**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Кафедра информационных систем управления

Чернышева Ксения Юрьевна

Программирование Мобильных и Встраиваемых Систем

Отчет по лабораторной работе №9

## студентки 3 курса 12 группы

**Преподаватель**

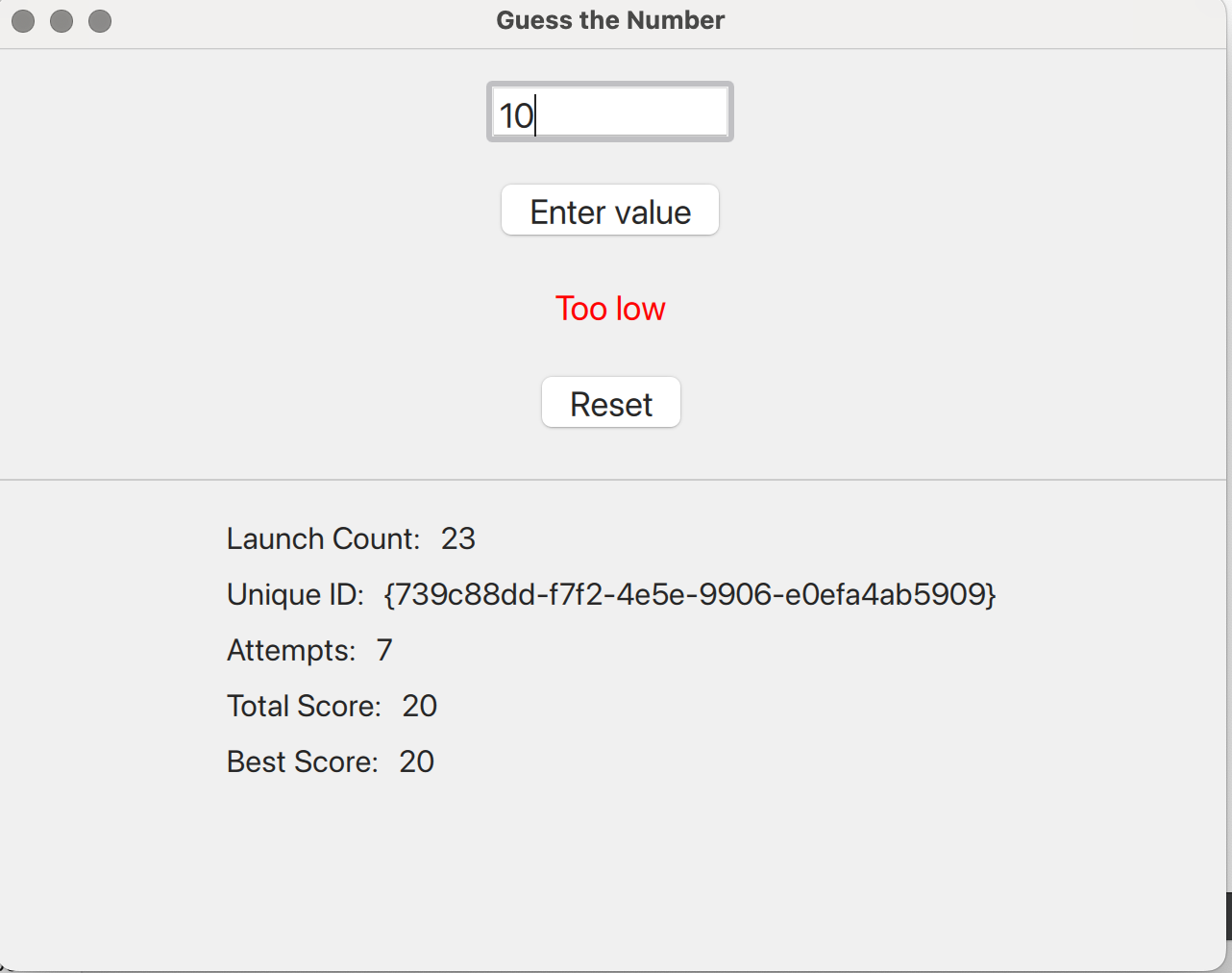
**Давидовская Мария Ивановна**

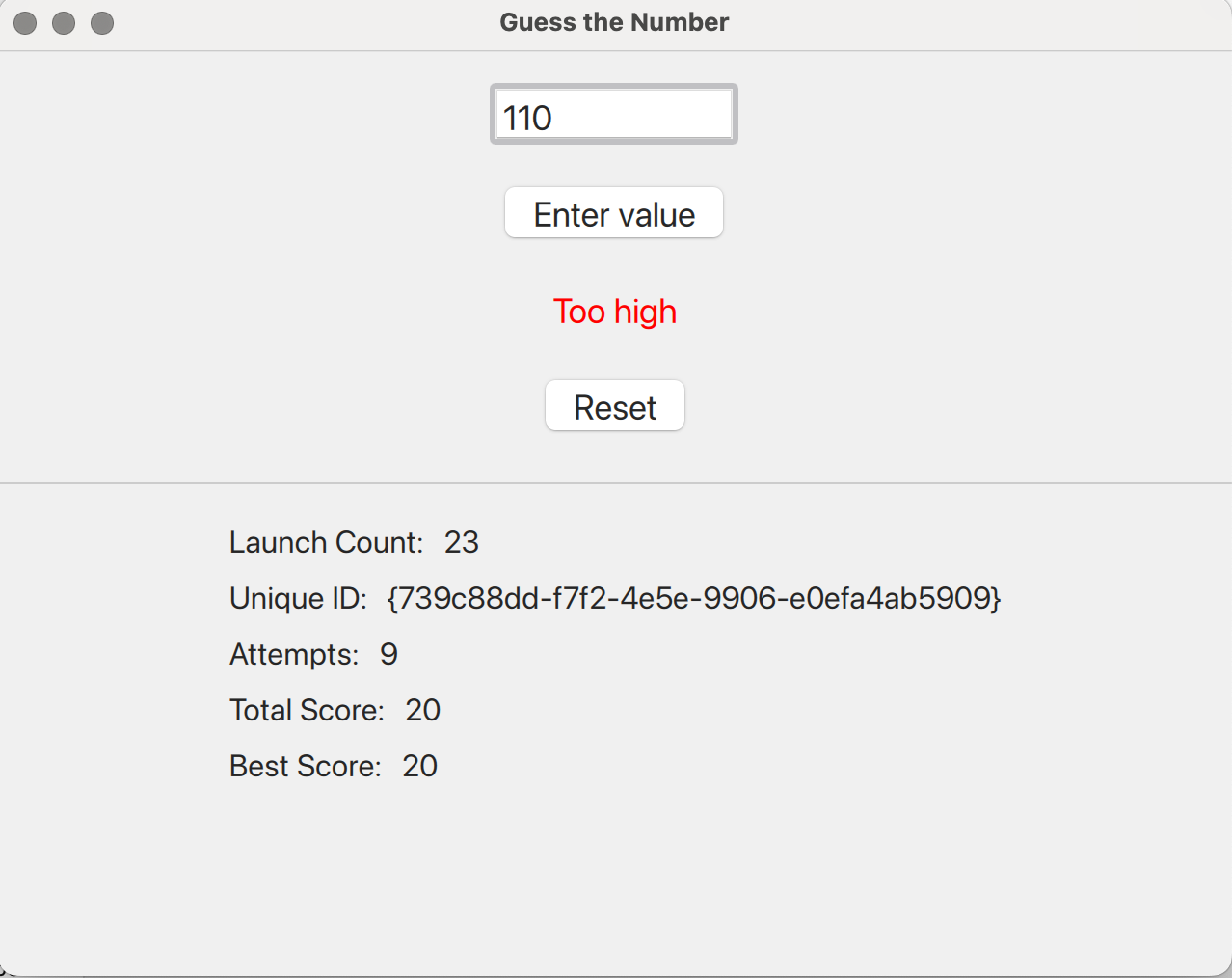
**Минск 2024**

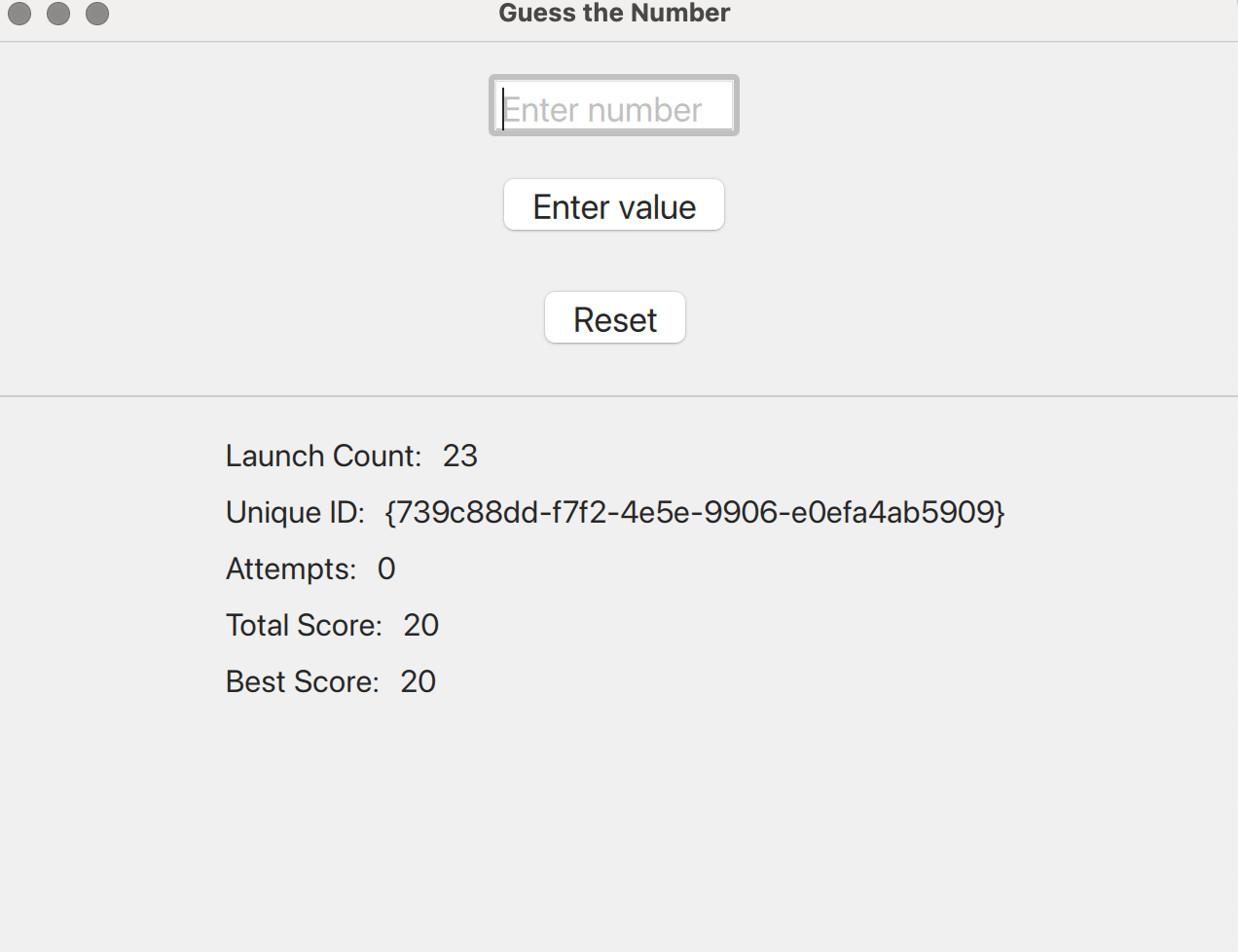
**Задание 2**

**Реализовать на языке C++ мобильное приложение «Угадай число» для ОС Аврора по аналогии с приложением для ОС Android. Требования к проект**

Реализовала







код:  



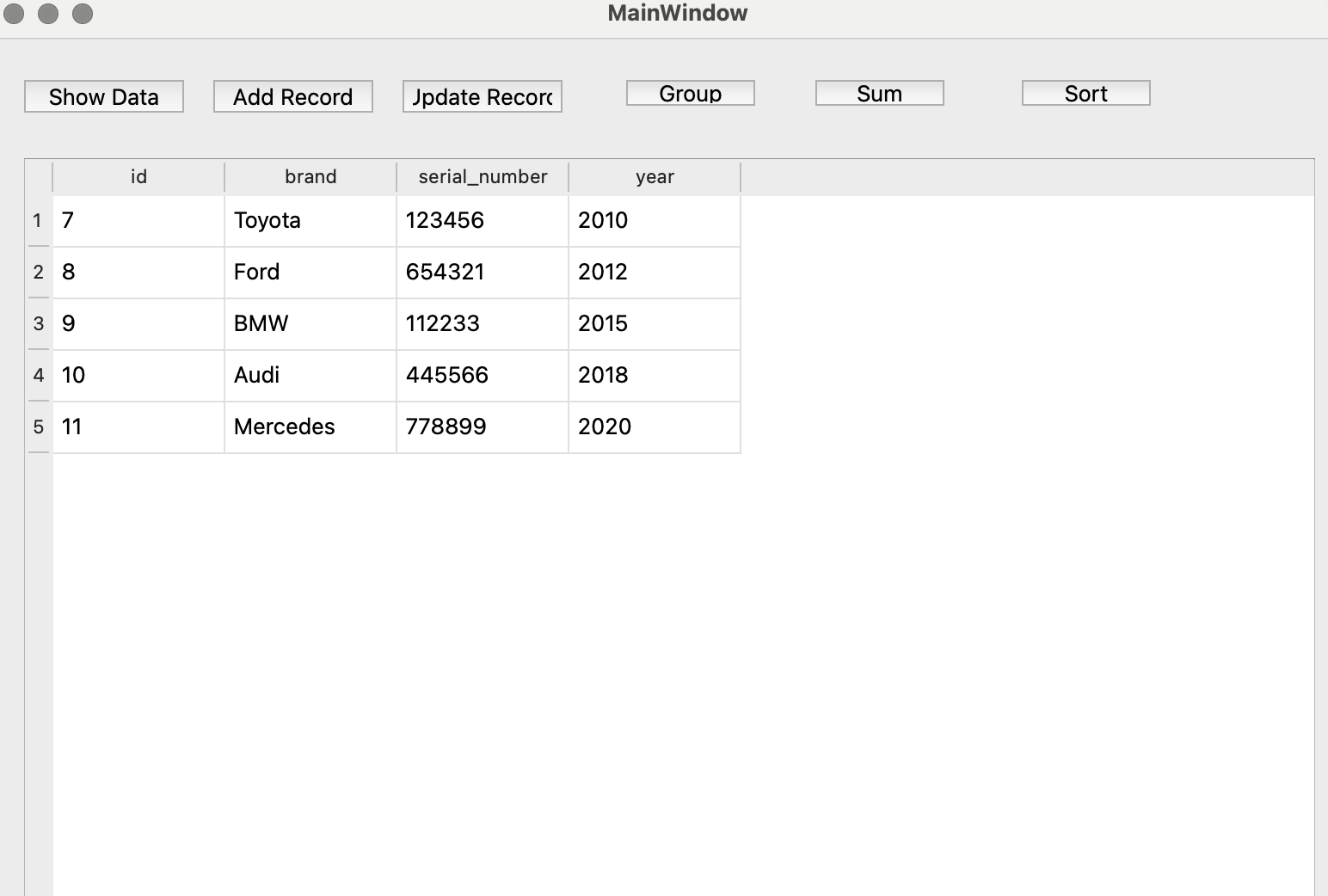


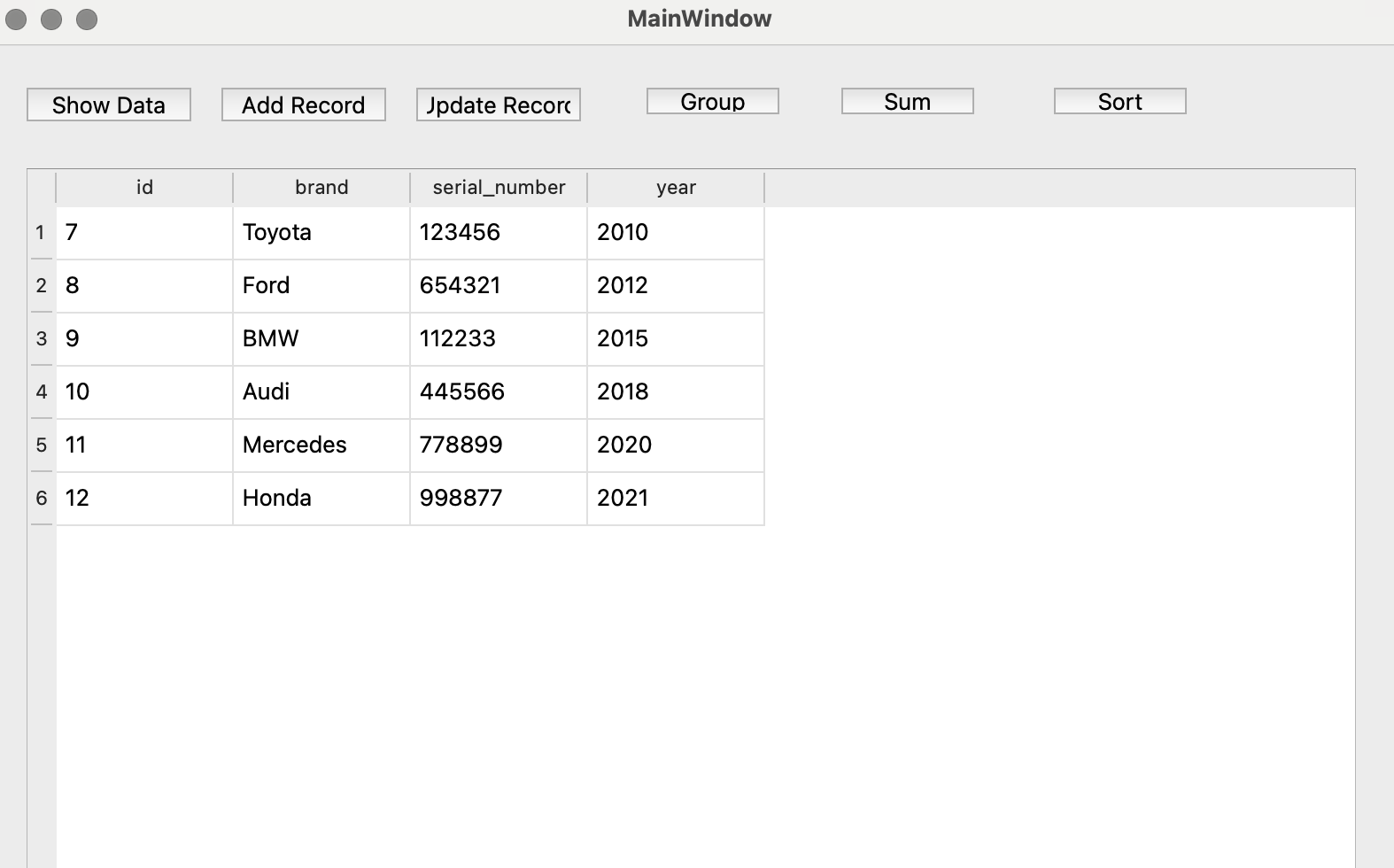


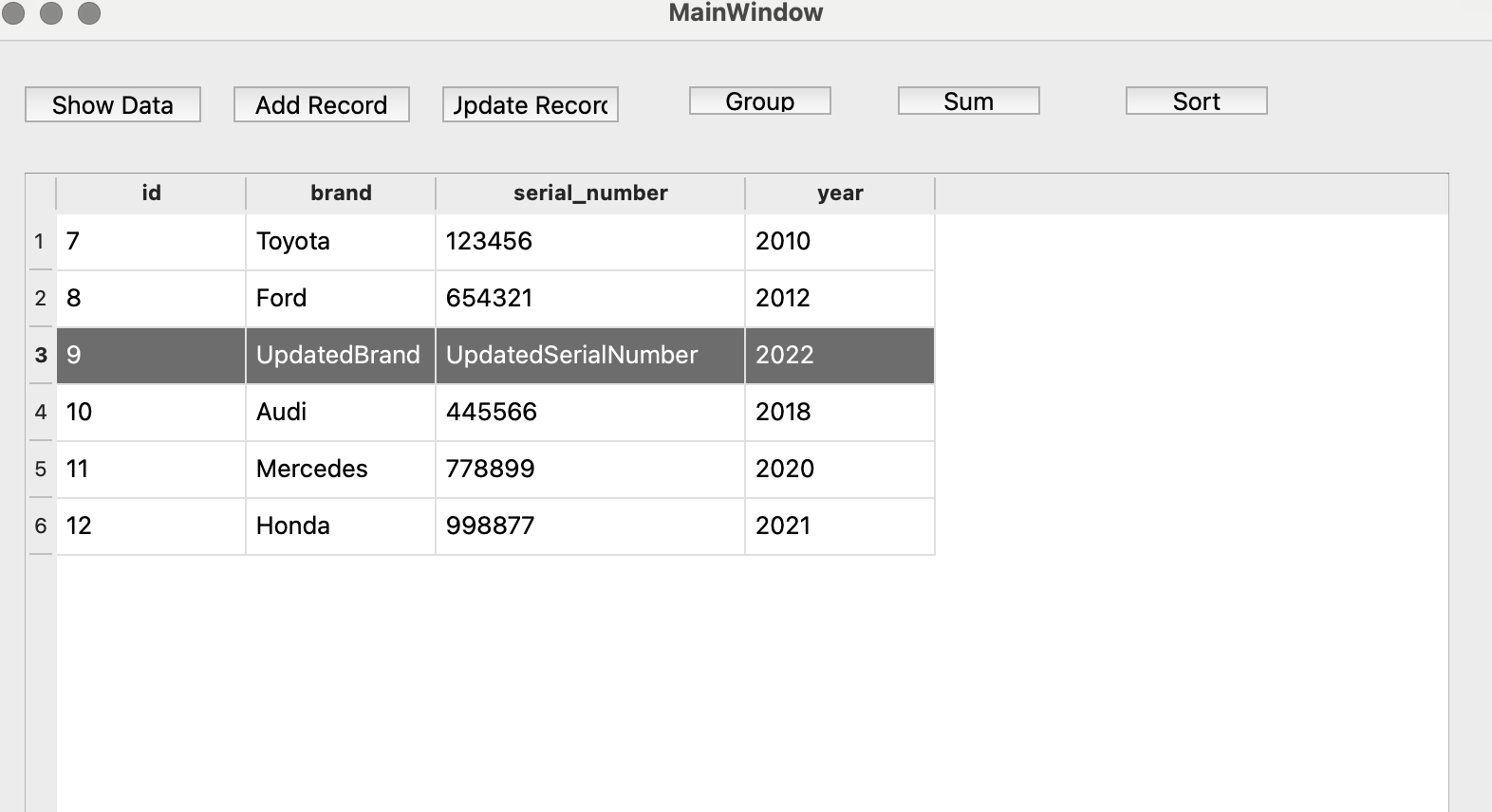


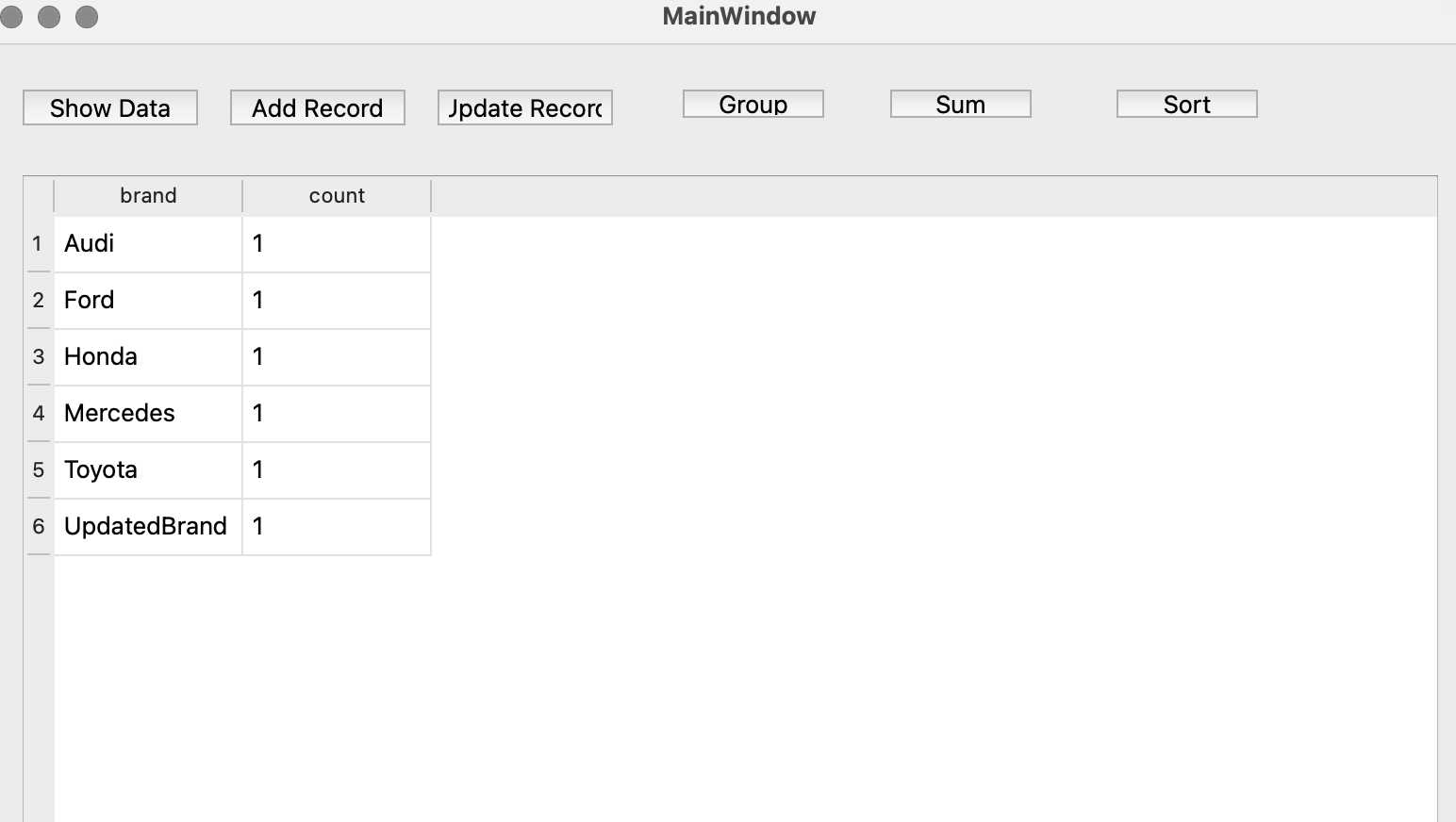

**Задание 3**

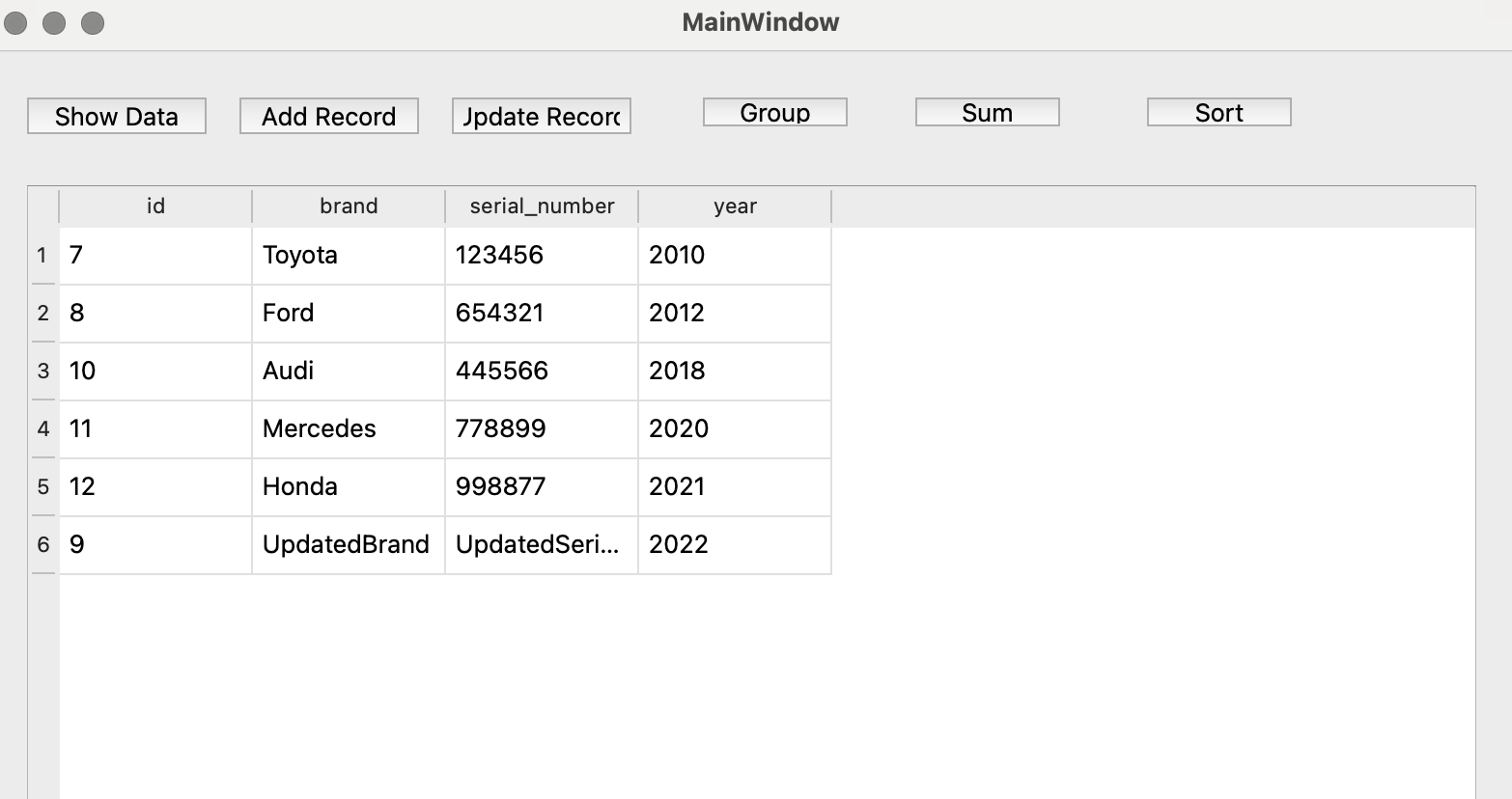
**Реализовать на языке C++ мобильное приложение для ОС Аврора с хранением данных в БД SQlite согласно заданию 4.4 из лабораторной работы 6 и Вашему варианту.**

****

****

****

****

****

**Контрольные вопросы**

1. Основные этапы установки и развертывания Aurora SDK

*Скачивание SDK: Получите установочные файлы с официального сайта Aurora.*

*Установка зависимостей: Установите необходимые зависимости, такие как компиляторы и библиотеки.*

*Установка SDK: Запустите установщик и следуйте инструкциям на экране.*

*Настройка среды разработки: Настройте переменные окружения и интеграцию с используемой IDE.*

*Создание нового проекта: Используйте шаблоны проекта SDK для создания и настройки нового проекта.*

*Сборка и развертывание: Соберите проект с помощью предоставленных инструментов и разверните его на целевой платформе.*

2. Роль Build Engine в Aurora SDK

*Build Engine в Aurora SDK выполняет следующие функции:*

* *Компиляция кода: Преобразует исходный код в исполняемые файлы.*
* *Управление зависимостями: Обрабатывает зависимости проекта и подключаемые библиотеки.*
* *Автоматизация сборки: Автоматизирует процесс сборки проекта, включая тестирование и генерацию артефактов.*
* *Оптимизация: Применяет оптимизации для улучшения производительности и уменьшения размера конечного продукта.*

3. Пример создания форматированного текста на языке QML

import QtQuick 2.15

import QtQuick.Controls 2.15

ApplicationWindow {

visible: true

width: 640

height: 480

Text {

text: "<b>Bold Text</b> <i>Italic Text</i> <font color='red'>Red Text</font>"

anchors.centerIn: parent

wrapMode: Text.WordWrap

textFormat: Text.RichText

}

}

4. Приведите примеры TextField и обработки ввода.

import QtQuick 2.15

import QtQuick.Controls 2.15

ApplicationWindow {

visible: true

width: 640

height: 480

TextField {

id: inputField

width: 200

placeholderText: "Enter text"

onAccepted: {

console.log("Input accepted:", text)

}

}

}

5. Приведите пример анимации элементов на языке QML.

import QtQuick 2.15

import QtQuick.Controls 2.15

ApplicationWindow {

visible: true

width: 640

height: 480

Rectangle {

width: 100

height: 100

color: "blue"

PropertyAnimation {

target: parent

property: "x"

from: 0

to: 500

duration: 2000

loops: Animation.Infinite

}

}

}

6. Приведите пример создания меню в приложении для ОС Аврора.

import QtQuick 2.15

import QtQuick.Controls 2.15

ApplicationWindow {

visible: true

width: 640

height: 480

MenuBar {

Menu {

title: "File"

MenuItem {

text: "Open"

onTriggered: console.log("Open triggered")

}

MenuItem {

text: "Save"

onTriggered: console.log("Save triggered")

}

}

Menu {

title: "Edit"

MenuItem {

text: "Cut"

onTriggered: console.log("Cut triggered")

}

MenuItem {

text: "Copy"

onTriggered: console.log("Copy triggered")

}

MenuItem {

text: "Paste"

onTriggered: console.log("Paste triggered")

}

}

}

}

7. Приведите пример создания диалогового окна в приложении для ОС Аврора.

import QtQuick 2.15

import QtQuick.Controls 2.15

ApplicationWindow {

visible: true

width: 640

height: 480

Button {

text: "Show Dialog"

onClicked: dialog.open()

}

Dialog {

id: dialog

title: "Dialog Title"

standardButtons: Dialog.Ok | Dialog.Cancel

onAccepted: console.log("Dialog accepted")

onRejected: console.log("Dialog rejected")

Text {

text: "This is a dialog"

width: 200

}

}

}

8. Охарактеризуйте QTObject.

QObject — это базовый класс всех объектов Qt. Его основные характеристики:

* Сигналы и слоты: Обеспечивает механизм связи между объектами.
* Иерархия объектов: Поддерживает иерархическую структуру объектов, где каждый объект может иметь родителя и детей.
* Свойства: Поддерживает систему свойств, которые могут быть динамически добавлены и изменены.
* События и обработка событий: Позволяет объектам отправлять и получать события.
* Жизненный цикл управления: Упрощает управление объектами, автоматически удаляя дочерние объекты при удалении родителя.

9. Перечислите особенности сигналов и слотов.

Сигналы: Сигналы — это механизм уведомления о событиях. Они не имеют реализации и объявляются с помощью ключевого слова signals.

Слоты: Слоты — это функции, которые могут быть вызваны в ответ на сигналы. Они могут быть объявлены как обычные функции или с ключевым словом slots.

Автоматическое подключение: Сигналы и слоты могут быть автоматически подключены при помощи механизма connect().

Типизированные параметры: Сигналы и слоты могут передавать данные различного типа.

Различные потоки: Сигналы и слоты могут работать между объектами, находящимися в разных потоках.

10. Какой метод используется для выполнения SQL-запросов в мобильном приложении для ОС Аврора?

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlError>

#include <QDebug>

QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("example.db");

if (db.open()) {

QSqlQuery query;

if (query.exec("SELECT \* FROM example\_table")) {

while (query.next()) {

QString value = query.value(0).toString();

qDebug() << value;

}

} else {

qDebug() << "Query error:" << query.lastError();

}

} else {

qDebug() << "Database connection error:" << db.lastError();

}

Для выполнения SQL-запросов в мобильном приложении для ОС Аврора используется класс QSqlQuery из модуля QtSql