Національний університет “Львівська політехніка”

Кафедра програмного забезпечення

**КУРСОВА РОБОТА**

**з дисципліни «Об’єктно-орієнтоване програмування»**

**На тему: «**Оптимізація бази даних спортивного комплексу**»**

Студента групи ПЗ-23

спеціальності 6.121

“Інженерія програмного забезпечення”

 Михалевича П.-І.

Керівник: доцент кафедри ПЗ,

к.т.н., доцент Коротєєва Т. О.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів\_\_\_Оцінка ECTS\_\_\_

Члени комісії          \_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів 2021

ЗАВДАННЯ

до курсової роботи з дисципліни «Об’єктно-орієнтоване програмування»

студента групи ПЗ-23 Михалевича Павла-Івана

**Тема: «База даних спорткомплексу»**

**Завдання:** автоматизувати процесс формування та опрацювання телефонного довідника.

**Зміст завдання та календарний план його виконання**

|  |  |  |
| --- | --- | --- |
| № з/п | Зміст завдання | Дата |
| 1 | Здійснити аналiтичний огляд лiтератури за заданою темою та обгрунтувати вибір інструментальних засобів реалізації. |  |
| 2 | Побудова UML діаграм |  |
| 3 | Розробка алгоритмів реалізації |  |
| 4 | Реалізація завдання (кодування) |  |
| 5 | Формування інструкції користувача |  |
| 6 | Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури:  ·        зміст;  ·        алгоритм розв‘язку задачі у покроковому представленні;  ·        діаграми UML [клас](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54317&displayformat=dictionary)ів, прецедентів, послідовності виконання;  ·        код розробленої програми з коментарями;  ·        протокол роботи програми для кожного пункту завдання  ·        інструкція користувача та системні вимоги;  ·        опис виняткових ситуацій;  ·        структура файлу вхідних даних;  ·        висновки;  ·        список використаних джерел. |  |

Завдання прийнято до виконання: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Прізвище І.П.)

Керівник роботи: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Прізвище І .П./

**Індивідуальне завдання**

**Варіант - 8**

Створити таблицю в візуальному середовищі

№ | Прізвище | Ім‘я | Телефон | Тип абонементу | Строк дії абонементу

За заданим номером визначити cтрок дії абонементу (врахувати наявність в таблиці однакових прізвищ)

За прізвищем визначити телефон, тип абонементу та його строк дії

За телефоном визначити прізвище та ім’я

Знайти імена та прізвища людей, у яких однаковий тип абонементу.

Визначити тип абонементу який має найбільшу кількість записів

Вивести всі прізвища які мають однакове значення «Строк дії абонементу».

Для [клас](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54317&displayformat=dictionary)у створити: 1) [Конструктор](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54330&displayformat=dictionary) за замовчуванням; 2) [Конструктор](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54330&displayformat=dictionary) з параметрами; 3) [конструктор](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54330&displayformat=dictionary) копій; 4) перевизначити операції >>, << для зчитування та запису у файл.

**Зміст**

1. Алгоритми розв’язку задачі у покроковому представленні. ()

2. Діаграми UML [клас](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54317&displayformat=dictionary)ів, прецедентів, послідовності виконання. ()

3. Код розробленої програми. ()

4. Протокол роботи програми. ()

5. Інструкція користувача. ()

6. Опис виняткових ситуацій. ()

7. Структура файлу вхідних даних. ()

**1. Алгоритми розв’язку задачі у покроковому представленні**

Покроковий алгоритм роботи програми:

K1. Зчитування даних з файлу, або з клавіатури.

К2. Обробка даних і виведення на екран.

К3. Вивід на екран потрібної користувачу інформації.

К4. Кінець.

Покроковий алгоритм пошуку інформації за заданим номером, прізвищем або номером телефону:

L1. Дано A = {x1,…, xn} – масив структур з потрібним полем Х. Цикл за індексом проходження i = 1..n. Y - елемент пошуку.

L2. Якщо A[i].X = Y, виводимо його на екран.

L3. Кінець. Вихід.

Покроковий алгоритм пошуку даних з найбільш використовуваним типом або сроком дії:

M1. Дано A = {x1,…, xn} – масив структур з потрібним полем. Цикл за індексом проходження i = 1..n.

М2. Обчислюєм кількість використання різних за значенням потрібних елементів.

М3. Знаходимо елемент з нацбільшою кількістю використання. Виводимо на екран.

М4. Кінець.

**2. Діаграми UML**

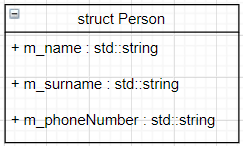
****

Рис 2.1. UML діаграма структури Person

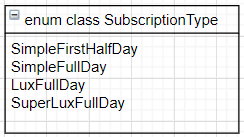
****

Рис 2.1. UML діаграма enum класу SubscriptionType

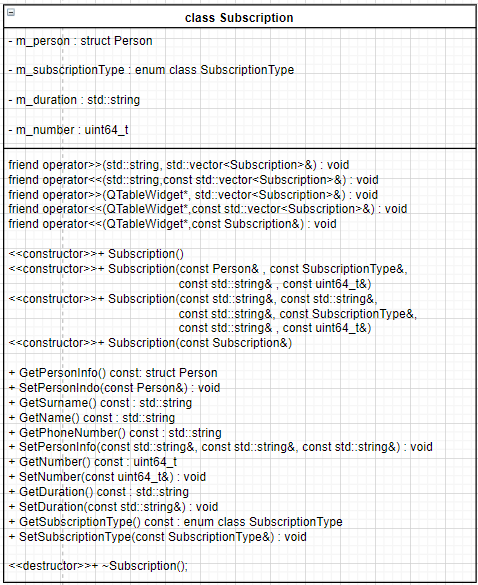


Рис 2.3. UML діаграма класу Subscription

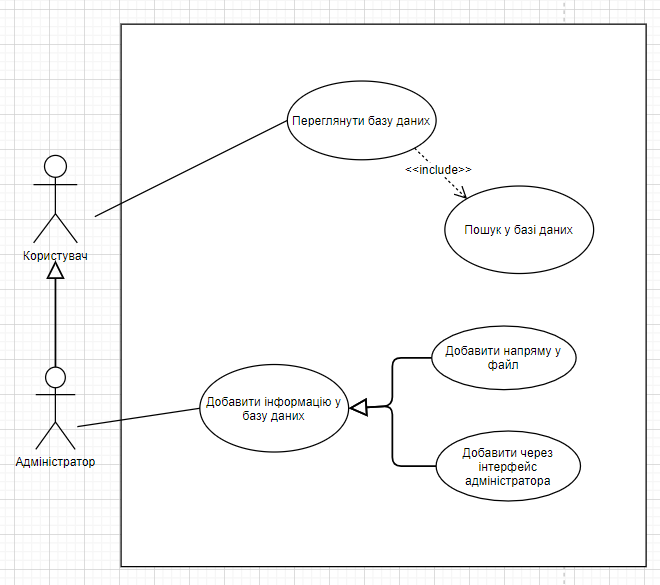


Рис 2.4. Діаграма прецендентів

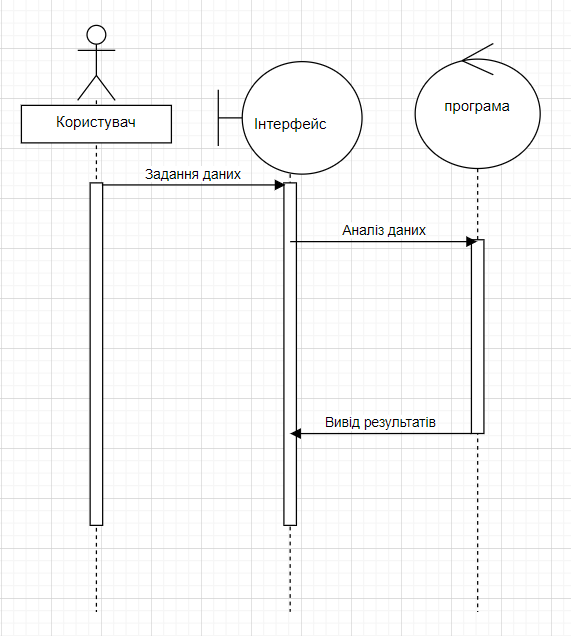


Рис 2.5. Діаграма послідовності виконяння

**3. Код розробленої програми**

**Function.h**

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

#include <string>

#include <vector>

#include <QTableWidget>

#include "subscription.h"

#include "mainwindow.h"

void **RemoveSpaces**(std::string& myString);

void **ShowSameType**(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect,*const* *enum* SubscriptionType type);

void **ShowSameDuration**(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect);

void **ShowMostType**(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect);

void **FuncForResizeEvent**(QTableWidget\* tableWidget, MainWindow \* mainWindow);

void **SetTableWidget**(QTableWidget\* tableWidget, MainWindow \* mainWindow);

#endif *//* *FUNCTIONS\_H*

**Function.cpp**

#include "Functions.h"

#include <algorithm>

#include <map>

#include <string>

#include <QMessageBox>

*//* *Removes* *unnecessary* *spaces.*

void RemoveSpaces(std::string& myString){

*for* (size\_t i = 1; i <= myString.size(); ++i) {

*if*(i-1 == 0 && myString[i - 1] == ' '){

myString.erase(myString.begin() + (i - 1));

myString.shrink\_to\_fit();

i--;

*continue*;

}

*if* (myString[i - 1] == ' ' && (myString[i] == ' ' || myString[i] == '\0')) {

myString.erase(myString.begin() + (i - 1));

myString.shrink\_to\_fit();

i--;

}

}

}

*//* *Show* *person* *with* *the* *subscription* *type* *you* *pick.*

void ShowSameType(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect, *const* *enum* SubscriptionType type){

*if*(mySubscriptionVect.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

*for* (*auto* i = 0; i < tableWidget->rowCount(); i++) {

*for* (*auto* j = 0; j < tableWidget->columnCount(); j++) {

tableWidget->item(i,j)->setText("");

}

}

int count = 0;

*for*(*const* *auto*& subscription : mySubscriptionVect){

*if*(subscription.GetSubscriptionType() == type){

tableWidget << subscription;

++count;

}

}

*if*(count == 0){

QMessageBox::information(NULL,"Info","There is no person with such subscription.");

}

}

*//* *Show* *people* *who* *have* *the* *most* *used* *duration.*

void ShowSameDuration(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect){

*if*(mySubscriptionVect.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

std::map<std::string,int> myMap;

*for*(*const* *auto*& subscription : mySubscriptionVect){

myMap[subscription.GetDuration()]++;

}

std::map<std::string,int>::iterator maxIter =

std::max\_element(myMap.begin(),myMap.end(),[](*const* std::pair<std::string,int>& a, *const* std::pair<std::string,int>& b)->bool

{

*return* a.second < b.second;

});

*for* (*auto* i = 0; i < tableWidget->rowCount(); i++) {

*for* (*auto* j = 0; j < tableWidget->columnCount(); j++) {

tableWidget->item(i,j)->setText("");

}

}

*for*(*const* *auto*& subscription : mySubscriptionVect){

*if*(subscription.GetDuration() == maxIter->first){

tableWidget << subscription;

}

}

}

*//* *Show* *people* *who* *have* *the* *most* *used* *subscription* *type.*

void ShowMostType(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect){

*if*(mySubscriptionVect.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

int count\_1 = 0;

int count\_2 = 0;

int count\_3 = 0;

int count\_4 = 0;

*for*(*const* *auto*& subscription : mySubscriptionVect){

*switch*(subscription.GetSubscriptionType()){

*case* SubscriptionType::SimpleFirstHalfDay:{

++count\_1;

*break*;

}

*case* SubscriptionType::SimpleFullDay:{

++count\_2;

*break*;

}

*case* SubscriptionType::LuxFullDay:{

++count\_3;

*break*;

}

*case* SubscriptionType::SuperLuxFullDay:{

++count\_4;

*break*;

}

}

}

*const* *auto* maxValue = std::max({count\_1,count\_2,count\_3,count\_4});

SubscriptionType type;

*if*(maxValue == count\_1){

type = SubscriptionType::SimpleFirstHalfDay;

}*else* *if*(maxValue == count\_2){

type = SubscriptionType::SimpleFullDay;

}*else* *if*(maxValue == count\_3){

type = SubscriptionType::LuxFullDay;

}*else* *if*(maxValue == count\_4){

type = SubscriptionType::SuperLuxFullDay;

}

*for* (*auto* i = 0; i < tableWidget->rowCount(); i++) {

*for* (*auto* j = 0; j < tableWidget->columnCount(); j++) {

tableWidget->item(i,j)->setText("");

}

}

*for*(*const* *auto*& subscription : mySubscriptionVect){

*if*(subscription.GetSubscriptionType() == type){

tableWidget << subscription;

}

}

}

*//* *Function* *for* *avoiding* *duplication.*

void FuncForResizeEvent(QTableWidget\* tableWidget, MainWindow \* mainWindow){

int widthToSet = (mainWindow->width() - 340) / 3;

*if*(widthToSet < 150){

widthToSet = 150;

}

tableWidget->setColumnWidth(1,widthToSet);

tableWidget->setColumnWidth(2,widthToSet);

tableWidget->setColumnWidth(4,widthToSet);

}

*//* *Function* *for* *avoiding* *duplication.*

void SetTableWidget(QTableWidget\* tableWidget, MainWindow \* mainWindow){

tableWidget->setColumnWidth(0,50);

int widthToSet = (mainWindow->width() - 340) / 3;

*if*(widthToSet < 150){

widthToSet = 150;

}

tableWidget->setColumnWidth(1,widthToSet);

tableWidget->setColumnWidth(2,widthToSet);

tableWidget->setColumnWidth(3,150);

tableWidget->setColumnWidth(4,widthToSet);

tableWidget->setColumnWidth(5,140-3);

*for* (*auto* i = 0; i < tableWidget->rowCount(); i++) {

*for* (*auto* j = 0; j < tableWidget->columnCount(); j++) {

QTableWidgetItem\* item = *new* QTableWidgetItem();

item->setText("");

item->setTextAlignment(Qt::AlignCenter);

tableWidget->setItem(i,j,item);

}

}

}

**SubscriptionType.h**

#ifndef SUBSCRIPTIONTYPE\_H

#define SUBSCRIPTIONTYPE\_H

*enum* *class* **SubscriptionType**

{

*SimpleFirstHalfDay*,

*SimpleFullDay*,

*LuxFullDay*,

*SuperLuxFullDay*

};

#endif *//* *SUBSCRIPTIONTYPE\_H*

**Person.h**

#ifndef PERSON\_H

#define PERSON\_H

#include <string>

*struct* **Person**

{

std::string m\_name;

std::string m\_surname;

std::string m\_phoneNumber;

};

#endif *//* *PERSON\_H*

**Subscription.h**

#ifndef SUBSCRIPTION\_H

#define SUBSCRIPTION\_H

#include <vector>

#include <QTableWidget>

#include "Person.h"

#include "SubscriptionType.h"

*class* **Subscription**

{

*public*:

*friend* void *operator*>>(std::string filePath,std::vector<Subscription>& mySubscription);

*friend* void *operator*<<(std::string filePath,*const* std::vector<Subscription>& mySubscription);

*friend* void *operator*>>(QTableWidget\* tableWidget,std::vector<Subscription>& mySubscription);

*friend* void *operator*<<(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscription);

*friend* void *operator*<<(QTableWidget\* tableWidget,*const* Subscription& mySubscription);

**Subscription**();

**Subscription**(*const* Person& person, *const* SubscriptionType& type, *const* std::string& duration, *const* uint64\_t& number);

**Subscription**(*const* std::string& surname, *const* std::string& name, *const* std::string& phoneNumber,

*const* SubscriptionType& type, *const* std::string& duration, *const* uint64\_t& number);

**Subscription**(*const* Subscription& otherSubscription);

Person **GetPersonInfo**() *const*;

void **SetPersonInfo**(*const* Person& person);

std::string **GetSurname**() *const*;

std::string **GetName**() *const*;

std::string **GetPhoneNumber**() *const*;

void **SetPersonInfo**(*const* std::string& surname, *const* std::string& name, *const* std::string& phoneNumber);

uint64\_t **GetNumber**() *const*;

void **SetNumber**(*const* uint64\_t& newNumber);

std::string **GetDuration**() *const*;

void **SetDuration**(*const* std::string& newDuration);

SubscriptionType **GetSubscriptionType**() *const*;

void **SetSubscriptionType**(*const* SubscriptionType& newSubscriptionType);

~**Subscription**();

*private*:

Person m\_person;

SubscriptionType m\_subscriptionType;

std::string m\_duration;

uint64\_t m\_number;

};

#endif *//* *SUBSCRIPTION\_H*

**Subscription.cpp**

#include <subscription.h>

#include <fstream>

#include <iostream>

#include <iomanip>

#include <vector>

#include <QString>

#include <QStringList>

#include <QMessageBox>

#include "Functions.h"

Subscription::Subscription()

{

*this*->SetDuration("");

*this*->SetNumber(std::numeric\_limits<uint64\_t>::max());

*this*->SetPersonInfo("", "", "");

*this*->SetSubscriptionType(SubscriptionType::SimpleFirstHalfDay);

}

Subscription::Subscription(*const* Person& person, *const* SubscriptionType& type, *const* std::string& duration, *const* uint64\_t& number)

{

*this*->SetDuration(duration);

*this*->SetNumber(number);

*this*->SetPersonInfo(person);

*this*->SetSubscriptionType(type);

}

Subscription::Subscription(*const* std::string& surname, *const* std::string& name, *const* std::string& phoneNumber,

*const* SubscriptionType& type, *const* std::string& duration, *const* uint64\_t& number)

{

*this*->SetDuration(duration);

*this*->SetNumber(number);

*this*->SetPersonInfo(surname, name, phoneNumber);

*this*->SetSubscriptionType(type);

}

Subscription::Subscription(*const* Subscription& otherSubscription)

{

*this*->SetDuration(otherSubscription.m\_duration);

*this*->SetNumber(otherSubscription.m\_number);

*this*->SetPersonInfo(otherSubscription.m\_person.m\_surname,

otherSubscription.m\_person.m\_name,

otherSubscription.m\_person.m\_phoneNumber);

*this*->SetSubscriptionType(otherSubscription.m\_subscriptionType);

}

Person Subscription::GetPersonInfo() *const*

{

*return* *this*->m\_person;

}

void Subscription::SetPersonInfo(*const* std::string& surname, *const* std::string& name, *const* std::string& phoneNumber)

{

*this*->m\_person.m\_surname = surname;

*this*->m\_person.m\_name = name;

*this*->m\_person.m\_phoneNumber = phoneNumber;

}

void Subscription::SetPersonInfo(*const* Person &person)

{

*this*->m\_person.m\_surname = person.m\_surname;

*this*->m\_person.m\_name = person.m\_name;

*this*->m\_person.m\_phoneNumber = person.m\_phoneNumber;

}

std::string Subscription::GetSurname() *const*

{

*return* m\_person.m\_surname;

}

std::string Subscription::GetName() *const*

{

*return* m\_person.m\_name;

}

std::string Subscription::GetPhoneNumber() *const*

{

*return* m\_person.m\_phoneNumber;

}

uint64\_t Subscription::GetNumber() *const*

{

*return* *this*->m\_number;

}

void Subscription::SetNumber(*const* uint64\_t& newNumber)

{

*this*->m\_number = newNumber;

}

std::string Subscription::GetDuration() *const*

{

*return* *this*->m\_duration;

}

void Subscription::SetDuration(*const* std::string& newDuration)

{

*this*->m\_duration = newDuration;

}

SubscriptionType Subscription::GetSubscriptionType() *const*

{

*return* *this*->m\_subscriptionType;

}

void Subscription::SetSubscriptionType(*const* SubscriptionType& newSubscriptionType)

{

*this*->m\_subscriptionType = newSubscriptionType;

}

Subscription::~Subscription()

{

}

*//* *Fill* *vector* *of* *Subscription* *from* *file.*

void *operator*>>(std::string filePath,std::vector<Subscription>& mySubscriptionVect){

std::ifstream file;

file.open(filePath);

*while*(!file.eof()){

Subscription mySubscription;

Person person;

std::string subscriptionType;

std::string line;

std::getline(file, line);

*if*(line == ""){

*continue*;

}

std::stringstream strstream(line);

std::vector<std::string> tokenVect;

std::string token;

*while*(std::getline(strstream, token, '|')){

RemoveSpaces(token);

tokenVect.push\_back(token);

}

*if*(tokenVect.size() != 5){

*throw* std::runtime\_error("The input data isn't correct!");

*return*;

}

*if*(!tokenVect.empty()){

*if*(tokenVect[4].size() > 10 || tokenVect[4].size() < 8 || tokenVect[4].size() == 9){

*throw* std::runtime\_error("The duration is incorrect!");

*return*;

}

*if*(tokenVect[4].size() == 8 && tokenVect[4].find(".") == std::string::npos){

*throw* std::runtime\_error("Please, write the duration in format: dd.mm.yyyy");

*return*;

}

mySubscription.SetDuration(tokenVect[4]);

*if*(tokenVect[0].size() > 35 || tokenVect[0].size() < 3){

*throw* std::runtime\_error("The surname is incorrect!");

*return*;

}

person.m\_surname = tokenVect[0];

*if*(tokenVect[1].size() > 35 || tokenVect[1].size() < 3){

*throw* std::runtime\_error("The name is incorrect!");

*return*;

}

person.m\_name = tokenVect[1];

*if*(tokenVect[2].size() > 12 || tokenVect[2].size() < 10 || tokenVect[2].size() == 11){

*throw* std::runtime\_error("The phone number is incorrect!");

*return*;

}

*if*(tokenVect[2].size() == 10){

*throw* std::runtime\_error("Please, start the phone number with 380...");

*return*;

}

bool ok;

QString::fromStdString(tokenVect[2]).toULongLong(&ok);

*if*(!ok){

*throw* std::runtime\_error("The phone number is incorrect!");

*return*;

}

person.m\_phoneNumber = tokenVect[2];

mySubscription.SetPersonInfo(person);

*if*(tokenVect[3].empty()){

*throw* std::runtime\_error("There is no subscription type!");

*return*;

}

subscriptionType = QString::fromStdString(tokenVect[3]).toLower().toStdString();

*if*(subscriptionType.find("lux") != std::string::npos &&

subscriptionType.find("full") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::LuxFullDay);

}*else* *if*(subscriptionType.find("simple") != std::string::npos &&

subscriptionType.find("half") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::SimpleFirstHalfDay);

}*else* *if*(subscriptionType.find("simple") != std::string::npos &&

subscriptionType.find("full") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::SimpleFullDay);

}*else* *if*(subscriptionType.find("super") != std::string::npos &&

subscriptionType.find("lux") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::SuperLuxFullDay);

}*else*{

*throw* std::runtime\_error("There isn't such a subscription type!");

*return*;

}

}*else*{

*throw* std::runtime\_error("The input file is empty!");

*return*;

}

*if*(!mySubscriptionVect.empty()){

mySubscription.SetNumber(mySubscriptionVect.back().GetNumber() + 1);

}*else*{

mySubscription.SetNumber(1);

}

mySubscriptionVect.push\_back(mySubscription);

}

file.close();

}

*//* *Write* *vector* *of* *Subscription* *to* *file.*

void *operator*<<(std::string filePath,*const* std::vector<Subscription>& mySubscriptionVect){

*if*(mySubscriptionVect.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

std::ofstream file;

file.open(filePath);

std::string myType;

*for*(*const* *auto*& mySubscription : mySubscriptionVect){

*switch*(mySubscription.m\_subscriptionType){

*case* SubscriptionType::LuxFullDay:{

myType = "Lux full day";

*break*;

}

*case* SubscriptionType::SimpleFirstHalfDay:{

myType = "Simple first half day";

*break*;

}

*case* SubscriptionType::SimpleFullDay:{

myType = "Simple full day";

*break*;

}

*case* SubscriptionType::SuperLuxFullDay:{

myType = "Super Lux";

*break*;

}

}

file << std::setw(5) << std::left << mySubscription.m\_number << std::setw(1) << "|"

<< std::setw(35) << mySubscription.m\_person.m\_surname << std::setw(1) << "|"

<< std::setw(35) << mySubscription.m\_person.m\_name << std::setw(1) << "|"

<< std::setw(15) << mySubscription.m\_person.m\_phoneNumber << std::setw(1) << "|"

<< std::setw(30) << myType<< std::setw(1) << "|"

<< std::setw(15) << mySubscription.m\_duration << std::setw(1) << "|" << std::endl;

}

file.close();

}

*//* *Fill* *vector* *of* *Subscription* *from* *QTableWidget.*

void *operator*>>(QTableWidget\* tableWidget,std::vector<Subscription>& mySubscriptionVect){

Subscription mySubscription;

Person person;

std::string subscriptionType;

std::vector<std::string> tokenVect{5};

*for*(*auto* i = 0; i < 5; ++i){

tokenVect[i] = tableWidget->item(0,i)->text().toStdString();

RemoveSpaces(tokenVect[i]);

}

*if*(tokenVect[4].size() > 10 || tokenVect[4].size() < 8 || tokenVect[4].size() == 9){

*throw* std::runtime\_error("The duration is incorrect!");

*return*;

}

*if*(tokenVect[4].size() == 8 || tokenVect[4].find(".") == std::string::npos ||

tokenVect[4][2] != '.' || tokenVect[4][5] != '.'){

*throw* std::runtime\_error("Please, write the duration in format: dd.mm.yyyy");

*return*;

}

mySubscription.SetDuration(tokenVect[4]);

*if*(tokenVect[0].size() > 35 || tokenVect[0].size() < 3){

*throw* std::runtime\_error("The surname is incorrect!");

*return*;

}

person.m\_surname = tokenVect[0];

*if*(tokenVect[1].size() > 35 || tokenVect[1].size() < 3){

*throw* std::runtime\_error("The name is incorrect!");

*return*;

}

person.m\_name = tokenVect[1];

*if*(tokenVect[2].size() > 12 || tokenVect[2].size() < 10 || tokenVect[2].size() == 11){

*throw* std::runtime\_error("The phone number is incorrect!");

*return*;

}

*if*(tokenVect[2].size() == 10){

*throw* std::runtime\_error("Please, start the phone number with 380...");

*return*;

}

bool ok;

QString::fromStdString(tokenVect[2]).toULongLong(&ok);

*if*(!ok){

*throw* std::runtime\_error("The phone number is incorrect! \n Write only numbers!");

*return*;

}

person.m\_phoneNumber = tokenVect[2];

mySubscription.SetPersonInfo(person);

*if*(tokenVect[3].empty()){

*throw* std::runtime\_error("There is no subscription type!");

*return*;

}

subscriptionType = QString::fromStdString(tokenVect[3]).toLower().toStdString();

*if*(subscriptionType.find("lux") != std::string::npos &&

subscriptionType.find("full") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::LuxFullDay);

}*else* *if*(subscriptionType.find("simple") != std::string::npos &&

subscriptionType.find("half") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::SimpleFirstHalfDay);

}*else* *if*(subscriptionType.find("simple") != std::string::npos &&

subscriptionType.find("full") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::SimpleFullDay);

}*else* *if*(subscriptionType.find("super") != std::string::npos &&

subscriptionType.find("lux") != std::string::npos){

mySubscription.SetSubscriptionType(SubscriptionType::SuperLuxFullDay);

}*else*{

*throw* std::runtime\_error("There isn't such a subscription type!");

*return*;

}

*if*(!mySubscriptionVect.empty()){

mySubscription.SetNumber(mySubscriptionVect.back().GetNumber() + 1);

}*else*{

mySubscription.SetNumber(1);

}

mySubscriptionVect.push\_back(mySubscription);

}

*//* *Write* *vector* *of* *Subscription* *to* *QTableWidget.*

void *operator*<<(QTableWidget\* tableWidget,*const* std::vector<Subscription>& mySubscriptionVect){

*if*(mySubscriptionVect.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

std::string myType;

tableWidget->setRowCount(mySubscriptionVect.size()+1);

*for* (*auto* i = 0; i < tableWidget->rowCount(); i++) {

*for* (*auto* j = 0; j < tableWidget->columnCount(); j++) {

QTableWidgetItem\* item = *new* QTableWidgetItem();

item->setText("");

item->setTextAlignment(Qt::AlignCenter);

tableWidget->setItem(i,j,item);

}

}

*for*(size\_t i = 0; i < mySubscriptionVect.size(); ++i){

*switch*(mySubscriptionVect[i].GetSubscriptionType()){

*case* SubscriptionType::LuxFullDay:{

myType = "Lux full day";

*break*;

}

*case* SubscriptionType::SimpleFirstHalfDay:{

myType = "Simple first half day";

*break*;

}

*case* SubscriptionType::SimpleFullDay:{

myType = "Simple full day";

*break*;

}

*case* SubscriptionType::SuperLuxFullDay:{

myType = "Super Lux";

*break*;

}

}

tableWidget->item(i,0)->setText(QString::number(mySubscriptionVect[i].GetNumber()));

tableWidget->item(i,1)->setText(QString::fromStdString(mySubscriptionVect[i].GetSurname()));

tableWidget->item(i,2)->setText(QString::fromStdString(mySubscriptionVect[i].GetName()));

tableWidget->item(i,3)->setText(QString::fromStdString(mySubscriptionVect[i].GetPhoneNumber()));

tableWidget->item(i,4)->setText(QString::fromStdString(myType));

tableWidget->item(i,5)->setText(QString::fromStdString(mySubscriptionVect[i].GetDuration()));

}

}

*//* *Write* *Subscription* *object* *to* *the* *end* *of* *QTableWidget.*

void *operator*<<(QTableWidget\* tableWidget,*const* Subscription& mySubscription){

std::string myType;

tableWidget->insertRow( tableWidget->rowCount() );

*for* (*auto* j = 0; j < tableWidget->columnCount(); j++) {

QTableWidgetItem\* item\_2 = *new* QTableWidgetItem();

item\_2->setText("");

item\_2->setTextAlignment(Qt::AlignCenter);

tableWidget->setItem(tableWidget->rowCount()-1,j,item\_2);

}

*switch*(mySubscription.GetSubscriptionType()){

*case* SubscriptionType::LuxFullDay:{

myType = "Lux full day";

*break*;

}

*case* SubscriptionType::SimpleFirstHalfDay:{

myType = "Simple first half day";

*break*;

}

*case* SubscriptionType::SimpleFullDay:{

myType = "Simple full day";

*break*;

}

*case* SubscriptionType::SuperLuxFullDay:{

myType = "Super Lux";

*break*;

}

}

tableWidget->item(tableWidget->rowCount()-2,0)->setText(QString::number(mySubscription.GetNumber()));

tableWidget->item(tableWidget->rowCount()-2,1)->setText(QString::fromStdString(mySubscription.GetSurname()));

tableWidget->item(tableWidget->rowCount()-2,2)->setText(QString::fromStdString(mySubscription.GetName()));

tableWidget->item(tableWidget->rowCount()-2,3)->setText(QString::fromStdString(mySubscription.GetPhoneNumber()));

tableWidget->item(tableWidget->rowCount()-2,4)->setText(QString::fromStdString(myType));

tableWidget->item(tableWidget->rowCount()-2,5)->setText(QString::fromStdString(mySubscription.GetDuration()));

}

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "subscription.h"

QT\_BEGIN\_NAMESPACE

*namespace* Ui { *class* MainWindow; }

QT\_END\_NAMESPACE

*class* MainWindow : *public* QMainWindow

{

Q\_OBJECT

*public*:

MainWindow(QWidget \*parent = *nullptr*);

~MainWindow();

*private* *slots*:

void on\_actionEdit\_table\_triggered();

void on\_actionBack\_to\_table\_triggered();

void resizeEvent(QResizeEvent\*);

void on\_actionGive\_number\_Get\_duration\_triggered();

void on\_actionGive\_surname\_triggered();

void on\_actionGive\_phone\_number\_triggered();

void on\_actionShow\_most\_used\_type\_triggered();

void on\_actionShow\_with\_same\_duration\_triggered();

void on\_pushButton\_num\_clicked();

void on\_pushButton\_surname\_clicked();

void on\_pushButton\_ph\_num\_clicked();

void on\_actionSimple\_Full\_Day\_triggered();

void on\_actionLux\_Full\_Day\_triggered();

void on\_actionSuper\_Lux\_Full\_Day\_triggered();

void on\_actionSimple\_First\_Half\_Day\_triggered();

void on\_actionRead\_from\_file\_triggered();

void on\_actionWrite\_to\_file\_triggered();

void on\_actionAdd\_triggered();

void BackToTableOff();

*private*:

Ui::MainWindow \*ui;

std::vector<Subscription> SubscriptionVector;

};

#endif *//* *MAINWINDOW\_H*

**Mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "QStringList"

#include <QMessageBox>

#include "Functions.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

*//* *Setting* *UI*

setCentralWidget(*ui->stackedWidget*);

ui->stackedWidget->insertWidget(0,*ui->tableWidget*);

ui->stackedWidget->insertWidget(1,*ui->tableWidget\_2*);

ui->stackedWidget->insertWidget(2,*ui->tableWidget\_same\_type*);

ui->stackedWidget->insertWidget(3,*ui->tableWidget\_most\_type*);

ui->stackedWidget->insertWidget(4,*ui->tableWidget\_sameduration*);

ui->page->setLayout(*ui->verticalLayout\_number*);

ui->page\_2->setLayout(*ui->verticalLayout\_surname*);

ui->page\_3->setLayout(*ui->verticalLayout\_ph\_num*);

ui->tableWidget->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_2->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_number->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_surname->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_ph\_num->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_same\_type->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_most\_type->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget\_sameduration->horizontalHeader()->*setVisible*(*true*);

ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_2->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_number->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_surname->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_ph\_num->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_same\_type->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_most\_type->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->tableWidget\_sameduration->horizontalHeader()->setSectionResizeMode(QHeaderView::*Fixed*);

ui->stackedWidget->setCurrentIndex(0);

SetTableWidget(*ui->tableWidget*,*this*);

int widthToSet\_2 = (*this*->width() - 290) / 3;

*if*(widthToSet\_2 < 150){

widthToSet\_2 = 150;

}

ui->tableWidget\_2->setColumnWidth(0,widthToSet\_2);

ui->tableWidget\_2->setColumnWidth(1,widthToSet\_2);

ui->tableWidget\_2->setColumnWidth(2,150);

ui->tableWidget\_2->setColumnWidth(3,widthToSet\_2);

ui->tableWidget\_2->setColumnWidth(4,140-3);

*for* (*auto* i = 0; i < ui->tableWidget\_2->rowCount(); i++) {

*for* (*auto* j = 0; j < ui->tableWidget\_2->columnCount(); j++) {

QTableWidgetItem\* item = *new* QTableWidgetItem();

item->setText("");

item->setTextAlignment(Qt::*AlignCenter*);

ui->tableWidget\_2->setItem(i,j,*item*);

}

}

SetTableWidget(*ui->tableWidget\_number*,*this*);

SetTableWidget(*ui->tableWidget\_surname*,*this*);

SetTableWidget(*ui->tableWidget\_ph\_num*,*this*);

SetTableWidget(*ui->tableWidget\_sameduration*,*this*);

SetTableWidget(*ui->tableWidget\_most\_type*,*this*);

SetTableWidget(*ui->tableWidget\_same\_type*,*this*);

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

*//* *Open* *the* *widget* *with* *editable* *table.*

void MainWindow::**on\_actionEdit\_table\_triggered**()

{

ui->stackedWidget->setCurrentIndex(1);

ui->actionAdd->setVisible(*true*);

ui->actionAdd->setEnabled(*true*);

ui->actionBack\_to\_table->setVisible(*true*);

ui->actionBack\_to\_table->setEnabled(*true*);

ui->actionEdit\_table->setVisible(*false*);

ui->actionEdit\_table->setEnabled(*false*);

ui->actionRead\_from\_file->setVisible(*false*);

ui->actionRead\_from\_file->setEnabled(*false*);

ui->actionWrite\_to\_file->setVisible(*false*);

ui->actionWrite\_to\_file->setEnabled(*false*);

}

*//* *Go* *back* *to* *table.*

void MainWindow::**on\_actionBack\_to\_table\_triggered**()

{

ui->stackedWidget->setCurrentIndex(0);

ui->actionAdd->setVisible(*false*);

ui->actionAdd->setEnabled(*false*);

ui->actionBack\_to\_table->setVisible(*false*);

ui->actionBack\_to\_table->setEnabled(*false*);

ui->actionEdit\_table->setVisible(*true*);

ui->actionEdit\_table->setEnabled(*true*);

ui->actionRead\_from\_file->setVisible(*true*);

ui->actionRead\_from\_file->setEnabled(*true*);

ui->actionWrite\_to\_file->setVisible(*true*);

ui->actionWrite\_to\_file->setEnabled(*true*);

}

*//* *Event* *that* *resize* *the* *table.*

void MainWindow::***resizeEvent***(QResizeEvent \*)

{

int widthToSet\_2 = (*this*->width() - 290) / 3;

*if*(widthToSet\_2 < 150){

widthToSet\_2 = 150;

}

ui->tableWidget\_2->setColumnWidth(0,widthToSet\_2);

ui->tableWidget\_2->setColumnWidth(1,widthToSet\_2);

ui->tableWidget\_2->setColumnWidth(3,widthToSet\_2);

FuncForResizeEvent(*ui->tableWidget*,*this*);

FuncForResizeEvent(*ui->tableWidget\_most\_type*,*this*);

FuncForResizeEvent(*ui->tableWidget\_number*,*this*);

FuncForResizeEvent(*ui->tableWidget\_ph\_num*,*this*);

FuncForResizeEvent(*ui->tableWidget\_same\_type*,*this*);

FuncForResizeEvent(*ui->tableWidget\_sameduration*,*this*);

FuncForResizeEvent(*ui->tableWidget\_surname*,*this*);

}

*//* *Open* *the* *widget* *where* *you* *give* *the* *number* *and* *get* *the* *info* *about* *person* *with* *such* *number.*

void MainWindow::**on\_actionGive\_number\_Get\_duration\_triggered**()

{

ui->stackedWidget->setCurrentIndex(5);

BackToTableOff();

}

*//* *Open* *the* *widget* *where* *you* *give* *the* *surname* *and* *get* *the* *info* *about* *person* *with* *such* *surname.*

void MainWindow::**on\_actionGive\_surname\_triggered**()

{

ui->stackedWidget->setCurrentIndex(6);

BackToTableOff();

}

*//* *Open* *the* *widget* *where* *you* *give* *the* *phone* *number* *and* *get* *the* *info* *about* *person* *with* *such* *phone* *number.*

void MainWindow::**on\_actionGive\_phone\_number\_triggered**()

{

ui->stackedWidget->setCurrentIndex(7);

BackToTableOff();

}

*//* *Show* *people* *who* *have* *the* *most* *used* *subscription* *type.*

void MainWindow::**on\_actionShow\_most\_used\_type\_triggered**()

{

*try*{

ui->tableWidget\_most\_type->setRowCount(1);

ui->stackedWidget->setCurrentIndex(3);

BackToTableOff();

ShowMostType(*ui->tableWidget\_most\_type*, SubscriptionVector);

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *people* *who* *have* *the* *most* *used* *duration.*

void MainWindow::**on\_actionShow\_with\_same\_duration\_triggered**()

{

*try*{

ui->tableWidget\_sameduration->setRowCount(1);

ui->stackedWidget->setCurrentIndex(4);

BackToTableOff();

ShowSameDuration(*ui->tableWidget\_sameduration*, SubscriptionVector);

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *person* *with* *the* *number* *you* *give.*

void MainWindow::**on\_pushButton\_num\_clicked**()

{

*try*{

*if*(SubscriptionVector.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

ui->tableWidget\_number->setRowCount(1);

*for* (*auto* i = 0; i < ui->tableWidget\_number->rowCount(); i++) {

*for* (*auto* j = 0; j < ui->tableWidget\_number->columnCount(); j++) {

ui->tableWidget\_number->item(i,j)->setText("");

}

}

bool ok;

*const* *auto* number = ui->lineEdit\_number->text().toUInt(*&ok*);

*if*(!ok){

QMessageBox::warning(NULL,"Wrong input","Please, enter the positive integer number.");

*return*;

}

int count = 0;

*for*(*const* *auto*& subscription : SubscriptionVector){

*if*(subscription.GetNumber() == number){

ui->tableWidget\_number << subscription;

++count;

}

}

*if*(count == 0){

QMessageBox::information(NULL,"Info","There is no person with such number.");

}

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *people* *with* *the* *surname* *you* *give.*

void MainWindow::**on\_pushButton\_surname\_clicked**()

{

*try*{

*if*(SubscriptionVector.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

ui->tableWidget\_surname->setRowCount(1);

*for* (*auto* i = 0; i < ui->tableWidget\_surname->rowCount(); i++) {

*for* (*auto* j = 0; j < ui->tableWidget\_surname->columnCount(); j++) {

ui->tableWidget\_surname->item(i,j)->setText("");

}

}

*auto* surname = ui->lineEdit\_surname->text().toStdString();

RemoveSpaces(*surname*);

*if*(surname.size() > 35 || surname.size() < 3){

QMessageBox::warning(NULL,"Wrong input","Please, enter the right surname.");

*return*;

}

int count = 0;

*for*(*const* *auto*& subscription : SubscriptionVector){

*if*(subscription.GetSurname() == surname){

ui->tableWidget\_surname << subscription;

++count;

}

}

*if*(count == 0){

QMessageBox::information(NULL,"Info","There is no person with such surname.");

}

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *people* *with* *the* *phone* *number* *you* *give.*

void MainWindow::**on\_pushButton\_ph\_num\_clicked**()

{

*try*{

*if*(SubscriptionVector.empty()){

QMessageBox::information(NULL,"Info","There is no data.");

*return*;

}

ui->tableWidget\_ph\_num->setRowCount(1);

*for* (*auto* i = 0; i < ui->tableWidget\_surname->rowCount(); i++) {

*for* (*auto* j = 0; j < ui->tableWidget\_surname->columnCount(); j++) {

ui->tableWidget\_ph\_num->item(i,j)->setText("");

}

}

*auto* ph\_number = ui->lineEdit\_ph\_num->text().toStdString();

RemoveSpaces(*ph\_number*);

*if*(ph\_number.size() > 12 || ph\_number.size() < 10 || ph\_number.size() == 11){

*throw* std::runtime\_error("The phone number is incorrect!");

*return*;

}

*if*(ph\_number.size() == 10){

*throw* std::runtime\_error("Please, start the phone number with 380...");

*return*;

}

bool ok;

QString::fromStdString(ph\_number).toULongLong(*&ok*);

*if*(!ok){

*throw* std::runtime\_error("The phone number is incorrect!");

*return*;

}

int count = 0;

*for*(*const* *auto*& subscription : SubscriptionVector){

*if*(subscription.GetPhoneNumber() == ph\_number){

ui->tableWidget\_ph\_num << subscription;

++count;

}

}

*if*(count == 0){

QMessageBox::information(NULL,"Info","There is no person with such phone number.");

}

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *person* *with* *the* *subscription* *type* *you* *pick.*

void MainWindow::**on\_actionSimple\_First\_Half\_Day\_triggered**()

{

*try*{

ui->tableWidget\_same\_type->setRowCount(1);

ui->stackedWidget->setCurrentIndex(2);

BackToTableOff();

ShowSameType(*ui->tableWidget\_same\_type*, SubscriptionVector, SubscriptionType::*SimpleFirstHalfDay*);

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *person* *with* *the* *subscription* *type* *you* *pick.*

void MainWindow::**on\_actionSimple\_Full\_Day\_triggered**()

{

*try*{

ui->tableWidget\_same\_type->setRowCount(1);

ui->stackedWidget->setCurrentIndex(2);

BackToTableOff();

ShowSameType(*ui->tableWidget\_same\_type*, SubscriptionVector, SubscriptionType::*SimpleFullDay*);

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *person* *with* *the* *subscription* *type* *you* *pick.*

void MainWindow::**on\_actionLux\_Full\_Day\_triggered**()

{

*try*{

ui->tableWidget\_same\_type->setRowCount(1);

ui->stackedWidget->setCurrentIndex(2);

BackToTableOff();

ShowSameType(*ui->tableWidget\_same\_type*, SubscriptionVector, SubscriptionType::*LuxFullDay*);

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Show* *person* *with* *the* *subscription* *type* *you* *pick.*

void MainWindow::**on\_actionSuper\_Lux\_Full\_Day\_triggered**()

{

*try* {

ui->tableWidget\_same\_type->setRowCount(1);

ui->stackedWidget->setCurrentIndex(2);

BackToTableOff();

ShowSameType(*ui->tableWidget\_same\_type*, SubscriptionVector, SubscriptionType::*SuperLuxFullDay*);

} *catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Read* *the* *info* *from* *file* *and* *fill* *the* *table.*

void MainWindow::**on\_actionRead\_from\_file\_triggered**()

{

*try*{

"C:\\LPNU\\kursova\\code\\OOP\_kursova\\Input.txt" >> SubscriptionVector;

ui->tableWidget << SubscriptionVector;

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Write* *the* *info* *from* *table* *to* *file.*

void MainWindow::**on\_actionWrite\_to\_file\_triggered**()

{

*try*{

"C:\\LPNU\\kursova\\code\\OOP\_kursova\\Output.txt" << SubscriptionVector;

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Add* *person* *you* *write* *to* *table.*

void MainWindow::**on\_actionAdd\_triggered**()

{

*try*{

ui->tableWidget\_2 >> SubscriptionVector;

ui->tableWidget << SubscriptionVector.back();

*for* (*auto* i = 0; i < ui->tableWidget\_2->rowCount(); i++) {

*for* (*auto* j = 0; j < ui->tableWidget\_2->columnCount(); j++) {

ui->tableWidget\_2->item(i,j)->setText("");

}

}

}*catch*(*const* std::exception& exc){

QMessageBox::warning(NULL,"Error",QString::fromStdString(exc.*what*()));

}

}

*//* *Some* *prosperities* *needed* *when* *you* *go* *to* *additional* *widgets.*

void MainWindow::**BackToTableOff**()

{

ui->actionAdd->setVisible(*false*);

ui->actionAdd->setEnabled(*false*);

ui->actionBack\_to\_table->setVisible(*true*);

ui->actionBack\_to\_table->setEnabled(*true*);

ui->actionEdit\_table->setVisible(*false*);

ui->actionEdit\_table->setEnabled(*false*);

ui->actionRead\_from\_file->setVisible(*false*);

ui->actionRead\_from\_file->setEnabled(*false*);

ui->actionWrite\_to\_file->setVisible(*false*);

ui->actionWrite\_to\_file->setEnabled(*false*);

}

**4. Протокол роботи програми**

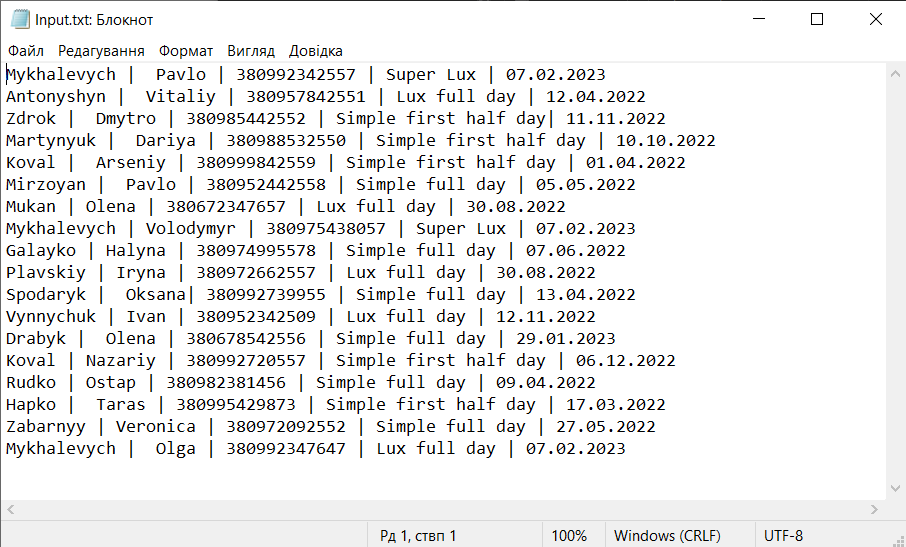
****

Рис 4.1. Вхідний файл

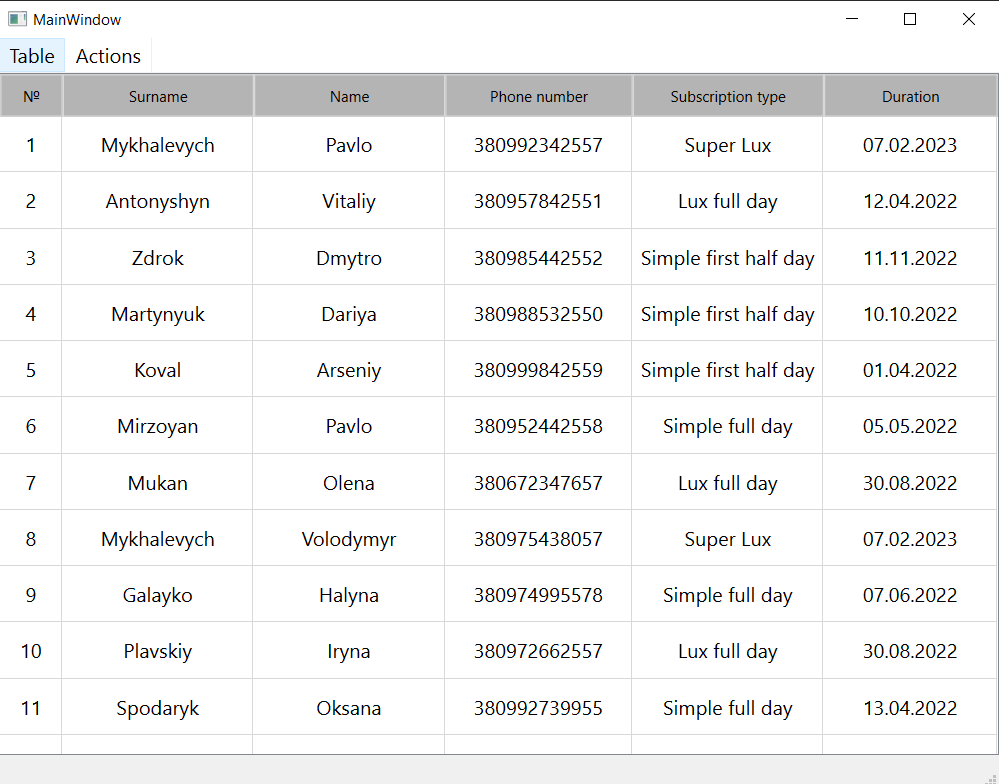
****

Рис 4.2. Вивід заданих даних на екран

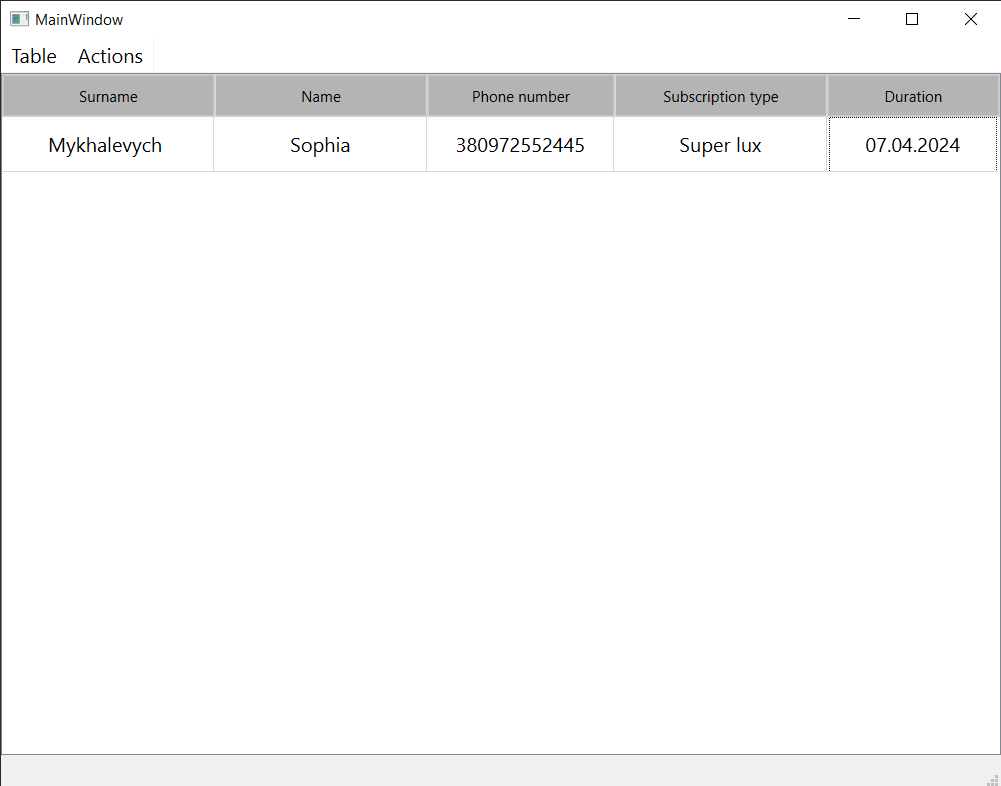
****

Рис 4.3. Додавання нових даних з клавіатури

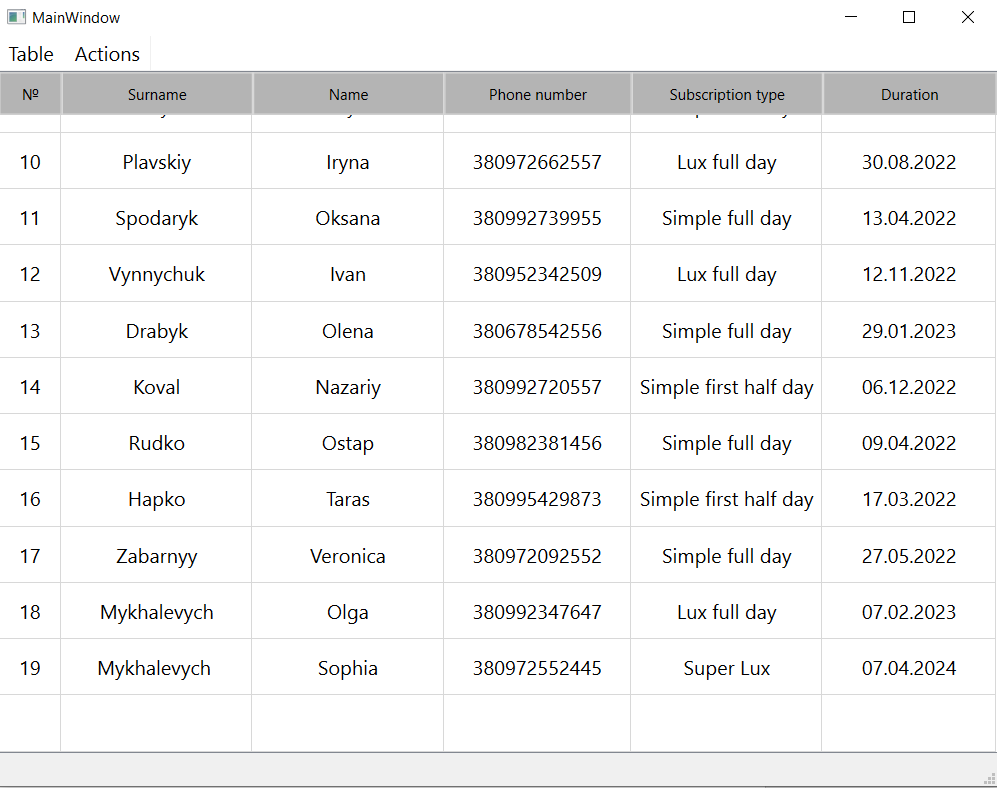
****

Рис 4.4. Вивід доданих з клавіатури даних на екран

****

Рис 4.5. Вивід даних у файл

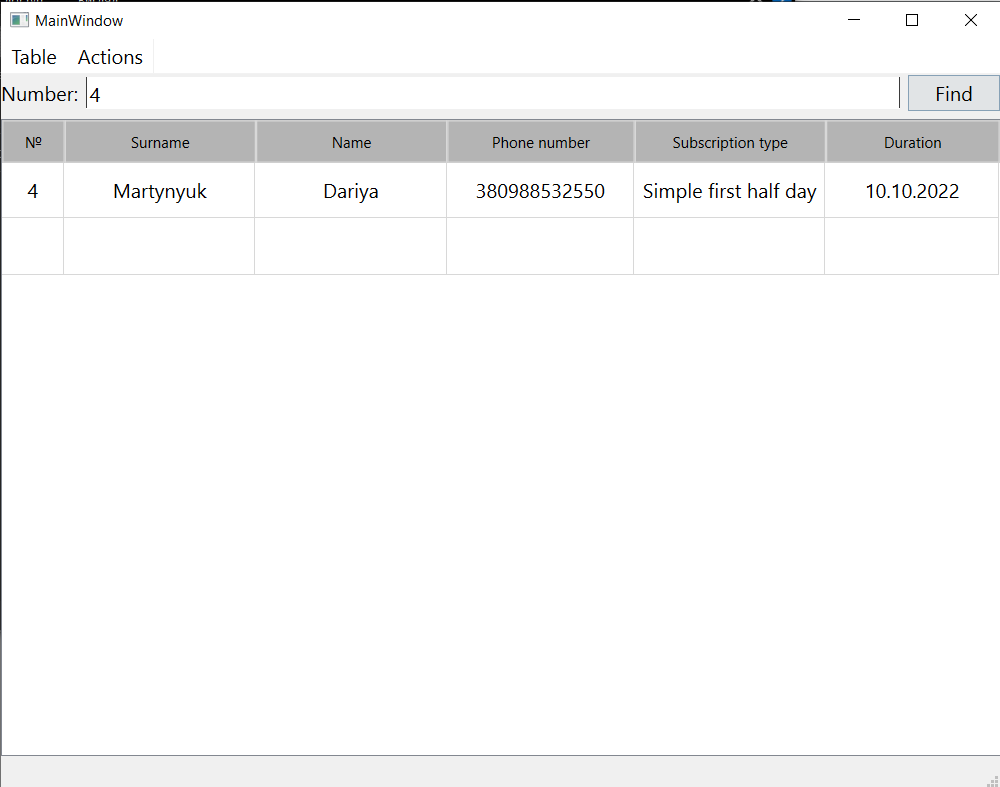
****

Рис 4.6. Пошук даних за номером

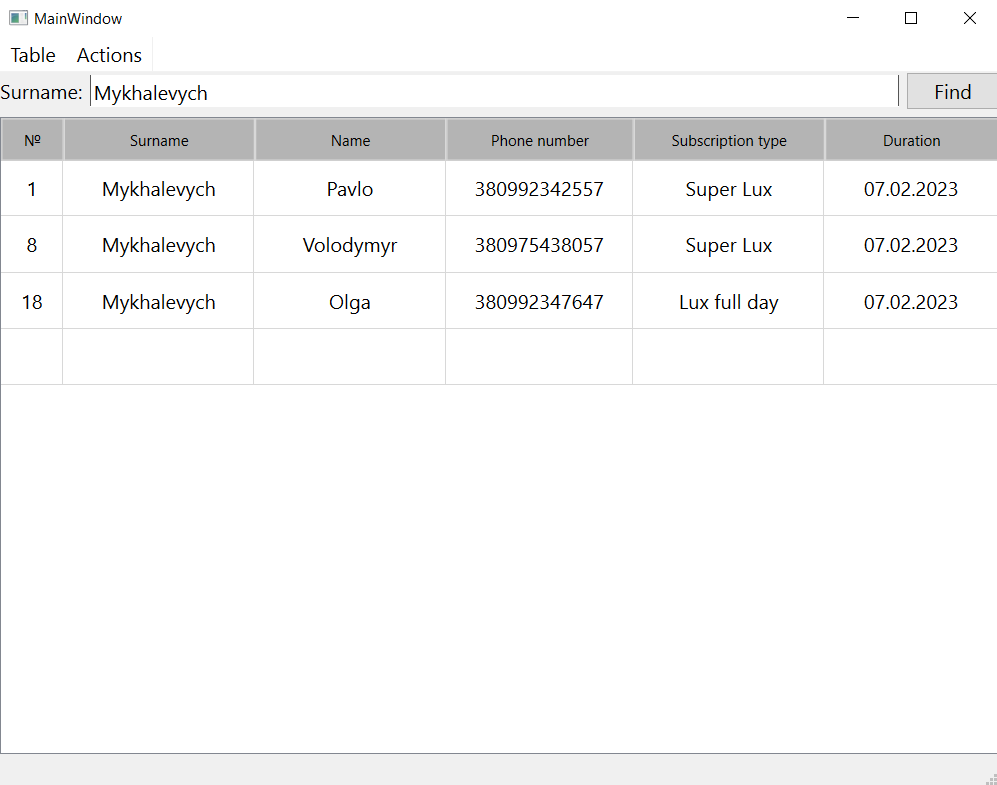
****

Рис 4.7. Пошук даних за прізвищем

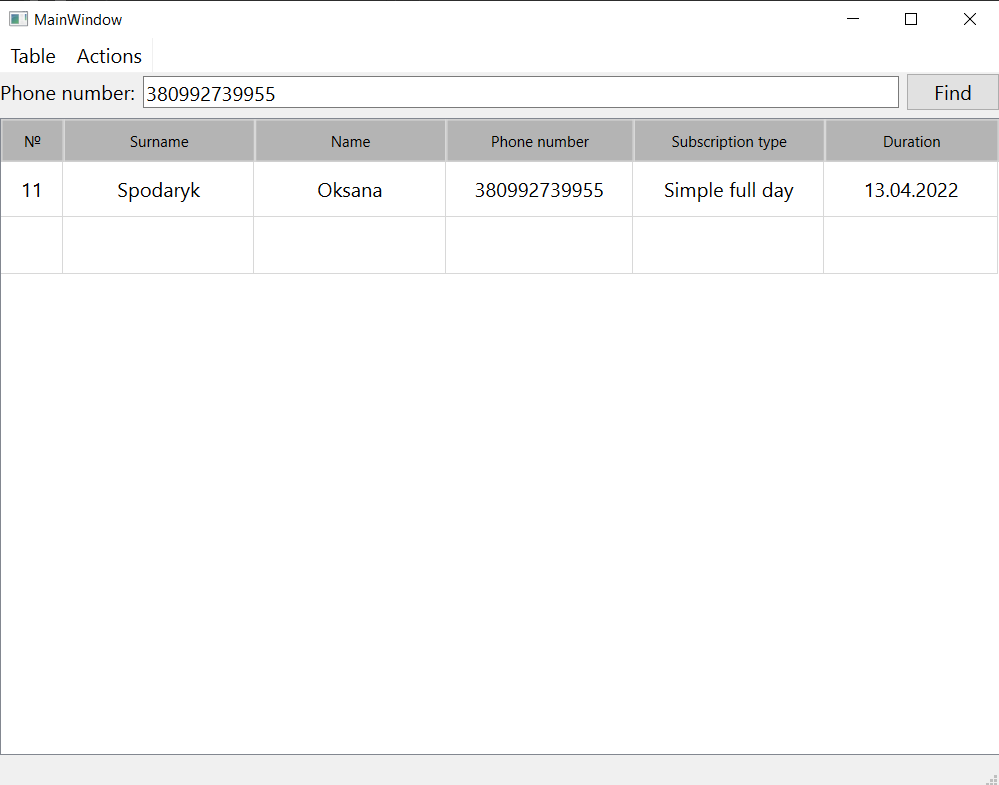
****

Рис 4.8. Пошук даних за номером телефону

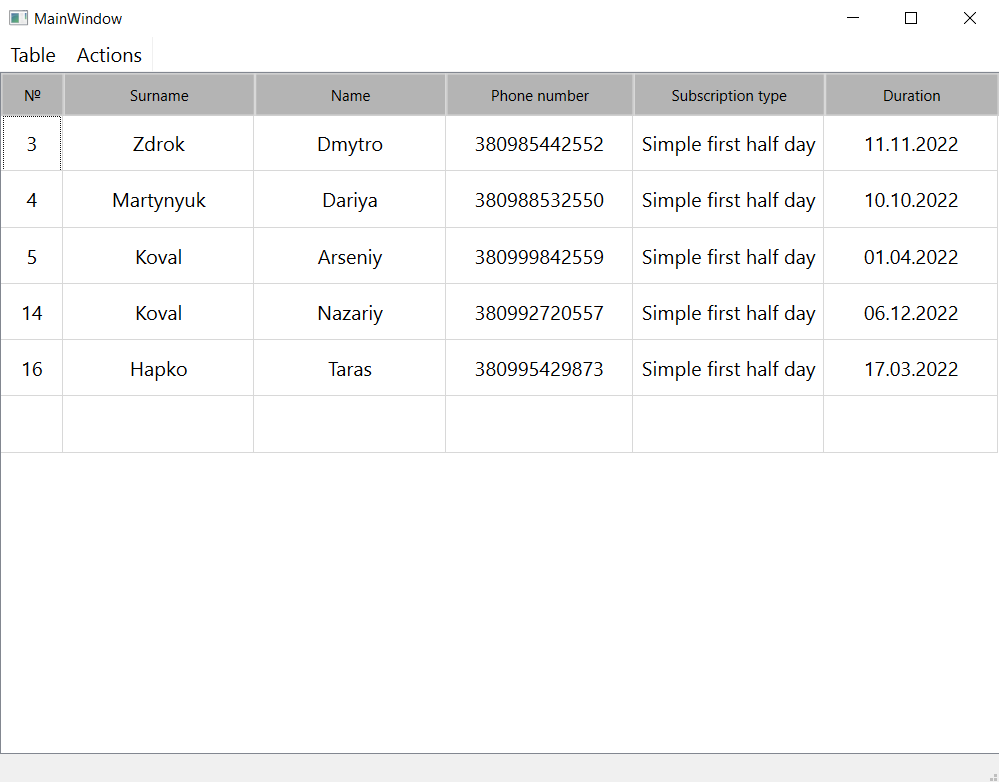
****

Рис 4.9. Пошук даних з заданим типом підписки

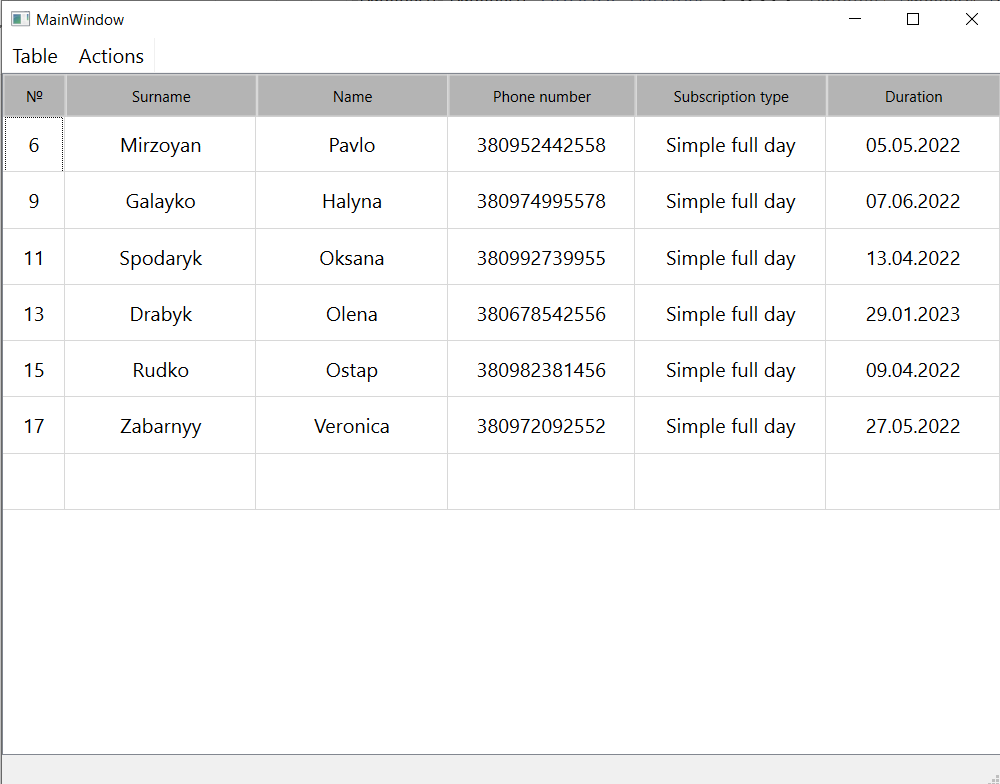
****

Рис 4.10. Пошук даних з найбільш використовуваним типом підписки

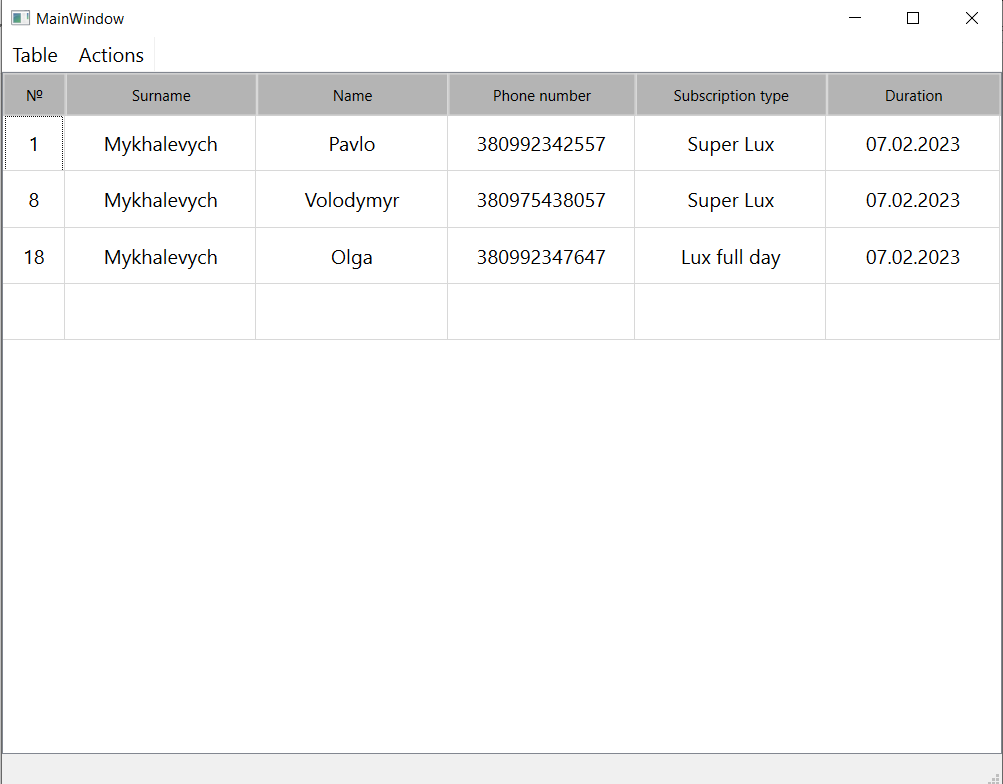
****

Рис 4.11. Пошук даних з найбільш використовуваним строком дії

**5. Інструкція користувача та системні вимоги**

**Зміст**

5.1. Загальні відомості.

5.1.1 Короткий опис.

5.1.2 Необхідне апаратне забезпечення.

5.2. Встановлення ПЗ.

5.3. Інтерфейс програми.

5.4. Повідомлення про помилки.

**5.1.Загальні відомості.**

**5.1.1 Короткий опис ПЗ.**

Програма створена для оптимізації бази даних спорткомплексу. Пакет розроблено мовою C++ у середовищі розробки Qt Creator 4.14.0. Використано об’єктно-орієнтований підхід програмування.

**5.1.2 Необхідне апаратне забезпечення.**

Для коректної роботи програми необхідно:

**Операційна система**: Windows 7, Windows 8, Windows 10.

**Процесор**: Intel Core i5 з частотою 2.4 Гц або кращий, AMD Sempron 180 з частотою 2.4 Ghz або кращий.

**Оперативна пам’ять**: 2 Гігабайт.

**Об’єм жорстоко диску**: 50 Мегабайт.

**Відеокарта**: AMD Radeon 4100 або краща.

**5.2. Встановлення ПЗ.**

Для роботи пакета потрібно:

- встановити Qt 5.12.2;

Опісля запустити на виконання файл **OOP\_kursova.exe.**

**5.3. Інтерфейс програми.**

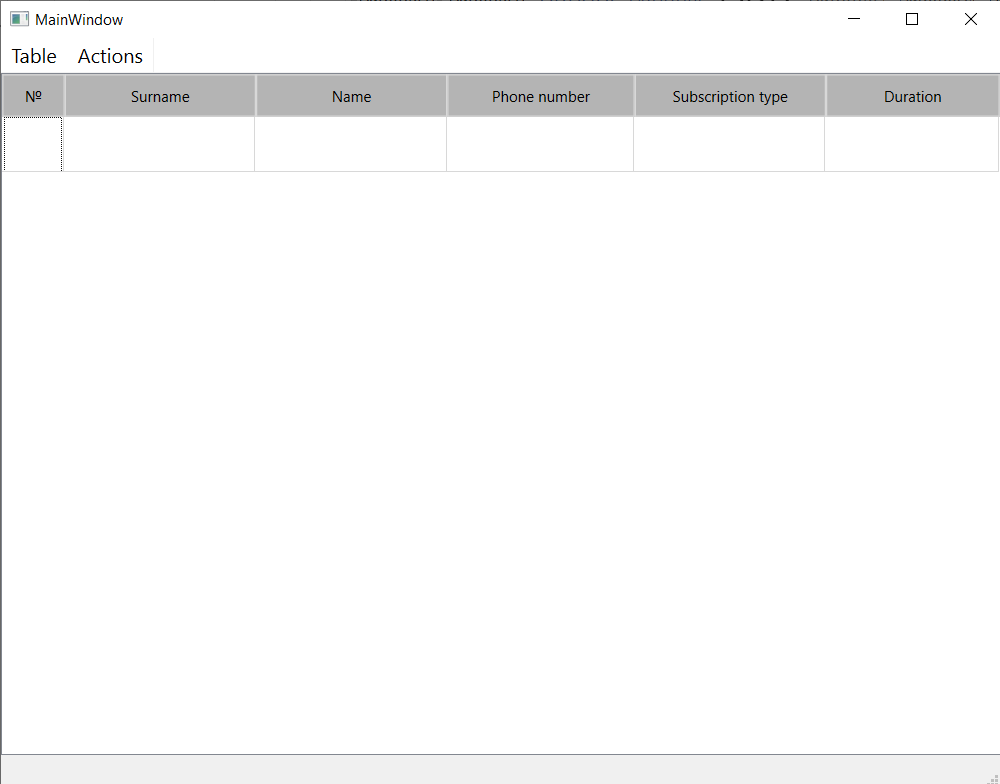
****

Рис 5.3.1. Приклад вхідного файлу

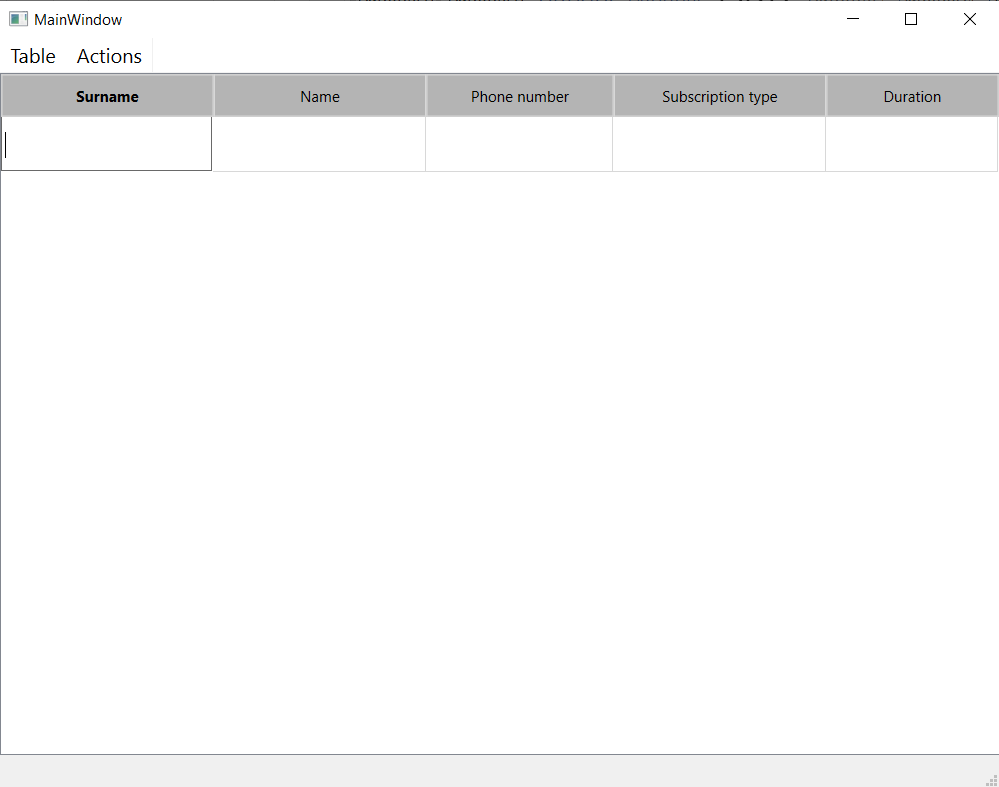
****

Рис 5.3.2. Приклад вхідного файлу

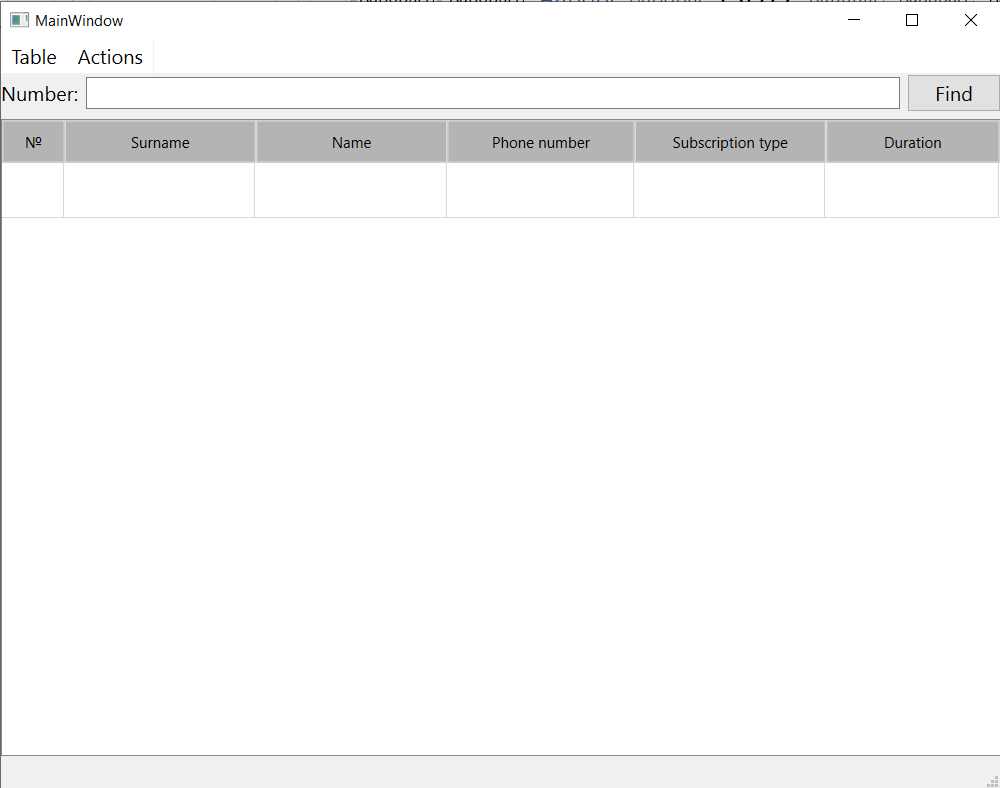
****

Рис 5.3.3. Приклад вхідного файлу

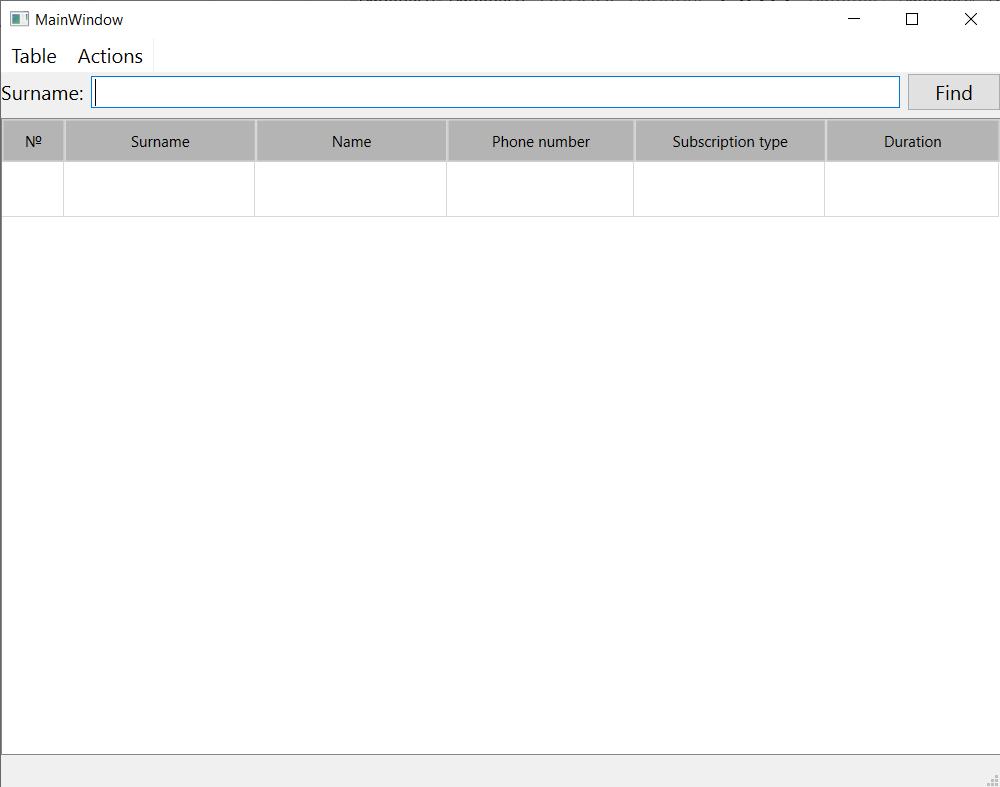


Рис 5.3.4. Приклад вхідного файлу

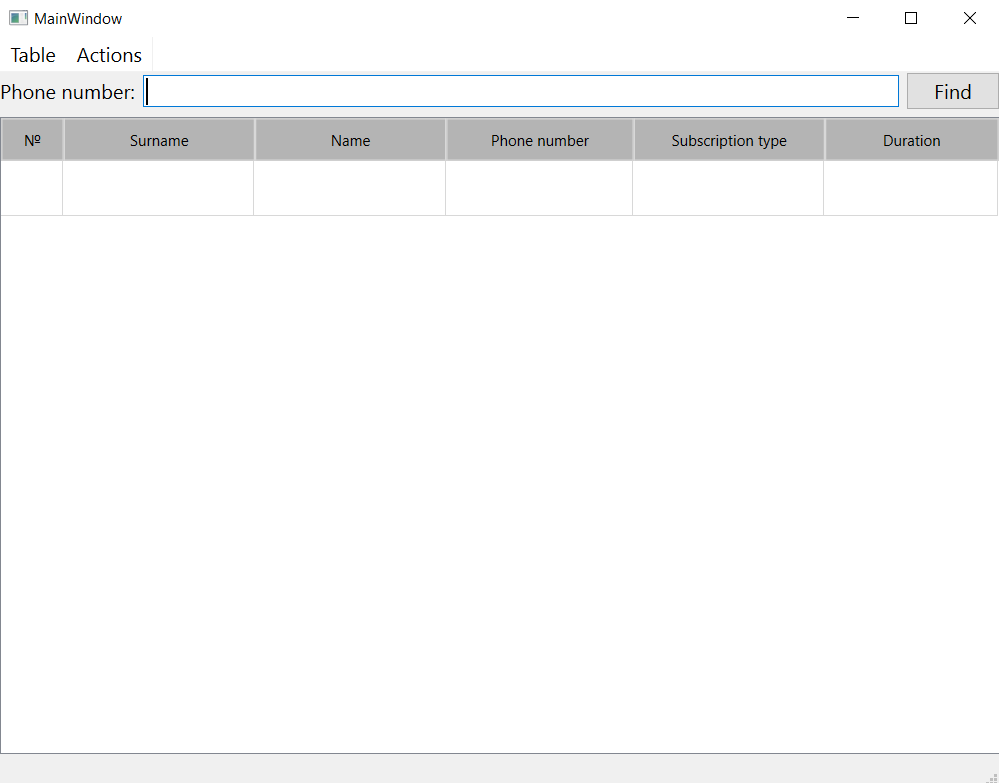
****

Рис 5.3.5. Приклад вхідного файлу

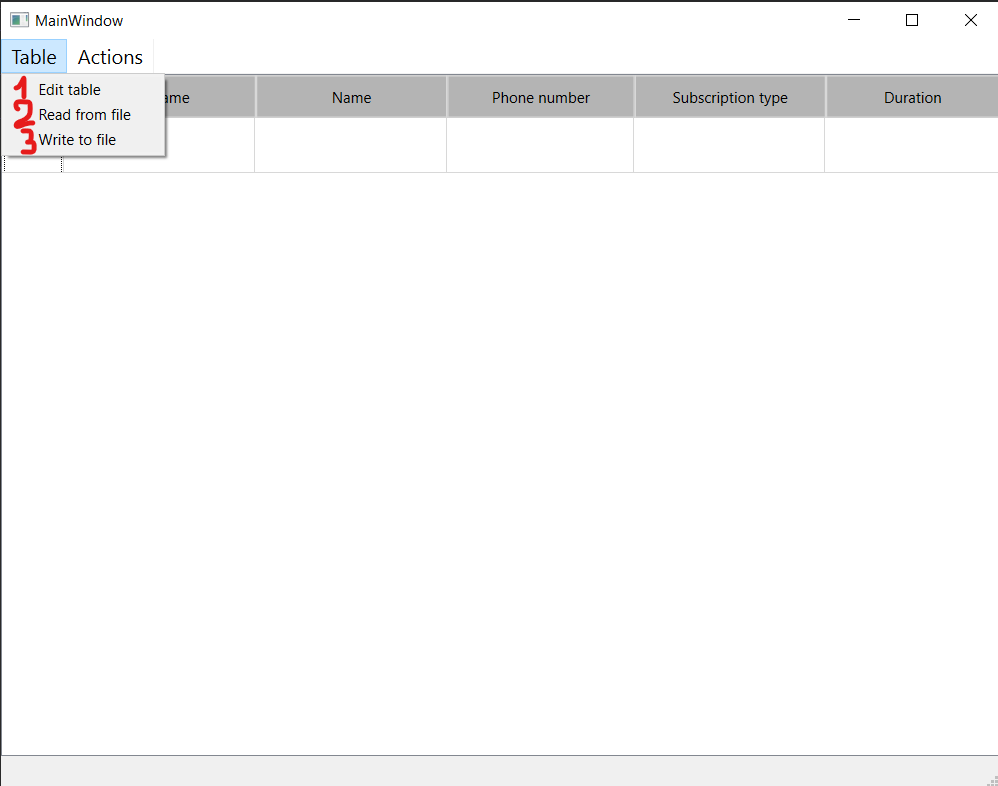
****

Рис 5.3.6. Приклад вхідного файлу

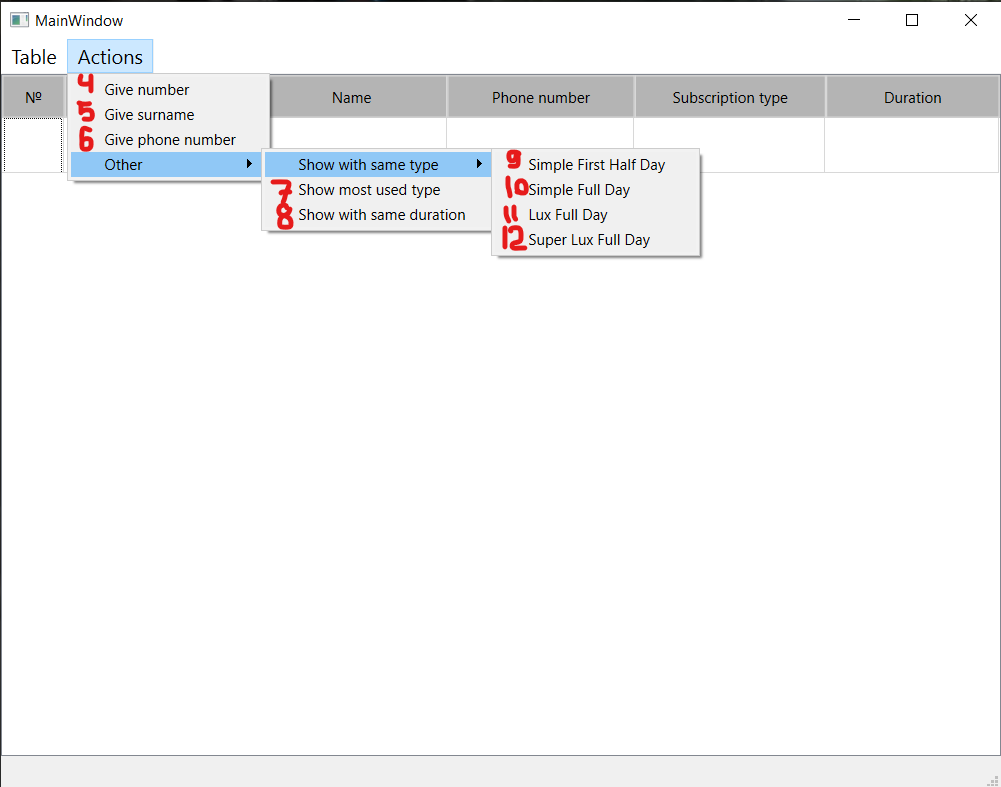
****

Рис 5.3.7. Приклад вхідного файлу

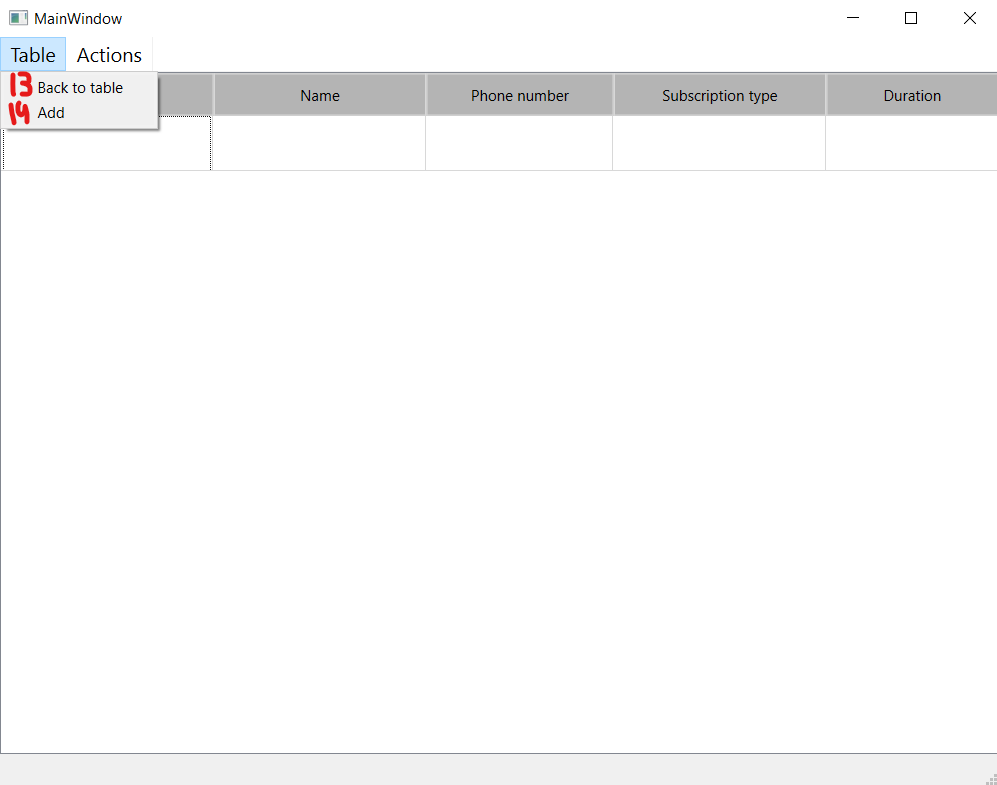
****

Рис 5.3.8. Приклад вхідного файлу

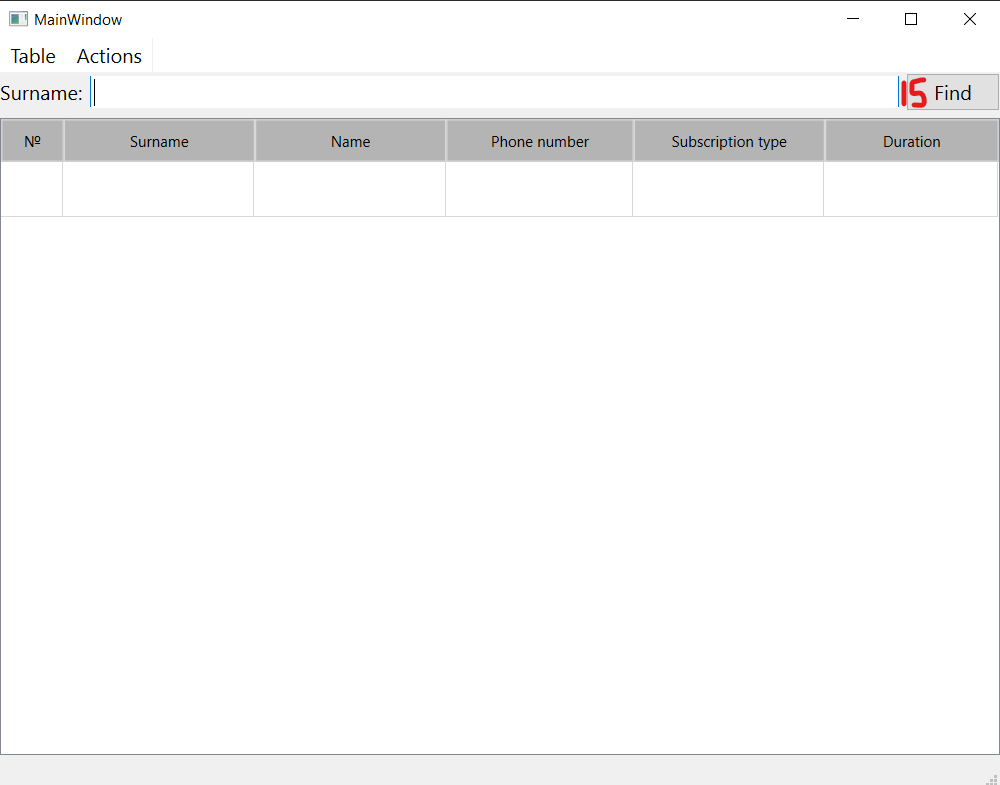
****

Рис 5.3.9. Приклад вхідного файлу

**5.4. Повідомлення про помилки.**

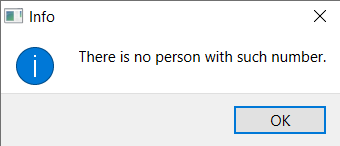


Рис 5.4.1. Приклад вхідного файлу

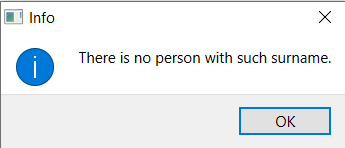


Рис 5.4.2. Приклад вхідного файлу

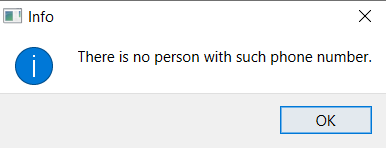


Рис 5.4.3. Приклад вхідного файлу

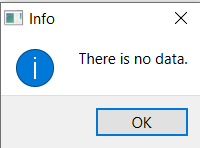


Рис 5.4.4. Приклад вхідного файлу

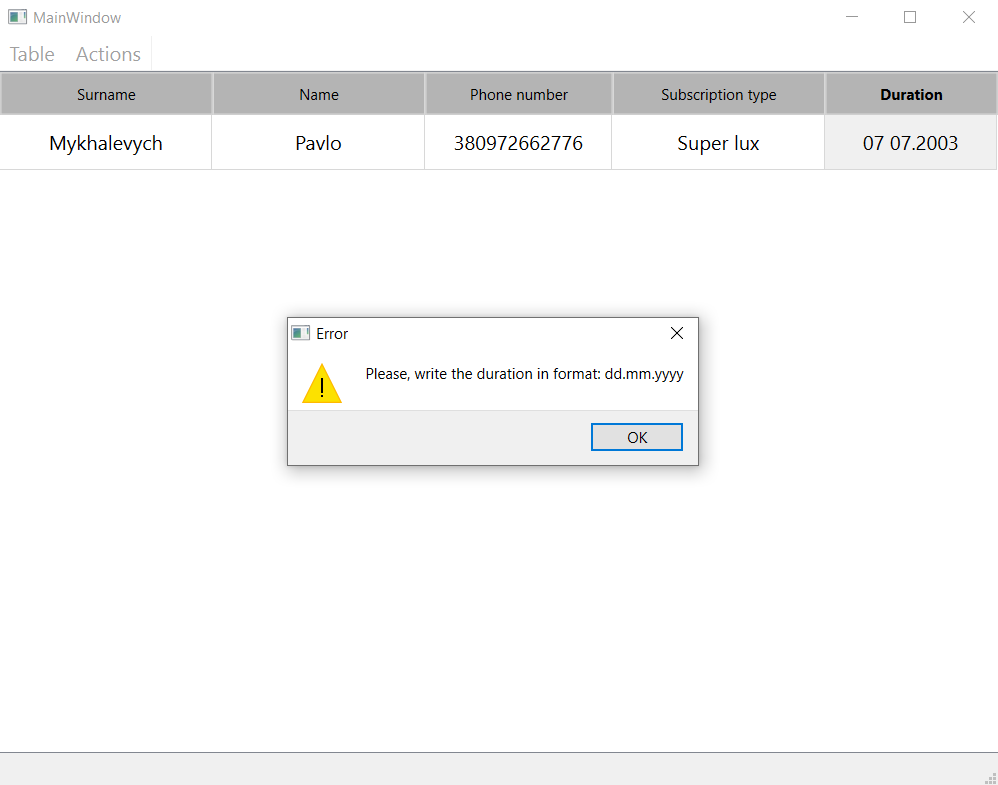
****

Рис 5.4.5. Приклад вхідного файлу

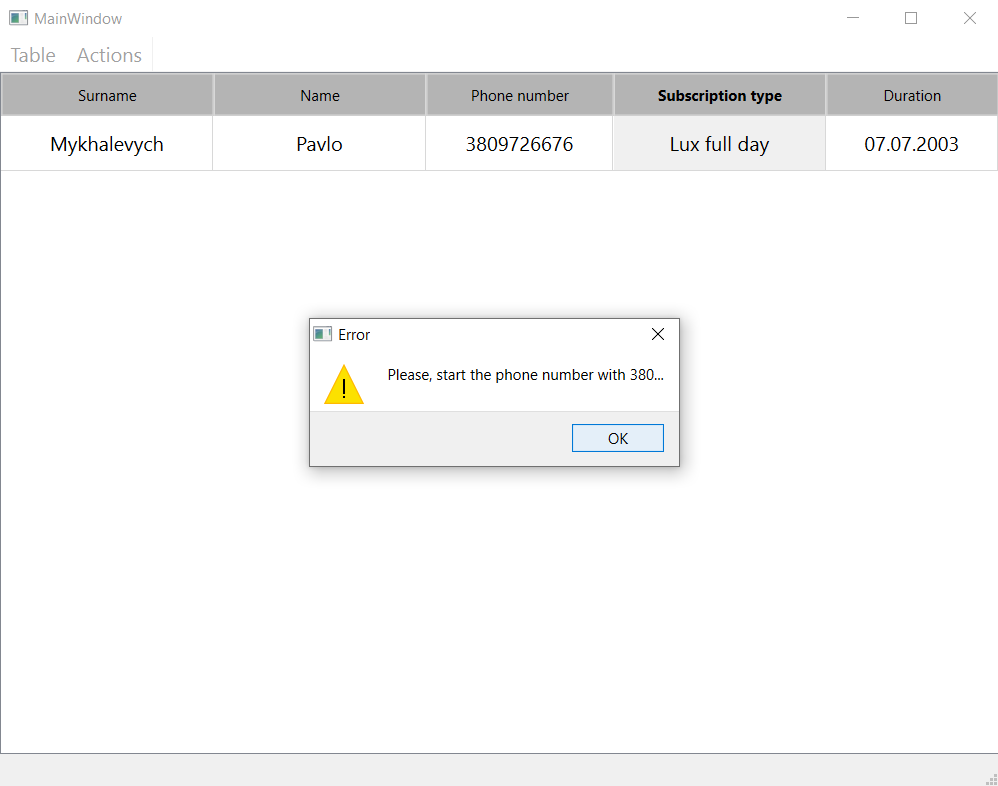
****

Рис 5.4.6. Приклад вхідного файлу

**6. Опис виняткових ситуацій**

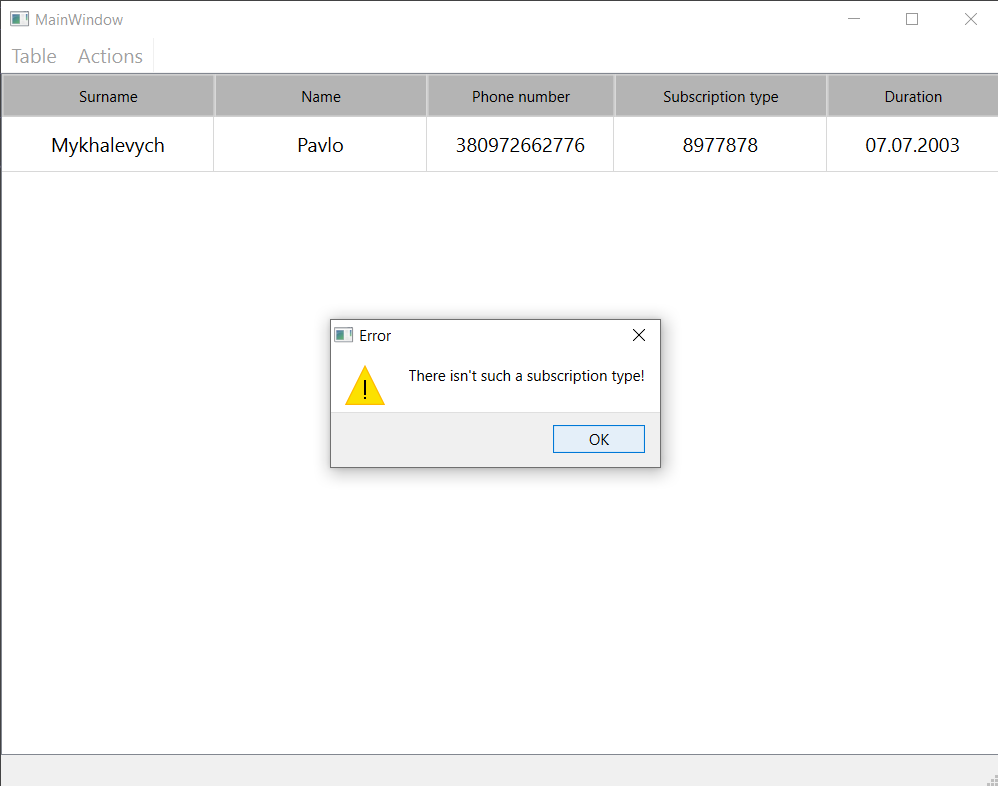
****

Рис 6.1. Приклад вхідного файлу

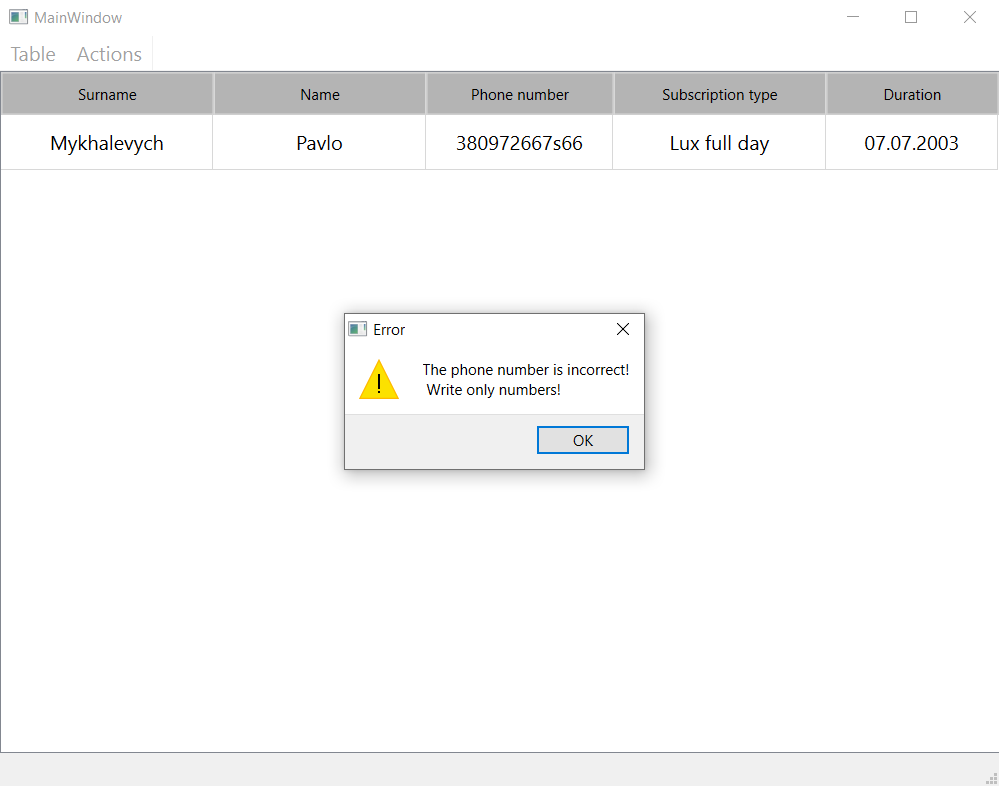


Рис 6.2. Приклад вхідного файлу

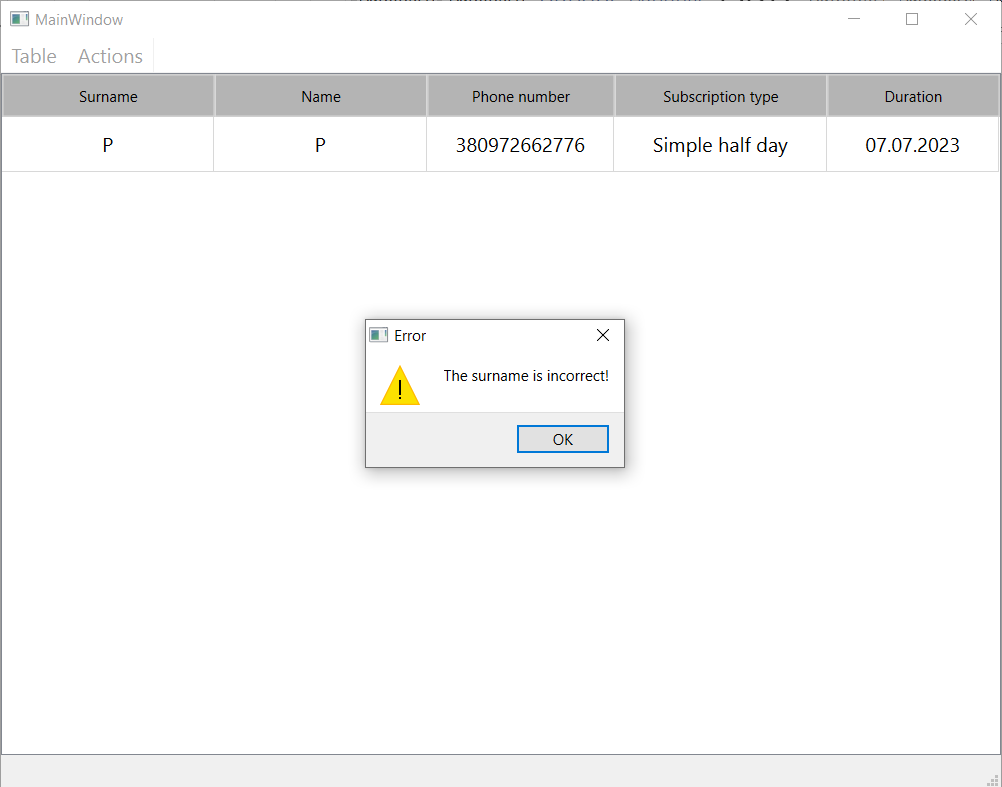


Рис 6.3. Приклад вхідного файлу

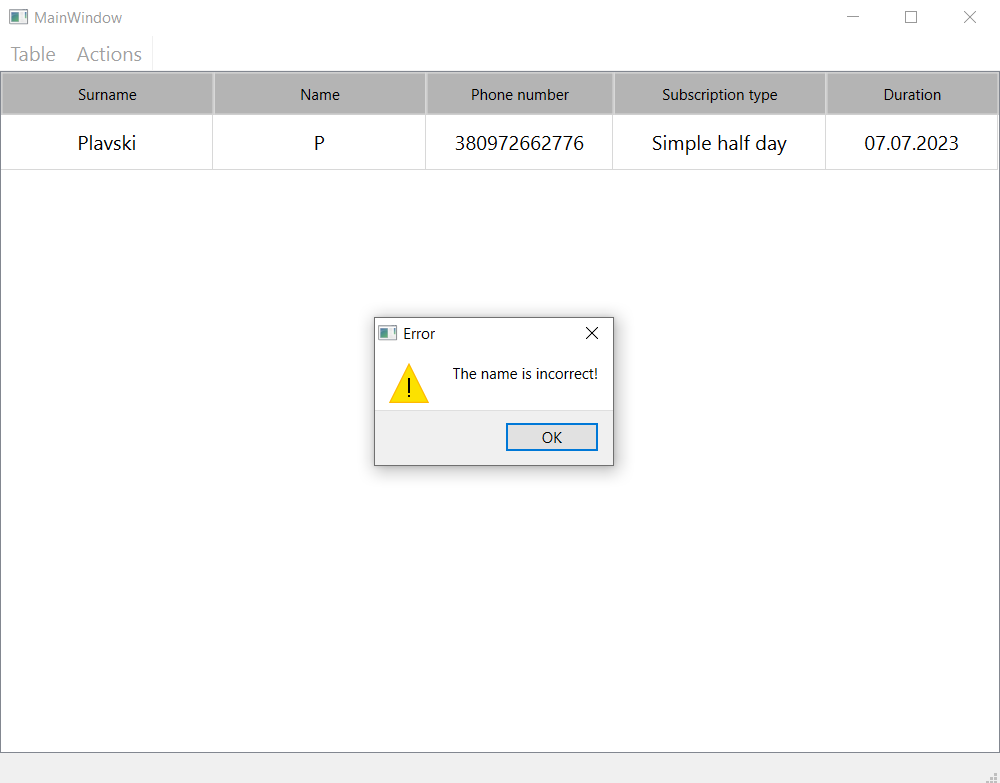
****

Рис 6.4. Приклад вхідного файлу

**7.Структура файлу вхідних даних**

Передбачено введення даних у двох режимах:

1) з клавіатури

2) з файлу.

1) Щоб ввести дані з клавіатури потрібно перейти до вікна додавання даних в базу:

Table->Edit:

Тепер потрібно вводити дані у відповідну комірку, при неправильних даних появиться помилка з відповідним текстом. Після введення всіх даних потрібно нажати сполучення Table->Add, щоб додати дані до бази.

2) У файлі з вхідними даними структура має бути у форматі:

Прізвище(стрічка довжиною len, 3<len<35) | Імя(стрічка довжиною len, 3<len<35) | Номер телефону(починається з 38…, 12 цифр) | Тип абонементу(має містити зв’язку слів: Simle+full, Simple+half, Lux+full,Super+lux ) | Срок дії (формат: дд.мм.рррр)

Інформація про кожну людину починається з нового рядка.

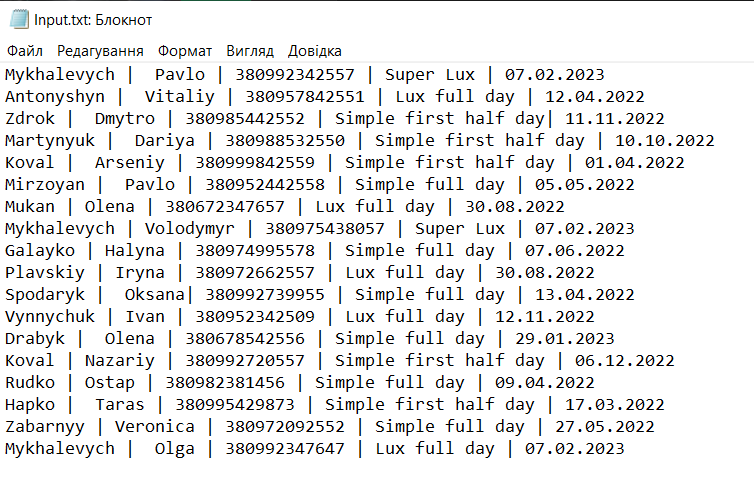
****

Рис 7.1. Приклад вхідного файлу

**Висновок**

На даній курсовій роботі, я розробив програмне забезпечення, з використанням об’єктно-орієнтовного підходу програмування на мові С++, для оптимізації бази даних у спортивному комплексі. Програма розроблена у віжуальному середовищі Qt Creator. Програма обробляє задані дані і видає користувачу бажану інформацію.

В подальшому можливе збільшення функціоналу програми для задоволення потреб користувачів.

**Список використаних джерел**

1. <https://codescracker.com/cpp/cpp-detecting-eof.htm> ;

2. <https://stackoverflow.com/> ;

3. <https://doc.qt.io/qt-5/qtcore-module.html> ;

4. <https://ravesli.com/uroki-cpp/> ;

5. <https://www.geeksforgeeks.org/> ;