## **Постановка задачи**

Создать базу данных, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа разрабатываемого 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 Qt5. Среда программировании Qt Creater.

В качестве элементов контейнеров должны использоваться объекты классов, характеризующие автомобиль.

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.

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

## **Разработка структуры программы**

В программе содержатся следующие модули:

1. auto.cpp
2. controllerdb.cpp
3. vector.cpp
4. main.cpp
5. mainwindow.cpp
6. mainwindow.h
7. info.cpp
8. info.h
9. editdatabase.cpp
10. editdatabase.h
11. editdatabase.ui
12. info.ui
13. mainwindow.ui

Одним из требований к курсовой работе является создание класса, характеризующего автомобиль и контейнер для объектов этого класса.

Класс, характеризующий автомобиль, называется «Auto», он включает в себя следующие поля:

1. Mark – Марка автомобиля (Тип данных QString)
2. Color – Цвет автомобиля (Тип данных QString)
3. Number – Гос. Номер автомобиля (Тип данных QString)
4. Age – Год выпуска автомобиля (Тип данных integer)

Класс-контейнер для объектов класса «Auto» называется «Vector». Он включает в себя методы, позволяющие:

1. Получить количество объектов
2. Добавить объект в контейнер
3. Получить объект по номеру
4. Изменить объект по номеру
5. Очистить контейнер

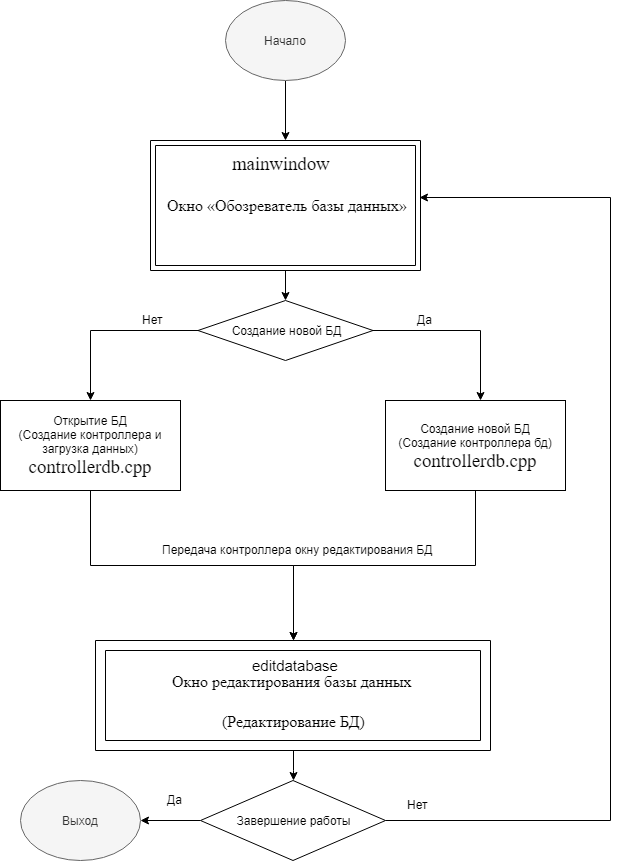
Программа работает с файлами расширения «.db». Созданные базы данных сохраняет в папку db в директории с программой.

.

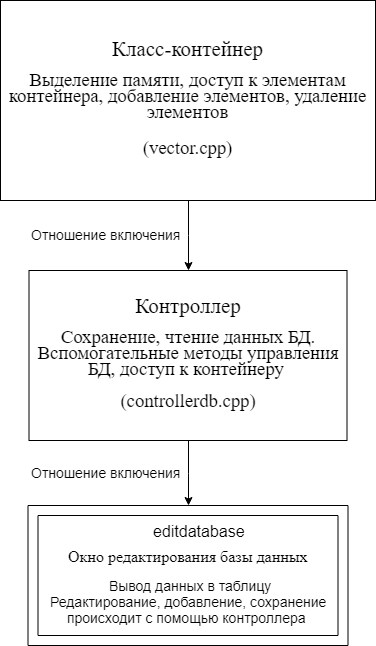
## **Структура программы**

|  |  |
| --- | --- |
| **Модуль** | **Описание модуля** |
| «main.cpp» | Главная точка входа в программу. Создает и вызывает первое окно. |
| «mainwindow.cpp»  «mainwindow.h» «mainwindow.ui» | Модуль отвечает за окно «Обозреватель базы данных».  Файл реализации функций - «mainwindow.cpp».  Файл формы - «mainwindow.ui».  Заголовочный файл – «mainwindow.h».  В этом модуле реализованы функции обновления списка доступных баз данных, открытия БД, удаления БД, объединения БД |
| «controllerdb.cpp» | Контроллер базы данных. Управляет классом-контейнером, хранит имя базы данных, позволяет сохранять БД в файл, считывать БД из файла, удалять файл с БД, объединять БД. Позволяет получать элементы из контейнера. |
| « editdatabase.cpp»  «editdatabase.h»  «editdatabase.ui» | Модуль отвечает за окно редактирования базы данных.  Файл реализации функций - «editdatabase.cpp».  Файл формы - «editdatabase.ui».  Заголовочный файл – «editdatabase.h».  Содержит в контроллер базы данных, который позволяет отображать БД в таблице, изменять и сохранять базу данных. Так же в модуле реализованы функции поиска и контекстного меню. |
| «info.cpp»  «info.h»  «info.ui» | Окно «About» (О программе) |
| «vector.cpp» | Класс-контейнер. |

### **Алгоритм**



### **Структура классов управления базой данных**



### **Модуль «main.cpp»**

1. #include "mainwindow.h"
2. #include <QApplication>
4. **int** main(**int** argc, **char** \*argv[])
5. {
6. QApplication a(argc, argv);
7. MainWindow   w;
9. QApplication::setAttribute(Qt::AA\_Use96Dpi);
10. w.setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);
11. w.show();
12. **return**(a.exec());
13. }

### **Модуль «mainwindow.h»**

1. #ifndef MAINWINDOW\_H
2. #define MAINWINDOW\_H
4. #include <QMainWindow>
5. #include <editdatabase.h>
7. QT\_BEGIN\_NAMESPACE
8. **namespace** Ui {
9. **class** MainWindow;
10. }
11. QT\_END\_NAMESPACE
13. **class** MainWindow : **public** QMainWindow
14. {
15. Q\_OBJECT
17. **public** slots:
18. **void** UpdateListDB();
20. **public**:
21. MainWindow(QWidget \*parent = nullptr);
22. ~MainWindow();
24. **private** slots:
25. **void** on\_CreateDB\_clicked();
27. **void** on\_OpenSelectDB\_clicked();
29. **void** on\_DelDB\_clicked();
31. **void** on\_ListDB\_doubleClicked(**const** QModelIndex&index);
33. **void** on\_About\_clicked();
35. **void** on\_ListDB\_itemSelectionChanged();
37. **void** on\_AssociationDB\_clicked();
39. **void** on\_OpenFileDB\_clicked();
41. **private**:
42. Ui::MainWindow \*ui;
43. EditDataBase \*EditForm;
44. };
45. #endif // MAINWINDOW\_H

### **Модуль «mainwindow.cpp»**

1. #include "mainwindow.h"
2. #include "ui\_mainwindow.h"
3. #include <QDir>
4. #include <QFile>
5. #include "controllerdb.cpp"
6. #include "info.h"
7. #include "auto.cpp"
8. #include <iostream>
9. #include <QInputDialog>
10. #include <QFileDialog>
11. MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(**new** Ui::MainWindow)
12. {
13. ui->setupUi(**this**);
15. EditForm = **new** EditDataBase();
17. connect(EditForm, &EditDataBase::ReviewerWindow, **this**, &MainWindow::show);
19. QApplication::setAttribute(Qt::AA\_Use96Dpi);
20. **this**->setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);
21. **if**(!QDir("db").exists()){
22. QDir().mkdir("db");
23. }
25. UpdateListDB();
26. }
28. MainWindow::~MainWindow()
29. {
30. **delete** ui;
31. }
33. **void** MainWindow::UpdateListDB() //Обновиться лист с БД
34. {
35. QFileInfoList dirContent = QDir("db/").entryInfoList(QStringList() << "\*.db",
36. QDir::Files | QDir::NoDotAndDotDot, QDir::Time);
38. ui->ListDB->clear();
39. **for**(auto obj : dirContent){
40. ui->ListDB->addItem(obj.fileName());
41. }
42. statusBar()->showMessage("Кол-во БД: " + QString::number(ui->ListDB->count()));
43. }
45. **void** MainWindow::on\_CreateDB\_clicked() //Создание новой бд
46. {
47. **bool**    NewDBName;
48. QString str = QInputDialog::getText(nullptr,
49. "Создание пустой базы данных",
50. "Имя БД:",
51. QLineEdit::Normal,
52. "AutoDB",
53. &NewDBName
54. );
56. str = str.replace(" ", "");
58. **if**(NewDBName && !str.isEmpty()){
59. ControllerDB db(str);
60. **if**(!db.isExistsFile()){
61. db.Save();
62. UpdateListDB();
63. EditForm->show(); // Показываем второе окно
64. EditForm->SetDataBase(db);
65. **this**->close();    // Закрываем основное окно
66. }
67. **else**{
68. on\_CreateDB\_clicked();
69. }
70. }
71. **else** **if**(NewDBName && str.isEmpty()){
72. QMessageBox::warning(nullptr, "Ошибка", "Имя базы данных не должно быть пустым",
73. QMessageBox::Ok);
74. on\_CreateDB\_clicked();
75. }
76. }
78. **void** MainWindow::on\_OpenSelectDB\_clicked() //Открыть бд
79. {
80. QList <QListWidgetItem \*> lst;
82. lst = ui->ListDB->selectedItems();
83. ControllerDB db;
84. db.load("db/" + lst[0]->text());
85. EditForm->show();   // Показываем второе окно
86. EditForm->SetDataBase(db);
87. **this**->close();      // Закрываем основное окно
88. }
90. **void** MainWindow::on\_DelDB\_clicked() //Удалить БД
91. {
92. ControllerDB db;
94. db.load("db/" + ui->ListDB->item(ui->ListDB->currentRow())->text());
95. db.Delete();
96. UpdateListDB();
97. }
99. **void** MainWindow::on\_ListDB\_doubleClicked(**const** QModelIndex&index) //Двойной клик
100. {
101. ControllerDB db;
103. db.load("db/" + ui->ListDB->item(ui->ListDB->currentRow())->text());
104. EditForm->show();    // Показываем второе окно
105. EditForm->SetDataBase(db);
106. **this**->close();       // Закрываем основное окно
107. }
109. **void** MainWindow::on\_About\_clicked()
110. {
111. Info \*infoform = **new** Info;
113. infoform->show();
114. }
116. **void** MainWindow::on\_ListDB\_itemSelectionChanged()
117. {
118. QList <QListWidgetItem \*> lst;
120. lst = ui->ListDB->selectedItems();
121. **if**(lst.count() == 1){
122. ui->OpenSelectDB->setEnabled(1);
123. ui->DelDB->setEnabled(1);
124. }
125. **else**{
126. ui->OpenSelectDB->setEnabled(0);
127. ui->DelDB->setEnabled(0);
128. }
130. **if**(lst.count() >= 2){
131. ui->AssociationDB->setEnabled(1);
132. }
133. **else**{
134. ui->AssociationDB->setEnabled(0);
135. }
136. }
138. **void** MainWindow::on\_AssociationDB\_clicked()
139. {
140. QList <QListWidgetItem \*> selected\_items = ui->ListDB->selectedItems();
141. ControllerDB ct;
143. **if**(selected\_items.size() > 0){
144. **for**(auto obj : selected\_items){
145. ControllerDB db;
146. db.load("db/" + obj->text());
147. ct = ct.append(db);
148. }
149. }
150. ct.Save();
151. UpdateListDB();
152. }
154. **void** MainWindow::on\_OpenFileDB\_clicked()
155. {
156. QString      str = QFileDialog::getOpenFileName(0, "Open Dialog", "", "\*.db");
157. ControllerDB db;
159. db.load(str);
160. EditForm->show();   // Показываем второе окно
161. EditForm->SetDataBase(db);
162. **this**->close();      // Закрываем основное окно
163. }

### **Модуль «editdatabase.h»**

1. #ifndef EDITDATABASE\_H
2. #define EDITDATABASE\_H
4. #include <QMainWindow>
5. #include "controllerdb.cpp"
6. #include <QTableWidget>
7. **namespace** Ui {
8. **class** EditDataBase;
9. }
11. **class** EditDataBase : **public** QMainWindow
12. {
13. Q\_OBJECT
15. **public**:
16. **explicit** EditDataBase(QWidget \*parent = nullptr);
17. ~EditDataBase();
18. **void** SetDataBase(**const** ControllerDB db);
19. **virtual** **void** closeEvent(QCloseEvent \*event);
21. signals:
22. **void** ReviewerWindow(); //Сигнал открытия "Обозреватель базы данных"
24. **private** slots:
25. **void** on\_OpenReviewerWindow\_triggered();
27. **void** on\_AddLinebtn\_clicked();
29. **void** on\_RemoveLinebtn\_clicked();
31. **void** appendTableRow(QTableWidget \*p\_table);
32. **void** insertTableRow(QTableWidget \*p\_table);
33. **void** deleteTableRow(QTableWidget \*p\_table);
35. **void** on\_TableDB\_cellChanged(**int** row, **int** column);
36. **void** on\_TableDB\_customContextMenuRequested(**const** QPoint&pos);
38. **void** on\_Save\_triggered();
40. **void** on\_search\_textChanged(**const** QString&arg1);
42. **void** on\_Open\_triggered();
44. **void** on\_SaveAs\_triggered();
46. **void** on\_appenddb\_triggered();
48. **void** on\_about\_triggered();
50. **void** on\_SaveTool\_clicked();
52. **void** on\_OpenAsTool\_clicked();
54. **private**:
55. Ui::EditDataBase \*ui;
56. ControllerDB db = \***new** ControllerDB();
57. QString previousTextSearch;
58. };
60. #endif // EDITDATABASE\_H

### **Модуль «editdatabase.cpp»**

1. #include "editdatabase.h"
2. #include "ui\_editdatabase.h"
3. #include <QFileDialog>
4. #include "info.h"
5. EditDataBase::EditDataBase(QWidget \*parent) :
6. QMainWindow(parent),
7. ui(**new** Ui::EditDataBase)
8. {
9. ui->setupUi(**this**);
11. **this**->setContextMenuPolicy(Qt::CustomContextMenu); //ПКМ меню
12. connect(**this**, SIGNAL(customContextMenuRequested(**const** QPoint&)), **this**, SLOT(on\_TableDB\_customContextMenuRequested(**const** QPoint&)));
13. QApplication::setAttribute(Qt::AA\_Use96Dpi);
14. **this**->setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);
15. //Настройки таблицы
16. ui->TableDB->setColumnCount(4);
17. ui->TableDB->setRowCount(1);
18. ui->TableDB->horizontalHeader()->setStretchLastSection(**true**);
19. ui->TableDB->setColumnWidth(0, 250);
20. ui->TableDB->setColumnWidth(2, 200);
21. ui->TableDB->setHorizontalHeaderLabels(QStringList() << "Марка" << "Цвет" << "Гос. номер" << "Год выпуска");
22. db.isModified = **false**;
23. statusBar()->showMessage("База данных готова");
24. }
26. **void** EditDataBase::on\_TableDB\_customContextMenuRequested(**const** QPoint&pos) //ПКМ меню
27. {
28. QMenu menu(**this**);
29. auto  infoA = menu.addAction(tr("Добавить строку в конец"));
30. auto  infoI = menu.addAction(tr("Вставить пустую строку выше"));
31. auto  infoD = menu.addAction(tr("Удалить строку"));
33. **if**((ui->TableDB->hasFocus())){
34. connect(infoA, &QAction::triggered, **this**, [**this**]() {
35. appendTableRow(ui->TableDB);
36. update();
37. });
38. connect(infoI, &QAction::triggered, **this**, [**this**]() {
39. insertTableRow(ui->TableDB);
40. update();
41. });
42. connect(infoD, &QAction::triggered, **this**, [**this**]() {
43. deleteTableRow(ui->TableDB);
44. update();
45. });
46. menu.exec(QCursor::pos());
47. }
48. **else** **if**((ui->TableDB->hasFocus())){
49. connect(infoA, &QAction::triggered, **this**, [**this**]() {
50. appendTableRow(ui->TableDB);
51. update();
52. });
53. connect(infoI, &QAction::triggered, **this**, [**this**]() {
54. insertTableRow(ui->TableDB);
55. update();
56. });
57. connect(infoD, &QAction::triggered, **this**, [**this**]() {
58. deleteTableRow(ui->TableDB);
59. update();
60. });
61. menu.exec(QCursor::pos());
62. }
63. **if**(db.isModified){
64. statusBar()->showMessage("База данных не сохранена");
65. }
66. **else**{
67. statusBar()->showMessage("База данных сохранена");
68. }
69. }
71. **void** EditDataBase::appendTableRow(QTableWidget \*p\_table) //Add row
72. {
73. **int** rows = p\_table->rowCount();
75. p\_table->setRowCount(rows + 1);
76. Auto obj;
77. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 0, **new** QTableWidgetItem(obj.GetMark()));
78. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 1, **new** QTableWidgetItem(obj.GetColor()));
79. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 2, **new** QTableWidgetItem(obj.GetNumber()));
80. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 3, **new** QTableWidgetItem(QString::number(obj.GetAge())));
81. db.isModified = **true**;
82. }
84. **void** EditDataBase::insertTableRow(QTableWidget \*p\_table) //Вставка строки
85. {
86. db.isModified = **true**;
87. **int** row = p\_table->currentRow();
88. p\_table->insertRow(row);
89. }
91. **void** EditDataBase::deleteTableRow(QTableWidget \*p\_table) //Удалить строку
92. {
93. db.isModified = **true**;
94. **int** cur\_row = p\_table->currentRow();
95. p\_table->removeRow(cur\_row);
96. }
98. EditDataBase::~EditDataBase()
99. {
100. **delete** ui;
101. }
103. **void** EditDataBase::SetDataBase(**const** ControllerDB db) //Установить БД
104. {
105. **this**->db = db;
106. ui->TableDB->setRowCount(0);
107. **for**(auto obj : **this**->db.db){
108. ui->TableDB->insertRow(ui->TableDB->rowCount());
109. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 0, **new** QTableWidgetItem(obj.GetMark()));
110. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 1, **new** QTableWidgetItem(obj.GetColor()));
111. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 2, **new** QTableWidgetItem(obj.GetNumber()));
112. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 3, **new** QTableWidgetItem(QString::number(obj.GetAge())));
113. }
114. **this**->db.isModified = **false**;
115. statusBar()->showMessage("База данных сохранена");
116. **this**->setWindowTitle(**this**->db.GetName());
117. ui->search->setText("");
118. }
120. **void** EditDataBase::on\_OpenReviewerWindow\_triggered()
121. {
122. **if**(db.isModified){
123. QMessageBox::StandardButton reply;
124. reply = QMessageBox::question(nullptr, "Подтвердите действие", "Сохранить БД?",
125. QMessageBox::Yes | QMessageBox::No);
127. **if**(reply == QMessageBox::Yes){
128. on\_Save\_triggered();
129. }
130. }
131. emit ReviewerWindow();
132. **this**->close();
133. }
135. **void** EditDataBase::closeEvent(QCloseEvent \*event) //Закрытие окна
136. {
137. //event->ignore();
138. **if**(db.isModified){
139. QMessageBox::StandardButton reply;
140. reply = QMessageBox::question(nullptr, "Подтвердите действие", "Сохранить БД?",
141. QMessageBox::Yes | QMessageBox::No);
143. **if**(reply == QMessageBox::Yes){
144. on\_Save\_triggered();
145. }
146. }
147. emit ReviewerWindow();
149. event->accept();
150. }
152. **void** EditDataBase::on\_AddLinebtn\_clicked() //Добавить новую строку
153. {
154. ui->TableDB->insertRow(ui->TableDB->rowCount());
155. Auto obj;
156. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 0, **new** QTableWidgetItem(obj.GetMark()));
157. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 1, **new** QTableWidgetItem(obj.GetColor()));
158. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 2, **new** QTableWidgetItem(obj.GetNumber()));
159. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 3, **new** QTableWidgetItem(QString::number(obj.GetAge())));
160. db.isModified = **true**;
161. }
163. **void** EditDataBase::on\_RemoveLinebtn\_clicked() //Удалить строку
164. {
165. ui->TableDB->setRowCount(ui->TableDB->rowCount() - 1);
166. db.isModified = **true**;
167. }
169. **void** EditDataBase::on\_TableDB\_cellChanged(**int** row, **int** column) //Изменение ячейки
170. {
171. **if**(column == 3){
172. **bool** ok;
173. **int**  dec = ui->TableDB->item(row, column)->text().toInt(&ok, 10);
174. **if**(!ok){
175. ui->TableDB->setItem(row, column, **new** QTableWidgetItem("0"));
176. update();
177. QMessageBox::warning(nullptr, "Ошибка", "Только цифры",
178. QMessageBox::Ok);
179. }
180. }
181. db.isModified = **true**;
182. statusBar()->showMessage("База данных не сохранена");
183. }
185. **void** EditDataBase::on\_Save\_triggered() //Сохранить
186. {
187. ui->search->setText("");
188. **for**(**int** i = 0; i < ui->TableDB->rowCount(); ++i){
189. ui->TableDB->showRow(i);
190. }
192. statusBar()->showMessage("База данных сохранена");
193. Vector <Auto> newDB;
194. // "Марка" << "Цвет" << "Гос. номер" << "Год выпуска")
195. //  Auto(QString Mark = "null", QString Color = "null", QString Number = "null", int Age = 0)
196. **for**(**int** row = 0; row < ui->TableDB->rowCount(); row++){
197. newDB.push\_back(Auto(ui->TableDB->item(row, 0)->text().isNull() ? "null" : ui->TableDB->item(row, 0)->text(),
198. ui->TableDB->item(row, 1)->text().isNull() ? "null" : ui->TableDB->item(row, 1)->text(),
199. ui->TableDB->item(row, 2)->text().isNull() ? "null" : ui->TableDB->item(row, 2)->text(),
200. ui->TableDB->item(row, 3)->text().isNull() ? 0 : ui->TableDB->item(row, 3)->text().toInt()));
201. }
203. db.db = newDB;
204. db.Save();
205. }
207. **void** EditDataBase::on\_search\_textChanged(**const** QString&arg1) //Поиск
208. {
209. **if**(arg1.isEmpty()){
210. **for**(**int** i = 0; i < ui->TableDB->rowCount(); ++i){
211. ui->TableDB->showRow(i);
212. statusBar()->showMessage("База данных готова");
213. }
214. **return**;
215. }
216. **int** show = 0;
217. **for**(**int** i = 0; i < ui->TableDB->rowCount(); ++i){
218. ui->TableDB->hideRow(i);
219. **for**(**int** q = 0; q < 4; ++q){
220. **if**(ui->TableDB->item(i, q)->text().toLower().contains(arg1.toLower())){
221. ui->TableDB->showRow(i);
222. show++;
223. statusBar()->showMessage("Кол-во элементов удовлетворяющих поиску: " + QString::number(show));
224. }
225. }
226. }
227. }
229. **void** EditDataBase::on\_Open\_triggered()
230. {
231. QString str = QFileDialog::getOpenFileName(nullptr, "Open Dialog", "", "\*.db");
233. **if**(str == ""){
234. **return**;
235. }
237. ControllerDB db;
239. db.load(str);
240. SetDataBase(db);
241. }
243. **void** EditDataBase::on\_SaveAs\_triggered()
244. {
245. QString str = QFileDialog::getOpenFileName(nullptr, "Open Dialog", "", "\*.db");
247. **if**(str == ""){
248. **return**;
249. }
251. ControllerDB db;
253. db.Save(str);
254. }
256. **void** EditDataBase::on\_appenddb\_triggered()
257. {
258. ControllerDB ct;
259. QString      str = QFileDialog::getOpenFileName(nullptr, "Open Dialog", "", "\*.db");
261. **if**(str == ""){
262. **return**;
263. }

266. ct.load(str);
267. db = db.append(ct);
268. **for**(auto obj : db.db){
269. ui->TableDB->insertRow(ui->TableDB->rowCount());
271. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 0, **new** QTableWidgetItem(obj.GetMark()));
272. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 1, **new** QTableWidgetItem(obj.GetColor()));
273. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 2, **new** QTableWidgetItem(obj.GetNumber()));
274. ui->TableDB->setItem(ui->TableDB->rowCount() - 1, 3, **new** QTableWidgetItem(QString::number(obj.GetAge())));
275. }
276. }
278. **void** EditDataBase::on\_about\_triggered()
279. {
280. Info \*infoform = **new** Info;
282. infoform->show();
283. }
285. **void** EditDataBase::on\_SaveTool\_clicked()
286. {
287. on\_Save\_triggered();
288. }
290. **void** EditDataBase::on\_OpenAsTool\_clicked()
291. {
292. on\_Open\_triggered();
293. }

### **Модуль «vector.cpp»**

1. #include <iterator>
2. #include <memory>
4. **template** <**typename** T, **typename** Allocator = std::allocator <T> >
5. **class** Vector {
6. **public**:
7. **typedef** T        value\_type;
8. **typedef** Allocator    allocator\_type;
9. **typedef** std::**size\_t**   size\_type;
10. **typedef** std::**ptrdiff\_t** difference\_type;
11. **typedef** value\_type&   reference;
12. **typedef** **const** value\_type&               const\_reference;
13. **typedef** **typename** std::allocator\_traits <Allocator>::pointer             pointer;
14. **typedef** **typename** std::allocator\_traits <Allocator>::const\_pointer          const\_pointer;
16. **typedef** **typename** std::iterator <std::random\_access\_iterator\_tag, T>::pointer     iterator;
17. **typedef** **typename** std::iterator <std::random\_access\_iterator\_tag, **const** T>::pointer  const\_iterator;
19. **typedef** std::reverse\_iterator <iterator>       reverse\_iterator;
20. **typedef** std::reverse\_iterator <const\_iterator>    const\_reverse\_iterator;
22. Vector() = **default**;
24. Vector(Vector **const**& rhs)
25. {
26. auto newdata = alloc\_n\_copy(rhs.begin(), rhs.end());
28. elements  = newdata.first;
29. first\_free = cap = newdata.second;
30. }
32. Vector& operator =(Vector **const**& rhs)
33. {
34. auto newdata = alloc\_n\_copy(rhs.begin(), rhs.end());
36. free();
37. elements  = newdata.first;
38. first\_free = cap = newdata.second;
39. **return**(\***this**);
40. }
42. ~Vector()
43. {
44. free();
45. }
47. iterator  begin() noexcept
48. {
49. **return**(elements);
50. }
52. const\_iterator  begin() **const** noexcept
53. {
54. **return**(elements);
55. }
57. const\_iterator  cbegin() **const** noexcept
58. {
59. **return**(elements);
60. }
62. iterator  end() noexcept
63. {
64. **return**(first\_free);
65. }
67. const\_iterator  end() **const** noexcept
68. {
69. **return**(first\_free);
70. }
72. const\_iterator  cend() **const** noexcept
73. {
74. **return**(first\_free);
75. }
77. reference operator [](size\_type n)
78. {
79. **return**(elements[n]);
80. }
82. const\_reference operator [](size\_type n) **const**
83. {
84. **return**(elements[n]);
85. }
87. **void** push\_back(T **const**& val)
88. {
89. chk\_n\_alloc();
90. alloc.construct(first\_free++, val);
91. }
93. **void** push\_back(T&&val)
94. {
95. chk\_n\_alloc();
96. alloc.construct(first\_free++, std::move(val));
97. }
99. **template** <**typename** ... Args>
100. **void** emplace\_back(Args&& ... args)
101. {
102. chk\_n\_alloc();
103. alloc.construct(first\_free++, std::forward <Args>(args)...);
104. }
106. size\_type size() **const** noexcept
107. {
108. **return**(std::distance(begin(), end()));
109. }
111. size\_type capacity() **const** noexcept
112. {
113. **return**(cap - elements);
114. }
116. **private**:
117. std::allocator <T> alloc;
118. T \*elements  = nullptr;
119. T \*first\_free = nullptr;
120. T \*cap    = nullptr;
122. **void** free()
123. {
124. **if**(elements){
125. **for**(auto p = first\_free; p != elements; /\*...\*/){
126. alloc.destroy(--p);
127. }
128. alloc.deallocate(elements, cap - elements);
129. }
130. }
132. **void** reallocate()
133. {
134. auto newcapacity = size() ? 2 \* size() : 1;
135. auto first    = alloc.allocate(newcapacity);
136. auto last    = std::uninitialized\_copy(std::make\_move\_iterator(begin()),
137. std::make\_move\_iterator(end()),
138. first);
140. free();
141. elements  = first;
142. first\_free = last;
143. cap    = elements + newcapacity;
144. }
146. std::pair <T \*, T \*> alloc\_n\_copy(**const** T \*b, **const** T \*e)
147. {
148. auto data = alloc.allocate(e - b);
150. **return** { data, std::uninitialized\_copy(b, e, data) };
151. }
153. **void** chk\_n\_alloc()
154. {
155. **if**(size() == capacity()){
156. reallocate();
157. }
158. }
159. };

### **Модуль «controllerdb.cpp»**

1. #ifndef CONTROLLERDB\_H
2. #define CONTROLLERDB\_H
3. #include <QDir>
4. #include <QFile>
5. #include <QTextStream> // Подключаем класс QTextStream
6. #include <string>
7. #include "vector.cpp"
8. #include "auto.cpp"
9. #include <QMessageBox>
10. #include <QtGui>
11. #include <iostream>
12. #include <fstream>
13. #include <string>
15. **class** ControllerDB //Контроллер БД
16. {
17. **private**:
18. QString \_NameDB; //Имя БД
20. **public**:
21. Vector <Auto> db;        //Данные
22. **bool** isModified = **false**; //БД изменена
23. ControllerDB() = **default**;
24. ControllerDB(**const** QString NameDB)
25. {
26. \_NameDB = NameDB;
27. }
29. **void** Save(**const** QString path = nullptr) //Сохранение в файл
30. {
31. QFile file;
33. **if**(path == nullptr){
34. file.setFileName("db\\" + \_NameDB + ".db");
35. }
36. **else**{
37. file.setFileName(path);
38. }
40. **if**(file.open(QIODevice::WriteOnly | QIODevice::Text)){
41. QTextStream writeStream(&file);
42. writeStream.setCodec("Windows-1251");
43. **for**(auto obj : db){
44. writeStream << obj.GetMark() << endl;
45. writeStream << obj.GetColor() << endl;
46. writeStream << obj.GetNumber() << endl;
47. writeStream << obj.GetAge() << endl;
48. }
49. file.close();
50. isModified = **false**;
51. }
52. **else**{
53. QMessageBox messageBox;
54. messageBox.critical(nullptr, "Error", "Не удалось сохранить базу данных в файл(" + \_NameDB + ")");
55. messageBox.setFixedSize(500, 200);
56. }
57. }
59. **bool** isExistsFile(**const** QString path = nullptr) //Файл сущестует?
60. {
61. QFile file;
63. **if**(path == nullptr){
64. file.setFileName("db\\" + \_NameDB + ".db");
65. }
66. **else**{
67. file.setFileName(path);
68. }
70. **if**(!file.exists()){
71. **return**(**false**);
72. }
74. QMessageBox::StandardButton reply;
75. reply = QMessageBox::question(nullptr, "Перезаписать файл?", "Файл с таким именем уже существует, перезаписать файл?",
76. QMessageBox::Yes | QMessageBox::No);
78. **if**(reply == QMessageBox::Yes){
79. **return**(**false**);
80. }
82. **return**(**true**);
83. }
85. ControllerDB& load(**const** QString path) //Загрузка
86. {
87. QFile file(path);
89. \_NameDB = QFileInfo(file).fileName().split(".", QString::SkipEmptyParts).at(0);
91. QList <QString> dateList;
92. std::string     line;
93. std::ifstream   in(path.toStdString()); // окрываем файл для чтения
95. **if**(in.is\_open()){
96. **while**(getline(in, line)){
97. dateList << QString::fromLocal8Bit(line.c\_str());
98. }
99. }
100. in.close();         // закрываем файл
102. **for**(**int** i = 0; i < dateList.size(); ++i){
103. Auto temp;
104. //dateList[i], dateList[++i], dateList[++i], dateList[++i].toInt()
105. temp.SetMark(dateList[i]);
106. temp.SetColor(dateList[++i]);
107. temp.SetNumber(dateList[++i]);
108. temp.SetAge(dateList[++i].toInt());
109. **this**->db.push\_back(temp);
110. }
111. }
113. ControllerDB& Delete() //Удаление файла
114. {
115. QFile file("db\\" + \_NameDB + ".db");
117. QMessageBox::StandardButton reply;
119. reply = QMessageBox::question(nullptr, "Подтвердите действие", "Удалить БД?(" + file.fileName() + ")",
120. QMessageBox::Yes | QMessageBox::No);
122. **if**(reply == QMessageBox::Yes){
123. file.remove();
124. }
125. }
127. ControllerDB append(**const** ControllerDB& db) // append БД
128. {
129. Vector <Auto> newdb;
131. **for**(auto obj : **this**->db){
132. newdb.push\_back(obj);
133. }
135. **for**(auto obj : db.db){
136. newdb.push\_back(obj);
137. }
138. ControllerDB ct("append\_" + \_NameDB + "\_" + db.\_NameDB);
139. ct.db = newdb;
141. **return**(ct);
142. }
144. **void** SetName(**const** QString name) //Устоновить имя БД
145. {
146. \_NameDB = name;
147. }
149. QString GetName()
150. {
151. **return**(\_NameDB);
152. }
153. };
155. #endif // CONTROLLERDB\_H

### 

### **Модуль «auto.cpp»**

1. #ifndef AUTO\_H
2. #define AUTO\_H
3. #include <QString>
5. **class** Auto //Авто
6. {
7. **private**:
8. QString Mark   = "null"; //Марка
9. QString Color  = "null"; //Цвет
10. QString Number = "null"; //Гос номер
11. **int** Age        = 0;      //Год выпуска
12. **public**:
13. Auto(**const** QString Mark = "null", **const** QString Color = "null", **const** QString Number = "null", **const** **int** Age = 0)
14. {
15. **this**->Mark   = Mark;
16. **this**->Color  = Color;
17. **this**->Number = Number;
18. **this**->Age    = Age;
19. }
21. //Сетеры
22. **void** SetMark(**const** QString Mark)
23. {
24. **this**->Mark = Mark;
25. }
27. **void** SetColor(**const** QString Color)
28. {
29. **this**->Color = Color;
30. }
32. **void** SetNumber(**const** QString Number)
33. {
34. **this**->Number = Number;
35. }
37. **void** SetAge(**const** **int** Age)
38. {
39. **this**->Age = Age;
40. }
42. //Гетеры
43. QString GetMark() **const** noexcept
44. {
45. **return**(**this**->Mark);
46. }
48. QString GetColor() **const** noexcept
49. {
50. **return**(**this**->Color);
51. }
53. QString GetNumber() **const** noexcept
54. {
55. **return**(**this**->Number);
56. }
58. **int** GetAge() **const** noexcept
59. {
60. **return**(**this**->Age);
61. }
62. };
64. #endif // AUTO\_H

### 

### **Модуль «info.cpp»**

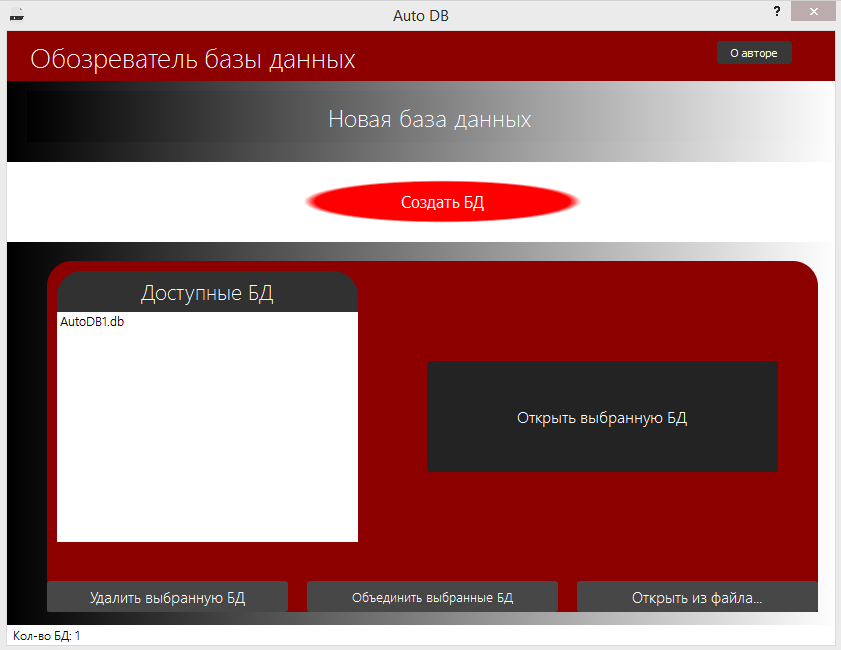
1. #include "info.h"
2. #include "ui\_info.h"
4. Info::Info(QWidget \*parent) :
5. QWidget(parent),
6. ui(**new** Ui::Info)
7. {
8. ui->setupUi(**this**);
9. }
11. Info::~Info()
12. {
13. **delete** ui;
14. }

### **Модуль «info.h»**

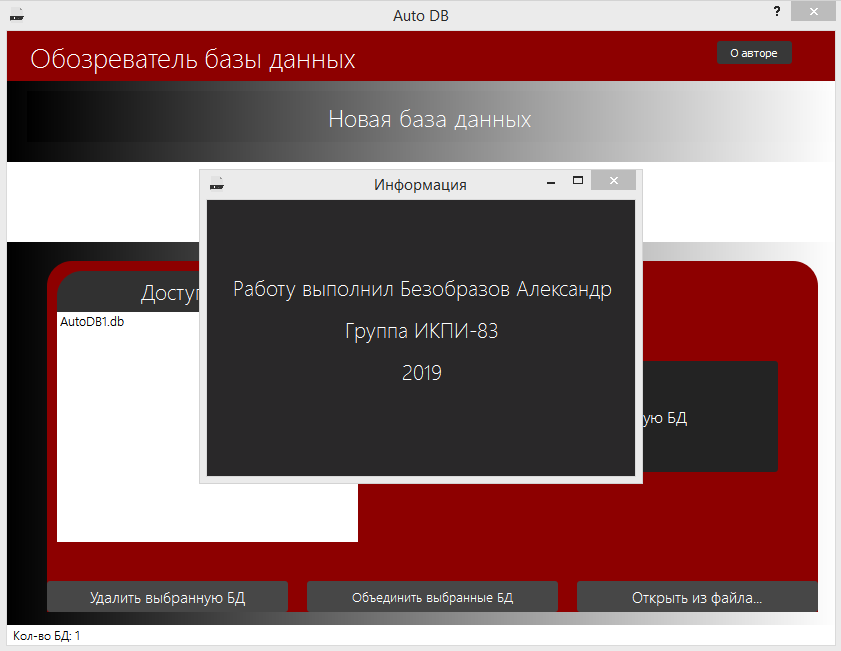
1. #ifndef INFO\_H
2. #define INFO\_H
4. #include <QWidget>
6. **namespace** Ui {
7. **class** Info;
8. }
10. **class** Info : **public** QWidget
11. {
12. Q\_OBJECT
14. **public**:
15. **explicit** Info(QWidget \*parent = nullptr);
16. ~Info();
18. **private**:
19. Ui::Info \*ui;
20. };
22. #endif // INFO\_H

## **Тестирование**

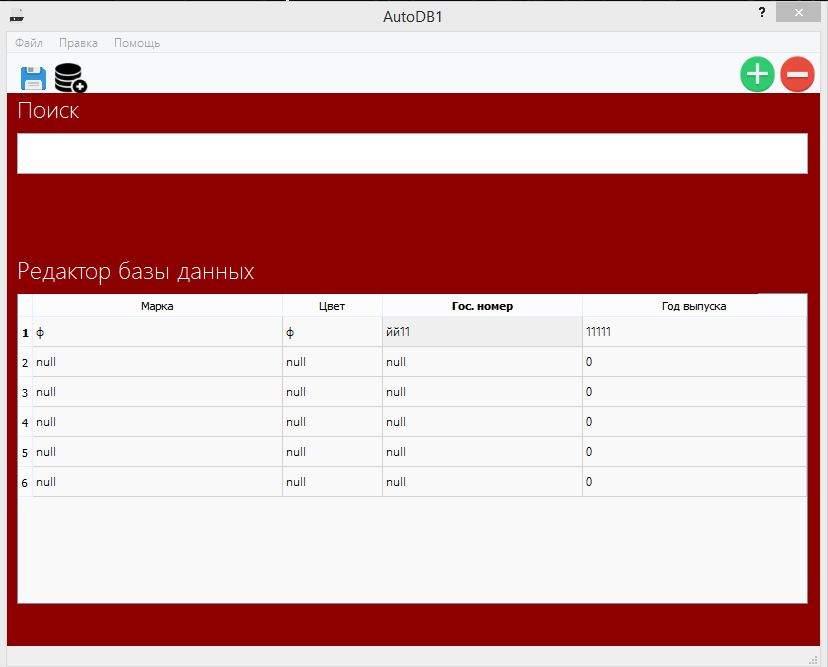
**Главное окно программы**



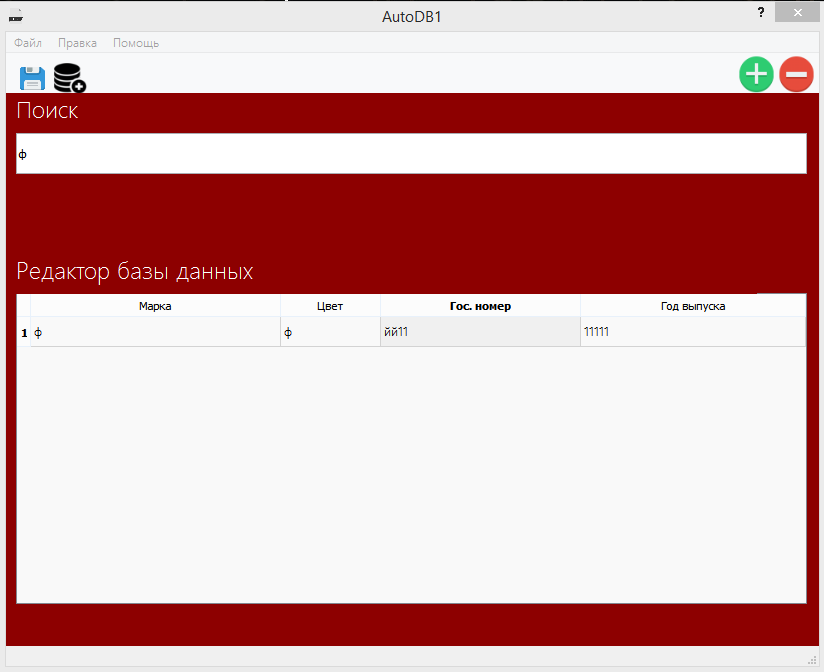
**Информация о программе**



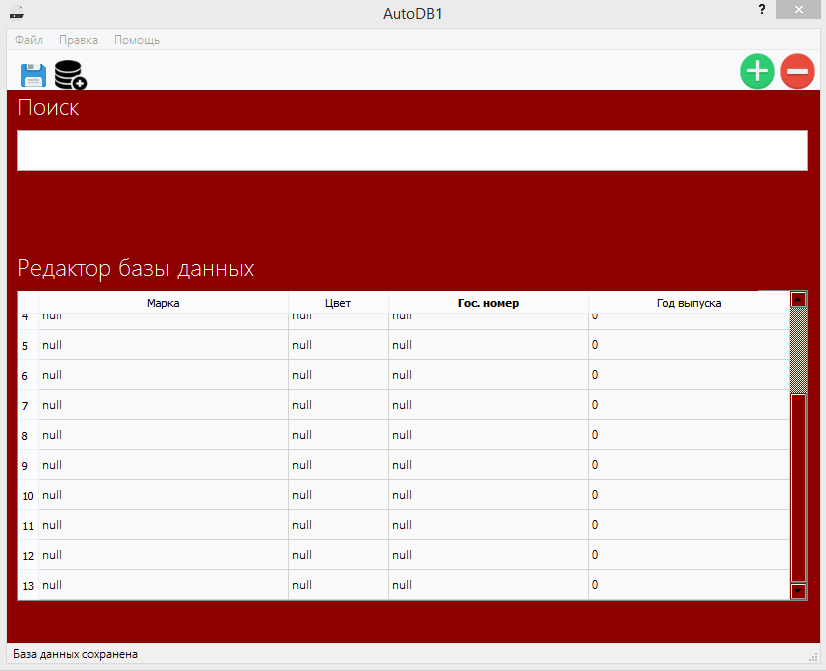
**Окно редактирования базы данных**

****

**Поиск по базе данных**

****

**Работа программы**

****