Федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича»

(СПбГУТ)

Кафедра Программной инженерии и вычислительной техники

**Курсовая работа по дисциплине**

**«Объектно-ориентированное программирование»**

**Работу выполнил:**

Студент 2-го курса

Группа ИКПИ-73

Сударев Андрей Владимирович

**Приняла:**

Петрова Ольга Борисовна

Санкт-Петербург

2018 уч. г.

Оглавление

1. ЦЕЛЬ РАБОТЫ 3

2. ОБОСНОВАНИЕ ВЫБРАННОГО КОНТЕЙНЕРА 3

3. РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ 4

4. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММЫ 8

5. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ 18

6. ЗАКЛЮЧЕНИЕ 21

7. СПИСОК ЛИТЕРАТУРЫ 22

**Цель работы**

Создать базу данных, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа контейнера можно использовать контейнеры vector или list из библиотеки STL. Для работы с контейнером следует предусмотреть итератор. Остальные требования к контейнеру определяются студентом самостоятельно.

Разрабатываемая база данных должна быть создана использованием C++ и библиотеки Qt4. Среда программирования Qt Creator.

В качестве элементов контейнеров должны использоваться объекты классов.

База данных должна обеспечивать выполнение следующих операций

* создание базы данных,
* объединение баз данных,
* добавления и удаления записей,
* редактирования записей,
* просмотра базы данных,
* поиск данных (найти данные, удовлетворяющие заданному критерию),
* запись на диск,
* чтения с диска.

Интерфейсная часть программы должна содержать следующие компоненты:

* окно «ABOUT»,
* главное меню,
* всплывающее меню,
* строку состояния,
* панель инструментов,
* подсказки по командам меню и панели инструментов.

**Вариант 20**

Класс, характеризующий планеты солнечной системы.

**Обоснование выбранного контейнера**

В качестве прототипа разрабатываемого контейнера я решил использовать контейнер vector из библиотеки STL. Выбор данного контейнера объясняется простотой своей разработки и организации, а также возможностью произвольного доступа к своим элементам, чего нет у контейнера типа list. Контейнер list обладает только последовательным доступом.

**Разработка структуры программы**

Программа должна состоять из графического интерфейса и кода, который остается «за кадром». Интерфейс составляют: элемент отображения LineEdit, панель инструментов, главное меню и функциональные кнопки PushButton.

Фоновую часть составляют классы: контейнер VectPlanet, содержимое контейнера - класс планет Planet, и сама база данных Planets, которая хранит в себе данные и содержит методы обработки базы.

**Таблица идентификаторов**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Файлы** | **Классы** | **Методы класса** | **Функция** | **Назначение** |
| Basevect.h | VectPlanet | VectPlanet () | Конструктор | Контейнер |
| ~ VectPlanet () | Деструктор |
| VectPlanet(const VectPlanet&) | Конструктор копирования |
| VectPlanet(const Planet\*, unsigned int) | Конструктор, который копирует планеты из указанного массива |
| void add\_planet () | Добавить элемент по умолчанию в конец |
| void add\_planet(const Planet&) | Добавить указанную планету в конец |
| void delete\_planet(int) | Убрать элемент с указанным номером |
| void new\_size(int) | Удаляем все планеты и устанавливаем новый размер контейнера |
| void add\_size(int) | Добавляем указанный размер памяти к вектору с сохранением планет |
| Planet& get\_planet(const int&) | Получить адрес указанной планеты |
| get\_size() | Получить размер контейнера |
| Iterator begin() | Получить указатель на начало контейнера |
| Iterator end() | Получить указатель на фиктивный конец контейнера |
| Planet& operator[](const int&) | Перегруженный оператор индексирования |
| Iterator | Iterator() | Конструктор | Итератор |
| ~Iterator() | Деструктор |
| Planet& operator \* () | Перегруженный оператор разыменования |
| Planet& operator ++ () | Перегруженный оператор инкремента |
| Planet& operator -- () | Перегруженный оператор декремента |
| bool operator == (const Planet& it) | Перегруженный оператор равенства |
| bool operator != (const Planet& it) | Перегруженный оператор неравенства |
| friend ostream& operator<< | Перегруженный оператор вывода в поток |
| friend istream& operator>> | Перегруженный оператор ввода в поток |
| Planet.h | Planet | Planet () | Конструктор | Класс |
| Planet (string, int, int, string) | Конструктор с параметрами |
| Planet (const Planet &) | Конструктор копирования |
| ~ Planet() | Деструктор |
| string get\_name(void) | Получить имя объекта |
| string get\_Who\_opening(void) | Получить имя первооткрывателя |
| int get\_Year\_of\_opening(void) | Получить год открытия |
| int get\_mass(void) | Получить массу объекта |
| void in\_name(string) | Ввести имя объекта |
| void in\_Who\_opening(string) | Ввести имя первооткрывателя |
| void in\_Year\_of\_opening(int); | Установить год открытия |
| void in\_mass(int); | Ввести массу объекта |
| friend istream& operator>> | Перегруженный оператор ввода в поток |
| mainwindow.h | MainWindow | void on\_add\_item\_clicked() | Добавление новой планеты в контейнер | Слоты |
| void on\_delete\_item\_clicked() | Удаление планеты из контейнера |
| void on\_pushButton\_SelectPlanet\_clicked() | Выбрать планету для просмотра и редактирования |
| void on\_pushButton\_ConfChang\_clicked() | Применить изменения |
| void on\_pushButton\_sizeVictor\_clickd() | Показать количество планет в контейнере |
| void on\_Author\_triggered(); | Показать информацию о разработчике |
| void on\_action\_triggered(); | Записать планеты из контейнера в файл |
| void on\_action\_2\_triggered(); | Записать планеты в контейнер из файла, старые планеты удаляются |
| void on\_Merge\_in\_file\_triggered() | Планеты из контейнера объединяются с планетами в файле |
| void on\_Merge\_from\_file\_triggered() | Планеты из файла объединяются с планетами в контейнере |
| void on\_pushButton\_back\_clicked() | Показать предыдущую планету |
| void on\_pushButton\_next\_clicked() | Показать следующую планету |
|  |  |

**Поля классов**

**VectPlanet:**

* Planet \* T; – начало вектора
* int size – размер списка

**Planet:**

* string name – имя объекта
* string Who\_opening – имя первооткрывателя
* int mass - масса
* int Year\_of\_opening – год открытия

**MainWindow:**

* Ui::MainWindow \*ui
* QMenu \*Menu – меню
* QAction \*action – действие “Сохранить в файл”
* QAction \*action\_2 – действие “Открыть из файла”
* QAction \*Merge\_in\_file – действие “Добавить в файл”
* QAction \*Merge\_from\_file – действие “Объединить из файла”
* QMenu \*about\_of – меню
* QAction \*Author – действие “Информация об авторе”
* QLineEdit \* lineEdit\_EnterPlanet– строка для ввода номера планеты
* QLineEdit \* lineEdit\_Mass– строка для работы с массой планеты
* QLineEdit \* lineEdit\_Mass\_2– заблокированная строка с надписью «Идентификатор»
* QLineEdit \* lineEdit\_Name– строка для работы с именем планеты
* QLineEdit \* lineEdit\_Name\_2– заблокированная строка с надписью «Имя»
* QLineEdit \* lineEdit\_WhoOpen– строка для работы с первооткрывателем планеты
* QLineEdit \* lineEdit\_WhoOpen\_2– заблокированная строка с надписью «Кто открыл»
* QLineEdit \* lineEdit\_YearOpen– строка для работы с годом открытия планеты
* QLineEdit \* lineEdit\_YearOpen\_2– заблокированная строка с надписью «Год открытия»
* QPushButton \* add\_item– кнопка добавления новой планеты
* QPushButton \* delete\_item– кнопка удаления планеты
* QPushButton \* pushButton\_ConfChang– кнопка применения изменений
* QPushButton \* pushButton\_SelectPlanet– кнопка выбора планеты
* QPushButton \* pushButton\_back– кнопка для выбора предыдущей планеты
* QPushButton \* pushButton\_next– кнопка для выбора следующей планеты
* QPushButton \* pushButton\_sizeVictor– кнопка для показа количества планет

**Разработка модулей программы**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**basevect.h**

#ifndef BASEVECT\_H

#define BASEVECT\_H

class VectPlanet

{

private:

Planet \* T;

int size;

public:

**VectPlanet**();

VectPlanet(const Planet\*, unsigned int);

**VectPlanet**(const VectPlanet&);

~**VectPlanet**();

void add\_planet(const Planet&);

void **add\_planet**(void);

void **delete\_planet**(int);

void **new\_size**(int); // Удаляем все планеты и устанавливаем новый размер контейнера

void **add\_size**(int); // Добавляем указанный размер памяти к вектору с сохранением планет

Planet& get\_planet(const int&);

int **get\_size**();

Planet& operator[](const int&);

class Iterator

{

private:

Planet\* cur;

public:

**Iterator**() : **cur**(**NULL**)

{

}

Iterator(Planet\* first) : cur(first)

{

}

~**Iterator**()

{

delete cur;

}

string get\_name() { return cur->get\_name(); }

string get\_Who\_opening() { return cur->get\_Who\_opening(); }

int **get\_Year\_of\_opening**() { return cur->get\_Year\_of\_opening(); }

int **get\_mass**() { return cur->get\_mass(); }

Planet& operator+ (int n) { return \*(cur + n); }

Planet& operator- (int n) { return \*(cur - n); }

Planet& operator++ (int) { return \*cur++; }

Planet& operator+= (int add) { return \*(cur+=add); }

Planet& operator-- (int) { return \*cur--; }

Planet& operator++ () { return \*++cur; }

Planet& operator-- () { return \*--cur; }

void operator=(Planet& \_planet)

{

cur->in\_mass(\_planet.get\_mass());

cur->in\_name(\_planet.get\_name());

cur->in\_Year\_of\_opening(\_planet.get\_Year\_of\_opening());

cur->in\_Who\_opening(\_planet.get\_Who\_opening());

}

bool operator!= (const Iterator& it) { return cur != it.cur; }

bool operator== (const Iterator& it) { return cur == it.cur; }

Planet& operator\* () { return \*cur; }

friend ostream& operator<< (ostream &out, const Iterator planeta)

{

out << planeta.cur->get\_name() << endl

<< planeta.cur->get\_mass() << endl

<< planeta.cur->get\_Year\_of\_opening() << endl

<< planeta.cur->get\_Who\_opening() << endl;

return out;

}

friend istream& operator>> (istream &in, Iterator& planeta)

{

in >> planeta;

return in;

}

};

Iterator **begin**() { return T; };

Iterator **end**() { return T + size; };

};

VectPlanet::**VectPlanet**()//: T(new Planet[1]), size(1)

{

T = new Planet[1];

size = 1;

}

VectPlanet::VectPlanet(const Planet\* mass, unsigned int inp\_size) : T(new Planet[inp\_size]), size(inp\_size)

{

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

T[i] = mass[i];

}

VectPlanet::VectPlanet(const VectPlanet& cpy) : T(new Planet[cpy.size]), size(cpy.size)

{

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

T[i] = cpy.T[i];

}

VectPlanet::~VectPlanet()

{

delete[] T;

}

void VectPlanet::add\_planet(void)

{

size++;

Planet \*arr = new Planet[size];

for (int i = 0; i < size - 1; i++)

arr[i] = T[i];

delete[] T;

T = arr;

}

void VectPlanet::add\_planet(const Planet& add\_plan)

{

size++;

Planet \*arr = new Planet[size];

for (int i = 0; i < size - 1; i++)

arr[i] = T[i];

arr[size - 1] = add\_plan;

delete[] T;

T = arr;

}

void VectPlanet::delete\_planet(int numb\_del\_planet = -1) //номер удаляемой планеты, если не указан, то удаляется последняя в массиве

{

if (numb\_del\_planet <= -1 || numb\_del\_planet >= size)

{

size -= 1;

Planet \*arr = new Planet[size];

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

arr[i] = T[i];

delete[] T;

T = arr;

}

else

{

Planet \*arr = new Planet[size - 1];

int i = 0;

for (; i < numb\_del\_planet; i++)

arr[i] = T[i];

for (int j = i + 1; j < size; j++)

arr[i] = T[j];

size -= 1;

delete[] T;

T = arr;

}

}

Planet& VectPlanet::get\_planet(const int & number)

{

if (number < size)

return T[number];

else

return T[0];

}

int VectPlanet::get\_size()

{

return size;

}

Planet& VectPlanet::operator[] (const int & number)

{

return T[number];

}

void VectPlanet::new\_size(int inp)

{

size = inp;

delete[] T;

Planet \*arr = new Planet[size];

T = arr;

}

void VectPlanet::add\_size(int add)

{

Planet \*arr = new Planet[size + add];

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

arr[i] = T[i];

size += add;

delete[] T;

T = arr;

}

#endif // BASEVECT\_H

**Planet.h**

#ifndef PLANET\_H

#define PLANET\_H

class Planet

{

private:

string name;

string Who\_opening;

int mass;

int Year\_of\_opening;

public:

**Planet**();

Planet(string, int, int, string);

**Planet**(const Planet&);

~**Planet**();

string get\_name(void);

string get\_Who\_opening(void);

int **get\_Year\_of\_opening**(void);

int **get\_mass**(void);

void in\_name(string);

void in\_Who\_opening(string);

void **in\_Year\_of\_opening**(int);

void **in\_mass**(int);

friend istream& operator>> (istream &in, Planet &point);

};

Planet::**Planet**() : **name**("no name"), mass(-1), **Who\_opening**("ANNOnymous"), Year\_of\_opening(2205)

{

}

Planet::Planet(string name\_inp, int mass\_inp, int Year, string Who) : name(name\_inp), mass(mass\_inp), Year\_of\_opening(Year), Who\_opening(Who)

{

}

Planet::**Planet**(const Planet& planet\_inp) : **name**(**planet\_inp**.**name**), mass(planet\_inp.mass), Year\_of\_opening(planet\_inp.Year\_of\_opening), **Who\_opening**(**planet\_inp**.**Who\_opening**)

{

}

Planet::~**Planet**()

{

}

string Planet::get\_name(void)

{

return name;

}

string Planet::get\_Who\_opening(void)

{

return Who\_opening;

}

int Planet::**get\_Year\_of\_opening**(void)

{

return Year\_of\_opening;

}

int Planet::**get\_mass**(void)

{

return mass;

}

void Planet::in\_name(string inp\_name)

{

name = inp\_name;

}

void Planet::**in\_mass**(int inp\_mass)

{

mass = inp\_mass;

}

istream& operator>> (istream &in, Planet &point)

{

in >> point.name;

in >> point.mass;

in >> point.Year\_of\_opening;

in >> point.Who\_opening;

}

#endif // PLANET\_H

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QStandardItemModel>

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

private slots:

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

void **on\_delete\_item\_clicked**();

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

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

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

void **on\_Author\_triggered**();

void **on\_action\_triggered**();

void **on\_action\_2\_triggered**();

void **on\_Merge\_in\_file\_triggered**();

void **on\_Merge\_from\_file\_triggered**();

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

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

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QString>

using namespace std;

#include "Planet.h"

#include "basevect.h"

#include <QMessageBox>

#include <fstream>

Planet

Earth("Земля", 1, 0, "Human"),

Mars("Марс", 2, -2000, "Unknown"),

Venera("Венера", 3, 1610, "Galileo\_Galilei"),

Mercyri("Меркурий", 4, -3000, "the\_Sumerians"),

Upiter("Юпитер", 5, 1610, "Galileo\_Galilei"),

Saturn("Сатурн", 6, 1659, "Christiaan\_Huygens"),

Uran("Уран", 7, 1781, "William\_Herschel"),

Neptun("Нептун", 8, 1846, "Team"),

Pluton("Плутон", 9, 1930, "Clyde\_William");

Planet mass[] = { Earth, Mars, Venera, Mercyri, Upiter, Saturn, Uran, Neptun, Pluton };

VectPlanet Planets(mass, 9);

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

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

{

delete ui;

}

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

{

Planets.add\_planet();

}

void MainWindow::**on\_delete\_item\_clicked**()

{

Planets.delete\_planet(ui->lineEdit\_EnterPlanet->text().toInt());

}

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

{

int number\_planet = ui->lineEdit\_EnterPlanet->text().toInt();

if(number\_planet < Planets.get\_size() && number\_planet > -1)

{

ui->lineEdit\_EnterPlanet->setDisabled(true);

ui->lineEdit\_Mass->setEnabled(true);

ui->lineEdit\_Name->setEnabled(true);

ui->lineEdit\_WhoOpen->setEnabled(true);

ui->lineEdit\_YearOpen->setEnabled(true);

VectPlanet::Iterator iter;

QString T;

iter = Planets.begin();

iter += number\_planet;

T = QString::fromStdString(iter.get\_name());

ui->lineEdit\_Name->setText(T);

T = QString::fromStdString(iter.get\_Who\_opening());

ui->lineEdit\_WhoOpen->setText(T);

T.sprintf("%d", iter.get\_mass());

ui->lineEdit\_Mass->setText(T);

T.sprintf("%d", iter.get\_Year\_of\_opening());

ui->lineEdit\_YearOpen->setText(T);

}

else

{

ui->lineEdit\_Name->setText("Не существует");

ui->lineEdit\_Mass->setText("-1");

ui->lineEdit\_WhoOpen->setText("Не существует");

ui->lineEdit\_YearOpen->setText("-1");

}

}

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

{

int number\_planet = ui->lineEdit\_EnterPlanet->text().toInt();

if(number\_planet < Planets.get\_size() && number\_planet > -1 && ui->lineEdit\_Name->isEnabled())

{

ui->lineEdit\_EnterPlanet->setEnabled(true);

ui->lineEdit\_Mass->setDisabled(true);

ui->lineEdit\_Name->setDisabled(true);

ui->lineEdit\_WhoOpen->setDisabled(true);

ui->lineEdit\_YearOpen->setDisabled(true);

Planet tmp( ui->lineEdit\_Name->text().toStdString(),

ui->lineEdit\_Mass->text().toInt(),

ui->lineEdit\_YearOpen->text().toInt(),

ui->lineEdit\_WhoOpen->text().toStdString());

VectPlanet::Iterator iter;

iter = Planets.begin();

iter += ui->lineEdit\_EnterPlanet->text().toInt();

\*iter = tmp;

}

}

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

{

QString SizeVector;

SizeVector.sprintf("%d", Planets.get\_size());

ui->lineEdit\_EnterPlanet->setText(SizeVector);

}

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

{

QMessageBox::information(this,

"О программе:",

"Выполнил:\n"

"Студент 2-го курса\n"

"группы ИКПИ-73\n"

"Сударев Андрей Владимирович",

QMessageBox::Ok);

}

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

{

ofstream fout;

fout.open("Planets.txt");

fout << Planets.get\_size() << endl;

for(VectPlanet::Iterator iter = Planets.begin(); iter != Planets.end(); iter++)

fout << iter;

fout.close();

}

void MainWindow::**on\_action\_2\_triggered**()

{

ifstream fin;

fin.open("Planets.txt");

if(fin.is\_open())

{

int size = Planets.get\_size();

fin >> size;

Planets.new\_size(size);

for(VectPlanet::Iterator iter = Planets.begin(); iter != Planets.end(); iter++)

fin >> \*iter;

}

}

void MainWindow::**on\_Merge\_in\_file\_triggered**()

{

ifstream fin("Planets.txt");

if(fin.is\_open())

{

int size;

fin >> size;

VectPlanet tmp;

tmp.new\_size(size);

for(VectPlanet::Iterator iter = tmp.begin(); iter != tmp.end(); iter++)

fin >> \*iter;

fin.close();

ofstream fout("Planets.txt");

fout << size + Planets.get\_size() << endl;

for(VectPlanet::Iterator iter = tmp.begin(); iter != tmp.end(); iter++)

fout << iter;

for(VectPlanet::Iterator iter = Planets.begin(); iter != Planets.end(); iter++)

fout << iter;

fout.close();

}

else

{

ofstream fout;

fout.open("Planets.txt");

fout << Planets.get\_size() << endl;

for(VectPlanet::Iterator iter = Planets.begin(); iter != Planets.end(); iter++)

fout << iter;

fout.close();

}

}

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

{

ifstream fin("Planets.txt");

if(fin.is\_open())

{

int size, old\_size;

old\_size = Planets.get\_size();

fin >> size;

Planets.add\_size(size);

VectPlanet::Iterator iter = Planets.begin();

iter += old\_size;

for(; iter != Planets.end(); iter++)

fin >> \*iter;

}

}

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

{

ui->lineEdit\_EnterPlanet->setEnabled(true);

ui->lineEdit\_Mass->setDisabled(true);

ui->lineEdit\_Name->setDisabled(true);

ui->lineEdit\_WhoOpen->setDisabled(true);

ui->lineEdit\_YearOpen->setDisabled(true);

int number = ui->lineEdit\_EnterPlanet->text().toInt() - 1;

QString NumberVector;

NumberVector.sprintf("%d", number);

ui->lineEdit\_EnterPlanet->setText(NumberVector);

if(number < Planets.get\_size() && number > -1)

{

VectPlanet::Iterator iter;

QString T;

iter = Planets.begin();

iter += number;

T = QString::fromStdString(iter.get\_name());

ui->lineEdit\_Name->setText(T);

T = QString::fromStdString(iter.get\_Who\_opening());

ui->lineEdit\_WhoOpen->setText(T);

T.sprintf("%d", iter.get\_mass());

ui->lineEdit\_Mass->setText(T);

T.sprintf("%d", iter.get\_Year\_of\_opening());

ui->lineEdit\_YearOpen->setText(T);

}

else

{

ui->lineEdit\_Name->setText("Не существует");

ui->lineEdit\_Mass->setText("-1");

ui->lineEdit\_WhoOpen->setText("Не существует");

ui->lineEdit\_YearOpen->setText("-1");

}

}

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

{

ui->lineEdit\_EnterPlanet->setEnabled(true);

ui->lineEdit\_Mass->setDisabled(true);

ui->lineEdit\_Name->setDisabled(true);

ui->lineEdit\_WhoOpen->setDisabled(true);

ui->lineEdit\_YearOpen->setDisabled(true);

int number = ui->lineEdit\_EnterPlanet->text().toInt() + 1;

QString NumberVector;

NumberVector.sprintf("%d", number);

ui->lineEdit\_EnterPlanet->setText(NumberVector);

if(number < Planets.get\_size() && number > -1)

{

VectPlanet::Iterator iter;

QString T;

iter = Planets.begin();

iter += number;

T = QString::fromStdString(iter.get\_name());

ui->lineEdit\_Name->setText(T);

T = QString::fromStdString(iter.get\_Who\_opening());

ui->lineEdit\_WhoOpen->setText(T);

T.sprintf("%d", iter.get\_mass());

ui->lineEdit\_Mass->setText(T);

T.sprintf("%d", iter.get\_Year\_of\_opening());

ui->lineEdit\_YearOpen->setText(T);

}

else

{

ui->lineEdit\_Name->setText("Не существует");

ui->lineEdit\_Mass->setText("-1");

ui->lineEdit\_WhoOpen->setText("Не существует");

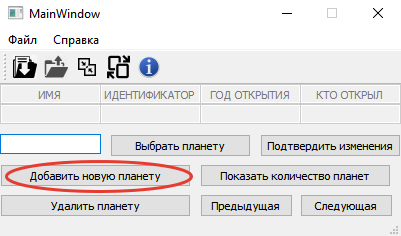
ui->lineEdit\_YearOpen->setText("-1");

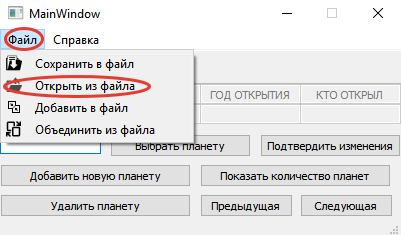
}

}

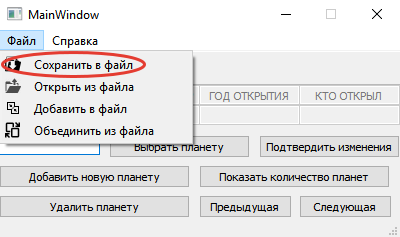
**Инструкция пользователя**

**1. Добавить новую планету или загрузить их из файла**

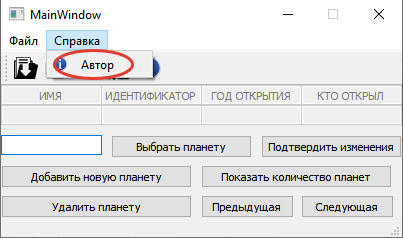


****

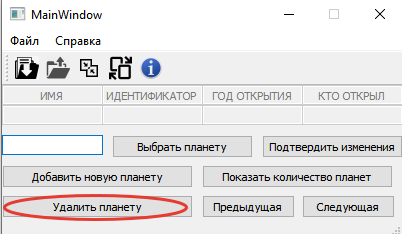
**2. Сохранить планеты в файл**

****

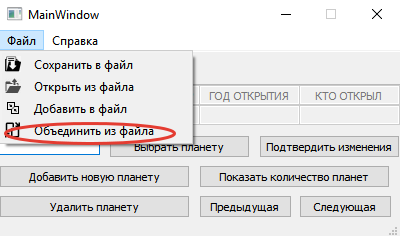
**3. Информация о программе**

****

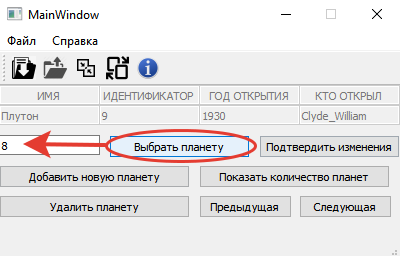
**4. Удалить элемент**

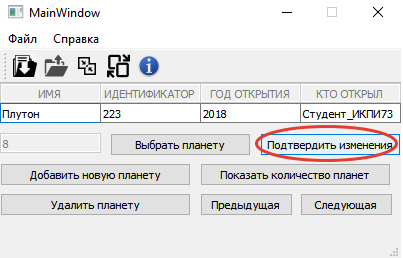
****

**5. Добавить элементы из другого контейнера**

****

**6. Редактирование планеты**

****

****

**Заключение**

Была разработана программа, обеспечивающая выполнение всех функций, объявленных в ТЗ. Также присутствует интерфейс, соответствующий требованиям.

Во время выполнения работы возникало большое количество трудностей. Они разрешались с помощью обращения к официальной документации. Также параллельно изучалась стандартная библиотека шаблонов (STL).

Данная работа дает большое количество опыта в разработке программного обеспечения. Также помогает лучше изучить структуры хранения данных.

**Список литературы**

* Макс Шлее – “Qt4.5. Профессиональное программирование на C++”
* Жасмин Бланшет – “Qt4. Программирование GUI на C++”