|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Обобщённое программирование и шаблоны»**

**ДИСЦИПЛИНА: «Высокоуровневое программирование»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-22Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Козина А.В. )  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга , 2021 | | |

**Цель:** приобретение практических навыков и знаний по обобщённому программированию.

**Задачи:**

1. Изучить основы и принципы обобщённого программирования;
2. Познакомиться с шаблонами функций;
3. Научиться создавать универсальные функции;
4. Познакомиться с шаблонами классов;
5. Получение навыков работы с шаблонами типа и шаблонами значения;

**Вариант 8**

**Задание:**

Просмотрите свой код программы и найдите места, где функции отличаются только типами параметров, с которыми они работают. Перепишите эти функции через шаблоны. Как правило, эти функции являются функциями сортировки и фильтрации. Но, в вашей работе могут найтись и другие функции, которые отличаются только типом.

**UML-диаграмма классов:**



**Рисунок 1.1.** UML-диаграмма классов



**Рисунок 1.2.** UML-диаграмма классов



**Рисунок 1.3.** UML-диаграмма классов



**Рисунок 1.4.** UML-диаграмма классов

**Листинг:**

**Add.h**

#ifndef ADD\_H

#define ADD\_H

#include "Entity.h"

namespace KMK

{

template<typename T>

void Add(T\*\*& entities, size\_t& size, T\* newElement)

{

T\*\* temp = new T \* [size + 1]{};

for (Iteration i{}; i < size; ++i)

{

temp[i] = entities[i];

}

temp[size] = newElement;

delete[] entities;

entities = temp;

++size;

}

}

#endif // !ADD\_H

**AuthorizationItem.h**

#ifndef AUTHORIZATION\_ITEM\_H

#define AUTHORIZATION\_ITEM\_H

#include "AbstractMenuItem.h"

#include "Storage.h"

namespace KMK

{

class AuthorizstionItem : public MenuItem

{

public:

AuthorizstionItem(std::string itemName, Storage\* storage);

int Run();

private:

Storage\* m\_storage{};

};

}

#endif // !AUTHORIZATION\_ITEM\_H

**AuthorizationItem.cpp**

#include "AuthorizationItem.h"

#include <iostream>

#include <Windows.h>

#include "TypeDefinitions.h"

#include "Sort.h"

#include "Add.h"

#include "Remove.h"

#include "Filter.h"

#include <vector>

#include <string>

namespace KMK

{

AuthorizstionItem::AuthorizstionItem(std::string itemName, Storage\* storage) : MenuItem(itemName)

{

m\_storage = storage;

}

int AuthorizstionItem::Run()

{

unsigned short command = 0;

while (command != 2)

{

system("cls");

std::cout << "0. Log in\n";

std::cout << "1. Sign in\n";

std::cout << "2. Exit\n";

std::cout << ">>> ";

std::cin >> command;

std::cin.ignore();

if (command == 0)

{

std::string login{};

std::cout << "Login: ";

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

std::string password{};

std::cout << "Password: ";

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

Id userId = 0;

for (size\_t i{}; i < m\_storage->GetUserListSize(); i++)

{

if (login == m\_storage->m\_userList[i]->GetLogin() &&

password == m\_storage->m\_userList[i]->GetPassword())

{

userId = m\_storage->m\_userList[i]->GetId();

break;

}

}

if (userId != 0)

{

unsigned short userCommand = 0;

std::string userNickname{};

for (size\_t i{}; i < m\_storage->GetUserListSize(); ++i)

{

if (m\_storage->m\_userList[i]->GetId() == userId)

{

userNickname = m\_storage->m\_userList[i]->GetName();

break;

}

}

while (userCommand != 4)

{

system("cls");

std::cout << "Your nickname: " << userNickname << "\n\n";

std::cout << "0. Dialogs\n";

std::cout << "1. Interests\n";

std::cout << "2. Reminders\n";

std::cout << "3. Themes\n";

std::cout << "4. Exit\n";

std::cout << ">>> ";

std::cin >> userCommand;

std::cin.ignore();

if (userCommand == 0)

{

unsigned short dialogCommand = 0;

while (dialogCommand != 2)

{

system("cls");

Sort((Entity\*\*&)m\_storage->m\_dialogueList, m\_storage->GetDialogueListSize(),

OrderMode::ASCENDING, FieldMode::DATE);

struct UserDialog

{

Id dialogUserId{};

std::string dialogUserNickname{};

std::string lastMessage{};

};

std::vector<UserDialog> userDialogs{};

for (size\_t i{}; i < m\_storage->GetDialogueListSize(); ++i)

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userId ||

m\_storage->m\_dialogueList[i]->GetAdresseeId() == userId)

{

size\_t userDialogIndex = 0;

for (size\_t j{}; j < userDialogs.size(); ++j)

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userId &&

m\_storage->m\_dialogueList[i]->GetAdresseeId() == userId &&

userDialogs[j].dialogUserId == userId)

{

break;

}

else if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userDialogs[j].dialogUserId &&

m\_storage->m\_dialogueList[i]->GetOwnerId() != userId ||

m\_storage->m\_dialogueList[i]->GetAdresseeId() == userDialogs[j].dialogUserId &&

m\_storage->m\_dialogueList[i]->GetAdresseeId() != userId)

{

break;

}

++userDialogIndex;

}

if (userDialogIndex < userDialogs.size())

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userId)

{

userDialogs[userDialogIndex].lastMessage = "You";

}

else

{

for (size\_t j{}; j < m\_storage->GetUserListSize(); j++)

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() ==

m\_storage->m\_userList[j]->GetId())

{

userDialogs[userDialogIndex].lastMessage = m\_storage->m\_userList[j]->GetName();

break;

}

}

}

userDialogs[userDialogIndex].lastMessage += ": ";

userDialogs[userDialogIndex].lastMessage += m\_storage->m\_dialogueList[i]->GetContent();

}

else

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userId &&

m\_storage->m\_dialogueList[i]->GetAdresseeId() == userId)

{

std::string message = userNickname;

message += ": ";

message += m\_storage->m\_dialogueList[i]->GetContent();

userDialogs.push\_back({ userId, userNickname, message });

}

else if (m\_storage->m\_dialogueList[i]->GetOwnerId() != userId)

{

std::string dialogUserNickname = "";

for (size\_t j{}; j < m\_storage->GetUserListSize(); j++)

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() ==

m\_storage->m\_userList[j]->GetId())

{

dialogUserNickname = m\_storage->m\_userList[j]->GetName();

break;

}

}

if (dialogUserNickname == "") { continue; }

std::string message = dialogUserNickname;

message += ": ";

message += m\_storage->m\_dialogueList[i]->GetContent();

userDialogs.push\_back({ m\_storage->m\_dialogueList[i]->GetOwnerId(),

dialogUserNickname, message });

}

else if (m\_storage->m\_dialogueList[i]->GetAdresseeId() != userId)

{

std::string dialogUserNickname = "";

for (size\_t j{}; j < m\_storage->GetUserListSize(); j++)

{

if (m\_storage->m\_dialogueList[i]->GetAdresseeId() ==

m\_storage->m\_userList[j]->GetId())

{

dialogUserNickname = m\_storage->m\_userList[j]->GetName();

break;

}

}

if (dialogUserNickname == "") { continue; }

std::string message = dialogUserNickname;

message += ": ";

message += m\_storage->m\_dialogueList[i]->GetContent();

userDialogs.push\_back({ m\_storage->m\_dialogueList[i]->GetAdresseeId(),

dialogUserNickname, message });

}

}

}

}

std::cout << "Your dialogs:\n";

for (size\_t i{}; i < userDialogs.size(); ++i)

{

std::cout << userDialogs[i].dialogUserNickname << "\n";

std::cout << "\t" << userDialogs[i].lastMessage << "\n\n";

}

std::cout << "0. Open dialog\n";

std::cout << "1. Find user\n";

std::cout << "2. Exit\n";

std::cout << ">>> ";

std::cin >> dialogCommand;

std::cin.ignore();

if (dialogCommand == 0)

{

std::string nickname{};

std::cout << "Input nickname: ";

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

Id dialogueUserId = 0;

for (size\_t i{}; i < m\_storage->GetUserListSize(); i++)

{

if (nickname == m\_storage->m\_userList[i]->GetName())

{

dialogueUserId = m\_storage->m\_userList[i]->GetId();

break;

}

}

if (dialogueUserId != 0)

{

unsigned short dialogueCommand = 0;

while (dialogueCommand != 1)

{

system("cls");

for (size\_t i{}; i < m\_storage->GetDialogueListSize(); i++)

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userId &&

m\_storage->m\_dialogueList[i]->GetAdresseeId() == dialogueUserId)

{

std::cout << "You: " << m\_storage->m\_dialogueList[i]->GetContent() << "\n";

}

else if (m\_storage->m\_dialogueList[i]->GetOwnerId() == dialogueUserId &&

m\_storage->m\_dialogueList[i]->GetAdresseeId() == userId)

{

std::cout << nickname << ": " << m\_storage->m\_dialogueList[i]->GetContent() << "\n";

}

}

std::cout << "\n";

std::cout << "0. Write a message\n";

std::cout << "1. Exit\n";

std::cout << ">>> ";

std::cin >> dialogueCommand;

std::cin.ignore();

if (dialogueCommand == 0)

{

std::string message{};

std::cout << "Message: ";

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

size\_t size = m\_storage->GetDialogueListSize();

Add<Dialogue>(m\_storage->m\_dialogueList, size,

new Dialogue{ m\_storage->GetNextDialogueId(),

{systemTime.wDay, systemTime.wMonth, systemTime.wYear},

message, userId, dialogueUserId });

m\_storage->SetDialogueListSize(size);

m\_storage->SaveDialogueList();

}

}

}

else

{

std::cout << "User wasn's found\n";

std::cin.ignore(1);

}

}

else if (dialogCommand == 1)

{

system("cls");

std::string nickname{};

std::cout << "Input nickname: ";

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

size\_t size = m\_storage->GetUserListSize();

Filter((Entity\*\*&)m\_storage->m\_userList, size, nickname, FieldMode::NAME);

m\_storage->SetUserListSize(size);

std::cout << "\nFound users:\n";

for (size\_t i{}; i < m\_storage->GetUserListSize(); ++i)

{

std::cout << m\_storage->m\_userList[i]->GetName() << "\n";

}

m\_storage->LoadUserList();

std::cout << "\nPress enter to return\n";

std::cin.ignore();

}

m\_storage->LoadDialogueList();

}

}

else if (userCommand == 1)

{

unsigned short interestCommand = 0;

while (interestCommand != 3)

{

system("cls");

std::vector<size\_t> indexes{};

std::cout << "Your interests:\n";

for (size\_t i{}; i < m\_storage->GetInterestListSize(); ++i)

{

if (m\_storage->m\_interestList[i]->GetOwnerId() == userId)

{

std::cout << indexes.size() << ". " << m\_storage->m\_interestList[i]->GetContent() << "\n";

indexes.push\_back(i);

}

}

std::cout << "\n0. Create new interest\n";

std::cout << "1. Delete interest\n";

std::cout << "2. Find by interest\n";

std::cout << "3. Exit\n";

std::cout << ">>> ";

std::cin >> interestCommand;

std::cin.ignore();

if (interestCommand == 0)

{

std::cout << "Input interest: ";

std::string interest{};

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

size\_t size = m\_storage->GetInterestListSize();

Add<Interest>(m\_storage->m\_interestList, size,

new Interest{ m\_storage->GetNextInterestId(),

{systemTime.wDay, systemTime.wMonth, systemTime.wYear},

interest, userId });

m\_storage->SetInterestListSize(size);

m\_storage->SaveInterestList();

}

else if (interestCommand == 1)

{

std::cout << "Input interest number: ";

size\_t interestNumber{};

std::cin >> interestNumber;

std::cin.ignore();

size\_t size = m\_storage->GetInterestListSize();

Remove((Entity\*\*&)m\_storage->m\_interestList, size,

m\_storage->m\_interestList[indexes[interestNumber]]->GetId());

m\_storage->SetInterestListSize(size);

m\_storage->SaveInterestList();

}

else if (interestCommand == 2)

{

std::cout << "\nInput interest: ";

std::string interest{};

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

size\_t size = m\_storage->GetInterestListSize();

Filter((Entity\*\*&)m\_storage->m\_interestList, size, interest, FieldMode::CONTENT);

m\_storage->SetInterestListSize(size);

std::cout << "\nUsers with similar interest:\n";

for (size\_t i{}; i < m\_storage->GetInterestListSize(); ++i)

{

for (size\_t j{}; j < m\_storage->GetUserListSize(); ++j)

{

if (m\_storage->m\_interestList[i]->GetOwnerId() ==

m\_storage->m\_userList[j]->GetId())

{

std::cout << m\_storage->m\_userList[j]->GetName() << ": " <<

m\_storage->m\_interestList[i]->GetContent() << "\n";

break;

}

}

}

m\_storage->LoadInterestList();

std::cout << "\nPress enter to return\n";

std::cin.ignore();

}

}

}

else if (userCommand == 2)

{

unsigned short reminderCommand = 0;

while (reminderCommand != 2)

{

system("cls");

Sort((Entity\*\*&)m\_storage->m\_reminderList, m\_storage->GetReminderListSize(),

OrderMode::ASCENDING, FieldMode::REMINDER\_TIME);

std::cout << "Today's reminders:\n";

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

for (size\_t i{}; i < m\_storage->GetReminderListSize(); ++i)

{

if (m\_storage->m\_reminderList[i]->GetOwnerId() == userId &&

m\_storage->m\_reminderList[i]->GetReminderTime().day == systemTime.wDay &&

m\_storage->m\_reminderList[i]->GetReminderTime().month == systemTime.wMonth &&

m\_storage->m\_reminderList[i]->GetReminderTime().year == systemTime.wYear)

{

std::cout << m\_storage->m\_reminderList[i]->GetReminderTime().day << "/" <<

m\_storage->m\_reminderList[i]->GetReminderTime().month << "/" <<

m\_storage->m\_reminderList[i]->GetReminderTime().year << ": " <<

m\_storage->m\_reminderList[i]->GetContent() << "\n";

}

}

std::cout << "\nAll reminders:\n";

std::vector<size\_t> indexes{};

for (size\_t i{}; i < m\_storage->GetReminderListSize(); ++i)

{

if (m\_storage->m\_reminderList[i]->GetOwnerId() == userId)

{

std::cout << indexes.size() << ") " <<

m\_storage->m\_reminderList[i]->GetReminderTime().day << "/" <<

m\_storage->m\_reminderList[i]->GetReminderTime().month << "/" <<

m\_storage->m\_reminderList[i]->GetReminderTime().year << ": " <<

m\_storage->m\_reminderList[i]->GetContent() << "\n";

indexes.push\_back(i);

}

}

std::cout << "\n0. Create new reminder\n";

std::cout << "1. Delete reminder\n";

std::cout << "2. Exit\n";

std::cout << ">>> ";

std::cin >> reminderCommand;

std::cin.ignore();

if (reminderCommand == 0)

{

Entity::Date newDate{};

std::cout << "Input day: ";

std::cin >> newDate.day;

std::cout << "Input month: ";

std::cin >> newDate.month;

std::cout << "Input year: ";

std::cin >> newDate.year;

std::cin.ignore();

std::cout << "Input text: ";

std::string text{};

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

size\_t size = m\_storage->GetReminderListSize();

Add<Reminder>(m\_storage->m\_reminderList, size,

new Reminder{ m\_storage->GetNextReminderId(),

{ systemTime.wDay, systemTime.wMonth, systemTime.wYear},

text, userId, newDate });

m\_storage->SetReminderListSize(size);

m\_storage->SaveReminderList();

}

else if (reminderCommand == 1)

{

std::cout << "Input reminder number: ";

size\_t reminderNumber{};

std::cin >> reminderNumber;

std::cin.ignore();

size\_t size = m\_storage->GetReminderListSize();

Remove((Entity\*\*&)m\_storage->m\_reminderList, size,

m\_storage->m\_reminderList[indexes[reminderNumber]]->GetId());

m\_storage->SetReminderListSize(size);

m\_storage->SaveReminderList();

}

m\_storage->LoadReminderList();

}

}

else if (userCommand == 3)

{

unsigned short themeCommand = 0;

while (themeCommand != 2)

{

system("cls");

for (size\_t i{}; i < m\_storage->GetThemeListSize(); ++i)

{

std::cout << i << ") " << m\_storage->m\_themeList[i]->GetContent() << "\n";

}

std::cout << "\n0. Open theme\n";

std::cout << "1. Create theme\n";

std::cout << "2. Exit\n";

std::cout << ">>> ";

std::cin >> themeCommand;

std::cin.ignore();

if (themeCommand == 0)

{

std::cout << "\nInput theme number: ";

size\_t themeNumber{};

std::cin >> themeNumber;

std::cin.ignore();

unsigned short themeDialogCommand = 0;

while (themeDialogCommand != 1)

{

system("cls");

std::cout << "\t" << m\_storage->m\_themeList[themeNumber]->GetContent() << "\n\n";

Sort((Entity\*\*&)m\_storage->m\_dialogueList, m\_storage->GetDialogueListSize(),

OrderMode::ASCENDING, FieldMode::DATE);

for (size\_t i{}; i < m\_storage->GetDialogueListSize(); ++i)

{

if (m\_storage->m\_dialogueList[i]->GetAdresseeId() ==

m\_storage->m\_themeList[themeNumber]->GetId())

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() == userId)

{

std::cout << "You: " << m\_storage->m\_dialogueList[i]->GetContent() << "\n";

}

else

{

std::string themeUserNickname{};

for (size\_t j{}; j < m\_storage->GetUserListSize(); ++j)

{

if (m\_storage->m\_dialogueList[i]->GetOwnerId() ==

m\_storage->m\_userList[j]->GetId())

{

themeUserNickname = m\_storage->m\_userList[j]->GetName();

break;

}

}

std::cout << themeUserNickname << ": " << m\_storage->m\_dialogueList[i]->GetContent() << "\n";

}

}

}

std::cout << "\n0. Write a message\n";

std::cout << "1. Exit\n";

std::cout << ">>> ";

std::cin >> themeDialogCommand;

std::cin.ignore();

if (themeDialogCommand == 0)

{

std::cout << "\nInput message: ";

std::string message{};

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

size\_t size = m\_storage->GetDialogueListSize();

Add<Dialogue>(m\_storage->m\_dialogueList, size,

new Dialogue{ m\_storage->GetNextDialogueId(),

{ systemTime.wDay, systemTime.wMonth, systemTime.wYear },

message, userId, m\_storage->m\_themeList[themeNumber]->GetId() });

m\_storage->SetDialogueListSize(size);

m\_storage->SaveDialogueList();

}

m\_storage->LoadDialogueList();

}

}

else if (themeCommand == 1)

{

std::cout << "Input theme: ";

std::string theme{};

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

size\_t size = m\_storage->GetThemeListSize();

Add<Theme>(m\_storage->m\_themeList, size,

new Theme{ m\_storage->GetNextThemeId(),

{ systemTime.wDay, systemTime.wMonth, systemTime.wYear},

theme, userId });

m\_storage->SetThemeListSize(size);

m\_storage->SaveThemeList();

}

}

}

}

}

else

{

std::cout << "Wrong login or password\n";

std::cin.ignore(1);

}

}

else if (command == 1)

{

std::string nickname{};

std::cout << "Nickname: ";

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

bool isTaken = false;

for (size\_t i{}; i < m\_storage->GetUserListSize(); i++)

{

if (nickname == m\_storage->m\_userList[i]->GetName())

{

isTaken = true;

std::cout << "This nickname is already taken\n";

std::cin.ignore(1);

}

}

if (isTaken) { continue; }

std::string login{};

std::cout << "Login: ";

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

for (size\_t i{}; i < m\_storage->GetUserListSize(); i++)

{

if (login == m\_storage->m\_userList[i]->GetLogin())

{

isTaken = true;

std::cout << "This login is already taken\n";

std::cin.ignore(1);

}

}

if (isTaken) { continue; }

std::string password{};

std::cout << "Password: ";

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

size\_t size = m\_storage->GetUserListSize();

Add<User>(m\_storage->m\_userList, size,

new User{ m\_storage->GetNextUserId(),

{ systemTime.wDay, systemTime.wMonth, systemTime.wYear },

nickname, login, password });

m\_storage->SetUserListSize(size);

m\_storage->SaveUserList();

std::cout << "Account has been created\n";

std::cin.ignore(1);

}

}

return 0;

}

}

**Main.cpp**

#include "Menu.h"

#include "UserListItem.h"

#include "DialogueListItem.h"

#include "InterestListItem.h"

#include "ReminderListItem.h"

#include "ThemeListItem.h"

#include <iostream>

#include "Storage.h"

#include "AuthorizationItem.h"

using namespace KMK;

int main()

{

Storage\* storage = new Storage("Dialogue database.dat", "Interst database.dat",

"Reminder database.dat", "Theme database.dat", "User database.dat", "Dialogue IDs.dat", "Interst IDs.dat",

"Reminder IDs.dat", "Theme and user IDs.dat", "Theme and user IDs.dat");

UserListItem users = UserListItem("User list", storage);

DialogueListItem dialogs = DialogueListItem("Dialogue list", storage);

InterestListItem interests = InterestListItem("Interest list", storage);

ReminderListItem reminders = ReminderListItem("Reminder list", storage);

ThemeListItem themes = ThemeListItem("Theme list", storage);

AuthorizstionItem authorization = AuthorizstionItem("Log in / sign in", storage);

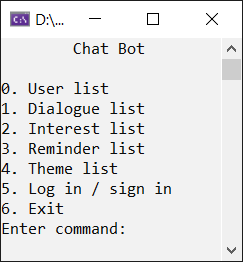
Menu menu = Menu("Chat Bot", new MenuItem\*[6] { &users, &dialogs, &interests, &reminders, &themes, &authorization }, 6);

std::cin >> menu;

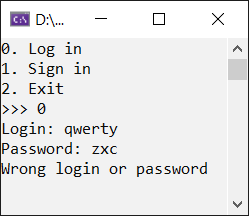
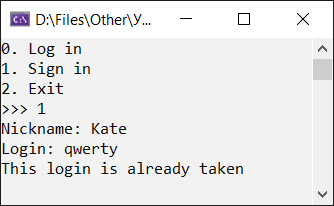
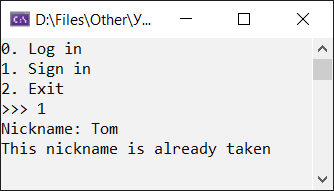
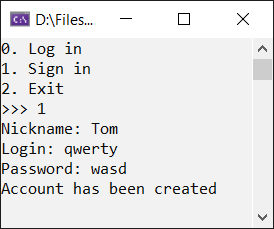
return 0;

}

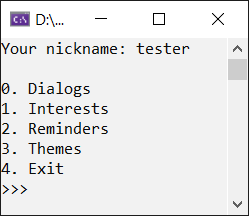
**Демонстрация:**



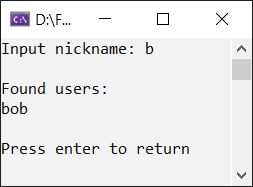
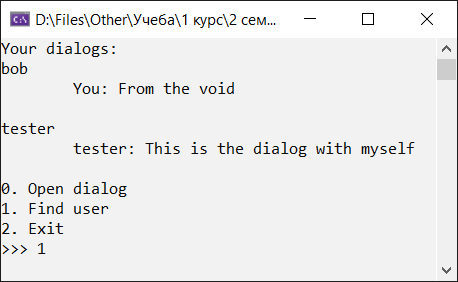
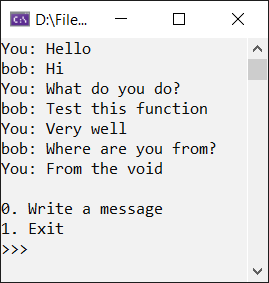
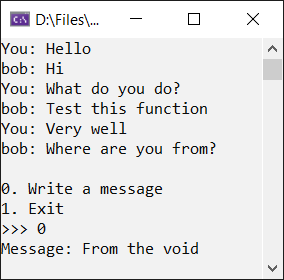
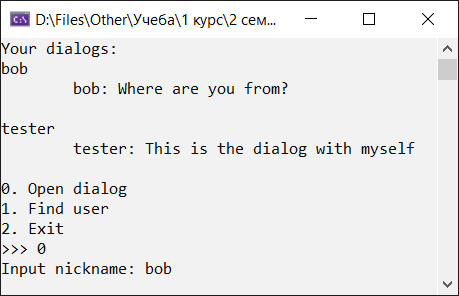
**Рисунок 2.** Главное меню



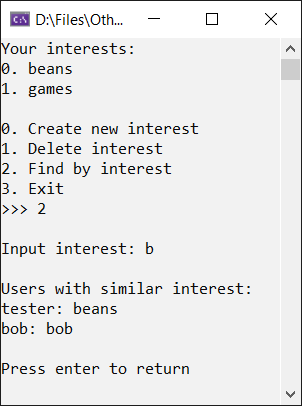
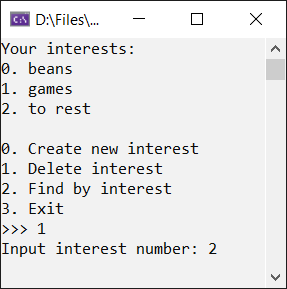
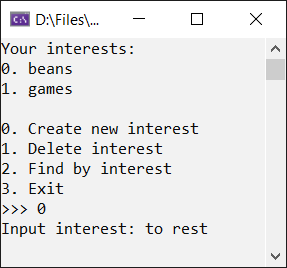
**Рисунок 3.** Регистрация и вход



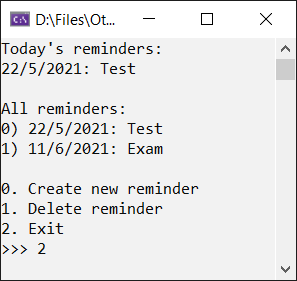
**Рисунок 4.** Пользовательское меню

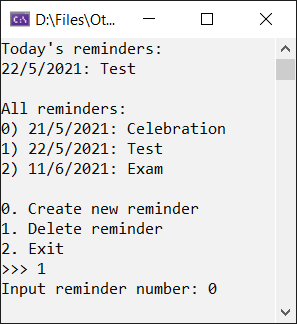
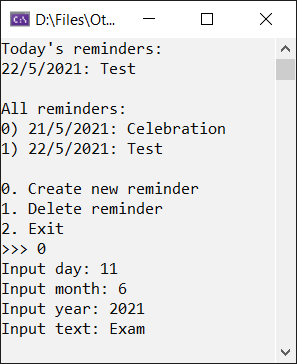


**Рисунок 5.** Меню диалогов

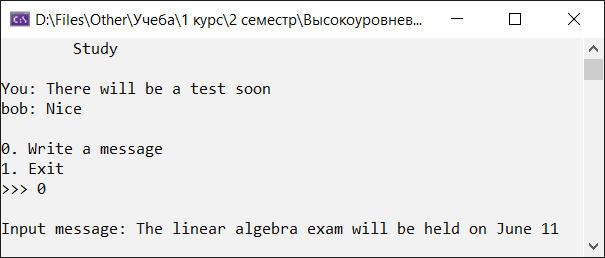
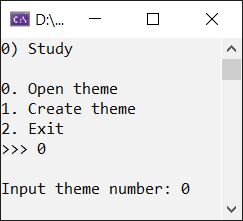


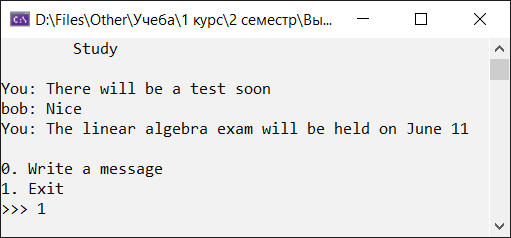
**Рисунок 6.** Меню интересов

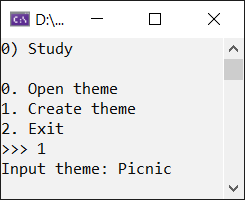
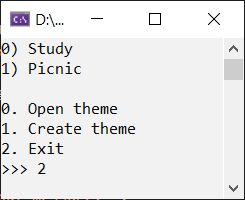
****



**Рисунок 7.** Меню напоминаний



****

** **

**Рисунок 8.** Меню тем

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки создания шаблонных функций, использования векторов.