl = 0; kk--;

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

if (mas[i].count > mas[i + 1].count)

{

fl = 1; t = mas[i];

mas[i] = mas[i + 1];

mas[i + 1] = t;

}

} while (fl == 1);

cout << "Перечень абонентских плат упорядочен по количеству абонентов по каждому виду платежа. \n";

system("pause");

}

//Определение дружественной функции вывода в файл operator<<(ofstream& fout, masВ& z) класса masВ

ofstream& operator<<(ofstream& fout, masB& z)

{

string file;

int i;

cout << "Введите имя выходного файла: \n"; cin >> file;

fout.open(file.c\_str());

if (fout.fail()) {

cout << file << "Ошибка. Файл не создан.\n";

system("pause"); return fout;

}

// Вывод заголовка перечня районов

fout << " Перечень районов с указанием кол-ва абонентов: \n";

fout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << " | Район | Кол - во | \n";

fout << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| \n";

//Вывод строк таблицы

for (i = 0; i < z.k; i++)

fout << " |" << setw(10) << z.py[i].region << "|"<< setw(10) << z.py[i].nr << "| \n";

fout << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| \n";

// Вывод заголовка перечня дат

fout << endl << " Перечень дат заключения договора с указанием кол-ва договоров: \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << "| Дата заключения | |\n";

fout << "| День| Месяц| Год|Кол-во|\n";

fout << "|\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_|\_\_\_\_\_\_|\n";

//Вывод строк таблицы

for (i = 0; i < z.k2; i++)

fout << "|" << setw(5) << z.py2[i].datzakl.d << "|" << setw(6) << z.py2[i].datzakl.m << "|" << setw(4) << z.py2[i].datzakl.y << "|"<< setw(6) << z.py2[i].nd << "|\n";

fout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| " << "\n";

// Вывод заголовка перечня плат

fout << endl << " Перечень абонентских плат с указанием числа абонентов: \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << "| Плата | Количество | \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

for (i = 0; i < z.k3; i++)

fout << z.mas[i];

// fout<<z.mas[i] это перегруженная операция вывода на экран для структуры типа PLATA

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout.close();

cout << "Перечни сохранены в файле " << file << "\n";

system("pause");

return fout;

}//Конец определения дружественной функции вывода в файл класса masВ

//Определение дружественной функции вывода на экран operator<<(ostream& out, masВ& z) класса masВ

ostream& operator<<(ostream& out, masB& z)

{

int i;

// Вывод заголовка перечня районов

out << " Перечень районов с указанием кол-ва абонентов: \n";

out << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << " | Район | Кол - во | \n";

out << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| \n";

for (i = 0; i < z.k; i++)

out << " |" << setw(10) << z.py[i].region << "|" << setw(10) << z.py[i].nr << "|\n";

out << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| \n";

// Вывод заголовка перечня дат

out << endl << " Перечень дат заключения договора с указанием кол-ва договоров: \n";

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << "| Дата заключения | |\n";

out << "| День| Месяц| Год|Кол-во|\n";

out << "|\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_|\_\_\_\_\_\_|\n";

for (i = 0; i < z.k2; i++)

out << "|" << setw(5) << z.py2[i].datzakl.d << "|" << setw(6) << z.py2[i].datzakl.m << "|" << setw(4) << z.py2[i].datzakl.y << "|" << setw(6) << z.py2[i].nd << "|\n";

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| " << "\n";

// Вывод заголовка перечня плат

out << endl << " Перечень абонентских плат с указанием числа абонентов: \n";

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << "| Плата | Количество | \n";

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

for (i = 0; i < z.k3; i++)

out << z.mas[i];

// fout<<z.mas[i] это перегруженная операция вывода на экран для структуры типа PLATA

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

system("pause");

return out;

}

//Объявление и определение производного класса masС

class masC : public masB

{

protected:

abonent\* pz;

int l;

abonent\* pz2;

int l2;

abonent\* pz3;

int l3;

public:

masC() : pz(NULL), l(0), pz2(NULL), l2(0), pz3(NULL), l3(0) {}

masC(masC& z);

~masC() {

if (pz != NULL) delete[] pz;

if (pz2 != NULL) delete[] pz2;

if (pz3 != NULL) delete[] pz3;

}

masC& operator=(masC& z);

void sortPlataFIO();

void sortAddressDate();

void sortAddressNum();

void findPlata();

void findAddress();

void findbyFIO();

friend ostream& operator<<(ostream& out, masC& z);

friend ofstream& operator<<(ofstream& fout, masC& z);

};

//Определение конструктора копирования класса masС

masC::masC(masC& z) : masB(z)

{

int i;

l = z.l;

if (z.pz == NULL) pz = NULL;

else {

pz = new abonent[l];

if (pz == NULL) {

cout << "нет памяти.\n";

cout << "Конструктор копирования для поиска по абонентской плате.\n";

system("pause"); exit(1);

}

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

pz[i] = z.pz[i];

}

l2 = z.l2;

if (z.pz2 == NULL) pz2 = NULL;

else {

pz2 = new abonent[l2];

if (pz2 == NULL) {

cout << "нет памяти.\n";

cout << "Конструктор копирования для поиска по адресу.\n";

system("pause"); exit(1);

}

for (i = 0; i < l2; i++)

pz2[i] = z.pz2[i];

}

l3 = z.l3;

if (z.pz3 == NULL) pz3 = NULL;

else {

pz3 = new abonent[l3];

if (pz3 == NULL) {

cout << "нет памяти.\n";

cout << "Конструктор копирования для поиска информации по Ф.И.О.\n";

system("pause"); exit(1);

}

for (i = 0; i < l3; i++)

pz3[i] = z.pz3[i];

}

} // Конец определения конструктора копирования класса masC

//Определение перегруженной операции присваивания класса masC

masC& masC :: operator= (masC& z)

{

int i;

if (this == &z) return \*this;

// Вызов перегруженной операции присваивания класса masB для

masB :: operator=(z); // присваивания родительской части объекта z

//Перегруженная операция присваивания для поиска по плате

if (pz!= NULL) delete[] pz;

l = z.l;

if (z.pz == NULL) pz = NULL;

else {

pz = new abonent[l];

if (pz == NULL) {

cout << "Нет памяти при присваивании. \n";

system("pause"); return z;

}

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

pz[i] = z.pz[i];

}

//Перегруженная операция присваивания для поиска по адресу

if (pz2!= NULL) delete[] pz2;

l2 = z.l2;

if (z.pz2 == NULL) pz2 = NULL;

else {

pz2 = new abonent[l2];

if (pz2 == NULL) {

cout << "Нет памяти при присваивании. \n";

system("pause"); return z;

}

for (i = 0; i < l2; i++)

pz2[i] = z.pz2[i];

}

//Перегруженная операция присваивания для поиска по Ф.И.О.

if (pz3!= NULL) delete[] pz3;

l3 = z.l3;

if (z.pz3 == NULL) pz3 = NULL;

else {

pz3 = new abonent[l3];

if (pz3 == NULL) {

cout << "Нет памяти при присваивании. \n";

system("pause"); return z;

}

for (i = 0; i < l3; i++)

pz3[i] = z.pz3[i];

}

return\* this;

}//Конец определения перезагрузки оператора присваивания класса masС \*/

//Определение метода findPlata() класса masС

void masC::findPlata()

{

int i, j;

int plata0;

abonent\* y;

y = new abonent[n];

if (y == NULL) {

cout << "нет памяти.\n";

system("pause"); return;

}

cout << " Введите искомую абонентскую плату: ";

cin >> plata0;

l = 0;

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

if (px[i].plata == plata0) { y[l] = px[i]; l++; }

if (pz != NULL) delete[] pz;

pz = new abonent[l];

if (pz == NULL) {

cout << "нет памяти.\n";

system("pause"); l = 0; delete[]y; return;

}

for (j = 0; j < l; j++)

pz[j] = y[j];

delete[] y;

cout << "Массив поиска по абонентской плате сформирован. \n";

system("pause");

} // Конец определения метода поиска по названию группы

//Определение метода findAddress() класса masС

void masC::findAddress()

{

int i, j;

string adress0;

abonent\* y2;

y2 = new abonent[n];

if (y2 == NULL) {

cout << "нет памяти.\n";

system("pause"); return;

}

cout << " Введите искомый адрес: ";

cin >> adress0;

l2 = 0;

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

if (px[i].pers.adress == adress0) { y2[l2] = px[i]; l2++; }

if (pz2 != NULL) delete[] pz2;

pz2 = new abonent[l2];

if (pz2 == NULL) {

cout << "нет памяти.\n";

system("pause"); l2 = 0; delete[]y2; return;

}

for (j = 0; j < l2; j++)

pz2[j] = y2[j];

delete[] y2;

cout << "Массив поиска по адресу сформирован. \n";

system("pause");

}

//Определение метода findbyFIO() класса masС

void masC::findbyFIO()

{

int i, j;

string name0;

string iniz;

abonent\* y3;

y3 = new abonent[n];

if (y3 == NULL) {

cout << "нет памяти.\n";

system("pause"); return;

}

cout << " Введите искомое Ф.И.О. : ";

cin >> name0 >> iniz;;

l3 = 0;

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

if (px[i].pers.name == name0 + " " + iniz) { y3[l3] = px[i]; l3++; }

if (pz3 != NULL) delete[] pz3;

pz3 = new abonent[l3];

if (pz3 == NULL) {

cout << "нет памяти.\n";

system("pause"); l3 = 0; delete[]y3; return;

}

for (j = 0; j < l3; j++)

pz3[j] = y3[j];

delete[] y3;

cout << "Массив поиска по Ф.И.О. сформирован. \n";

system("pause");

} // Конец определения метода поиска по названию группы

//Определение метода sortPlataFIO() класса masС

void masC::sortPlataFIO() // Сортировка массива структур по ФИО

{

int i, fl, nn;

abonent t;

nn = l;

do {

fl = 0; nn--;

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

if (pz[i].pers.name > pz[i + 1].pers.name)

{

fl = 1; t = pz[i];

pz[i] = pz[i + 1];

pz[i + 1] = t;

}

} while (fl == 1);

cout << "Массив поиска по абонентской плате упорядочен по ФИО в алфавитном порядке\n";

system("pause");

}

//Определение метода sortAddressDate() класса masС

void masC::sortAddressDate() // Сортировка массива структур адресов по дате

{

int i, fl, nn;

abonent t;

nn = l2;

do {

fl = 0; nn--;

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

if (pz2[i].pers.adress > pz2[i + 1].pers.adress)

{

fl = 1; t = pz2[i];

pz2[i] = pz2[i + 1];

pz2[i + 1] = t;

}

} while (fl == 1);

cout << "Массив поиска по адресу упорядочен по дате заключения договора\n";

system("pause");

}

//Определение метода sortAddressNum() класса masС

void masC::sortAddressNum() // Сортировка массива структур адресов по номеру договора

{

int i, fl, nn;

abonent t;

nn = l3;

do {

fl = 0; nn--;

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

if (pz3[i].number > pz3[i + 1].number)

{

fl = 1; t = pz3[i];

pz3[i] = pz3[i + 1];

pz3[i + 1] = t;

}

} while (fl == 1);

cout << "Массив поиска по адресу упорядочен по номеру договора\n";

system("pause");

}

//Определение дружественной функции вывода на экран operator<<(ostream& out, masC& z) класса masc

ostream& operator<< (ostream& out, masC& z)

{

int i;

// Вывод заголовка таблицы результатов поиска по плате

cout << " Результат поиска по абонентской плате: \n";

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << "| | | Дата заключения | | | | | |\n";

out << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

// Вывод строк таблицы

for (i = 0; i < z.l; i++)

out << "| " << setw(2) << i + 1 << z.pz[i];

// out<<z.pz[i] это перегруженная операция вывода для типа student

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

// Вывод заголовка таблицы результатов поиска по адресу

cout << endl << " Результат поиска по адресу: \n";

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << "| | | Дата заключения | | | | | |\n";

out << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

// Вывод строк таблицы

for (i = 0; i < z.l2; i++)

out << "| " << setw(2) << i + 1 << z.pz2[i];

// out<<z.pz2[i] это перегруженная операция вывода для типа student

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

// Вывод заголовка таблицы результатов поиска по фио

cout << endl << " Результат поиска по Ф.И.О.: \n";

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

out << "| | | Дата заключения | | | | | |\n";

out << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

// Вывод строк таблицы

for (i = 0; i < z.l3; i++)

out << "| " << setw(2) << i + 1 << z.pz3[i];

// out<<z.pz3[i] это перегруженная операция вывода для типа student

out << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

system("pause");

return out;

}

//Определение дружественной функции вывода в файл operator<<(ofstream& fout, masС& z) класса masС

ofstream& operator<<(ofstream& fout, masC& z)

{

string file;

int i;

cout << "Введите имя выходного файла: "; cin >> file;

fout.open(file.c\_str());

if (fout.fail()) {

cout << file << "не создается.\n";

system("pause"); return fout;

}

// Вывод заголовка таблицы результатов поиска по плате

fout << " Результат поиска по абонентской плате: \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << "| | | Дата заключения | | | | | |\n";

fout << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

fout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | " << "\n";

// Вывод остальных строк таблицы

for (i = 0; i < z.l; i++)

fout << "|" << setw(3) << i + 1 << "|" << setw(16) << z.pz[i].number << "|" << setw(5) << z.pz[i].datzakl.d << "|" << setw(6) << z.pz[i].datzakl.m << "|" << setw(4) << z.pz[i].datzakl.y << "|" << setw(19) << z.pz[i].plata << "|" << setw(20) << z.pz[i].pers.name << "|" << setw(10) << z.pz[i].pers.region << "|" << setw(29) << z.pz[i].pers.adress << "|" << setw(11) << z.pz[i].pers.phone << "| \n";

fout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | \n";

// Вывод заголовка таблицы результатов поиска по адресу

fout << endl <<" Результат поиска по адресу: \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << "| | | Дата заключения | | | | | |\n";

fout << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

fout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | " << "\n";

// Вывод остальных строк таблицы

for (i = 0; i < z.l2; i++)

fout << "|" << setw(3) << i + 1 << "|" << setw(16) << z.pz2[i].number << "|" << setw(5) << z.pz2[i].datzakl.d << "|" << setw(6) << z.pz2[i].datzakl.m << "|" << setw(4) << z.pz2[i].datzakl.y << "|" << setw(19) << z.pz2[i].plata << "|" << setw(20) << z.pz2[i].pers.name << "|" << setw(10) << z.pz2[i].pers.region << "|" << setw(29) << z.pz2[i].pers.adress << "|" << setw(11) << z.pz2[i].pers.phone << "| \n";

fout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | \n";

// Вывод заголовка таблицы результатов поиска по фио

fout << endl << " Результат поиска по Ф.И.О.: \n";

fout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

fout << "| | | Дата заключения | | | | | |\n";

fout << "| № | Номер договора | День| Месяц| Год| Абонентская плата | Фамилия и инициалы | Район | Адрес | Телефон |\n";

fout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | " << "\n";

// Вывод остальных строк таблицы

for (i = 0; i < z.l3; i++)

fout << "|" << setw(3) << i + 1 << "|" << setw(16) << z.pz3[i].number << "|" << setw(5) << z.pz3[i].datzakl.d << "|" << setw(6) << z.pz3[i].datzakl.m << "|" << setw(4) << z.pz3[i].datzakl.y << "|" << setw(19) << z.pz3[i].plata << "|" << setw(20) << z.pz3[i].pers.name << "|" << setw(10) << z.pz3[i].pers.region << "|" << setw(29) << z.pz3[i].pers.adress << "|" << setw(11) << z.pz3[i].pers.phone << "| \n";

fout << "|" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << " | \n";

fout.close();

cout << "Массив структур сохранен в фаиле " << file << "\n";

system("pause");

return fout;

} // Конец определения перегруженной операции вывода в файл

//Основная программа

void main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "");

masC a;

int j;

ifstream fin;

ofstream fout;

while (1)

{

system("cls");

cout << " ------------------------------------------- МЕНЮ -----------------------------------------\n";

cout << " \n";

cout << " 1. Загрузка из файла\n";

cout << " 2. Вывод исходного массива на экран. \n";

cout << " 3. Вывод исходного массива в файл. \n";

cout << " 4. Добавление записи в исходный массив. \n";

cout << " 5. Удаление записи из исходного массива. \n";

cout << " 6. Сортировка исходного массива по ФИО. \n";

cout << " 7. Сортировка исходного массива по дате заключения договора. \n";

cout << " 8. Сортировка исходного массива по номеру договора. \n";

cout << " \n";

cout << " ПЕРЕЧНИ:\n";

cout << " 9. Формирование перечня дат заключения с кол - ом договоров. \n";

cout << " 10. Сортировка перечня дат заключения с кол - ом договоров по дате. \n";

cout << " \n";

cout << " 11. Формирование перечня районов с кол - ом абонентов \n";

cout << " 12. Сортировка перечня районов с кол - ом абонентов по району. \n";

cout << " \n";

cout << " 13. Формирование перечня плат \n";

cout << " 14. Сортировка перечня плат по кол - ву. \n";

cout << " \n";

cout << " 15. Вывод перечней на экран. \n";

cout << " 16. Вывод перечней в файл. \n";

cout << " \n";

cout << " ПОИСК:\n";

cout << " 17. Поиск по абонентской плате с кол - ом абонентов. \n";

cout << " 18. Сортировка платы по ФИО. \n";

cout << " 19. Поиск по адресу. \n";

cout << " 20. Сортировка адресов по дате \n";

cout << " 21. Сортировка адресов по номеру \n";

cout << " 22. Поиск по ФИО. \n";

cout << " 23. Вывод результатов поиска на экран. \n";

cout << " 24. Вывод результатов поиска в файл. \n";

cout << " \n";

cout << " 25. Проверка конструкторов копирования.\n";

cout << " 26. Проверка операций присваивания. \n";

cout << " 27. Выход из программы.\n";

cout << " ------------------------------------------------------------------------------------------\n";

cout << " Выберите пункт меню (1 - 27) :"; cin >> j;

cout << " ------------------------------------------------------------------------------------------\n";

cout << " \n";

//Защита от неправильного ввода пункта меню

if (cin.fail())

{

string s;

cin.clear();

cin >> s;

cout << "Это не пункт меню\n"; system("pause");

continue;//K следующей итерации

}

switch (j)

{

case 1: fin >> a; break;

case 2: cout << (masA&)a; break;

case 3: fout << (masA&)a; break;

case 4: a.addAbonent(); break;

case 5: a.deleteAbonent(); break;

case 6: a.masA::sortName(); break;

case 7: a.masA::sortDat(); break;

case 8: a.masA::sortNumber(); break;

case 9: a.makePerechDates(); break;

case 15: cout << (masB&)a; break;

case 16: fout << (masB&)a; break;

case 10: a.sortDates(); break;

case 11: a.makePerechRegions(); break;

case 12: a.masB::sortRegions(); break;

case 13: a.makePerechPlata(); break;

case 14: a.masB::sortCount(); break;

case 17: a.findPlata(); break;

case 23: cout << a; break;

case 24: fout << a; break;

case 18: a.sortPlataFIO(); break;

case 19: a.findAddress(); break;

case 20: a.sortAddressDate(); break;

case 21: a.sortAddressNum(); break;

case 22: a.findbyFIO(); break;

case 25: { masC b(a);

cout << (masA&)b << (masA&)a;

cout << (masB&)b << (masB&)a;

cout << b << a; } break;

case 26: { masC b, c; c = b = a;

cout << (masA&)c << (masA&)b << (masA&)a;

cout << (masB&)c << (masB&)b << (masB&)a;

cout << c << b << a; } break;

case 27: cout << "Конец работы\n"; system("pause"); return;

default: cout << "Ошибка : нет такого пункта меню \n"; system("pause");

}

}

}