МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №6

**«Разработка консольного приложения в Windows»**

Выполнил:

студент 3 курса

группы ПО-9

Мисиюк Алексей Сергеевич

Проверил:

Козик И. Д.

**Цель работы**: отработать навыки по созданию консольных приложений в Windows, используя C++.

**Вариант №5**

Создать консольную программу для работы с базой данных. Программа должна уметь выводить в консоль данные из БД, записывать новые данные, а также редактировать и удалять уже существующие.

**Код программы**

**main.cpp**

/\*

\* Вариант #5

\* Создать консольную программу для работы с базой данных.

\* Программа должна уметь выводить в консоль данные из БД,

\* записывать новые данные, а также редактировать и удалять уже существующие.

\*/

#include "ShellWrapper.h"

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

{

try {

std::string dbFilename;

if (argc > 1) {

dbFilename = argv[1];

}

else {

cout << "Enter dbFilename as a program argument!\n";

return 0;

}

ShellWrapper sw(dbFilename);

sw.Run();

}

catch (const std::exception& e) {

std::cerr << "Exception caught: " << e.what() << std::endl;

return 10;

}

catch (...) {

std::cerr << "Unknown exception caught." << std::endl;

return 100;

}

return 0;

}

**ShellWrapper.h**

#pragma once

#include <iostream>

#include <string>

#include "Car.h"

using namespace std;

class ShellWrapper {

public:

ShellWrapper(const string& dbName);

void Run();

private:

Car carTable;

};

**Car.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <sqlite3.h>

using namespace std;

class Car {

public:

Car(const std::string& dbName);

~Car();

bool CreateTable();

bool InsertCar(const string& brandModel, int year, const string& color, int mileage);

bool UpdateCar(int id, const string& brandModel, int year, const string& color, int mileage);

bool DeleteCar(int id);

vector<string> GetCars();

private:

sqlite3\* db;

};

**ShellWrapper.cpp**

#include "ShellWrapper.h"

using namespace std;

ShellWrapper::ShellWrapper(const string& dbName) : carTable(dbName) {}

void ShellWrapper::Run() {

cout << "Welcome!\n";

int choice;

vector<string> cars\_list;

string brandModel, color;

int id, year, mileage;

while (true) {

cout << endl;

cout << "1. Display all cars\n";

cout << "2. Add a new car\n";

cout << "3. Update car information\n";

cout << "4. Delete a car\n";

cout << "5. Exit\n";

cout << endl << "Choose an action: ";

cin >> choice;

cout << endl;

switch (choice) {

case 1:

cars\_list = carTable.GetCars();

cout << "id\t|\tbrandModel\t|\tyear\t|\tcolor\t|\tmileage\n";

cout << "---------------------------------------------------------------------------------\n";

for (const auto& car : cars\_list) {

cout << car << endl;

}

break;

case 2:

cout << "Enter brand model (text): ";

cin.ignore();

getline(cin, brandModel);

cout << "Enter year (int): ";

cin >> year;

cin.ignore();

cout << "Enter color (string): ";

getline(cin, color);

cout << "Enter mileage (int): ";

cin >> mileage;

cout << endl << (carTable.InsertCar(brandModel, year, color, mileage) ? "Ok." : "Error.") << endl;

break;

case 3:

cout << "Choose a car (id: int): ";

cin >> id;

cin.ignore();

cout << "Enter brand model (text): ";

getline(cin, brandModel);

cout << "Enter year (int): ";

cin >> year;

cin.ignore();

cout << "Enter color (string): ";

getline(cin, color);

cout << "Enter mileage (int): ";

cin >> mileage;

cout << endl << (carTable.UpdateCar(id, brandModel, year, color, mileage) ? "Ok." : "Error.") << endl;

break;

case 4:

cout << "Choose a car (id: int): ";

cin >> id;

cout << endl << (carTable.DeleteCar(id) ? "Ok." : "Error.") << endl;

break;

case 5:

case 0:

cout << "Exiting the program.\n";

return;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

**Car.cpp**

#include "Car.h"

using namespace std;

Car::Car(const string& dbName) {

int rc = sqlite3\_open(dbName.c\_str(), &db);

if (rc != SQLITE\_OK) {

throw runtime\_error("Cannot open database: " + string(sqlite3\_errmsg(db)));

}

this->CreateTable();

}

Car::~Car() {

sqlite3\_close(db);

}

bool Car::CreateTable() {

string checkQuery = "SELECT id FROM cars;";

int checkResult = sqlite3\_exec(db, checkQuery.c\_str(), nullptr, nullptr, nullptr);

if (checkResult == SQLITE\_OK) {

return true;

}

string createQuery = "CREATE TABLE IF NOT EXISTS cars (id INTEGER PRIMARY KEY, brandModel TEXT, year INTEGER, color TEXT, mileage INTEGER);";

int createResult = sqlite3\_exec(db, createQuery.c\_str(), nullptr, nullptr, nullptr);

if (createResult == SQLITE\_OK) {

InsertCar("Toyota Camry", 2022, "Blue", 5000);

InsertCar("Honda Civic", 2021, "Silver", 12000);

InsertCar("Ford Mustang", 2020, "Red", 15000);

}

return createResult == SQLITE\_OK;

}

bool Car::InsertCar(const string& brandModel, int year, const string& color, int mileage) {

string query = "INSERT INTO cars (brandModel, year, color, mileage) VALUES ('" + brandModel + "', " + to\_string(year) + ", '" + color + "', " + to\_string(mileage) + ");";

int rc = sqlite3\_exec(db, query.c\_str(), nullptr, nullptr, nullptr);

return rc == SQLITE\_OK;

}

bool Car::UpdateCar(int id, const string& brandModel, int year, const string& color, int mileage) {

string query = "UPDATE cars SET brandModel='" + brandModel + "', year=" + to\_string(year) + ", color='" + color + "', mileage=" + to\_string(mileage) + " WHERE id=" + to\_string(id) + ";";

int rc = sqlite3\_exec(db, query.c\_str(), nullptr, nullptr, nullptr);

return rc == SQLITE\_OK;

}

bool Car::DeleteCar(int id) {

string query = "DELETE FROM cars WHERE id=" + to\_string(id) + ";";

int rc = sqlite3\_exec(db, query.c\_str(), nullptr, nullptr, nullptr);

return rc == SQLITE\_OK;

}

vector<string> Car::GetCars() {

vector<string> result;

string query = "SELECT \* FROM cars;";

sqlite3\_exec(db, query.c\_str(), [](void\* data, int argc, char\*\* argv, char\*\* /\*azColName\*/) -> int {

string rowData;

for (int i = 0; i < argc; ++i) {

rowData += argv[i];

if (i+1 < argc) rowData += "\t|\t";

}

reinterpret\_cast<vector<string>\*>(data)->emplace\_back(rowData);

return 0;

}, &result, nullptr);

return result;

}

**Пример работы:**

