Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных технологий

Кафедра информационных технологий и систем

Дисциплина: Основы конструирования программ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

РАЗРАБОТКА ПРОГРАММЫ УЧЕТА СВЕДЕНИЙ ОБ АБОНЕНТАХ СОТОВОЙ СВЯЗИ

Студент:
гр. 980161 Алейчик И.Д.

Руководитель:
Раднёнок А.Л.

Минск 2019

СОДЕРЖАНИЕ

1 ТРЕБОВАНИЯ К ПРОГРАММЕ

Разработать программу учета сведений об абонентах сотовой связи.

Оператор сотовой связи хранит ифнормацию о своих абонентах: Ф.И.О. абонента, номер телефона, год подключения и наименование текущего тарифоного плана.

Вывести список и подсчитать общее колвичество абонентов, подключенных с хххх года (год вводится с клавиатуры).

Реализовать авторизацию для входа в систему (без регистрации!), функционал администратора и функционал пользователя, как минимум три вида поиска, как минимум три вида сортировки.

Исходные требования к курсовой работе

1. Язык программирования C++.
2. Среда разработки Microsoft Visual Studio версии 2017 и выше.
3. Вид приложения – консольное.
4. Парадигма программирования – процедурная.
5. Способ организации данных – структуры.
6. Способ хранения данных – файлы.
7. Каждая логически завершенная задача программы должна быть реализована в виде функции.
8. Текст пояснительной записки оформляется в соответствии со стандартом «СТП 01–2017».

Функциональные требования к курсовой работе

Первым этапом работы программы является авторизация – предоставление прав. В рамках данного этапа необходимо считать данные файла, содержащей учетные записи пользователей следующего вида:

- login;
- password;
- role (данное поле служит для разделения в правах администраторов и пользователей).

После ввода пользователем своих персональных данных (логина и пароля) и сверки с информацией, находящейся в файле пользователя, необходимо предусмотреть возможность входа в качестве администратора (в

этом случае, например, role = 0) или в качестве пользователя (в этом случае, например, role = 1).

Если файла с учетными записями пользователей не существует, то необходимо её программно создать и записать учетные данные администратора.

Регистрация новых пользователей при входе в систему не предусмотрена. Данную задачу выполняет администратор в режиме работы с учетными записями пользователей.

Вторым этапом работы программы является собственно работа с данными, которая становится доступной только после прохождения авторизации. Данные хранятся в отдельном текстовом файле.

Для работы с данными должны быть предусмотрены два режима: режим администратора и режим пользователя.

Режим администратора включает следующие подмодули (с указанием функциональных возможностей):

1. Управление учетными записями пользователей:
    i. просмотр всех учетных записей;
    ii. добавление новой учетной записи;
    iii. редактирование учетной записи;
    iv. удаление учетной записи.
2. Работа с данными:
    a. режим редактирования:
        i. просмотр всех данных;
        ii. добавление новой записи;
        iii. удаление записи;
        iv. редактирование записи;
    b. режим обработки данных:
        i. вывести список всех абонентов;
        ii. поиск(как минимум три вида):
            – поиск абонентов по фамилии и имени;
            – поиск абонентов по имени и отчеству;
            – поиск абонентов по номеру телефона;
            – поиск абонентов абонентов подключенных с хххх года
        iii. сортировка (как минимум три вида):
            – сортировка абонентов по алфавиту;
            – сортировка абонентов по году подключения;
            – сортировка абонентов по имени.

Режим пользователя включает подмодуль работы с данными со следующими функциональными возможностями:

1. Просмотр всех данных.
2. Поиск данных:
    a. поиск абонентов по фамилиии и имени;
    b. поиск абонентов по имени и отчеству;
    c. поиск абонентов по номеру телефона;
    d. поиск абонентов абонентов подключенных с хххх года
3. Сортировка:
    a. сортировка абонентов по алфавиту;
    b. сортировка абонентов по году подключения;
    c. сортировка абонентов по имени.

Для реализации перечисленных режимов необходимо создавать меню с соответствующими пунктами.

Предусмотреть:

1. обработку исключительных ситуаций:
    a. имя пользователя или пароль не верны;
    b. запись с указанным идентификатором или логином не найдена;
    c. пользователь с таким именем уже существует;
    d. ведённые данные не соответствуют формату поля;
2. возможность возврата назад (навигация);
3. вывод сообщения об успешности создания/создания файла/записи.

Требования к программной реализации

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта к курсовой работе.
2. Имена функций должны быть осмысленными и строится по принципу «глагол + существительное». Если функция выполняет какую-либо проверку и возвращает результат типа bool, то ее название должно начинаться с глагола is (например, isFileExist, isUnicLogin).
3. Код не должен содержать неименованных числовых констант (так называемых «магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует выносить в глобальные переменные с атрибутом const. По правилам хорошего стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.

4. Код необходимо комментировать (как минимум в части нетривиальной логики).
5. Код не должен дублироваться – для этого существуют методы и функции.
6. Одна функция решает только одну задачу (например, не допускается в одной функции считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри функции возможен вызов других функций.
7. Следует избегать длинных функций и глубокой вложенности: текст функции должен умещаться на один экран, а вложенность блоков и операторов должна быть не более трёх.

2 КОНСТРУИРОВАНИЕ ПРОГРАММЫ

Реализация программы будет осуществляться на языке C++ в IDE-среде Microsoft Visual Studio 2019. Программа будет компилироваться и использоваться в операционных системах семейства Microsoft Windows версии 7 и выше.

2.1 Разработка модульной структуры программы

Согласно требованиям к программе, необходимо наличие исполняемой программы, которая работает с файлами данных пользователей программы и файлами данных абонентов. Следовательно, можно выделить два основных блока: блок работы с файломи данных пользователей и блок работы с файлами данных абонентов. На рисунке 1 показаны основные блоки программы.

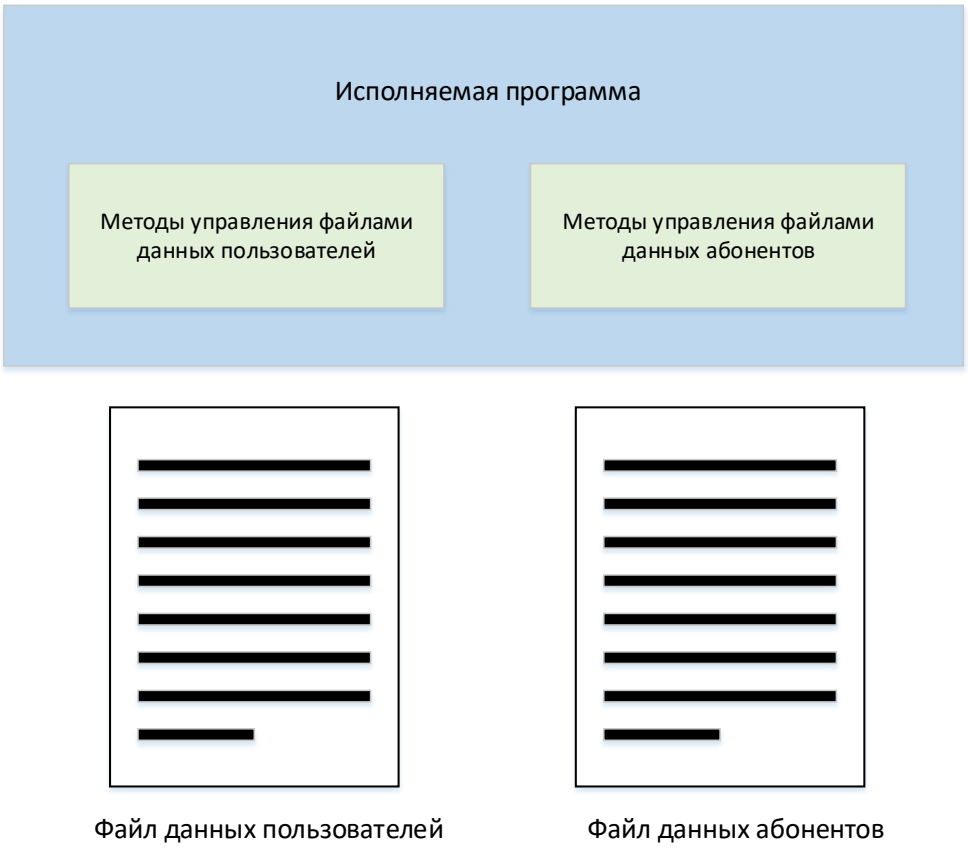

Рисунок 1 - Основные модули программы

Функция аутентификации пользователя заключается в проверке существования в файле данных пользователей введённого логина и соответствующего ему пароля. Авторизация пользователя подразумевает получение его роли из файла данных и предоставление ему соответствующих привилегий.

После успешной авторизации пользователя создаётся пользовательская сессия с соответствующими привилегиями, согласно роли пользователя. Сессия администратора имеет полный доступ к методам управления обеими файлами данных, а сессия пользователя, в свою очередь, имеет доступ только к методам управления файлом данных абонентов в режиме «только для чтения».

Программа подразумевает наличие консольного пользовательского интерфейса. В качестве модели (Model) здесь выступают классы с методами управления над файлами данных.

Связь сессий пользователей с управлением файлами данных будут осуществляться непосредственно через методы. Так, для сессии администратора требуется доступ к методам пользователей и к методам абонентов, а сессии пользователя – только к методам управления абонентами.

На рисунке 2 показаны основные классы программы.

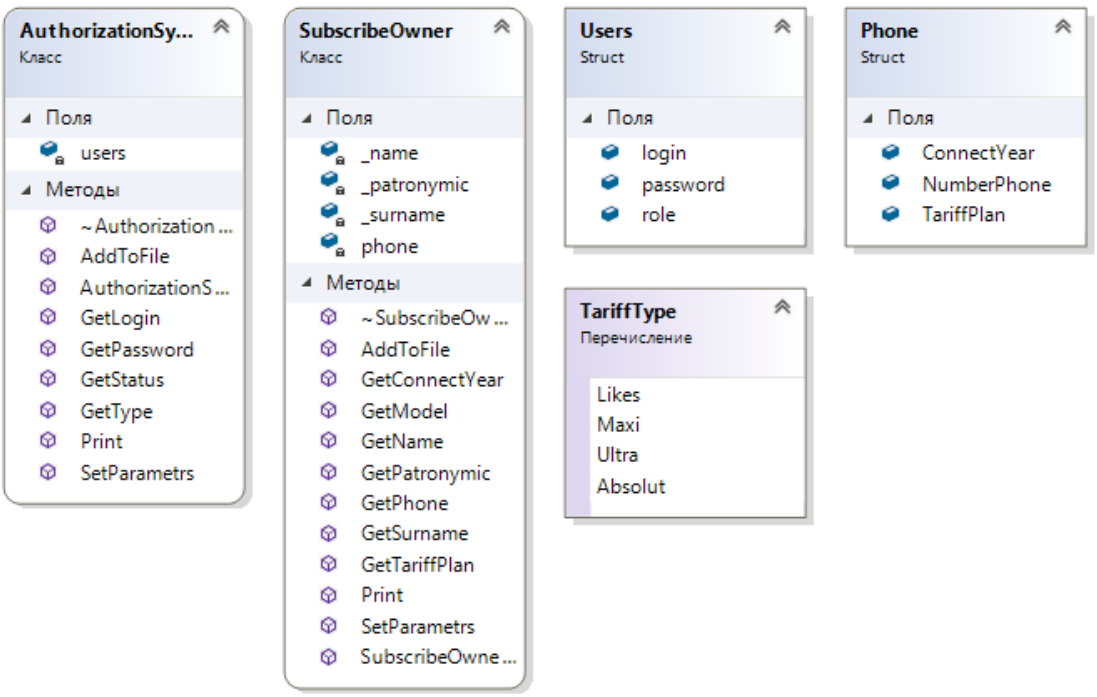

Рисунок 2 - Диаграмма классов программы

2.2 Выбор способа организации данных

Для представления в программе объекта пользователя вводится класс AuthorizationSystem, содержащий имя пользователя, пароль и его роль.

```
struct Users
{
    string login;
    string password;
    int role;
};

class AuthorizationSystem
{
private:
    Users users;
public:

    AuthorizationSystem();
    AuthorizationSystem(string login, string password, int role);
    ~AuthorizationSystem();

    void Print();
    void SetParametrs(string login, string password, int role);
    void AddToFile(string nameFile);

    string GetLogin();
    string GetPassword();
    string GetStatus(int a);
    int GetType();

protected:
};
```

Для представления в программе объекта абонента вводится класс SubscribeOwner, содержащий: ФИО абонента, номер телефона, тарифный план, год поключения.

```
enum TariffType
{
    Likes = 1,
    Maxi = 2,
    Ultra = 3,
    Absolut = 4

};

struct Phone
{
    string NumberPhone;
    string TariffPlan;
    string ConnectYear;
};

class SubscribeOwner
{
private:
    string _name, _surname, _patronymic;
    Phone phone;
protected:
public:
```

```
    SubscribeOwner();
    SubscribeOwner(string s, string n, string p, Phone ph);
    ~SubscribeOwner();

    void Print();
    void SetParametrs(string s, string n, string p, Phone ph);

    Phone GetModel();
    string GetSurname();
    string GetName();
    string GetPatronymic();
    string GetPhone();
    string GetConnectYear();
    string GetTariffPlan();

    void AddToFile(string nameFile);

};
```

В перечислении TariffType указано четыре тарифа для демонстрации возможности дальнейшего расширения списка.

Программа обслуживает данные, хранимые в двух текстовых файлах: файле данных пользователей и файле данных абонентов.

2.2.1 Структура файла данных пользователей

Для файла данных пользователей требуется хранить следующую информацию

- логин;
- пароль;
- роль .

2.2.2 Структура файла данных абонентов

Файл данных абонентов, согласно требованиям, должен содержать следующую информацию:

- ФИО абонента;
- Номер телефона;
- Год подключения;
- Текущий тарифный план;

2.3 Разработка перечня пользовательских функций программы

Ниже приведено описание публичных интерфейсов всех основных классов.

```
class SubscribeOwner
{
public:

   SubscribeOwner(string s, string n, string p, Phone ph);

   void Print();
   void SetParametrs(string s, string n, string p, Phone ph);

   Phone GetModel();
   string GetSurname();
   string GetName();
   string GetPatronymic();
   string GetPhone();
   string GetConnectYear();
   string GetTariffPlan();

   void AddToFile(string nameFile);

};


class AuthorizationSystem
{
public:

   AuthorizationSystem(string login, string password, int role);
   void Print();
   void SetParametrs(string login, string password, int role);
   void AddToFile(string nameFile);

   string GetLogin();
   string GetPassword();
   string GetStatus(int a);
   int GetType();

};
```

3 РАЗРАБОТКА АЛГОРИТМОВ РАБОТЫ ПРОГРАММЫ

3.1 Алгоритм функции main

Функция main является точкой входа в программу, вызывает основные функции инициализации, создаёт объект класса AuthorizationSystem. На рисунке 3 показана блок-схема алгоритма функции main.

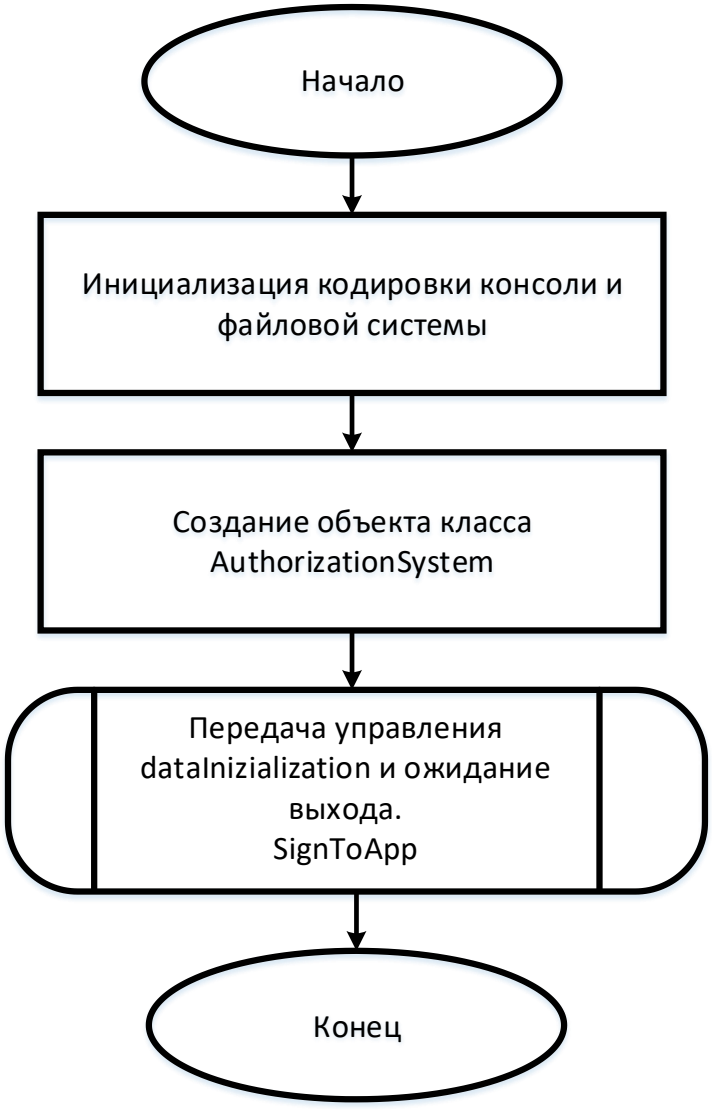

Рисунок 3 - Блок-схема алгоритма функции main

3.2 Алгоритм функции signToApp

Метод signToApp авторизует пользователя и запускает сеанс диалогового взаимодействия с пользователем (рисунок 4).

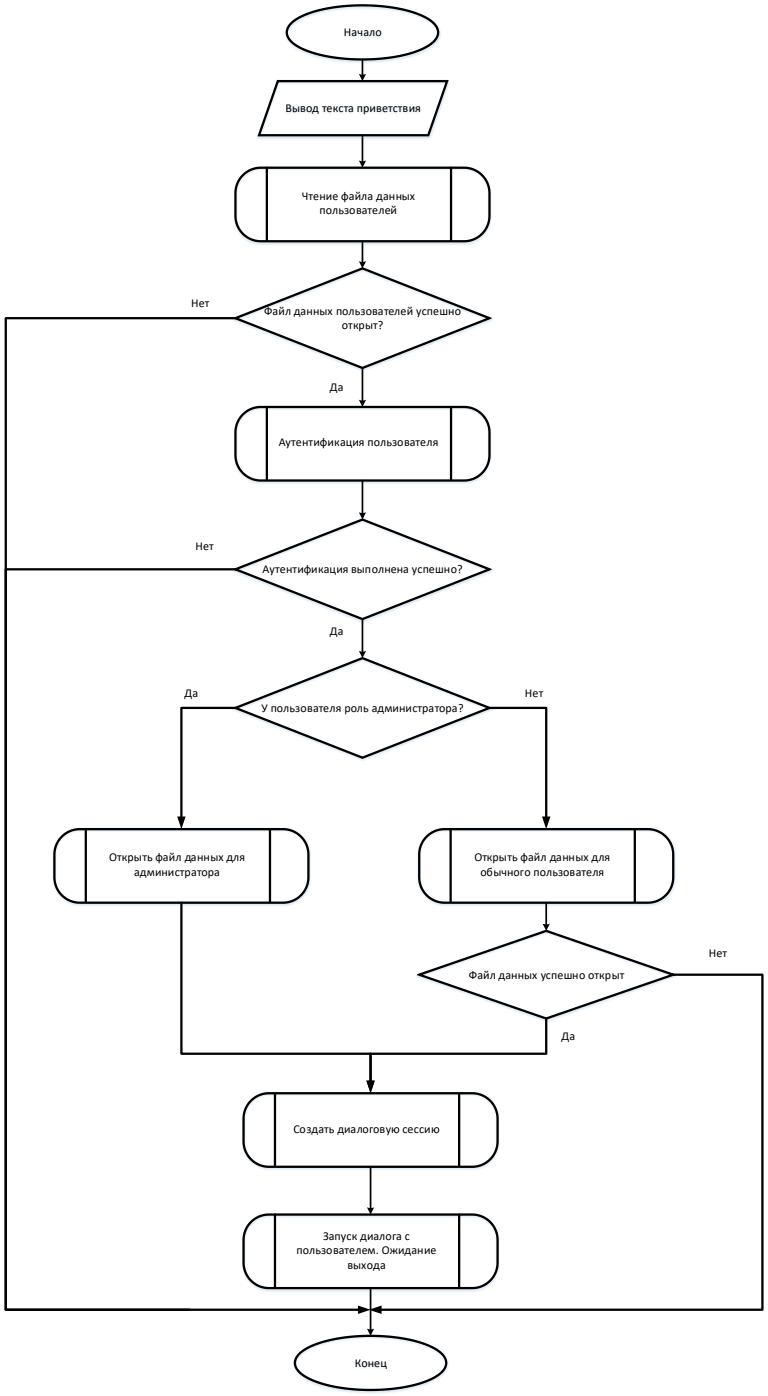

Рисунок 4 - Блок-схема алгоритма метода signToApp

3.3 Алгоритм функции checkUserPassword

Проверка правильности ввода логина пользователя и пароля при аутентификации заключается в поиске записи с указанным логином и ключом, полученным из пароля.

Алгоритм метода checkUserPassword показан на рисунке 5.

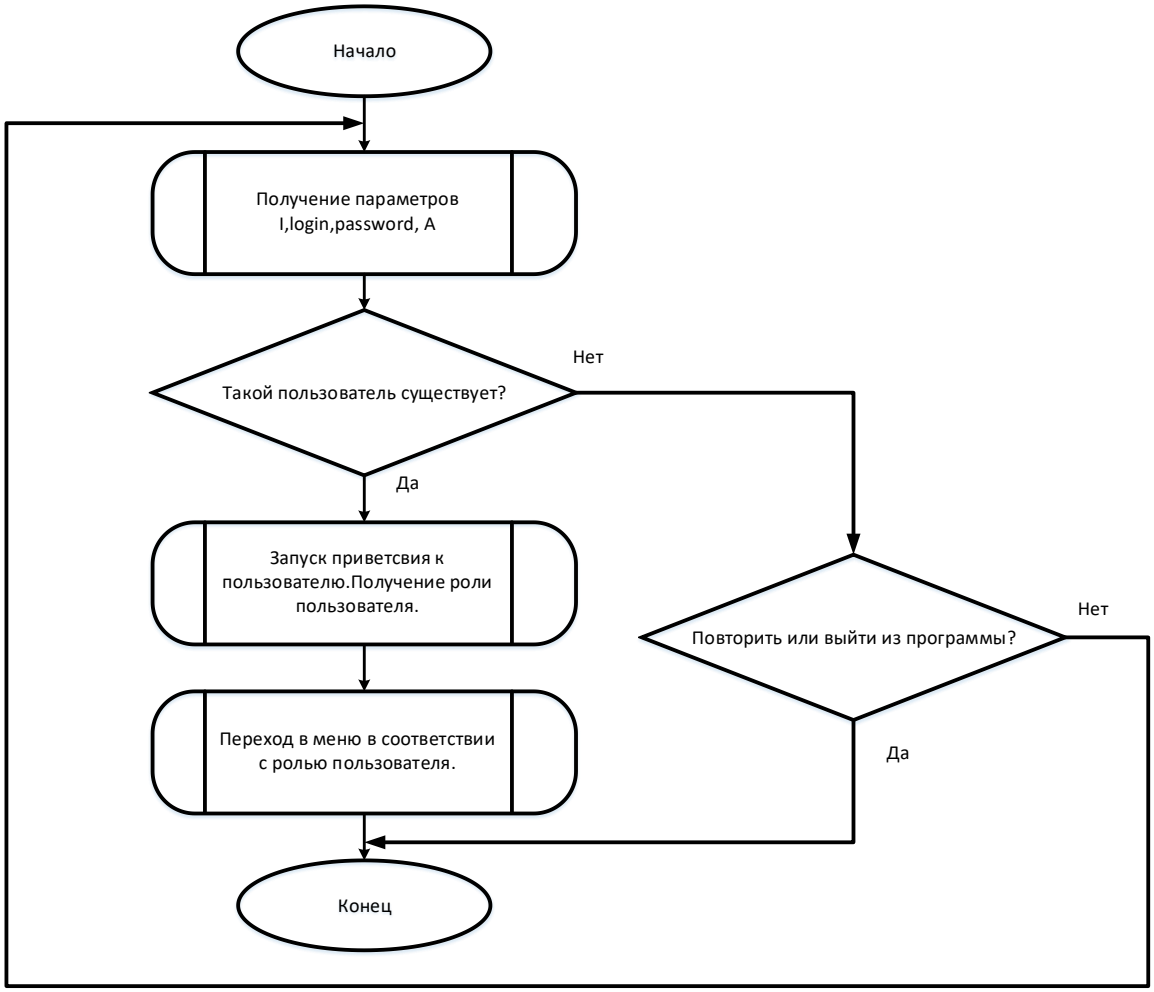

Рисунок 5 - Блок-схема алгоритма метода checkUserPassword

4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

При запуске программа печатает приветствие и пытается найти и открыть файл с данными пользователей. Если файл не найден, то программа автоматически создает файл данных. При создании нового файла необходимо сразу добавить в неё учётную запись администратора.

Поиск и создание файла данных пользователей производится в текущей директории программы. Имя файла жёстко прописано в коде программы: «Userdata.txt».

Если файл данных успешно открыт, то программа проверяет в ней наличие учётных записей администратора и если их нет, предлагает добавить (рисунок 6).

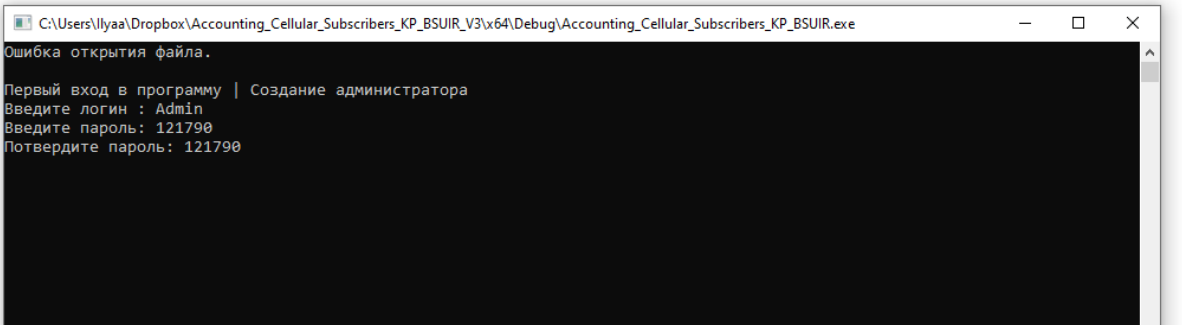

Рисунок 6 - Создание файла данных пользователей при запуске программы

4.1 Авторизация

После успешного открытия файла с данными пользователей (только для чтения) программа запрашивает учётные записи пользователя (логин и пароль) для его авторизации до тех пор, пока авторизация не будет успешно пройдена, либо пока пользователь не решит отменить процедуру входа: в этом случае программа завершит работу.

Для проверки корректности ввода логина и пароля программа обращается напрямую к функции системы авторизации, который инкапсулирует алгоритм обработки и проверки пароля и его соответствия имени пользователя.

После успешной аутентификации пользователя программа получает из файла данных пользователей роль аутентифицированного пользователя и авторизует его, создав соответствующую роли сессию: сессию администратора, которой будет соответствовать в дальнейшем описании

режим администратора либо сессию пользователя, которой будет соответствовать режим пользоватля.

Если пользователь авторизован с правами администратора, то программа выведет меню администратора (рисунок 7).

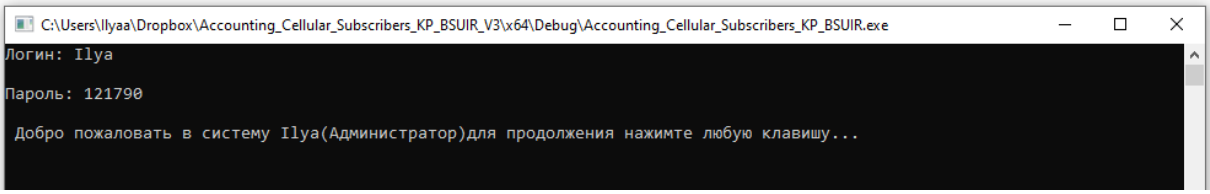

Рисунок 7 - Авторизация пользователя (администратора)

4.2 Модуль администратора

После входа администратора программа, для удобства, автоматически считывает файл данных абонентов «по умолчанию», если она есть. Файл данных «по умолчанию» должен находится в текущей директории программы, а его имя зашито в коде программы: «InputSubscribeOwner.txt».

Далее программа запускает диалоговое взаимодействие с пользователем путём отображения пронумерованных пунктов меню. На рисунке 8 показано главное меню модуля администратора.

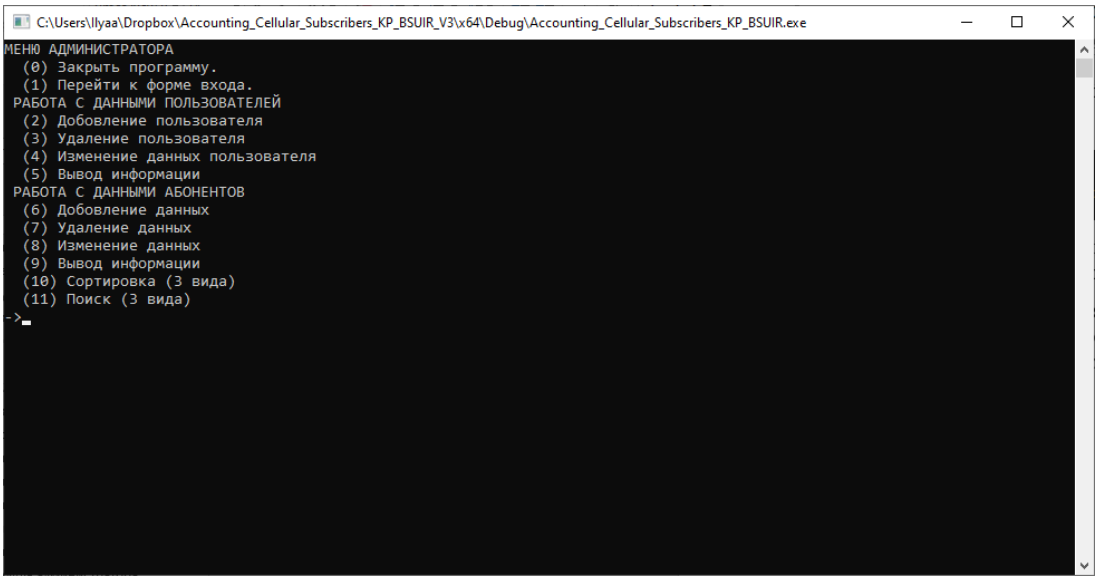

Рисунок 8 - Главное меню режима администратора

На рисунке 9 показано меню «Поиск абонентов».

Рисунок 9 - Меню «Поиск абонентов»

На рисунке 10 показано меню «Сортировка абонентов».

Рисунок 10 - Меню «Сортировка абонентов»

На рисунке 11 показан пример вывода всех учётных записей пользователей.

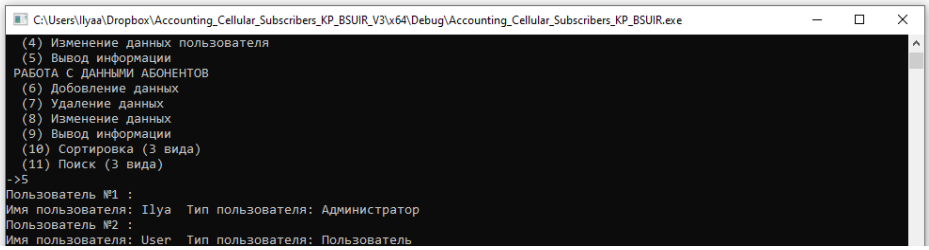

Рисунок 11 - Просмотр всех учётных записей пользователей

На рисунке 12 показан пример добавления новой учётной записи пользователя.

Рисунок 12 - Добавление новой учётной записи пользователя

На рисунке 13 показан пример изменения существующей учётной записи пользователя.

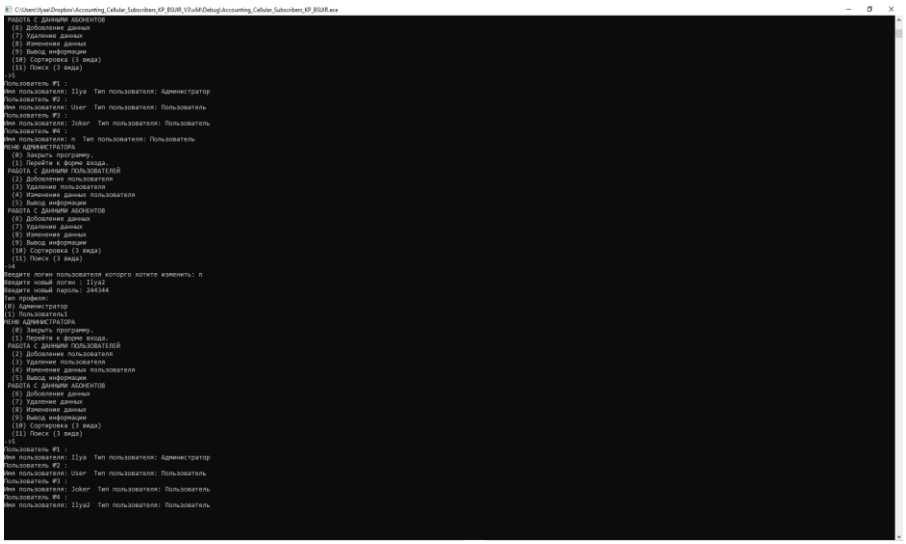

Рисунок 13 - Изменение учётной записи пользователя

На рисунке 14 показан пример удаления существующей учётной записи пользователя.

Рисунок 14 - Удаление учётной записи пользователя

На рисунке 15 показан пример добавления новой записи об абоненте в открытый файл данных абонентов. Если при попытке добавления новой записи открытых (активных) файлов не будет, то программа автоамтически создаст его.

Рисунок 15 - Добавление абонента

На рисунке 16 показан пример вывода списка всех абонентов.

Рисунок 16 - Просмотр всех записей абонентов

На рисунке 17 показан пример изменения существующей записи о абоненте.

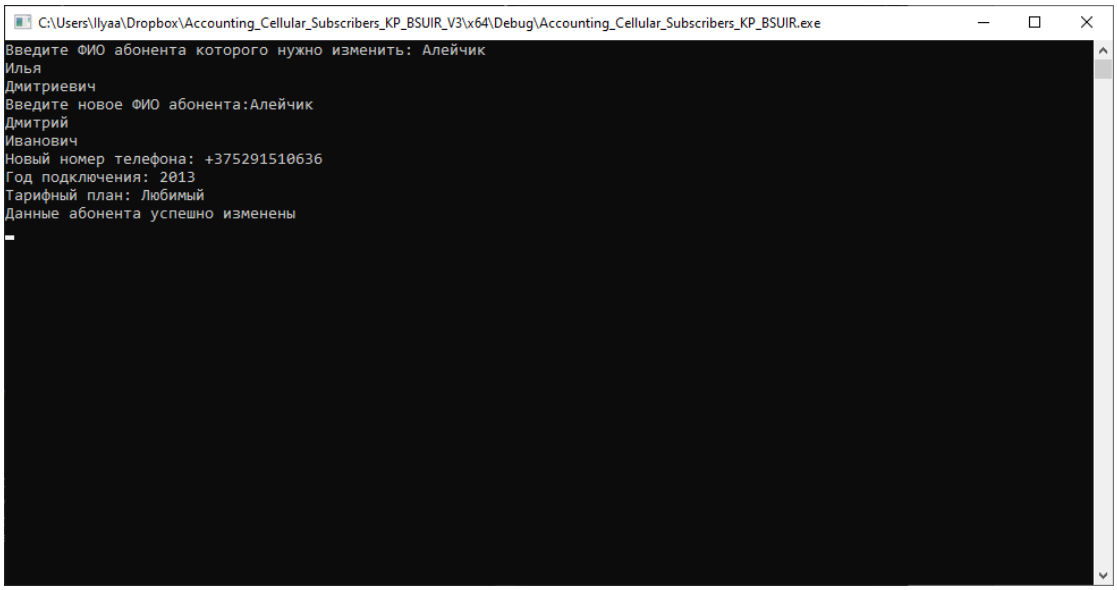

Рисунок 17 - Изменение записи абонента

На рисунке 18 показан пример удаления существующей записи об абоненте.

Рисунок 18 - Удаление записи об абоненте

На рисунке 19 показан пример вывода списка всех абонентов.

Рисунок 19 – Вывод списка всех абонентов

На рисунке 20 показан пример вывода списка абонентов подключенных с какого-то года (вводит пользователь) и подсчитывает их общее количество.

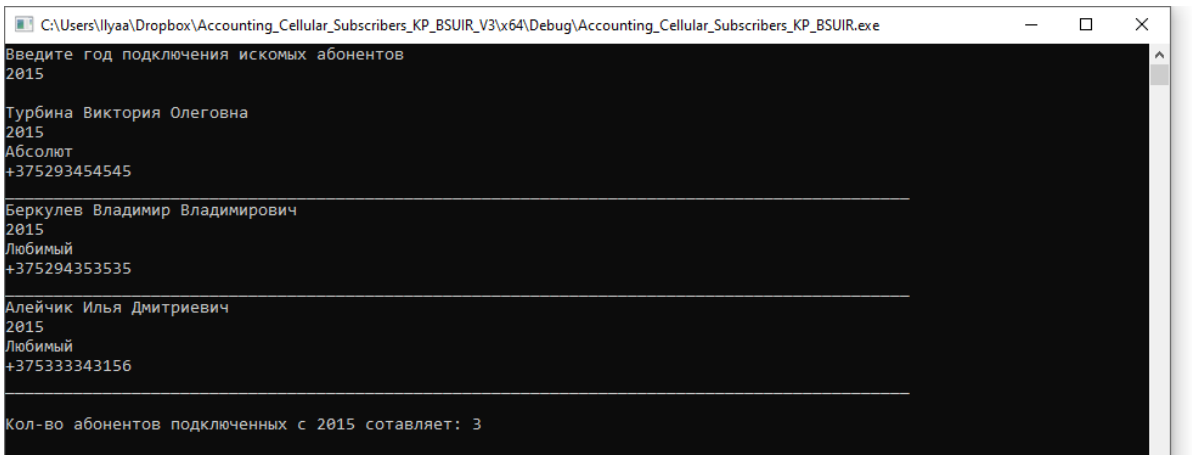

Рисунок 20 - Вывод абонентов подключенных с XXXX года с общим количеством

На рисунке 21 показан пример поиска абонентов по имени и фамилии.

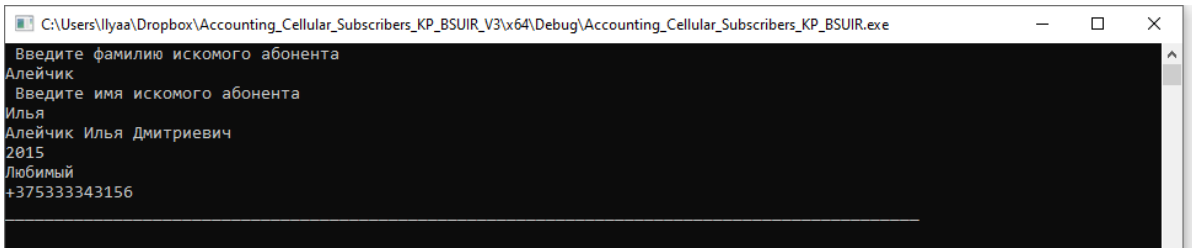

Рисунок 21 - Поиск абонентов по имени и фамилии

На рисунке 22 показан пример поиска абонентов по имени и отчеству.

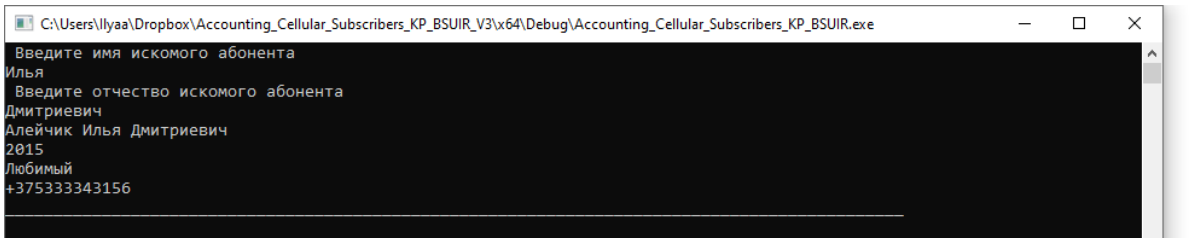

Рисунок 22 - Поиск абонентов по имени и отчеству

На рисунке 23 показан пример поиска абонентов по номеру телефона.

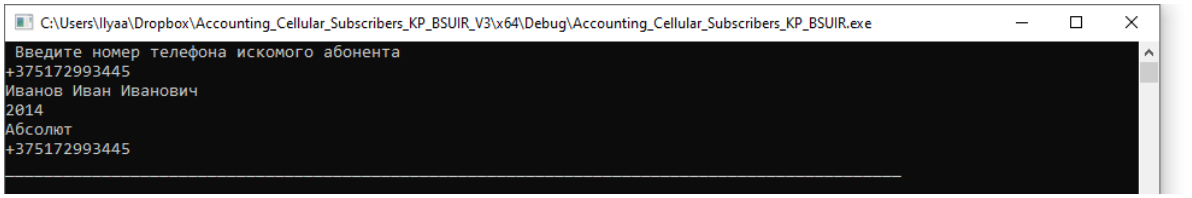

Рисунок 23 - Поиск абонентов по номеру телефона

На рисунке 24 показан пример вывода абонентов, отсортированных по алфавиту в порядке возрастания.

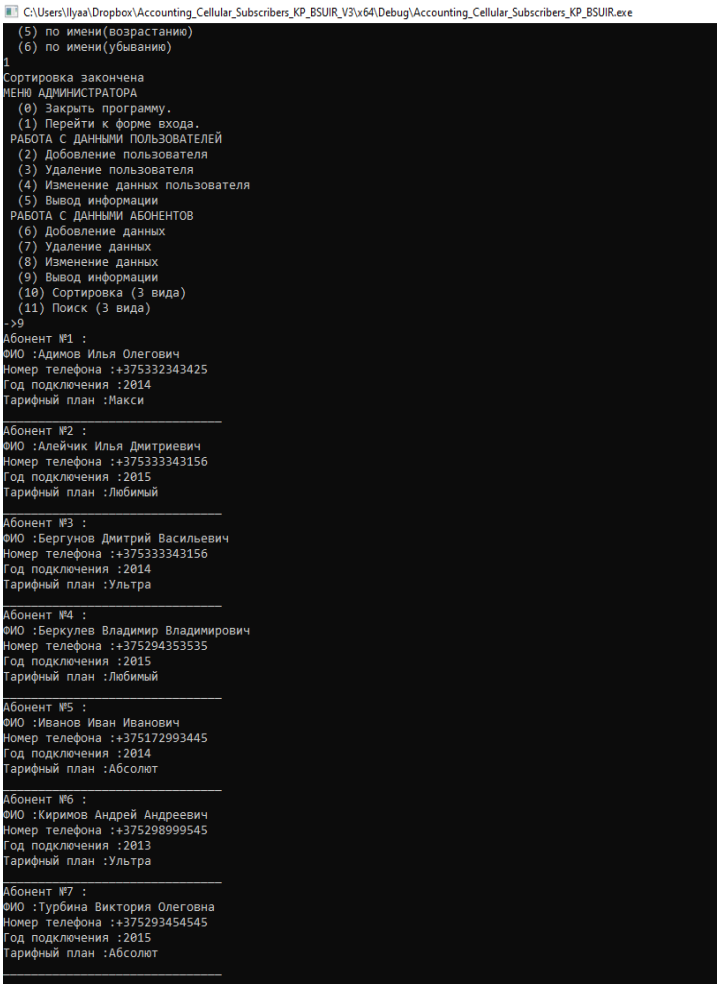

Рисунок 24 - Сортировка абонентов по алфавиту

На рисунке 25 показан пример вывода абонентов, отсортированных по году подключения в порядке убывания.

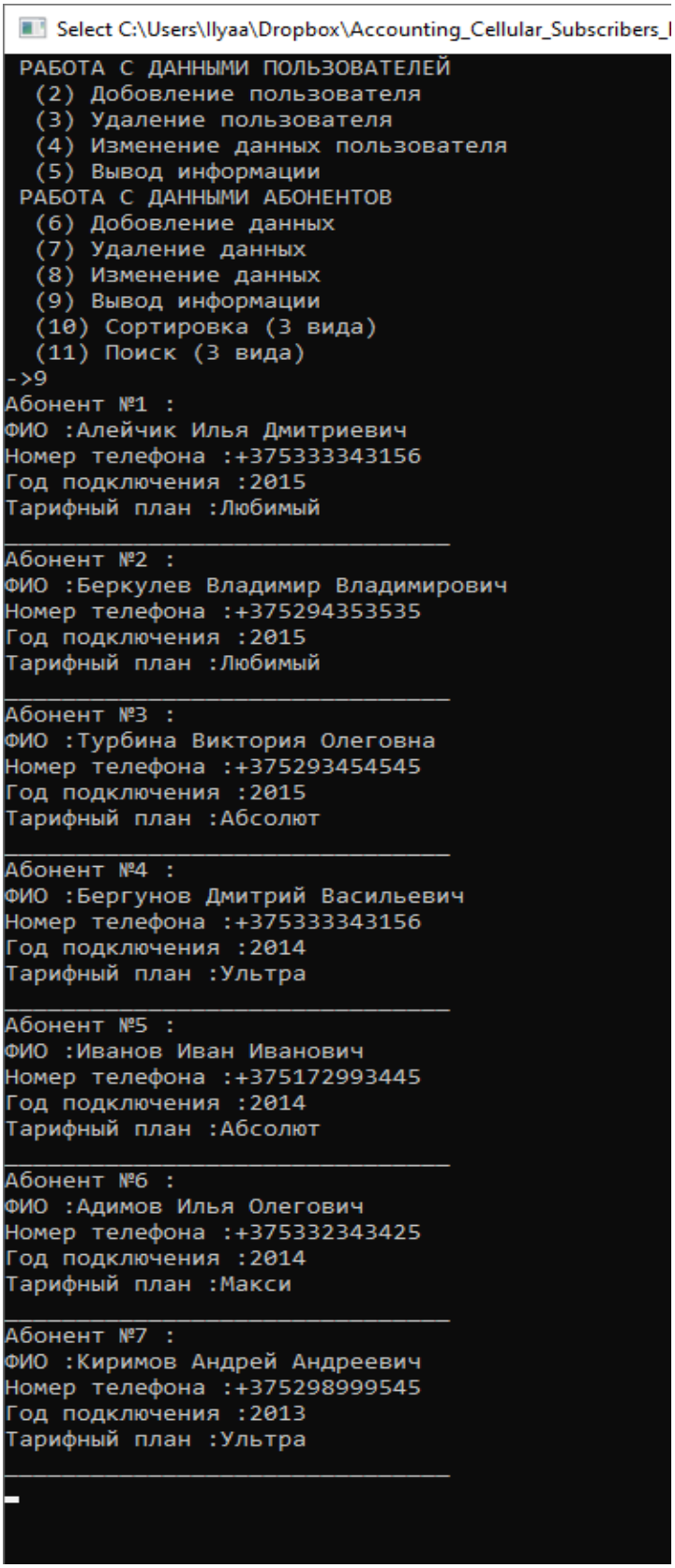

Рисунок 25 - Сортировка абонентов по году подключения

На рисунке 26 показан пример вывода абонентов, отсортированных по имени в порядке возрастания.

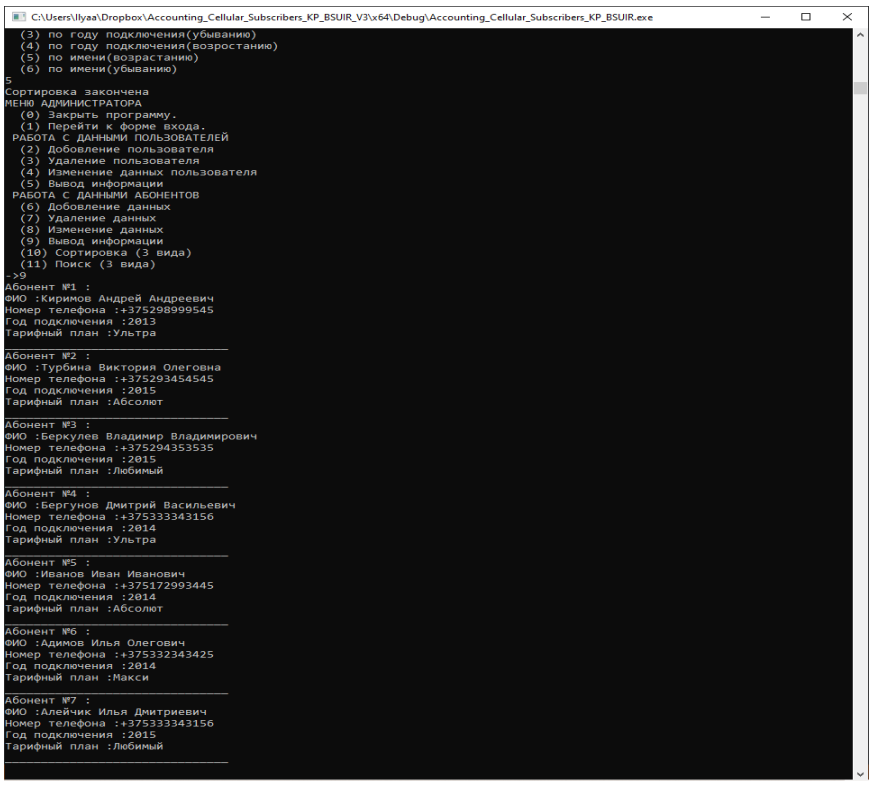

Рисунок 26 - Сортировка абонентов по имени

4.3 Модуль пользователя

Для входа как пользователь необходимо ввести соотвествующие данные (рисунок 27).

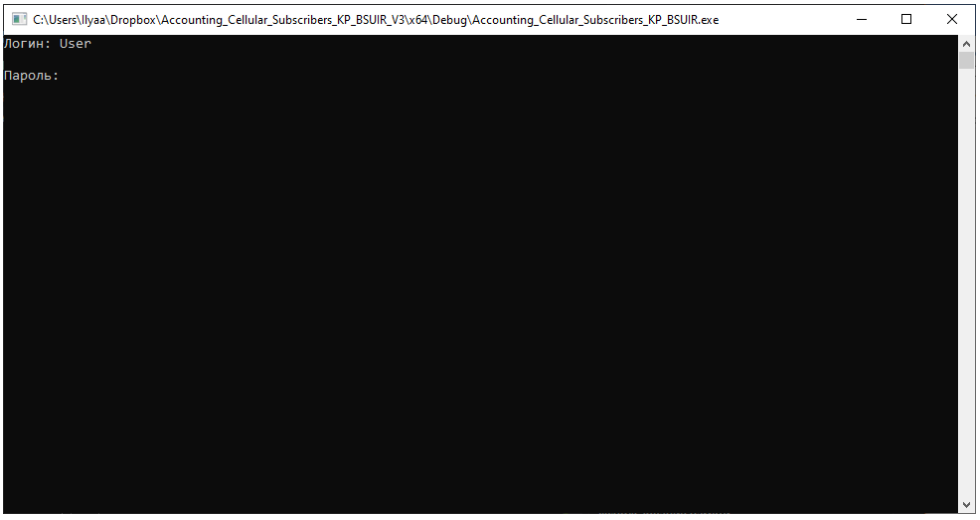

Рисунок 27 - Запрос пути к данным абонентов в модуле пользователя

После входа пользователя и успешного считывания файла данных абонентов программа запускает диалоговое взаимодействие с пользователем путём отображения пронумерованных пунктов меню и запросом у пользователя ввода желаемого номера (рисунок 28).

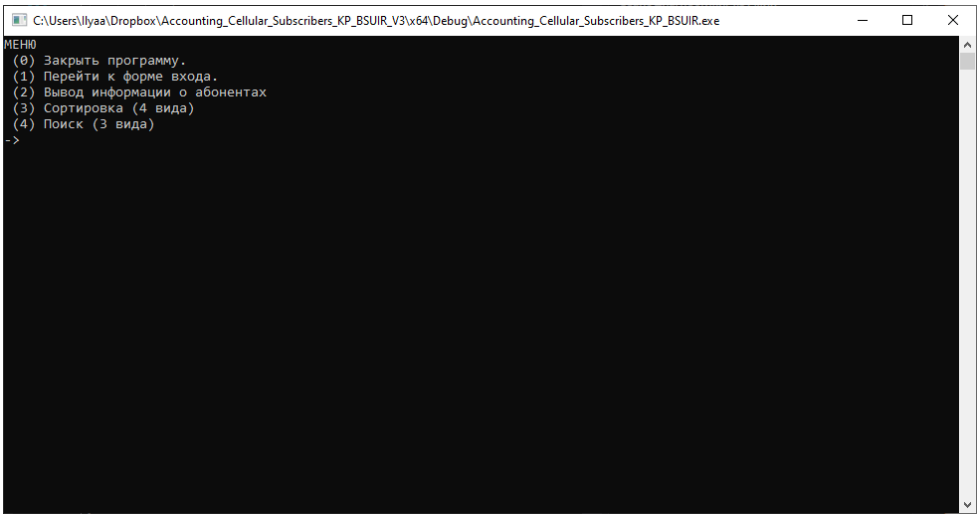

Рисунок 28 - Главное меню пользователя

Снимки экрана для всех операций в режиме пользователя совпадают со снимками экрана соответствующих операций в режиме администратора.

4.4 Исключительные ситуации

Обработка исключительных ситуаций, как правило, занимает существенную часть кода программы. Это неизбежно, потому что программа без обработки ошибок и пограничных ситуаций выглядит для пользователя некачественной и может приводить к множеству негативных последствий: повреждению данных, неверной трактовке результатов работы программы или нагрузкой на разработчика. В связи с этим, в программу встроено достаточно много проверок данных и результатов выполнения функций, обработок программных исключений, сопровождающихся уведомлением пользователя о проблеме и, при возможности, логичной реакцией на ситуацию либо прерыванием работы программы, если программа не смогла обработать исключительную ситуацию.

Первая, простейшая исключительная ситуация: введён неверный логин пользователя либо пароль. Программа сообщает об этом пользователю и спрашивает у пользователя, желает ли он повторить ввод данных (рисунок 29). Если пользователь ответит отрицательно, то программа завершит работу.

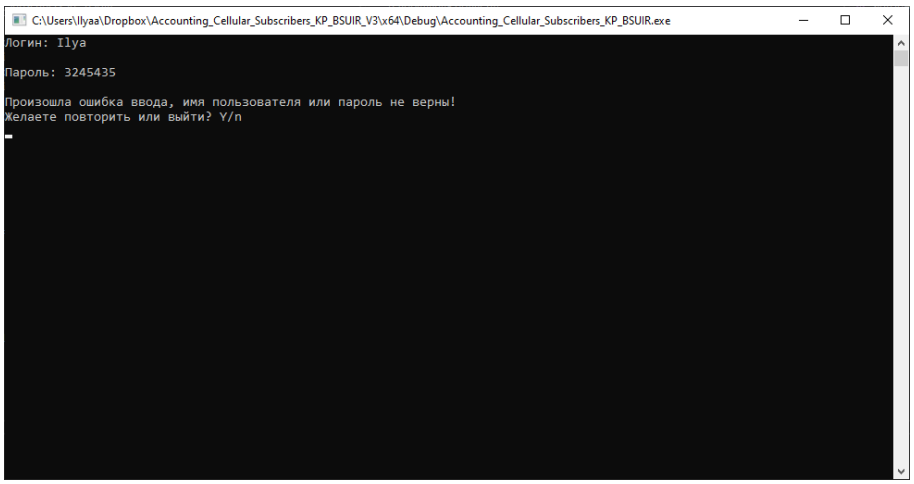

Рисунок 29 - Имя пользователя или пароль не верны

При вводе пользователем несуществующего имени абонента для таких операций, как редактирование или удаление, программа выведет соответствующее сообщение. На рисунке 30 показан пример отображения ошибки при вводе несуществующего имени учётной записи пользователя.

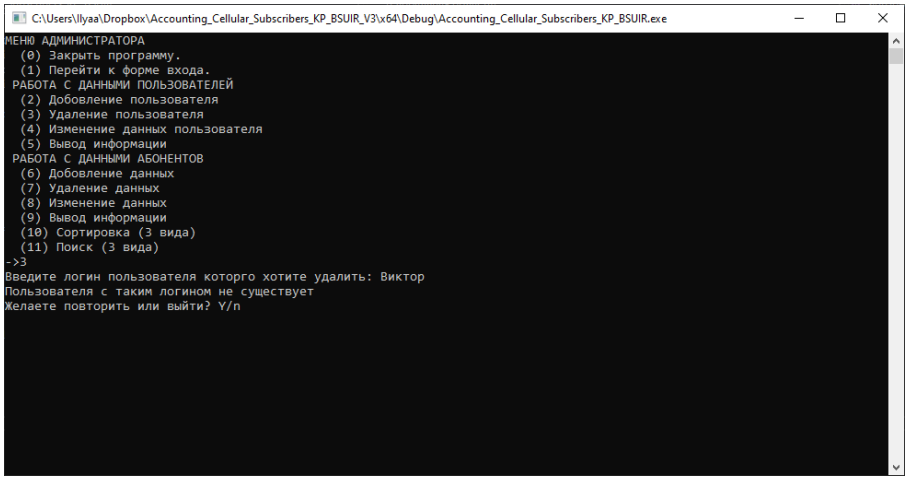

Рисунок 30 - Попытка удаления несуществующей записи

При попытке создания новой учётной записи с именем, которое уже существует, программа выведет соответствующее сообщение об ошибке (рисунок 31).

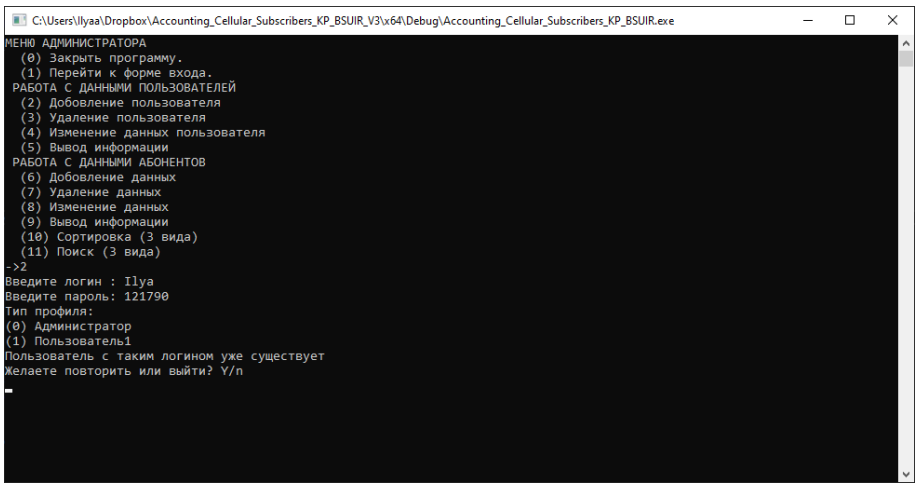

Рисунок 31 - Попытка создания учётной записи пользователя с уже существующим именем

Аналогичное сообщение будет выведено и при попытке изменения имени пользователя на такое, какое уже существует в текстовом файле (рисунок 32).

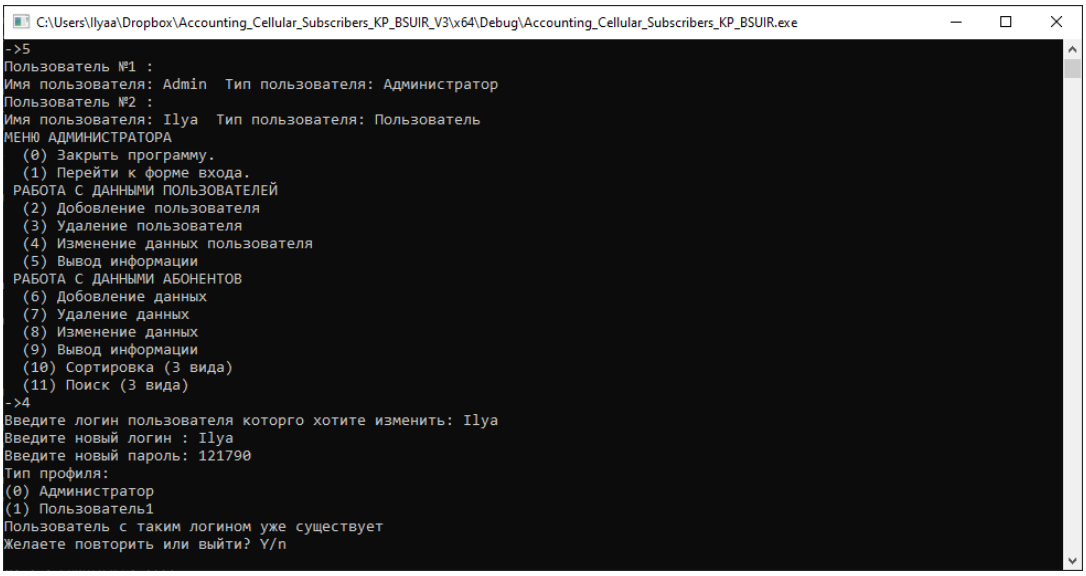

Рисунок 32 - Попытка изменения у пользователя имени на такое, какое уже существует в файле данных

При открытии файла с данными программа проверяет его существование. Если файла не существует, то программа выводит соответствующее сообщение и автоматически создает его. (рисунок 33).

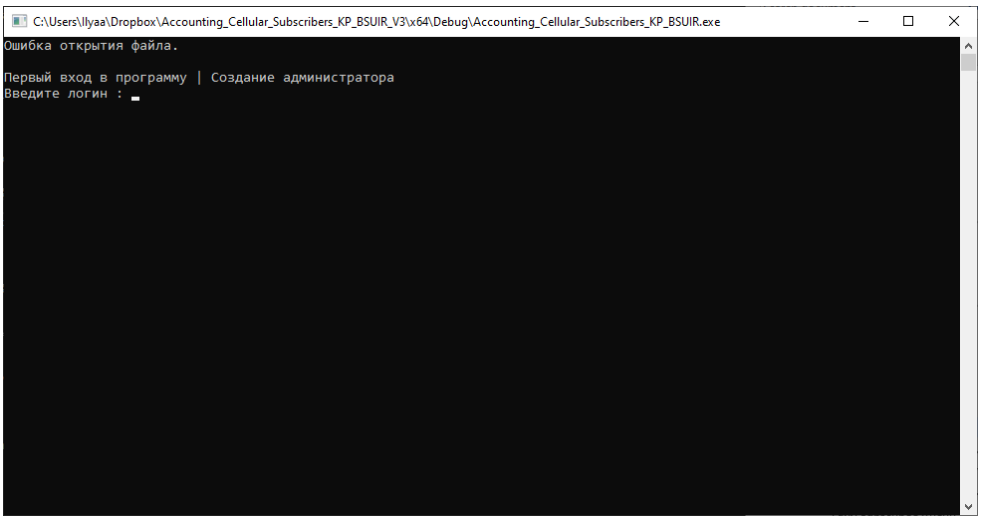

Рисунок 33 - Попытка открытия несуществующего файла

При вводе данных о абонентах программа проверяет корректность ввода полей. На рисунке 34 показан пример вывода сообщений об ошибках ввода значений для ФИО абонента (оно не может быть пустым), номера телефона, года подключения (должен быть в формате ГГГГ) и тарифного плана (не может содержать буквы).

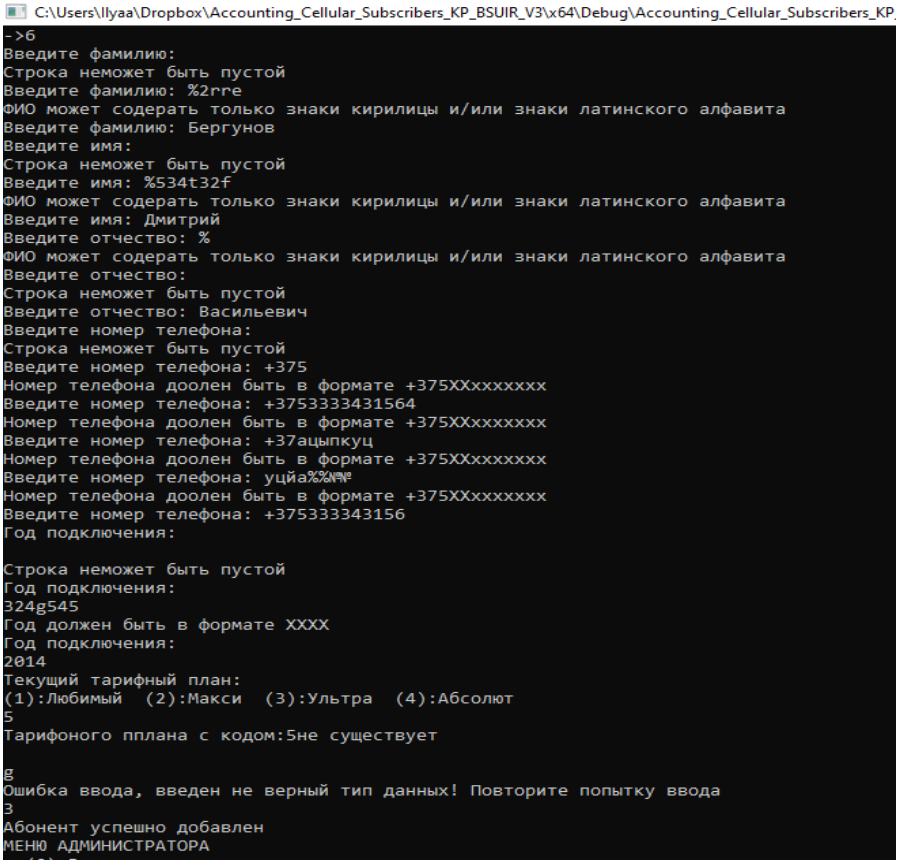

Рисунок 34 - Проверка корректности вводимых пользователем данных

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода с комментариями

Файл AuthorizationSystem.h

```
using namespace std;
struct Users
{
	string login;
	string password;
	int   role;
};

class AuthorizationSystem
{
private:
	Users users;
public:

	AuthorizationSystem();
	AuthorizationSystem(string login, string password, int role);
	~AuthorizationSystem();

	void Print();
	void SetParametrs(string login, string password, int role);
	void AddToFile(string nameFile);

	string GetLogin();
	string GetPassword();
	string GetStatus(int a);
	int GetType();

protected:
};
```

Файл AuthorizationSystem.cpp

```
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <stdlib.h>
#include <cstdlib>
#include <clocale>
#include <string>
#include <fstream>
#include <iomanip>
#include "AuthorizationSystem.h"

using namespace std;

AuthorizationSystem::AuthorizationSystem()
{
}

AuthorizationSystem::AuthorizationSystem(string login, string password, int role)
{
}

AuthorizationSystem::~AuthorizationSystem()
```

```
{
}

void AuthorizationSystem::Print()
{
	if (users.login != "") {
		cout << "Имя пользователя: " << users.login << "  Тип пользователя: " << GetStatus(users.role) << endl;
	}
}

void AuthorizationSystem::SetParametrs(string login, string password, int role)
{
	users.login = login;
	users.password = password;
	users.role = role;
}

void AuthorizationSystem::AddToFile(string nameFile)
{
	ofstream fout(nameFile, std::ios::app);

	if (fout.is_open())
	{
		if (users.login != "")
		{
			fout << users.login << "\n" << users.password << "\n" << users.role << "\n";
		}
	}
	else
	{
		cout << "Ошибка открытия файла\n";
	}
}

string AuthorizationSystem::GetLogin()
{
	return users.login;
}


string AuthorizationSystem::GetPassword()
{
	return users.password;
}

string AuthorizationSystem::GetStatus(int a)
{
	string res;
	a == 0 ? res = "Администратор" : res = "Пользователь";
	return res;
}

int AuthorizationSystem::GetType()
{
	return users.role;
}
```

Файл SubscribeOwner.h

```
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
```

```
#include <stdlib.h>

using namespace std;

enum TariffType
{
	Likes = 1,
	Maxi = 2,
	Ultra = 3,
	Absolut = 4

};

struct Phone
{
	string NumberPhone;
	string TariffPlan;
	string ConnectYear;
};

class SubscribeOwner
{
private:
	string _name, _surname, _patronymic;
	Phone phone;
protected:
public:

	SubscribeOwner();
	SubscribeOwner(string s, string n, string p, Phone ph);
	~SubscribeOwner();

	void Print();
	void SetParametrs(string s, string n, string p, Phone ph);

	Phone GetModel();
	string GetSurname();
	string GetName();
	string GetPatronymic();
	string GetPhone();
	string GetConnectYear();
	string GetTariffPlan();

	void AddToFile(string nameFile);

};
```

Файл SubscribeOwner.cpp

```
#include "SubscribeOwner.h"
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Main.h"

using namespace std;

SubscribeOwner::SubscribeOwner()
{
}

SubscribeOwner::SubscribeOwner(string s, string n, string p, Phone ph)
```

```
{
}

SubscribeOwner::~SubscribeOwner()
{
}

void SubscribeOwner::Print()
{
	if (_surname != "") {
		cout << "ФИО :" << _surname << " " << _name << " " << _patronymic << endl;
		cout << "Номер телефона :" << phone.NumberPhone << "\n";
		cout << "Год подключения :" << phone.ConnectYear << "\n";
		cout << "Тарифный план :" << phone.TariffPlan << "\n";
	}
}

void SubscribeOwner::SetParametrs(string s, string n, string p, Phone ph)
{
	_surname = s;
	_name = n;
	_patronymic = p;
	phone.ConnectYear = ph.ConnectYear;
	phone.NumberPhone = ph.NumberPhone;
	phone.TariffPlan = ph.TariffPlan;

}

Phone SubscribeOwner::GetModel()
{
	return phone;
}

string SubscribeOwner::GetSurname()
{
	return _surname;
}

string SubscribeOwner::GetName()
{
	return _name;
}

string SubscribeOwner::GetPatronymic()
{
	return _patronymic;
}

string SubscribeOwner::GetPhone()
{
	return phone.NumberPhone;
}

string SubscribeOwner::GetConnectYear()
{
	return phone.ConnectYear;
}

string SubscribeOwner::GetTariffPlan()
{
	return phone.TariffPlan;
}

void SubscribeOwner::AddToFile(string nameFile)
{
	ofstream fout(nameFile, std::ios::app);

	if (fout.is_open())
	{
```

```
            if (_surname != "")
            {
                fout << _surname << " " << _name << " " << _patronymic << "\n";
                fout << phone.NumberPhone << "\n" << phone.ConnectYear << "\n" << phone.TariffPlan << "\n";
            }

        }
        else
        {
            cout << "Ошибка открытия файла\n";
        }
}
```

Файл Main.cpp

```
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <stdlib.h>
#include <cstdlib>
#include <clocale>
#include <string>
#include <fstream>

#include "AuthorizationSystem.h"
#include "SubscribeOwner.h"
#include <regex>

using namespace std;


//входные и выходыне файлы
string InputUserdata = { "Userdata.txt" },
InputSubscribeOwner = { "InputSubscribeOwner.txt" };


//прототипы функций
void searchBy(int countSubcsribe, string nameInputFile, int stateMenu);
void sortBy(int countSubcsribe, string nameInputFile, int stateMenu);
void signToApp();
void readFromFileSubscribeOwner(SubscribeOwner* S, string nameFile);
void administratorMenu(int st);
void writeInFileInputSubscribeOwner(int n, string nameF, SubscribeOwner* C);
void userMenu(int st);
void viewMenu(int n);
void administratorMenu(int st);

//переход в главное меню с выбором режима пользователя/администратора
void viewMenu(int n) {
    n == 1 ? administratorMenu(0) : userMenu(0);
}

//функция поиска
void searchBy(int countSubcsribe, string nameInputFile, int stateMenu) {
    cout << " ПОИСК\n"
        << "  (1) по фамилии и имени\n"
          "  (2) по имени и отчеству\n"
        << "  (3) по номеру телефона\n"
        << "  (4) по году подключения (индивидуальное)\n";

    Phone ph;
    string Surname, Name, Patronymic;
```

```
	SubscribeOwner* S1 = new SubscribeOwner[countSubcsribe];

	readFromFileSubscribeOwner(S1, nameInputFile);

	int a,count=0;
	cin >> a;

	switch (a)
	{
	case 1:
		system("cls");

		cout << " Введите фамилию искомого абонента\n";
		cin >> Surname;
		cout << " Введите имя искомого абонента\n";
		cin >> Name;
		for (int i = 0; i < countSubcsribe; i++)
		{
			if (S1[i].GetName() == Name && S1[i].GetSurname() == Surname)
			{
				if (S1[i].GetSurname() != "")
				{
					cout << S1[i].GetSurname() << " " << S1[i].GetName() << " " << S1[i].GetPatronymic() << "\n";
					cout << S1[i].GetConnectYear() << "\n" << S1[i].GetTariffPlan() << "\n" << S1[i].GetPhone() << "\n"
					<<
"_____\n";

				}

			}
		}
		break;
	case 2:
		system("cls");

		cout << " Введите имя искомого абонента\n";
		cin >> Name;
		cout << " Введите отчество искомого абонента\n";
		cin >> Patronymic;

		for (int i = 0; i < countSubcsribe; i++)
		{
			if (S1[i].GetName() == Name && S1[i].GetPatronymic() == Patronymic) {
				if (S1[i].GetSurname() != "")
				{
					cout << S1[i].GetSurname() << " " << S1[i].GetName() << " " << S1[i].GetPatronymic() << "\n";
					cout << S1[i].GetConnectYear() << "\n" << S1[i].GetTariffPlan() << "\n" << S1[i].GetPhone() << "\n"
						<<
"_____\n";

				}
			}
		}
		break;
	case 3:
		system("cls");

		cout << " Введите номер телефона искомого абонента \n";
		cin >> ph.NumberPhone;

		for (int i = 0; i < countSubcsribe; i++)
		{
			if (S1[i].GetPhone() == ph.NumberPhone)
			{
				if (S1[i].GetSurname() != "")
				{
					cout << S1[i].GetSurname() << " " << S1[i].GetName() << " " << S1[i].GetPatronymic() << "\n";
					cout << S1[i].GetConnectYear() << "\n" << S1[i].GetTariffPlan() << "\n" << S1[i].GetPhone() << "\n"
```

```
                                    <<
"_____\n";

                        }
                    }

            }
            break;

        case 4:
            system("cls");

            cout << "Введите год подключения искомых абонентов \n";
            cin >> ph.ConnectYear;
            cout << "\n";
            for (int i = 0; i < countSubcsribe; i++)
            {
                if (S1[i].GetConnectYear() == ph.ConnectYear)
                {
                    if (S1[i].GetSurname() != "")
                    {
                        cout << S1[i].GetSurname() << " " << S1[i].GetName() << " " << S1[i].GetPatronymic() << "\n";
                        cout << S1[i].GetConnectYear() << "\n" << S1[i].GetTariffPlan() << "\n" << S1[i].GetPhone() << "\n"
                            <<
"_____\n";
                        count++;
                    }
                }

            }
            cout << "\nКол-во абонентов подключенных с "  + ph.ConnectYear + " сотавляет: " << count <<"\n";
            break;
        }

        _getch();
      viewMenu(stateMenu);

}

//функция сортировки
void sortBy(int countSubcsribe, string nameInputFile, int stateMenu) {
        Phone ph,ph2;
        string Surname, Name, Patronymic;

        cout << " СОРТИРОВКА\n"
            << "  (1) по алфавиту(возрастанию)\n"
            << "  (2) по алфавиту(убыванию)\n"
            << "  (3) по году подключения(убыванию)\n"
            << "  (4) по году подключения(возростанию)\n"
            << "  (5) по имени(возрастанию) \n"
           << "  (6) по имени(убыванию) \n";

        int sel;
        cin >> sel;
        SubscribeOwner *C = new SubscribeOwner[countSubcsribe];

        readFromFileSubscribeOwner(C, nameInputFile);

        switch (sel)
        {
        case 1:

            for (int i = 0; i < countSubcsribe; i++)
            {
                for (int j = i + 1; j < countSubcsribe; j++)
                {
                    if (C[i].GetSurname() > C[j].GetSurname())
                    {
                        Surname = C[i].GetSurname();
```

```
                    Name = C[i].GetName();
                    Patronymic = C[i].GetPatronymic();

                    ph.ConnectYear = C[i].GetConnectYear();
                    ph.TariffPlan = C[i].GetTariffPlan();
                    ph.NumberPhone = C[i].GetPhone();

                    ph2.ConnectYear = C[j].GetConnectYear();
                    ph2.TariffPlan = C[j].GetTariffPlan();
                    ph2.NumberPhone = C[j].GetPhone();


                    SubscribeOwner Buf(Surname, Name, Patronymic, ph);
                    C[i].SetParametrs(C[j].GetSurname(), C[j].GetName(), C[j].GetPatronymic(), C[j].GetModel());
                    C[j].SetParametrs(Surname, Name, Patronymic, ph);
                }
            }
        }
        break;
    case 2:

        for (int i = 0; i < countSubcsribe; i++)
        {
            for (int j = i + 1; j < countSubcsribe; j++)
            {
                if (C[i].GetSurname() < C[j].GetSurname())
                {
                    Surname = C[i].GetSurname();
                    Name = C[i].GetName();
                    Patronymic = C[i].GetPatronymic();

                    ph.ConnectYear = C[i].GetConnectYear();
                    ph.TariffPlan = C[i].GetTariffPlan();
                    ph.NumberPhone = C[i].GetPhone();

                    ph2.ConnectYear = C[j].GetConnectYear();
                    ph2.TariffPlan = C[j].GetTariffPlan();
                    ph2.NumberPhone = C[j].GetPhone();


                    SubscribeOwner Buf(Surname, Name, Patronymic, ph);
                    C[i].SetParametrs(C[j].GetSurname(), C[j].GetName(), C[j].GetPatronymic(), C[j].GetModel());
                    C[j].SetParametrs(Surname, Name, Patronymic, ph);
                }
            }
        }
        break;
    case 3:

        for (int i = 0; i < countSubcsribe; i++)
        {
            for (int j = i + 1; j < countSubcsribe; j++)
            {
                if (C[i].GetConnectYear() < C[j].GetConnectYear())
                {
                    Surname = C[i].GetSurname();
                    Name = C[i].GetName();
                    Patronymic = C[i].GetPatronymic();

                    ph.ConnectYear = C[i].GetConnectYear();
                    ph.TariffPlan = C[i].GetTariffPlan();
                    ph.NumberPhone = C[i].GetPhone();

                    ph2.ConnectYear = C[j].GetConnectYear();
                    ph2.TariffPlan = C[j].GetTariffPlan();
                    ph2.NumberPhone = C[j].GetPhone();


                    SubscribeOwner Buf(Surname, Name, Patronymic, ph);
```

```
                    C[i].SetParametrs(C[j].GetSurname(), C[j].GetName(), C[j].GetPatronymic(), C[j].GetModel());
                    C[j].SetParametrs(Surname, Name, Patronymic, ph);
                }
            }
        }

        break;
    case 4:

        for (int i = 0; i < countSubcsribe; i++)
        {
            for (int j = i + 1; j < countSubcsribe; j++)
            {
                if (C[i].GetConnectYear() > C[j].GetConnectYear())
                {
                    Surname = C[i].GetSurname();
                    Name = C[i].GetName();
                    Patronymic = C[i].GetPatronymic();

                    ph.ConnectYear = C[i].GetConnectYear();
                    ph.TariffPlan = C[i].GetTariffPlan();
                    ph.NumberPhone = C[i].GetPhone();

                    ph2.ConnectYear = C[j].GetConnectYear();
                    ph2.TariffPlan = C[j].GetTariffPlan();
                    ph2.NumberPhone = C[j].GetPhone();


                    SubscribeOwner Buf(Surname, Name, Patronymic, ph);
                    C[i].SetParametrs(C[j].GetSurname(), C[j].GetName(), C[j].GetPatronymic(), C[j].GetModel());
                    C[j].SetParametrs(Surname, Name, Patronymic, ph);
                }
            }
        }

        break;
    case 5:

        for (int i = 0; i < countSubcsribe; i++)
        {
            for (int j = i + 1; j < countSubcsribe; j++)
            {
                if (C[i].GetName() > C[j].GetName())
                {
                    Surname = C[i].GetSurname();
                    Name = C[i].GetName();
                    Patronymic = C[i].GetPatronymic();

                    ph.ConnectYear = C[i].GetConnectYear();
                    ph.TariffPlan = C[i].GetTariffPlan();
                    ph.NumberPhone = C[i].GetPhone();

                    ph2.ConnectYear = C[j].GetConnectYear();
                    ph2.TariffPlan = C[j].GetTariffPlan();
                    ph2.NumberPhone = C[j].GetPhone();


                    SubscribeOwner Buf(Surname, Name, Patronymic, ph);
                    C[i].SetParametrs(C[j].GetSurname(), C[j].GetName(), C[j].GetPatronymic(), C[j].GetModel());
                    C[j].SetParametrs(Surname, Name, Patronymic, ph);
                }
            }
        }

        break;
    case 6:

        for (int i = 0; i < countSubcsribe; i++)
        {
```

```
                for (int j = i + 1; j < countSubcsribe; j++)
                {
                    if (C[i].GetName() < C[j].GetName())
                    {
                        Surname = C[i].GetSurname();
                        Name = C[i].GetName();
                        Patronymic = C[i].GetPatronymic();

                        ph.ConnectYear = C[i].GetConnectYear();
                        ph.TariffPlan = C[i].GetTariffPlan();
                        ph.NumberPhone = C[i].GetPhone();

                        ph2.ConnectYear = C[j].GetConnectYear();
                        ph2.TariffPlan = C[j].GetTariffPlan();
                        ph2.NumberPhone = C[j].GetPhone();


                        SubscribeOwner Buf(Surname, Name, Patronymic, ph);
                        C[i].SetParametrs(C[j].GetSurname(), C[j].GetName(), C[j].GetPatronymic(), C[j].GetModel());
                        C[j].SetParametrs(Surname, Name, Patronymic, ph);
                    }
                }
            }

            break;
        }



        writeInFileInputSubscribeOwner(countSubcsribe,nameInputFile, C);
        cout << "Сортировка закончена\n";
        _getch();
        viewMenu(stateMenu);
}




//подсчет кол-ва элементов файла
int countFromFile(string nameFile) {
    int k = 0;

    ifstream fin(nameFile);

    if (fin.is_open()) {
        fin >> k;
    }
    else {
        cout << "Ошибка открытия файла.\n";
    }
    fin.close();
    return k;
}
//запись в файл
void writeInFile(int n, string nameF, AuthorizationSystem* A) {
    ofstream fout(nameF, std::ios::out);

    if (fout.is_open())
    {
        fout << n << endl;
        for (int i = 0; i < n; i++)
        {
            A[i].AddToFile(nameF);
        }
    }
    else {
        cout << "Ошибка открытия файла \"OutputApartment.txt\".\n";
    }
```

```
	fout.close();
}
//чтение из файла
void readFromFile(AuthorizationSystem* A, string nameFile) {
	int k;
	Users user;

	ifstream fin(nameFile);

	if (fin.is_open()) {
		fin >> k;
		for (int i = 0; i < k; i++) {
			fin >> user.login >> user.password >> user.role;

			A[i].SetParametrs(user.login, user.password, user.role);

		}
	}
}
//функция добовления пользователя
void addDataUser(int countUser, string nameInputFile) {

	Users us;
	int NewCountUser = countUser + 1;
	bool isLogin = true;
	AuthorizationSystem* A1 = new AuthorizationSystem[NewCountUser];
	readFromFile(A1, nameInputFile);



	cout << "Введите логин : ";
	cin >> us.login;



	cout << "Введите пароль: ";
	cin >> us.password;
	cout << "Тип профиля: \n" <<
		"(0) Администратор \n" <<
		"(1) Пользователь\n>";

	cin >> us.role;

	for (int i = 0; i < countUser; i++)
	{
		if (us.login == A1[i].GetLogin())
		{
			isLogin = true;
			cout << "Пользователь с таким логином уже существует\n";
			break;
		}
		else
		{
			isLogin = false;
		}
	}
	if (isLogin == true)
	{

		cout << "Желаете повторить или выйти? Y/n \n";
		char c = _getch();
		if (c == 'n' || c == 'N')
		{
			administratorMenu(0);
		}
	}
	else
	{
		A1[NewCountUser - 1].SetParametrs(us.login, us.password, us.role);
```

```
            writeInFile(NewCountUser, nameInputFile, A1);

            cout << "Новый пользователь успешно создан\n";
        }

        _getch();
        administratorMenu(0);

}
//функция изменения данных пользователя
void alterDataUser(int countUser, string nameInputFile) {

        bool isLogin = true;
        string  login, password;
        int role, altIndex = 0;
        AuthorizationSystem *A = new AuthorizationSystem[countUser];

        readFromFile(A, nameInputFile);

        cout << "Введите логин пользователя которго хотите изменить: ";
        cin >> login;

        for (int i = 0; i < countUser; i++)
        {
            if (login == A[i].GetLogin())
            {
                altIndex = i;
            }
        }

        cout << "Введите новый логин : ";
        cin >> login;
        cout << "Введите новый пароль: ";
        cin >> password;
        cout << "Тип профиля: \n" <<
            "(0) Администратор \n" <<
            "(1) Пользователь\n>";
        cin >> role;


        for (int i = 0; i < countUser; i++)
        {
            if (login == A[i].GetLogin())
            {
                isLogin = true;
                cout << "Пользователь с таким логином уже существует\n";
                break;
            }
            else
            {
                isLogin = false;
            }
        }
        if (isLogin == true)
        {

            cout << "Желаете повторить или выйти? Y/n \n";
            char c = _getch();
            if (c == 'n' || c == 'N')
            {
                administratorMenu(0);
            }
        }
        else
        {
            A[altIndex].SetParametrs(login, password, role);
            writeInFile(countUser, nameInputFile, A);

            cout << "Изменение данных пользователя прошло успешно\n";
```

```
        }



        _getch();
        administratorMenu(0);
}
//функция удаления пользователя
void deleteDataUser(int countUser, string nameInputFile) {

        AuthorizationSystem *A = new AuthorizationSystem[countUser];
        readFromFile(A, nameInputFile);

        string  login, password = "";
        int delIndex, role = 0;
        cout << "Введите логин пользователя которго хотите удалить: ";
        cin >> login;


        for (int i = 0; i < countUser; i++)
        {
                if (login == A[i].GetLogin())
                {
                        delIndex = i;
                }
                else
                {
                        delIndex = -1;
                }
        }

        if (delIndex >= 0)
        {
                login = "";

                A[delIndex].SetParametrs(login, password, role);
                writeInFile(countUser, nameInputFile, A);

                A = new AuthorizationSystem[countFromFile(nameInputFile)];
                readFromFile(A, nameInputFile);

                writeInFile(countUser - 1, nameInputFile, A);

                cout << "Пользователь успешно удален \n";
                _getch();
                administratorMenu(0);
                system("cls");
        }
        else
        {
                cout << "Пользователя с таким логином не существует\n";
        }


        cout << "Желаете повторить или выйти? Y/n \n";
        char c = _getch();
        if (c =='n')
        {
                administratorMenu(0);
        }
}
//функция вывода списка пользователей
void viewDataUser(int countUser, string nameInputFile) {

                AuthorizationSystem* A = new AuthorizationSystem[countUser];
                readFromFile(A, nameInputFile);

                for (int i = 0; i < countUser; i++)
```

```
            {
                cout << "Пользователь №" << i + 1 << " :\n";
                A[i].Print();
            }

    _getch();
    administratorMenu(0);
}



//запись в файл данных абонента
void writeInFileInputSubscribeOwner(int n, string nameF, SubscribeOwner *C) {

    ofstream fout(nameF, std::ios::out);

    if (fout.is_open())
    {
        fout << n << endl;
        for (int i = 0; i < n; i++)
        {
            C[i].AddToFile(nameF);
        }
    }
    else {
        cout << "Ошибка открытия файла \"OutputApartment.txt\".\n";
    }
    fout.close();
}
//чтение из файла данных абонента
void readFromFileSubscribeOwner(SubscribeOwner *S, string nameFile) {
    int k;
    string Name, Surname, Patronymic;
    Phone ph;

    ifstream fin(nameFile);

    if (fin.is_open()) {
        fin >> k;
        for (int i = 0; i < k; i++) {
            fin >> Surname >> Name >> Patronymic;
            fin >> ph.NumberPhone >> ph.ConnectYear >> ph.TariffPlan;

            S[i].SetParametrs(Surname, Name, Patronymic, ph);

        }
    }
}


//добовление абонента
void addData(int countSubcsribe, string nameInputFile, int stateMenu) {


    string Surname, Name, Patronymic;
    Phone ph;

    regex phone("[+](375)[0-9]{2}[0-9]{7}");
    regex year("[0-9]{4}");
    regex text("^[a-яA-ЯёЁa-zA-Z]+$");

    int setType;
    int NewCountSubscribeOwner = countSubcsribe + 1;

    SubscribeOwner *S1 = new SubscribeOwner[NewCountSubscribeOwner];
    readFromFileSubscribeOwner(S1, nameInputFile);

    bool isValueName = false;
```

```
cin.ignore(32767, '\n');
do
{
	isValueName = false;
	cout << "Введите фамилию: ";
	getline(cin,Surname);
	if (Surname == "")
	{
		cout << "Строка неможет быть пустой\n";
		isValueName = true;
	}
	else
	{
		if (regex_match(Surname, text))
		{
			isValueName = false;
		}
		else
		{
			cout << "ФИО может содерать только знаки кирилицы и/или знаки латинского алфавита  \n";
			isValueName = true;
		}
	}
} while (isValueName != false);

do
{
	isValueName = false;
	cout << "Введите имя: ";
	getline(cin, Name);
	if (Name == "")
	{
		cout << "Строка неможет быть пустой\n";
		isValueName = true;
	}
	else
	{
		if (regex_match(Name, text))
		{
			isValueName = false;
		}
		else
		{
			cout << "ФИО может содерать только знаки кирилицы и/или знаки латинского алфавита  \n";
			isValueName = true;
		}
	}
} while (isValueName != false);

do
{
	isValueName = false;
	cout << "Введите отчество: ";
	getline(cin, Patronymic);
	if (Patronymic == "")
	{
		cout << "Строка неможет быть пустой\n";
		isValueName = true;
	}
	else
	{
		if (regex_match(Patronymic, text))
		{
			isValueName = false;
		}
		else
		{
			cout << "ФИО может содерать только знаки кирилицы и/или знаки латинского алфавита  \n";
			isValueName = true;
```

```
            }
        }

} while (isValueName != false);

do
{
        isValueName = false;
        cout << "Введите номер телефона: ";
        getline(cin, ph.NumberPhone);
        if (ph.NumberPhone == "")
        {
                cout << "Строка неможет быть пустой\n";
                isValueName = true;
        }
        else
        {
                if (regex_match(ph.NumberPhone, phone))
                {
                        isValueName = false;
                }
                else
                {
                        cout << "Номер телефона доолен быть в формате +375XXxxxxxxx \n";
                        isValueName = true;
                }
        }

} while (isValueName != false);

do
{
        isValueName = false;
        cout << "Год подключения:\n";
        getline(cin, ph.ConnectYear);
        if (ph.ConnectYear == "")
        {
                cout << "Строка неможет быть пустой\n";
                isValueName = true;
        }
        else
        {
                if (regex_match(ph.ConnectYear, year))
                {
                        isValueName = false;
                }
                else
                {
                        cout << "Год должен быть в формате XXXX \n";
                        isValueName = true;
                }
        }

} while (isValueName != false);



cout << "Текущий тарифный план: \n"
                << "(1):Любимый  (2):Макси  (3):Ультра  (4):Абсолют\n";



        do {

                while (!(cin >> setType) || (cin.peek() != '\n')) {

                        cin.clear();

                        while (cin.get() != '\n');
```

```
                cout << "Ошибка ввода, введен не верный тип данных! Повторите попытку ввода" << endl;
            }

            isValueName = false;
            if (setType >= 1 && setType <= 4)
            {
                isValueName = false;
                switch (setType) {
                case TariffType::Likes:
                    ph.TariffPlan = "Любимый";
                    break;
                case TariffType::Maxi:
                    ph.TariffPlan = "Макси";
                    break;
                case TariffType::Ultra:
                    ph.TariffPlan = "Ультра";
                    break;
                case TariffType::Absolut:
                    ph.TariffPlan = "Абсолют";
                    break;
                }
            }
            else
            {
                isValueName = true;
                cout << "Тарифоного пплана с кодом:" << setType << "не существует\n";
            }

        } while (isValueName != false);



        S1[NewCountSubscribeOwner - 1].SetParametrs(Surname, Name, Patronymic, ph);
        writeInFileInputSubscribeOwner(NewCountSubscribeOwner, nameInputFile, S1);
        cout << "Абонент успешно добавлен\n";
        _getch();
        viewMenu(stateMenu);
}
//изменение данных абонента
void alterData(int countSubcsribe, string nameInputFile, int stateMenu) {
    system("cls");

    string Surname, Name, Patronymic;
    Phone P;
    int altIndex;

    SubscribeOwner *S1 = new SubscribeOwner[countSubcsribe];

    readFromFileSubscribeOwner(S1, nameInputFile);

    cout << "Введите ФИО абонента которого нужно изменить: ";
    cin >> Surname >> Name >> Patronymic;
    for (int i = 0; i < countSubcsribe; i++)
    {
        if (Surname == S1[i].GetSurname())
        {
            if (Name == S1[i].GetName())
            {
                if (Patronymic == S1[i].GetPatronymic())
                {
                    altIndex = i;
                }

            }

        }
    }
```

```
	cout << "Введите новое ФИО абонента:";
	cin >> Surname >> Name >> Patronymic;
	cout << "Новый номер телефона: ";
	cin >> P.NumberPhone;
	cout << "Год подключения: ";
	cin >> P.ConnectYear;
	cout << "Тарифный план: ";
	cin >> P.TariffPlan;

	S1[altIndex].SetParametrs(Surname, Name, Patronymic, P);
	writeInFileInputSubscribeOwner(countSubcsribe, nameInputFile, S1);
	cout << "Данные абонента успешно изменены\n";
	_getch();

	viewMenu(stateMenu);
}
//удаление абонента
void deleteData(int countSubcsribe, string nameInputFile, int stateMenu) {

	string Surname, Name, Patronymic;
	Phone P;
	int delIndex;

	SubscribeOwner *S1 = new SubscribeOwner[countSubcsribe];

	readFromFileSubscribeOwner(S1, nameInputFile);

	cout << "Введите ФИО: ";
	cin >> Surname >> Name >> Patronymic;

	for (int i = 0; i < countSubcsribe; i++)
	{
		if (Surname == S1[i].GetSurname())
		{
			if (Name == S1[i].GetName())
			{
				if (Patronymic == S1[i].GetPatronymic())
				{
					delIndex = i;
				}

			}

		}
	}

	Surname ="", Name = "", Patronymic = "";
	S1[delIndex].SetParametrs(Surname, Name, Patronymic, P);
	writeInFileInputSubscribeOwner(countSubcsribe-1, nameInputFile, S1);

	cout << "Абонент успешно удален\n";
	_getch();

	viewMenu(stateMenu);

}
//вывод списка абонентов
void viewData(int countSubcsribe, string nameInputFile, int stateMenu) {

		countSubcsribe = countFromFile(nameInputFile);
		SubscribeOwner *S = new SubscribeOwner[countSubcsribe];
		readFromFileSubscribeOwner(S, nameInputFile);

		for (int i = 0; i < countSubcsribe; i++)
		{
			cout << "Абонент №" << i + 1 << " :\n";
			S[i].Print();
			cout << "_____\n";
		}
```

```
	_getch();
	viewMenu(stateMenu);
}


//меню режима администратора
void administratorMenu(int st) {
	cout << "МЕНЮ АДМИНИСТРАТОРА\n"
		<< "  (0) Закрыть программу.\n"
		<< "  (1) Перейти к форме входа.\n"
		<< " РАБОТА С ДАННЫМИ ПОЛЬЗОВАТЕЛЕЙ\n"
		<< "  (2) Добовление пользователя\n"
		<< "  (3) Удаление пользователя\n"
		<< "  (4) Изменение данных пользователя\n"
		<< "  (5) Вывод информации\n"
		<< " РАБОТА С ДАННЫМИ АБОНЕНТОВ\n"
		<< "  (6) Добовление данных\n"
		<< "  (7) Удаление данных\n"
		<< "  (8) Изменение данных\n"
		<< "  (9) Вывод информации\n"
		<< "  (10) Сортировка (3 вида)\n"
		<< "  (11) Поиск (3 вида)\n";
	cout << "->"; cin >> st;


	while (st != 0) {
		if (st == 1) {
			system("cls");
			signToApp();
		}
		if (st == 2) {
			addDataUser(countFromFile(InputUserdata), InputUserdata);
		}
		if (st == 3) {
			deleteDataUser(countFromFile(InputUserdata), InputUserdata);
		}
		if (st == 4) {
			alterDataUser(countFromFile(InputUserdata), InputUserdata);
		}
		if (st == 5) {
			viewDataUser(countFromFile(InputUserdata), InputUserdata);
		}
		if (st == 6) {
			addData(countFromFile(InputSubscribeOwner), InputSubscribeOwner,1);
		}
		if (st == 7) {
			deleteData(countFromFile(InputSubscribeOwner), InputSubscribeOwner,1);
		}
		if (st == 8) {
			alterData(countFromFile(InputSubscribeOwner), InputSubscribeOwner,1);
		}
		if (st == 9) {
			viewData(countFromFile(InputSubscribeOwner), InputSubscribeOwner,1);
		}
		if (st == 10) {
			sortBy(countFromFile(InputSubscribeOwner), InputSubscribeOwner,1);
		}
		if (st == 11) {
			searchBy(countFromFile(InputSubscribeOwner), InputSubscribeOwner,1);
		}

	}
	exit(0);
}

//меню режима пользователя
void userMenu(int st) {
	cout << "МЕНЮ\n"
```

```
            << " (0) Закрыть программу.\n"
            << " (1) Перейти к форме входа.\n"
            << " (2) Вывод информации о абонентах\n"
            << " (3) Сортировка (4 вида)\n"
            << " (4) Поиск (3 вида)\n";
        cout << "->"; cin >> st;

        while (st != 0) {
            if (st == 1) {
                system("cls");
                signToApp();
            }
            if (st == 2) {
                viewData(countFromFile(InputSubscribeOwner), InputSubscribeOwner,0);
            }
            if (st == 3) {
                sortBy(countFromFile(InputSubscribeOwner), InputSubscribeOwner,0);
            }
            if (st == 4) {
                searchBy(countFromFile(InputSubscribeOwner), InputSubscribeOwner,0);
            }

        }
        exit(0);
}

// функция переключения режима
void mainMenu(int userMode) {
        system("cls");
        int stateMenu = 0;

        userMode == 0 ? administratorMenu(stateMenu) : userMenu(stateMenu);

        mainMenu(userMode);
}

#pragma endregion

void checkUserPassword(int i, string l,string p, AuthorizationSystem *A) {
        if (l == A[i].GetLogin() && p == A[i].GetPassword())
        {
            cout << " Добро пожаловать в систему " << A[i].GetLogin() << "(" << A[i].GetStatus(A[i].GetType()) << ")" <<
"для продолжения нажимте любую клавишу...\n";
            _getch();
            mainMenu(A[i].GetType());
        }
        else {
            cout << "Произошла ошибка ввода, имя пользователя или пароль не верны!\n";
            cout << "Желаете повторить или выйти? Y/n \n";
            char c = _getch();
            if (c == 'n')
            {
                exit(0);
            }

            _getch();
            system("cls");
            signToApp();
        }
}

//функция авторизации
void signToApp() {

        char c;
        string tLogin = "", tPassword = "";
        int index = 0;

        int NewCountUserData = countFromFile(InputUserdata);
```

```
	AuthorizationSystem *A = new AuthorizationSystem[NewCountUserData];

	readFromFile(A, InputUserdata);

	cout << "Логин: " << "";
	cin >> tLogin; cout << "\n";
	cout << "Пароль: " << "";
	cin >> tPassword; cout << "\n";

	for (int i = 0; i < NewCountUserData; i++)
	{
		if (tLogin == A[i].GetLogin())
		{
			index = i;
		}
	}

	checkUserPassword(index, tLogin,tPassword,A);

}

#pragma endregion

#pragma region DataInizializtion
//функция очистки консоли
void clear() {
	system("cls");
}

//инициализация данных при первом входе
void dataUserInizialization(int countUser, string nameInputFile) {
	cout << "\nПервый вход в программу | Создание администратора" << "\n";
	int count = 0;
	Users us;
	string tmpPassword;
	int NewCountSubscribeOwner = countUser + 1;


	AuthorizationSystem *A1 = new AuthorizationSystem[NewCountSubscribeOwner];

	readFromFile(A1, nameInputFile);

	cout << "Введите логин : ";
	cin >> us.login;
	cout << "Введите пароль: ";
	cin >> us.password;
	cout << "Потвердите пароль: ";
	cin >> tmpPassword;


	if (tmpPassword == us.password)
	{
		cout << "Создание файла учетных записей ... " << "\n";
		A1[NewCountSubscribeOwner - 1].SetParametrs(us.login, us.password, 0);
		writeInFile(NewCountSubscribeOwner, nameInputFile, A1);
	}
	else
	{
		count++;
		cout << "Желаете повторить или выйти? Y/n";
		char c = _getch();
		if (count != 0)
		{
			clear();
		}
		c == 'y' ? dataUserInizialization(countUser, nameInputFile) : c == 'n' ? exit(0) : exit(0);

	}
```

```
	delete[] A1;
}


//инициализация данных если это необходимо с проверкой существования файла пользователей
void dataInitialization(int count, string nameFile, AuthorizationSystem *A) {
	int countAdmin = 0;

	ifstream uout(nameFile);

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

		if (A[i].GetType() != 0)
		{

		}

		if (A[i].GetType() == 0)
		{
			countAdmin++;
		}


	}


	if (uout) {
		cout << "Файл учетных записей существует " << "\n";

		if (countAdmin > 0)
		{
			cout << "Администратор системы существует " << "\n";
		}
		else
		{
			cout << "Администратора системы не существует " << "\n";
			cout << "Переход к системе создания администратора " << "\n";
			dataUserInizialization(count, nameFile);
		}
		uout.close();
	}
	else {

		dataUserInizialization(count, nameFile);

	}
}


//точка входа
int main()
{
	setlocale(LC_ALL, "russian");
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	int CountUserData = countFromFile(InputUserdata);
	AuthorizationSystem *A = new AuthorizationSystem[CountUserData];
	readFromFile(A, InputUserdata);
	dataInitialization(CountUserData, InputUserdata, A);
	system("cls");
	signToApp();
	delete[] A;
	_getch();

}
```