МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ

им. проф. М. А. Бонч-Бруевича

Факультет ИКСС

**Курсовая работа**

по дисциплине

«Объектно-ориентированное программирование»

По теме:

**«Класс, характеризующий книгу»**

Вариант 6

Выполнил: Кузьмин А.А.

Курс: Третий

Группа: ИКПИ-85

Специальность: Программная инженерия

Принимает: Коробов. С.А.

Санкт-Петербург

2020

**Содержание**

[Задача 3](#_Toc528877971)

[Выбор и обоснование типа разрабатываемого контейнера 5](#_Toc528877972)

[Выбор и обоснование используемых классов 6](#_Toc528877973)

[Разработка собственных классов 7](#_Toc528877974)

[Разработка Интерфейс пользователя 9](#_Toc528877975)

[Требования к составу и параметрам технических средств 10](#_Toc528877976)

[Вывод 11](#_Toc528877977)

[Литература 12](#_Toc528877978)

[Приложение А – Листинг программы 13](#_Toc528877979)

[Файлы реализации 13](#_Toc528877980)

[Adddialog.cpp 13](#_Toc528877981)

[database.cpp 13](#_Toc528877982)

[main.cpp 16](#_Toc528877984)

[mainwindow.cpp 16](#_Toc528877985)

[Заголовочные файлы 20](#_Toc528877986)

[adddialog.h 20](#_Toc528877987)

[database.h 20](#_Toc528877988)

[mainwindow.h 21](#_Toc528877989)

[kniga.h 21](#_Toc528877990)

[vector.h 22](#_Toc528877991)

[Приложение Б – Руководство пользователя 25](#_Toc528877992)

[Работа с базами данных 25](#_Toc528877993)

[Работа с записями 26](#_Toc528877994)

1.Задача

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, которые есть в библиотеки STL. Для работы с контейнером следует предусмотреть итератор.

Оставшиеся требования к контейнеру определяются студентом самостоятельно.

Разрабатываемая база данных должна быть создана использованием C++ и библиотеки Qt4. Взятые из среды программировании Qt Creater.

В качестве элементов контейнеров должны использоваться объекты класса, характеризующего книгу (автор, название, описание, жанр и т.д.).

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.

Пояснительная записка должна содержать:

* постановку задачи;
* выбор и обоснование типа разрабатываемого контейнера;
* разработку структуры программы;
* разработку отдельных модулей;
* выбор и обоснование используемых классов;
* разработку собственных классов;
* разработку интерфейса пользователя;
* инструкцию пользователя;
* заключение;
* список литературы.

Все решения, принятые в процессе разработки должны быть обоснованы. В приложении к пояснительной записке необходимо привести распечатку разработанной программы.

2 Выбор и обоснование типа разрабатываемого контейнера

Одной из задач данной курсовой работы является разработка контейнера для хранения записей в базе данных. В качестве прототипа контейнера предлагалось использовать контейнеры vector или list из библиотеки STL.

Для того чтобы выбрать контейнер, который будет служить прототипом необходимо разобраться в отличиях между контейнерами vector и list.

Вектор-это контейнер с произвольным доступом, оптимизированный таким образом, чтобы обеспечить быстрый доступ к его элементам по индексу. Список - это контейнер с последовательным доступом, разработанный для обеспечения оптимального выполнения частых вставок и удалений элементов.

Для операции редактирования записей в базе данных требуется произвольный доступ к элементам контейнера, поэтому в качестве прототипа собственного контейнера я взял контейнер vector.

Разработка структуры программы:

Разрабатываемая программа состоит из графического пользовательского интерфейса созданного при помощи библиотеки Qt и кода, отвечающего за обработку данных и управление базой данных.

Для обработки данных и управления базой данных был разработан класс database отвечающий за управление базой данных, структура Kniga для хранения данных о книгах и контейнер vector для хранения записей в базе данных.

3 Выбор и обоснование используемых классов

MainWindow — класс для работы с пользовательским интерфейсом.

AddDialog — класс для добавления и редактирования записей в базе.

database — класс для управления базой данных.

vector — класс-контейнер выбранного типа.

Kniga — структура для хранения данных о книгах.

4 Разработка собственных классов

MainWindow

Класс для работы с пользовательским интерфейсом.

Содержит поля:

Ui::MainWindow \*ui; - для обращение к элементам окна

database \*db; - база данных

Методы:

void tablereset(); - очистка таблицы

void on\_actionAbout\_triggered(); - отображение окна About

void onfindlinechanged(QString); - поиск записи в базе данных

void on\_actionNew\_triggered(); - создание новой базы данных

void on\_actionAdd\_triggered(); - добавление записи в базу данных

void on\_actionChange\_triggered(); -изменение записи в базе данных

void on\_actionDelete\_triggered(); - удаление записи из базы данных

void on\_actionExit\_triggered(); - выход из программы

void on\_actionSave\_triggered(); - сохранение базы данных

void on\_actionOpen\_triggered(); - открытие базы данных

void on\_actionSave\_as\_triggered(); - сохранение базы данных как

void on\_actionMerge\_triggered(); - объединение баз данных

void on\_actionSearch\_triggered(); - отображение строки поиска

void on\_actionClosefind\_triggered(); - скрыть строку поиска

void closeEvent(QCloseEvent \*); - действие при выходе из программы

AddDialog — класс для добавления и редактирования записей в базе.

Содержит поля:

Ui::AddDialog \*ui; - для обращения к элементам окна

Методы:

Kniga getdata(); - возвращает объект kniga содержащий значения полей окна

database — класс для управления базой данных.

Содержит поля:

Vector <kniga> v; - контейнер для хранения элементов базы данных

bool saved; - хранение состояния базы данных (сохранена или нет)

Методы:

void find (QString s,QTableWidget \*t); - поиск по базе данных

void saveToFile(const QString &fileName); - сохранение в файл

void loadFromFile(const QString &fileName); - чтение из файла

void add(const banking& t); - добавление записей в базу

void change(int n); - изменение базы

void del(int k); - удаление базы

void addtotable(QTableWidget \*t); - вывод элементов базы в таблицу

void merge(const QString &fileName); - объединение баз данных

void clear(); - очистка базы данных

bool isSaved(); - возвращает состояние базы данных

vector — класс-контейнер выбранного типа.

Содержит поля:

T \*vfirst; - указатель на первый элемент контейнера

int size\_; - количество элементов, содержащихся в контейнере

int capacity\_; - количество выделенной памяти

Методы:

vector(); - конструктор контейнера

void push\_back(const T& t); - добавление элемента в конец

void del(int index); - удаление элемента

T& operator[] (int i) const; - доступ к элементу по индексу

const T& last (); - возвращает последний элемента в контейнере

iterator begin (); - возвращает указатель на первый элемент контейнера

iterator end (); - возвращает указатель на последний элемент контейнера

int size() const; - возвращает количество элементов содержащихся в контейнере

bool empty() const; - возвращает true если контейнер пустой false если нет

int capacity() const; - возвращает объем выделенной памяти

void clear(); - очищает контейнер

Kniga — структура для хранения данных о книгах.

Так как отсутствуют требования работы с числами для удобства для всех полей был выбран тип string.

5 Разработка интерфейса пользователя

Главное окно содержит панель инструментов, главное меню, таблицу для отображения базы. Главное меню содержит три пункта ***File, Edit, Help.***

В пункте ***File*** содержатся подпункты ***New, Open, Save, Save as, Exit***.

В пункте ***Edit – Add, Edit, Delete, Find.***

В пункте ***Help – About.***

Также эти пункты есть на панели инструментов.

В нижней части окна находится строка состояния, которая отображает текущее состояние базы данных. Также, при нажатии кнопки поиск на панели инструментов появляется строка поиска. Интерфейс программы продемонстрирован на рисунке 1.

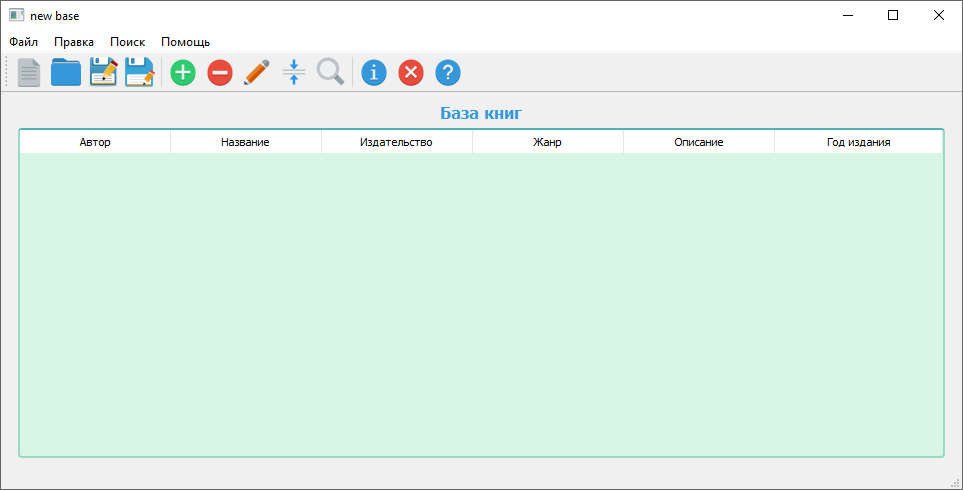


Рисунок 1 – Интерфейс программы

6 Требования к составу и параметрам технических средств

Системные требования для работы с программой указаны в таблице 4.

Таблица 4 – Системные требования

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Минимальные** | **Рекомендуемые** |
| ОС | WindowsVista, Windows 7, Windwos8/8.1/10 | |
| Процессор | Intel/AMD dual-core  2.0 GHz | Intel Core i3 E5-series/  AMD Athlon X2 6400+ 2.5 GHz |
| ОЗУ | 2 GB | 4 GB |
| Видеоадаптер | GeForce 8600 GT  512 MB  ATI (AMD) X1950  512 MB | GeForce GTX 250  1024 MB  ATI (AMD) HD 4500  1024 MB |
| Место на ЖД | 6 GB | |
| Сеть | Доступ в интернет по ADSL 56 кб/с или более скоростной канал. | |
| Устройства ввода | Клавиатура, мышь | |
| Устройства вывода | ЖК монитор с разрешением 1024х768 или более | |

Вывод

Поставленная задача была выполнена успешно. В ходе работы над курсовой работой были освоены основные возможности языка С++ и библиотеки Qt 5.

Написанная программа выполняет все поставленные задачи, но также есть направления, в которых стоило бы улучшить программу. Например, организовать функцию сортировки.

Литература

1. С. В. Козин. Лекции по дисциплине Объектно-ориентированное

программирование.

1. http://doc.qt.io/

Приложение А – Листинг программы

Файлы реализации

Adddialog.cpp

#include "adddialog.h"

#include "ui\_adddialog.h"

AddDialog::**AddDialog**(QWidget \*parent,const Kniga& Kniga\_) :

QDialog(parent),

ui(new Ui::AddDialog)

{

ui->setupUi(this);

ui->avtorLine->setText(Kniga\_.avtor);

ui->nazvanieLine->setText(Kniga\_.nazvanie);

ui->izdatelstvoLine->setText(Kniga\_.izdatestvo);

ui->janrLine->setText(Kniga\_.janr);

ui->opisanieLine->setText(Kniga\_.opisanie);

ui->godizdanLine->setText(Kniga\_.godizdan);

}

AddDialog::~***AddDialog***()

{

delete ui;

}

Kniga AddDialog::**getdata**()

{

Kniga Kniga\_;

Kniga\_.avtor=ui->avtorLine->text();

Kniga\_.nazvanie=ui->nazvanieLine->text();

Kniga\_.izdatestvo=ui->izdatelstvoLine->text();

Kniga\_.janr=ui->janrLine->text();

Kniga\_.opisanie=ui->opisanieLine->text();

Kniga\_.godizdan=ui->godizdanLine->text();

return Kniga\_;

}

database.cpp

#include "database.h"

database::database(QObject \*parent):

QObject(parent),saved(true){}

database::~database(){}

void database::add(const Kniga &Kniga\_)

{

v.push\_back(Kniga\_);

saved=false;

}

void database::change(int n)

{

AddDialog myDialog(0,v[n]);

int result = myDialog.exec();

if (QDialog::Accepted == result)

{

v[n]=myDialog.getdata();

saved=false;

}

}

void database::del(int numRow)

{

saved=false;

v.del(numRow);

}

void database::saveToFile(const QString &fileName)

{

if (fileName.isEmpty())

return;

else {

QFile file(fileName);

if (!file.open(QIODevice::WriteOnly)) {

return;

}

else

{

QString str("");

typename vector<Kniga>::iterator iter;

for(iter=v.begin(); iter!=v.end();++iter)

{

str.append(iter->avtor);

str.append("|");

str.append(iter->nazvanie);

str.append("|");

str.append(iter->izdatestvo);

str.append("|");

str.append(iter->janr);

str.append("|");

str.append(iter->opisanie);

str.append("|");

str.append(iter->godizdan);

str.append("\n");

}

QByteArray ar;

ar.append(str);

file.write(ar);

}

file.close();

saved=true;

}

}

void database::loadFromFile(const QString &fileName)

{

if (fileName.isEmpty())

return;

else {

QFile file(fileName);

if (!file.open(QIODevice::ReadOnly)) {

//QMessageBox::information(this, tr("Unable to open file"),

// file.errorString());

return;

}

QTextStream stream(&file);

QString str;

QStringList fields;

Kniga tmp;

while(!stream.atEnd())

{

str=stream.readLine();

str=str.simplified();

fields=str.split("|");

tmp.avtor=fields[0];

tmp.nazvanie=fields[1];

tmp.izdatestvo=fields[2];

tmp.janr=fields[3];

tmp.opisanie=fields[4];

tmp.godizdan=fields[5];

add(tmp);

}

file.close();

saved=true;

}

}

bool database::isSaved()

{

return saved;

}

void database::clear()

{

saved=false;

v.clear();

}

void database::addtotable(QTableWidget \*table)

{

typename vector<Kniga>::iterator iter=v.begin();

for (iter=v.begin();iter!=v.end();++iter)

{

table->setRowCount(table->rowCount()+1);

table->setItem(table->rowCount()-1,0,new QTableWidgetItem(iter->avtor));

table->setItem(table->rowCount()-1,1,new QTableWidgetItem(iter->nazvanie));

table->setItem(table->rowCount()-1,2,new QTableWidgetItem(iter->izdatestvo));

table->setItem(table->rowCount()-1,3,new QTableWidgetItem(iter->janr));

table->setItem(table->rowCount()-1,4,new QTableWidgetItem(iter->opisanie));

table->setItem(table->rowCount()-1,5,new QTableWidgetItem(iter->godizdan));

}

}

void database::merge(const QString &fileName)

{

if (fileName.isEmpty())

return;

else {

QFile file(fileName);

if (!file.open(QIODevice::ReadOnly)) {

return;

}

QTextStream stream(&file);

QString str;

QStringList fields;

Kniga tmp;

while(!stream.atEnd())

{

str=stream.readLine();

str=str.simplified();

fields=str.split("|");

tmp.avtor=fields[0];

tmp.nazvanie=fields[1];

tmp.izdatestvo=fields[2];

tmp.janr=fields[3];

tmp.opisanie=fields[4];

tmp.godizdan=fields[5];

add(tmp);

}

file.close();

saved=false;

}

}

void database::find(QString s, QTableWidget \*table)

{

typename vector<Kniga>::iterator iter;

for (iter=v.begin();iter!=v.end();++iter)

{

if ( iter->avtor.contains(s)||

iter->nazvanie.contains(s)||

iter->izdatestvo.contains(s)||

iter->janr.contains(s)||

iter->opisanie.contains(s)||

iter->godizdan.contains(s))

{

table->setRowCount(table->rowCount()+1);

table->setItem(table->rowCount()-1,0,new QTableWidgetItem(iter->avtor));

table->setItem(table->rowCount()-1,1,new QTableWidgetItem(iter->nazvanie));

table->setItem(table->rowCount()-1,2,new QTableWidgetItem(iter->izdatestvo));

table->setItem(table->rowCount()-1,3,new QTableWidgetItem(iter->janr));

table->setItem(table->rowCount()-1,4,new QTableWidgetItem(iter->opisanie));

table->setItem(table->rowCount()-1,5,new QTableWidgetItem(iter->godizdan));

}

}

}

main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include <QDebug>

#include <QTableView>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

base\_=new database();

ui->toolBar->addWidget(ui->search);

ui->toolBar->setVisible(false);

ui->toolBar->addAction(ui->actionClosefind);

connect(ui->findline,SIGNAL(textChanged(QString)),this,SLOT(onfindlinechanged(QString)));

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_actionNew\_triggered()

{

if (!base\_->isSaved())

{

QMessageBox msgBox;

msgBox.setText("Вы хотите сохранить изменения?");

msgBox.setIcon(QMessageBox::Warning);

msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

msgBox.setDefaultButton(QMessageBox::No);

int ret = msgBox.exec();

if (ret==QMessageBox::Yes)

on\_actionSave\_triggered();

}

setWindowTitle("new base");

base\_->clear();

tablereset();

}

void MainWindow::on\_actionAdd\_triggered()

{

AddDialog myDialog;

int result = myDialog.exec();

if (QDialog::Accepted == result)

{

base\_->add( myDialog.getdata());

tablereset();

base\_->addtotable(ui->tableWidget);

}

ui->statusBar->showMessage("Запись добавлена");

}

void MainWindow::on\_actionChange\_triggered()

{

if (!ui->tableWidget->selectedItems().isEmpty())

base\_->change(ui->tableWidget->selectedItems().back()->row());

tablereset();

base\_->addtotable(ui->tableWidget);

}

void MainWindow::on\_actionDelete\_triggered()

{

QMessageBox msgBox;

msgBox.setIcon(QMessageBox::Warning);

msgBox.setText("Вы хотите удалить запись?");

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msgBox.setDefaultButton(QMessageBox::Ok);

int ret = msgBox.exec();

if (ret==QMessageBox::Ok)

if (!ui->tableWidget->selectedItems().isEmpty())

{

int numRow=ui->tableWidget->selectedItems().last()->row();

ui->tableWidget->removeRow(numRow);

base\_->del(numRow);

}

ui->statusBar->showMessage("Запись удалена");

}

void MainWindow::on\_actionExit\_triggered()

{

close();

}

void MainWindow::on\_actionSave\_triggered()

{

QString fileName;

if (windowTitle()=="new base")

{

fileName = QFileDialog::getSaveFileName(this,

tr("Open Data Base"), "new base.tvbd",

tr("Base (\*.tvbd);;All Files (\*)"));

setWindowTitle(fileName);

}

else

fileName=windowTitle();

base\_->saveToFile(fileName);

ui->statusBar->showMessage("Сохранено");

}

void MainWindow::on\_actionOpen\_triggered()

{

if (!base\_->isSaved())

{

QMessageBox msgBox;

msgBox.setIcon(QMessageBox::Warning);

msgBox.setText("Вы хотите сохранить изменения?");

msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

msgBox.setDefaultButton(QMessageBox::No);

int ret = msgBox.exec();

if (ret==QMessageBox::Yes)

on\_actionSave\_triggered();

}

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open Data Base"), "",

tr("Base (\*.tvbd);;All Files (\*)"));

setWindowTitle(fileName);

base\_->clear();

tablereset();

base\_->loadFromFile(fileName);

base\_->addtotable(ui->tableWidget);

ui->statusBar->showMessage("База открыта");

}

void MainWindow::tablereset()

{

ui->tableWidget->clear();

ui->tableWidget->setRowCount(0);

QStringList s;

s.append("Автор");

s.append("Название");

s.append("Издательство");

s.append("Жанр");

s.append("Описание");

s.append("Год издания");

ui->tableWidget->setHorizontalHeaderLabels(s);

}

void MainWindow::on\_actionSaveAs\_triggered()

{

QString fileName;

fileName = QFileDialog::getSaveFileName(this,

tr("Open Data Base"), "new base.tvbd",

tr("Base (\*.tvbd);;All Files (\*)"));

setWindowTitle(fileName);

base\_->saveToFile(fileName);

}

void MainWindow::on\_actionMerge\_triggered()

{

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open Data Base"), "",

tr("Base (\*.tvbd);;All Files (\*)"));

if (fileName!=windowTitle())

{

base\_->merge(fileName);

tablereset();

base\_->addtotable(ui->tableWidget);

}

ui->statusBar->showMessage("Базы объединены");

}

void MainWindow::on\_actionSearch\_triggered()

{

ui->toolBar->setVisible(true);

ui->findline->setFocus();

}

void MainWindow::on\_actionClosefind\_triggered()

{

ui->toolBar->setVisible(false);

ui->statusBar->showMessage(" ");

}

void MainWindow::onfindlinechanged(QString str)

{

tablereset();

base\_->find(str,ui->tableWidget);

ui->statusBar->showMessage("Найдено "+QString::number(ui->tableWidget->rowCount())+" записей" );

}

void MainWindow::closeEvent(QCloseEvent \*)

{

if (!base\_->isSaved())

{

QMessageBox msgBox;

msgBox.setIcon(QMessageBox::Warning);

msgBox.setText("Вы хотите сохранить изменения?");

msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

msgBox.setDefaultButton(QMessageBox::No);

int ret = msgBox.exec();

if (ret==QMessageBox::Yes)

on\_actionSave\_triggered();

}

}

void MainWindow::on\_actionAbout\_triggered()

{

QMessageBox::about(this,"О программе","Курсовая работа по дисциплине ООП\n""Тема: База данных, характеризующая книгу\n""Вариант 6\n"

"Выполнил студент группы ИКПИ-72 \n""Ганеев Александр\n""Санкт-Петербург\n""2018 год");

}

Заголовочные файлы

adddialog.h

#ifndef ADDDIALOG\_H

#define ADDDIALOG\_H

#include <QDialog>

#include "database.h"

#include "knigi.h"

#include<QAbstractButton>

namespace Ui {

class AddDialog;

}

class AddDialog : public QDialog

{

Q\_OBJECT

public:

explicit AddDialog(QWidget \*parent = 0,

const Kniga& kniga\_=Kniga());

~AddDialog();

Kniga getdata();

private:

Ui::AddDialog \*ui;

};

#endif // ADDDIALOG\_H

database.h

#ifndef database\_H

#define database\_H

#include "vector.h"

#include <QTableWidget>

#include <QMessageBox>

#include <algorithm>

#include <QDebug>

#include <QObject>

#include <QFileDialog>

#include "adddialog.h"

#include <knigi.h>

class database : public QObject

{

Q\_OBJECT

public:

explicit database(QObject \*parent = 0);

~database();

void find(QString s,QTableWidget \*t);

void saveToFile(const QString &fileName);

void loadFromFile(const QString &fileName);

void add(const Kniga& Kniga\_);

void change(int n);

void del(int numRow);

void addtotable(QTableWidget \*t);

void merge(const QString &fileName);

void clear();

private:

vector<Kniga> v;

bool saved;

signals:

public slots:

bool isSaved();

};

#endif // database\_H

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "adddialog.h"

#include "database.h"

#include <QMainWindow>

#include <QSpacerItem>

namespace Ui

{

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

void tablereset();

private slots:

void onfindlinechanged(QString);

void on\_actionNew\_triggered();

void on\_actionAdd\_triggered();

void on\_actionChange\_triggered();

void on\_actionDelete\_triggered();

void on\_actionExit\_triggered();

void on\_actionSave\_triggered();

void on\_actionOpen\_triggered();

void on\_actionSaveAs\_triggered();

void on\_actionMerge\_triggered();

void on\_actionSearch\_triggered();

void on\_actionClosefind\_triggered();

void closeEvent(QCloseEvent \*);

void on\_actionAbout\_triggered();

private:

Ui::MainWindow \*ui;

database \*base\_;

};

#endif // MAINWINDOW\_H

knigi.h

#ifndef KNIGI\_H

#define KNIGI\_H

#include <QString>

struct **Kniga**

{

QString avtor;

QString nazvanie;

QString izdatestvo;

QString janr;

QString opisanie;

QString godizdan;

};

#endif // KNIGI\_H

vector.h

#ifndef VECTOR\_H

#define VECTOR\_H

#include <algorithm>

template <typename T>

class **vector**

{

private:

T \*vfirst;

int size\_;

int capacity\_;

public:

typedef T\* iterator;

**vector**();

void **push\_back**(const T& t);

void **del**(int index);

T& operator[] (int i) const;

const T& **last** ();

iterator **begin** ();

iterator **end** ();

int **size**() const;

bool **empty**() const;

int **capacity**() const;

void **clear**();

};

template <typename T>

vector<T>::**vector**()

{

vfirst=new T[3];

size\_=0;

capacity\_=3;

}

template <typename T>

void vector<T>::**clear**()

{

delete[] vfirst;

vfirst=new T[3];

size\_=0;

capacity\_=3;

}

template <typename T>

void vector<T>::**push\_back**(const T& t)

{

if (capacity\_==size\_)

{

int newcapacity=2\*capacity();

T \*newmem=new T[newcapacity];

std::copy(begin(),end(),newmem);

delete[] vfirst;

vfirst=newmem;

capacity\_\*=2;

}

\*(vfirst+size\_)=t;

size\_++;

}

template <typename T>

void vector<T>::**del**(int index)

{

for (int i=index+1;i<size\_;i++)

{

vfirst[i-1]=vfirst[i];

}

size\_--;

}

template <typename T>

int vector<T>::**capacity**() const

{

return capacity\_;

}

template <typename T>

int vector<T>::**size**() const

{

return size\_;

}

template <typename T>

bool vector<T>::**empty**() const

{

return (size\_==0);

}

template <typename T>

T& vector<T>::operator[](int pos) const

{

return (\*(vfirst + pos));

}

template <typename T>

typename vector<T>::iterator vector<T>::**begin** ()

{

return vfirst;

}

template <typename T>

typename vector<T>::iterator vector<T>::**end** ()

{

return vfirst+size\_;

}

template <typename T>

const T& vector<T>::**last**()

{

return \*(vfirst+size\_-1);

}

#endif // VECTOR\_H

Приложение Б – Руководство пользователя

Работа с базами данных

Создание базы данных

Для создания новой базы в пункте меню «File» выберете команду «New» или нажмите на кнопку «New» на панели инструментов.

Открытие базы данных

Чтобы открыть существующую базу в пункте меню «File» выберете команду «Open» или нажмите на кнопку «Open» на панели инструментов.

Сохранение базы данных

В пункте меню «File» выберете команду «Save» или нажмите на кнопку «Save» на панели инструментов. При работе с уже существующей базой изменения сохранятся в открытый файл. При работе с новой базой данных необходимо выбрать имя файла и каталог, в который он будет сохранен.

Так же можно сохранить уже существующую базу под другим именем. Для этого в пункте меню «File» выберете команду «Save As.» В открывшемся окне введите имя файла и укажите каталог в котором он будет сохранен.

Объединение баз данных

Чтобы объединить базы данных в пункте меню «Edit» выберете команду «Объединить» или нажмите на кнопку «Merge» на панели инструментов. В появившемся окне выберите файл базы данных с которой вы хотите объединить текущую базу.

Работа с записями

Создание записи

Чтобы создать новую запись в базе данных в пункте меню «Edi» выберете команду «Add» или нажмите на кнопку «Add» на панели инструментов. В появившемся окне заполните поля и нажмите «Ok». Интерфейс создания записи продемонстрирован на рисунке 2.

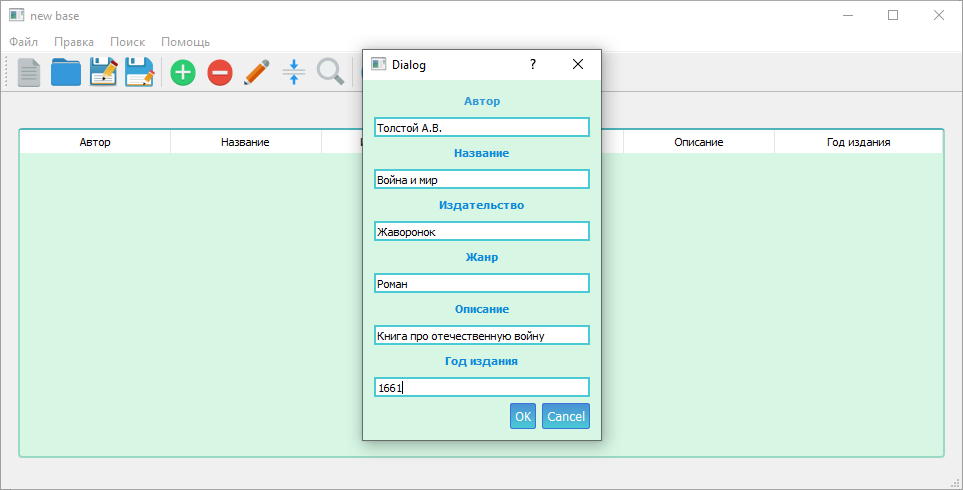


Рисунок 2 – Интерфейс создания записи

Редактирование записи

Чтобы редактировать существующую запись в базе данных выделите необходимую запись, нажав на нее правой кнопкой мыши. Далее в пункте меню «Edit» выберете команду «Edit» или нажмите на кнопку «Edit» на панели инструментов. Отредактируйте поля в появившемся окне и нажмите «Ok». Интерфейс редактирования записи продемонстрирован на рисунке 3.

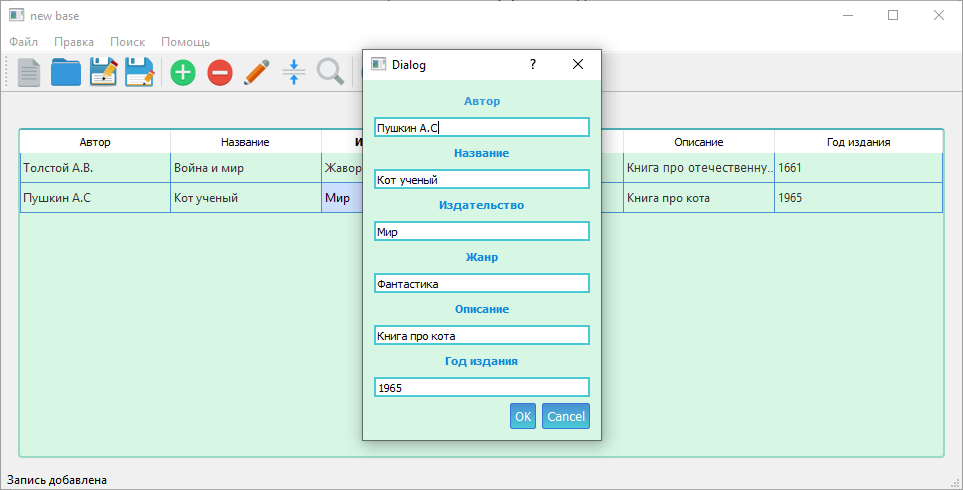


Рисунок 3 – Интерфейс редактирования

Удаление записи

Чтобы удалить существующую запись в базе данных выделите необходимую запись, нажав на нее правой кнопкой мыши. Далее в пункте меню «Edit» выберете команду «Delete» или нажмите на кнопку «Delete» на панели инструментов.

Поиск записи

Чтобы найти запись в базе данных в пункте меню «Edit» выберете команду «Find» или нажмите на кнопку «Find» на панели инструментов. В появившемся в нижней части окна поле введите критерий, по которому нужно искать. Результаты поиска будут отображены на экране.

О программе

Чтобы посмотреть данные о программе в пункте меню «Help» выберите пункт «About» или нажмите на кнопку «About» в панели инструментов.

Так же предусмотрены горячие клавиши:

Ctrl+N — создать новую базу.

Ctrl+O — открыть существующую базу.

Ctrl+M — объединить базы.

Ctrl+S — сохранить базу.

Ctrl+Alt+S — сохранить как.

Ctrl+A — добавить запись.

Ctrl+F — открыть форму поиска.

Ctrl+E — редактировать выбранную строку

Delete — удалить выбранную запись.