БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Международный институт дистанционного образования

Кафедра «Информационные системы и технологии»

**КУРСОВОЙ ПРОЕКТ**

по учебной дисциплине

«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»

Исполнитель: студент 3 курса, группы 41703120 Реут В.Л.

Руководитель: Радкевич А. С.

Минск 2023

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc125733098)

[ЗАДАНИЕ 1 4](#_Toc125733099)

[Постановка задачи 4](#_Toc125733100)

[Исходный код программы 4](#_Toc125733101)

[ЗАДАНИЕ 2 9](#_Toc125733102)

[Постановка задачи 9](#_Toc125733103)

[***Диета.*** Определить иерархию овощей. Собрать в салат. Посчитать каллорийность. 9](#_Toc125733104)

[Исходный код программы 9](#_Toc125733105)

[Исходный код программы 16](#_Toc125733106)

[Результат выполнения программы 17](#_Toc125733107)

[ЗАДАНИЕ 3 18](#_Toc125733108)

[Постановка задачи 18](#_Toc125733109)

[Исходный код программы 18](#_Toc125733110)

[Результат выполнения программы 24](#_Toc125733111)

[ЗАКЛЮЧЕНИЕ 27](#_Toc125733112)

[ЛИТЕРАТУРА 28](#_Toc125733113)

# **ВВЕДЕНИЕ**

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

С++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко С++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на С++

С++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на С++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на С++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на С++ в приложения под эти платформы.

В 1979-80 годах Бьерн Страуструп разработал расширение к языку Си - "Си с классами". В 1983 язык был переименован в С++.

В 1985 году была выпущена первая коммерческая версия языка С++, а также первое издание книги "Языка программирования C++", которая представляла первое описание этого языка при отсутствии официального стандарта.

В 1989 была выпущена новая версия языка C++ 2.0, которая включала ряд новых возможностей. После этого язык развивался относительно медленно вплоть до 2011 года. Но при этом в 1998 году была предпринята первая попытка по стандартизации языка организацией ISO (International Organiztion for Standartization). Первый стандарт получил название ISO/IEC 14882:1998 или сокращенно С++98. В дальнейшем в 2003 была издана новая версия стандарта C++03.

# **ЗАДАНИЕ 1**

## **Постановка задачи**

Целью данного задания является: Создать класс библиографических описаний книг CBookCard. Класс должен содержать поля согласно таблицам 1 – 3, в соответствии с номером варианта.

1. Все поля должны быть закрытыми. Для всех их должны быть созданы методы доступа (или свойства).
2. В конструкторе за исключением рейтинга и комментария должна осуществляться инициализация всех полей.
3. Для всех вариантов необходимо создать метод, выводящий описание книги в строку (переопределение метода ToString()), в которой будут присутствовать все значения полей класса, кроме поля – «комментария». В данном методе значения полей ISBN, УДК, ББК выводятся через пробел без использования знаков препинания. Значения атрибутов рейтинг и тираж должны выводиться в следующей форме: Тираж: <значение тиража>. Рейтинг: <значение рейтинга>.
4. Создать два объекта библиографических описаний книг. Вызвать для них метод, осуществляющий вывод строки библиографического описания. Можно использовать как реально существующие книги, так и чисто гипотетические.
5. Создать массив (размерностью не более 5) объектов библиографических описаний книг. Создать статический метод для упорядочивания (по году издания) данного массива. Вывести на экран значения массива до сортировки и после сортировки.

## **Исходный код программы**

**Header:**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

class CBookCard

{

public:

CBookCard();

CBookCard(string \_author, string \_title,

string \_publishing\_house, string \_BBK,

int \_year\_publishing);

string get\_author() const;

string get\_title() const;

string get\_publishing\_house() const;

string get\_BBK() const;

int get\_year\_publishing() const;

void set\_comment(string \_comment);

string get\_comment() const;

void set\_rating(double \_rating);

double get\_rating() const;

string book\_description() const;

static void sort\_book(vector <CBookCard>& \_vec);

~CBookCard();

private:

string author, title, publishing\_house, BBK,

comment;

double rating;

int year\_publishing;

};

CPP:

#include "Course\_Project\_OOP\_Header.h"

// CBookCard

//============================================================

CBookCard::CBookCard()

{

rating = 0;

comment = "";

}

CBookCard::CBookCard(string \_author, string \_title,

string \_publishing\_house, string \_BBK, int \_year\_publishing) :

author(\_author), title(\_title),

publishing\_house(\_publishing\_house), BBK(\_BBK),

year\_publishing(\_year\_publishing)

{

rating = 0;

comment = "";

}

string CBookCard::get\_author() const

{

return author;

}

string CBookCard::get\_title() const

{

return title;

}

string CBookCard::get\_publishing\_house() const

{

return publishing\_house;

}

string CBookCard::get\_BBK() const

{

return BBK;

}

int CBookCard::get\_year\_publishing() const

{

return year\_publishing;

}

void CBookCard::set\_comment(string \_comment)

{

comment = \_comment;

}

string CBookCard::get\_comment() const

{

return comment;

}

void CBookCard::set\_rating(double \_rating)

{

if (\_rating >= 0 && \_rating <= 6)

{

rating = \_rating;

}

}

double CBookCard::get\_rating() const

{

return rating;

}

string CBookCard::book\_description() const

{

string tmp;

tmp = "Author: " + author + '\n';

tmp += "Title: " + title + '\n';

tmp += "Publishing house: " + publishing\_house + '\n';

tmp += "The year of publishing: " + to\_string(year\_publishing) + '\n';

tmp += BBK + '\n';

tmp += "Rating: " + to\_string(rating).erase(to\_string(rating).size() - 5) + '\n' + '\n';

return tmp;

}

void CBookCard::sort\_book(vector <CBookCard>& \_vec)

{

sort(\_vec.begin(), \_vec.end(), [&](CBookCard l, CBookCard r) {return l.get\_year\_publishing() < r.get\_year\_publishing(); });

}

CBookCard::~CBookCard() {}

//============================================================

**Main:**

#include "Course\_Project\_OOP\_Header.h"

int main()

{

CBookCard Glukhovsky("Dmitrii Glukhovsky", "Metro 2033", "Ecsmo", "KHG765", 2005);

Glukhovsky.set\_rating(4.5);

CBookCard Diakov("Andrey Diakov", "To light", "AST", "84(2Ros=Rus)6", 2010);

CBookCard Azimov("Isaac Azimov", "Fundamintation", "AM", "IJK11", 1942);

CBookCard Cormen("Thomas H. Cormen", "Introduction to Algorithms, CLRS", "MIT Press", "231", 1990);

Cormen.set\_rating(6);

CBookCard Knuth("Donald Ervin Knuth", "The Art of Computer Programming", "Addison - Wesley", "IJK11", 1968);

Knuth.set\_rating(6);

vector <CBookCard> vec;

vec.push\_back(Glukhovsky);

vec.push\_back(Diakov);

vec.push\_back(Azimov);

vec.push\_back(Cormen);

vec.push\_back(Knuth);

cout << "Before sorting: \n" << endl;

for\_each(vec.begin(), vec.end(), [&](const CBookCard& a) {cout << a.book\_description(); });

CBookCard::sort\_book(vec);

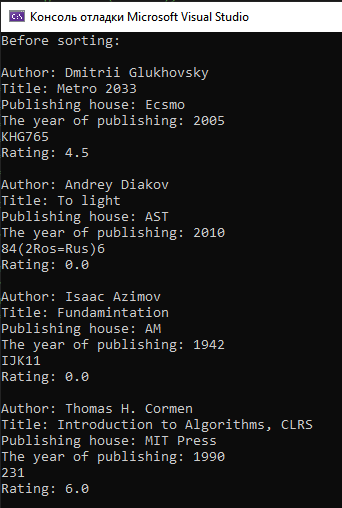
cout << "After sorting: \n" << endl;

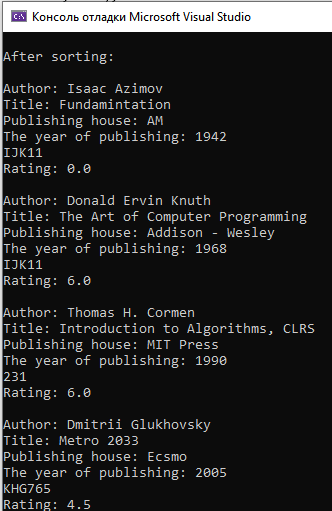
for\_each(vec.begin(), vec.end(), [&](const CBookCard& a) {cout << a.book\_description(); });

return 0;

}

**Результат выполнения программы:**





# ЗАДАНИЕ 2

## Постановка задачи

Целями данного задания являются: необходимо реализовать иерархию классов предметной области на языке C++ в соответствии с указанным вариантом задания.

1. Иерархия классов должна состоять не менее чем из 3-х уровней.
2. Каждый класс должен содержать уникальный набор полей.
3. В классах не должно быть открытых полей (должны быть модификаторы доступа private или protected). Для доступа к полям класса необходимо использовать свойства.
4. В процессе реализации иерархии классов использовать абстрактные классы.
5. Классы рекомендуется оформить в виде библиотеки классов. Проверку функциональности осуществить в консольном проекте.

## ***Диета.*** Определить иерархию овощей. Собрать в салат. Посчитать каллорийность.

## **Исходный код программы**

**Библиотека классов:**

**Header:**

#pragma once

#ifdef OOP\_DLL\_Lib

#define DECLSPEC \_\_declspec(dllexport)

#else

#define DECLSPEC \_\_declspec(dllimport)

#endif // OOP\_DLL\_Lib

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

// Hierarchy vegetable

//============================================================

// abstract class Vegetable

class DECLSPEC Vegetable

{

public:

Vegetable();

Vegetable(string \_name);

Vegetable(string \_name, double \_kcal\_per\_100g);

Vegetable(string \_name, double \_kcal\_per\_100g, double \_weight);

virtual string get\_name() const;

virtual double get\_kcal\_per\_100g() const;

virtual double get\_weight() const;

virtual void set\_category(string \_category);

virtual string get\_category() const;

double get\_total\_calories() const;

virtual void all\_info() = 0;

virtual ~Vegetable();

protected:

string name, category;

double kcal\_per\_100g, weight;

};

//============================================================

// CucurbitaceaeVegetable classes

//============================================================

class DECLSPEC CucurbitaceaeVegetable : public Vegetable

{

public:

CucurbitaceaeVegetable();

CucurbitaceaeVegetable(string \_name, double \_kcal\_per\_100g);

CucurbitaceaeVegetable(string \_name, double \_kcal\_per\_100g, double \_weight);

void set\_harvest\_country(string \_harvest\_country);

string get\_harvest\_country();

virtual void all\_info() override;

~CucurbitaceaeVegetable();

protected:

string harvest\_country;

};

class DECLSPEC Cucumber : public CucurbitaceaeVegetable

{

public:

Cucumber();

Cucumber(double \_weight);

~Cucumber();

};

//============================================================

// NightshadeVegetable classes

//============================================================

class DECLSPEC NightshadeVegetable : public Vegetable

{

public:

NightshadeVegetable();

NightshadeVegetable(string \_name, double \_kcal\_per\_100g);

NightshadeVegetable(string \_name, double \_kcal\_per\_100g, double \_weight);

void set\_variety(string \_variety);

string get\_variety();

virtual void all\_info() override;

~NightshadeVegetable();

protected:

string variety;

};

class DECLSPEC Potato : public NightshadeVegetable

{

public:

Potato();

Potato(double \_weight);

~Potato();

};

class DECLSPEC Tomato : public NightshadeVegetable

{

public:

Tomato();

Tomato(double \_weight);

~Tomato();

};

//============================================================

// RootVegetable classes

//============================================================

class DECLSPEC RootVegetable : public Vegetable

{

public:

RootVegetable();

RootVegetable(string \_name, double \_kcal\_per\_100g);

RootVegetable(string \_name, double \_kcal\_per\_100g, double \_weight);

void set\_crop\_year(string \_crop\_year);

string get\_crop\_year();

virtual void all\_info() override;

~RootVegetable();

protected:

string crop\_year;

};

class DECLSPEC Carrot : public RootVegetable

{

public:

Carrot();

Carrot(double \_weight);

~Carrot();

};

class DECLSPEC Beet : public RootVegetable

{

public:

Beet();

Beet(double \_weight);

~Beet();

};

//============================================================

class DECLSPEC Salad

{

public:

Salad();

double get\_total\_calories() const;

void add\_ingredients(Vegetable\* veg);

void about\_salad();

~Salad();

private:

vector<Vegetable\*> salad;

//Flavoring flavoring;

};

**CPP:**

#include "Course\_Project\_OOP\_DLL.h"

// Hierarchy vegetable

//============================================================

// abstract class Vegetable

Vegetable::Vegetable() : kcal\_per\_100g(0.0), weight(0.0) {}

Vegetable::Vegetable(string \_name) : name(\_name), kcal\_per\_100g(0.0), weight(0.0) {}

Vegetable::Vegetable(string \_name, double \_kcal\_per\_100g)

: name(\_name), kcal\_per\_100g(\_kcal\_per\_100g), weight(0.0) {}

Vegetable::Vegetable(string \_name, double \_kcal\_per\_100g, double \_weight)

: name(\_name), kcal\_per\_100g(\_kcal\_per\_100g), weight(\_weight) {}

string Vegetable::get\_name() const

{

return name;

}

double Vegetable::get\_kcal\_per\_100g() const

{

return kcal\_per\_100g;

}

double Vegetable::get\_weight() const

{

return weight;

}

void Vegetable::set\_category(string \_category)

{

category = \_category;

}

string Vegetable::get\_category() const

{

return category;

}

double Vegetable::get\_total\_calories() const

{

return kcal\_per\_100g \* weight / 100.0;

}

Vegetable::~Vegetable() {}

//============================================================

// CucurbitaceaeVegetable classes

//============================================================

CucurbitaceaeVegetable::CucurbitaceaeVegetable()

{

set\_category("Cucurbitaceae Vegetable");

}

CucurbitaceaeVegetable::CucurbitaceaeVegetable(string \_name, double \_kcal\_per\_100g)

: Vegetable(\_name, \_kcal\_per\_100g), harvest\_country("Unknow")

{

set\_category("Cucurbitaceae Vegetable");

}

CucurbitaceaeVegetable::CucurbitaceaeVegetable(string \_name, double \_kcal\_per\_100g, double \_weight)

: Vegetable(\_name, \_kcal\_per\_100g, \_weight), harvest\_country("Unknow")

{

set\_category("Cucurbitaceae Vegetable");

}

void CucurbitaceaeVegetable::set\_harvest\_country(string \_harvest\_country)

{

harvest\_country = \_harvest\_country;

}

string CucurbitaceaeVegetable::get\_harvest\_country()

{

return harvest\_country;

}

void CucurbitaceaeVegetable::all\_info()

{

cout << "Name: " << get\_name() << '\n'

<< "Harvest country: " << get\_harvest\_country() << '\n'

<< "Category: " << get\_category() << '\n'

<< "Calories per 100 gramm: " << get\_kcal\_per\_100g() << '\n'

<< "Weight: " << get\_weight() << '\n'

<< "Total calories: " << get\_total\_calories() << endl;

}

CucurbitaceaeVegetable::~CucurbitaceaeVegetable() {}

// class Cucumber

Cucumber::Cucumber() {}

Cucumber::Cucumber(double \_weight) : CucurbitaceaeVegetable("Cucumber", 41.0, \_weight) {}

Cucumber::~Cucumber() {}

//============================================================

// NightshadeVegetable classes

//============================================================

NightshadeVegetable::NightshadeVegetable()

{

variety = "Unknow";

set\_category("Nightshade Vegetable");

}

NightshadeVegetable::NightshadeVegetable(string \_name, double \_kcal\_per\_100g)

: Vegetable(\_name, \_kcal\_per\_100g), variety("Unknow")

{

set\_category("Nightshade Vegetable");

}

NightshadeVegetable::NightshadeVegetable(string \_name, double \_kcal\_per\_100g, double \_weight)

: Vegetable(\_name, \_kcal\_per\_100g, \_weight), variety("Unknow")

{

set\_category("Nightshade Vegetable");

}

void NightshadeVegetable::set\_variety(string \_variety)

{

variety = \_variety;

}

string NightshadeVegetable::get\_variety()

{

return variety;

}

void NightshadeVegetable::all\_info()

{

cout << "Name: " << get\_name() << '\n'

<< "Category: " << get\_category() << '\n'

<< "Variety: " << get\_variety() << '\n'

<< "Calories per 100 gramm: " << get\_kcal\_per\_100g() << '\n'

<< "Weight: " << get\_weight() << '\n'

<< "Total calories: " << get\_total\_calories() << endl;

}

NightshadeVegetable::~NightshadeVegetable() {}

// Class Potato

Potato::Potato() {}

Potato::Potato(double \_weight) : NightshadeVegetable("Potato", 29, \_weight) {}

Potato::~Potato() {}

// Class Tomato

Tomato::Tomato() {}

Tomato::Tomato(double \_weight) : NightshadeVegetable("Tomato", 15, \_weight) {}

Tomato::~Tomato()

{

}

//============================================================

// RootVegetable classes

//============================================================

RootVegetable::RootVegetable()

{

set\_category("Root Vegetable");

}

RootVegetable::RootVegetable(string \_name, double \_kcal\_per\_100g)

: Vegetable(\_name, \_kcal\_per\_100g), crop\_year("Unknow")

{

set\_category("Root Vegetable");

}

RootVegetable::RootVegetable(string \_name, double \_kcal\_per\_100g, double \_weight)

: Vegetable(\_name, \_kcal\_per\_100g, \_weight), crop\_year("Unknow")

{

set\_category("Root Vegetable");

}

void RootVegetable::set\_crop\_year(string \_crop\_year)

{

crop\_year = \_crop\_year;

}

string RootVegetable::get\_crop\_year()

{

return crop\_year;

}

void RootVegetable::all\_info()

{

cout << "Name: " << get\_name() << '\n'

<< "Category: " << get\_category() << '\n'

<< "Crop year: " << get\_crop\_year() << '\n'

<< "Calories per 100 gramm: " << get\_kcal\_per\_100g() << '\n'

<< "Weight: " << get\_weight() << '\n'

<< "Total calories: " << get\_total\_calories() << endl;

}

RootVegetable::~RootVegetable() {}

// Class Carrot

Carrot::Carrot() {}

Carrot::Carrot(double \_weight) : RootVegetable("Carrot", 22, \_weight) {}

Carrot::~Carrot() {}

// Class Beet

Beet::Beet() {}

Beet::Beet(double \_weight) : RootVegetable("Beet", 22, \_weight) {}

Beet::~Beet() {}

//============================================================

Salad::Salad() {}

double Salad::get\_total\_calories() const

{

double sum = 0.0;

for (auto& it : salad)

{

sum += it->get\_total\_calories();

}

return sum;

}

void Salad::add\_ingredients(Vegetable\* veg)

{

salad.push\_back(veg);

}

void Salad::about\_salad()

{

cout << "Your salad ingredients: ";

for (auto& it : salad)

{

cout << it->get\_name() << "| ";

}

cout << endl << "Total calories: " << get\_total\_calories() << endl;

}

Salad::~Salad() {}

Проект с использованием библиотеки:

## **Исходный код программы**

#include <iostream>

#include "Course\_Project\_OOP\_DLL.h"

int main()

{

Carrot carrot(100);

Beet beet(100);

Potato potato(100);

Tomato tomato(100);

Cucumber cucumber(100);

vector<Vegetable\*> salad;

Salad vinigrette;

vinigrette.add\_ingredients(&carrot);

vinigrette.add\_ingredients(&beet);

vinigrette.add\_ingredients(&potato);

vinigrette.add\_ingredients(&tomato);

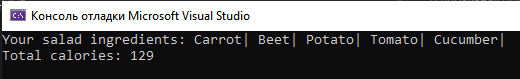
vinigrette.add\_ingredients(&cucumber);

vinigrette.about\_salad();

return 0;

}

## **Результат выполнения программы**



# **ЗАДАНИЕ 3**

## **Постановка задачи**

Цель задания:  Создать класс, соответствующий индивидуальному варианту задания. Создать коллекцию для хранения экземпляров созданного класса. Вид коллекции выбрать самостоятельно. Написать Windows-приложение для работы с этой коллекцией, которое позволит выполнять:

1. добавление элемента в коллекцию с клавиатуры;
2. считывание данных из файла;
3. запись данных в тот же или указанный файл;
4. сортировку данных по различным критериям;
5. поиск элемента по заданному полю;
6. вывод всех элементов, удовлетворяющих заданному условию;
7. удаление элемента из коллекции.

Приложение должно содержать меню, диалоговые окна и предусматривать обработку ошибок.

#### **Вариант 5**

Описать класс с именем AEROFLOT, содержащий следующие поля:

* название пункта назначения рейса;
* номер рейса;
* тип самолета.

Написать программу, выполняющую следующие действия:

* см. общее задание;
* ввод с клавиатуры данных (записи должны быть размещены в алфавитном порядке по названиям пунктов назначения);
* вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры (если таких рейсов нет, вывести соответствующее сообщение).

## **Исходный код программы**

**Header:**

#ifndef MAINWINDOW\_H

#define **MAINWINDOW\_H**

#include <QMainWindow>

#include <QMultiMap>

#include <QStandardItemModel>

#include <QMenuBar>

#include <QMenu>

#include <QFileDialog>

#include <QFile>

#include <QEvent>

#include <QKeyEvent>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*struct* **Aeroflot** {

QString **dest**;

QString **flight\_num** = 0;

**Aeroflot**(QString **d**, QString **fn**) : dest(d), flight\_num(fn) {}

};

*public*:

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

~**MainWindow**();

*protected*:

void ***keyPressEvent***(QKeyEvent \***event**) *override*;

*private* slots:

void **on\_pb\_add\_clicked**();

void **on\_pb\_del\_clicked**();

void **on\_pushButton\_5\_clicked**();

void **on\_pushButton\_6\_clicked**();

void **on\_pb\_open\_clicked**();

void **on\_pb\_save\_clicked**();

*private*:

Ui::MainWindow \***ui**;

QMultiMap<QString, Aeroflot\*> **aerofl**;

QStandardItemModel \***model**;

QStandardItemModel \***modelSearch**;

QString **pathSave**;

QString **path**;

};

#endif *//* *MAINWINDOW\_H*

**CPP:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

model = *new* QStandardItemModel(0, 0, *this*);

model->setHorizontalHeaderItem(0, *new* QStandardItem("Тип самолёта"));

model->setHorizontalHeaderItem(1, *new* QStandardItem("Направление"));

model->setHorizontalHeaderItem(2, *new* QStandardItem("Номер"));

ui->tableView\_2->*setModel*(*model*);

modelSearch = *new* QStandardItemModel(*this*);

modelSearch->setHorizontalHeaderItem(0, *new* QStandardItem("Тип самолёта"));

modelSearch->setHorizontalHeaderItem(1, *new* QStandardItem("Направление"));

modelSearch->setHorizontalHeaderItem(2, *new* QStandardItem("Номер"));

QMenu \***file\_menu** = menuBar()->addMenu("Файл");

file\_menu->addAction("Открыть CTRL+O", *this*, *SLOT*(on\_pb\_open\_clicked()));

file\_menu->addSeparator();

file\_menu->addAction("Сохранить как... CTRL+S", *this*, *SLOT*(on\_pb\_save\_clicked()));

QMenu \***tools\_menu** = menuBar()->addMenu("Инструменты");

tools\_menu->addAction("Добавить", *this*, *SLOT*(on\_pb\_add\_clicked()));

tools\_menu->addSeparator();

tools\_menu->addAction("Удалить", *this*, *SLOT*(on\_pb\_del\_clicked()));

ui->tableView->*setModel*(*modelSearch*);

}

MainWindow::~**MainWindow**()

{

*delete* ui;

}

void MainWindow::**on\_pb\_add\_clicked**()

{

int *const* **row\_count** = model->*rowCount*();

model->insertRow(row\_count);

model->*setData*(model->*index*(row\_count, 3), Qt::*DisplayRole*);

}

void MainWindow::**on\_pb\_del\_clicked**()

{

*if*(!ui->tableView\_2->selectionModel()->selectedIndexes().isEmpty())

{

QModelIndexList **indexList** = ui->tableView\_2->selectionModel()->selectedIndexes();

model->removeRow(indexList.last().row());

}

*else*

{

*return*;

}

}

void MainWindow::**on\_pushButton\_5\_clicked**()

{

ui->tableView\_2->model()->*sort*(0, Qt::*AscendingOrder*);

}

void MainWindow::**on\_pushButton\_6\_clicked**()

{

QString **str** = ui->lineEdit->text();

ui->lineEdit->clear();

*for* (int **index** = 0; index < model->*columnCount*(); ++index)

{

QList<QStandardItem\*> **items** = model->findItems(str,Qt::*MatchExactly*, index);

int **count** = items.count();

*if* (count > 0)

{

*for* (int **k** = 0; k < count; ++k)

{

QModelIndex **modelIndex** = model->indexFromItem(items[k]);

QModelIndex **index1** = model->*index*(modelIndex.row(), 0);

QModelIndex **index2** = model->*index*(modelIndex.row(), 1);

QModelIndex **index3** = model->*index*(modelIndex.row(), 2);

QStandardItem \***item\_col\_1** = *new* QStandardItem(index1.data(Qt::*DisplayRole*).toString());

QStandardItem \***item\_col\_2** = *new* QStandardItem(index2.data(Qt::*DisplayRole*).toString());

QStandardItem \***item\_col\_3** = *new* QStandardItem(index3.data(Qt::*DisplayRole*).toString());

modelSearch->appendRow(QList<QStandardItem\*>() << item\_col\_1 << item\_col\_2 << item\_col\_3);

qDebug() << "Column = " << index << "Row = " << modelIndex.row();

}

}

}

}

void MainWindow::**on\_pb\_open\_clicked**()

{

path = QFileDialog::getOpenFileName();

*if* (path.isEmpty())

{

*return*;

}

QFile **file**(path);

*if* (file.*open*(QIODevice::*ReadOnly*))

{

int **lineindex** = 0; *//* *file* *line* *counter*

QTextStream **in**(&*file*); *//* *read* *to* *text* *stream*

*while* (!in.atEnd()) {

QString **fileLine** = in.readLine();

QStringList **lineToken** = fileLine.split(";", Qt::*SkipEmptyParts*);

*//* *load* *parsed* *data* *to* *model* *accordingly*

*for* (int **j** = 0; j < lineToken.size(); j++) {

QString **value** = lineToken.at(j);

QStandardItem \***item** = *new* QStandardItem(value);

model->setItem(lineindex, j, *item*);

}

lineindex++;

}

file.*close*();

}

}

void MainWindow::**on\_pb\_save\_clicked**()

{

QString **filters**("CSV files (\*.csv);;All files (\*.\*)");

QString **defaultFilter**("CSV files (\*.csv)");

QString **fileName** = QFileDialog::getSaveFileName(0, "Save file", QCoreApplication::applicationDirPath(),

filters, &*defaultFilter*);

QFile **file**(fileName);

QAbstractItemModel \***model** = ui->tableView\_2->model();

*if* (file.*open*(QFile::*WriteOnly* | QFile::*Truncate*)) {

QTextStream **data**(&*file*);

QStringList **strList**;

*//data* *<<* *strList.join(";")* *<<* *"\n";*

*for* (int **i** = 0; i < model->*rowCount*(); i++)

{

strList.clear();

*for* (int **j** = 0; j < model->*columnCount*(); j++) {

*if* (model->*data*(model->*index*(i, j)).toString().length() > 0)

strList.append(model->*data*(model->*index*(i, j)).toString());

*else*

strList.append("");

}

data << strList.join(";") + "\n";

}

file.*close*();

}

}

void MainWindow::***keyPressEvent***(QKeyEvent \***event**)

{

*switch* (event->key()) {

*case* Qt::*Key\_O*:

*if* (event->modifiers() & Qt::*ControlModifier*)

{

*this*->on\_pb\_open\_clicked();

}

*break*;

*case* Qt::*Key\_S*:

*if* (event->modifiers() & Qt::*ControlModifier*)

{

*this*->on\_pb\_save\_clicked();

}

*break*;

*case* Qt::*Key\_Q*:

*if* (event->modifiers() & Qt::*ControlModifier*)

{

exit(0);

}

*break*;

*default*:

*break*;

}

}

**Main:**

#include "mainwindow.h"

#include <QApplication>

int **main**(int **argc**, char \***argv**[])

{

QApplication **a**(*argc*, *argv*);

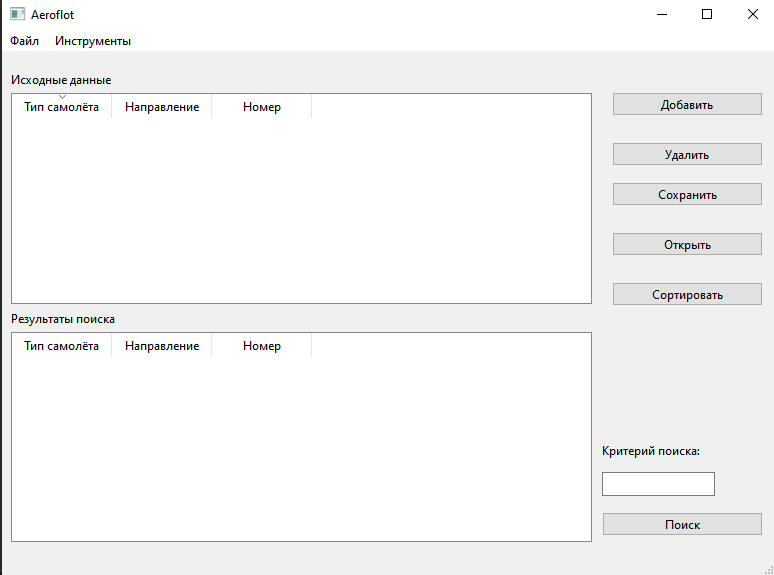
MainWindow **w**;

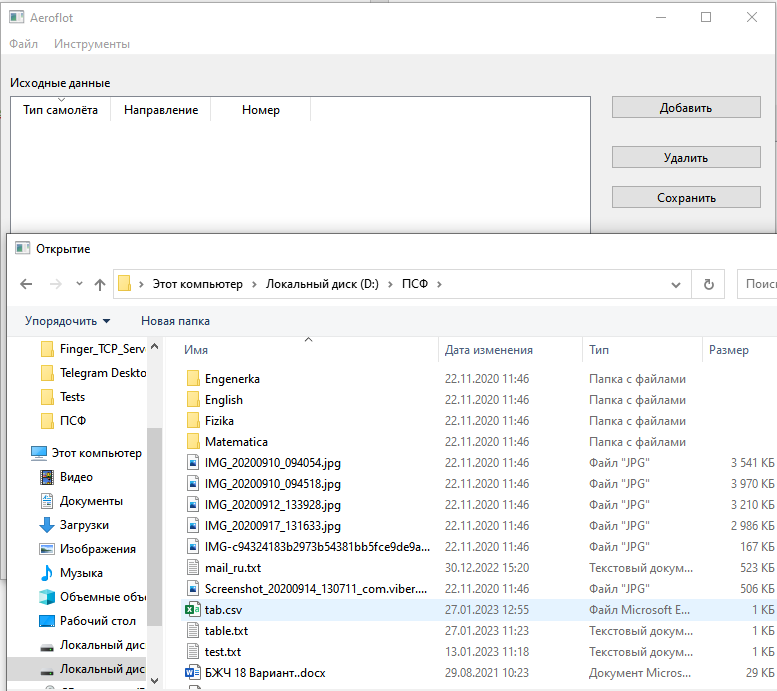
w.show();

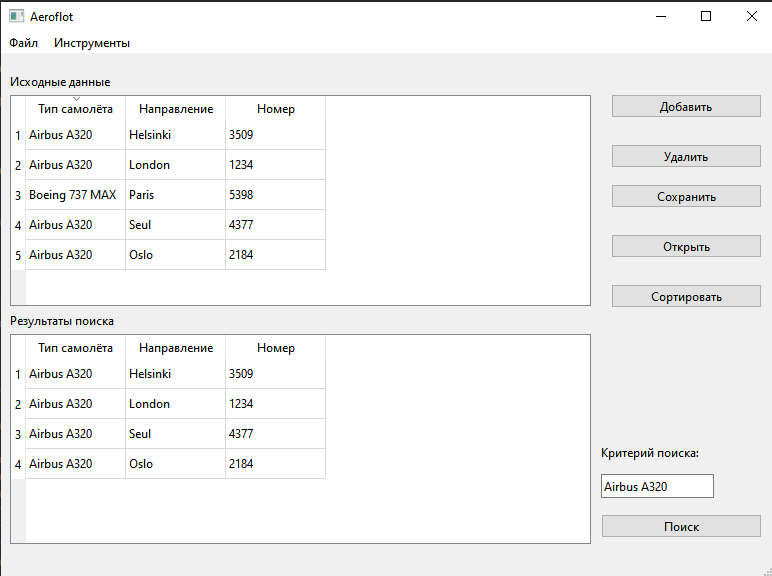
*return* a.exec();

}

## **Результат выполнения программы**







# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были изучены основные принципы ООП, такие как: абстракция, наследование, полиморфизм и инкапсуляция. Закреплены практические навыки разработки программ на языке C# (построение алгоритма программы, составление на его основе исходного текста программы, откладка программы, получение и анализ результатов работы программы). Таким образом, данная работа показала, что язык C# является необходимым инструментальным средством для решения сложных инженерно-технических задач.

# **ЛИТЕРАТУРА**

1. Шилдт, Герберт С# 4.0 Полное руководство. : Пер. с английского. М. – ООО «И.Д. Вильямс», 2011 – 1056 с. : ил.
2. Уотсон, К. Visual С# 2010: полный курс.: Пер. с англ. - М.: ООО "И.Д. Вильяме", 2011. - 960 с. : ил.
3. Троелсен, Эндрю. Язык программирования С# 2010 и платформа .NET 4.0, 5-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильяме", 2011. — 1392 с. : ил.
4. Нейгел, Уотсон C# 5.0 и платформа .NET\_2014
5. Троелсен Э. - Язык программирования C# 5.0