Министерство науки и высшего образования Российской Федерации

1330

Калужский филиал

федерального государственного бюджетного образовательного учреждения высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

льныи исслеоовательскии университеі (КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ <u>ИУК «Информатика и управление»</u>

КАФЕДРА <u>ИУК4 «Программное обеспечение ЭВМ, информационные</u> технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

«Знакомство с контейнерами»

ДИСЦИПЛИНА: «Высокоуровневое программирование»

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

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 с — 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
 - о Создание / удаление / редактирование записей
 - о Сортировка / фильтрация
- Ведение базы диалогов, тем, интересов и напоминаний
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

UML-диаграмма классов:

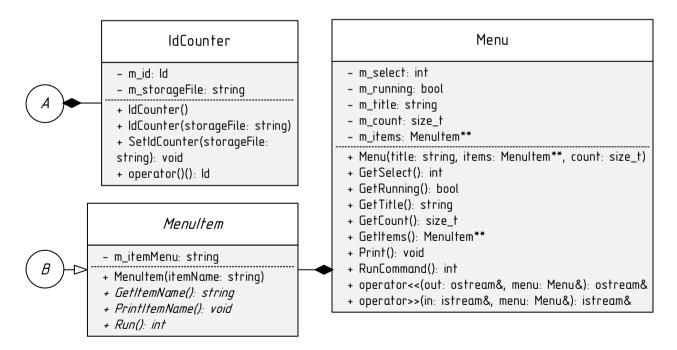


Рисунок 1.1. UML-диаграмма классов

Storage - m_dialogueListFilePath: string - m_interestListFilePath: string - m_reminderListFilePath: string - m themeListFilePath: string - m_userListFilePath: string - m_dialogueListSize: size_t - m interestListSize: size t - m reminderListSize: size t - m_themeListSize: size_t - m_userListSize: size_t - m_dialogueldCounterFilePath: string - m_interestIdCounterFilePath: string - m_reminderIdCounterFilePath: string - m themeIdCounterFilePath: string - m userIdCounterFilePath: string - m dialoqueldCounter: IdCounter - m_interestIdCounter: IdCounter - m reminderldCounter: ldCounter m themeldCounter: IdCounter - m userldCounter: ldCounter + Storage() + Storage(dialogueListFilePath: string, interestListFilePath: string, reminderListFilePath: string, themeListFilePath: string, userListFilePath: string, dialogueldCounterFilePath: string, interestIdCounterFilePath: string, reminderIdCounterFilePath: string, m_themeIdCounterFilePath: string, m_userIdCounterFilePath: string) + GetDialoqueListSize(): size_t + GetInterestListSize(): size_t + GetReminderListSize(): size_t + GetThemeListSize(): size_t + GetUserListSize(): size_t + SetDialogueListSize(size: size_t): void + SetInterestListSize(size: size_t): void + SetReminderListSize(size: size_t): void + SetThemeListSize(size: size_t): void + SetUserListSize(size: size_t): void + SaveDialogueList(): void + SaveInterestList(): void + SaveReminderList(): void + SaveThemeList(): void + SaveUserList(): void + LoadDialogueList(): void + LoadInterestList(): void + LoadReminderList(): void + LoadThemeList(): void + LoadUserList(): void + GetNextDialogueId(): Id + GetNextInterestId(): Id + GetNexReminderId(): Id + GetNextThemeld(): Id + GetNextUserId(): Id + m_dialoqueList: Dialoque** + m_interestList: Interest** + m_reminderList: Reminder** + m_themeList: Theme** + m_userList: User**

Рисунок 1.2. UML-диаграмма классов

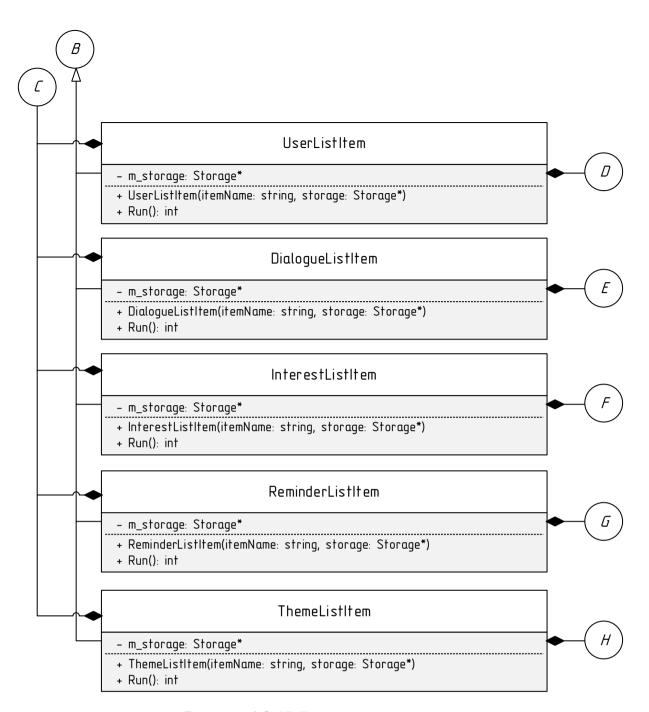


Рисунок 1.3. UML-диаграмма классов

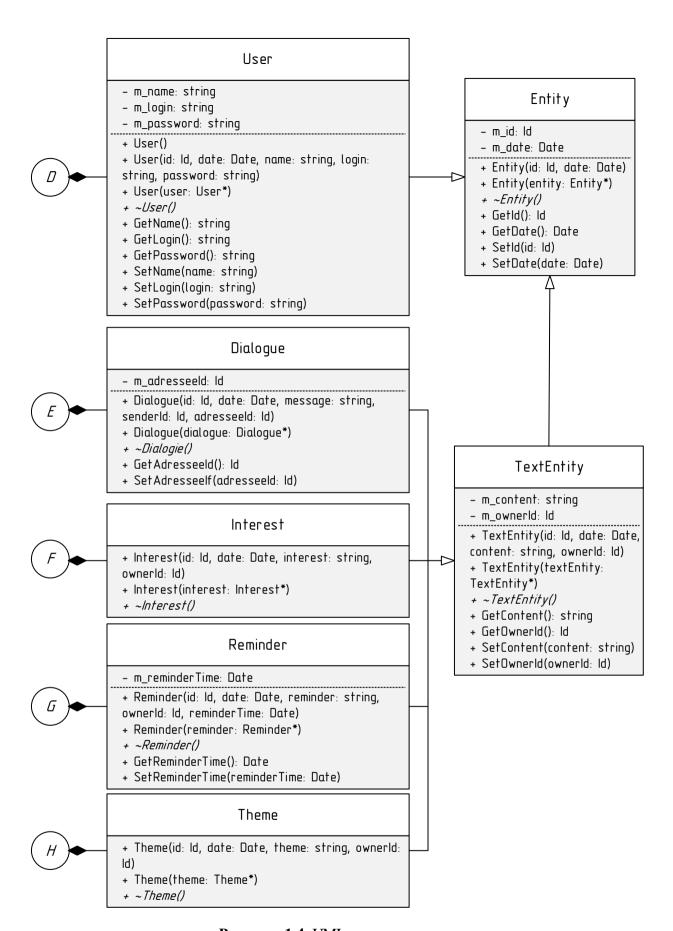


Рисунок 1.4. UML-диаграмма классов

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 DialoqueListItem :
public MenuItem
    public:
     DialogueListItem(std::strin
q 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::DialogueListIt
em(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";</pre>
          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" << "|";</pre>
```

```
std::cout <<
                                                std::cout << FILTER <<</pre>
std::setw(11) << "Sender ID" <<
                                     ". Filter list\n";
"|";
                                                std::cout << ID << ".
          std::cout <<
                                     Choose ID\n";
std::setw(11) << "Adressee ID";</pre>
                                                std::cout << EXIT <<
                                     ". Exit\n";
          std::cout << '\n';</pre>
                                                std::cout << "Input</pre>
                                     command: ";
          for (Iteration i{}; i
< size; ++i)
                                                std::cin >> command;
                                                std::cin.ignore();
                                                std::cout << '\n';
               std::cout <<
std::setw(11) << m storage-</pre>
>m dialogueList[i]->GetId() <<</pre>
                                                if (command == RESET)
"|";
               std::cout <<
                                                     m storage-
std::setw(3) << m storage-</pre>
                                     >LoadDialogueList();
>m dialogueList[i]-
>GetDate().day << "|";
                                                else if (command ==
               std::cout <<
                                     ADD)
std::setw(3) << m storage-</pre>
                                                {
>m dialogueList[i]-
                                                     std::string
>GetDate().month << "|";
                                     message{};
               std::cout <<
                                                     std::cout <<
                                     "Input message: ";
std::setw(5) << m storage-</pre>
>m dialogueList[i]-
>GetDate().year << "|";
                                          std::getline(std::cin,
                std::cout <<
                                     message);
std::setw(maximumMessageLength +
1) << m storage-
                                                     Id senderId;
>m dialogueList[i]->GetContent()
                                                     std::cout <<
<< " | ";
                                     "Input sender ID: ";
               std::cout <<
                                                     std::cin >>
std::setw(11) << m storage-</pre>
                                     senderId;
>m dialogueList[i]->GetOwnerId()
<< " | ";
                                                     Id adresseeId;
               std::cout <<
                                                     std::cout <<
std::setw(11) << m storage-</pre>
                                     "Input adressee ID: ";
>m dialogueList[i]-
                                                     std::cin >>
>GetAdresseeId();
                                     adresseeId;
               std::cout <<
'\n';
                                          std::cin.ignore();
          }
                                                     SYSTEMTIME
          std::cout << '\n';</pre>
                                     systemTime;
          std::cout << RESET <<
". Reset list\n";
                                          GetLocalTime(&systemTime);
          std::cout << ADD << ".
                                                     Dialogue*
Add new dialogue\n";
                                     newDialogue = new
          std::cout << REMOVE <<
                                     Dialogue (m storage-
". Delete dialogue\n";
                                     >GetNextDialogueId(), {
          std::cout << EDIT <<
                                     systemTime.wDay,
". Edit dialogue\n";
                                     systemTime.wMonth,
          std::cout << SORT <<
                                     systemTime.wYear }, message,
". Sort list\n";
                                     senderId, adresseeId);
```

```
std::cout <<
     Add((Entity**&)m storage-
                                    '\n';
>m dialogueList, size,
newDialoque);
                                                   if (fieldToChange
               m storage-
                                    == 0)
>SetDialogueListSize(size);
                                                        std::cout <<
          else if (command ==
                                    "Input new ID: ";
                                                        Id* newId =
REMOVE)
                                    new Id{};
                                                         std::cin >>
               std::cout <<
"Input ID: ";
                                    *newId;
               Id id;
               std::cin >> id;
                                         std::cin.ignore();
     std::cin.ignore();
                                         Edit((Entity**&)m storage-
                                    >m dialogueList, size, id,
                                    (void*)newId, FieldMode::ID);
     Remove((Entity**&)m storage
                                                        delete
->m dialogueList, size, id);
                                    newId;
               m storage-
>SetDialogueListSize(size);
                                                   if (fieldToChange
                                    == 1)
          else if (command ==
EDIT)
                                                        std::cout <<
                                    "Input new date\n";
               std::cout <<
                                                        std::cout <<
                                    "Day:";
"Input ID: ";
               Id id;
                                                        unsigned
               std::cin >> id;
                                    short day;
                                                        std::cin >>
     std::cin.iqnore();
                                    day;
                                                        std::cout <<
               std::cout <<
'\n';
                                    "Month:";
                                                        unsigned
               std::cout <<
                                    short month;
"Fields to edit\n";
                                                        std::cin >>
               std::cout << "0.
                                    month;
ID\n";
                                                        std::cout <<
               std::cout << "1.
                                    "Year:";
Date\n";
                                                        unsigned
               std::cout << "2.
                                    short year;
Message\n";
                                                        std::cin >>
               std::cout << "3.
                                    year;
Sender ID\n";
               std::cout << "4.
                                         std::cin.ignore();
Adressee ID\n";
               std::cout <<
                                         Entity::Date* newDate = new
"Choose field: ";
                                    Entity::Date{ day, month, year
               unsigned short
                                    };
fieldToChange;
                                         Edit((Entity**&)m storage-
               std::cin >>
fieldToChange;
                                    >m dialogueList, size, id,
                                    (void*)newDate,
     std::cin.ignore();
                                    FieldMode::DATE);
```

```
(void*)newAdresseeId,
                    delete
newDate;
                                    FieldMode::OWNER ID);
                                                         delete
               if (fieldToChange
                                    newAdresseeId;
== 2)
                     std::cout <<
                                              else if (command ==
"Input new message: ";
                                    SORT)
                    std::string*
message = new std::string{};
                                                    std::cout <<
                                    "Orders for sort\n";
                                                    std::cout << "0.
     std::getline(std::cin,
*message);
                                    Descending\n";
                                                    std::cout << "1.
     Edit((Entity**&)m storage-
                                    Ascending\n";
>m dialogueList, size, id,
                                                    std::cout <<
(void*) message,
                                    "Choose order: ";
FieldMode::CONTENT);
                                                    unsigned short
                    delete
                                    order;
                                                    std::cin >>
message;
                                    order;
               if (fieldToChange
                                                    std::cout <<
                                    '\n';
== 3)
                    std::cout <<
                                                    std::cout <<
"Input new sender ID: ";
                                    "Fields for sort\n";
                                                    std::cout << "0.
newSenderId = new Id{};
                                    ID\n";
                    std::cin >>
                                                    std::cout << "1.
*newSenderId;
                                    Date\n";
                                                    std::cout << "2.
     std::cin.ignore();
                                    Message\n";
                                                    std::cout << "3.
     Edit((Entity**&)m storage-
                                    Sender ID\n";
>m dialogueList, size, id,
                                                    std::cout << "4.
(void*) newSenderId,
                                    Adressee ID\n";
FieldMode::OWNER ID);
                                                    std::cout <<
                    delete
                                    "Choose field: ";
newSenderId;
                                                    unsigned short
                                    field;
               if (fieldToChange
                                                    std::cin >>
== 4)
                                    field;
                                         std::cin.ignore();
                     std::cout <<
"Input new adressee ID: ";
                                                    FieldMode
                                    sortMode = (FieldMode) -1;
                     Id*
newAdresseeId = new Id{};
                                                    switch (field)
                    std::cin >>
                                                    {
*newAdresseeId;
                                                    case 0:
                                                         sortMode =
     std::cin.ignore();
                                    FieldMode::ID;
                                                         break;
     Edit((Entity**&)m storage-
                                                    case 1:
>m dialogueList, size, id,
                                                         sortMode =
                                    FieldMode::DATE;
```

```
break;
               case 2:
                                         Filter((Entity**&)m storage
                                    ->m dialogueList, size, id,
                    sortMode =
FieldMode::CONTENT;
                                    FieldMode::ID);
                    break;
               case 3:
                                                   if (field == 1)
                    sortMode =
FieldMode::OWNER ID;
                                                        std::cout <<
                                    "Input date (if you don't want
                    break;
               case 4:
                                    to filter by the field, input
                    sortMode =
                                    0)\n";
FieldMode::ADRESSEE ID;
                                                        std::cout <<
                                    "Day: ";
                    break;
               }
                                                        unsigned
                                    short day;
     Sort((Entity**&)m storage-
                                                        std::cin >>
>m dialogueList, size,
                                    day;
(OrderMode) order, sortMode);
                                                        std::cout <<
                                    "Month: ";
          else if (command ==
                                                        unsigned
FILTER)
                                    short month;
                                                        std::cin >>
               std::cout <<
                                   month;
"Fields for filter\n";
                                                        std::cout <<
               std::cout << "0.
                                    "Year: ";
ID\n";
                                                        unsigned
               std::cout << "1.
                                    short year;
                                                        std::cin >>
Date\n";
               std::cout << "2.
                                    year;
Message\n";
                                         std::cin.ignore();
               std::cout << "3.
Sender ID\n";
                                                        Entity::Date
                                   date{ day, month, year };
               std::cout << "4.
Adressee ID\n";
               std::cout <<
                                         Filter((Entity**&)m storage
"Choose field: ";
                                    ->m dialogueList, size, date,
               unsigned short
                                    FieldMode::DATE);
field;
               std::cin >>
                                                   if (field == 2)
field;
                                                        std::cout <<
     std::cin.ignore();
                                    "Input part of message: ";
               std::cout <<
                                                        std::string
'\n';
                                    message{};
               if (field == 0)
                                         std::getline(std::cin,
                                   message);
                    std::cout <<
"Input part of ID: ";
                                         Filter((Entity**&)m storage
                    Id id;
                                    ->m dialogueList, size, message,
                    std::cin >>
                                    FieldMode::CONTENT);
id;
                                                   if (field == 3)
     std::cin.ignore();
```

```
std::cout <<
"Input part of sender ID: ";
                                   namespace KMK
                    Id senderId;
                    std::cin >>
                                        class InterestListItem :
                                   public MenuItem
senderId;
     std::cin.iqnore();
                                        public:
    Filter((Entity**&)m storage
                                       InterestListItem(std::strin
->m dialogueList, size,
                                   g itemName, Storage* storage);
senderId, FieldMode::OWNER ID);
                                             int Run();
               if (field == 4)
                                        private:
                    std::cout <<
                                             Storage* m storage{};
"Input part of adressee ID: ";
                                        };
                    Ιd
adresseeId;
                    std::cin >>
                                  #endif // !INTEREST LIST ITEM H
adresseeId;
                                        InterestListItem.cpp
     std::cin.ignore();
                                   #include "InterestListItem.h"
     Filter((Entity**&)m storage
                                   #include <iostream>
->m dialogueList, size,
                                   #include <fstream>
adresseeId,
                                   #include "Add.h"
FieldMode::ADRESSEE ID);
                                   #include <Windows.h>
                                   #include "Remove.h"
               m storage-
                                   #include "Edit.h"
>SetDialogueListSize(size);
                                   #include "Sort.h"
          }
                                   #include "Filter.h"
                                   #include <iomanip>
          if (command == ADD ||
                                   #include <string>
command == REMOVE || command ==
EDIT)
                                   using namespace KMK;
               m storage-
                                   InterestListItem::InterestListIt
>SaveDialogueList();
                                   em(std::string itemName,
                                   Storage* storage) :
                                        MenuItem(itemName)
          system("cls");
     }
                                        m storage = storage;
                                   }
     return 0;
}
                                   int InterestListItem::Run()
                                        enum Command
    InterestListItem.h
                                             RESET,
                                             ADD,
#ifndef INTEREST LIST ITEM H
                                             REMOVE,
#define INTEREST LIST ITEM H
                                             EDIT,
#include "AbstractMenuItem.h"
                                             SORT,
#include "Interest.h"
                                             FILTER,
#include "Storage.h"
                                             ID,
```

```
std::cout <<
          EXIT
     };
                                     std::setw(3) << m storage-</pre>
                                     >m interestList[i]-
                                     >GetDate().day << "|";</pre>
     unsigned short command = 0;
                                                     std::cout <<
     while (command != EXIT)
                                     std::setw(3) << m storage-</pre>
                                     >m interestList[i]-
                                     >GetDate().month << "|";
          size t size =
                                                     std::cout <<
m storage-
>GetInterestListSize();
                                     std::setw(5) << m storage-</pre>
                                     >m interestList[i]-
                                     >GetDate().year << "|";
          unsigned short
maximumInterestLength = 8;
                                                     std::cout <<
          for (Iteration i{}; i
                                     std::setw(maximumInterestLength
< size; ++i)
                                     + 1) << m storage-
                                     >m interestList[i]->GetContent()
                                     << "|";
                if (m storage-
>m interestList[i]-
                                                     std::cout <<
>GetContent().length() >
                                     std::setw(11) << m storage-</pre>
maximumInterestLength)
                                     >m interestList[i]-
                                     >GetOwnerId();
                                                     std::cout <<
     maximumInterestLength =
                                     '\n';
m storage->m interestList[i]-
>GetContent().length();
                                                std::cout << '\n';
                                                std::cout << RESET <<</pre>
                                     ". Reset list\n";
          std::cout <<
                                                std::cout << ADD << ".
std::setw((11 + 3 + 3 + 5 +
                                     Add new interest\n";
maximumInterestLength + 1 + 11 +
                                                std::cout << REMOVE <<
5 + GetItemName().length()) / 2)
                                     ". Delete interest\n";
<< GetItemName() << "\n\n";
                                               std::cout << EDIT <<
          std::cout <<
                                     ". Edit interest\n";
std::setw(11) << "ID" << "|";
                                                std::cout << SORT <<
          std::cout <<
                                     ". Sort list\n";
std::setw(3) << "dd" << "|";
                                               std::cout << FILTER <<</pre>
                                     ". Filter list\n";
          std::cout <<
std::setw(3) << "mm" << "|";
                                                std::cout << ID << ".
          std::cout <<
                                     Choose ID\n";
std::setw(5) << "yyyy" << "|";
                                                std::cout << EXIT <<</pre>
          std::cout <<
                                     ". Exit\n";
std::setw(maximumInterestLength
                                               std::cout << "Input</pre>
                                     command: ";
+ 1) << "Interest" << "|";
          std::cout <<
                                                std::cin >> command;
std::setw(11) << "Owner ID";</pre>
                                                std::cin.ignore();
          std::cout << '\n';
                                               std::cout << '\n';</pre>
          for (Iteration i{}; i
                                                if (command == RESET)
< size; ++i)
           {
                                                     m storage-
                std::cout <<
                                     >LoadInterestList();
std::setw(11) << m storage-</pre>
>m interestList[i]->GetId() <<</pre>
                                               else if (command ==
"|";
                                     ADD)
```

```
{
               std::string
                                                   std::cout <<
                                    "Input ID: ";
interest{};
               std::cout <<
                                                   Id id;
"Input interest: ";
                                                   std::cin >> id;
     std::getline(std::cin,
                                         std::cin.ignore();
interest);
                                                   std::cout <<
                                    '\n';
               Id ownerId;
               std::cout <<
                                                   std::cout <<
"Input owner ID: ";
                                    "Fields to edit\n";
                                                   std::cout << "0.
               std::cin >>
                                    ID\n";
ownerId;
                                                   std::cout << "1.
     std::cin.ignore();
                                    Date\n";
                                                   std::cout << "2.
               SYSTEMTIME
                                    Interest\n";
                                                   std::cout << "3.
systemTime;
                                    Owner ID\n";
     GetLocalTime(&systemTime);
                                                   std::cout <<
                                    "Choose field: ";
               Interest*
newInterest = new
                                                   unsigned short
                                    fieldToChange;
Interest (m storage-
>GetNextInterestId(), {
                                                   std::cin >>
                                    fieldToChange;
systemTime.wDay,
systemTime.wMonth,
systemTime.wYear }, interest,
                                         std::cin.ignore();
ownerId);
                                                   std::cout <<
                                    '\n';
     Add((Entity**&)m storage-
                                                   if (fieldToChange
>m interestList, size,
                                    == 0)
newInterest);
               m storage-
                                                        std::cout <<
>SetInterestListSize(size);
                                    "Input new ID: ";
                                                        Id* newId =
          else if (command ==
                                    new Id{};
REMOVE)
                                                        std::cin >>
                                    *newId;
               std::cout <<
"Input ID: ";
                                         std::cin.ignore();
               Id id;
               std::cin >> id;
                                         Edit((Entity**&)m storage-
                                    >m interestList, size, id,
                                    (void*)newId, FieldMode::ID);
     std::cin.ignore();
                                                        delete
                                    newId;
     Remove((Entity**&)m storage
->m interestList, size, id);
                                                   if (fieldToChange
               m storage-
                                    == 1)
>SetInterestListSize(size);
                                                        std::cout <<
          else if (command ==
                                    "Input new date\n";
EDIT)
```

```
std::cout <<
                                                         Id*
"Day: ";
                                    newOwnerId = new Id{};
                                                         std::cin >>
                    unsigned
short day;
                                    *newOwnerId;
                    std::cin >>
                                         std::cin.ignore();
day;
                    std::cout <<
"Month: ";
                                         Edit((Entity**&)m storage-
                    unsigned
                                    >m interestList, size, id,
short month;
                                    (void*)newOwnerId,
                                    FieldMode::OWNER ID);
                    std::cin >>
month;
                                                         delete
                    std::cout <<
                                    newOwnerId;
"Year: ";
                    unsigned
                                              else if (command ==
short year;
                     std::cin >>
                                    SORT)
year;
                                                    std::cout <<
                                    "Orders for sort\n";
     std::cin.ignore();
                                                    std::cout << "0.
     Entity::Date* newDate = new
                                    Descending\n";
Entity::Date{ day, month, year
                                                    std::cout << "1.
                                    Ascending\n";
};
                                                    std::cout <<
     Edit((Entity**&)m storage-
                                    "Choose order: ";
>m interestList, size, id,
                                                    unsigned short
(void*) newDate,
                                    order;
FieldMode::DATE);
                                                    std::cin >>
                    delete
                                    order;
                                                    std::cout <<
newDate;
                                    '\n';
               if (fieldToChange
== 2)
                                                    std::cout <<
                                    "Fields for sort\n";
                     std::cout <<
                                                    std::cout << "0.
"Input new interest: ";
                                    ID\n";
                                                    std::cout << "1.
                     std::string*
                                    Date\n";
interest = new std::string{};
                                                    std::cout << "2.
     std::getline(std::cin,
                                    Interest\n";
*interest);
                                                    std::cout << "3.
                                    Owner ID\n";
     Edit((Entity**&)m storage-
                                                    std::cout <<
>m interestList, size, id,
                                    "Choose field: ";
(void*) interest,
                                                    unsigned short
FieldMode::CONTENT);
                                    field;
                    delete
                                                    std::cin >>
interest;
                                    field;
               if (fieldToChange
                                        std::cin.ignore();
== 3)
                                                    FieldMode
                                    sortMode = (FieldMode) -1;
                                                    switch (field)
                     std::cout <<
"Input new owner ID: ";
```

```
case 0:
                    sortMode =
                                         Filter((Entity**&)m storage
FieldMode::ID:
                                    ->m interestList, size, id,
                                    FieldMode::ID);
                    break;
               case 1:
                    sortMode =
                                                   if (field == 1)
FieldMode::DATE;
                    break;
                                                        std::cout <<
               case 2:
                                    "Input date (if you don't want
                    sortMode =
                                    to filter by the field, input
                                    0) \n";
FieldMode::CONTENT;
                                                        std::cout <<
                    break;
                                    "Day: ";
               case 3:
                    sortMode =
                                                        unsigned
FieldMode::OWNER ID;
                                    short day;
                                                        std::cin >>
                    break;
               }
                                    day;
                                                        std::cout <<
     Sort((Entity**&)m storage-
                                    "Month: ";
>m interestList, size,
                                                        unsigned
(OrderMode) order, sortMode);
                                    short month;
                                                        std::cin >>
          else if (command ==
                                   month;
                                                        std::cout <<
FILTER)
                                    "Year: ";
               std::cout <<
                                                        unsigned
"Fields for filter\n";
                                    short year;
               std::cout << "0.
                                                        std::cin >>
ID\n";
                                    year;
               std::cout << "1.
Date\n";
                                         std::cin.ignore();
               std::cout << "2.
                                                        Entity::Date
Interest\n";
                                    date{ day, month, year };
               std::cout << "3.
Owner ID\n";
                                         Filter((Entity**&)m storage
               std::cout <<
                                    ->m interestList, size, date,
                                    FieldMode::DATE);
"Choose field: ";
               unsigned short
field;
                                                   if (field == 2)
               std::cin >>
field;
                                                        std::cout <<
                                    "Input part of interest: ";
     std::cin.ignore();
                                                        std::string
               std::cout <<
                                    interest{};
'\n';
                                         std::getline(std::cin,
               if (field == 0)
                                    interest);
                                         Filter((Entity**&)m storage
                    std::cout <<
"Input part of ID: ";
                                    ->m interestList, size,
                    Id id;
                                    interest, FieldMode::CONTENT);
                    std::cin >>
id;
                                                   if (field == 3)
```

std::cin.ignore();

```
std::cout <<
                                   #endif // !REMINDER LIST ITEM H
"Input part of owner ID: ";
                    Id ownerId;
                                         ReminderListItem.cpp
                    std::cin >>
ownerId;
                                    #include "ReminderListItem.h"
                                    #include <iostream>
     std::cin.ignore();
                                    #include <fstream>
                                    #include "Add.h"
     Filter((Entity**&)m storage
                                    #include <Windows.h>
->m interestList, size, ownerId,
                                    #include "Remove.h"
FieldMode::OWNER ID);
                                    #include "Edit.h"
               }
                                    #include "Sort.h"
               m storage-
                                    #include "Filter.h"
>SetInterestListSize(size);
                                    #include <iomanip>
          }
                                    using namespace KMK;
          if (command == ADD ||
command == REMOVE || command ==
                                    ReminderListItem::ReminderListIt
EDIT)
                                    em(std::string itemName,
                                    Storage* storage) :
               m storage-
                                        MenuItem(itemName)
>SaveInterestList();
          }
                                        m storage = storage;
                                    }
          system("cls");
     }
                                    int ReminderListItem::Run()
     return 0;
                                         enum Command
}
                                              RESET,
     ReminderListItem.h
                                              ADD,
                                              REMOVE,
#ifndef REMINDER LIST ITEM H
                                              EDIT,
#define REMINDER LIST ITEM H
                                              SORT,
#include "AbstractMenuItem.h"
                                              FILTER,
#include "Reminder.h"
                                              ID,
#include "Storage.h"
                                              EXIT
                                         };
namespace KMK
                                         unsigned short command = 0;
{
     class ReminderListItem :
public MenuItem
                                         while (command != EXIT)
     {
     public:
                                              size t size =
                                   m storage-
                                    >GetReminderListSize();
     ReminderListItem(std::strin
g itemName, Storage* storage);
                                              unsigned short
          int Run();
                                   maximumReminderLength = 8;
                                              for (Iteration i{}; i
                                    < size; ++i)
     private:
          Storage* m storage{};
     };
                                                   if (m storage-
}
                                    >m reminderList[i]-
```

```
>GetContent().length() >
                                      >m reminderList[i]-
                                      >GetDate().year << "|";</pre>
maximumReminderLength)
                                                      std::cout <<
                                      std::setw(11) << m storage-</pre>
     maximumReminderLength =
                                      >m reminderList[i]->GetOwnerId()
                                      << "|";
m storage->m reminderList[i]-
>GetContent().length();
                                                      std::cout <<
                                      std::setw(maximumReminderLength
                                      + 1) << m storage-
                                      >m reminderList[i]->GetContent()
                                      <<<sup>"</sup>":";
          std::cout <<
std::setw((11 + 3 + 3 + 5 + 11 +
                                                      std::cout <<
maximumReminderLength + 1 + 3 +
                                      std::setw(3) << m storage-</pre>
                                      >m reminderList[i]-
3 + 5 + 8 +
GetItemName().length()) / 2) <<</pre>
                                      >GetReminderTime().day << "|";</pre>
GetItemName() << "\n\n";</pre>
                                                      std::cout <<
          std::cout <<
                                      std::setw(3) << m storage-</pre>
std::setw(11) << "ID" << "|";
                                      >m reminderList[i]-
                                      >GetReminderTime().month << "|";</pre>
          std::cout <<
std::setw(3) << "dd" << "|";
                                                      std::cout <<
          std::cout <<
                                      std::setw(5) << m storage-</pre>
std::setw(3) << "mm" << "|";
                                      >m reminderList[i]-
                                      >GetReminderTime().year;
          std::cout <<
std::setw(5) << "yyyy" << "|";
                                                      std::cout <<
                                      '\n';
          std::cout <<
std::setw(11) << "Owner ID" <<
"|";
          std::cout <<
                                                std::cout << '\n';</pre>
std::setw(maximumReminderLength
                                                 std::cout << RESET <<
                                      ". Reset list\n";
+ 1) << "Reminder" << ":";
                                                 std::cout << ADD << ".
          std::cout <<
std::setw(3) << "dd" << "|";
                                      Add new reminder\n";
                                                 std::cout << REMOVE <<
          std::cout <<
std::setw(3) << "mm" << "|";
                                      ". Delete reminder\n";
          std::cout <<
                                                 std::cout << EDIT <<
std::setw(5) << "yyyy";</pre>
                                      ". Edit reminder\n";
          std::cout << '\n';
                                                std::cout << SORT <<
                                      ". Sort list\n";
                                                 std::cout << FILTER <<</pre>
          for (Iteration i{}; i
< size; ++i)
                                      ". Filter list\n";
                                                 std::cout << ID << ".
                std::cout <<
                                      Choose ID\n";
std::setw(11) << m storage-</pre>
                                                 std::cout << EXIT <<</pre>
                                      ". Exit\n";
>m reminderList[i]->GetId() <<</pre>
" | <del>"</del>;
                                                 std::cout << "Input</pre>
                                      command: ";
                std::cout <<
std::setw(3) << m storage-</pre>
                                                 std::cin >> command;
>m reminderList[i]-
                                                 std::cin.ignore();
>GetDate().day << "|";
                                                std::cout << '\n';</pre>
                std::cout <<
std::setw(3) << m storage-</pre>
                                                if (command == RESET)
>m reminderList[i]-
>GetDate().month << "|";
                                                      m storage-
                std::cout <<
                                      >LoadReminderList();
std::setw(5) << m storage-</pre>
                                                 }
```

```
else if (command ==
                                    >m reminderList, size,
ADD)
                                    newReminder);
                                                   m storage-
               Id ownerId;
                                    >SetReminderListSize(size);
               std::cout <<
                                              else if (command ==
"Input owner ID: ";
               std::cin >>
                                    REMOVE)
ownerId;
                                                   std::cout <<
                                    "Input ID: ";
     std::cin.ignore();
                                                   Id id:
                                                   std::cin >> id;
               std::string
reminder{};
               std::cout <<
                                         std::cin.ignore();
"Input reminder: ";
     std::getline(std::cin,
                                         Remove((Entity**&)m storage
reminder);
                                    ->m reminderList, size, id);
                                                   m storage-
                                    >SetReminderListSize(size);
               unsigned short
day;
                                              else if (command ==
               std::cout <<
"Input reminder day: ";
                                    EDIT)
               std::cin >> day;
                                                   std::cout <<
                                    "Input ID: ";
               unsigned short
month;
                                                   Id id;
                                                   std::cin >> id;
               std::cout <<
"Input reminder month: ";
               std::cin >>
                                         std::cin.ignore();
                                                   std::cout <<
month;
                                    '\n';
               unsigned short
                                                   std::cout <<
year;
               std::cout <<
                                    "Fields to edit\n";
"Input reminder year: ";
                                                   std::cout << "0.
               std::cin >> year;
                                    ID\n";
                                                   std::cout << "1.
     std::cin.ignore();
                                    Date\n";
                                                   std::cout << "2.
               SYSTEMTIME
                                    Owner ID\n";
                                                   std::cout << "3.
systemTime;
                                   Reminder\n";
     GetLocalTime(&systemTime);
                                                   std::cout << "4.
               Reminder*
                                   Reminder time\n";
newReminder = new
                                                   std::cout <<
                                    "Choose field: ";
Reminder (m storage-
>GetNextReminderId(), {
                                                   unsigned short
systemTime.wDay,
                                    fieldToChange;
systemTime.wMonth,
                                                   std::cin >>
systemTime.wYear }, reminder,
                                   fieldToChange;
ownerId, {day, month, year});
                                         std::cin.ignore();
                                                   std::cout <<
     Add((Entity**&)m storage-
                                    '\n';
```

```
if (fieldToChange
                                                    if (fieldToChange
== 0)
                                    == 2)
               {
                    std::cout <<
                                                         std::cout <<
"Input new ID: ";
                                    "Input new owner ID: ";
                    Id* newId =
                                                         *bT
new Id{};
                                    newOwnerId = new Id{};
                    std::cin >>
                                                         std::cin >>
*newId;
                                    *newOwnerId;
     std::cin.ignore();
                                         std::cin.ignore();
     Edit((Entity**&)m storage-
                                         Edit((Entity**&)m storage-
>m reminderList, size, id,
                                    >m reminderList, size, id,
(void*)newId, FieldMode::ID);
                                    (void*)newOwnerId,
                    delete
                                    FieldMode::OWNER ID);
newId;
                                                         delete
                                    newOwnerId;
               if (fieldToChange
== 1)
               {
                                                    if (fieldToChange
                    std::cout <<
                                    == 3)
"Input new date\n";
                    std::cout <<
                                                         std::cout <<
"Day: ";
                                    "Input new reminder: ";
                    unsigned
                                                         std::string*
short day;
                                    reminder = new std::string{};
                    std::cin >>
                                         std::getline(std::cin,
day;
                    std::cout <<
                                    *reminder);
"Month: ";
                    unsigned
                                         Edit((Entity**&)m storage-
short month;
                                    >m reminderList, size, id,
                                    (void*)reminder,
                    std::cin >>
                                    FieldMode::CONTENT);
month:
                    std::cout <<
                                                         delete
"Year: ";
                                    reminder;
                    unsigned
short year;
                                                    if (fieldToChange
                    std::cin >>
                                    == 4)
year;
                                                         std::cout <<
     std::cin.iqnore();
                                    "Input new reminder time\n";
                                                         std::cout <<
                                    "Day: ";
     Entity::Date* newDate = new
Entity::Date{ day, month, year
                                                         unsigned
};
                                    short day;
                                                         std::cin >>
     Edit((Entity**&)m storage-
                                    day;
>m reminderList, size, id,
                                                         std::cout <<
(void*) newDate,
                                    "Month: ";
FieldMode::DATE);
                                                         unsigned
                                    short month;
                    delete
newDate;
```

```
std::cin >>
                                                    std::cout <<
month;
                                    "Choose field: ";
                    std::cout <<
                                                    unsigned short
"Year: ";
                                    field;
                    unsigned
                                                    std::cin >>
                                    field;
short year;
                     std::cin >>
year;
                                         std::cin.iqnore();
                                                    FieldMode
     std::cin.ignore();
                                    sortMode = (FieldMode) -1;
                                                    switch (field)
     Entity::Date*
newReminderDate = new
                                                    case 0:
Entity::Date{ day, month, year
                                                         sortMode =
                                    FieldMode::ID;
};
                                                         break;
                                                    case 1:
     Edit((Entity**&)m storage-
>m reminderList, size, id,
                                                         sortMode =
(void*) newReminderDate,
                                    FieldMode::DATE;
FieldMode::REMINDER TIME);
                                                         break;
                    delete
                                                    case 2:
newReminderDate;
                                                         sortMode =
                                    FieldMode::OWNER ID;
               }
                                                         break;
          else if (command ==
                                                    case 3:
SORT)
                                                         sortMode =
                                    FieldMode::CONTENT;
               std::cout <<
                                                         break;
"Orders for sort\n";
                                                    case 4:
               std::cout << "0.
                                                         sortMode =
Descending\n";
                                    FieldMode::REMINDER TIME;
               std::cout << "1.
                                                         break;
Ascending\n";
                                                    }
               std::cout <<
"Choose order: ";
                                         Sort((Entity**&)m storage-
               unsigned short
                                    >m reminderList, size,
order;
                                    (OrderMode) order, sortMode);
               std::cin >>
                                               else if (command ==
order;
               std::cout <<
                                    FILTER)
'\n';
                                                    std::cout <<
               std::cout <<
                                    "Fields for filter\n";
"Fields for sort\n";
                                                    std::cout << "0.
               std::cout << "0.
                                    ID\n";
ID\n";
                                                    std::cout << "1.
               std::cout << "1.
                                    Date\n";
Date\n";
                                                    std::cout << "2.
               std::cout << "2.
                                    Owner ID\n";
Owner ID\n";
                                                    std::cout << "3.
                                    Reminder\n";
               std::cout << "3.
Reminder\n";
                                                    std::cout << "4.
               std::cout << "4.
                                    Reminder time\n";
Reminder time\n";
                                                    std::cout <<
                                    "Choose field: ";
```

```
unsigned short
field;
                                                   if (field == 2)
               std::cin >>
field;
                                                        std::cout <<
                                    "Input part of owner ID: ";
     std::cin.ignore();
                                                        Id ownerId;
               std::cout <<
                                                        std::cin >>
'\n';
                                   ownerId;
               if (field == 0)
                                        std::cin.ignore();
                                        Filter((Entity**&)m storage
                    std::cout <<
"Input part of ID: ";
                                   ->m reminderList, size, ownerId,
                                   FieldMode::OWNER ID);
                    Id id;
                    std::cin >>
id;
                                                   if (field == 3)
     std::cin.ignore();
                                                        std::cout <<
     Filter((Entity**&)m storage
                                   "Input part of reminder: ";
->m reminderList, size, id,
                                                        std::string
FieldMode::ID);
                                   reminder{};
               if (field == 1)
                                         std::getline(std::cin,
                                   reminder);
                    std::cout <<
"Input date (if you don't want
                                         Filter((Entity**&)m storage
to filter by the field, input
                                   ->m reminderList, size,
0) \n";
                                   reminder, FieldMode::CONTENT);
                    std::cout <<
"Day: ";
                                                   if (field == 4)
                    unsigned
short day;
                                                        std::cout <<
                    std::cin >>
                                   "Input reminder time (if you
                                   don't want to filter by the
day;
                                   field, input 0) \n";
                    std::cout <<
"Month: ";
                                                        std::cout <<
                                   "Day: ";
                    unsigned
short month;
                                                        unsigned
                    std::cin >>
                                   short day;
                                                        std::cin >>
month;
                    std::cout <<
                                   day;
"Year: ";
                                                        std::cout <<
                    unsigned
                                   "Month: ";
short year;
                                                        unsigned
                    std::cin >>
                                   short month;
                                                        std::cin >>
year;
                                   month;
     std::cin.ignore();
                                                        std::cout <<
                                    "Year: ";
                    Entity::Date
date{ day, month, year };
                                                        unsigned
                                   short year;
     Filter((Entity**&)m storage
                                                        std::cin >>
->m reminderList, size, date,
                                   year;
FieldMode::DATE);
```

ThemeListItem.cpp

```
std::cin.ignore();
                    Entity::Date
                                   #include "ThemeListItem.h"
date{ day, month, year };
                                   #include <iostream>
                                   #include <fstream>
     Filter((Entity**&)m storage
                                   #include "Add.h"
->m reminderList, size, date,
                                   #include <Windows.h>
FieldMode::REMINDER TIME);
                                   #include "Remove.h"
                                   #include "Edit.h"
              m storage-
                                   #include "Sort.h"
>SetReminderListSize(size);
                                   #include "Filter.h"
         }
                                   #include <iomanip>
          if (command == ADD ||
                                   using namespace KMK;
command == REMOVE || command ==
EDIT)
                                   ThemeListItem::ThemeListItem(std
                                   ::string itemName, Storage*
              m storage-
                                   storage) :
>SaveReminderList();
                                        MenuItem(itemName)
                                   {
                                        m storage = storage;
          system("cls");
     }
                                   int ThemeListItem::Run()
     return 0;
}
                                        enum Command
    ThemeListItem.h
                                             RESET,
                                             ADD,
#ifndef THEME LIST ITEM H
                                             REMOVE,
#define THEME LIST ITEM H
                                             EDIT,
#include "AbstractMenuItem.h"
                                             SORT,
#include "Theme.h"
                                             FILTER,
#include "Storage.h"
                                             ID,
                                             EXIT
namespace KMK
                                        };
     class ThemeListItem :
                                       unsigned short command = 0;
public MenuItem
                                        while (command != EXIT)
    {
    public:
                                             size t size =
     ThemeListItem(std::string
                                   m storage->GetThemeListSize();
itemName, Storage* storage);
                                            unsigned short
         int Run();
                                   maximumThemeLength = 5;
                                             for (Iteration i{}; i
     private:
                                   < size; ++i)
          Storage* m storage{};
     };
                                                 if (m storage-
                                   >m themeList[i]-
                                   >GetContent().length() >
#endif // !THEME LIST ITEM H
                                   maximumThemeLength)
                                                  {
```

```
maximumThemeLength =
                                               std::cout << '\n';</pre>
m storage->m themeList[i]-
                                               std::cout << RESET <<
>GetContent().length();
                                     ". Reset list\n";
                }
                                               std::cout << ADD << ".
                                     Add new theme\n";
                                               std::cout << REMOVE <<
                                     ". Delete theme\n";
          std::cout <<
std::setw((11 + 3 + 3 + 5 +
                                               std::cout << EDIT <<
                                     ". Edit theme\n";
maximumThemeLength + 1 + 11 + 5
+ GetItemName().length()) / 2)
                                               std::cout << SORT <<
                                     ". Sort list\n";
<< GetItemName() << "\n\n";
          std::cout <<
                                               std::cout << FILTER <<</pre>
std::setw(11) << "ID" << "|";
                                     ". Filter list\n";
          std::cout <<
                                               std::cout << ID << ".
std::setw(3) << "dd" << "|";
                                     Choose ID\n";
          std::cout <<
                                               std::cout << EXIT <<
std::setw(3) << "mm" << "|";
                                     ". Exit\n";
          std::cout <<
                                               std::cout << "Input</pre>
std::setw(5) << "yyyy" << "|";
                                     command: ";
          std::cout <<
                                               std::cin >> command;
std::setw(maximumThemeLength +
                                               std::cin.ignore();
1) << "Theme" << "|";
                                               std::cout << '\n';</pre>
          std::cout <<
std::setw(11) << "Owner ID";</pre>
                                               if (command == RESET)
          std::cout << '\n';
                                                    m storage-
                                    >LoadThemeList();
          for (Iteration i{}; i
< size; ++i)
                                               else if (command ==
          {
               std::cout <<
                                     ADD)
std::setw(11) << m storage-</pre>
                                               {
>m themeList[i]->GetId() << "|";</pre>
                                                    std::string
               std::cout <<
                                     theme{};
std::setw(3) << m storage-</pre>
                                                    std::cout <<
>m themeList[i]->GetDate().day
                                     "Input theme: ";
<< "|";
                std::cout <<
                                          std::getline(std::cin,
std::setw(3) << m storage-</pre>
                                     theme);
>m themeList[i]->GetDate().month
<< "|";
                                                    Id ownerId;
               std::cout <<
                                                    std::cout <<
std::setw(5) << m storage-</pre>
                                     "Input owner ID: ";
>m themeList[i]->GetDate().year
                                                    std::cin >>
<< "|";
                                     ownerId;
               std::cout <<
std::setw(maximumThemeLength +
                                          std::cin.ignore();
1) << m storage->m themeList[i]-
>GetContent() << "|";</pre>
                                                    SYSTEMTIME
                std::cout <<
                                     systemTime;
std::setw(11) << m storage-</pre>
>m themeList[i]->GetOwnerId();
                                          GetLocalTime(&systemTime);
               std::cout <<
                                                    Theme* newTheme =
'\n';
                                     new Theme (m storage-
          }
                                     >GetNextThemeId(), {
```

```
systemTime.wDay,
systemTime.wMonth,
                                         std::cin.ignore();
systemTime.wYear }, theme,
                                                   std::cout <<
                                    '\n';
ownerId);
                                                   if (fieldToChange
     Add((Entity**&)m storage-
                                    == () )
>m themeList, size, newTheme);
               m storage-
                                                        std::cout <<
>SetThemeListSize(size);
                                    "Input new ID: ";
                                                        Id* newId =
          else if (command ==
                                    new Id{};
REMOVE)
                                                        std::cin >>
                                    *newId;
               std::cout <<
"Input ID: ";
                                         std::cin.ignore();
               Id id;
               std::cin >> id;
                                         Edit((Entity**&)m storage-
                                    >m themeList, size, id,
                                    (void*)newId, FieldMode::ID);
     std::cin.ignore();
                                                        delete
                                    newId;
     Remove((Entity**&)m storage
->m themeList, size, id);
                                                   if (fieldToChange
               m storage-
                                    == 1)
>SetThemeListSize(size);
                                                        std::cout <<
          else if (command ==
                                    "Input new date\n";
EDIT)
                                                        std::cout <<
                                    "Day: ";
          {
               std::cout <<
                                                        unsigned
"Input ID: ";
                                    short day;
                                                        std::cin >>
               Id id;
               std::cin >> id;
                                    day;
                                                        std::cout <<
                                    "Month: ";
     std::cin.iqnore();
               std::cout <<
                                                        unsigned
'\n';
                                    short month;
                                                        std::cin >>
               std::cout <<
                                   month;
"Fields to edit\n";
                                                        std::cout <<
               std::cout << "0.
                                    "Year: ";
ID\n";
                                                        unsigned
               std::cout << "1.
                                    short year;
Date\n";
                                                        std::cin >>
               std::cout << "2.
                                    year;
Theme\n";
               std::cout << "3.
                                         std::cin.ignore();
Owner ID\n";
               std::cout <<
                                         Entity::Date* newDate = new
"Choose field: ";
                                    Entity::Date{ day, month, year
               unsigned short
                                    };
fieldToChange;
                                         Edit((Entity**&)m storage-
               std::cin >>
fieldToChange;
                                    >m themeList, size, id,
```

```
(void*) newDate,
                                                   unsigned short
FieldMode::DATE);
                                    order;
                                                    std::cin >>
                    delete
newDate;
                                    order;
                                                    std::cout <<
               if (fieldToChange
                                    '\n';
== 2)
                                                    std::cout <<
                                    "Fields for sort\n";
                    std::cout <<
"Input new theme:
                                                    std::cout << "0.
                    std::string*
                                    ID\n";
                                                    std::cout << "1.
theme = new std::string{};
                                    Date\n";
                                                    std::cout << "2.
     std::getline(std::cin,
*theme);
                                    Theme\n";
                                                    std::cout << "3.
     Edit((Entity**&)m storage-
                                    Owner ID\n";
>m themeList, size, id,
                                                    std::cout <<
(void*) theme,
                                    "Choose field: ";
FieldMode::CONTENT);
                                                   unsigned short
                    delete
                                    field;
                                                    std::cin >>
theme;
                                    field;
               if (fieldToChange
== 3)
                                         std::cin.iqnore();
                                                    FieldMode
                    std::cout <<
                                    sortMode = (FieldMode) -1;
"Input new owner ID: ";
                                                    switch (field)
newOwnerId = new Id{};
                                                    case 0:
                    std::cin >>
                                                         sortMode =
*newOwnerId;
                                    FieldMode::ID;
                                                         break;
     std::cin.ignore();
                                                    case 1:
                                                         sortMode =
     Edit((Entity**&)m storage-
                                    FieldMode::DATE;
>m themeList, size, id,
                                                         break;
(void*) newOwnerId,
                                                    case 2:
FieldMode::OWNER ID);
                                                         sortMode =
                                    FieldMode::CONTENT;
                    delete
newOwnerId;
                                                         break;
               }
                                                    case 3:
                                                         sortMode =
                                    FieldMode::OWNER ID;
          else if (command ==
SORT)
                                                         break;
               std::cout <<
                                         Sort((Entity**&)m storage-
"Orders for sort\n";
                                    >m themeList, size,
               std::cout << "0.
                                    (OrderMode) order, sortMode);
Descending\n";
               std::cout << "1.
Ascending\n";
                                              else if (command ==
               std::cout <<
                                    FILTER)
"Choose order: ";
```

```
std::cout <<
                                                        unsigned
"Fields for filter\n";
                                    short year;
               std::cout << "0.
                                                        std::cin >>
ID\n";
                                    year;
               std::cout << "1.
Date\n";
                                         std::cin.ignore();
               std::cout << "2.
                                                        Entity::Date
Theme\n";
                                    date { day, month, year };
               std::cout << "3.
Owner ID\n";
                                         Filter((Entity**&)m storage
                                    ->m themeList, size, date,
               std::cout <<
"Choose field: ";
                                    FieldMode::DATE);
               unsigned short
                                                   if (field == 2)
field;
               std::cin >>
field;
                                                        std::cout <<
                                    "Input part of theme: ";
     std::cin.ignore();
                                                        std::string
               std::cout <<
                                    theme{};
'\n';
                                         std::getline(std::cin,
               if (field == 0)
                                    theme);
                    std::cout <<
                                         Filter((Entity**&)m storage
"Input part of ID: ";
                                    ->m themeList, size, theme,
                                    FieldMode::CONTENT);
                    Id id;
                    std::cin >>
id;
                                                   if (field == 3)
     std::cin.ignore();
                                                        std::cout <<
                                    "Input part of owner ID: ";
     Filter((Entity**&)m storage
                                                        Id ownerId;
                                                        std::cin >>
->m themeList, size, id,
FieldMode::ID);
                                    ownerId;
               if (field == 1)
                                         std::cin.ignore();
                    std::cout <<
                                         Filter((Entity**&)m storage
"Input date (if you don't want
                                    ->m themeList, size, ownerId,
to filter by the field, input
                                    FieldMode::OWNER ID);
0)\n";
                    std::cout <<
                                                   m storage-
"Day: ";
                                    >SetThemeListSize(size);
                    unsigned
short day;
                    std::cin >>
                                              if (command == ADD ||
day;
                                    command == REMOVE || command ==
                    std::cout <<
                                    EDIT)
"Month: ";
                    unsigned
                                                   m storage-
short month;
                                    >SaveThemeList();
                    std::cin >>
month;
                    std::cout <<
                                              system("cls");
"Year: ";
                                         }
```

```
int UserListItem::Run()
     return 0;
}
                                         enum Command
     UserListItem.h
                                              RESET,
                                              ADD,
                                              REMOVE,
#ifndef USER LIST ITEM H
                                              EDIT,
#define USER LIST ITEM H
                                              SORT,
#include "AbstractMenuItem.h"
                                              FILTER,
#include "User.h"
#include "Storage.h"
                                              ID,
                                              EXIT
                                         };
namespace KMK
                                         unsigned short command = 0;
     class UserListItem : public
MenuItem
                                         while (command != EXIT)
     {
     public:
                                              unsigned short
                                    maximumNameLength = 4;
     UserListItem(std::string
                                              unsigned short
itemName, Storage* storage);
                                    maximumLoginLength = 5;
                                              unsigned short
          int Run();
                                    maximumPasswordLength = 8;
                                              for (Iteration i{}; i
     private:
                                    < m storage->GetUserListSize();
          Storage* m storage{};
                                    ++i)
     };
}
                                                   if (m storage-
                                    >m userList[i]-
#endif // !USER LIST ITEM H
                                    >GetName().length() >
                                    maximumNameLength)
     UserListItem.cpp
                                                   {
#include "UserListItem.h"
                                         maximumNameLength =
#include <iostream>
                                    m storage->m userList[i]-
#include <fstream>
                                    >GetName().length();
#include "Add.h"
#include <Windows.h>
                                                   if (m storage-
#include "Remove.h"
                                    >m userList[i]-
#include "Edit.h"
                                    >GetLogin().length() >
#include "Sort.h"
                                    maximumLoginLength)
#include "Filter.h"
                                                   {
#include <iomanip>
                                         maximumLoginLength =
using namespace KMK;
                                    m storage->m userList[i]-
                                    >GetLogin().length();
UserListItem::UserListItem(std::
string itemName, Storage*
                                                   if (m storage-
storage) :
                                    >m userList[i]-
     MenuItem(itemName)
                                    >GetPassword().length() >
{
                                    maximumPasswordLength)
     m storage = storage;
                                                   {
}
```

maximumPasswordLength =

```
std::cout <<
m storage->m userList[i]-
>GetPassword().length();
                                    std::setw(maximumLoginLength +
                                    1) << m storage->m userList[i]-
                                    >GetLogin() << "|";
                                                    std::cout <<
          std::cout <<
                                    std::setw(maximumPasswordLength
std::setw((11 + 3 + 3 + 5 +
                                    + 1) << m storage-
maximumNameLength + 1 +
                                    >m userList[i]->GetPassword();
maximumLoginLength + 1 +
                                                    std::cout <<
maximumPasswordLength + 1 + 6 +
                                     '\n';
GetItemName().length()) / 2) <<</pre>
GetItemName() << "\n\n";</pre>
          std::cout <<
                                               std::cout << '\n';
std::setw(11) << "ID" << "|";
                                               std::cout << RESET <<
          std::cout <<
                                     ". Reset list\n";
std::setw(3) << "dd" << "|";
                                               std::cout << ADD << ".
          std::cout <<
                                    Add new user\n";
std::setw(3) << "mm" << "|";
                                               std::cout << REMOVE <<
                                     ". Delete user\n";
          std::cout <<
std::setw(5) << "yyyy" << "|";
                                               std::cout << EDIT <<
          std::cout <<
                                     ". Edit user\n";
std::setw(maximumNameLength + 1)
                                               std::cout << SORT <<
<< "Name" << "|";
                                     ". Sort list\n";
                                               std::cout << FILTER <<</pre>
          std::cout <<
std::setw(maximumLoginLength +
                                     ". Filter list\n";
1) << "Login" << "|";
                                               std::cout << ID << ".
          std::cout <<
                                    Choose ID\n";
std::setw(maximumPasswordLength
                                               std::cout << EXIT <<
+ 1) << "Password";
                                     ". Exit\n";
          std::cout << '\n';
                                               std::cout << "Input</pre>
                                    command: ";
          for (Iteration i{}; i
                                               std::cin >> command;
< m storage->GetUserListSize();
                                               std::cin.ignore();
++i)
                                               std::cout << '\n';
               std::cout <<
                                               size t size =
std::setw(11) << m storage-</pre>
                                    m storage->GetUserListSize();
>m userList[i]->GetId() << "|";</pre>
               std::cout <<
                                               if (command == RESET)
std::setw(3) << m storage-</pre>
>m userList[i]->GetDate().day <<</pre>
                                                    m storage-
"|";
                                    >LoadUserList();
                std::cout <<
                                               else if (command ==
std::setw(3) << m storage-</pre>
>m userList[i]->GetDate().month
                                    ADD)
<< "|";
                                               {
                std::cout <<
                                                    std::string
std::setw(5) << m storage-</pre>
                                    name{};
>m userList[i]->GetDate().year
                                                    std::cout <<
<< " | ";
                                     "Input name: ";
                std::cout <<
std::setw(maximumNameLength + 1)
                                          std::getline(std::cin,
<< m storage->m userList[i]-
                                    name);
>GetName() << "|";
```

```
Id id;
               std::string
login{};
                                                   std::cin >> id;
               std::cout <<
"Input login: ";
                                         std::cin.ignore();
                                                   std::cout <<
     std::getline(std::cin,
                                    '\n';
login);
                                                   std::cout <<
                                    "Fields to edit\n";
               std::string
password{};
                                                   std::cout << "0.
               std::cout <<
                                    ID\n";
"Input password: ";
                                                   std::cout << "1.
                                    Date\n";
                                                   std::cout << "2.
     std::getline(std::cin,
password);
                                    Name\n";
                                                   std::cout << "3.
                                    Login\n";
               SYSTEMTIME
systemTime;
                                                   std::cout << "4.
                                    Password\n";
     GetLocalTime(&systemTime);
                                                   std::cout <<
               User* newUser =
                                    "Choose field: ";
new User (m storage-
                                                   unsigned short
>GetNextUserId(), {
                                    fieldToChange;
systemTime.wDay,
                                                   std::cin >>
                                    fieldToChange;
systemTime.wMonth,
systemTime.wYear }, name, login,
password);
                                         std::cin.iqnore();
                                                   std::cout <<
                                    '\n';
     Add((Entity**&)m storage-
>m userList, size, newUser);
                                                   if (fieldToChange
               m storage-
                                    == 0)
>SetUserListSize(size);
                                                        std::cout <<
          else if (command ==
                                    "Input new ID: ";
REMOVE)
                                                        Id* newId =
                                    new Id{};
          {
                                                        std::cin >>
               std::cout <<
"Input ID: ";
                                    *newId;
               Id id;
               std::cin >> id;
                                         std::cin.ignore();
     std::cin.ignore();
                                         Edit((Entity**&)m storage-
                                    >m userList, size, id,
                                    (void*)newId, FieldMode::ID);
                                                        delete
     Remove((Entity**&)m storage
->m_userList, size, id);
                                    newId;
               m storage-
>SetUserListSize(size);
                                                   if (fieldToChange
                                    == 1)
          else if (command ==
EDIT)
                                                        std::cout <<
                                    "Input new date\n";
               std::cout <<
                                                        std::cout <<
"Input ID: ";
                                    "Day: ";
```

```
unsigned
short day;
                                         std::getline(std::cin,
                    std::cin >>
                                    *login);
day;
                    std::cout <<
                                         Edit((Entity**&)m storage-
                                    >m userList, size, id,
"Month: ";
                    unsigned
                                    (void*)login, FieldMode::LOGIN);
short month;
                                                        delete
                    std::cin >>
                                    login;
month;
                    std::cout <<
                                                   if (fieldToChange
"Year: ";
                                    == 4)
                    unsigned
short year;
                                                        std::cout <<
                                    "Input new password: ";
                    std::cin >>
year;
                                                        std::string*
                                    password = new std::string{};
     std::cin.ignore();
                                         std::getline(std::cin,
     Entity::Date* newDate = new
                                    *password);
Entity::Date{ day, month, year
};
                                         Edit((Entity**&)m storage-
                                    >m userList, size, id,
     Edit((Entity**&)m storage-
                                    (void*)password,
                                    FieldMode::PASSWORD);
>m userList, size, id,
(void*) newDate,
                                                        delete
FieldMode::DATE);
                                    password;
                    delete
                                                   }
newDate;
                                              else if (command ==
               if (fieldToChange
                                    SORT)
== 2)
                                              {
                                                   std::cout <<
                    std::cout <<
                                    "Orders for sort\n";
"Input new name: ";
                                                   std::cout << "0.
                    std::string*
                                    Descending\n";
name = new std::string{};
                                                   std::cout << "1.
                                    Ascending\n";
                                                   std::cout <<
     std::getline(std::cin,
                                    "Choose order: ";
*name);
                                                   unsigned short
     Edit((Entity**&)m storage-
                                    order;
>m userList, size, id,
                                                   std::cin >>
(void*)name, FieldMode::NAME);
                                    order;
                    delete name;
                                                   std::cout <<
                                    '\n';
               if (fieldToChange
== 3)
                                                   std::cout <<
                                    "Fields for sort\n";
                    std::cout <<
                                                   std::cout << "0.
"Input new login: ";
                                    ID\n";
                                                   std::cout << "1.
                    std::string*
login = new std::string{};
                                    Date\n";
                                                   std::cout << "2.
                                    Name\n";
```

```
std::cout << "3.
                                                    std::cout << "4.
Login\n";
                                    Password\n";
               std::cout << "4.
                                                    std::cout <<
                                    "Choose field: ";
Password\n";
               std::cout <<
                                                    unsigned short
"Choose field: ";
                                    field;
               unsigned short
                                                    std::cin >>
                                    field;
field;
               std::cin >>
field;
                                         std::cin.ignore();
                                                    std::cout <<
                                    '\n';
     std::cin.ignore();
               FieldMode
sortMode = (FieldMode) -1;
                                                    if (field == 0)
               switch (field)
                                                         std::cout <<
               case 0:
                                    "Input part of ID: ";
                                                         Id id;
                    sortMode =
                                                         std::cin >>
FieldMode::ID;
                    break;
                                    id;
               case 1:
                    sortMode =
                                         std::cin.ignore();
FieldMode::DATE;
                    break;
                                         Filter((Entity**&)m storage
                                    ->m userList, size, id,
               case 2:
                                    FieldMode::ID);
                    sortMode =
FieldMode::NAME;
                                                    if (field == 1)
                    break;
               case 3:
                    sortMode =
                                                         std::cout <<
FieldMode::LOGIN;
                                    "Input date (if you don't want
                                    to filter by the field, input
                    break;
                                    0)\n";
               case 4:
                    sortMode =
                                                         std::cout <<
FieldMode::PASSWORD;
                                    "Day: ";
                    break:
                                                         unsigned
               }
                                    short day;
                                                         std::cin >>
     Sort((Entity**&)m storage-
                                    day;
>m userList, size,
                                                         std::cout <<
(OrderMode) order, sortMode);
                                    "Month: ";
                                                         unsigned
          else if (command ==
                                    short month;
                                                         std::cin >>
FILTER)
                                    month;
               std::cout <<
                                                         std::cout <<
"Fields for filter\n";
                                    "Year: ";
               std::cout << "0.
                                                         unsigned
ID\n";
                                    short year;
                                                         std::cin >>
               std::cout << "1.
Date\n";
                                    year;
               std::cout << "2.
Name\n";
                                         std::cin.ignore();
               std::cout << "3.
                                                         Entity::Date
Login\n";
                                    date{ day, month, year };
```

```
m storage-
     Filter((Entity**&)m storage
                                   >SaveUserList();
->m userList, size, date,
FieldMode::DATE);
                                             system("cls");
               if (field == 2)
                    std::cout <<
                                        return 0;
"Input part of name: ";
                                   }
                    std::string
name{};
                                        Storage.h
     std::getline(std::cin,
                                   #ifndef STORAGE H
name);
                                   #define STORAGE H
                                   #include "Dialogue.h"
     Filter((Entity**&)m storage
                                   #include "Interest.h"
->m userList, size, name,
                                   #include "Reminder.h"
FieldMode::NAME);
                                   #include "Theme.h"
                                   #include "User.h"
               if (field == 3)
                                   #include "IdCounter.h"
                    std::cout <<
                                   namespace KMK
"Input part of login: ";
                    std::string
                                        class Storage
login{};
                                        public:
     std::getline(std::cin,
                                             Storage();
login);
                                             Storage(std::string
                                   dialogueListFilePath,
     Filter((Entity**&)m storage
                                   std::string
->m userList, size, login,
                                   interestListFilePath,
FieldMode::LOGIN);
                                                  std::string
                                   reminderListFilePath,
               if (field == 4)
                                   std::string themeListFilePath,
                                   std::string userListFilePath,
                    std::cout <<
                                                  std::string
"Input part of password: ";
                                   dialogueIdCounterFilePath,
                    std::string
                                   std::string
password{};
                                   interestIdCounterFilePath,
                                                  std::string
     std::getline(std::cin,
                                   reminderIdCounterFilePath,
password);
                                   std::string
                                   themeIdCounterFilePath,
     Filter((Entity**&)m storage
                                                  std::string
->m userList, size, password,
                                   userIdCounterFilePath);
FieldMode::PASSWORD);
                                             size t
               m storage-
                                   GetDialogueListSize();
>SetUserListSize(size);
                                             size t
          }
                                   GetInterestListSize();
                                             size t
          if (command == ADD ||
                                   GetReminderListSize();
command == REMOVE || command ==
                                             size t
EDIT)
                                   GetThemeListSize();
          {
```

```
std::string
          size t
GetUserListSize();
                                    m dialogueListFilePath{};
                                              std::string
                                    m interestListFilePath{};
          void
SetDialoqueListSize(size t
                                              std::string
                                    m reminderListFilePath{};
size);
          void
                                              std::string
                                    m themeListFilePath{};
SetInterestListSize(size t
                                              std::string
size);
          void
                                    m userListFilePath{};
SetReminderListSize(size t
size);
                                              size t
                                    m dialogueListSize{};
          void
SetThemeListSize(size t size);
                                              size t
          void
                                    m interestListSize{};
SetUserListSize(size t size);
                                              size t
                                    m reminderListSize{};
          void
                                              size t
SaveDialogueList();
                                    m themeListSize{};
          void
                                              size t
SaveInterestList();
                                    m userListSize{};
          void
SaveReminderList();
                                              std::string
          void SaveThemeList();
                                   m dialogueIdCounterFilePath{};
          void SaveUserList();
                                              std::string
                                    m interestIdCounterFilePath{};
                                              std::string
          void
                                    m reminderIdCounterFilePath{};
LoadDialogueList();
          void
                                              std::string
LoadInterestList();
                                    m themeIdCounterFilePath{};
                                              std::string
          void
LoadReminderList();
                                    m userIdCounterFilePath{};
          void LoadThemeList();
          void LoadUserList();
                                              IdCounter
                                    m dialogueIdCounter{};
          Ιd
                                              IdCounter
GetNextDialogueId();
                                    m interestIdCounter{};
          Ιd
                                              IdCounter
                                    m reminderIdCounter{};
GetNextInterestId();
          Ιd
                                              IdCounter
GetNextReminderId();
                                    m themeIdCounter{};
          Id GetNextThemeId();
                                              IdCounter
                                    m userIdCounter{};
          Id GetNextUserId();
                                         } ;
          Dialoque**
                                    }
m dialogueList{};
          Interest**
                                    #endif // !STORAGE H
m interestList{};
          Reminder**
                                         Storage.cpp
m reminderList{};
          Theme** m themeList{};
                                    #include "Storage.h"
          User** m userList{};
                                    #include <fstream>
     private:
                                    namespace KMK
```

```
Storage::Storage(std::strin
     Storage::Storage()
                                    g dialogueListFilePath,
          m dialogueList = {};
                                    std::string
          m interestList = {};
                                    interestListFilePath,
          m reminderList = {};
                                              std::string
          m themeList = {};
                                    reminderListFilePath,
          m userList = {};
                                    std::string themeListFilePath,
                                    std::string userListFilePath,
          m dialogueListFilePath
                                              std::string
= { } ;
                                    dialogueIdCounterFilePath,
                                    std::string
          m interestListFilePath
                                    interestIdCounterFilePath,
= { };
          m reminderListFilePath
                                              std::string
                                    reminderIdCounterFilePath,
= { } ;
          m themeListFilePath =
                                    std::string
                                    themeIdCounterFilePath,
{ };
          m userListFilePath =
                                              std::string
{ };
                                    userIdCounterFilePath)
          m dialogueListSize =
                                              m dialoqueListFilePath
                                    = dialogueListFilePath;
0;
          m interestListSize =
                                              m interestListFilePath
0;
                                    = interestListFilePath;
          m reminderListSize =
                                              m reminderListFilePath
                                    = reminderListFilePath;
0;
                                              m themeListFilePath =
          m themeListSize = 0;
          m userListSize = 0;
                                    themeListFilePath;
                                              m userListFilePath =
                                    userListFilePath;
     m dialogueIdCounterFilePath
                                              LoadDialogueList();
= { };
                                              LoadInterestList();
                                              LoadReminderList();
    m interestIdCounterFilePath
                                              LoadThemeList();
= { };
                                              LoadUserList();
     m reminderIdCounterFilePath
= { };
                                         m dialogueIdCounterFilePath
     m themeIdCounterFilePath =
                                    = dialogueIdCounterFilePath;
{ };
                                         m interestIdCounterFilePath
                                    = interestIdCounterFilePath;
     m userIdCounterFilePath =
{ };
                                         m reminderIdCounterFilePath
                                    = reminderIdCounterFilePath;
          m dialogueIdCounter =
{ };
          m interestIdCounter=
                                         m themeIdCounterFilePath =
                                    themeIdCounterFilePath;
{ };
          m reminderIdCounter =
{ };
                                         m userIdCounterFilePath =
                                    userIdCounterFilePath;
          m themeIdCounter = {};
          m userIdCounter = {};
     }
                                         m dialogueIdCounter.SetIdCo
```

```
void
unter(dialogueIdCounterFilePath)
                                   Storage::SaveDialogueList()
    m interestIdCounter.SetIdCo
                                             std::ofstream
unter(interestIdCounterFilePath)
                                   fileWrite(m dialogueListFilePath
                                   , std::ios::binary);
    m reminderIdCounter.SetIdCo
                                        fileWrite.write((char*)&m d
unter(reminderIdCounterFilePath)
                                   ialoqueListSize,
                                   sizeof(size t));
                                             for (Iteration i{); i
     m themeIdCounter.SetIdCount
                                   < m dialogueListSize; ++i)</pre>
er(themeIdCounterFilePath);
                                                  Id id =
    m userIdCounter.SetIdCounte
                                   m dialoqueList[i]->GetId();
r(userIdCounterFilePath);
                                        fileWrite.write((char*)&id,
                                   sizeof(Id));
     size t
Storage::GetDialogueListSize() {
return m dialogueListSize; }
                                        fileWrite.write((char*)&m d
                                   ialogueList[i]->GetDate(),
     size t
Storage::GetInterestListSize() {
                                   sizeof(Entity::Date));
return m interestListSize; }
     size t
                                                  size t stringSize
Storage::GetReminderListSize() {
                                   = m dialogueList[i]-
return m reminderListSize; }
                                   >GetContent().length() + 1;
     size t
Storage::GetThemeListSize() {
                                        fileWrite.write((char*)&str
return m themeListSize; }
                                   ingSize, sizeof(size t));
     size t
Storage::GetUserListSize() {
                                        fileWrite.write(m dialogueL
return m userListSize; }
                                   ist[i]->GetContent().c str(),
                                   stringSize);
     void
Storage::SetDialogueListSize(siz
                                                   id =
e_t size) { m dialogueListSize =
                                   m dialogueList[i]->GetOwnerId();
size; }
     void
                                        fileWrite.write((char*)&id,
                                   sizeof(Id));
Storage::SetInterestListSize(siz
e t size) { m interestListSize =
                                                  id =
size; }
     void
                                   m dialogueList[i]-
Storage::SetReminderListSize(siz
                                   >GetAdresseeId();
e t size) { m reminderListSize =
size; }
                                        fileWrite.write((char*)&id,
    void
                                   sizeof(Id));
Storage::SetThemeListSize(size t
size) { m themeListSize = size;
                                             fileWrite.close();
Storage::SetUserListSize(size t
                                        void
size) { m userListSize = size; }
                                   Storage::SaveInterestList()
```

```
std::ofstream
                                                   Id id =
                                   m reminderList[i]->GetId();
fileWrite(m interestListFilePath
, std::ios::binary);
                                         fileWrite.write((char*)&id,
     fileWrite.write((char*)&m i
                                   sizeof(Id));
nterestListSize,
sizeof(size t));
          for (Iteration i{}; i
                                        fileWrite.write((char*)&m r
                                   eminderList[i]->GetDate(),
< m interestListSize; ++i)</pre>
                                   sizeof(Entity::Date));
               Id id =
m interestList[i]->GetId();
                                                   size_t stringSize
                                   = m reminderList[i]-
                                   >GetContent().length() + 1;
     fileWrite.write((char*)&id,
sizeof(Id));
                                         fileWrite.write((char*)&str
                                   ingSize, sizeof(size t));
     fileWrite.write((char*)&m i
                                         fileWrite.write(m reminderL
nterestList[i]->GetDate(),
sizeof(Entity::Date));
                                   ist[i]->GetContent().c str(),
                                   stringSize);
               size t stringSize
= m interestList[i]-
>GetContent().length() + 1;
                                   m reminderList[i]->GetOwnerId();
     fileWrite.write((char*)&str
                                         fileWrite.write((char*)&id,
ingSize, sizeof(size t));
                                   sizeof(Id));
     fileWrite.write(m interestL
ist[i]->GetContent().c str(),
                                         fileWrite.write((char*)&m r
stringSize);
                                   eminderList[i]-
                                   >GetReminderTime(),
                                   sizeof(Entity::Date));
               id =
m interestList[i]->GetOwnerId();
                                              fileWrite.close();
     fileWrite.write((char*)&id,
                                         }
sizeof(Id));
                                        void
          fileWrite.close();
                                   Storage::SaveThemeList()
     }
                                              std::ofstream
    void
                                   fileWrite(m themeListFilePath,
Storage::SaveReminderList()
                                   std::ios::binary);
          std::ofstream
                                         fileWrite.write((char*)&m t
fileWrite(m reminderListFilePath
                                   hemeListSize, sizeof(size t));
, std::ios::binary);
                                             for (Iteration i{); i
                                   < m themeListSize; ++i)</pre>
     fileWrite.write((char*)&m r
eminderListSize,
                                                   Id id =
sizeof(size t));
                                   m themeList[i]->GetId();
          for (Iteration i{}; i
< m reminderListSize; ++i)</pre>
                                         fileWrite.write((char*)&id,
                                   sizeof(Id));
```

```
fileWrite.write((char*)&m t
                                        fileWrite.write(m userList[
hemeList[i]->GetDate(),
                                   i]->GetName().c str(),
sizeof(Entity::Date));
                                   stringSize);
               size t stringSize
                                                   stringSize =
= m themeList[i]-
                                   m userList[i]-
>GetContent().length() + 1;
                                   >GetLogin().length() + 1;
     fileWrite.write((char*)&str
                                        fileWrite.write((char*)&str
ingSize, sizeof(size t));
                                   ingSize, sizeof(size t));
     fileWrite.write(m themeList
                                        fileWrite.write(m userList[
[i]->GetContent().c str(),
                                   i]->GetLogin().c str(),
stringSize);
                                   stringSize);
               id =
                                                   stringSize =
m themeList[i]->GetOwnerId();
                                   m userList[i]-
                                   >GetPassword().length() + 1;
     fileWrite.write((char*)&id,
sizeof(Id));
                                        fileWrite.write((char*)&str
                                   ingSize, sizeof(size t));
          fileWrite.close();
                                        fileWrite.write(m userList[
     }
                                   i]->GetPassword().c str(),
     void
                                   stringSize);
Storage::SaveUserList()
                                             fileWrite.close();
          std::ofstream
                                        }
fileWrite(m userListFilePath,
std::ios::binary);
                                        void
                                   Storage::LoadDialogueList()
     fileWrite.write((char*)&m u
serListSize, sizeof(size t));
                                             std::ifstream
          for (Iteration i{}; i
                                   fileRead (m dialoqueListFilePath,
< m userListSize; ++i)</pre>
                                   std::ios::binary);
                                        fileRead.read((char*)&m dia
               Id id =
m userList[i]->GetId();
                                   logueListSize, sizeof(size t));
                                             m dialogueList = new
                                   Dialogue * [m dialogueListSize]
     fileWrite.write((char*)&id,
sizeof(Id));
                                   { };
                                              for (Iteration i{}; i
                                   < m dialogueListSize; ++i)</pre>
     fileWrite.write((char*)&m u
serList[i]->GetDate(),
                                                   Id id{};
sizeof(Entity::Date));
                                        fileRead.read((char*)&id,
               size t stringSize
                                   sizeof(Id));
= m userList[i] -
>GetName().length() + 1;
                                                   Entity::Date
                                   date{};
     fileWrite.write((char*)&str
ingSize, sizeof(size t));
                                        fileRead.read((char*)&date,
                                   sizeof(Entity::Date));
```

```
size t
                                         fileRead.read((char*)&date,
                                   sizeof(Entity::Date));
stringSize{};
     fileRead.read((char*)&strin
                                                   size t
gSize, sizeof(size t));
                                   stringSize{};
               char* message =
new char[stringSize] {};
                                         fileRead.read((char*)&strin
                                   qSize, sizeof(size t));
     fileRead.read(message,
                                                   char* interest =
                                   new char[stringSize] {};
stringSize);
               Id ownerId{};
                                         fileRead.read(interest,
                                   stringSize);
     fileRead.read((char*)&owner
Id, sizeof(Id));
                                                   Id ownerId{};
               Id adresseeId{};
                                        fileRead.read((char*)&owner
                                   Id, sizeof(Id));
     fileRead.read((char*)&adres
seeId, sizeof(Id));
                                                   m interestList[i]
                                   = new Interest{ id, date,
                                   interest, ownerId };
               m dialoqueList[i]
= new Dialogue{ id, date,
message, ownerId, adresseeId };
                                                   delete[]
                                    interest;
               delete[] message;
                                              fileRead.close();
          fileRead.close();
                                         }
     }
                                        void
     void
                                   Storage::LoadReminderList()
Storage::LoadInterestList()
                                              std::ifstream
          std::ifstream
                                   fileRead(m reminderListFilePath,
fileRead(m interestListFilePath,
                                   std::ios::binary);
std::ios::binary);
                                         fileRead.read((char*)&m rem
     fileRead.read((char*)&m int
                                   inderListSize, sizeof(size t));
erestListSize, sizeof(size t));
                                              m reminderList = new
         m interestList = new
                                   Reminder * [m reminderListSize]
Interest * [m interestListSize]
                                   { };
                                              for (Iteration i{}; i
{ };
          for (Iteration i{}; i
                                   < m reminderListSize; ++i)</pre>
< m interestListSize; ++i)</pre>
                                                   Id id{};
               Id id{};
                                         fileRead.read((char*)&id,
     fileRead.read((char*)&id,
                                   sizeof(Id));
sizeof(Id));
                                                   Entity::Date
               Entity::Date
                                   date{};
date{};
                                         fileRead.read((char*)&date,
                                   sizeof(Entity::Date));
```

```
size t
                                         fileRead.read((char*)&date,
                                    sizeof(Entity::Date));
stringSize{};
     fileRead.read((char*)&strin
                                                   size t
gSize, sizeof(size t));
                                    stringSize{};
               char* reminder =
new char[stringSize] {};
                                         fileRead.read((char*)&strin
                                    qSize, sizeof(size t));
     fileRead.read(reminder,
                                                   char* theme = new
stringSize);
                                    char[stringSize] {};
               Id ownerId{};
                                         fileRead.read(theme,
                                    stringSize);
     fileRead.read((char*)&owner
Id, sizeof(Id));
                                                   Id ownerId{};
                                        fileRead.read((char*)&owner
               Entity::Date
                                    Id, sizeof(Id));
reminderDate{};
     fileRead.read((char*)&remin
                                                   m themeList[i] =
derDate, sizeof(Entity::Date));
                                    new Theme{ id, date, theme,
                                    ownerId };
               m reminderList[i]
= new Reminder{ id, date,
                                                   delete[] theme;
reminder, ownerId, reminderDate
                                              fileRead.close();
};
                                         }
               delete[]
reminder:
                                         void
                                    Storage::LoadUserList()
          fileRead.close();
     }
                                              std::ifstream
                                    fileRead (m userListFilePath,
                                    std::ios::binary);
     void
Storage::LoadThemeList()
                                         fileRead.read((char*)&m use
          std::ifstream
                                    rListSize, sizeof(size t));
fileRead(m themeListFilePath,
                                              m userList = new User
std::ios::binary);
                                      [m userListSize] {};
                                              for (Iteration i{}; i
                                    < m userListSize; ++i)</pre>
     fileRead.read((char*)&m the
meListSize, sizeof(size t));
                                              {
          m themeList = new
                                                   Id id{};
Theme * [m themeListSize] {};
          for (Iteration i{}; i
                                         fileRead.read((char*)&id,
< m themeListSize; ++i)</pre>
                                    sizeof(Id));
               Id id{};
                                                   Entity::Date
                                    date{};
     fileRead.read((char*)&id,
                                         fileRead.read((char*)&date,
sizeof(Id));
                                    sizeof(Entity::Date));
               Entity::Date
date{};
```

```
}
               size t
stringSize{};
                                         Add.h
     fileRead.read((char*)&strin
qSize, sizeof(size t));
                                    #ifndef ADD H
               char* name = new
                                    #define ADD H
char[stringSize] {};
                                    #include "Entity.h"
     fileRead.read(name,
                                    namespace KMK
stringSize);
                                         void Add(Entity**&
                                    entities, size t& size, Entity*
     fileRead.read((char*)&strin
                                    newElement);
gSize, sizeof(size t));
               char* login = new
char[stringSize] {};
                                    #endif // !ADD H
     fileRead.read(login,
                                         Add.cpp
stringSize);
                                    #include "Add.h"
                                    #include "Dialogue.h"
     fileRead.read((char*)&strin
                                    #include "Interest.h"
qSize, sizeof(size t));
                                    #include "Reminder.h"
               char* password =
                                    #include "Theme.h"
new char[stringSize] {};
                                    #include "User.h"
     fileRead.read(password,
                                    using namespace KMK;
stringSize);
                                    void KMK::Add(Entity**&
               m userList[i] =
                                    entities, size t& size, Entity*
new User{ id, date, name, login,
                                    newElement)
password };
                                         Entity** temp = new Entity
               delete[] name;
                                    * [size + 1]{};
               delete[] login;
               delete[]
                                         for (Iteration i{}; i <</pre>
password;
                                    size; ++i)
                                         {
          fileRead.close();
                                              if
     }
                                    (dynamic cast<Dialogue*>(entitie
                                    s[i]))
     Ιd
Storage::GetNextDialogueId() {
                                                   temp[i] = new
return m dialogueIdCounter(); }
                                    Dialoque {
     Ιd
                                    dynamic cast<Dialogue*>(entities
Storage::GetNextInterestId() {
                                    [i]) };
return m interestIdCounter(); }
     Ιd
                                              else if
Storage::GetNextReminderId() {
                                    (dynamic cast<Interest*>(entitie
return m reminderIdCounter(); }
                                    s[i]))
Storage::GetNextThemeId() {
                                                   temp[i] = new
return m themeIdCounter(); }
                                    Interest{
     Id Storage::GetNextUserId()
```

{ return m userIdCounter(); }

```
dynamic cast<Interest*>(entities
                                    idToEdit, void* newField,
[i]) };
                                    FieldMode mode);
          else if
(dynamic cast<Reminder*>(entitie
                                    #endif // !EDIT H
          {
                                          Edit.cpp
               temp[i] = new
Reminder{
                                    #include "Edit.h"
dynamic cast<Reminder*>(entities
                                    #include "Dialogue.h"
[i]) };
                                    #include "Interest.h"
          }
                                    #include "Reminder.h"
          else if
                                    #include "Theme.h"
(dynamic cast<Theme*>(entities[i
                                    #include "User.h"
]))
                                    namespace KMK
               temp[i] = new
Theme {
                                         void Edit(Entitv**&
dynamic cast<Theme*>(entities[i]
                                    entities, size t size, Id
) };
                                    idToEdit, void* newField,
                                    FieldMode mode)
          else if
                                          {
(dynamic cast<User*>(entities[i]
                                               unsigned short
) )
                                    themeNumber = 0;
          {
                                               while (themeNumber <</pre>
               temp[i] = new
                                    size)
User{
dynamic cast<User*>(entities[i])
                                                    i f
};
                                     (entities[themeNumber] ->GetId()
          }
                                     != idToEdit)
     }
     temp[size] = newElement;
                                         ++themeNumber;
                                                    }
     for (Iteration i{}; i <</pre>
                                                    else
size; ++i)
     {
                                                         break;
          delete entities[i];
     delete[] entities;
     entities = temp;
                                               if (themeNumber <</pre>
     ++size;
                                    size)
}
                                                    if (mode ==
     Edit.h
                                    FieldMode::ID)
#ifndef EDIT H
#define EDIT H
                                          entities[themeNumber]-
#include "GetField.h"
                                    >SetId(*(Id*)newField);
                                                    else if (mode ==
namespace KMK
                                    FieldMode::DATE)
     void Edit(Entity**&
entities, size t size, Id
                                          entities[themeNumber]-
```

```
>SetDate(*(Entity::Date*)newFiel
                                                   else if (mode ==
d);
                                    FieldMode::PASSWORD)
               else if (mode ==
FieldMode::CONTENT)
                                         dynamic cast<User*>(entitie
                                    s[themeNumber])-
                                    >SetPassword(*(std::string*)newF
     dynamic cast<TextEntity*>(e
                                    ield);
ntities[themeNumber]) -
>SetContent(*(std::string*)newFi
eld);
                                         }
                                    }
               else if (mode ==
FieldMode::OWNER ID)
                                         Filter.h
                                    #ifndef FILTER H
     dynamic cast<TextEntity*>(e
                                    #define FILTER H
ntities[themeNumber])-
                                    #include "GetField.h"
>SetOwnerId(*(Id*)newField);
                                    namespace KMK
               else if (mode ==
FieldMode::ADRESSEE ID)
                                         void Filter(Entity**&
                                    entities, size t& size, Id
                                    fieldForSearch, FieldMode mode);
     dynamic cast<Dialogue*>(ent
                                         void Filter(Entity**&
ities[themeNumber])-
                                    entities, size t& size,
>SetAdresseeId(*(Id*)newField);
                                    std::string fieldForSearch,
                                    FieldMode mode);
               else if (mode ==
                                         void Filter(Entity**&
FieldMode::REMINDER TIME)
                                    entities, size_t& size,
                                    Entity::Date fieldForSearch,
                                    FieldMode mode);
     dynamic cast<Reminder*>(ent
                                    }
ities[themeNumber])-
>SetReminderTime(*(Entity::Date*
                                    #endif // !FILTER H
) newField);
                                         Filter.cpp
               else if (mode ==
FieldMode::NAME)
                                    #include "Filter.h"
                                    #include "Constants.h"
                                    #include <cmath>
     dynamic cast<User*>(entitie
                                    #include "Dialogue.h"
s[themeNumber])-
                                    #include "Interest.h"
>SetName(*(std::string*)newField
                                    #include "Reminder.h"
);
                                    #include "Theme.h"
                                    #include "User.h"
               else if (mode ==
FieldMode::LOGIN)
                                    namespace KMK
                                         void
     dynamic cast<User*>(entitie
                                    CreateNewList (Entity * * &
s[themeNumber])-
                                    entities, size t& size, bool*
>SetLogin(*(std::string*)newFiel
                                    indexes, size t newSize)
d);
               }
```

```
Entity**
filteredEntities = new Entity *
                                       filteredEntities[numberOfEn
[newSize] {};
                                   tityl = new User{
          Iteration
                                   dynamic cast<User*>(entities[i])
numberOfEntity = 0;
                                   };
          for (Iteration i{}; i
< size; ++i)
                                        ++numberOfEntity;
               if (indexes[i] ==
true)
                                                  delete
                                   entities[i];
                    if
(dynamic cast<Dialogue*>(entitie
                                             delete[] entities;
                                             entities =
s[i]))
                    {
                                   filteredEntities;
                                             size = newSize;
     filteredEntities[numberOfEn
tity] = new Dialogue{
dynamic cast<Dialogue*>(entities
                                        void Filter(Entity**&
[i]) };
                                   entities, size t& size, Id
                                   fieldForSearch, FieldMode mode)
                    else if
                                             unsigned long int tens
(dynamic cast<Interest*>(entitie
s[i]))
                                   = 10;
                                             unsigned short
                                   numberOfDigits = 1;
     filteredEntities[numberOfEn
                                             while (fieldForSearch
tity] = new Interest{
                                   / tens != 0)
dynamic cast<Interest*>(entities
[i]) };
                                                  tens *= 10;
                                                  ++numberOfDigits;
                    else if
                                             }
(dynamic cast<Reminder*>(entitie
s[i]))
                                             bool* indexes = new
                    {
                                   bool[size] {};
                                             size t newSize = 0;
     filteredEntities[numberOfEn
                                             for (Iteration i{}; i
tity] = new Reminder{
                                   < MAXIMUM NUMBER OF DIGITS IN ID
dynamic cast<Reminder*>(entities
                                   - numberOfDigits + 1; ++i)
[i]) };
                                                  for (Iteration
                    else if
                                   j{}; j < size; j++)
(dynamic cast<Theme*>(entities[i
                                                       if
]))
                                   (fieldForSearch ==
                                   (GetIdField(entities[j], mode) /
     filteredEntities[numberOfEn
                                   (int)pow(10, i)) % tens)
tity] = new Theme{
dynamic cast<Theme*>(entities[i]
                                                             if
) };
                                   (indexes[j] != true)
                    else if
(dynamic cast<User*>(entities[i]
                                       indexes[j] = true;
) )
                                        ++newSize;
```

```
}
                    }
                                         CreateNewList(entities,
                                    size, indexes, newSize);
               }
          }
                                         void Filter(Entity**&
     CreateNewList(entities,
                                    entities, size t& size,
                                    Entity::Date fieldForSearch,
size, indexes, newSize);
                                    FieldMode mode)
     void Filter(Entity**&
                                              bool* indexes = new
entities, size t& size,
                                   bool[size] {};
std::string fieldForSearch,
                                              unsigned short newSize
FieldMode mode)
                                    = 0;
                                              for (Iteration i{}; i
          unsigned short
                                    < size; ++i)
fieldForSearchLength =
fieldForSearch.length();
                                    ((GetDateField(entities[i],
                                    mode).day == fieldForSearch.day
          bool* indexes = new
bool[size] {};
                                    || fieldForSearch.day == 0) &&
          int newSize = 0;
          for (Iteration i{}; i
                                         (GetDateField(entities[i],
                                    mode).month ==
< size; ++i)
                                    fieldForSearch.month ||
                                    fieldForSearch.month == 0) &&
               for (Iteration
j{}; j <
GetTextField(entities[i],
                                         (GetDateField(entities[i],
mode).length() -
                                   mode) .year ==
fieldForSearchLength + 1; ++j)
                                    fieldForSearch.year ||
                                    fieldForSearch.year == 0))
                    std::string
temp{};
                                                        indexes[i] =
                    for
                                    true;
(Iteration k{}; k <
                                                        ++newSize;
fieldForSearchLength; ++k)
                                              }
                         temp +=
GetTextField(entities[i],
mode)[j + k];
                                        CreateNewList(entities,
                                    size, indexes, newSize);
                    if (temp ==
fieldForSearch)
                                    }
                                         Remove.h
     indexes[i] = true;
                                    #ifndef REMOVE H
     ++newSize;
                                    #define REMOVE H
                         break;
                                    #include "Entity.h"
                    }
               }
                                   namespace KMK
                                         void Remove(Entity**&
                                    entites, size t& size, Id
                                    idToRemove);
```

```
else if
}
                                    (dynamic cast<Interest*>(entites
#endif // !REMOVE H
                                    [i]))
                                                        {
    Remove.cpp
                                        temp[tempElementNumber] =
                                   new Interest{
#include "Remove.h"
                                   dynamic cast<Interest*>(entites[
#include "Dialogue.h"
#include "Interest.h"
                                   i]) };
#include "Reminder.h"
                                                        else if
#include "Theme.h"
                                    (dynamic cast<Reminder*>(entites
#include "User.h"
                                    [i]))
using namespace KMK;
                                        temp[tempElementNumber] =
void KMK::Remove(Entity**&
                                   new Reminder{
entites, size t& size, Id
                                   dynamic cast<Reminder*>(entites[
idToRemove)
    bool found = false;
                                                        else if
    for (Iteration i{}; i <</pre>
                                    (dynamic cast<Theme*>(entites[i]
size; ++i)
                                   ) )
          if (idToRemove ==
entites[i]->GetId())
                                        temp[tempElementNumber] =
                                   new Theme {
               found = true;
                                   dynamic cast<Theme*>(entites[i])
               break;
                                                        }
     }
                                                        else if
                                    (dynamic cast<User*>(entites[i])
     if (found == true)
          Entity** temp = new
Entity * [size - 1]{};
                                        temp[tempElementNumber] =
         unsigned short
                                   new User{
tempElementNumber = 0;
                                   dynamic cast<User*>(entites[i])
          for (Iteration i{}; i
< size; ++i)
               if (entites[i]-
                                         ++tempElementNumber;
>GetId() != idToRemove)
                    if
(dynamic cast<Dialogue*>(entites
                                              for (Iteration i{}; i
[i]))
                                   < size; ++i)
                                                   delete
     temp[tempElementNumber] =
                                   entites[i];
new Dialogue{
dynamic_cast<Dialogue*>(entites[
                                              delete[] entites;
i]) };
                                              entites = temp;
                    }
                                              --size;
                                         }
```

}	<pre>GetIdField(secondEntity, mode) </pre>
Sort.h	order ==
	OrderMode::ASCENDING &&
#ifndef SORT_H	GetIdField(firstEntity, mode)
#define SORT_H	GetIdField(secondEntity, mode)
<pre>#include "GetField.h"</pre>	{
namespace KMK	JustSwap(firstEntity,
{	secondEntity);
enum class OrderMode	}
{	}
DESCENDING,	else if (mode ==
ASCENDING	FieldMode::CONTENT mode ==
} ;	FieldMode::NAME mode ==
	FieldMode::LOGIN mode ==
<pre>void Sort(Entity**&</pre>	FieldMode::PASSWORD)
entities, size_t size, OrderMode	T++
order, FieldMode mode);	Iteration letter
}	= 0;
	while
<pre>#endif // !SORT_H</pre>	(GetTextField(firstEntity,
	mode) [letter] ==
Sort.cpp	GetTextField(secondEntity,
	mode) [letter] &&
<pre>#include "Sort.h"</pre>	letter <
<pre>#include <iostream></iostream></pre>	GetTextField(firstEntity,
	mode).length() - 1 && letter <
namespace KMK	<pre>GetTextField(secondEntity, mode).length() - 1)</pre>
{	mode).length() - 1)
void JustSwap(Entity*&	++letter;
firstEntity, Entity*&	1116661,
secondEntity)	J
{	if (order ==
Entity* temp =	OrderMode::DESCENDING &&
firstEntity;	GetTextField(firstEntity,
firstEntity =	mode) [letter] <
secondEntity;	GetTextField(secondEntity,
<pre>secondEntity = temp;</pre>	mode) [letter]
}	order ==
	OrderMode::ASCENDING &&
void Swap (Entity*&	<pre>GetTextField(firstEntity,</pre>
firstEntity, Entity*&	<pre>mode) [letter] ></pre>
secondEntity, OrderMode order,	GetTextField(secondEntity,
FieldMode mode)	mode)[letter])
if (mode ==	{
FieldMode::ID mode ==	
	<pre>JustSwap(firstEntity,</pre>
FieldMode::OWNER_ID mode ==	secondEntity);
FieldMode::ADRESSEE_ID)	}
if (order	}
<pre>if (order == OrderMode::DESCENDING &&</pre>	else if (mode ==
GetIdField(firstEntity, mode) <	FieldMode::DATE mode ==
occidence (in stancing, mode)	FieldMode::REMINDER_TIME)

```
if (order ==
OrderMode::DESCENDING &&
                                         JustSwap (firstEntity,
                                    secondEntity);
     (GetDateField(firstEntity,
mode).year <</pre>
GetDateField(secondEntity,
mode).year ||
                                         void Sort(Entity**&
     GetDateField(firstEntity,
                                    entities, size t size, OrderMode
                                    order, FieldMode mode)
mode).vear ==
GetDateField(secondEntity,
mode).year &&
                                              if (size > 1)
     (GetDateField(firstEntity,
                                                    short
mode).month <</pre>
                                    bottomBorder = 0;
                                                    short upperBorder
GetDateField(secondEntity,
                                    = size - 1;
mode).month ||
                                                    while
     GetDateField(firstEntity,
                                    (upperBorder - bottomBorder > 0)
mode) .month ==
GetDateField(secondEntity,
                                                         for
                                    (Iteration i = bottomBorder; i <
mode).month &&
                                    upperBorder; ++i)
     GetDateField(firstEntity,
mode) .day <</pre>
GetDateField(secondEntity,
                                         Swap (entities [i],
                                    entities[i + 1], order, mode);
mode).day)) ||
                    order ==
OrderMode::ASCENDING &&
                                    upperBorder;
     (GetDateField(firstEntity,
mode).year >
                                                         for
GetDateField(secondEntity,
                                    (Iteration i = upperBorder; i >
mode).year ||
                                    bottomBorder; --i)
     GetDateField(firstEntity,
                                         Swap(entities[i - 1],
mode).year ==
GetDateField(secondEntity,
                                    entities[i], order, mode);
mode).year &&
     (GetDateField(firstEntity,
                                         ++bottomBorder;
mode).month >
GetDateField(secondEntity,
mode).month ||
     GetDateField(firstEntity,
mode) .month ==
                                         GetField.h
GetDateField(secondEntity,
mode).month &&
                                    #ifndef GET FIELD H
                                    #define GET FIELD H
     GetDateField(firstEntity,
                                    #include "Entity.h"
mode).day >
                                    #include <string>
GetDateField(secondEntity,
mode).day)))
                                    namespace KMK
```

```
{
                                                   return
                                   dynamic cast<Dialogue*>(entity) -
     enum class FieldMode
                                   >GetAdresseeId();
          ID,
          DATE,
          CONTENT,
          OWNER ID,
                                        std::string
          ADRESSEE ID,
                                   GetTextField(Entity* entity,
          REMINDER TIME,
                                   FieldMode mode)
          NAME,
                                             if (mode ==
          LOGIN,
          PASSWORD
                                   FieldMode::CONTENT)
     };
                                                   return
     Id GetIdField(Entity*
                                   dynamic cast<TextEntity*>(entity
entity, FieldMode mode);
                                   )->GetContent();
     std::string
GetTextField(Entity* entity,
                                             else if (mode ==
FieldMode mode);
                                   FieldMode::NAME)
     Entity::Date
GetDateField(Entity* entity,
                                                  return
FieldMode mode);
                                   dynamic cast<User*>(entity)-
                                   >GetName();
#endif // !GET FIELD H
                                             else if (mode ==
                                   FieldMode::LOGIN)
     GetField.cpp
                                             {
                                                  return
                                   dynamic cast<User*>(entity) -
#include "GetField.h"
                                   >GetLogin();
#include "Dialogue.h"
#include "User.h"
                                             else if (mode ==
#include "Reminder.h"
                                   FieldMode::PASSWORD)
namespace KMK
                                                  return
                                   dynamic cast<User*>(entity) -
    Id GetIdField(Entity*
                                   >GetPassword();
entity, FieldMode mode)
                                              }
          if (mode ==
FieldMode::ID)
                                        Entity::Date
                                   GetDateField(Entity* entity,
               return entity-
                                   FieldMode mode)
>GetId();
                                             if (mode ==
          else if (mode ==
                                   FieldMode::DATE)
FieldMode::OWNER ID)
                                                  return entity-
               return
                                   >GetDate();
dynamic_cast<TextEntity*>(entity
)->GetOwnerId();
                                             else if (mode ==
                                   FieldMode::REMINDER TIME)
          else if (mode ==
FieldMode::ADRESSEE ID)
```

```
std::cin >> menu;
               return
dynamic cast<Reminder*>(entity) -
>GetReminderTime();
                                         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"
using namespace KMK;
int main()
     Storage* storage = new
Storage ("Dialogue database.dat",
"Interst database.dat",
          "Reminder
database.dat", "Theme
```

"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",

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

storage);

Демонстрация:

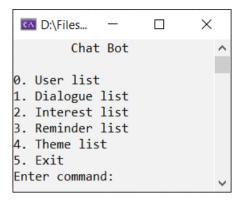


Рисунок 2. Главное меню

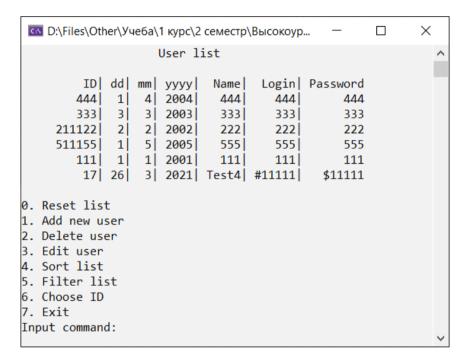


Рисунок 3. База данных пользователей

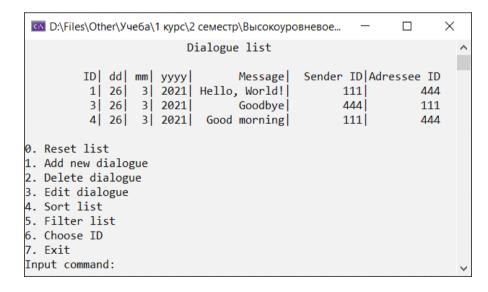


Рисунок 4. База данных диалогов

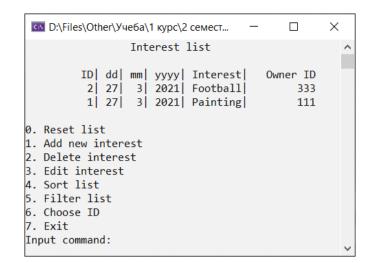


Рисунок 5. База данных интересов

```
■ D:\Files\Other\Учеба\1 курс\2 семестр\Высокоуровневое пр...

                         Reminder list
        ID | dd | mm | yyyy |
                                           Reminder: dd | mm | yyyy
                            Owner ID
         1 27 3 2021
                              111 Do homework: 28 3 2021
         25 | 27 | 3 | 2021 |
                                 333 Go to doctor: 1 4 2021
         26 27 3 2021
                                 444
                                            Buy car: 1 | 10 | 2022
Reset list
1. Add new reminder
Delete reminder
3. Edit reminder
4. Sort list
5. Filter list
6. Choose ID
7. Exit
Input command:
```

Рисунок 6. База данных напоминаний

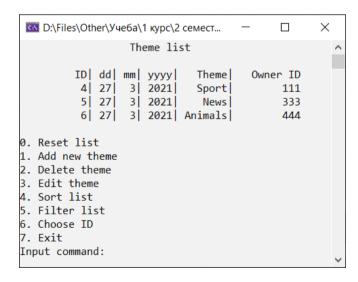


Рисунок 7. База данных тем

Вывод: в ходе выполнения лабораторной работы были получены практические навыки работы с dynamic_cast, контейнерами, string.