|  |  |
| --- | --- |
| **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 №4**

**«Знакомство с контейнерами»**

**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 созданию и обработки классов – контейнеров данных.

**Задачи:**

1. Изучить понятие контейнера;
2. Научиться описывать простой контейнер класса;
3. Изучить написание элементов для контейнера;
4. Познакомиться с умными указателями;
5. Познакомиться с итераторами и научиться применять их;
6. Реализовать свою структуру контейнера.

**Вариант 8**

**Задание:**

**Общее задание:**

*Задача 1*

Создайте библиотеку (папку в корне вашего проекта, а в ней файлы), которая будет являться неким глобальным хранилищем данных вашей программы, назовите её Store. В ней создайте структуру данных State (напишите два файла State.h и State.cpp), в заголовочном файле этой структуры пропишите поля, в которых будут храниться наборы ваших сущностей (по одному полю для каждого типа набора) для реализации наборов можете использовать библиотеку STL, а конкретно тип: vector. Причём данные, которые будет хранить вектор должны быть указателями на объекты сущностей.

*Задача 2*

После того, как вы стали уверенно чувствовать себя при работе с классами, мы можем упростить нашу архитектуру программы. По факту наши сущности – это просто объекты с данными, которые не хранят в себе никаких методов по работе с ними. В таких случаях не применяется ООП, а используется обычная структура. Поэтому перепишите те сущности, в которых не используется ООП на обычные структуры, удалив геттеры/сеттеры, а также убрав модификаторы доступа. (если вы использовали для полей префикс \_m, в структурах его следует убрать). Стоит отметить, что если в сущностях используется наследование, то это уже ООП, такие сущности следует реализовывать только через классы. Также, если сеттер отвечает за обработку корректного значения ввода, то также оставляем сущность классом.

*Задача 3*

Перепишите все поля, которые использовали c – строки, на использование класса string (можно из STL библиотеки, а можно из методических указаний).

*Задача 4*

Создайте библиотеку экранов (папку в корне своего проекта под названием Screens, в которой будут лежать файлы). В этой библиотеки создайте под директории, в которых уже будут находиться сами экраны. Каждый экран будет представлять из себя класс, который наследуется от интерфейса InterfaceScreen (который будет лежать в корне папки с экранами). В интерфейсе будет две виртуальные чистые функции: int start(int) и void renderMain() const. Первая будет являться точкой входа в экран, а вторая будет отвечать за базовую его отрисовку в консоли. Каждый экран будет переопределять эти методы по своему усмотрению. Помимо переопределённых методов в экранах, будут и частные методы, которые относятся непосредственно к самому функционалу экрана, например: показ списка пользователей, сортировка, добавление, удаление и т д. Экран – это по сути логическая область вашей программы, которая будет отвечать за тот, или иной раздел функциональности. Разбейте вашу программу на логические блоки (по сути, она уже почти разбита в индивидуальном задании) – экраны и реализуйте в них частную функциональность. Пока, в качестве заглушки, включите в интерфейсный экран библиотеку Store и используйте объекты State, чтобы управлять данными.

**Основная задача:**

Вам будет предложено написать программу – «Автоматизированная система диалога (чат бот)». Которая будет включать следующий функционал:

* Ведение базы пользователей
  + Создание / удаление / редактирование записей
  + Сортировка / фильтрация
* Ведение базы диалогов, тем, интересов и напоминаний
* Возможность авторизации
* Создание файлов-отчётов и сохранения состояния

**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:**

**DialogueListItem.h**

#ifndef DIALOGUE\_LIST\_ITEM\_H

#define DIALOGUE\_LIST\_ITEM\_H

#include "AbstractMenuItem.h"

#include "Dialogue.h"

#include "Storage.h"

namespace KMK

{

class DialogueListItem : public MenuItem

{

public:

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

int Run();

private:

Storage\* m\_storage{};

};

}

#endif // !DIALOGUE\_LIST\_ITEM\_H

**DialogueListItem.cpp**

#include "DialogueListItem.h"

#include <iostream>

#include <fstream>

#include "Add.h"

#include <Windows.h>

#include "Remove.h"

#include "Edit.h"

#include "Sort.h"

#include "Filter.h"

#include <iomanip>

#include <string>

using namespace KMK;

DialogueListItem::DialogueListItem(std::string itemName, Storage\* storage) :

MenuItem(itemName)

{

m\_storage = storage;

}

int DialogueListItem::Run()

{

enum Command

{

RESET,

ADD,

REMOVE,

EDIT,

SORT,

FILTER,

ID,

EXIT

};

unsigned short command = 0;

while (command != EXIT)

{

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

unsigned short maximumMessageLength = 7;

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

{

if (m\_storage->m\_dialogueList[i]->GetContent().length() > maximumMessageLength)

{

maximumMessageLength = m\_storage->m\_dialogueList[i]->GetContent().length();

}

}

std::cout << std::setw((11 + 3 + 3 + 5 + maximumMessageLength + 1 + 11 + 11 + 6 + GetItemName().length()) / 2) << GetItemName() << "\n\n";

std::cout << std::setw(11) << "ID" << "|";

std::cout << std::setw(3) << "dd" << "|";

std::cout << std::setw(3) << "mm" << "|";

std::cout << std::setw(5) << "yyyy" << "|";

std::cout << std::setw(maximumMessageLength + 1) << "Message" << "|";

std::cout << std::setw(11) << "Sender ID" << "|";

std::cout << std::setw(11) << "Adressee ID";

std::cout << '\n';

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

{

std::cout << std::setw(11) << m\_storage->m\_dialogueList[i]->GetId() << "|";

std::cout << std::setw(3) << m\_storage->m\_dialogueList[i]->GetDate().day << "|";

std::cout << std::setw(3) << m\_storage->m\_dialogueList[i]->GetDate().month << "|";

std::cout << std::setw(5) << m\_storage->m\_dialogueList[i]->GetDate().year << "|";

std::cout << std::setw(maximumMessageLength + 1) << m\_storage->m\_dialogueList[i]->GetContent() << "|";

std::cout << std::setw(11) << m\_storage->m\_dialogueList[i]->GetOwnerId() << "|";

std::cout << std::setw(11) << m\_storage->m\_dialogueList[i]->GetAdresseeId();

std::cout << '\n';

}

std::cout << '\n';

std::cout << RESET << ". Reset list\n";

std::cout << ADD << ". Add new dialogue\n";

std::cout << REMOVE << ". Delete dialogue\n";

std::cout << EDIT << ". Edit dialogue\n";

std::cout << SORT << ". Sort list\n";

std::cout << FILTER << ". Filter list\n";

std::cout << ID << ". Choose ID\n";

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

std::cout << "Input command: ";

std::cin >> command;

std::cin.ignore();

std::cout << '\n';

if (command == RESET)

{

m\_storage->LoadDialogueList();

}

else if (command == ADD)

{

std::string message{};

std::cout << "Input message: ";

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

Id senderId;

std::cout << "Input sender ID: ";

std::cin >> senderId;

Id adresseeId;

std::cout << "Input adressee ID: ";

std::cin >> adresseeId;

std::cin.ignore();

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

Dialogue\* newDialogue = new Dialogue(m\_storage->GetNextDialogueId(), { systemTime.wDay, systemTime.wMonth, systemTime.wYear }, message, senderId, adresseeId);

Add((Entity\*\*&)m\_storage->m\_dialogueList, size, newDialogue);

m\_storage->SetDialogueListSize(size);

}

else if (command == REMOVE)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

Remove((Entity\*\*&)m\_storage->m\_dialogueList, size, id);

m\_storage->SetDialogueListSize(size);

}

else if (command == EDIT)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

std::cout << '\n';

std::cout << "Fields to edit\n";

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

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

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

std::cout << "3. Sender ID\n";

std::cout << "4. Adressee ID\n";

std::cout << "Choose field: ";

unsigned short fieldToChange;

std::cin >> fieldToChange;

std::cin.ignore();

std::cout << '\n';

if (fieldToChange == 0)

{

std::cout << "Input new ID: ";

Id\* newId = new Id{};

std::cin >> \*newId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_dialogueList, size, id, (void\*)newId, FieldMode::ID);

delete newId;

}

if (fieldToChange == 1)

{

std::cout << "Input new date\n";

std::cout << "Day:";

unsigned short day;

std::cin >> day;

std::cout << "Month:";

unsigned short month;

std::cin >> month;

std::cout << "Year:";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date\* newDate = new Entity::Date{ day, month, year };

Edit((Entity\*\*&)m\_storage->m\_dialogueList, size, id, (void\*)newDate, FieldMode::DATE);

delete newDate;

}

if (fieldToChange == 2)

{

std::cout << "Input new message: ";

std::string\* message = new std::string{};

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

Edit((Entity\*\*&)m\_storage->m\_dialogueList, size, id, (void\*)message, FieldMode::CONTENT);

delete message;

}

if (fieldToChange == 3)

{

std::cout << "Input new sender ID: ";

Id\* newSenderId = new Id{};

std::cin >> \*newSenderId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_dialogueList, size, id, (void\*)newSenderId, FieldMode::OWNER\_ID);

delete newSenderId;

}

if (fieldToChange == 4)

{

std::cout << "Input new adressee ID: ";

Id\* newAdresseeId = new Id{};

std::cin >> \*newAdresseeId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_dialogueList, size, id, (void\*)newAdresseeId, FieldMode::OWNER\_ID);

delete newAdresseeId;

}

}

else if (command == SORT)

{

std::cout << "Orders for sort\n";

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

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

std::cout << "Choose order: ";

unsigned short order;

std::cin >> order;

std::cout << '\n';

std::cout << "Fields for sort\n";

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

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

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

std::cout << "3. Sender ID\n";

std::cout << "4. Adressee ID\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

FieldMode sortMode = (FieldMode)-1;

switch (field)

{

case 0:

sortMode = FieldMode::ID;

break;

case 1:

sortMode = FieldMode::DATE;

break;

case 2:

sortMode = FieldMode::CONTENT;

break;

case 3:

sortMode = FieldMode::OWNER\_ID;

break;

case 4:

sortMode = FieldMode::ADRESSEE\_ID;

break;

}

Sort((Entity\*\*&)m\_storage->m\_dialogueList, size, (OrderMode)order, sortMode);

}

else if (command == FILTER)

{

std::cout << "Fields for filter\n";

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

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

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

std::cout << "3. Sender ID\n";

std::cout << "4. Adressee ID\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

std::cout << '\n';

if (field == 0)

{

std::cout << "Input part of ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_dialogueList, size, id, FieldMode::ID);

}

if (field == 1)

{

std::cout << "Input date (if you don't want to filter by the field, input 0)\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date date{ day, month, year };

Filter((Entity\*\*&)m\_storage->m\_dialogueList, size, date, FieldMode::DATE);

}

if (field == 2)

{

std::cout << "Input part of message: ";

std::string message{};

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

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

}

if (field == 3)

{

std::cout << "Input part of sender ID: ";

Id senderId;

std::cin >> senderId;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_dialogueList, size, senderId, FieldMode::OWNER\_ID);

}

if (field == 4)

{

std::cout << "Input part of adressee ID: ";

Id adresseeId;

std::cin >> adresseeId;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_dialogueList, size, adresseeId, FieldMode::ADRESSEE\_ID);

}

m\_storage->SetDialogueListSize(size);

}

if (command == ADD || command == REMOVE || command == EDIT)

{

m\_storage->SaveDialogueList();

}

system("cls");

}

return 0;

}

**InterestListItem.h**

#ifndef INTEREST\_LIST\_ITEM\_H

#define INTEREST\_LIST\_ITEM\_H

#include "AbstractMenuItem.h"

#include "Interest.h"

#include "Storage.h"

namespace KMK

{

class InterestListItem : public MenuItem

{

public:

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

int Run();

private:

Storage\* m\_storage{};

};

}

#endif // !INTEREST\_LIST\_ITEM\_H

**InterestListItem.cpp**

#include "InterestListItem.h"

#include <iostream>

#include <fstream>

#include "Add.h"

#include <Windows.h>

#include "Remove.h"

#include "Edit.h"

#include "Sort.h"

#include "Filter.h"

#include <iomanip>

#include <string>

using namespace KMK;

InterestListItem::InterestListItem(std::string itemName, Storage\* storage) :

MenuItem(itemName)

{

m\_storage = storage;

}

int InterestListItem::Run()

{

enum Command

{

RESET,

ADD,

REMOVE,

EDIT,

SORT,

FILTER,

ID,

EXIT

};

unsigned short command = 0;

while (command != EXIT)

{

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

unsigned short maximumInterestLength = 8;

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

{

if (m\_storage->m\_interestList[i]->GetContent().length() > maximumInterestLength)

{

maximumInterestLength = m\_storage->m\_interestList[i]->GetContent().length();

}

}

std::cout << std::setw((11 + 3 + 3 + 5 + maximumInterestLength + 1 + 11 + 5 + GetItemName().length()) / 2) << GetItemName() << "\n\n";

std::cout << std::setw(11) << "ID" << "|";

std::cout << std::setw(3) << "dd" << "|";

std::cout << std::setw(3) << "mm" << "|";

std::cout << std::setw(5) << "yyyy" << "|";

std::cout << std::setw(maximumInterestLength + 1) << "Interest" << "|";

std::cout << std::setw(11) << "Owner ID";

std::cout << '\n';

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

{

std::cout << std::setw(11) << m\_storage->m\_interestList[i]->GetId() << "|";

std::cout << std::setw(3) << m\_storage->m\_interestList[i]->GetDate().day << "|";

std::cout << std::setw(3) << m\_storage->m\_interestList[i]->GetDate().month << "|";

std::cout << std::setw(5) << m\_storage->m\_interestList[i]->GetDate().year << "|";

std::cout << std::setw(maximumInterestLength + 1) << m\_storage->m\_interestList[i]->GetContent() << "|";

std::cout << std::setw(11) << m\_storage->m\_interestList[i]->GetOwnerId();

std::cout << '\n';

}

std::cout << '\n';

std::cout << RESET << ". Reset list\n";

std::cout << ADD << ". Add new interest\n";

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

std::cout << EDIT << ". Edit interest\n";

std::cout << SORT << ". Sort list\n";

std::cout << FILTER << ". Filter list\n";

std::cout << ID << ". Choose ID\n";

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

std::cout << "Input command: ";

std::cin >> command;

std::cin.ignore();

std::cout << '\n';

if (command == RESET)

{

m\_storage->LoadInterestList();

}

else if (command == ADD)

{

std::string interest{};

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

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

Id ownerId;

std::cout << "Input owner ID: ";

std::cin >> ownerId;

std::cin.ignore();

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

Interest\* newInterest = new Interest(m\_storage->GetNextInterestId(), { systemTime.wDay, systemTime.wMonth, systemTime.wYear }, interest, ownerId);

Add((Entity\*\*&)m\_storage->m\_interestList, size, newInterest);

m\_storage->SetInterestListSize(size);

}

else if (command == REMOVE)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

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

m\_storage->SetInterestListSize(size);

}

else if (command == EDIT)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

std::cout << '\n';

std::cout << "Fields to edit\n";

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

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

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

std::cout << "3. Owner ID\n";

std::cout << "Choose field: ";

unsigned short fieldToChange;

std::cin >> fieldToChange;

std::cin.ignore();

std::cout << '\n';

if (fieldToChange == 0)

{

std::cout << "Input new ID: ";

Id\* newId = new Id{};

std::cin >> \*newId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_interestList, size, id, (void\*)newId, FieldMode::ID);

delete newId;

}

if (fieldToChange == 1)

{

std::cout << "Input new date\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date\* newDate = new Entity::Date{ day, month, year };

Edit((Entity\*\*&)m\_storage->m\_interestList, size, id, (void\*)newDate, FieldMode::DATE);

delete newDate;

}

if (fieldToChange == 2)

{

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

std::string\* interest = new std::string{};

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

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

delete interest;

}

if (fieldToChange == 3)

{

std::cout << "Input new owner ID: ";

Id\* newOwnerId = new Id{};

std::cin >> \*newOwnerId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_interestList, size, id, (void\*)newOwnerId, FieldMode::OWNER\_ID);

delete newOwnerId;

}

}

else if (command == SORT)

{

std::cout << "Orders for sort\n";

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

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

std::cout << "Choose order: ";

unsigned short order;

std::cin >> order;

std::cout << '\n';

std::cout << "Fields for sort\n";

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

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

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

std::cout << "3. Owner ID\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

FieldMode sortMode = (FieldMode)-1;

switch (field)

{

case 0:

sortMode = FieldMode::ID;

break;

case 1:

sortMode = FieldMode::DATE;

break;

case 2:

sortMode = FieldMode::CONTENT;

break;

case 3:

sortMode = FieldMode::OWNER\_ID;

break;

}

Sort((Entity\*\*&)m\_storage->m\_interestList, size, (OrderMode)order, sortMode);

}

else if (command == FILTER)

{

std::cout << "Fields for filter\n";

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

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

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

std::cout << "3. Owner ID\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

std::cout << '\n';

if (field == 0)

{

std::cout << "Input part of ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

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

}

if (field == 1)

{

std::cout << "Input date (if you don't want to filter by the field, input 0)\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date date{ day, month, year };

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

}

if (field == 2)

{

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

std::string interest{};

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

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

}

if (field == 3)

{

std::cout << "Input part of owner ID: ";

Id ownerId;

std::cin >> ownerId;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_interestList, size, ownerId, FieldMode::OWNER\_ID);

}

m\_storage->SetInterestListSize(size);

}

if (command == ADD || command == REMOVE || command == EDIT)

{

m\_storage->SaveInterestList();

}

system("cls");

}

return 0;

}

**ReminderListItem.h**

#ifndef REMINDER\_LIST\_ITEM\_H

#define REMINDER\_LIST\_ITEM\_H

#include "AbstractMenuItem.h"

#include "Reminder.h"

#include "Storage.h"

namespace KMK

{

class ReminderListItem : public MenuItem

{

public:

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

int Run();

private:

Storage\* m\_storage{};

};

}

#endif // !REMINDER\_LIST\_ITEM\_H

**ReminderListItem.cpp**

#include "ReminderListItem.h"

#include <iostream>

#include <fstream>

#include "Add.h"

#include <Windows.h>

#include "Remove.h"

#include "Edit.h"

#include "Sort.h"

#include "Filter.h"

#include <iomanip>

using namespace KMK;

ReminderListItem::ReminderListItem(std::string itemName, Storage\* storage) :

MenuItem(itemName)

{

m\_storage = storage;

}

int ReminderListItem::Run()

{

enum Command

{

RESET,

ADD,

REMOVE,

EDIT,

SORT,

FILTER,

ID,

EXIT

};

unsigned short command = 0;

while (command != EXIT)

{

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

unsigned short maximumReminderLength = 8;

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

{

if (m\_storage->m\_reminderList[i]->GetContent().length() > maximumReminderLength)

{

maximumReminderLength = m\_storage->m\_reminderList[i]->GetContent().length();

}

}

std::cout << std::setw((11 + 3 + 3 + 5 + 11 + maximumReminderLength + 1 + 3 + 3 + 5 + 8 + GetItemName().length()) / 2) << GetItemName() << "\n\n";

std::cout << std::setw(11) << "ID" << "|";

std::cout << std::setw(3) << "dd" << "|";

std::cout << std::setw(3) << "mm" << "|";

std::cout << std::setw(5) << "yyyy" << "|";

std::cout << std::setw(11) << "Owner ID" << "|";

std::cout << std::setw(maximumReminderLength + 1) << "Reminder" << ":";

std::cout << std::setw(3) << "dd" << "|";

std::cout << std::setw(3) << "mm" << "|";

std::cout << std::setw(5) << "yyyy";

std::cout << '\n';

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

{

std::cout << std::setw(11) << m\_storage->m\_reminderList[i]->GetId() << "|";

std::cout << std::setw(3) << m\_storage->m\_reminderList[i]->GetDate().day << "|";

std::cout << std::setw(3) << m\_storage->m\_reminderList[i]->GetDate().month << "|";

std::cout << std::setw(5) << m\_storage->m\_reminderList[i]->GetDate().year << "|";

std::cout << std::setw(11) << m\_storage->m\_reminderList[i]->GetOwnerId() << "|";

std::cout << std::setw(maximumReminderLength + 1) << m\_storage->m\_reminderList[i]->GetContent() << ":";

std::cout << std::setw(3) << m\_storage->m\_reminderList[i]->GetReminderTime().day << "|";

std::cout << std::setw(3) << m\_storage->m\_reminderList[i]->GetReminderTime().month << "|";

std::cout << std::setw(5) << m\_storage->m\_reminderList[i]->GetReminderTime().year;

std::cout << '\n';

}

std::cout << '\n';

std::cout << RESET << ". Reset list\n";

std::cout << ADD << ". Add new reminder\n";

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

std::cout << EDIT << ". Edit reminder\n";

std::cout << SORT << ". Sort list\n";

std::cout << FILTER << ". Filter list\n";

std::cout << ID << ". Choose ID\n";

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

std::cout << "Input command: ";

std::cin >> command;

std::cin.ignore();

std::cout << '\n';

if (command == RESET)

{

m\_storage->LoadReminderList();

}

else if (command == ADD)

{

Id ownerId;

std::cout << "Input owner ID: ";

std::cin >> ownerId;

std::cin.ignore();

std::string reminder{};

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

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

unsigned short day;

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

std::cin >> day;

unsigned short month;

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

std::cin >> month;

unsigned short year;

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

std::cin >> year;

std::cin.ignore();

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

Reminder\* newReminder = new Reminder(m\_storage->GetNextReminderId(), { systemTime.wDay, systemTime.wMonth, systemTime.wYear }, reminder, ownerId, {day, month, year});

Add((Entity\*\*&)m\_storage->m\_reminderList, size, newReminder);

m\_storage->SetReminderListSize(size);

}

else if (command == REMOVE)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

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

m\_storage->SetReminderListSize(size);

}

else if (command == EDIT)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

std::cout << '\n';

std::cout << "Fields to edit\n";

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

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

std::cout << "2. Owner ID\n";

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

std::cout << "4. Reminder time\n";

std::cout << "Choose field: ";

unsigned short fieldToChange;

std::cin >> fieldToChange;

std::cin.ignore();

std::cout << '\n';

if (fieldToChange == 0)

{

std::cout << "Input new ID: ";

Id\* newId = new Id{};

std::cin >> \*newId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_reminderList, size, id, (void\*)newId, FieldMode::ID);

delete newId;

}

if (fieldToChange == 1)

{

std::cout << "Input new date\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date\* newDate = new Entity::Date{ day, month, year };

Edit((Entity\*\*&)m\_storage->m\_reminderList, size, id, (void\*)newDate, FieldMode::DATE);

delete newDate;

}

if (fieldToChange == 2)

{

std::cout << "Input new owner ID: ";

Id\* newOwnerId = new Id{};

std::cin >> \*newOwnerId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_reminderList, size, id, (void\*)newOwnerId, FieldMode::OWNER\_ID);

delete newOwnerId;

}

if (fieldToChange == 3)

{

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

std::string\* reminder = new std::string{};

std::getline(std::cin, \*reminder);

Edit((Entity\*\*&)m\_storage->m\_reminderList, size, id, (void\*)reminder, FieldMode::CONTENT);

delete reminder;

}

if (fieldToChange == 4)

{

std::cout << "Input new reminder time\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date\* newReminderDate = new Entity::Date{ day, month, year };

Edit((Entity\*\*&)m\_storage->m\_reminderList, size, id, (void\*)newReminderDate, FieldMode::REMINDER\_TIME);

delete newReminderDate;

}

}

else if (command == SORT)

{

std::cout << "Orders for sort\n";

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

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

std::cout << "Choose order: ";

unsigned short order;

std::cin >> order;

std::cout << '\n';

std::cout << "Fields for sort\n";

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

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

std::cout << "2. Owner ID\n";

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

std::cout << "4. Reminder time\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

FieldMode sortMode = (FieldMode)-1;

switch (field)

{

case 0:

sortMode = FieldMode::ID;

break;

case 1:

sortMode = FieldMode::DATE;

break;

case 2:

sortMode = FieldMode::OWNER\_ID;

break;

case 3:

sortMode = FieldMode::CONTENT;

break;

case 4:

sortMode = FieldMode::REMINDER\_TIME;

break;

}

Sort((Entity\*\*&)m\_storage->m\_reminderList, size, (OrderMode)order, sortMode);

}

else if (command == FILTER)

{

std::cout << "Fields for filter\n";

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

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

std::cout << "2. Owner ID\n";

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

std::cout << "4. Reminder time\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

std::cout << '\n';

if (field == 0)

{

std::cout << "Input part of ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_reminderList, size, id, FieldMode::ID);

}

if (field == 1)

{

std::cout << "Input date (if you don't want to filter by the field, input 0)\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date date{ day, month, year };

Filter((Entity\*\*&)m\_storage->m\_reminderList, size, date, FieldMode::DATE);

}

if (field == 2)

{

std::cout << "Input part of owner ID: ";

Id ownerId;

std::cin >> ownerId;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_reminderList, size, ownerId, FieldMode::OWNER\_ID);

}

if (field == 3)

{

std::cout << "Input part of reminder: ";

std::string reminder{};

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

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

}

if (field == 4)

{

std::cout << "Input reminder time (if you don't want to filter by the field, input 0)\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date date{ day, month, year };

Filter((Entity\*\*&)m\_storage->m\_reminderList, size, date, FieldMode::REMINDER\_TIME);

}

m\_storage->SetReminderListSize(size);

}

if (command == ADD || command == REMOVE || command == EDIT)

{

m\_storage->SaveReminderList();

}

system("cls");

}

return 0;

}

**ThemeListItem.h**

#ifndef THEME\_LIST\_ITEM\_H

#define THEME\_LIST\_ITEM\_H

#include "AbstractMenuItem.h"

#include "Theme.h"

#include "Storage.h"

namespace KMK

{

class ThemeListItem : public MenuItem

{

public:

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

int Run();

private:

Storage\* m\_storage{};

};

}

#endif // !THEME\_LIST\_ITEM\_H

**ThemeListItem.cpp**

#include "ThemeListItem.h"

#include <iostream>

#include <fstream>

#include "Add.h"

#include <Windows.h>

#include "Remove.h"

#include "Edit.h"

#include "Sort.h"

#include "Filter.h"

#include <iomanip>

using namespace KMK;

ThemeListItem::ThemeListItem(std::string itemName, Storage\* storage) :

MenuItem(itemName)

{

m\_storage = storage;

}

int ThemeListItem::Run()

{

enum Command

{

RESET,

ADD,

REMOVE,

EDIT,

SORT,

FILTER,

ID,

EXIT

};

unsigned short command = 0;

while (command != EXIT)

{

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

unsigned short maximumThemeLength = 5;

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

{

if (m\_storage->m\_themeList[i]->GetContent().length() > maximumThemeLength)

{

maximumThemeLength = m\_storage->m\_themeList[i]->GetContent().length();

}

}

std::cout << std::setw((11 + 3 + 3 + 5 + maximumThemeLength + 1 + 11 + 5 + GetItemName().length()) / 2) << GetItemName() << "\n\n";

std::cout << std::setw(11) << "ID" << "|";

std::cout << std::setw(3) << "dd" << "|";

std::cout << std::setw(3) << "mm" << "|";

std::cout << std::setw(5) << "yyyy" << "|";

std::cout << std::setw(maximumThemeLength + 1) << "Theme" << "|";

std::cout << std::setw(11) << "Owner ID";

std::cout << '\n';

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

{

std::cout << std::setw(11) << m\_storage->m\_themeList[i]->GetId() << "|";

std::cout << std::setw(3) << m\_storage->m\_themeList[i]->GetDate().day << "|";

std::cout << std::setw(3) << m\_storage->m\_themeList[i]->GetDate().month << "|";

std::cout << std::setw(5) << m\_storage->m\_themeList[i]->GetDate().year << "|";

std::cout << std::setw(maximumThemeLength + 1) << m\_storage->m\_themeList[i]->GetContent() << "|";

std::cout << std::setw(11) << m\_storage->m\_themeList[i]->GetOwnerId();

std::cout << '\n';

}

std::cout << '\n';

std::cout << RESET << ". Reset list\n";

std::cout << ADD << ". Add new theme\n";

std::cout << REMOVE << ". Delete theme\n";

std::cout << EDIT << ". Edit theme\n";

std::cout << SORT << ". Sort list\n";

std::cout << FILTER << ". Filter list\n";

std::cout << ID << ". Choose ID\n";

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

std::cout << "Input command: ";

std::cin >> command;

std::cin.ignore();

std::cout << '\n';

if (command == RESET)

{

m\_storage->LoadThemeList();

}

else if (command == ADD)

{

std::string theme{};

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

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

Id ownerId;

std::cout << "Input owner ID: ";

std::cin >> ownerId;

std::cin.ignore();

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

Theme\* newTheme = new Theme(m\_storage->GetNextThemeId(), { systemTime.wDay, systemTime.wMonth, systemTime.wYear }, theme, ownerId);

Add((Entity\*\*&)m\_storage->m\_themeList, size, newTheme);

m\_storage->SetThemeListSize(size);

}

else if (command == REMOVE)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

Remove((Entity\*\*&)m\_storage->m\_themeList, size, id);

m\_storage->SetThemeListSize(size);

}

else if (command == EDIT)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

std::cout << '\n';

std::cout << "Fields to edit\n";

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

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

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

std::cout << "3. Owner ID\n";

std::cout << "Choose field: ";

unsigned short fieldToChange;

std::cin >> fieldToChange;

std::cin.ignore();

std::cout << '\n';

if (fieldToChange == 0)

{

std::cout << "Input new ID: ";

Id\* newId = new Id{};

std::cin >> \*newId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_themeList, size, id, (void\*)newId, FieldMode::ID);

delete newId;

}

if (fieldToChange == 1)

{

std::cout << "Input new date\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date\* newDate = new Entity::Date{ day, month, year };

Edit((Entity\*\*&)m\_storage->m\_themeList, size, id, (void\*)newDate, FieldMode::DATE);

delete newDate;

}

if (fieldToChange == 2)

{

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

std::string\* theme = new std::string{};

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

Edit((Entity\*\*&)m\_storage->m\_themeList, size, id, (void\*)theme, FieldMode::CONTENT);

delete theme;

}

if (fieldToChange == 3)

{

std::cout << "Input new owner ID: ";

Id\* newOwnerId = new Id{};

std::cin >> \*newOwnerId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_themeList, size, id, (void\*)newOwnerId, FieldMode::OWNER\_ID);

delete newOwnerId;

}

}

else if (command == SORT)

{

std::cout << "Orders for sort\n";

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

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

std::cout << "Choose order: ";

unsigned short order;

std::cin >> order;

std::cout << '\n';

std::cout << "Fields for sort\n";

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

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

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

std::cout << "3. Owner ID\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

FieldMode sortMode = (FieldMode)-1;

switch (field)

{

case 0:

sortMode = FieldMode::ID;

break;

case 1:

sortMode = FieldMode::DATE;

break;

case 2:

sortMode = FieldMode::CONTENT;

break;

case 3:

sortMode = FieldMode::OWNER\_ID;

break;

}

Sort((Entity\*\*&)m\_storage->m\_themeList, size, (OrderMode)order, sortMode);

}

else if (command == FILTER)

{

std::cout << "Fields for filter\n";

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

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

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

std::cout << "3. Owner ID\n";

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

std::cout << '\n';

if (field == 0)

{

std::cout << "Input part of ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_themeList, size, id, FieldMode::ID);

}

if (field == 1)

{

std::cout << "Input date (if you don't want to filter by the field, input 0)\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date date { day, month, year };

Filter((Entity\*\*&)m\_storage->m\_themeList, size, date, FieldMode::DATE);

}

if (field == 2)

{

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

std::string theme{};

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

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

}

if (field == 3)

{

std::cout << "Input part of owner ID: ";

Id ownerId;

std::cin >> ownerId;

std::cin.ignore();

Filter((Entity\*\*&)m\_storage->m\_themeList, size, ownerId, FieldMode::OWNER\_ID);

}

m\_storage->SetThemeListSize(size);

}

if (command == ADD || command == REMOVE || command == EDIT)

{

m\_storage->SaveThemeList();

}

system("cls");

}

return 0;

}

**UserListItem.h**

#ifndef USER\_LIST\_ITEM\_H

#define USER\_LIST\_ITEM\_H

#include "AbstractMenuItem.h"

#include "User.h"

#include "Storage.h"

namespace KMK

{

class UserListItem : public MenuItem

{

public:

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

int Run();

private:

Storage\* m\_storage{};

};

}

#endif // !USER\_LIST\_ITEM\_H

**UserListItem.cpp**

#include "UserListItem.h"

#include <iostream>

#include <fstream>

#include "Add.h"

#include <Windows.h>

#include "Remove.h"

#include "Edit.h"

#include "Sort.h"

#include "Filter.h"

#include <iomanip>

using namespace KMK;

UserListItem::UserListItem(std::string itemName, Storage\* storage) :

MenuItem(itemName)

{

m\_storage = storage;

}

int UserListItem::Run()

{

enum Command

{

RESET,

ADD,

REMOVE,

EDIT,

SORT,

FILTER,

ID,

EXIT

};

unsigned short command = 0;

while (command != EXIT)

{

unsigned short maximumNameLength = 4;

unsigned short maximumLoginLength = 5;

unsigned short maximumPasswordLength = 8;

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

{

if (m\_storage->m\_userList[i]->GetName().length() > maximumNameLength)

{

maximumNameLength = m\_storage->m\_userList[i]->GetName().length();

}

if (m\_storage->m\_userList[i]->GetLogin().length() > maximumLoginLength)

{

maximumLoginLength = m\_storage->m\_userList[i]->GetLogin().length();

}

if (m\_storage->m\_userList[i]->GetPassword().length() > maximumPasswordLength)

{

maximumPasswordLength = m\_storage->m\_userList[i]->GetPassword().length();

}

}

std::cout << std::setw((11 + 3 + 3 + 5 + maximumNameLength + 1 + maximumLoginLength + 1 + maximumPasswordLength + 1 + 6 + GetItemName().length()) / 2) << GetItemName() << "\n\n";

std::cout << std::setw(11) << "ID" << "|";

std::cout << std::setw(3) << "dd" << "|";

std::cout << std::setw(3) << "mm" << "|";

std::cout << std::setw(5) << "yyyy" << "|";

std::cout << std::setw(maximumNameLength + 1) << "Name" << "|";

std::cout << std::setw(maximumLoginLength + 1) << "Login" << "|";

std::cout << std::setw(maximumPasswordLength + 1) << "Password";

std::cout << '\n';

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

{

std::cout << std::setw(11) << m\_storage->m\_userList[i]->GetId() << "|";

std::cout << std::setw(3) << m\_storage->m\_userList[i]->GetDate().day << "|";

std::cout << std::setw(3) << m\_storage->m\_userList[i]->GetDate().month << "|";

std::cout << std::setw(5) << m\_storage->m\_userList[i]->GetDate().year << "|";

std::cout << std::setw(maximumNameLength + 1) << m\_storage->m\_userList[i]->GetName() << "|";

std::cout << std::setw(maximumLoginLength + 1) << m\_storage->m\_userList[i]->GetLogin() << "|";

std::cout << std::setw(maximumPasswordLength + 1) << m\_storage->m\_userList[i]->GetPassword();

std::cout << '\n';

}

std::cout << '\n';

std::cout << RESET << ". Reset list\n";

std::cout << ADD << ". Add new user\n";

std::cout << REMOVE << ". Delete user\n";

std::cout << EDIT << ". Edit user\n";

std::cout << SORT << ". Sort list\n";

std::cout << FILTER << ". Filter list\n";

std::cout << ID << ". Choose ID\n";

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

std::cout << "Input command: ";

std::cin >> command;

std::cin.ignore();

std::cout << '\n';

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

if (command == RESET)

{

m\_storage->LoadUserList();

}

else if (command == ADD)

{

std::string name{};

std::cout << "Input name: ";

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

std::string login{};

std::cout << "Input login: ";

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

std::string password{};

std::cout << "Input password: ";

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

SYSTEMTIME systemTime;

GetLocalTime(&systemTime);

User\* newUser = new User(m\_storage->GetNextUserId(), { systemTime.wDay, systemTime.wMonth, systemTime.wYear }, name, login, password);

Add((Entity\*\*&)m\_storage->m\_userList, size, newUser);

m\_storage->SetUserListSize(size);

}

else if (command == REMOVE)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

Remove((Entity\*\*&)m\_storage->m\_userList, size, id);

m\_storage->SetUserListSize(size);

}

else if (command == EDIT)

{

std::cout << "Input ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

std::cout << '\n';

std::cout << "Fields to edit\n";

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

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

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

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

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

std::cout << "Choose field: ";

unsigned short fieldToChange;

std::cin >> fieldToChange;

std::cin.ignore();

std::cout << '\n';

if (fieldToChange == 0)

{

std::cout << "Input new ID: ";

Id\* newId = new Id{};

std::cin >> \*newId;

std::cin.ignore();

Edit((Entity\*\*&)m\_storage->m\_userList, size, id, (void\*)newId, FieldMode::ID);

delete newId;

}

if (fieldToChange == 1)

{

std::cout << "Input new date\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date\* newDate = new Entity::Date{ day, month, year };

Edit((Entity\*\*&)m\_storage->m\_userList, size, id, (void\*)newDate, FieldMode::DATE);

delete newDate;

}

if (fieldToChange == 2)

{

std::cout << "Input new name: ";

std::string\* name = new std::string{};

std::getline(std::cin, \*name);

Edit((Entity\*\*&)m\_storage->m\_userList, size, id, (void\*)name, FieldMode::NAME);

delete name;

}

if (fieldToChange == 3)

{

std::cout << "Input new login: ";

std::string\* login = new std::string{};

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

Edit((Entity\*\*&)m\_storage->m\_userList, size, id, (void\*)login, FieldMode::LOGIN);

delete login;

}

if (fieldToChange == 4)

{

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

std::string\* password = new std::string{};

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

Edit((Entity\*\*&)m\_storage->m\_userList, size, id, (void\*)password, FieldMode::PASSWORD);

delete password;

}

}

else if (command == SORT)

{

std::cout << "Orders for sort\n";

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

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

std::cout << "Choose order: ";

unsigned short order;

std::cin >> order;

std::cout << '\n';

std::cout << "Fields for sort\n";

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

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

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

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

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

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

FieldMode sortMode = (FieldMode)-1;

switch (field)

{

case 0:

sortMode = FieldMode::ID;

break;

case 1:

sortMode = FieldMode::DATE;

break;

case 2:

sortMode = FieldMode::NAME;

break;

case 3:

sortMode = FieldMode::LOGIN;

break;

case 4:

sortMode = FieldMode::PASSWORD;

break;

}

Sort((Entity\*\*&)m\_storage->m\_userList, size, (OrderMode)order, sortMode);

}

else if (command == FILTER)

{

std::cout << "Fields for filter\n";

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

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

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

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

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

std::cout << "Choose field: ";

unsigned short field;

std::cin >> field;

std::cin.ignore();

std::cout << '\n';

if (field == 0)

{

std::cout << "Input part of ID: ";

Id id;

std::cin >> id;

std::cin.ignore();

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

}

if (field == 1)

{

std::cout << "Input date (if you don't want to filter by the field, input 0)\n";

std::cout << "Day: ";

unsigned short day;

std::cin >> day;

std::cout << "Month: ";

unsigned short month;

std::cin >> month;

std::cout << "Year: ";

unsigned short year;

std::cin >> year;

std::cin.ignore();

Entity::Date date{ day, month, year };

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

}

if (field == 2)

{

std::cout << "Input part of name: ";

std::string name{};

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

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

}

if (field == 3)

{

std::cout << "Input part of login: ";

std::string login{};

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

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

}

if (field == 4)

{

std::cout << "Input part of password: ";

std::string password{};

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

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

}

m\_storage->SetUserListSize(size);

}

if (command == ADD || command == REMOVE || command == EDIT)

{

m\_storage->SaveUserList();

}

system("cls");

}

return 0;

}

**Storage.h**

#ifndef STORAGE\_H

#define STORAGE\_H

#include "Dialogue.h"

#include "Interest.h"

#include "Reminder.h"

#include "Theme.h"

#include "User.h"

#include "IdCounter.h"

namespace KMK

{

class Storage

{

public:

Storage();

Storage(std::string dialogueListFilePath, std::string interestListFilePath,

std::string reminderListFilePath, std::string themeListFilePath, std::string userListFilePath,

std::string dialogueIdCounterFilePath, std::string interestIdCounterFilePath,

std::string reminderIdCounterFilePath, std::string themeIdCounterFilePath,

std::string userIdCounterFilePath);

size\_t GetDialogueListSize();

size\_t GetInterestListSize();

size\_t GetReminderListSize();

size\_t GetThemeListSize();

size\_t GetUserListSize();

void SetDialogueListSize(size\_t size);

void SetInterestListSize(size\_t size);

void SetReminderListSize(size\_t size);

void SetThemeListSize(size\_t size);

void SetUserListSize(size\_t size);

void SaveDialogueList();

void SaveInterestList();

void SaveReminderList();

void SaveThemeList();

void SaveUserList();

void LoadDialogueList();

void LoadInterestList();

void LoadReminderList();

void LoadThemeList();

void LoadUserList();

Id GetNextDialogueId();

Id GetNextInterestId();

Id GetNextReminderId();

Id GetNextThemeId();

Id GetNextUserId();

Dialogue\*\* m\_dialogueList{};

Interest\*\* m\_interestList{};

Reminder\*\* m\_reminderList{};

Theme\*\* m\_themeList{};

User\*\* m\_userList{};

private:

std::string m\_dialogueListFilePath{};

std::string m\_interestListFilePath{};

std::string m\_reminderListFilePath{};

std::string m\_themeListFilePath{};

std::string m\_userListFilePath{};

size\_t m\_dialogueListSize{};

size\_t m\_interestListSize{};

size\_t m\_reminderListSize{};

size\_t m\_themeListSize{};

size\_t m\_userListSize{};

std::string m\_dialogueIdCounterFilePath{};

std::string m\_interestIdCounterFilePath{};

std::string m\_reminderIdCounterFilePath{};

std::string m\_themeIdCounterFilePath{};

std::string m\_userIdCounterFilePath{};

IdCounter m\_dialogueIdCounter{};

IdCounter m\_interestIdCounter{};

IdCounter m\_reminderIdCounter{};

IdCounter m\_themeIdCounter{};

IdCounter m\_userIdCounter{};

};

}

#endif // !STORAGE\_H

**Storage.cpp**

#include "Storage.h"

#include <fstream>

namespace KMK

{

Storage::Storage()

{

m\_dialogueList = {};

m\_interestList = {};

m\_reminderList = {};

m\_themeList = {};

m\_userList = {};

m\_dialogueListFilePath = {};

m\_interestListFilePath = {};

m\_reminderListFilePath = {};

m\_themeListFilePath = {};

m\_userListFilePath = {};

m\_dialogueListSize = 0;

m\_interestListSize = 0;

m\_reminderListSize = 0;

m\_themeListSize = 0;

m\_userListSize = 0;

m\_dialogueIdCounterFilePath = {};

m\_interestIdCounterFilePath = {};

m\_reminderIdCounterFilePath = {};

m\_themeIdCounterFilePath = {};

m\_userIdCounterFilePath = {};

m\_dialogueIdCounter = {};

m\_interestIdCounter= {};

m\_reminderIdCounter = {};

m\_themeIdCounter = {};

m\_userIdCounter = {};

}

Storage::Storage(std::string dialogueListFilePath, std::string interestListFilePath,

std::string reminderListFilePath, std::string themeListFilePath, std::string userListFilePath,

std::string dialogueIdCounterFilePath, std::string interestIdCounterFilePath,

std::string reminderIdCounterFilePath, std::string themeIdCounterFilePath,

std::string userIdCounterFilePath)

{

m\_dialogueListFilePath = dialogueListFilePath;

m\_interestListFilePath = interestListFilePath;

m\_reminderListFilePath = reminderListFilePath;

m\_themeListFilePath = themeListFilePath;

m\_userListFilePath = userListFilePath;

LoadDialogueList();

LoadInterestList();

LoadReminderList();

LoadThemeList();

LoadUserList();

m\_dialogueIdCounterFilePath = dialogueIdCounterFilePath;

m\_interestIdCounterFilePath = interestIdCounterFilePath;

m\_reminderIdCounterFilePath = reminderIdCounterFilePath;

m\_themeIdCounterFilePath = themeIdCounterFilePath;

m\_userIdCounterFilePath = userIdCounterFilePath;

m\_dialogueIdCounter.SetIdCounter(dialogueIdCounterFilePath);

m\_interestIdCounter.SetIdCounter(interestIdCounterFilePath);

m\_reminderIdCounter.SetIdCounter(reminderIdCounterFilePath);

m\_themeIdCounter.SetIdCounter(themeIdCounterFilePath);

m\_userIdCounter.SetIdCounter(userIdCounterFilePath);

}

size\_t Storage::GetDialogueListSize() { return m\_dialogueListSize; }

size\_t Storage::GetInterestListSize() { return m\_interestListSize; }

size\_t Storage::GetReminderListSize() { return m\_reminderListSize; }

size\_t Storage::GetThemeListSize() { return m\_themeListSize; }

size\_t Storage::GetUserListSize() { return m\_userListSize; }

void Storage::SetDialogueListSize(size\_t size) { m\_dialogueListSize = size; }

void Storage::SetInterestListSize(size\_t size) { m\_interestListSize = size; }

void Storage::SetReminderListSize(size\_t size) { m\_reminderListSize = size; }

void Storage::SetThemeListSize(size\_t size) { m\_themeListSize = size; }

void Storage::SetUserListSize(size\_t size) { m\_userListSize = size; }

void Storage::SaveDialogueList()

{

std::ofstream fileWrite(m\_dialogueListFilePath, std::ios::binary);

fileWrite.write((char\*)&m\_dialogueListSize, sizeof(size\_t));

for (Iteration i{}; i < m\_dialogueListSize; ++i)

{

Id id = m\_dialogueList[i]->GetId();

fileWrite.write((char\*)&id, sizeof(Id));

fileWrite.write((char\*)&m\_dialogueList[i]->GetDate(), sizeof(Entity::Date));

size\_t stringSize = m\_dialogueList[i]->GetContent().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_dialogueList[i]->GetContent().c\_str(), stringSize);

id = m\_dialogueList[i]->GetOwnerId();

fileWrite.write((char\*)&id, sizeof(Id));

id = m\_dialogueList[i]->GetAdresseeId();

fileWrite.write((char\*)&id, sizeof(Id));

}

fileWrite.close();

}

void Storage::SaveInterestList()

{

std::ofstream fileWrite(m\_interestListFilePath, std::ios::binary);

fileWrite.write((char\*)&m\_interestListSize, sizeof(size\_t));

for (Iteration i{}; i < m\_interestListSize; ++i)

{

Id id = m\_interestList[i]->GetId();

fileWrite.write((char\*)&id, sizeof(Id));

fileWrite.write((char\*)&m\_interestList[i]->GetDate(), sizeof(Entity::Date));

size\_t stringSize = m\_interestList[i]->GetContent().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_interestList[i]->GetContent().c\_str(), stringSize);

id = m\_interestList[i]->GetOwnerId();

fileWrite.write((char\*)&id, sizeof(Id));

}

fileWrite.close();

}

void Storage::SaveReminderList()

{

std::ofstream fileWrite(m\_reminderListFilePath, std::ios::binary);

fileWrite.write((char\*)&m\_reminderListSize, sizeof(size\_t));

for (Iteration i{}; i < m\_reminderListSize; ++i)

{

Id id = m\_reminderList[i]->GetId();

fileWrite.write((char\*)&id, sizeof(Id));

fileWrite.write((char\*)&m\_reminderList[i]->GetDate(), sizeof(Entity::Date));

size\_t stringSize = m\_reminderList[i]->GetContent().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_reminderList[i]->GetContent().c\_str(), stringSize);

id = m\_reminderList[i]->GetOwnerId();

fileWrite.write((char\*)&id, sizeof(Id));

fileWrite.write((char\*)&m\_reminderList[i]->GetReminderTime(), sizeof(Entity::Date));

}

fileWrite.close();

}

void Storage::SaveThemeList()

{

std::ofstream fileWrite(m\_themeListFilePath, std::ios::binary);

fileWrite.write((char\*)&m\_themeListSize, sizeof(size\_t));

for (Iteration i{}; i < m\_themeListSize; ++i)

{

Id id = m\_themeList[i]->GetId();

fileWrite.write((char\*)&id, sizeof(Id));

fileWrite.write((char\*)&m\_themeList[i]->GetDate(), sizeof(Entity::Date));

size\_t stringSize = m\_themeList[i]->GetContent().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_themeList[i]->GetContent().c\_str(), stringSize);

id = m\_themeList[i]->GetOwnerId();

fileWrite.write((char\*)&id, sizeof(Id));

}

fileWrite.close();

}

void Storage::SaveUserList()

{

std::ofstream fileWrite(m\_userListFilePath, std::ios::binary);

fileWrite.write((char\*)&m\_userListSize, sizeof(size\_t));

for (Iteration i{}; i < m\_userListSize; ++i)

{

Id id = m\_userList[i]->GetId();

fileWrite.write((char\*)&id, sizeof(Id));

fileWrite.write((char\*)&m\_userList[i]->GetDate(), sizeof(Entity::Date));

size\_t stringSize = m\_userList[i]->GetName().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_userList[i]->GetName().c\_str(), stringSize);

stringSize = m\_userList[i]->GetLogin().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_userList[i]->GetLogin().c\_str(), stringSize);

stringSize = m\_userList[i]->GetPassword().length() + 1;

fileWrite.write((char\*)&stringSize, sizeof(size\_t));

fileWrite.write(m\_userList[i]->GetPassword().c\_str(), stringSize);

}

fileWrite.close();

}

void Storage::LoadDialogueList()

{

std::ifstream fileRead(m\_dialogueListFilePath, std::ios::binary);

fileRead.read((char\*)&m\_dialogueListSize, sizeof(size\_t));

m\_dialogueList = new Dialogue \* [m\_dialogueListSize] {};

for (Iteration i{}; i < m\_dialogueListSize; ++i)

{

Id id{};

fileRead.read((char\*)&id, sizeof(Id));

Entity::Date date{};

fileRead.read((char\*)&date, sizeof(Entity::Date));

size\_t stringSize{};

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* message = new char[stringSize] {};

fileRead.read(message, stringSize);

Id ownerId{};

fileRead.read((char\*)&ownerId, sizeof(Id));

Id adresseeId{};

fileRead.read((char\*)&adresseeId, sizeof(Id));

m\_dialogueList[i] = new Dialogue{ id, date, message, ownerId, adresseeId };

delete[] message;

}

fileRead.close();

}

void Storage::LoadInterestList()

{

std::ifstream fileRead(m\_interestListFilePath, std::ios::binary);

fileRead.read((char\*)&m\_interestListSize, sizeof(size\_t));

m\_interestList = new Interest \* [m\_interestListSize] {};

for (Iteration i{}; i < m\_interestListSize; ++i)

{

Id id{};

fileRead.read((char\*)&id, sizeof(Id));

Entity::Date date{};

fileRead.read((char\*)&date, sizeof(Entity::Date));

size\_t stringSize{};

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* interest = new char[stringSize] {};

fileRead.read(interest, stringSize);

Id ownerId{};

fileRead.read((char\*)&ownerId, sizeof(Id));

m\_interestList[i] = new Interest{ id, date, interest, ownerId };

delete[] interest;

}

fileRead.close();

}

void Storage::LoadReminderList()

{

std::ifstream fileRead(m\_reminderListFilePath, std::ios::binary);

fileRead.read((char\*)&m\_reminderListSize, sizeof(size\_t));

m\_reminderList = new Reminder \* [m\_reminderListSize] {};

for (Iteration i{}; i < m\_reminderListSize; ++i)

{

Id id{};

fileRead.read((char\*)&id, sizeof(Id));

Entity::Date date{};

fileRead.read((char\*)&date, sizeof(Entity::Date));

size\_t stringSize{};

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* reminder = new char[stringSize] {};

fileRead.read(reminder, stringSize);

Id ownerId{};

fileRead.read((char\*)&ownerId, sizeof(Id));

Entity::Date reminderDate{};

fileRead.read((char\*)&reminderDate, sizeof(Entity::Date));

m\_reminderList[i] = new Reminder{ id, date, reminder, ownerId, reminderDate };

delete[] reminder;

}

fileRead.close();

}

void Storage::LoadThemeList()

{

std::ifstream fileRead(m\_themeListFilePath, std::ios::binary);

fileRead.read((char\*)&m\_themeListSize, sizeof(size\_t));

m\_themeList = new Theme \* [m\_themeListSize] {};

for (Iteration i{}; i < m\_themeListSize; ++i)

{

Id id{};

fileRead.read((char\*)&id, sizeof(Id));

Entity::Date date{};

fileRead.read((char\*)&date, sizeof(Entity::Date));

size\_t stringSize{};

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* theme = new char[stringSize] {};

fileRead.read(theme, stringSize);

Id ownerId{};

fileRead.read((char\*)&ownerId, sizeof(Id));

m\_themeList[i] = new Theme{ id, date, theme, ownerId };

delete[] theme;

}

fileRead.close();

}

void Storage::LoadUserList()

{

std::ifstream fileRead(m\_userListFilePath, std::ios::binary);

fileRead.read((char\*)&m\_userListSize, sizeof(size\_t));

m\_userList = new User \* [m\_userListSize] {};

for (Iteration i{}; i < m\_userListSize; ++i)

{

Id id{};

fileRead.read((char\*)&id, sizeof(Id));

Entity::Date date{};

fileRead.read((char\*)&date, sizeof(Entity::Date));

size\_t stringSize{};

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* name = new char[stringSize] {};

fileRead.read(name, stringSize);

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* login = new char[stringSize] {};

fileRead.read(login, stringSize);

fileRead.read((char\*)&stringSize, sizeof(size\_t));

char\* password = new char[stringSize] {};

fileRead.read(password, stringSize);

m\_userList[i] = new User{ id, date, name, login, password };

delete[] name;

delete[] login;

delete[] password;

}

fileRead.close();

}

Id Storage::GetNextDialogueId() { return m\_dialogueIdCounter(); }

Id Storage::GetNextInterestId() { return m\_interestIdCounter(); }

Id Storage::GetNextReminderId() { return m\_reminderIdCounter(); }

Id Storage::GetNextThemeId() { return m\_themeIdCounter(); }

Id Storage::GetNextUserId() { return m\_userIdCounter(); }

}

**Add.h**

#ifndef ADD\_H

#define ADD\_H

#include "Entity.h"

namespace KMK

{

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

}

#endif // !ADD\_H

**Add.cpp**

#include "Add.h"

#include "Dialogue.h"

#include "Interest.h"

#include "Reminder.h"

#include "Theme.h"

#include "User.h"

using namespace KMK;

void KMK::Add(Entity\*\*& entities, size\_t& size, Entity\* newElement)

{

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

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

{

if (dynamic\_cast<Dialogue\*>(entities[i]))

{

temp[i] = new Dialogue{ dynamic\_cast<Dialogue\*>(entities[i]) };

}

else if (dynamic\_cast<Interest\*>(entities[i]))

{

temp[i] = new Interest{ dynamic\_cast<Interest\*>(entities[i]) };

}

else if (dynamic\_cast<Reminder\*>(entities[i]))

{

temp[i] = new Reminder{ dynamic\_cast<Reminder\*>(entities[i]) };

}

else if (dynamic\_cast<Theme\*>(entities[i]))

{

temp[i] = new Theme{ dynamic\_cast<Theme\*>(entities[i]) };

}

else if (dynamic\_cast<User\*>(entities[i]))

{

temp[i] = new User{ dynamic\_cast<User\*>(entities[i]) };

}

}

temp[size] = newElement;

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

{

delete entities[i];

}

delete[] entities;

entities = temp;

++size;

}

**Edit.h**

#ifndef EDIT\_H

#define EDIT\_H

#include "GetField.h"

namespace KMK

{

void Edit(Entity\*\*& entities, size\_t size, Id idToEdit, void\* newField, FieldMode mode);

}

#endif // !EDIT\_H

**Edit.cpp**

#include "Edit.h"

#include "Dialogue.h"

#include "Interest.h"

#include "Reminder.h"

#include "Theme.h"

#include "User.h"

namespace KMK

{

void Edit(Entity\*\*& entities, size\_t size, Id idToEdit, void\* newField, FieldMode mode)

{

unsigned short themeNumber = 0;

while (themeNumber < size)

{

if (entities[themeNumber]->GetId() != idToEdit)

{

++themeNumber;

}

else

{

break;

}

}

if (themeNumber < size)

{

if (mode == FieldMode::ID)

{

entities[themeNumber]->SetId(\*(Id\*)newField);

}

else if (mode == FieldMode::DATE)

{

entities[themeNumber]->SetDate(\*(Entity::Date\*)newField);

}

else if (mode == FieldMode::CONTENT)

{

dynamic\_cast<TextEntity\*>(entities[themeNumber])->SetContent(\*(std::string\*)newField);

}

else if (mode == FieldMode::OWNER\_ID)

{

dynamic\_cast<TextEntity\*>(entities[themeNumber])->SetOwnerId(\*(Id\*)newField);

}

else if (mode == FieldMode::ADRESSEE\_ID)

{

dynamic\_cast<Dialogue\*>(entities[themeNumber])->SetAdresseeId(\*(Id\*)newField);

}

else if (mode == FieldMode::REMINDER\_TIME)

{

dynamic\_cast<Reminder\*>(entities[themeNumber])->SetReminderTime(\*(Entity::Date\*)newField);

}

else if (mode == FieldMode::NAME)

{

dynamic\_cast<User\*>(entities[themeNumber])->SetName(\*(std::string\*)newField);

}

else if (mode == FieldMode::LOGIN)

{

dynamic\_cast<User\*>(entities[themeNumber])->SetLogin(\*(std::string\*)newField);

}

else if (mode == FieldMode::PASSWORD)

{

dynamic\_cast<User\*>(entities[themeNumber])->SetPassword(\*(std::string\*)newField);

}

}

}

}

**Filter.h**

#ifndef FILTER\_H

#define FILTER\_H

#include "GetField.h"

namespace KMK

{

void Filter(Entity\*\*& entities, size\_t& size, Id fieldForSearch, FieldMode mode);

void Filter(Entity\*\*& entities, size\_t& size, std::string fieldForSearch, FieldMode mode);

void Filter(Entity\*\*& entities, size\_t& size, Entity::Date fieldForSearch, FieldMode mode);

}

#endif // !FILTER\_H

**Filter.cpp**

#include "Filter.h"

#include "Constants.h"

#include <cmath>

#include "Dialogue.h"

#include "Interest.h"

#include "Reminder.h"

#include "Theme.h"

#include "User.h"

namespace KMK

{

void CreateNewList(Entity\*\*& entities, size\_t& size, bool\* indexes, size\_t newSize)

{

Entity\*\* filteredEntities = new Entity \* [newSize] {};

Iteration numberOfEntity = 0;

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

{

if (indexes[i] == true)

{

if (dynamic\_cast<Dialogue\*>(entities[i]))

{

filteredEntities[numberOfEntity] = new Dialogue{ dynamic\_cast<Dialogue\*>(entities[i]) };

}

else if (dynamic\_cast<Interest\*>(entities[i]))

{

filteredEntities[numberOfEntity] = new Interest{ dynamic\_cast<Interest\*>(entities[i]) };

}

else if (dynamic\_cast<Reminder\*>(entities[i]))

{

filteredEntities[numberOfEntity] = new Reminder{ dynamic\_cast<Reminder\*>(entities[i]) };

}

else if (dynamic\_cast<Theme\*>(entities[i]))

{

filteredEntities[numberOfEntity] = new Theme{ dynamic\_cast<Theme\*>(entities[i]) };

}

else if (dynamic\_cast<User\*>(entities[i]))

{

filteredEntities[numberOfEntity] = new User{ dynamic\_cast<User\*>(entities[i]) };

}

++numberOfEntity;

}

delete entities[i];

}

delete[] entities;

entities = filteredEntities;

size = newSize;

}

void Filter(Entity\*\*& entities, size\_t& size, Id fieldForSearch, FieldMode mode)

{

unsigned long int tens = 10;

unsigned short numberOfDigits = 1;

while (fieldForSearch / tens != 0)

{

tens \*= 10;

++numberOfDigits;

}

bool\* indexes = new bool[size] {};

size\_t newSize = 0;

for (Iteration i{}; i < MAXIMUM\_NUMBER\_OF\_DIGITS\_IN\_ID - numberOfDigits + 1; ++i)

{

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

{

if (fieldForSearch == (GetIdField(entities[j], mode) / (int)pow(10, i)) % tens)

{

if (indexes[j] != true)

{

indexes[j] = true;

++newSize;

}

}

}

}

CreateNewList(entities, size, indexes, newSize);

}

void Filter(Entity\*\*& entities, size\_t& size, std::string fieldForSearch, FieldMode mode)

{

unsigned short fieldForSearchLength = fieldForSearch.length();

bool\* indexes = new bool[size] {};

int newSize = 0;

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

{

for (Iteration j{}; j < GetTextField(entities[i], mode).length() - fieldForSearchLength + 1; ++j)

{

std::string temp{};

for (Iteration k{}; k < fieldForSearchLength; ++k)

{

temp += GetTextField(entities[i], mode)[j + k];

}

if (temp == fieldForSearch)

{

indexes[i] = true;

++newSize;

break;

}

}

}

CreateNewList(entities, size, indexes, newSize);

}

void Filter(Entity\*\*& entities, size\_t& size, Entity::Date fieldForSearch, FieldMode mode)

{

bool\* indexes = new bool[size] {};

unsigned short newSize = 0;

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

{

if ((GetDateField(entities[i], mode).day == fieldForSearch.day || fieldForSearch.day == 0) &&

(GetDateField(entities[i], mode).month == fieldForSearch.month || fieldForSearch.month == 0) &&

(GetDateField(entities[i], mode).year == fieldForSearch.year || fieldForSearch.year == 0))

{

indexes[i] = true;

++newSize;

}

}

CreateNewList(entities, size, indexes, newSize);

}

}

**Remove.h**

#ifndef REMOVE\_H

#define REMOVE\_H

#include "Entity.h"

namespace KMK

{

void Remove(Entity\*\*& entites, size\_t& size, Id idToRemove);

}

#endif // !REMOVE\_H

**Remove.cpp**

#include "Remove.h"

#include "Dialogue.h"

#include "Interest.h"

#include "Reminder.h"

#include "Theme.h"

#include "User.h"

using namespace KMK;

void KMK::Remove(Entity\*\*& entites, size\_t& size, Id idToRemove)

{

bool found = false;

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

{

if (idToRemove == entites[i]->GetId())

{

found = true;

break;

}

}

if (found == true)

{

Entity\*\* temp = new Entity \* [size - 1]{};

unsigned short tempElementNumber = 0;

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

{

if (entites[i]->GetId() != idToRemove)

{

if (dynamic\_cast<Dialogue\*>(entites[i]))

{

temp[tempElementNumber] = new Dialogue{ dynamic\_cast<Dialogue\*>(entites[i]) };

}

else if (dynamic\_cast<Interest\*>(entites[i]))

{

temp[tempElementNumber] = new Interest{ dynamic\_cast<Interest\*>(entites[i]) };

}

else if (dynamic\_cast<Reminder\*>(entites[i]))

{

temp[tempElementNumber] = new Reminder{ dynamic\_cast<Reminder\*>(entites[i]) };

}

else if (dynamic\_cast<Theme\*>(entites[i]))

{

temp[tempElementNumber] = new Theme{ dynamic\_cast<Theme\*>(entites[i]) };

}

else if (dynamic\_cast<User\*>(entites[i]))

{

temp[tempElementNumber] = new User{ dynamic\_cast<User\*>(entites[i]) };

}

++tempElementNumber;

}

}

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

{

delete entites[i];

}

delete[] entites;

entites = temp;

--size;

}

}

**Sort.h**

#ifndef SORT\_H

#define SORT\_H

#include "GetField.h"

namespace KMK

{

enum class OrderMode

{

DESCENDING,

ASCENDING

};

void Sort(Entity\*\*& entities, size\_t size, OrderMode order, FieldMode mode);

}

#endif // !SORT\_H

**Sort.cpp**

#include "Sort.h"

#include <iostream>

namespace KMK

{

void JustSwap(Entity\*& firstEntity, Entity\*& secondEntity)

{

Entity\* temp = firstEntity;

firstEntity = secondEntity;

secondEntity = temp;

}

void Swap(Entity\*& firstEntity, Entity\*& secondEntity, OrderMode order, FieldMode mode)

{

if (mode == FieldMode::ID || mode == FieldMode::OWNER\_ID || mode == FieldMode::ADRESSEE\_ID)

{

if (order == OrderMode::DESCENDING && GetIdField(firstEntity, mode) < GetIdField(secondEntity, mode) ||

order == OrderMode::ASCENDING && GetIdField(firstEntity, mode) > GetIdField(secondEntity, mode))

{

JustSwap(firstEntity, secondEntity);

}

}

else if (mode == FieldMode::CONTENT || mode == FieldMode::NAME || mode == FieldMode::LOGIN || mode == FieldMode::PASSWORD)

{

Iteration letter = 0;

while (GetTextField(firstEntity, mode)[letter] == GetTextField(secondEntity, mode)[letter] &&

letter < GetTextField(firstEntity, mode).length() - 1 && letter < GetTextField(secondEntity, mode).length() - 1)

{

++letter;

}

if (order == OrderMode::DESCENDING && GetTextField(firstEntity, mode)[letter] < GetTextField(secondEntity, mode)[letter] ||

order == OrderMode::ASCENDING && GetTextField(firstEntity, mode)[letter] > GetTextField(secondEntity, mode)[letter])

{

JustSwap(firstEntity, secondEntity);

}

}

else if (mode == FieldMode::DATE || mode == FieldMode::REMINDER\_TIME)

{

if (order == OrderMode::DESCENDING &&

(GetDateField(firstEntity, mode).year < GetDateField(secondEntity, mode).year ||

GetDateField(firstEntity, mode).year == GetDateField(secondEntity, mode).year &&

(GetDateField(firstEntity, mode).month < GetDateField(secondEntity, mode).month ||

GetDateField(firstEntity, mode).month == GetDateField(secondEntity, mode).month &&

GetDateField(firstEntity, mode).day < GetDateField(secondEntity, mode).day)) ||

order == OrderMode::ASCENDING &&

(GetDateField(firstEntity, mode).year > GetDateField(secondEntity, mode).year ||

GetDateField(firstEntity, mode).year == GetDateField(secondEntity, mode).year &&

(GetDateField(firstEntity, mode).month > GetDateField(secondEntity, mode).month ||

GetDateField(firstEntity, mode).month == GetDateField(secondEntity, mode).month &&

GetDateField(firstEntity, mode).day > GetDateField(secondEntity, mode).day)))

{

JustSwap(firstEntity, secondEntity);

}

}

}

void Sort(Entity\*\*& entities, size\_t size, OrderMode order, FieldMode mode)

{

if (size > 1)

{

short bottomBorder = 0;

short upperBorder = size - 1;

while (upperBorder - bottomBorder > 0)

{

for (Iteration i = bottomBorder; i < upperBorder; ++i)

{

Swap(entities[i], entities[i + 1], order, mode);

}

--upperBorder;

for (Iteration i = upperBorder; i > bottomBorder; --i)

{

Swap(entities[i - 1], entities[i], order, mode);

}

++bottomBorder;

}

}

}

}

**GetField.h**

#ifndef GET\_FIELD\_H

#define GET\_FIELD\_H

#include "Entity.h"

#include <string>

namespace KMK

{

enum class FieldMode

{

ID,

DATE,

CONTENT,

OWNER\_ID,

ADRESSEE\_ID,

REMINDER\_TIME,

NAME,

LOGIN,

PASSWORD

};

Id GetIdField(Entity\* entity, FieldMode mode);

std::string GetTextField(Entity\* entity, FieldMode mode);

Entity::Date GetDateField(Entity\* entity, FieldMode mode);

}

#endif // !GET\_FIELD\_H

**GetField.cpp**

#include "GetField.h"

#include "Dialogue.h"

#include "User.h"

#include "Reminder.h"

namespace KMK

{

Id GetIdField(Entity\* entity, FieldMode mode)

{

if (mode == FieldMode::ID)

{

return entity->GetId();

}

else if (mode == FieldMode::OWNER\_ID)

{

return dynamic\_cast<TextEntity\*>(entity)->GetOwnerId();

}

else if (mode == FieldMode::ADRESSEE\_ID)

{

return dynamic\_cast<Dialogue\*>(entity)->GetAdresseeId();

}

}

std::string GetTextField(Entity\* entity, FieldMode mode)

{

if (mode == FieldMode::CONTENT)

{

return dynamic\_cast<TextEntity\*>(entity)->GetContent();

}

else if (mode == FieldMode::NAME)

{

return dynamic\_cast<User\*>(entity)->GetName();

}

else if (mode == FieldMode::LOGIN)

{

return dynamic\_cast<User\*>(entity)->GetLogin();

}

else if (mode == FieldMode::PASSWORD)

{

return dynamic\_cast<User\*>(entity)->GetPassword();

}

}

Entity::Date GetDateField(Entity\* entity, FieldMode mode)

{

if (mode == FieldMode::DATE)

{

return entity->GetDate();

}

else if (mode == FieldMode::REMINDER\_TIME)

{

return dynamic\_cast<Reminder\*>(entity)->GetReminderTime();

}

}

}

**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"

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 IDs.dat", "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);

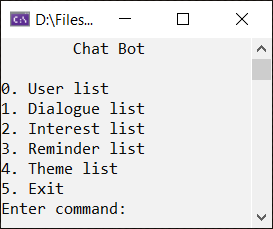
Menu menu = Menu("Chat Bot", new MenuItem\*[5] { &users, &dialogs, &interests, &reminders, &themes }, 5);

std::cin >> menu;

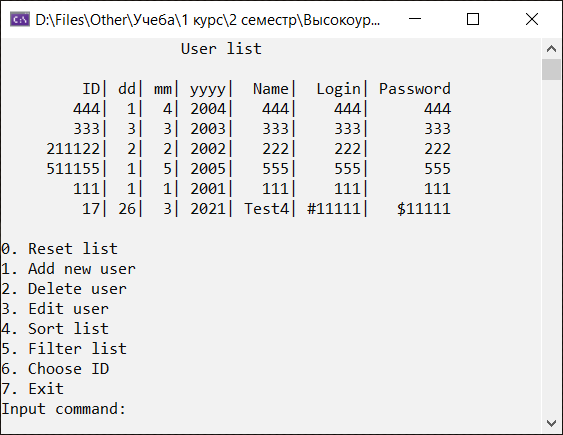
return 0;

}

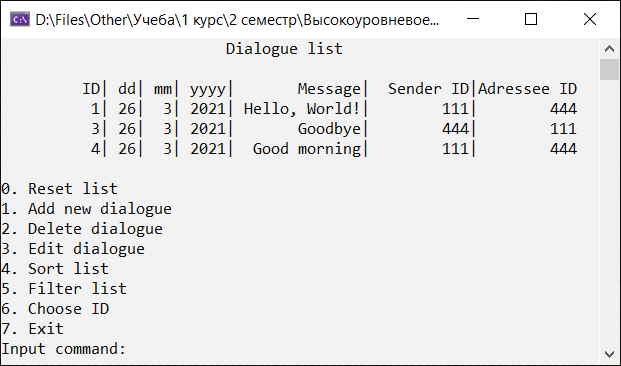
**Демонстрация:**



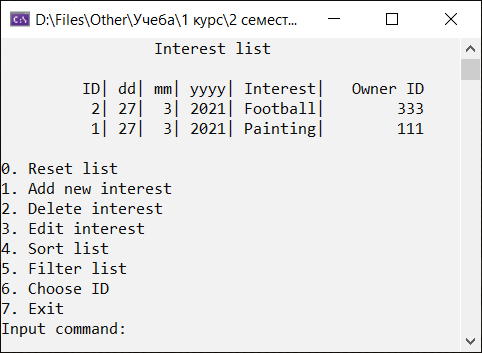
**Рисунок 2.** Главное меню



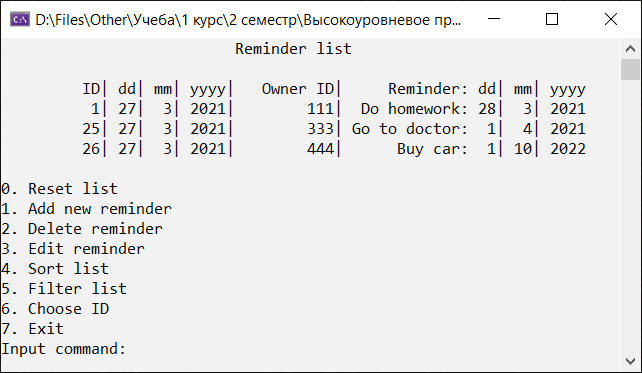
**Рисунок 3.** База данных пользователей



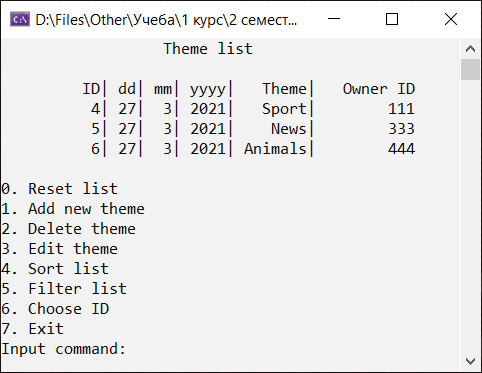
**Рисунок 4.** База данных диалогов



**Рисунок 5.** База данных интересов



**Рисунок 6.** База данных напоминаний



**Рисунок 7.** База данных тем

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки работы с dynamic\_cast, контейнерами, string.