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 №2

«Дружественные функции и перегрузка операторов»

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)1

**1 ПОСТАНОВКА ЗАДАЧИ**

Создать класс по выбранной теме. Реализовать CRUD методы. Написать дружественную функцию. Перегрузить оператор сравнения и еще 2 любых на выбор. Использовать хранилище данных (файлы или база данных).

**2 СТРУКТУРА ПРОЕКТА**

В проекте 5 файлов:

main.cpp – основной файл программы.

user.cpp – файл, содержащий описание класса User.

usercollection.cpp – файл, содержащий описание класса UserCollection.

user.h – заголовочный файл для класса User.

Usercollection.h – заголовочный файл для класса UserCollection.

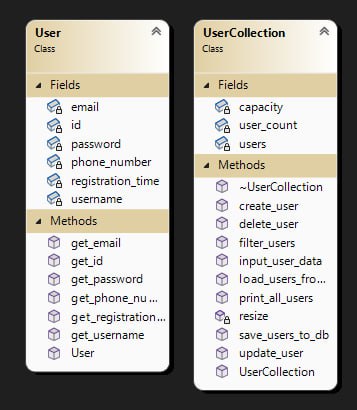
****

Рисунок 2.1 – Диаграмма классов проекта

**3 ЛИСТИНГ КОДА**

Файл main.cpp:

#include <iostream>

#include "usercollection.h"

void create\_table(sqlite3\* db) {

const char\* sql =

"CREATE TABLE IF NOT EXISTS Users ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

"username TEXT NOT NULL, "

"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);

}

}

void update\_user(UserCollection& user\_collection) {

int id;

std::string username;

std::string name;

std::string password;

std::string phone\_number;

std::string email;

std::cout << "Enter user ID to update: ";

std::cin >> id;

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;

if (user\_collection.update\_user(id, username, password, phone\_number,

email)) {

std::cout << "User updated successfully.\n";

} else {

std::cout << "User not found.\n";

}

}

void delete\_user(UserCollection& user\_collection) {

int id;

std::cout << "Enter user ID to delete: ";

std::cin >> id;

if (user\_collection.delete\_user(id)) {

std::cout << "User deleted successfully.\n";

} else {

std::cout << "User not found.\n";

}

}

void filter\_users(const UserCollection& user\_collection) {

std::string email;

std::string phone\_number;

std::cout << "Enter email (or leave empty): ";

std::cin >> email;

std::cout << "Enter phone number (or leave empty): ";

std::cin >> phone\_number;

auto users = user\_collection.filter\_users(email, phone\_number);

if (!users.empty()) {

for (const auto& user : users) {

std::cout << "id: " << user->get\_id() << "\n"

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

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

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

}

} else {

std::cout << "No users found with the given criteria.\n";

}

}

void input\_data(UserCollection& user\_collection) {

std::string username;

std::string email;

std::string phone\_number;

std::string password;

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;

user\_collection.input\_user\_data(username, password, email, phone\_number);

}

void compare\_user\_by\_name(const UserCollection& user\_collection) {

std::string input\_username;

std::cout << "Enter the username to compare: ";

std::cin >> input\_username;

User temp\_user(-1, input\_username, "", "", "");

bool found = false;

for (const auto& user : user\_collection.filter\_users()) {

if (\*user == temp\_user) {

std::cout << "User with username " << input\_username

<< " exists in the collection.\n";

found = true;

break;

}

}

if (!found) {

std::cout << "No user found with username " << input\_username << ".\n";

}

}

void handle\_choice(int choice, UserCollection& user\_collection, sqlite3\* db) {

switch (choice) {

case 1:

print\_all\_users(user\_collection);

break;

case 2:

input\_data(user\_collection);

break;

case 3:

update\_user(user\_collection);

break;

case 4:

delete\_user(user\_collection);

break;

case 5:

filter\_users(user\_collection);

break;

case 6:

compare\_user\_by\_name(user\_collection);

break;

case 7:

std::cout << "Exiting..." << std::endl;

sqlite3\_close(db);

exit(0);

default:

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

}

}

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

if (argc != 2) {

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

return 1;

}

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

sqlite3\* db;

UserCollection user\_collection;

if (sqlite3\_open(db\_name, &db) != SQLITE\_OK) {

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

<< std::endl;

return 1;

}

create\_table(db);

user\_collection.set\_database(db);

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

int choice;

do {

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

<< "1) Print all users\n"

<< "2) Add new user\n"

<< "3) Update user\n"

<< "4) Delete user\n"

<< "5) Filter users by argument\n"

<< "6) Compare users by name\n"

<< "7) Exit\n"

<< "Choose an option: ";

std::cin >> choice;

handle\_choice(choice, user\_collection, db);

} 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 "usercollection.h"

#include <iostream>

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

const std::string& password,

const std::string& phone\_number,

const std::string& email) {

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

\*this += new\_user;

if (!save\_users\_to\_db()) {

std::cerr << "Failed to save user to database.\n";

}

}

void UserCollection::set\_database(sqlite3\* database) { this->db = database; }

bool UserCollection::delete\_user(int id) {

for (size\_t i = 0; i < users.size(); ++i) {

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

users.erase(users.begin() + i);

std::cout << "User deleted successfully.\n";

save\_users\_to\_db();

return true;

}

}

std::cout << "User not found\n";

return false;

}

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

const std::string& password,

const std::string& phone\_number,

const std::string& email) {

for (auto& user : users) {

if (user->get\_id() == id) {

user = std::make\_unique<User>(id, username, password, phone\_number,

email);

save\_users\_to\_db();

return true;

}

}

return false;

}

std::vector<User\*> UserCollection::filter\_users(

const std::string\_view email, const std::string\_view phone\_number) const {

std::vector<User\*> result;

for (const auto& user : users) {

if ((email.empty() || user->get\_email() == email) &&

(phone\_number.empty() ||

user->get\_phone\_number() == phone\_number)) {

result.push\_back(user.get());

}

}

return result;

}

void print\_all\_users(const UserCollection& user\_collection) {

for (const auto& user : user\_collection.users) {

std::cout << \*user;

}

}

void UserCollection::input\_user\_data(const std::string& username,

const std::string& password,

const std::string& email,

const std::string& phone\_number) {

int id = users.size() + 1;

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

}

void UserCollection::load\_users\_from\_db() {

const char\* sql =

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

sqlite3\_stmt\* stmt;

if (sqlite3\_prepare\_v2(db, sql, -1, &stmt, nullptr) != SQLITE\_OK) {

exit(-1);

}

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

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

std::string username =

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

std::string password =

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

std::string phone\_number =

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

std::string email =

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

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

}

sqlite3\_finalize(stmt);

}

bool UserCollection::save\_users\_to\_db() const {

for (const auto& user : users) {

const char\* sql\_insert\_or\_update =

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

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

"ON CONFLICT(id) DO UPDATE SET "

"username = excluded.username, "

"password = excluded.password, "

"phone\_number = excluded.phone\_number, "

"email = excluded.email;";

sqlite3\_stmt\* stmt;

if (sqlite3\_prepare\_v2(db, sql\_insert\_or\_update, -1, &stmt, nullptr) !=

SQLITE\_OK) {

std::cerr << "Failed to prepare statement: " << sqlite3\_errmsg(db)

<< "\n";

return false;

}

sqlite3\_bind\_int(stmt, 1, user->get\_id());

sqlite3\_bind\_text(stmt, 2, user->get\_username().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 3, user->get\_password().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 4, user->get\_phone\_number().c\_str(), -1,

SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 5, user->get\_email().c\_str(), -1,

SQLITE\_STATIC);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

std::cerr << "Failed to execute statement: " << sqlite3\_errmsg(db)

<< "\n";

sqlite3\_finalize(stmt);

return false;

}

sqlite3\_finalize(stmt);

}

return true;

}

UserCollection& UserCollection::operator+=(const User& user) {

users.emplace\_back(std::make\_unique<User>(

user.get\_id(), user.get\_username(), user.get\_password(),

user.get\_phone\_number(), user.get\_email()));

return \*this;

}

**4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Пользовательское меню для выбора режима работы программы представлено на рисунке 4.1

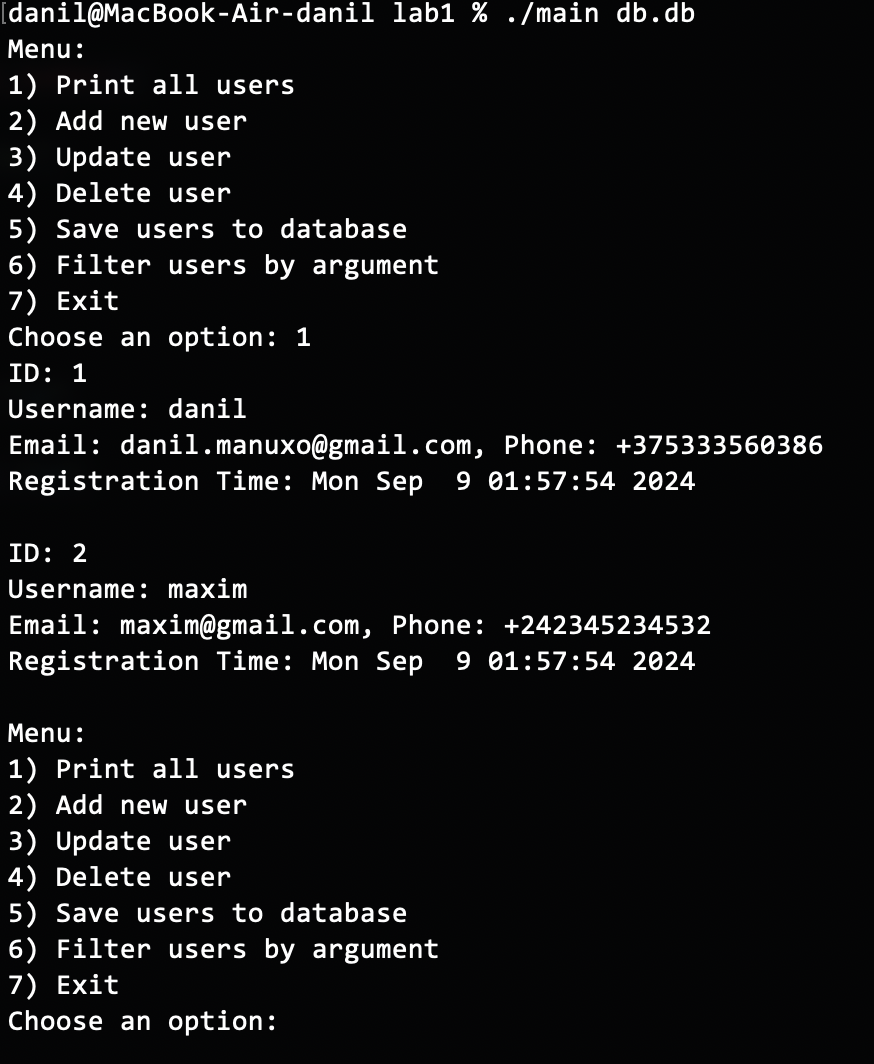


Рисунок 4.1 – Пример начала работы программы

Сохраненные в базу данных данные о пользователях можно посмотреть на рисунке 4.2.

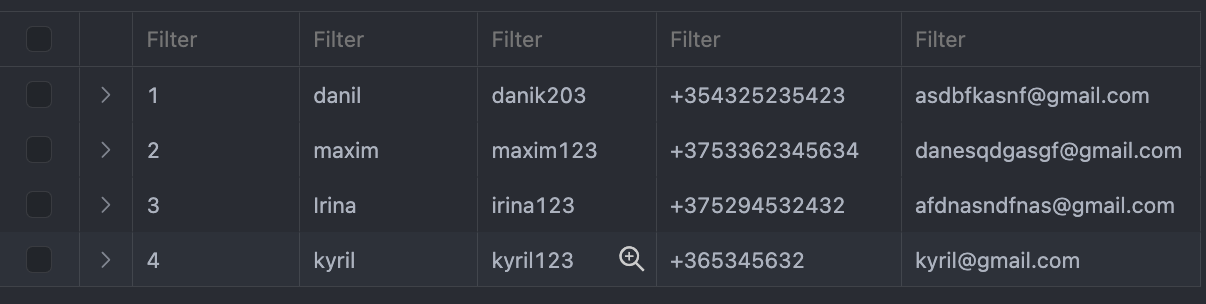


Рисунок 4.2 – База данных

**ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы была успешна реализована программа, позволяющая успешно создавать и сохранять пользователей.

В ходе выполнения был написан класс, содержащий все основные методы для выполнения CRUD-операций. Также была создана дружественная функция и переопределены некоторые операторы.

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 программ.