Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

Отчёт

По лабораторной работе №1

«Класс “Пользователь”»

Выполнил:

студент группы 350502

Манухо Д. А.

Проверила:

Скиба И.Г.

МИНСК 2024

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ](#_Toc136337238) 3

2 СТРУКТУРА ПРОЕКТА3

[3 ЛИСТИНГ КОДА 4](#_Toc136337241)

[4 РЕЗУЛЬТАТ РАБОТЫ](#_Toc136337244) 10

[ЗАКЛЮЧЕНИЕ 1](#_Toc136337245)2

**1 ПОСТАНОВКА ЗАДАЧИ**

Создать класс по выбранной теме. Реализовать CRUD методы. Организовать ввод-вывод данных с использованием потоков ввода cin-cout. Внедрить динамическое выделение и освобождение памяти.

**2 СТРУКТУРА ПРОЕКТА**

В проекте 3 основных файла:

main.cpp – основной файл программы.

user.cpp – файл, содержащий описание класса User

usercollection.cpp – файл, содержащий описание класса UserCollection

main – исполняемый файл.

db.db – файл, хранящий в себе базу данных sqlite3.

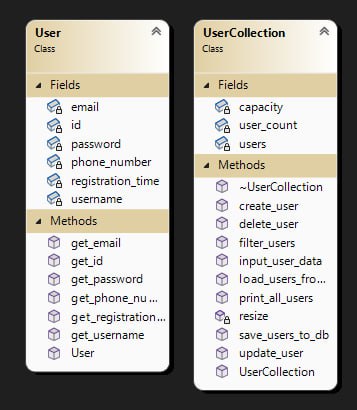
****

Рисунок 2.1 – диаграмма классов проекта

**3 ЛИСТИНГ КОДА**

Файл main.cpp:

#include <sqlite3.h>

#include <chrono>

#include <iostream>

#include <stdexcept>

#include <string>

#include "usercollection.cpp"

void create\_table(sqlite3\* db) {

const char\* sql =

"CREATE TABLE IF NOT EXISTS Users ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

"username TEXT NOT NULL UNIQUE, "

"password TEXT NOT NULL, "

"phone\_number TEXT NOT NULL, "

"email TEXT NOT NULL);";

char\* err\_msg;

int rc = sqlite3\_exec(db, sql, nullptr, nullptr, &err\_msg);

if (rc != SQLITE\_OK) {

std::cerr << "SQL error: " << err\_msg << std::endl;

sqlite3\_free(err\_msg);

}

}

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

if (argc != 2) {

std::cout << "Usage: " << argv[0] << " <database\_name>\n";

return 1;

}

const char\* db\_name = argv[1];

sqlite3\* db;

UserCollection user\_collection;

int rc = sqlite3\_open(db\_name, &db);

if (rc != SQLITE\_OK) {

std::cerr << "Cannot open database: " << sqlite3\_errmsg(db)

<< std::endl;

sqlite3\_close(db);

return 1;

}

create\_table(db);

user\_collection.load\_users\_from\_db(db);

int choice;

do {

std::cout << "Menu: \n";

std::cout << "1) Print all users\n";

std::cout << "2) Add new user\n";

std::cout << "3) Update user\n";

std::cout << "4) Delete user\n";

std::cout << "5) Save users to database\n";

std::cout << "6) Filter users by argument\n";

std::cout << "7) Exit\n";

std::cout << "Choose an option: ";

std::cin >> choice;

switch (choice) {

case 1:

user\_collection.print\_all\_users();

break;

case 2:

user\_collection.input\_user\_data();

break;

case 3: {

int id;

std::cout << "Enter id to update: ";

std::cin >> id;

std::string username, password, phone\_number, email;

std::cout << "Enter new username: ";

std::cin >> username;

std::cout << "Enter new password: ";

std::cin >> password;

std::cout << "Enter new phone number: ";

std::cin >> phone\_number;

std::cout << "Enter new email: ";

std::cin >> email;

try {

user\_collection.update\_user(id, username, password,

phone\_number, email);

std::cout << "User updated successfully." << std::endl;

} catch (const std::runtime\_error& e) {

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

}

break;

}

case 4: {

int id;

std::cout << "Enter id to delete: ";

std::cin >> id;

try {

user\_collection.delete\_user(id);

std::cout << "User deleted successfully." << std::endl;

} catch (const std::runtime\_error& e) {

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

}

break;

}

case 5:

user\_collection.save\_users\_to\_db(db);

std::cout << "Users saved to database." << std::endl;

break;

case 6: {

std::string username, email, phone\_number;

std::cout << "Enter username to filter (leave empty to skip): ";

std::cin.ignore();

std::getline(std::cin, username);

std::cout << "Enter email to filter (leave empty to skip): ";

std::getline(std::cin, email);

std::cout

<< "Enter phone number to filter (leave empty to skip): ";

std::getline(std::cin, phone\_number);

auto filtered\_users =

user\_collection.filter\_users(username, email, phone\_number);

if (filtered\_users.empty()) {

std::cout << "No users found matching the parameters."

<< std::endl;

} else {

for (const auto& user : filtered\_users) {

std::cout

<< "ID: " << user->get\_id() << "\n"

<< "Username: " << user->get\_username() << "\n"

<< "Email: " << user->get\_email() << ", "

<< "Phone: " << user->get\_phone\_number() << "\n"

<< "Registration Time: "

<< user->get\_registration\_time() << "\n";

}

}

break;

}

case 7:

break;

default:

std::cout << "Invalid choice." << std::endl;

}

} while (choice != 7);

sqlite3\_close(db);

return 0;

}

Файл user.cpp:

#include <chrono>

#include <iostream>

#include <stdexcept>

#include <string>

class User {

private:

int id;

std::string username;

std::string password;

std::string phone\_number;

std::string email;

std::chrono::system\_clock::time\_point registration\_time;

public:

User(int id, std::string username, std::string password,

std::string phone\_number, std::string email)

: id(id),

username(username),

password(password),

phone\_number(phone\_number),

email(email) {

this->registration\_time = std::chrono::system\_clock::now();

}

std::string get\_username() const { return username; }

std::string get\_password() const { return password; }

std::string get\_phone\_number() const { return phone\_number; }

std::string get\_email() const { return email; }

std::string get\_registration\_time() const {

std::time\_t time =

std::chrono::system\_clock::to\_time\_t(registration\_time);

return std::ctime(&time);

}

int get\_id() const { return id; }

};

Файл usercollection.cpp:

#include <sqlite3.h>

#include <chrono>

#include <iostream>

#include <stdexcept>

#include <string>

#include "user.cpp"

class UserCollection {

private:

User\*\* users;

int user\_count;

int capacity;

void resize() {

if (capacity == 0) {

capacity = 1;

} else {

capacity \*= 2;

}

users = static\_cast<User\*\*>(realloc(users, capacity \* sizeof(User\*)));

if (!users) {

throw std::runtime\_error("Memory allocation failed!");

}

}

public:

UserCollection() : users(nullptr), user\_count(0), capacity(0) {}

void create\_user(int id, const std::string& username,

const std::string& password,

const std::string& phone\_number,

const std::string& email) {

if (user\_count >= capacity) {

resize();

}

users[user\_count++] =

new User(id, username, password, phone\_number, email);

}

std::vector<User\*> filter\_users(const std::string& username = "",

const std::string& email = "",

const std::string& phone\_number = "") {

std::vector<User\*> result;

for (int i = 0; i < user\_count; ++i) {

bool matches = true;

if (!username.empty() && users[i]->get\_username() != username) {

matches = false;

}

if (!email.empty() && users[i]->get\_email() != email) {

matches = false;

}

if (!phone\_number.empty() &&

users[i]->get\_phone\_number() != phone\_number) {

matches = false;

}

if (matches) {

result.push\_back(users[i]);

}

}

return result;

}

void update\_user(int id, const std::string& username,

const std::string& password,

const std::string& phone\_number,

const std::string& email) {

for (int i = 0; i < user\_count; ++i) {

if (users[i]->get\_id() == id) {

delete users[i];

users[i] =

new User(id, username, password, phone\_number, email);

return;

}

}

throw std::runtime\_error("User not found for update!");

}

void delete\_user(int id) {

for (int i = 0; i < user\_count; ++i) {

if (users[i]->get\_id() == id) {

delete users[i];

users[i] = users[--user\_count];

return;

}

}

throw std::runtime\_error("User not found for deletion!");

}

void print\_all\_users() const {

for (int i = 0; i < user\_count; ++i) {

std::cout << "ID: " << users[i]->get\_id() << "\n"

<< "Username: " << users[i]->get\_username() << "\n"

<< "Email: " << users[i]->get\_email() << ", "

<< "Phone: " << users[i]->get\_phone\_number() << "\n"

<< "Registration Time: "

<< users[i]->get\_registration\_time() << "\n";

}

}

void input\_user\_data() {

std::string username, password, phone\_number, email;

int id = user\_count + 1;

std::cout << "Enter username: ";

std::cin >> username;

std::cout << "Enter password: ";

std::cin >> password;

std::cout << "Enter phone number: ";

std::cin >> phone\_number;

std::cout << "Enter email: ";

std::cin >> email;

create\_user(id, username, password, phone\_number, email);

}

void load\_users\_from\_db(sqlite3\* db) {

const char\* sql =

"SELECT id, username, password, phone\_number, email FROM Users;";

sqlite3\_stmt\* stmt;

int rc = sqlite3\_prepare\_v2(db, sql, -1, &stmt, nullptr);

if (rc != SQLITE\_OK) {

throw std::runtime\_error("Failed to prepare statement.");

}

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

int id = sqlite3\_column\_int(stmt, 0);

const char\* username =

reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 1));

const char\* password =

reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 2));

const char\* phone\_number =

reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 3));

const char\* email =

reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 4));

create\_user(id, username, password, phone\_number, email);

}

sqlite3\_finalize(stmt);

}

void save\_users\_to\_db(sqlite3\* db) {

const char\* sql = "DELETE FROM Users;";

sqlite3\_exec(db, sql, nullptr, nullptr, nullptr);

for (int i = 0; i < user\_count; ++i) {

const char\* sql\_insert =

"INSERT INTO Users (id, username, password, phone\_number, "

"email) "

"VALUES (?, ?, ?, ?, ?);";

sqlite3\_stmt\* stmt;

sqlite3\_prepare\_v2(db, sql\_insert, -1, &stmt, nullptr);

sqlite3\_bind\_int(stmt, 1, users[i]->get\_id());

sqlite3\_bind\_text(stmt, 2, users[i]->get\_username().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 3, users[i]->get\_password().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 4, users[i]->get\_phone\_number().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 5, users[i]->get\_email().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_step(stmt);

sqlite3\_finalize(stmt);

}

}

~UserCollection() {

for (int i = 0; i < user\_count; ++i) {

delete users[i];

}

free(users);

}

};

**4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Пользовательское меню для выбора режима работы программы представлено на рисунке 4.1

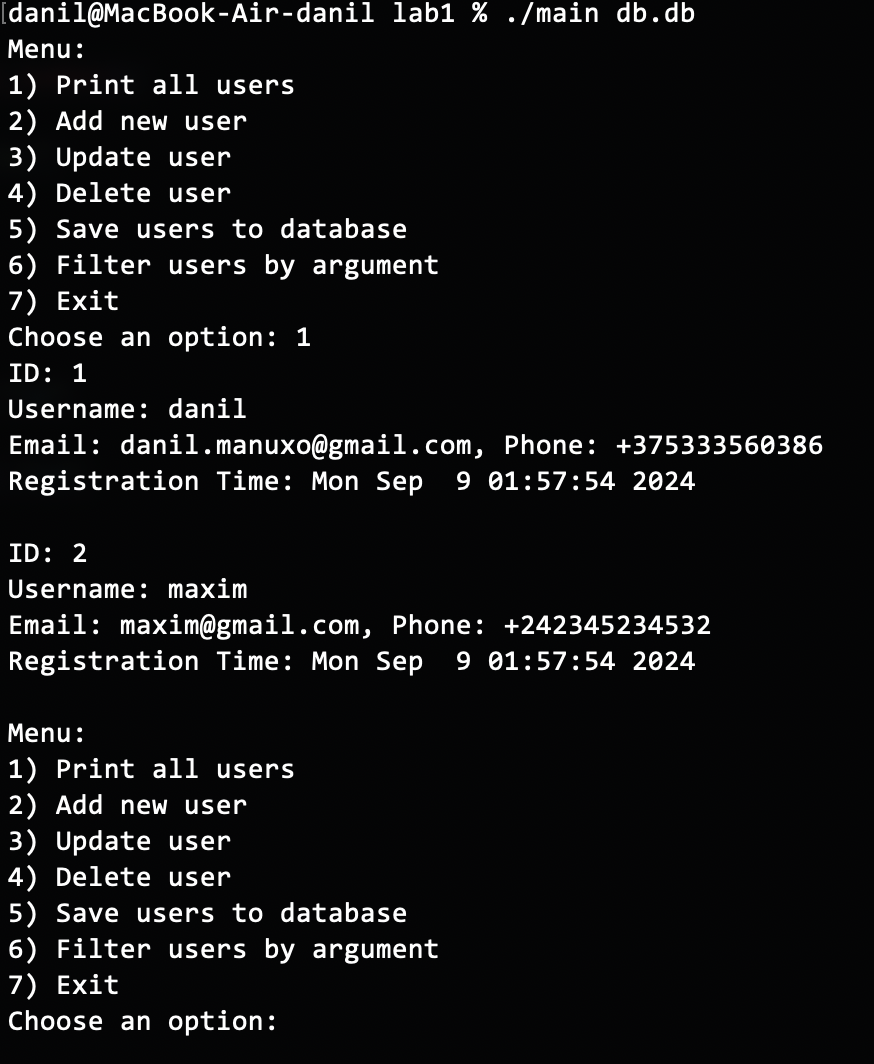


Рисунок 4.1 – Пример начала работы программы

**ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы была успешна реализована программа, позволяющая пользователю создавать, редактировать и считывать текстовые файлы, а также удалять уже созданный файл.

В ходе выполнения был написан класс, содержащий все основные методы для выполнения CRUD-операций.

Программа была протестирована и показала высокую эффективность в выполнении всех заявленных функций. Пользовательский интерфейс интуитивно понятен и удобен в использовании, что позволяет легко управлять файлами. В ходе тестирования не было выявлено значительных ошибок, что подтверждает надежность и стабильность разработанного решения.

Таким образом, данная лабораторная работа продемонстрировала успешное применение принципов объектно-ориентированного программирования для создания функционального и удобного инструмента для работы с текстовыми файлами.