Министерство науки и высшего образования Российской Федерации

1830

Калужский филиал

федерального государственного бюджетного образовательного учреждения высшего образования Осковский государственный технический университет имени Н.Э. Баул

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	_ИУК	«Информатика	и управление)	<u>></u>	
КАФЕДРА _	<u>ИУК4</u>	«Программное	обеспечение	ЭВМ,	информационные
технологии»					

ЛАБОРАТОРНАЯ РАБОТА №3

«Перегрузка операторов»

ДИСЦИПЛИНА: «Высокоуровневое программирование»

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

Калуга, 2021

Цель: приобретение практических навыков и знаний по работе с перегрузкой операторов.

Задачи:

- 1. Изучить понятия оператора и что он из себя представляет;
- 2. Выяснить виды и способы перегрузки операторов;
- 3. Научиться применять перегрузку операторов на практике;
- 4. Изучить методы и случаи применения перегрузок;
- 5. Научиться соединять пользовательские объекты с потоками ввода/вывода;
- 6. Познакомиться с понятием функтора.

Вариант 8

Задание:

Общая 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

Индивидуальные задания

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu — это объект класса Меnu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы — сущности данных вашей программы: пользователь, диалог, тема, интерес, напоминание. Для класса: пользователь создайте общий класс родитель — человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора — id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций — арр.срр (арр.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (пользователь, диалог, тема, интерес, напоминание). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (пользователи, напоминания, темы и т д) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (пользователь, напоминания, темы и т д).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять пользователя / диалог / темы / напоминание / интерес. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Все сущности программы – пока не должны быть жёстко связаны между собой.

UML-диаграмма классов:

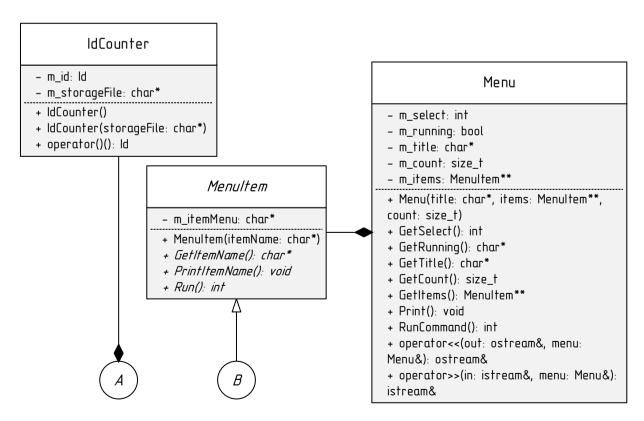


Рисунок 1.1. UML-диаграмма классов

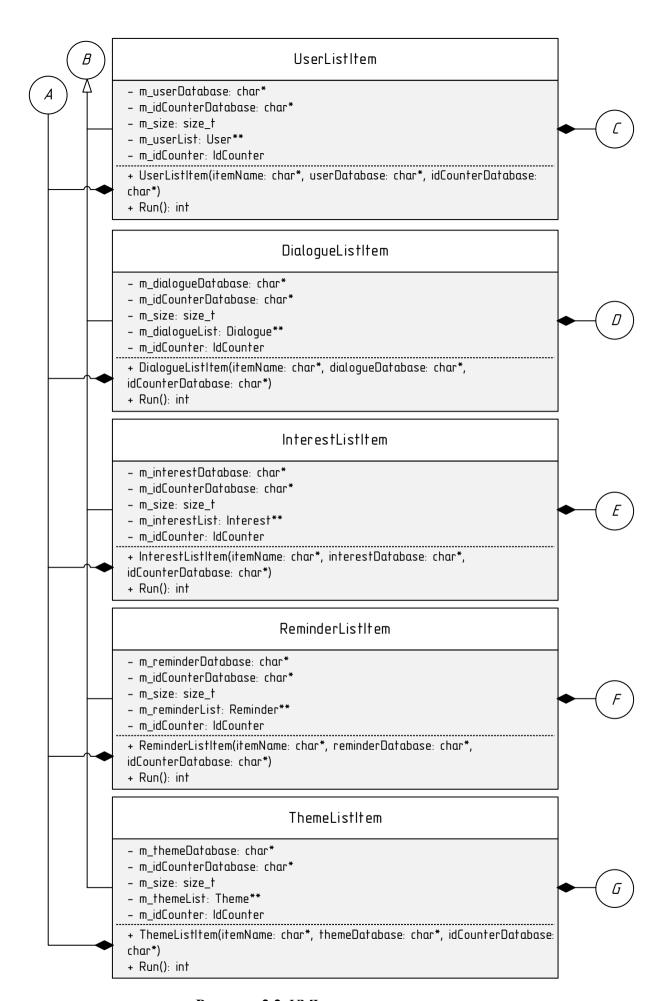


Рисунок 2.2. UML-диаграмма классов

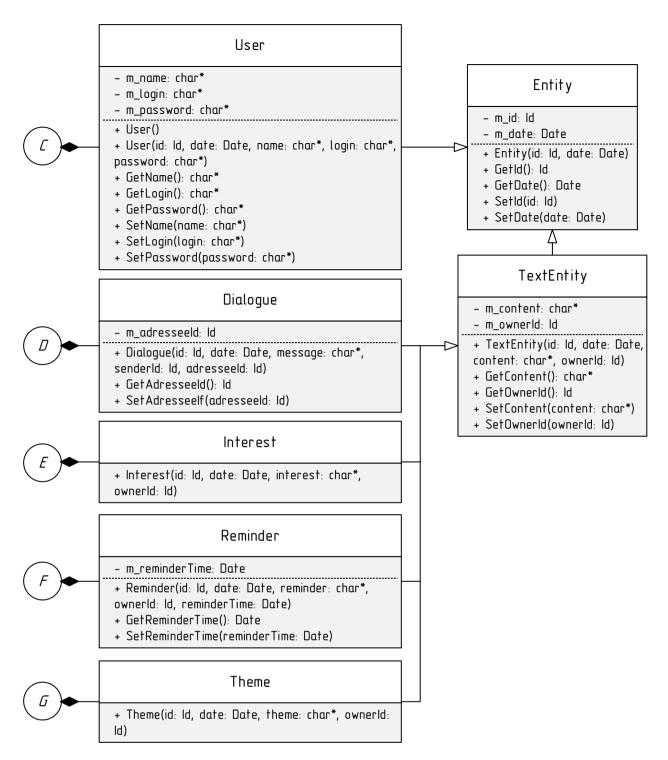


Рисунок 3.3. UML-диаграмма классов

Листинг:

```
Id m id{};
const int KMK::LENGTH OF FIELD
= 255;
                                                Date m date{};
                                           };
     const int
                                      }
KMK::MAXIMUM NUMBER OF DIGITS I
N ID = 10;
                                           #endif // !ENTITY H
     Type Definitions.h
                                           Entity.cpp
#ifndef TYPE DEFINITIONS
                                     #include "Entity.h"
#define TYPE DEFINITIONS H
                                     using namespace KMK;
namespace KMK
                                     Entity::Entity(Id id, Date
     typedef unsigned long int
                                     date)
Id;
     typedef unsigned short int
                                          m id = id;
Iteration;
                                          m date = date;
                                      }
     #endif //
                                     Id Entity::GetId() { return
!TYPE DEFINITIONS
                                     m id; }
                                     Entity::Date Entity::GetDate()
     Entity.h
                                      { return m date; }
#ifndef ENTITY H
                                     void Entity::SetId(Id id) {
#define ENTITY H
                                     m id = id; }
#include "TypeDefinitions.h"
                                     void Entity::SetDate(Date date)
                                           m date.day = date.day;
namespace KMK
                                           m date.month = date.month;
     class Entity
                                           m date.year= date.year;
                                      }
     public:
                                           TextEntity.h
          struct Date
                                     #ifndef TEXT ENTITY H
               unsigned short
                                      #define TEXT ENTITY H
day{};
               unsigned short
                                     #include "Entity.h"
month{};
               unsigned short
                                     namespace KMK
year{};
                                           class TextEntity : public
          };
                                     Entity
          Entity(Id id, Date
                                           {
date);
                                           public:
                                                TextEntity(Id id,
          Id GetId();
                                     Date date, char* content, Id
          Date GetDate();
                                     ownerId);
          void SetId(Id id);
                                                char* GetContent();
          void SetDate(Date
                                                Id GetOwnerId();
date);
                                                void SetContent(char*
     private:
                                     content);
```

```
void SetOwnerId(Id
                                          public:
ownerId);
                                                Dialogue (Id id, Date
                                     date, char* message, Id
                                     senderId, Id adresseeId);
     private:
          char* m content{};
          Id m ownerId{};
                                                Id GetAdresseeId();
     };
                                                void SetAdresseeId(Id
}
                                     adresseeId);
     #endif // !TEXT ENTITY H
                                          private:
     TextEntity.cpp
                                                Id m adresseeId{};
                                           };
#include "TextEntity.h"
                                     }
#include "Constants.h"
#include <iostream>
                                          #endif // !DIALOGUE H
                                          Dialogue.cpp
using namespace KMK;
TextEntity::TextEntity(Id id,
                                     #include "Dialogue.h"
Date date, char* content, Id
ownerId) : Entity(id, date)
                                     using namespace KMK;
     m content = new
                                     Dialogue::Dialogue(Id id, Date
char[LENGTH OF FIELD] {};
                                     date, char* message, Id
                                     senderId, Id adresseeId) :
     strcpy_s(m_content,
LENGTH OF FIELD, content);
                                          TextEntity(id, date,
     m ownerId = ownerId;
                                     message, senderId)
}
                                          m adresseeId = adresseeId;
char* TextEntity::GetContent()
{ return m content; }
                                     Id Dialogue::GetAdresseeId() {
Id TextEntity::GetOwnerId() {
                                     return m adresseeId; }
return m ownerId; }
                                          void
Void
                                     Dialogue::SetAdresseeId(Id
                                     adresseeId) { m adresseeId =
TextEntity::SetContent(char*
content) { strcpy s(m content,
                                     adresseeId; }
LENGTH_OF_FIELD, content); }
void TextEntity::SetOwnerId(Id
                                          Interest.h
ownerId) { m ownerId = ownerId;
                                     #ifndef INTEREST H
}
                                     #define INTEREST H
     Dialogue.h
                                     #include "TextEntity.h"
#ifndef DIALOGUE H
                                     namespace KMK
#define DIALOGUE H
#include "TextEntity.h"
                                          class Interest : public
                                     TextEntity
namespace KMK
                                           {
                                          public:
     class Dialogue : public
                                                Interest(Id id, Date
TextEntity
                                     date, char* interest, Id
                                     ownerId);
     {
```

```
Reminder::Reminder(Id id, Date
     } ;
}
                                     date, char* reminder, Id
                                     ownerId, Date reminderTime) :
     #endif // !INTEREST H
                                          TextEntity(id, date,
                                     reminder, ownerId)
     Interest.cpp
                                          m reminderTime =
                                     reminderTime;
#include "Interest.h"
using namespace KMK;
                                     Reminder::Date
                                     Reminder::GetReminderTime() {
Interest::Interest(Id id, Date
date, char* interest, Id
                                     return m reminderTime; }
ownerId) :
                                     void
          TextEntity(id, date,
                                     Reminder::SetReminderTime(Date
interest, ownerId) {}
                                     reminderTime)
     Reminder.h
                                          m reminderTime.day =
#ifndef REMINDER H
                                     reminderTime.day;
#define REMINDER H
                                          m reminderTime.month =
                                     reminderTime.month;
#include "TextEntity.h"
                                          m reminderTime.year =
                                     reminderTime.year;
namespace KMK
                                           }
     class Reminder : public
                                          Theme.h
TextEntity
     public:
                                     #ifndef THEME H
                                     #define THEME H
          Reminder (Id id, Date
                                     #include "TextEntity.h"
date, char* reminder, Id
ownerId, Date reminderTime);
                                     namespace KMK
          Date
GetReminderTime();
                                          class Theme : public
                                     TextEntity
          void
                                          {
SetReminderTime(Date
                                          public:
reminderTime);
                                                Theme (Id id, Date
                                     date, char* theme, Id ownerId);
     private:
                                          };
                                     }
          Date
m reminderTime{};
                                           #endif // !THEME H
     };
}
                                          Theme.cpp
     #endif // !REMINDER H
                                     #include "Theme.h"
     Reminder.cpp
                                     using namespace KMK;
#include "Reminder.h"
                                     Theme::Theme(Id id, Date date,
                                     char* theme, Id ownerId) :
using namespace KMK;
                                                TextEntity(id, date,
                                     theme, ownerId){}
```

```
{
                                          m name = new
#ifndef USER H
                                     char[LENGTH OF FIELD];
#define USER H
                                          strcpy_s(m_name,
#include "Entity.h"
                                     LENGTH OF FIELD, name);
                                          m login = new
                                     char[LENGTH OF FIELD];
namespace KMK
                                          strcpy s(m login,
     class User : public Entity
                                     LENGTH OF FIELD, login);
                                          m password = new
    public:
                                     char[LENGTH OF FIELD];
                                          strcpy_s(m_password,
          User();
         User(Id id, Date
                                     LENGTH OF FIELD, password);
date, char* name, char* login,
char* password);
                                     char* User::GetName() { return
          char* GetName();
                                     m name; }
                                     char* User::GetLogin() { return
          char* GetLogin();
          char* GetPassword();
                                     m login; }
                                     char* User::GetPassword() {
          void SetName(char*
                                     return m password; }
name);
          void SetLogin(char*
                                     void User::SetName(char* name)
login);
                                     { strcpy s(m name,
                                     LENGTH OF FIELD, name); }
          void
                                     void User::SetLogin(char*
SetPassword(char* password);
                                     login) { strcpy s(m login,
                                     LENGTH OF FIELD, login); }
    private:
          char* m name{};
                                          void
          char* m login{};
                                     User::SetPassword(char*
          char* m password{};
                                     password) {
     };
                                     strcpy s(m password,
                                     LENGTH OF FIELD, password); }
}
                                          IdCounter.h
     #endif // !USER H
     User.cpp
                                     #ifndef ID_COUNTER_H
                                     #define ID COUNTER H
#include "User.h"
                                     #include "TypeDefinitions.h"
#include "Constants.h"
#include <iostream>
                                     namespace KMK
using namespace KMK;
                                          class IdCounter
User::User() : Entity(0, {})
                                          public:
                                               IdCounter();
    m name = {};
                                               IdCounter(char*
    m login = {};
                                     storageFile);
    m password = {};
}
                                               Id operator() ();
User::User(Id id, Date date,
                                          private:
char* name, char* login, char*
                                               Id m id{};
                                               char* m storageFile =
password) :
    Entity(id, date)
                                     nullptr;
```

User.h

```
};
                                          class MenuItem
}
                                          public:
     #endif // !ID COUNTER H
                                              MenuItem(char*
                                     itemName);
     IdCounter.cpp
                                               virtual char*
#include "IdCounter.h"
                                     GetItemName();
#include "Constants.h"
                                               virtual void
#include <iostream>
                                     PrintItemName();
#include <fstream>
                                               virtual int Run() =
                                     0;
using namespace KMK;
                                          private:
IdCounter::IdCounter()
                                                char* m itemName =
                                     nullptr;
     m id = 0;
                                          };
     m storageFile = nullptr;
                                           #endif //
IdCounter::IdCounter(char*
                                     !ABSTRACT MENU ITEM H
storageFile)
                                          AbstractMenuItem.cpp
    m storageFile = new
char[LENGTH OF FIELD] {};
                                     #include "AbstractMenuItem.h"
     strcpy s(m storageFile,
                                     #include "Constants.h"
LENGTH OF FIELD, storageFile);
                                     #include <iostream>
     std::ifstream
                                     using namespace KMK;
fileRead(m storageFile,
std::ios::binary);
                                     MenuItem::MenuItem(char*
     fileRead.read((char*)&m id
                                     itemName)
, sizeof(m id));
     fileRead.close();
                                          m itemName = new
                                     char[LENGTH OF FIELD];
}
                                           strcpy s(m itemName,
Id IdCounter::operator() ()
                                     LENGTH OF FIELD, itemName);
     ++m id;
     std::ofstream
                                     char* MenuItem::GetItemName()
fileWrite(m storageFile,
std::ios::binary);
                                          return m itemName;
     fileWrite.write((char*)&m
id, sizeof(m id));
     fileWrite.close();
                                     void MenuItem::PrintItemName()
     return m id;
                                           std::cout << m itemName;</pre>
}
                                     }
     AbstractMenuItem.h
                                          DialogueListItem.h
#ifndef ABSTRACT MENU ITEM H
#define ABSTRACT MENU ITEM H
                                     #ifndef DIALOGUE LIST ITEM H
                                     #define DIALOGUE LIST ITEM H
                                     #include "AbstractMenuItem.h"
namespace KMK
                                     #include "Dialogue.h"
```

```
#include "IdCounter.h"
                                          m dialogueDatabase = new
                                     char[LENGTH OF FIELD] {};
                                           strcpy s(m dialogueDatabas
namespace KMK
                                     e, LENGTH OF FIELD,
                                     dialogueDatabase);
     class DialoqueListItem :
public MenuItem
                                          m idCounterDatabase = new
                                     char[LENGTH OF FIELD] {};
                                           strcpy s(m idCounterDataba
     public:
                                     se, LENGTH OF FIELD,
     DialogueListItem(char*
                                     idCounterDatabase);
itemName, char*
dialogueDatabase, char*
                                          m idCounter = {
idCounterDatabase);
                                     idCounterDatabase };
          int Run();
                                          std::ifstream
                                     fileRead(m dialogueDatabase,
     private:
                                     std::ios::binary);
                                          fileRead.read((char*)&m si
          char*
                                     ze, sizeof(size t));
m dialogueDatabase = nullptr;
                                          m dialogueList = new
          char*
                                     Dialoque * [m size] {};
m idCounterDatabase = nullptr;
          size t m size{};
                                          for (Iteration i{}; i <</pre>
          Dialogue**
                                     m size; ++i)
m dialogueList = nullptr;
                                           {
          IdCounter
                                                Id id{};
m idCounter{};
                                          fileRead.read((char*)&id,
     };
}
                                     sizeof(Id));
     #endif //
                                                Entity::Date date{};
!DIALOGUE LIST ITEM H
                                          fileRead.read((char*)&date
     DialogueListItem.cpp
                                     , sizeof(Entity::Date));
#include "DialogueListItem.h"
                                               char* message = new
#include "Constants.h"
                                     char[LENGTH OF FIELD] {};
#include <iostream>
#include <fstream>
                                          fileRead.read(message,
#include "Add.h"
                                     LENGTH OF FIELD);
#include <Windows.h>
#include "Remove.h"
                                                Id ownerId{};
#include "Edit.h"
#include "Sort.h"
                                          fileRead.read((char*)&owne
#include "Filter.h"
                                     rId, sizeof(Id));
#include <iomanip>
                                                Id adresseeId{};
using namespace KMK;
                                          fileRead.read((char*)&adre
                                     sseeId, sizeof(Id));
DialogueListItem::DialogueListI
tem(char* itemName, char*
dialogueDatabase, char*
                                               m dialogueList[i] =
idCounterDatabase) :
                                     new Dialogue{ id, date,
MenuItem(itemName)
                                     message, ownerId, adresseeId };
{
                                          fileRead.close();
```

```
}
                                                 std::cout <<
                                      std::setw(11) << "Sender ID" <<
                                      "|";
int DialogueListItem::Run()
                                                 std::cout <<
                                      std::setw(11) << "Adressee ID";</pre>
     enum Command
                                                 std::cout << '\n';</pre>
          RESET,
          ADD,
                                                 for (Iteration i{}; i
          REMOVE,
                                      < m size; ++i)
          EDIT,
          SORT,
                                                      std::cout <<
                                      std::setw(11) <<
          FILTER,
          ID,
                                      m dialogueList[i]->GetId() <<</pre>
                                      "<u>|</u>";
          EXIT
     };
                                                      std::cout <<
                                      std::setw(3) <<
     unsigned short command =
                                      m dialoqueList[i]-
                                      >GetDate().day << "|";</pre>
0;
                                                      std::cout <<
     while (command != EXIT)
                                      std::setw(3) <<
                                      m dialogueList[i]-
                                      >GetDate().month << "|";
          unsigned short
maximumMessageLength = 7;
                                                      std::cout <<
          for (Iteration i{}; i
                                      std::setw(5) <<
                                      m dialogueList[i]-
< m size; ++i)
                                      >GetDate().year << "|";</pre>
                if
                                                      std::cout <<
(strlen(m dialogueList[i]-
                                      std::setw(maximumMessageLength
>GetContent()) >
                                      + 1) << m dialogueList[i]-
                                      >GetContent() << "|";
maximumMessageLength)
                                                      std::cout <<
                                      std::setw(11) <<
                                      m dialogueList[i]->GetOwnerId()
     maximumMessageLength =
strlen(m dialogueList[i]-
                                      << "|";
>GetContent());
                                                      std::cout <<
                                      std::setw(11) <<
                                      m dialogueList[i]-
                                      >GetAdresseeId();
          std::cout <<
                                                      std::cout <<
std::setw((11 + 3 + 3 + 5 +
                                       '\n';
maximumMessageLength + 1 + 11 +
                                                 }
11 + 6 + strlen(GetItemName()))
/ 2) << GetItemName() <<</pre>
                                                 std::cout << '\n';
"\n\n";
                                                 std::cout << RESET <<
                                       ". Reset list\n";
          std::cout <<
std::setw(11) << "ID" << "|";
                                                 std::cout << ADD <<
          std::cout <<
                                       ". Add new dialogue\n";
std::setw(3) << "dd" << "|";
                                                 std::cout << REMOVE</pre>
                                      << ". Delete dialogue\n";
          std::cout <<
std::setw(3) << "mm" << "|";
                                                 std::cout << EDIT <<</pre>
                                       ". Edit dialogue\n";
          std::cout <<
std::setw(5) << "yyyy" << "|";
                                                 std::cout << SORT <<
          std::cout <<
                                       ". Sort list\n";
                                                 std::cout << FILTER</pre>
std::setw(maximumMessageLength
+ 1) << "Message" << "|";
                                      << ". Filter list\n";
```

```
std::cout << ID << ".
                                     Dialogue { id, date, message,
Choose ID\n";
                                     ownerId, adresseeId };
          std::cout << EXIT <<
". Exit\n";
          std::cout << "Input</pre>
                                           fileRead.close();
command: ";
          std::cin >> command;
                                                else if (command ==
          std::cin.ignore();
                                     ADD)
          std::cout << '\n';</pre>
                                                {
                                                     char* message =
                                     new char[LENGTH OF FIELD];
          if (command == RESET)
                                                     std::cout <<
               std::ifstream
                                      "Input message: ";
fileRead (m dialogueDatabase,
std::ios::binary);
                                           std::cin.getline(message,
                                     LENGTH OF FIELD, '\n');
     fileRead.read((char*)&m si
ze, sizeof(size t));
                                                     Id senderId;
                                                     std::cout <<
               m dialogueList =
new Dialogue * [m size] {};
                                      "Input sender ID: ";
               for (Iteration
                                                     std::cin >>
i{}; i < m size; ++i)
                                      senderId;
                                                     Id adresseeId;
                    Id id{};
                                                     std::cout <<
     fileRead.read((char*)&id,
                                      "Input adressee ID: ";
sizeof(Id));
                                                     std::cin >>
                                     adresseeId:
     Entity::Date date{};
                                           std::cin.ignore();
     fileRead.read((char*) &date
                                                     SYSTEMTIME
, sizeof(Entity::Date));
                                     systemTime;
                                           GetLocalTime(&systemTime);
                    char*
message = new
                                                     Dialogue
char[LENGTH OF FIELD] {};
                                     newDialogue =
                                     Dialogue(m idCounter(), {
                                     systemTime.wDay,
     fileRead.read(message,
LENGTH OF FIELD);
                                     systemTime.wMonth,
                                      systemTime.wYear }, message,
                                      senderId, adresseeId);
                    Ιd
ownerId{};
                                                     m dialogueList =
     fileRead.read((char*)&owne
                                     Add (m dialogueList, m size,
rId, sizeof(Id));
                                     newDialogue, m size);
                    Ιd
                                                else if (command ==
adresseeId{};
                                     REMOVE)
     fileRead.read((char*) &adre
                                                     std::cout <<
sseeId, sizeof(Id));
                                      "Input ID: ";
                                                     Id id;
                                                     std::cin >> id;
     m dialogueList[i] = new
```

```
Edit (m dialogueList, m size,
     std::cin.ignore();
                                      id, (void*) newId,
                                      EditMode::ID);
               m dialogueList =
Remove (m dialogueList, m size,
                                                     if
id, m size);
                                      (fieldToChange == 1)
          else if (command ==
                                                           std::cout
                                      << "Input new date\n";
EDIT)
                                                           std::cout
                                      << "Dav:";
               std::cout <<
"Input ID: ";
                                                           unsigned
               Id id;
                                      short day;
               std::cin >> id;
                                                           std::cin >>
                                      day;
     std::cin.ignore();
                                                           std::cout
               std::cout <<
                                      << "Month:";
'\n';
                                                          unsigned
                                      short month;
                                                           std::cin >>
               std::cout <<
                                      month;
"Fields to edit\n";
               std::cout << "0.
                                                           std::cout
ID\n";
                                      << "Year:";
               std::cout << "1.
                                                          unsigned
Date\n";
                                      short year;
               std::cout << "2.
                                                           std::cin >>
Message\n";
                                      year;
               std::cout << "3.
Sender ID\n";
                                           std::cin.ignore();
               std::cout << "4.
Adressee ID\n";
                                           m dialogueList =
                                      Edit (m dialoqueList, m size,
               std::cout <<
"Choose field: ";
                                      id, (void*)new Entity::Date{
               unsigned short
                                      day, month, year },
fieldToChange;
                                      EditMode::DATE);
               std::cin >>
fieldToChange;
                                                     if
                                      (fieldToChange == 2)
     std::cin.ignore();
               std::cout <<
                                                           std::cout
'\n';
                                      << "Input new message: ";
               if
                                      message = new
(fieldToChange == 0)
                                      char[LENGTH OF FIELD] {};
                    std::cout
                                           std::cin.getline(message,
<< "Input new ID: ";
                                      LENGTH OF FIELD, '\n');
                    Id* newId =
                                           m dialogueList =
new Id{};
                    std::cin >>
                                      Edit (m dialogueList, m size,
                                      id, (void*)message,
*newId;
                                      EditMode::CONTENT);
     std::cin.ignore();
                                                     }
                                                     if
     m dialogueList =
                                      (fieldToChange == 3)
```

```
{
                                                      std::cout << "1.
                     std::cout
                                      Date\n";
<< "Input new sender ID: ";
                                                      std::cout << "2.
                                      Message\n";
newSenderId = new Id{};
                                                      std::cout << "3.
                    std::cin >>
                                      Sender ID\n";
                                                      std::cout << "4.
*newSenderId:
                                      Adressee ID\n";
                                                      std::cout <<
     std::cin.ignore();
                                      "Choose field: ";
     m dialogueList =
                                                      unsigned short
Edit (m dialogueList, m size,
                                      field;
id, (void*)newSenderId,
                                                      std::cin >>
EditMode::OWNER ID);
                                      field;
               if
                                           std::cin.ignore();
(fieldToChange == 4)
                                                      SortMode
                                      sortMode = (SortMode) - 1;
                                                      switch (field)
                    std::cout
<< "Input new adressee ID: ";
                                                      {
                     Td*
                                                      case 0:
newAdresseeId = new Id{};
                                                           sortMode =
                    std::cin >>
                                      SortMode::ID;
*newAdresseeId;
                                                           break;
                                                      case 1:
     std::cin.ignore();
                                                           sortMode =
                                      SortMode::DATE;
     m dialoqueList =
                                                           break;
Edit (m dialogueList, m size,
                                                      case 2:
id, (void*)newAdresseeId,
                                                           sortMode =
EditMode::OWNER ID);
                                      SortMode::CONTENT;
                                                           break;
                                                      case 3:
          else if (command ==
                                                           sortMode =
SORT)
                                      SortMode::OWNER ID;
                                                           break:
               std::cout <<
                                                      case 4:
"Orders for sort\n";
                                                           sortMode =
               std::cout << "0.
                                      SortMode::ADRESSEE ID;
Descending\n";
                                                           break;
               std::cout << "1.
Ascending\n";
                                                     m dialogueList =
               std::cout <<
                                      Sort (m dialogueList, m size,
"Choose order: ";
                                      (OrderMode) order, sortMode);
               unsigned short
order;
                                                else if (command ==
               std::cin >>
                                      FILTER)
order;
               std::cout <<
                                                      std::cout <<
                                      "Fields for filter\n";
'\n';
                                                      std::cout << "0.
               std::cout <<
                                      ID\n";
"Fields for sort\n";
                                                      std::cout << "1.
               std::cout << "0.
                                      Date\n";
ID\n";
```

```
std::cout << "2.
                                                           std::cin >>
Message\n";
                                      year;
               std::cout << "3.
Sender ID\n";
                                           std::cin.ignore();
               std::cout << "4.
Adressee ID\n";
                                           m dialogueList =
               std::cout <<
                                      Filter (m dialogueList, m size,
                                      (void*)new Entity::Date{ day,
"Choose field: ";
               unsigned short
                                      month, year },
field;
                                      FilterMode::DATE, m size);
               std::cin >>
field;
                                                     if (field == 2)
     std::cin.ignore();
                                                           std::cout
                                      << "Input part of message: ";
               std::cout <<
'\n';
                                                           char*
                                      message = new
               if (field == 0)
                                      char[LENGTH OF FIELD] {};
                                           std::cin.getline(message,
                     std::cout
<< "Input part of ID: ";
                                      LENGTH OF FIELD, '\n');
                     Id id;
                     std::cin >>
                                           m dialogueList =
id;
                                      Filter (m dialogueList, m size,
                                      (void*) message,
                                      FilterMode::CONTENT, m size);
     std::cin.ignore();
                                                     if (field == 3)
     m dialoqueList =
Filter (m dialogueList, m size,
(void*)&id, FilterMode::ID,
                                                           std::cout
                                      << "Input part of sender ID: ";
m size);
               if (field == 1)
                                      senderId;
                                                           std::cin >>
                     std::cout
                                      senderId;
<< "Input date (if you don't
want to filter by the field,
                                           std::cin.ignore();
input 0) \n";
                                           m dialogueList =
                    std::cout
                                      Filter (m dialogueList, m size,
<< "Dav: ";
                                      (void*)&senderId,
                    unsigned
                                      FilterMode::OWNER ID, m size);
short day;
                    std::cin >>
                                                     if (field == 4)
day;
                    std::cout
<< "Month: ";
                                                           std::cout
                    unsigned
                                      << "Input part of adressee ID:
                                      ";
short month;
                    std::cin >>
                                                           Ιd
                                      adresseeId;
month;
                                                           std::cin >>
                    std::cout
<< "Year: ";
                                      adresseeId;
                    unsigned
                                           std::cin.ignore();
short year;
```

InterestListItem.h

```
m dialogueList =
Filter (m dialoqueList, m size,
                                     #ifndef INTEREST LIST ITEM H
(void*) &adresseeId,
                                     #define INTEREST LIST ITEM H
FilterMode::ADRESSEE ID,
                                     #include "AbstractMenuItem.h"
m size);
                                     #include "Interest.h"
               }
                                     #include "IdCounter.h"
          }
                                     namespace KMK
          if (command == ADD ||
command == REMOVE || command ==
                                          class InterestListItem :
EDIT)
                                     public MenuItem
               std::ofstream
                                          public:
fileWrite (m dialogueDatabase,
std::ios::binary);
                                          InterestListItem(char*
                                     itemName, char*
     fileWrite.write((char*)&m
                                     interestDatabase, char*
size, sizeof(size t));
                                     idCounterDatabase);
               for (Iteration
i{}; i < m size; ++i)
                                               int Run();
                                          private:
     fileWrite.write((char*)new
                                               char*
Id{ m dialogueList[i]->GetId()
                                     m interestDatabase = nullptr;
}, sizeof(Id));
                                               char*
                                     m idCounterDatabase = nullptr;
     fileWrite.write((char*)&m
                                               size t m size{};
dialogueList[i]->GetDate(),
                                               Interest**
sizeof(Entity::Date));
                                     m interestList = nullptr;
                                               IdCounter
     fileWrite.write(m dialogue
                                     m idCounter{};
List[i]->GetContent(),
                                          };
LENGTH OF FIELD);
     fileWrite.write((char*)new
                                          #endif //
Id{ m dialogueList[i]-
                                     !INTEREST LIST ITEM H
>GetOwnerId() }, sizeof(Id));
                                          InterestListItem.cpp
     fileWrite.write((char*)new
Id{ m dialogueList[i]-
                                     #include "InterestListItem.h"
>GetAdresseeId() },
                                     #include "Constants.h"
sizeof(Id));
                                     #include <iostream>
                                     #include <fstream>
                                     #include "Add.h"
     fileWrite.close();
                                     #include <Windows.h>
                                     #include "Remove.h"
                                     #include "Edit.h"
          system("cls");
                                     #include "Sort.h"
                                     #include "Filter.h"
     }
                                     #include <iomanip>
     return 0;
}
                                     using namespace KMK;
```

```
InterestListItem::InterestListI
                                          fileRead.close();
tem(char* itemName, char*
                                     }
interestDatabase, char*
idCounterDatabase) :
                                     int InterestListItem::Run()
MenuItem(itemName)
                                          enum Command
     m interestDatabase = new
char[LENGTH OF FIELD] {};
                                               RESET,
     strcpy s(m interestDatabas
                                               ADD,
e, LENGTH OF FIELD,
                                               REMOVE,
interestDatabase);
                                               EDIT.
     m idCounterDatabase = new
                                               SORT,
char[LENGTH OF FIELD] {};
                                               FILTER,
     strcpy s(m idCounterDataba
                                               ID,
se, LENGTH OF FIELD,
                                               EXIT
idCounterDatabase);
                                          };
    m idCounter = {
                                          unsigned short command =
idCounterDatabase };
                                    0;
     std::ifstream
                                          while (command != EXIT)
fileRead(m interestDatabase,
std::ios::binary);
                                              unsigned short
     fileRead.read((char*)&m si
                                    maximumInterestLength = 8;
                                               for (Iteration i{}; i
ze, sizeof(size t));
     m interestList = new
                                     < m size; ++i)
Interest * [m size] {};
                                               {
     for (Iteration i{}; i <</pre>
                                                    if
m size; ++i)
                                     (strlen(m interestList[i]-
                                     >GetContent()) >
     {
                                     maximumInterestLength)
          Id id{};
                                                   {
     fileRead.read((char*)&id,
sizeof(Id));
                                         maximumInterestLength =
                                     strlen(m interestList[i]-
          Entity::Date date{};
                                     >GetContent());
                                                    }
     fileRead.read((char*)&date
, sizeof(Entity::Date));
                                               std::cout <<
                                     std::setw((11 + 3 + 3 + 5 +
          char* interest = new
char[LENGTH OF FIELD] {};
                                    maximumInterestLength + 1 + 11
                                     + 5 + strlen(GetItemName())) /
                                     2) << GetItemName() << "\n\n";</pre>
     fileRead.read(interest,
LENGTH OF FIELD);
                                               std::cout <<
                                     std::setw(11) << "ID" << "|";
          Id ownerId{};
                                              std::cout <<
                                     std::setw(3) << "dd" << "|";
     fileRead.read((char*)&owne
                                               std::cout <<
rId, sizeof(Id));
                                     std::setw(3) << "mm" << "|";
                                              std::cout <<
                                     std::setw(5) << "yyyy" << "|";
          m interestList[i] =
new Interest{ id, date,
                                              std::cout <<
interest, ownerId };
                                     std::setw(maximumInterestLength
                                     + 1) << "Interest" << "|";
     }
```

```
std::cout <<
                                                std::cin.iqnore();
std::setw(11) << "Owner ID";</pre>
                                                std::cout << '\n';</pre>
          std::cout << '\n';</pre>
                                                if (command == RESET)
          for (Iteration i{}; i
< m size; ++i)
                                                     std::ifstream
                                     fileRead(m interestDatabase,
                                     std::ios::binary);
               std::cout <<
std::setw(11) <<
m interestList[i]->GetId() <<</pre>
                                           fileRead.read((char*)&m si
                                      ze, sizeof(size t));
                                                     m interestList =
               std::cout <<
std::setw(3) <<
                                     new Interest * [m size] {};
m interestList[i]-
                                                     for (Iteration
>GetDate().day << "|";
                                     i{}; i < m size; ++i)
               std::cout <<
                                                          Id id{};
std::setw(3) <<
m interestList[i]-
>GetDate().month << "|";
                                           fileRead.read((char*)&id,
               std::cout <<
                                     sizeof(Id));
std::setw(5) <<
m interestList[i]-
>GetDate().year << "|";</pre>
                                           Entity::Date date{};
               std::cout <<
std::setw(maximumInterestLength
                                           fileRead.read((char*)&date
+ 1) << m interestList[i]-
                                      , sizeof(Entity::Date));
>GetContent() << "|";
               std::cout <<
                                                          char*
std::setw(11) <<
                                     interest = new
m interestList[i]-
                                     char[LENGTH OF FIELD] {};
>GetOwnerId();
               std::cout <<
                                           fileRead.read(interest,
'\n';
                                     LENGTH OF FIELD);
                                                          Ιd
          std::cout << '\n';
                                     ownerId{};
          std::cout << RESET <<
". Reset list\n";
                                           fileRead.read((char*)&owne
          std::cout << ADD <<
                                     rId, sizeof(Id));
". Add new interest\n";
          std::cout << REMOVE</pre>
<< ". Delete interest\n";
                                           m interestList[i] = new
          std::cout << EDIT <<
                                     Interest{ id, date, interest,
". Edit interest\n";
                                     ownerId };
          std::cout << SORT <<
                                                     }
". Sort list\n";
          std::cout << FILTER
                                           fileRead.close();
<< ". Filter list\n";
                                                }
          std::cout << ID << ".
                                                else if (command ==
Choose ID\n";
                                     ADD)
          std::cout << EXIT <<</pre>
". Exit\n";
                                                     char* interest =
          std::cout << "Input
                                     new char[LENGTH OF FIELD];
command: ";
                                                     std::cout <<
          std::cin >> command;
                                      "Input interest: ";
```

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

```
std::cout << "0.
                    std::cout
<< "Year: ";
                                      Descending\n";
                    unsigned
                                                     std::cout << "1.
short year;
                                      Ascending\n";
                    std::cin >>
                                                     std::cout <<
                                      "Choose order: ";
year;
                                                     unsigned short
     std::cin.ignore();
                                      order;
                                                     std::cin >>
     m interestList =
                                      order;
Edit(m interestList, m size,
                                                     std::cout <<
id, (void*)new Entity::Date{
                                      '\n';
day, month, year },
EditMode::DATE);
                                                     std::cout <<
                                      "Fields for sort\n";
               if
                                                     std::cout << "0.
                                      ID\n";
(fieldToChange == 2)
                                                     std::cout << "1.
                                      Date\n";
                    std::cout
<< "Input new interest: ";
                                                     std::cout << "2.
                    char*
                                      Interest\n";
interest = new
                                                     std::cout << "3.
char[LENGTH OF FIELD] {};
                                      Owner ID\n";
                                                     std::cout <<
                                      "Choose field: ";
     std::cin.getline(interest,
LENGTH OF FIELD, '\n');
                                                     unsigned short
                                      field;
     m interestList =
                                                     std::cin >>
Edit(m interestList, m size,
                                      field;
id, (void*)interest,
EditMode::CONTENT);
                                           std::cin.ignore();
                                                     SortMode
               if
                                      sortMode = (SortMode) -1;
                                                     switch (field)
(fieldToChange == 3)
                                                     {
                    std::cout
                                                     case 0:
<< "Input new owner ID: ";
                                                          sortMode =
                                      SortMode::ID;
newOwnerId = new Id{};
                                                          break;
                    std::cin >>
                                                     case 1:
*newOwnerId;
                                                          sortMode =
                                      SortMode::DATE;
     std::cin.ignore();
                                                          break;
                                                     case 2:
                                                          sortMode =
     m interestList =
Edit (m interestList, m size,
                                      SortMode::CONTENT;
id, (void*) newOwnerId,
                                                          break;
EditMode::OWNER ID);
                                                     case 3:
                                                          sortMode =
                                      SortMode::OWNER ID;
          else if (command ==
                                                          break;
SORT)
                                                     m interestList =
                                      Sort(m interestList, m size,
               std::cout <<
"Orders for sort\n";
                                      (OrderMode) order, sortMode);
```

```
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::cout << "1.
                                                          std::cin >>
Date\n";
                                      year;
               std::cout << "2.
Interest\n";
                                           std::cin.ignore();
               std::cout << "3.
Owner ID\n";
                                           m interestList =
               std::cout <<
                                      Filter (m interestList, m size,
"Choose field: ";
                                      (void*)new Entity::Date{ day,
               unsigned short
                                      month, year },
field;
                                      FilterMode::DATE, m size);
               std::cin >>
field;
                                                     if (field == 2)
     std::cin.ignore();
                                                          std::cout
               std::cout <<
                                      << "Input part of interest: ";
'\n';
                                      interest = new
               if (field == 0)
                                      char[LENGTH OF FIELD] {};
                                           std::cin.getline(interest,
                    std::cout
<< "Input part of ID: ";
                                      LENGTH OF FIELD, '\n');
                    Id id;
                    std::cin >>
                                           m interestList =
id;
                                      Filter (m interestList, m size,
                                      (void*)interest,
     std::cin.ignore();
                                      FilterMode::CONTENT, m size);
     m interestList =
                                                     if (field == 3)
Filter (m interestList, m size,
                                                     {
(void*)&id, FilterMode::ID,
                                                          std::cout
                                      << "Input part of owner ID: ";
m size);
                                                          Id ownerId;
                                                          std::cin >>
               if (field == 1)
                                      ownerId;
                    std::cout
<< "Input date (if you don't
                                           std::cin.ignore();
want to filter by the field,
input 0) \n";
                                           m interestList =
                    std::cout
                                      Filter(m interestList, m size,
<< "Day: ";
                                      (void*) & ownerId,
                                      FilterMode::OWNER ID, m size);
                    unsigned
short day;
                    std::cin >>
                                                }
day;
                    std::cout
                                                if (command == ADD ||
<< "Month: ";
                                      command == REMOVE || command ==
                                      EDIT)
```

```
reminderDatabase, char*
               std::ofstream
                                     idCounterDatabase);
fileWrite(m interestDatabase,
std::ios::binary);
                                               int Run();
     fileWrite.write((char*)&m
                                          private:
size, sizeof(size t));
                                               char*
                                     m reminderDatabase = nullptr;
               for (Iteration
i{}; i < m size; ++i)
                                               char*
                                     m idCounterDatabase = nullptr;
                                               size t m size{};
     fileWrite.write((char*)new
                                               Reminder**
                                     m reminderList = nullptr;
Id{ m interestList[i]->GetId()
                                              IdCounter
}, sizeof(Id));
                                     m idCounter{};
     fileWrite.write((char*)&m
                                          } ;
interestList[i]->GetDate(),
sizeof(Entity::Date));
                                          #endif //
                                     !REMINDER LIST ITEM H
     fileWrite.write(m interest
List[i]->GetContent(),
LENGTH OF FIELD);
                                          ReminderListItem.cpp
     fileWrite.write((char*)new
                                     #include "ReminderListItem.h"
                                     #include "Constants.h"
Id{ m interestList[i]-
                                     #include <iostream>
>GetOwnerId() }, sizeof(Id));
                                     #include <fstream>
                                     #include "Add.h"
     fileWrite.close();
                                     #include <Windows.h>
                                     #include "Remove.h"
          }
                                     #include "Edit.h"
                                     #include "Sort.h"
          system("cls");
     }
                                     #include "Filter.h"
                                     #include <iomanip>
     return 0;
                                     using namespace KMK;
     ReminderListItem.h
                                     ReminderListItem::ReminderListI
                                     tem(char* itemName, char*
#ifndef REMINDER LIST ITEM H
                                     reminderDatabase, char*
#define REMINDER LIST ITEM H
                                     idCounterDatabase) :
#include "AbstractMenuItem.h"
                                     MenuItem(itemName)
#include "Reminder.h"
#include "IdCounter.h"
                                          m reminderDatabase = new
                                     char[LENGTH OF FIELD] {};
                                          strcpy s(m reminderDatabas
namespace KMK
                                     e, LENGTH OF FIELD,
                                     reminderDatabase);
     class ReminderListItem :
public MenuItem
                                          m idCounterDatabase = new
                                     char[LENGTH OF FIELD] {};
     {
    public:
                                          strcpy s(m idCounterDataba
                                     se, LENGTH OF FIELD,
     ReminderListItem(char*
                                     idCounterDatabase);
```

itemName, char*

```
m idCounter = {
                                               REMOVE,
idCounterDatabase };
                                               EDIT,
                                               SORT.
     std::ifstream
                                               FILTER,
fileRead(m reminderDatabase,
                                               ID,
std::ios::binary);
                                               EXIT
     fileRead.read((char*)&m si
                                          };
ze, sizeof(size t));
    m reminderList = new
                                          unsigned short command =
Reminder * [m size] {};
                                     0;
     for (Iteration i{}; i <</pre>
m size; ++i)
                                          while (command != EXIT)
     {
                                               unsigned short
          Id id{};
                                     maximumReminderLength = 8;
     fileRead.read((char*)&id,
                                               for (Iteration i{}; i
sizeof(Id));
                                     < m size; ++i)
                                                     if
          Entity::Date date{};
                                     (strlen(m reminderList[i]-
     fileRead.read((char*)&date
                                     >GetContent()) >
, sizeof(Entity::Date));
                                     maximumReminderLength)
                                                    {
          char* reminder = new
char[LENGTH OF FIELD] {};
                                          maximumReminderLength =
                                     strlen(m reminderList[i]-
     fileRead.read(reminder,
                                     >GetContent());
LENGTH OF FIELD);
                                                     }
          Id ownerId{};
                                               std::cout <<
                                     std::setw((11 + 3 + 3 + 5 + 11))
     fileRead.read((char*)&owne
rId, sizeof(Id));
                                     + maximumReminderLength + 1 + 3
                                     + 3 + 5 + 8 +
                                     strlen(GetItemName())) / 2) <<</pre>
          Entity::Date
reminderTime{};
                                     GetItemName() << "\n\n";</pre>
                                               std::cout <<
                                     std::setw(11) << "ID" << "|";
     fileRead.read((char*)&remi
nderTime,
                                               std::cout <<
                                     std::setw(3) << "dd" << "|";
sizeof(Entity::Date));
                                               std::cout <<
          m reminderList[i] =
                                     std::setw(3) << "mm" << "|";
new Reminder{ id, date,
                                               std::cout <<
reminder, ownerId, reminderTime
                                     std::setw(5) << "yyyy" << "|";
};
                                               std::cout <<
                                     std::setw(11) << "Owner ID" <<
                                     "|";
     fileRead.close();
}
                                               std::cout <<
                                     std::setw(maximumReminderLength
                                     + 1) << "Reminder" << ":";
int ReminderListItem::Run()
                                               std::cout <<
                                     std::setw(3) << "dd" << "|";
     enum Command
                                               std::cout <<
          RESET,
                                     std::setw(3) << "mm" << "|";
          ADD,
```

```
std::cout <<
                                                std::cout << EDIT <<
std::setw(5) << "yyyy";
                                      ". Edit reminder\n";
          std::cout << '\n';</pre>
                                                std::cout << SORT <<
                                      ". Sort list\n";
          for (Iteration i{}; i
                                                std::cout << FILTER</pre>
< m size; ++i)
                                      << ". Filter list\n";
                                                std::cout << ID << ".
                                      Choose ID\n";
               std::cout <<
std::setw(11) <<
                                                std::cout << EXIT <<</pre>
                                      ". Exit\n";
m reminderList[i]->GetId() <<</pre>
                                                std::cout << "Input</pre>
               std::cout <<
                                      command: ";
std::setw(3) <<
                                                std::cin >> command;
                                                std::cin.ignore();
m reminderList[i]-
>GetDate().day << "|";
                                                std::cout << '\n';</pre>
               std::cout <<
std::setw(3) <<
                                                if (command == RESET)
m reminderList[i]-
>GetDate().month << "|";
                                                      std::ifstream
               std::cout <<
                                      fileRead (m reminderDatabase,
std::setw(5) <<
                                      std::ios::binary);
m reminderList[i]-
>GetDate().year << "|";
                                           fileRead.read((char*)&m si
               std::cout <<
                                      ze, sizeof(size t));
std::setw(11) <<
                                                      m reminderList =
m reminderList[i]->GetOwnerId()
                                      new Reminder * [m size] {};
<< " | ";
                                                      for (Iteration
               std::cout <<
                                      i{}; i < m size; ++i)
std::setw(maximumReminderLength
+ 1) << m reminderList[i]-
                                                           Id id{};
>GetContent() << ":";
               std::cout <<
                                           fileRead.read((char*)&id,
std::setw(3) <<
                                      sizeof(Id));
m reminderList[i]-
>GetReminderTime().day << "|";
               std::cout <<
                                           Entity::Date date{};
std::setw(3) <<
m reminderList[i]-
                                           fileRead.read((char*) &date
>GetReminderTime().month <<
                                      , sizeof(Entity::Date));
"|";
               std::cout <<
                                                           char*
std::setw(5) <<
                                      reminder = new
m reminderList[i]-
                                      char[LENGTH OF FIELD] {};
>GetReminderTime().year;
               std::cout <<
                                           fileRead.read(reminder,
'\n';
                                      LENGTH OF FIELD);
          }
                                                           Ιd
          std::cout << '\n';
                                      ownerId{};
          std::cout << RESET <<
". Reset list\n";
                                           fileRead.read((char*)&owne
          std::cout << ADD <<
                                      rId, sizeof(Id));
". Add new reminder\n";
          std::cout << REMOVE</pre>
```

<< ". Delete reminder\n";

```
Entity::Date
                                         std::cin.ignore();
reminderTime{};
                                                   SYSTEMTIME
     nderTime,
sizeof(Entity::Date));
                                         GetLocalTime(&systemTime);
                                                  Reminder
                                    newReminder =
    m reminderList[i] = new
                                    Reminder(m idCounter(), {
Reminder{ id, date, reminder,
                                    systemTime.wDay,
ownerId, reminderTime };
                                    systemTime.wMonth,
                                    systemTime.wYear }, reminder,
                                    ownerId, {day, month, year});
     fileRead.close();
                                                  m reminderList =
          else if (command ==
                                    Add (m reminderList, m size,
                                    newReminder, m size);
ADD)
              Id ownerId;
                                              else if (command ==
              std::cout <<
                                    REMOVE)
"Input owner ID: ";
                                              {
              std::cin >>
                                                   std::cout <<
                                    "Input ID: ";
ownerId;
                                                   Id id;
                                                   std::cin >> id;
     std::cin.iqnore();
              char* reminder =
                                        std::cin.ignore();
new char[LENGTH OF FIELD];
              std::cout <<
                                                  m reminderList =
"Input reminder: ";
                                    Remove (m reminderList, m size,
                                    id, m size);
     std::cin.getline(reminder,
LENGTH OF FIELD, '\n');
                                              else if (command ==
                                    EDIT)
              unsigned short
day;
                                                   std::cout <<
              std::cout <<
                                    "Input ID: ";
"Input reminder day: ";
                                                   Id id;
              std::cin >> day;
                                                   std::cin >> id;
              unsigned short
                                        std::cin.ignore();
                                                   std::cout <<
month;
                                    '\n';
              std::cout <<
"Input reminder month: ";
              std::cin >>
                                                   std::cout <<
month;
                                    "Fields to edit\n";
                                                   std::cout << "0.
              unsigned short
                                    ID\n";
                                                   std::cout << "1.
year;
              std::cout <<
                                    Date\n";
"Input reminder year: ";
                                                   std::cout << "2.
              std::cin >>
                                    Owner ID\n";
                                                  std::cout << "3.
year;
                                    Reminder\n";
```

```
std::cout << "4.
Reminder time\n";
                                           m reminderList =
                                      Edit (m reminderList, m size,
               std::cout <<
"Choose field: ";
                                      id, (void*)new Entity::Date{
               unsigned short
                                      day, month, year },
                                      EditMode::DATE);
fieldToChange;
               std::cin >>
                                                     }
fieldToChange;
                                                     if
                                      (fieldToChange == 2)
     std::cin.ignore();
               std::cout <<
                                                           std::cout
'\n';
                                      << "Input new owner ID: ";
                                                           Id*
               if
                                      newOwnerId = new Id{};
(fieldToChange == 0)
                                                           std::cin >>
                                      *newOwnerId;
                    std::cout
<< "Input new ID: ";
                                           std::cin.ignore();
                    Id* newId =
new Id{};
                                           m reminderList =
                    std::cin >>
                                      Edit (m reminderList, m size,
*newId;
                                      id, (void*) newOwnerId,
                                      EditMode::OWNER ID);
     std::cin.ignore();
                                                     }
     m reminderList =
                                                     if
Edit (m reminderList, m size,
                                      (fieldToChange == 3)
id, (void*) newId,
EditMode::ID);
                                                           std::cout
                                      << "Input new reminder: ";
               }
               if
                                                           char*
(fieldToChange == 1)
                                      reminder = new
                                      char[LENGTH OF FIELD] {};
                     std::cout
<< "Input new date\n";
                                           std::cin.getline(reminder,
                    std::cout
                                      LENGTH OF FIELD, '\n');
<< "Day: ";
                    unsigned
                                           m reminderList =
                                      Edit (m reminderList, m size,
short day;
                    std::cin >>
                                      id, (void*) reminder,
                                      EditMode::CONTENT);
day;
                    std::cout
<< "Month: ";
                                                     if
                                      (fieldToChange == 4)
                    unsigned
short month;
                    std::cin >>
                                                           std::cout
month;
                                      << "Input new reminder time\n";
                    std::cout
                                                           std::cout
<< "Year: ";
                                      << "Day: ";
                    unsigned
                                                          unsigned
short year;
                                      short day;
                    std::cin >>
                                                           std::cin >>
year;
                                      day;
                                                           std::cout
     std::cin.ignore();
                                      << "Month: ";
```

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

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

```
std::cin.ignore();
    m reminderList =
Filter(m reminderList, m size,
(void*)new Entity::Date{ day,
month, year },
FilterMode::REMINDER TIME,
m size);
               }
         if (command == ADD ||
command == REMOVE || command ==
EDIT)
               std::ofstream
fileWrite(m reminderDatabase,
std::ios::binary);
     fileWrite.write((char*)&m
size, sizeof(size t));
              for (Iteration
i{}; i < m size; ++i)
     fileWrite.write((char*)new
Id{ m reminderList[i]->GetId()
}, sizeof(Id));
     fileWrite.write((char*)&m
reminderList[i]->GetDate(),
sizeof(Entity::Date));
     fileWrite.write(m reminder
List[i]->GetContent(),
LENGTH OF FIELD);
     fileWrite.write((char*)new
Id{ m reminderList[i]-
>GetOwnerId() }, sizeof(Id));
     fileWrite.write((char*)&m
reminderList[i]-
>GetReminderTime(),
sizeof(Entity::Date));
     fileWrite.close();
          system("cls");
     }
     return 0;
```

ThemeListItem.h

}

```
#ifndef THEME LIST ITEM H
#define THEME LIST ITEM H
#include "AbstractMenuItem.h"
#include "Theme.h"
#include "IdCounter.h"
namespace KMK
    class ThemeListItem :
public MenuItem
     public:
          ThemeListItem(char*
itemName, char* themeDatabase,
char* idCounterDatabase);
         int Run();
    private:
          char* m themeDatabase
= nullptr;
          char*
m idCounterDatabase = nullptr;
          size t m size{};
          Theme** m themeList =
nullptr;
          IdCounter
m idCounter{};
    } ;
#endif // !THEME LIST ITEM H
```

ThemeListItem.cpp

```
#include "ThemeListItem.h"
#include "Constants.h"
#include <iostream>
#include <fstream>
#include "Add.h"
#include "Windows.h>
#include "Remove.h"
#include "Edit.h"
#include "Sort.h"
#include "Filter.h"
#include <iomanip>

using namespace KMK;
ThemeListItem::ThemeListItem(char* itemName, char*
```

```
themeDatabase, char*
                                     int ThemeListItem::Run()
idCounterDatabase) :
                                          enum Command
MenuItem(itemName)
                                          {
     m themeDatabase = new
                                               RESET,
char[LENGTH OF FIELD] {};
                                               ADD,
     strcpy s (m themeDatabase,
                                               REMOVE,
LENGTH OF FIELD,
                                               EDIT,
themeDatabase);
                                               SORT,
     m idCounterDatabase = new
                                               FILTER,
char[LENGTH OF FIELD] { };
                                               ID,
     strcpy s (m idCounterDataba
                                               EXIT
se, LENGTH OF FIELD,
                                          };
idCounterDatabase);
                                          unsigned short command =
     m idCounter = {
                                     0;
idCounterDatabase };
                                          while (command != EXIT)
     std::ifstream
fileRead (m themeDatabase,
                                               unsigned short
std::ios::binary);
                                     maximumThemeLength = 5;
     fileRead.read((char*)&m si
                                               for (Iteration i{}; i
ze, sizeof(size t));
                                     < m size; ++i)
     m themeList = new Theme *
                                                     if
[m size] {};
     for (Iteration i{}; i <</pre>
                                     (strlen(m themeList[i]-
                                     >GetContent()) >
m size; ++i)
                                     maximumThemeLength)
     {
          Id id{};
     fileRead.read((char*)&id,
                                          maximumThemeLength =
                                     strlen(m themeList[i]-
sizeof(Id));
                                     >GetContent());
          Entity::Date date{};
                                               }
     fileRead.read((char*)&date
, sizeof(Entity::Date));
                                               std::cout <<
                                     std::setw((11 + 3 + 3 + 5 +
                                     maximumThemeLength + 1 + 11 + 5
          char* theme = new
char[LENGTH OF FIELD] {};
                                     + strlen(GetItemName())) / 2)
          fileRead.read(theme,
                                     << GetItemName() << "\n\n";
                                               std::cout <<
LENGTH OF FIELD);
                                     std::setw(11) << "ID" << "|";
          Id ownerId{};
                                               std::cout <<
                                     std::setw(3) << "dd" << "|";
     fileRead.read((char*)&owne
                                               std::cout <<
                                     std::setw(3) << "mm" << "|";
rId, sizeof(Id));
                                               std::cout <<
                                     std::setw(5) << "yyyy" << "|";
          m themeList[i] = new
Theme{ id, date, theme, ownerId
                                               std::cout <<
                                     std::setw(maximumThemeLength +
};
                                     1) << "Theme" << "|";
     fileRead.close();
                                               std::cout <<
                                     std::setw(11) << "Owner ID";</pre>
}
                                               std::cout << '\n';
```

```
for (Iteration i{}; i
                                         fileRead.read((char*)&m si
< m size; ++i)
                                     ze, sizeof(size t));
                                                    m themeList =
          {
               std::cout <<
                                     new Theme * [m size] {};
std::setw(11) <<
                                                    for (Iteration
m themeList[i]->GetId() << "|";</pre>
                                     i{}; i < m size; ++i)
               std::cout <<
std::setw(3) << m themeList[i]-</pre>
                                                          Id id{};
>GetDate().day << "|";
               std::cout <<
                                         fileRead.read((char*)&id,
std::setw(3) << m themeList[i] - sizeof(Id));</pre>
>GetDate().month << "|";
               std::cout <<
std::setw(5) << m themeList[i]-</pre>
                                         Entity::Date date{};
>GetDate().year << "|";
               std::cout <<
                                          fileRead.read((char*)&date
std::setw(maximumThemeLength +
                                     , sizeof(Entity::Date));
1) << m themeList[i]-</pre>
>GetContent() << "|";
                                                          char* theme
               std::cout <<
                                     = new char[LENGTH OF FIELD] {};
std::setw(11) <<
m themeList[i]->GetOwnerId();
                                          fileRead.read(theme,
               std::cout <<
                                     LENGTH OF FIELD);
'\n';
          }
                                                          Ιd
                                     ownerId{};
          std::cout << '\n';
          std::cout << RESET <<
                                          fileRead.read((char*)&owne
". Reset list\n";
                                     rId, sizeof(Id));
          std::cout << ADD <<
". Add new theme\n";
          std::cout << REMOVE</pre>
                                          m themeList[i] = new
<< ". Delete theme\n";
                                     Theme{ id, date, theme, ownerId
          std::cout << EDIT <<
                                     } ;
". Edit theme\n";
          std::cout << SORT <<
". Sort list\n";
                                          fileRead.close();
          std::cout << FILTER</pre>
                                               }
<< ". Filter list\n";
                                               else if (command ==
          std::cout << ID << ".
                                     ADD)
Choose ID\n";
          std::cout << EXIT <<</pre>
                                                    char* theme =
". Exit\n";
                                     new char[LENGTH OF FIELD];
          std::cout << "Input</pre>
                                                    std::cout <<
command: ";
                                     "Input theme: ";
          std::cin >> command;
          std::cin.ignore();
                                          std::cin.getline(theme,
          std::cout << '\n';
                                     LENGTH OF FIELD, '\n');
          if (command == RESET)
                                                    Id ownerId;
                                                    std::cout <<
               std::ifstream
                                     "Input owner ID: ";
                                                    std::cin >>
fileRead(m themeDatabase,
std::ios::binary);
                                     ownerId;
```

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

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

```
if (field == 2)
               unsigned short
field;
               std::cin >>
                                                          std::cout
                                     << "Input part of theme: ";
field;
                                                          char* theme
     std::cin.ignore();
                                     = new char[LENGTH OF FIELD] { };
               std::cout <<
'\n';
                                           std::cin.getline(theme,
                                     LENGTH OF FIELD, '\n');
               if (field == 0)
                                                          m themeList
                                     = Filter(m themeList, m size,
                                      (void*) theme,
                    std::cout
<< "Input part of ID: ";
                                     FilterMode::CONTENT, m size);
                    Id id;
                    std::cin >>
                                                     if (field == 3)
id:
                                                          std::cout
                                      << "Input part of owner ID: ";
     std::cin.ignore();
                                                          Id ownerId;
                    m themeList
                                                          std::cin >>
= Filter(m themeList, m size,
(void*)&id, FilterMode::ID,
                                     ownerId;
m size);
                                           std::cin.ignore();
               if (field == 1)
                                                          m themeList
                                     = Filter(m themeList, m size,
                                      (void*) & ownerId,
                    std::cout
<< "Input date (if you don't
                                     FilterMode::OWNER ID, m size);
want to filter by the field,
input 0) \n";
                    std::cout
<< "Day: ";
                                                if (command == ADD ||
                    unsigned
                                      command == REMOVE || command ==
short day;
                                     EDIT)
                    std::cin >>
                                                     std::ofstream
day;
                                      fileWrite (m themeDatabase,
                    std::cout
<< "Month: ";
                                      std::ios::binary);
                    unsigned
short month;
                                           fileWrite.write((char*)&m
                    std::cin >>
                                      size, sizeof(size t));
                                                     for (Iteration
month;
                    std::cout
                                      i{}; i < m size; ++i)
<< "Year: ";
                    unsigned
                                           fileWrite.write((char*)new
short year;
                    std::cin >>
                                      Id{ m themeList[i]->GetId() },
                                      sizeof(Id));
year;
     std::cin.ignore();
                                           fileWrite.write((char*)&m
                    m themeList
                                     themeList[i]->GetDate(),
                                     sizeof(Entity::Date));
= Filter(m themeList, m size,
(void*) new Entity::Date{ day,
month, year },
                                           fileWrite.write(m themeLis
FilterMode::DATE, m size);
                                     t[i]->GetContent(),
                                     LENGTH OF FIELD);
```

```
#include <fstream>
     fileWrite.write((char*)new
                                    #include "Add.h"
                                    #include <Windows.h>
Id{ m themeList[i]-
>GetOwnerId() }, sizeof(Id));
                                    #include "Remove.h"
                                    #include "Edit.h"
                                    #include "Sort.h"
                                    #include "Filter.h"
     fileWrite.close();
                                    #include <iomanip>
          system("cls");
                                    using namespace KMK;
     }
                                    UserListItem::UserListItem(char
                                    * itemName, char* userDatabase,
     return 0;
                                    char* idCounterDatabase) :
}
                                    MenuItem(itemName)
    UserListItem.h
                                         m userDatabase = new
                                    char[LENGTH OF FIELD] {};
#ifndef USER LIST ITEM H
#define USER LIST ITEM H
                                         strcpy s (m userDatabase,
#include "AbstractMenuItem.h"
                                    LENGTH OF FIELD, userDatabase);
#include "User.h"
                                         m idCounterDatabase = new
#include "IdCounter.h"
                                    char[LENGTH OF FIELD] {};
                                         strcpy s(m idCounterDataba
                                    se, LENGTH OF FIELD,
namespace KMK
                                    idCounterDatabase);
    class UserListItem :
public MenuItem
                                         m idCounter = {
                                    idCounterDatabase };
    {
    public:
    UserListItem(char*
                                         std::ifstream
itemName, char* userDatabase,
                                    fileRead (m userDatabase,
char* idCounterDatabase);
                                    std::ios::binary);
                                         fileRead.read((char*)&m si
         int Run();
                                    ze, sizeof(size t));
                                         m userList = new User *
                                     [m size] {};
    private:
         char* m_userDatabase
                                         for (Iteration i{}; i <</pre>
= nullptr;
                                    m size; ++i)
         char*
                                       {
m idCounterDatabase = nullptr;
                                              Id id{};
          size t m size{};
                                       fileRead.read((char*)&id,
         User** m userList =
                                    sizeof(Id));
nullptr;
         IdCounter
m idCounter{};
                                              Entity::Date date{};
    } ;
                                         fileRead.read((char*)&date
                                     , sizeof(Entity::Date));
#endif // !USER LIST ITEM H
                                              char* name = new
    UserListItem.cpp
                                    char[LENGTH OF FIELD] {};
                                              fileRead.read(name,
                                    LENGTH OF FIELD);
#include "UserListItem.h"
#include "Constants.h"
#include <iostream>
```

```
char* login = new
                                                     }
char[LENGTH OF FIELD] {};
                                                     if
          fileRead.read(login,
                                      (strlen(m userList[i]-
                                      >GetLogin()) >
LENGTH OF FIELD);
                                      maximumLoginLength)
          char* password = new
char[LENGTH OF FIELD] {};
                                           maximumLoginLength =
     fileRead.read(password,
                                      strlen(m userList[i]-
LENGTH OF FIELD);
                                      >GetLogin());
                                                     if
          m userList[i] = new
User{ id, date, name, login,
                                      (strlen(m userList[i]-
                                      >GetPassword()) >
password };
                                      maximumPasswordLength)
     fileRead.close();
}
                                           maximumPasswordLength =
                                      strlen(m userList[i]-
int UserListItem::Run()
                                      >GetPassword());
     enum Command
                                                }
     {
          RESET,
                                                std::cout <<
          ADD,
                                      std::setw((11 + 3 + 3 + 5 +
          REMOVE,
                                      maximumNameLength + 1 +
          EDIT,
                                      maximumLoginLength + 1 +
          SORT,
          FILTER,
                                      maximumPasswordLength + 1 + 6 +
                                      strlen(GetItemName())) / 2) <<</pre>
          ID,
                                      GetItemName() << "\n\n";</pre>
          EXIT
                                                std::cout <<
     };
                                      std::setw(11) << "ID" << "|";
                                                std::cout <<</pre>
     unsigned short command =
0;
                                      std::setw(3) << "dd" << "|";
                                                std::cout <<
     while (command != EXIT)
                                      std::setw(3) << "mm" << "|";
                                                std::cout <<
          unsigned short
                                      std::setw(5) << "yyyy" << "|";
maximumNameLength = 4;
                                                std::cout <<
          unsigned short
                                      std::setw(maximumNameLength +
                                      1) << "Name" << "|";
maximumLoginLength = 5;
          unsigned short
                                                std::cout <<
maximumPasswordLength = 8;
                                      std::setw(maximumLoginLength +
                                      1) << "Login" << "|";
          for (Iteration i{}; i
< m size; ++i)
                                                std::cout <<
          {
                                      std::setw(maximumPasswordLength
               if
                                      + 1) << "Password";
(strlen(m userList[i]-
                                                std::cout << '\n';</pre>
>GetName()) >
maximumNameLength)
                                                for (Iteration i{}; i
                                      < m size; ++i)
               {
     maximumNameLength =
                                                     std::cout <<
strlen(m userList[i]-
                                      std::setw(11) << m userList[i]-</pre>
>GetName());
                                      >GetId() << "|";
```

```
std::cout <<
std::setw(3) << m userList[i]-</pre>
                                           fileRead.read((char*)&m si
>GetDate().day << "|";
                                      ze, sizeof(size t));
               std::cout <<
                                                     m userList = new
std::setw(3) << m userList[i]-</pre>
                                      User * [m size] {};
>GetDate().month << "|";
                                                     for (Iteration
               std::cout <<
                                      i{}; i < m size; ++i)
std::setw(5) << m userList[i]-</pre>
>GetDate().year << "|";
                                                           Id id{};
               std::cout <<
std::setw(maximumNameLength +
                                           fileRead.read((char*)&id,
1) << m userList[i]->GetName()
                                      sizeof(Id));
<< "|";
               std::cout <<
std::setw(maximumLoginLength +
                                           Entity::Date date{};
1) << m userList[i]->GetLogin()
<< "|";
                                           fileRead.read((char*) &date
               std::cout <<
                                      , sizeof(Entity::Date));
std::setw(maximumPasswordLength
+ 1) << m userList[i]-
                                                           char* name
>GetPassword();
                                      = new char[LENGTH OF FIELD] {};
               std::cout <<
'\n';
                                           fileRead.read(name,
                                      LENGTH OF FIELD);
          std::cout << '\n';
                                                           char* login
          std::cout << RESET <<</pre>
                                      = new char[LENGTH OF FIELD] {};
". Reset list\n";
                                           fileRead.read(login,
          std::cout << ADD <<
". Add new user\n";
                                      LENGTH OF FIELD);
          std::cout << REMOVE</pre>
<< ". Delete user\n";
                                                           char*
          std::cout << EDIT <<</pre>
                                      password = new
". Edit user\n";
                                      char[LENGTH OF FIELD] {};
          std::cout << SORT <<
". Sort list\n";
                                           fileRead.read(password,
          std::cout << FILTER</pre>
                                      LENGTH OF FIELD);
<< ". Filter list\n";
          std::cout << ID << ".
Choose ID\n";
                                           m userList[i] = new User{
          std::cout << EXIT <<
                                      id, date, name, login, password
". Exit\n";
                                      };
          std::cout << "Input</pre>
                                                     }
command: ";
          std::cin >> command;
                                           fileRead.close();
          std::cin.iqnore();
          std::cout << '\n';
                                                else if (command ==
                                      ADD)
          if (command == RESET)
                                                     char* name = new
               std::ifstream
                                      char[LENGTH OF FIELD];
fileRead(m userDatabase,
                                                     std::cout <<
std::ios::binary);
                                      "Input name: ";
```

```
std::cin >> id;
     std::cin.getline(name,
LENGTH OF FIELD, '\n');
                                          std::cin.ignore();
                                                    std::cout <<
               char* login =
                                     '\n';
new char[LENGTH OF FIELD];
               std::cout <<
                                                     std::cout <<
                                     "Fields to edit\n";
"Input login: ";
                                                    std::cout << "0.
     std::cin.getline(login,
                                     ID\n";
LENGTH OF FIELD, '\n');
                                                     std::cout << "1.
                                     Date\n";
               char* password =
                                                     std::cout << "2.
new char[LENGTH OF FIELD];
                                     Name\n";
               std::cout <<
                                                     std::cout << "3.
"Input password: ";
                                     Login\n";
                                                     std::cout << "4.
     std::cin.getline(password,
                                     Password\n";
LENGTH OF FIELD, '\n');
                                                     std::cout <<
                                     "Choose field: ";
               SYSTEMTIME
                                                     unsigned short
systemTime;
                                     fieldToChange;
                                                     std::cin >>
     GetLocalTime(&systemTime);
                                     fieldToChange;
               User newUser =
User(m idCounter(), {
                                          std::cin.ignore();
systemTime.wDay,
                                                     std::cout <<
systemTime.wMonth,
                                     '\n';
systemTime.wYear }, name,
login, password);
                                                     if
                                      (fieldToChange == 0)
               m userList =
Add(m userList, m size,
                                                          std::cout
newUser, m size);
                                     << "Input new ID: ";
          }
                                                          Id* newId =
          else if (command ==
                                     new Id{};
REMOVE)
                                                          std::cin >>
                                     *newId;
               std::cout <<
"Input ID: ";
                                          std::cin.ignore();
               Id id;
                                                         m userList
               std::cin >> id;
                                     = Edit(m userList, m size, id,
                                     (void*)newId, EditMode::ID);
     std::cin.ignore();
                                                     if
               m userList =
                                     (fieldToChange == 1)
Remove (m userList, m size, id,
m size);
                                                          std::cout
                                     << "Input new date\n";
          else if (command ==
                                                          std::cout
                                     << "Day: ";
EDIT)
                                                          unsigned
               std::cout <<
                                     short day;
"Input ID: ";
                                                          std::cin >>
               Id id;
                                     day;
```

```
std::cout
<< "Month: ";
                                           std::cin.getline(password,
                                      LENGTH OF FIELD, '\n');
                    unsigned
short month;
                                                          m userList
                    std::cin >>
                                      = Edit(m userList, m size, id,
                                      (void*)password,
month;
                                      EditMode::PASSWORD);
                    std::cout
<< "Year: ";
                    unsigned
short year;
                                                else if (command ==
                     std::cin >>
                                      SORT)
year;
                                                     std::cout <<
                                      "Orders for sort\n";
     std::cin.ignore();
                    m userList
                                                     std::cout << "0.
= Edit(m userList, m size, id,
                                      Descending\n";
(void*)new Entity::Date{ day,
                                                     std::cout << "1.
month, year }, EditMode::DATE);
                                      Ascending\n";
                                                     std::cout <<
               if
                                      "Choose order: ";
(fieldToChange == 2)
                                                     unsigned short
                                      order;
                                                     std::cin >>
                    std::cout
<< "Input new name: ";
                                      order;
                     char* name
                                                     std::cout <<
= new char[LENGTH OF FIELD] {};
                                      '\n';
     std::cin.getline(name,
                                                     std::cout <<
LENGTH OF FIELD, '\n');
                                      "Fields for sort\n";
                                                     std::cout << "0.
                    m userList
= Edit(m userList, m size, id,
                                      ID\n";
(void*) name, EditMode::NAME);
                                                     std::cout << "1.
                                      Date\n";
               }
               if
                                                     std::cout << "2.
                                      Name\n";
(fieldToChange == 3)
                                                     std::cout << "3.
                    std::cout
                                      Login\n";
<< "Input new login: ";
                                                     std::cout << "4.
                                      Password\n";
                     char* login
= new char[LENGTH OF FIELD] {};
                                                     std::cout <<
                                      "Choose field: ";
     std::cin.getline(login,
                                                     unsigned short
LENGTH OF FIELD, '\n');
                                      field;
                                                     std::cin >>
                    m userList
= Edit(m userList, m size, id,
                                      field;
(void*)login, EditMode::LOGIN);
                                           std::cin.ignore();
               if
                                                     SortMode
(fieldToChange == 4)
                                      sortMode = (SortMode) -1;
                                                     switch (field)
                    std::cout
                                                     {
<< "Input new password: ";
                                                     case 0:
                    char*
                                                          sortMode =
password = new
                                      SortMode::ID;
char[LENGTH OF FIELD] { };
                                                          break;
```

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

```
std::cout
<< "Input part of login: ";
                                          fileWrite.write(m userList
                    char* login
                                     [i]->GetLogin(),
= new char[LENGTH OF FIELD] {};
                                     LENGTH OF FIELD);
     std::cin.getline(login,
                                          fileWrite.write(m userList
LENGTH OF FIELD, '\n');
                                     [i]->GetPassword(),
                                     LENGTH OF FIELD);
                    m userList
= Filter(m userList, m size,
(void*)login,
FilterMode::LOGIN, m size);
                                          fileWrite.close();
                                               }
               if (field == 4)
                                               system("cls");
                    std::cout
<< "Input part of password: ";
                    char*
                                          return 0;
password = new
                                     }
char[LENGTH OF FIELD] {};
                                          CopyList.h
     std::cin.getline(password,
LENGTH OF FIELD, '\n');
                                     #ifndef COPY LIST H
                                     #define COPY LIST H
                    m userList
= Filter(m userList, m_size,
                                     #include "Dialogue.h"
(void*) password,
                                     #include "Interest.h"
FilterMode::PASSWORD, m size);
                                     #include "Reminder.h"
                                     #include "Theme.h"
          }
                                     #include "User.h"
          if (command == ADD ||
                                     namespace KMK
command == REMOVE || command ==
EDIT)
                                          Dialogue**
                                     CopyList(Dialogue** list,
               std::ofstream
                                     size t size);
fileWrite (m userDatabase,
                                          Interest**
std::ios::binary);
                                     CopyList(Interest** list,
                                     size t size);
                                          Reminder**
     fileWrite.write((char*)&m
size, sizeof(size t));
                                     CopyList(Reminder** list,
               for (Iteration
                                     size t size);
                                          Theme** CopyList(Theme**
i{}; i < m size; ++i)
                                     list, size t size);
                                          User** CopyList(User**
     fileWrite.write((char*)new
                                     list, size t size);
Id{ m userList[i]->GetId() },
sizeof(Id));
                                     #endif // !COPY LIST H
     fileWrite.write((char*)&m
                                          CopyList.cpp
userList[i]->GetDate(),
sizeof(Entity::Date));
                                     #include "CopyList.h"
     fileWrite.write(m userList
[i] -> GetName(),
                                     using namespace KMK;
LENGTH OF FIELD);
```

```
Dialogue**
KMK::CopyList(Dialogue** list,
                                           return temp;
size t size)
    Dialogue** temp = new
                                      User** KMK::CopyList(User**
Dialogue * [size];
                                      list, size t size)
     for (Iteration i{}; i <</pre>
size; ++i)
                                           User** temp = new User *
     {
                                      [size];
                                           for (Iteration i{}; i <</pre>
          temp[i] = new
Dialogue{ *list[i] };
                                      size; ++i)
                                            {
                                                temp[i] = new User{
     return temp;
                                      *list[i] };
}
Interest**
                                           return temp;
KMK::CopyList(Interest** list,
size t size)
                                           Add.h
     Interest** temp = new
Interest * [size];
                                      #ifndef ADD_H
    for (Iteration i{}; i <</pre>
                                      #define ADD H
size; ++i)
                                      #include "Dialogue.h"
                                      #include "Interest.h"
     {
                                      #include "Reminder.h"
          temp[i] = new
Interest{ *list[i] };
                                      #include "Theme.h"
                                      #include "User.h"
     return temp;
}
                                      namespace KMK
Reminder**
                                           Dialogue** Add(Dialogue**
KMK::CopyList(Reminder** list,
                                      dialogs, size t size, Dialogue
                                      newElement, size_t& newSize);
size_t size)
                                           Interest** Add(Interest**
     Reminder** temp = new
                                      interests, size t size,
Reminder * [size];
                                      Interest newElement, size t&
    for (Iteration i{}; i <</pre>
                                      newSize);
                                           Reminder** Add (Reminder**
size; ++i)
                                      reminders, size t size,
          temp[i] = new
                                      Reminder newElement, size t&
Reminder{ *list[i] };
                                      newSize);
                                           Theme** Add(Theme**
                                      themes, size t size, Theme
     return temp;
                                      newElement, size_t& newSize);
User** Add(User** users,
}
Theme** KMK::CopyList(Theme**
                                      size_t size, User newElement,
list, size t size)
                                      size t& newSize);
     Theme** temp = new Theme *
[size];
                                      #endif // !ADD H
    for (Iteration i{}; i <</pre>
size; ++i)
                                           Add.cpp
          temp[i] = new Theme{
                                      #include "Add.h"
                                      #include "CopyList.h"
*list[i] };
```

```
using namespace KMK;
                                     Reminder{ *reminders[i] };
Dialogue** KMK::Add(Dialogue**
                                          temp[size] = new Reminder{
dialogs, size t size, Dialogue
                                     newElement };
newElement, size t &newSize)
                                          ++newSize;
     Dialogue** temp = new
                                          return temp;
Dialogue * [size + 1];
                                     }
                                     Theme** KMK::Add(Theme**
     for (Iteration i{}; i <</pre>
size; ++i)
                                     themes, size t size, Theme
                                     newElement, size t& newSize)
         temp[i] = new
Dialogue{ *dialogs[i] };
                                          Theme** temp = new Theme *
                                      [size + 1];
     temp[size] = new Dialogue{
newElement };
                                          for (Iteration i{}; i <</pre>
                                     size; ++i)
    ++newSize;
                                           {
    return temp;
                                                temp[i] = new Theme{
}
                                      *themes[i] };
Interest** KMK::Add(Interest**
                                          temp[size] = new Theme{
interests, size t size,
                                     newElement };
Interest newElement, size t&
newSize)
                                          ++newSize;
                                          return temp;
     Interest** temp = new
                                     }
Interest * [size + 1];
                                     User** KMK::Add(User** users,
     for (Iteration i{}; i <</pre>
                                     size t size, User newElement,
size; ++i)
                                     size t& newSize)
     {
                                          User** temp = new User *
          temp[i] = new
Interest{ *interests[i] };
                                     [size + 1];
     temp[size] = new Interest{
                                          for (Iteration i{}; i <</pre>
newElement };
                                     size; ++i)
                                           {
    ++newSize;
                                               temp[i] = new User{
    return temp;
                                      *users[i] };
}
                                          temp[size] = new User{
Reminder** KMK::Add(Reminder**
                                     newElement };
reminders, size t size,
Reminder newElement, size t&
                                          ++newSize;
newSize)
                                          return temp;
                                     }
     Reminder** temp = new
Reminder * [size + 1];
                                          Edit.h
     for (Iteration i{}; i <</pre>
                                     #ifndef EDIT H
size; ++i)
                                     #define EDIT H
                                     #include "Dialogue.h"
     {
```

temp[i] = new

```
#include "Interest.h"
                                          Dialogue** temp =
#include "Reminder.h"
                                     CopyList(dialogs, size);
#include "Theme.h"
#include "User.h"
                                          unsigned short
                                     dialogueNumber = 0;
namespace KMK
                                          while (dialogueNumber <</pre>
                                     size)
     enum class EditMode
                                                if
          ID,
                                      (temp[dialogueNumber]->GetId()
                                      != idToEdit)
          DATE,
          CONTENT,
          OWNER ID,
          ADRESSEE ID,
                                          ++dialogueNumber;
          REMINDER TIME,
                                                }
                                                else
          NAME,
          LOGIN,
                                                {
          PASSWORD
                                                     break;
     };
                                                }
                                           }
     Dialogue** Edit(Dialogue**
dialogs, size t size, Id
                                          if (dialogueNumber < size)</pre>
idToEdit, void* newField,
EditMode mode);
                                                if (mode ==
     Interest** Edit(Interest**
                                     EditMode::ID)
interests, size t size, Id
idToEdit, void* newField,
EditMode mode);
                                          temp[dialogueNumber] -
     Reminder** Edit(Reminder**
                                     >SetId(*(Id*)newField);
reminders, size t size, Id
                                                }
idToEdit, void* newField,
                                                if (mode ==
EditMode mode);
                                     EditMode::DATE)
     Theme** Edit(Theme**
themes, size t size, Id
idToEdit, void* newField,
                                          temp[dialogueNumber]-
EditMode mode);
                                     >SetDate(*(Entity::Date*)newFie
     User** Edit(User** users,
size t size, Id idToEdit, void*
newField, EditMode mode);
                                                if (mode ==
                                     EditMode::CONTENT)
}
                                                {
#endif // !EDIT H
                                          temp[dialogueNumber]-
     Edit.cpp
                                     >SetContent((char*)newField);
#include "Edit.h"
                                                if (mode ==
#include "CopyList.h"
                                     EditMode::OWNER ID)
                                                {
using namespace KMK;
                                           temp[dialogueNumber]-
Dialogue** KMK::Edit(Dialogue**
                                     >SetOwnerId(*(Id*)newField);
dialogs, size t size, Id
idToEdit, void* newField,
                                                if (mode ==
EditMode mode)
                                     EditMode::ADRESSEE ID)
                                                {
```

```
temp[dialogueNumber]-
                                          temp[interestNumber] -
>SetAdresseeId(*(Id*)newField); >SetContent((char*)newField);
                                               if (mode ==
                                     EditMode::OWNER ID)
    return temp;
                                          temp[interestNumber] -
Interest** KMK::Edit(Interest**
                                     >SetOwnerId(*(Id*)newField);
interests, size t size, Id
idToEdit, void* newField,
EditMode mode)
                                         return temp;
     Interest** temp =
                                     }
CopyList(interests, size);
                                     Reminder** KMK::Edit(Reminder**
                                     reminders, size t size, Id
    unsigned short
                                     idToEdit, void* newField,
interestNumber = 0;
    while (interestNumber <</pre>
                                     EditMode mode)
size)
                                          Reminder** temp =
     {
         if
                                     CopyList(reminders, size);
(temp[interestNumber]->GetId()
!= idToEdit)
                                          unsigned short
                                     reminderNumber = 0;
                                          while (reminderNumber <</pre>
     ++interestNumber;
                                     size)
          }
          else
                                               if
                                     (temp[reminderNumber]->GetId()
                                     != idToEdit)
               break;
     }
                                          ++reminderNumber;
     if (interestNumber < size)</pre>
                                              }
                                               else
         if (mode ==
EditMode::ID)
                                                    break;
          {
                                          }
     temp[interestNumber]-
>SetId(*(Id*)newField);
                                          if (reminderNumber < size)</pre>
         if (mode ==
                                               if (mode ==
                                     EditMode::ID)
EditMode::DATE)
     temp[interestNumber] -
                                          temp[reminderNumber] -
>SetDate(*(Entity::Date*)newFie
                                     >SetId(*(Id*)newField);
ld);
                                               if (mode ==
         if (mode ==
                                     EditMode::DATE)
EditMode::CONTENT)
```

```
temp[reminderNumber]-
                                          if (themeNumber < size)</pre>
>SetDate(*(Entity::Date*)newFie
                                               if (mode ==
ld);
                                     EditMode::ID)
          if (mode ==
EditMode::CONTENT)
                                          temp[themeNumber]-
                                     >SetId(*(Id*)newField);
    temp[reminderNumber]-
                                               if (mode ==
>SetContent((char*)newField);
                                     EditMode::DATE)
         if (mode ==
EditMode::OWNER ID)
                                          temp[themeNumber]-
                                     >SetDate(*(Entity::Date*)newFie
     temp[reminderNumber] -
                                     ld);
>SetOwnerId(*(Id*)newField);
                                               if (mode ==
          if (mode ==
                                     EditMode::CONTENT)
EditMode::REMINDER TIME)
          {
                                          temp[themeNumber]-
     temp[reminderNumber]-
                                     >SetContent((char*)newField);
>SetReminderTime(*(Entity::Date
*) newField);
                                               if (mode ==
                                     EditMode::OWNER ID)
     }
                                               {
     return temp;
                                          temp[themeNumber]-
                                     >SetOwnerId(*(Id*)newField);
}
Theme** KMK::Edit(Theme**
                                          }
themes, size t size, Id
idToEdit, void* newField,
                                          return temp;
EditMode mode)
                                     User** KMK::Edit(User** users,
    Theme** temp =  
CopyList(themes, size);
                                     size t size, Id idToEdit, void*
                                     newField, EditMode mode)
     unsigned short themeNumber
= 0;
                                          User** temp =
     while (themeNumber < size)</pre>
                                     CopyList(users, size);
          if
                                          unsigned short userNumber
(temp[themeNumber]->GetId() !=
                                     = 0;
idToEdit)
                                          while (userNumber < size)</pre>
                                               if (temp[userNumber]-
               ++themeNumber;
                                     >GetId() != idToEdit)
          else
                                                    ++userNumber;
               break;
                                               else
     }
```

```
break;
                                     #include "User.h"
          }
     }
                                     namespace KMK
     if (userNumber < size)</pre>
                                          enum class FilterMode
          if (mode ==
                                                ID,
EditMode::ID)
                                                DATE,
                                                CONTENT,
                                                OWNER ID,
     temp[userNumber]-
                                                ADRESSEE ID,
>SetId(*(Id*)newField);
                                                REMINDER TIME,
                                                NAME,
          if (mode ==
                                                LOGIN,
EditMode::DATE)
                                                PASSWORD
                                           } ;
     temp[userNumber]-
                                          Dialogue**
                                     Filter(Dialogue** dialogs,
>SetDate(*(Entity::Date*)newFie
                                     size t size, void*
ld);
                                     fieldForSearch, FilterMode
          if (mode ==
                                     mode, size t& newSize);
EditMode::NAME)
                                          Interest**
                                     Filter(Interest** interests,
                                     size t size, void*
                                     fieldForSearch, FilterMode
     temp[userNumber]-
>SetName((char*)newField);
                                     mode, size t& newSize);
                                          Reminder**
          if (mode ==
                                     Filter (Reminder** reminders,
                                     size t size, void*
EditMode::LOGIN)
                                     fieldForSearch, FilterMode
          {
                                     mode, size t& newSize);
                                          Theme** Filter(Theme**
     temp[userNumber]-
>SetLogin((char*)newField);
                                     themes, size t size, void*
                                     fieldForSearch, FilterMode
          }
          if (mode ==
                                     mode, size t& newSize);
                                          User** Filter(User**
EditMode::PASSWORD)
                                     users, size t size, void*
          {
                                     fieldForSearch, FilterMode
                                     mode, size t& newSize);
     temp[userNumber]-
>SetPassword((char*)newField);
     }
                                     #endif // !FILTER H
                                          Filter.cpp
     return temp;
}
                                     #include "Filter.h"
     Filter.h
                                     #include "Constants.h"
                                     #include <cmath>
                                     #include <iostream>
#ifndef FILTER H
                                     #include "CopyList.h"
#define FILTER H
#include "Dialogue.h"
#include "Interest.h"
                                     using namespace KMK;
#include "Reminder.h"
#include "Theme.h"
```

```
void RemoveUnwanted(Entity**
&entities, Id* fields, Id*
                                          ++numberOfEntity;
fieldForSearch, size t &size)
                                                }
                                           }
     unsigned long int tens =
10;
                                          delete[] entities;
     unsigned short
                                          entities =
numberOfDigits = 1;
                                     filteredEntities;
     while (*fieldForSearch /
                                          size = newSize;
tens != 0)
     {
          tens *= 10;
                                     void RemoveUnwanted(Entity**
                                     &entities, char** fields, char*
          ++numberOfDigits;
                                     fieldForSearch, size t &size)
     }
     bool* indexes = new
                                          unsigned short
bool[size] {};
                                     fieldForSearchLength =
     size t newSize = 0;
                                     strlen(fieldForSearch);
     for (Iteration i{}; i <</pre>
MAXIMUM NUMBER OF DIGITS IN ID
                                          bool* indexes = new
- numberOfDigits + 1; ++i)
                                     bool[size] {};
                                          int newSize = 0;
     {
                                          for (Iteration i{}; i <</pre>
          for (Iteration j{); j
                                     size; ++i)
< size; j++)
                                           {
               if
                                                for (Iteration j{); j
(*fieldForSearch == (fields[j]
                                     < strlen(fields[i]) -
/ (int)pow(10, i)) % tens)
                                     fieldForSearchLength + 1; ++j)
                    if
                                                     char* temp = new
(indexes[j] != true)
                                     char[fieldForSearchLength + 1]
                                     { };
                                                    for (Iteration
                                     k{}; k < fieldForSearchLength;</pre>
     indexes[j] = true;
                                     ++k)
     ++newSize:
                                                          temp[k] =
                                     fields[i][j + k];
               }
          }
     }
                                                     if (strcmp(temp,
     Entity** filteredEntities
                                     fieldForSearch) == 0)
= new Entity * [newSize] {};
     Iteration numberOfEntity =
                                                          indexes[i]
0;
                                     = true;
     for (Iteration i{}; i <</pre>
                                                          ++newSize;
size; ++i)
                                                          break;
                                                     }
     {
          if (indexes[i] ==
                                                }
                                           }
true)
          {
                                          Entity** newList = new
     filteredEntities[numberOfE
                                     Entity * [newSize] {};
ntity] = entities[i];
                                          unsigned short
                                     numberOfElement = 0;
```

```
for (Iteration i{}; i <</pre>
                                                if (indexes[i] ==
size; ++i)
                                     true)
          if (indexes[i] ==
                                          newList[numberOfElement] =
true)
                                     entities[i];
     newList[numberOfElement] =
                                          ++numberOfElement;
entities[i];
                                               }
     ++numberOfElement;
                                          delete[] entities;
          }
                                          entities = newList;
                                           size = newSize;
     delete[] entities;
                                     }
     entities = newList;
     size = newSize;
                                     void FilterEntities(Entity**
                                     &entities, size t &size, void*
}
                                     fieldForSearch, FilterMode
void RemoveUnwanted(Entity**&
                                     mode)
entities, Entity::Date* fields,
Entity::Date* fieldForSearch,
                                          if (mode ==
size t& size)
                                     FilterMode::ID)
     bool* indexes = new
                                                Id* fields = new
bool[size] {};
                                     Id[size]{};
     unsigned short newSize =
                                               for (Iteration i{}; i
                                     < size; ++i)
0:
     for (Iteration i{}; i <</pre>
size; ++i)
                                                     fields[i] =
                                     entities[i]->GetId();
          if ((fields[i].day ==
fieldForSearch->day ||
fieldForSearch->day == 0) &&
                                          RemoveUnwanted(entities,
               (fields[i].month
                                     fields, (Id*) fieldForSearch,
== fieldForSearch->month ||
                                     size);
fieldForSearch->month == 0) &&
                                          else if (mode ==
               (fields[i].year
== fieldForSearch->year ||
                                     FilterMode::DATE)
fieldForSearch->year == 0))
                                                Entity::Date* fields
                                     = new Entity::Date[size]{};
               indexes[i] =
                                                for (Iteration i{}; i
true;
                                     < size; ++i)
               ++newSize;
     }
                                                     fields[i] =
                                     entities[i]->GetDate();
     Entity** newList = new
                                                }
Entity * [newSize] {};
     unsigned short
                                          RemoveUnwanted(entities,
numberOfElement = 0;
     for (Iteration i{}; i <</pre>
                                     (Entity::Date*)fieldForSearch,
size; ++i)
                                     size);
     {
                                     }
```

```
temp, newSize, fieldForSearch,
void
                                     mode);
FilterTextEntities(TextEntity**
& entities, size t& size, void*
                                          else if (mode ==
fieldForSearch, FilterMode
                                     FilterMode::CONTENT || mode ==
mode)
                                     FilterMode::OWNER ID)
     if (mode ==
FilterMode::CONTENT)
                                          FilterTextEntities((TextEn
                                     tity**&) temp, newSize,
                                     fieldForSearch, mode);
          char** fields = new
char* [size] {};
          for (Iteration i{}; i
                                          else if (mode ==
                                     FilterMode::ADRESSEE ID)
< size; ++i)
                                               Id* fields = new
               fields[i] =
entities[i]->GetContent();
                                     Id[size]{};
                                               for (Iteration i{}; i
                                     < size; ++i)
     RemoveUnwanted((Entity**&)
entities, fields,
                                                    fields[i] =
(char*) fieldForSearch, size);
                                     dialogs[i]->GetAdresseeId();
                                               }
     else if (mode ==
FilterMode::OWNER ID)
                                          RemoveUnwanted((Entity**&)
                                     temp, fields,
          Id* fields = new
                                     (Id*) fieldForSearch, newSize);
Id[size]{};
          for (Iteration i{}; i
< size; ++i)
                                          return temp;
               fields[i] =
                                     Interest**
entities[i]->GetOwnerId();
                                     KMK::Filter(Interest**
                                     interests, size t size, void*
                                     fieldForSearch, FilterMode
     RemoveUnwanted((Entity**&)
entities, fields,
                                     mode, size t &newSize)
(Id*)fieldForSearch, size);
                                          Interest** temp =
                                     CopyList(interests, size);
                                          newSize = size;
Dialogue**
                                          if (mode == FilterMode::ID
KMK::Filter(Dialogue** dialogs,
                                     | | mode == FilterMode::DATE)
size t size, void*
fieldForSearch, FilterMode
mode, size t& newSize)
                                          FilterEntities((Entity**&)
                                     temp, newSize, fieldForSearch,
     Dialoque** temp =
                                     mode);
CopyList(dialogs, size);
                                          else if (mode ==
     newSize = size;
     if (mode == FilterMode::ID
                                     FilterMode::CONTENT || mode ==
|| mode == FilterMode::DATE)
                                     FilterMode::OWNER ID)
     FilterEntities((Entity**&)
                                          FilterTextEntities((TextEn
```

```
Theme** KMK::Filter(Theme**
tity**&) temp, newSize,
fieldForSearch, mode);
                                     themes, size t size, void*
                                     fieldForSearch, FilterMode
                                     mode, size t& newSize)
     return temp;
                                          Theme** temp =
}
                                     CopyList(themes, size);
Reminder**
                                          newSize = size;
KMK::Filter(Reminder**
                                          if (mode == FilterMode::ID
reminders, size t size, void*
                                     | | mode == FilterMode::DATE)
fieldForSearch, FilterMode
mode, size t& newSize)
                                          FilterEntities((Entity**&)
    Reminder** temp =
                                     temp, newSize, fieldForSearch,
CopyList(reminders, size);
                                    mode);
    newSize = size;
     if (mode == FilterMode::ID
                                          else if (mode ==
|| mode == FilterMode::DATE)
                                     FilterMode::CONTENT || mode ==
                                     FilterMode::OWNER ID)
     FilterEntities((Entity**&)
temp, newSize, fieldForSearch,
                                          FilterTextEntities((TextEn
mode);
                                     tity**&) temp, newSize,
                                     fieldForSearch, mode);
     else if (mode ==
                                          }
FilterMode::CONTENT || mode ==
FilterMode::OWNER ID)
                                          return temp;
                                     }
     {
     FilterTextEntities((TextEn
                                     User** KMK::Filter(User**
tity**&) temp, newSize,
                                     users, size t size, void*
fieldForSearch, mode);
                                     fieldForSearch, FilterMode
                                     mode, size t& newSize)
     else if (mode ==
FilterMode::REMINDER TIME)
                                          User** temp =
                                     CopyList(users, size);
          Entity::Date* fields
                                          newSize = size;
= new Entity::Date[size]{};
                                          if (mode == FilterMode::ID
          for (Iteration i{}; i
                                     || mode == FilterMode::DATE)
< size; ++i)
               fields[i] =
                                          FilterEntities((Entity**&)
reminders[i]-
                                     temp, newSize, fieldForSearch,
>GetReminderTime();
                                     mode);
                                          }
                                          else if (mode ==
                                     FilterMode::NAME || mode ==
    RemoveUnwanted((Entity**&)
temp, fields,
                                     FilterMode::LOGIN || mode ==
(Entity::Date*)fieldForSearch,
                                     FilterMode::PASSWORD)
newSize);
                                              char** fields = new
     }
                                     char* [size] {};
                                              if (mode ==
     return temp;
                                     FilterMode::NAME)
}
                                              {
```

```
for (Iteration
                                     size t size, Id idToRemove,
i{}; i < size; ++i)
                                     size t& newSize);
                                          Reminder**
                    fields[i] =
                                     Remove (Reminder** reminders,
                                     size t size, Id idToRemove,
users[i]->GetName();
                                     size t& newSize);
          }
                                          Theme** Remove(Theme**
                                     themes, size t size, Id
          if (mode ==
FilterMode::LOGIN)
                                     idToRemove, size t& newSize);
                                          User** Remove(User**
               for (Iteration
                                     users, size t size, Id
i{}; i < size; ++i)
                                     idToRemove, size t& newSize);
                    fields[i] =
users[i]->GetLogin();
                                     #endif // !REMOVE H
                                          Remove.cpp
          if (mode ==
FilterMode::PASSWORD)
                                     #include "Remove.h"
               for (Iteration
                                     using namespace KMK;
i{}; i < size; ++i)
                                     Dialogue**
                    fields[i] =
                                     KMK::Remove(Dialogue** dialogs,
                                     size t size, Id idToRemove,
users[i]->GetPassword();
                                     size t& newSize)
               }
                                          bool found = false;
     RemoveUnwanted((Entity**&)
                                          for (Iteration i{}; i <</pre>
temp, fields,
                                     size; ++i)
(char*) fieldForSearch,
newSize);
                                               if (idToRemove ==
     }
                                     dialogs[i]->GetId())
     return temp;
                                                    found = true;
}
                                                }
                                          }
     Remove.h
                                          if (found == true)
#ifndef REMOVE H
#define REMOVE H
                                               Dialogue** temp = new
#include "Dialogue.h"
                                     Dialogue * [size - 1];
#include "Interest.h"
                                               unsigned short
#include "Reminder.h"
                                     tempElementNumber = 0;
#include "Theme.h"
                                               for (Iteration i{}; i
                                     < size; ++i)
#include "User.h"
namespace KMK
                                                    if (dialogs[i]-
                                     >GetId() != idToRemove)
     Dialogue**
Remove(Dialogue** dialogs,
size t size, Id idToRemove,
                                          temp[tempElementNumber] =
                                     new Dialogue{ *dialogs[i] };
size t& newSize);
     Interest**
Remove(Interest** interests,
                                          ++tempElementNumber;
```

```
}
                                   Reminder**
         --newSize;
         return temp;
                                   KMK::Remove(Reminder**
                                   reminders, size t size, Id
    }
    else
                                   idToRemove, size t& newSize)
                                        bool found = false;
         return dialogs;
                                        for (Iteration i{}; i <</pre>
}
                                   size; ++i)
Interest**
                                             if (idToRemove ==
KMK::Remove(Interest**
                                   reminders[i]->GetId())
interests, size t size, Id
idToRemove, size t& newSize)
                                                  found = true;
    bool found = false;
    for (Iteration i{}; i <</pre>
size; ++i)
                                        if (found == true)
         if (idToRemove ==
                                             Reminder** temp = new
                                   Reminder * [size - 1];
interests[i]->GetId())
                                            unsigned short
                                   tempElementNumber = 0;
              found = true;
                                             for (Iteration i{}; i
                                   < size; ++i)
    }
    if (found == true)
                                                  if
                                   (reminders[i]->GetId() !=
         Interest** temp = new
                                   idToRemove)
Interest * [size - 1];
         unsigned short
tempElementNumber = 0;
                                        temp[tempElementNumber] =
         < size; ++i)
                                        ++tempElementNumber;
              if
(interests[i]->GetId() !=
idToRemove)
                                             --newSize;
                                             return temp;
    temp[tempElementNumber] =
                                        }
new Interest{ *interests[i] };
    ++tempElementNumber;
                                             return reminders;
         --newSize;
                                   Theme** KMK::Remove(Theme**
                                   themes, size t size, Id
         return temp;
                                   idToRemove, size t& newSize)
    }
    else
                                        bool found = false;
         return interests;
```

```
for (Iteration i{}; i <</pre>
                                       if (found == true)
size; ++i)
                                            User** temp = new
        if (idToRemove ==
                                  User * [size - 1];
themes[i]->GetId())
                                            unsigned short
                                  tempElementNumber = 0;
              found = true;
                                            for (Iteration i{}; i
                                   < size; ++i)
    }
                                                 if (users[i]-
    if (found == true)
                                >GetId() != idToRemove)
         Theme** temp = new
Theme * [size - 1];
                                       temp[tempElementNumber] =
         tempElementNumber = 0;
         for (Iteration i{}; i
                                     ++tempElementNumber;
< size; ++i)
              if (themes[i]-
>GetId() != idToRemove)
                                            --newSize;
                                            return temp;
                                        }
                                       else
    temp[tempElementNumber] =
new Theme{ *themes[i] };
                                        {
                                            return users;
    ++tempElementNumber;
              }
                                   }
                                       Sort.h
         --newSize;
                                   #ifndef SORT H
         return temp;
    }
                                   #define SORT H
    else
                                   #include "Dialogue.h"
                                   #include "Interest.h"
         return themes;
                                   #include "Reminder.h"
                                   #include "Theme.h"
                                   #include "User.h"
}
                                  namespace KMK
User** KMK::Remove(User**
                                       enum class OrderMode
users, size t size, Id
idToRemove, size t& newSize)
                                            DESCENDING,
    bool found = false;
                                            ASCENDING
    for (Iteration i{}; i <</pre>
                                       } ;
size; ++i)
                                       enum class SortMode
         if (idToRemove ==
users[i]->GetId())
                                            ID,
                                            DATE,
              found = true;
                                            CONTENT,
                                            OWNER ID,
    }
                                            ADRESSEE ID,
                                            REMINDER TIME,
```

```
void JustSwap(Entity::Date&
          NAME,
          LOGIN,
                                     firstField, Entity::Date&
          PASSWORD
                                     secondField)
     };
                                          Entity::Date temp =
     Dialogue** Sort(Dialogue**
                                     firstField;
dialogs, size t size, OrderMode
                                          firstField = secondField;
order, SortMode mode);
                                          secondField = temp;
     Interest** Sort(Interest**
                                     }
interests, size t size,
OrderMode order, SortMode
                                     void JustSwap(char*
mode);
                                     &firstField, char*
     Reminder** Sort (Reminder**
                                     &secondField)
reminders, size t size,
OrderMode order, SortMode
                                          char* temp = firstField;
mode);
                                          firstField = secondField;
     Theme** Sort (Theme**
                                          secondField = temp;
themes, size t size, OrderMode
                                     }
order, SortMode mode);
     User** Sort(User** users,
                                     void Swap (Entity * & firstEntity,
                                     Entity* &secondEntity,
size t size, OrderMode order,
SortMode mode);
                                          Id& firstField, Id&
                                     secondField, OrderMode order)
#endif // !SORT H
                                          if (order ==
                                     OrderMode::DESCENDING)
     Sort.cpp
                                               if (firstField <</pre>
                                     secondField)
#include "Sort.h"
#include <iostream>
                                               {
#include "CopyList.h"
                                          JustSwap (firstEntity,
                                     secondEntity);
using namespace KMK;
                                          JustSwap(firstField,
void JustSwap(Entity*
&firstEntity, Entity*
                                     secondField);
&secondEntity)
    Entity* temp =
                                          else if (order ==
firstEntity;
                                     OrderMode::ASCENDING)
     firstEntity =
secondEntity;
                                               if (firstField >
     secondEntity = temp;
                                     secondField)
}
void JustSwap(Id& firstField,
                                          JustSwap(firstEntity,
Id& secondField)
                                     secondEntity);
     Id temp = firstField;
                                          JustSwap(firstField,
                                     secondField);
     firstField = secondField;
     secondField = temp;
                                                }
                                          }
                                     }
```

```
void Swap(Entity* &firstEntity,
                                               if (firstField.year >
Entity* &secondEntity,
                                     secondField.year)
     Entity::Date& firstField,
Entity::Date& secondField,
OrderMode order)
                                          JustSwap (firstEntity,
                                     secondEntity);
     if (order ==
OrderMode::DESCENDING)
                                          JustSwap(firstField,
                                     secondField);
         if (firstField.year <</pre>
secondField.year)
                                               else if
                                     (firstField.year ==
                                     secondField.year)
     JustSwap (firstEntity,
secondEntity);
                                     (firstField.month >
                                     secondField.month)
     JustSwap(firstField,
secondField);
         else if
                                          JustSwap (firstEntity,
(firstField.year ==
                                     secondEntity);
secondField.year)
                                          JustSwap(firstField,
               if
                                     secondField);
(firstField.month <</pre>
secondField.month)
                                                     else if
                                     (firstField.month ==
                                     secondField.month)
     JustSwap (firstEntity,
secondEntity);
                                                          if
                                     (firstField.day >
                                     secondField.day)
     JustSwap(firstField,
secondField);
                                                          {
                                          JustSwap (firstEntity,
               else if
(firstField.month ==
                                     secondEntity);
secondField.month)
                                          JustSwap(firstField,
                    if
                                     secondField);
(firstField.day <</pre>
secondField.day)
                                                }
                                          }
     JustSwap (firstEntity,
secondEntity);
                                     void Swap(Entity* &firstEntity,
     JustSwap(firstField,
                                     Entity* &secondEntity,
secondField);
                                          char* &firstField, char*
                                     &secondField, OrderMode order)
                                          Iteration letter = 0;
                                          while (firstField[letter]
     else if (order ==
                                     == secondField[letter] &&
OrderMode::ASCENDING)
                                               letter <</pre>
                                     strlen(firstField) - 1 &&
```

```
letter < strlen(secondField) -</pre>
                                      entities[i + 1], fields[i],
1)
                                      fields[i + 1], order);
          ++letter;
                                                --upperBorder;
                                                for (Iteration i =
     if (order ==
                                      upperBorder; i > bottomBorder;
OrderMode::DESCENDING)
                                      --i)
          if
                                                     Swap (entities [i
                                      - 1], entities[i], fields[i -
(firstField[letter] <</pre>
secondField[letter])
                                      1], fields[i], order);
                                                ++bottomBorder;
     JustSwap (firstEntity,
                                           }
secondEntity);
                                      }
     JustSwap (firstField,
                                      void SortByShaker(Entity**
                                      entities, Entity::Date* fields,
secondField);
                                      size t size, OrderMode order)
          }
     }
     else if (order ==
                                           short bottomBorder = 0;
OrderMode::ASCENDING)
                                           short upperBorder = size -
                                      1;
                                           while (upperBorder -
          if
(firstField[letter] >
                                      bottomBorder > 0)
secondField[letter])
                                           {
                                                for (Iteration i =
          {
                                      bottomBorder; i < upperBorder;</pre>
     JustSwap (firstEntity,
                                      ++i)
secondEntity);
     JustSwap(firstField,
                                           Swap (entities [i],
secondField);
                                      entities[i + 1], fields[i],
                                      fields[i + 1], order);
          }
     }
}
                                                --upperBorder;
void SortByShaker(Entity**
                                                for (Iteration i =
entities, Id* fields, size t
                                      upperBorder; i > bottomBorder;
size, OrderMode order)
                                      --i)
     short bottomBorder = 0;
                                                     Swap (entities [i
     short upperBorder = size -
                                      - 1], entities[i], fields[i -
                                      1], fields[i], order);
1;
     while (upperBorder -
bottomBorder > 0)
                                                ++bottomBorder;
                                           }
          for (Iteration i =
bottomBorder; i < upperBorder;</pre>
++i)
                                      void SortByShaker(Entity**
                                      entities, char** fields, size t
                                      size, OrderMode order)
     Swap(entities[i],
                                           short bottomBorder = 0;
```

```
short upperBorder = size -
                                                    fields[i] =
1;
                                     entities[i]->GetDate();
     while (upperBorder -
bottomBorder > 0)
                                          SortByShaker(entities,
          for (Iteration i =
                                     fields, size, order);
bottomBorder; i < upperBorder;</pre>
++i)
                                    void
     Swap (entities [i],
                                     SortTextEntities(TextEntity**
entities[i + 1], fields[i],
                                     entities, size t size,
fields[i + 1], order);
                                     OrderMode order, SortMode mode)
          --upperBorder;
                                          if (mode ==
                                     SortMode::CONTENT)
          for (Iteration i =
upperBorder; i > bottomBorder;
                                              char** fields = new
--i)
                                     char*[size] {};
                                              for (Iteration i = 0;
                                     i < size; ++i)
               Swap (entities [i
- 1], entities[i], fields[i -
1], fields[i], order);
                                                   fields[i] =
                                     entities[i]->GetContent();
          ++bottomBorder;
                                          SortByShaker((Entity**)ent
}
                                     ities, fields, size, order);
void SortEntities(Entity**
entities, size t size,
                                          else if (mode ==
OrderMode order, SortMode mode)
                                     SortMode::OWNER ID)
     if (mode == SortMode::ID)
                                              Id* fields = new
                                     Id[size] {};
          Id* fields = new
                                              for (Iteration i = 0;
                                     i < size; ++i)
Id[size] {};
         for (Iteration i = 0;
i < size; ++i)
                                                    fields[i] =
                                     entities[i]->GetOwnerId();
              fields[i] =
entities[i]->GetId();
                                          SortByShaker((Entity**)ent
                                     ities, fields, size, order);
     SortByShaker(entities,
fields, size, order);
                                     Dialogue** KMK::Sort(Dialogue**
     else if (mode ==
SortMode::DATE)
                                     dialogs, size t size, OrderMode
                                     order, SortMode mode)
          Entity::Date* fields
= new Entity::Date[size] {};
                                         if (size > 1)
         for (Iteration i = 0;
i < size; ++i)
                                               Dialogue** temp =
                                     CopyList(dialogs, size);
```

```
if (mode ==
SortMode::ID || mode ==
                                         SortEntities((Entity**)tem
SortMode::DATE)
                                    p, size, order, mode);
         {
                                              else if (mode ==
     SortEntities((Entity**)tem
                                    SortMode::CONTENT || mode ==
p, size, order, mode);
                                    SortMode::OWNER ID)
                                              {
         else if (mode ==
SortMode::CONTENT || mode ==
                                         SortTextEntities((TextEnti
SortMode::OWNER ID)
                                    ty**) temp, size, order, mode);
     SortTextEntities((TextEnti
                                             return temp;
ty**) temp, size, order, mode);
                                         }
                                         else
         else if (mode ==
                                         {
SortMode::ADRESSEE ID)
                                             return interests;
              Id* fields = new
Id[size]{};
                                    Reminder** KMK::Sort(Reminder**
              for (Iteration i
= 0; i < size; ++i)
                                    reminders, size t size,
                                    OrderMode order, SortMode mode)
                  fields[i] =
temp[i]->GetAdresseeId();
                                         if (size > 1)
                                             Reminder** temp =
     SortByShaker((Entity**)tem
                                    CopyList(reminders, size);
p, fields, size, order);
                                             if (mode ==
                                    SortMode::ID || mode ==
                                    SortMode::DATE)
         return temp;
     }
    else
                                         SortEntities((Entity**)tem
         return dialogs;
                                   p, size, order, mode);
}
                                              else if (mode ==
                                    SortMode::CONTENT || mode ==
Interest** KMK::Sort(Interest**
                                    SortMode::OWNER ID)
interests, size t size,
OrderMode order, SortMode mode)
                                         SortTextEntities((TextEnti
    if (size > 1)
                                    ty**) temp, size, order, mode);
         Interest** temp =
                                             else if (mode ==
CopyList(interests, size);
                                    SortMode::REMINDER TIME)
         if (mode ==
                                                  Entity::Date*
SortMode::ID || mode ==
                                    fields = new
SortMode::DATE)
                                    Entity::Date[size]{};
         {
                                                  for (Iteration i
                                    = 0; i < size; ++i)
```

```
fields[i] =
temp[i]->GetReminderTime();
                                              User** temp =
               }
                                    CopyList(users, size);
     SortByShaker((Entity**)tem
                                              if (mode ==
p, fields, size, order);
                                    SortMode::ID || mode ==
                                    SortMode::DATE)
         return temp;
     }
                                         SortEntities((Entity**)tem
                                    p, size, order, mode);
     else
        return reminders;
                                              else if (mode ==
                                    SortMode::NAME)
}
                                                  char** fields =
Theme** KMK::Sort(Theme**
                                    new char* [size] {};
themes, size t size, OrderMode
                                                   for (Iteration i
order, SortMode mode)
                                    = 0; i < size; ++i)
     if (size > 1)
                                                        fields[i] =
                                    temp[i]->GetName();
         Theme** temp =
CopyList(themes, size);
                                         SortByShaker((Entity**)tem
         if (mode ==
                                    p, fields, size, order);
SortMode::ID || mode ==
SortMode::DATE)
                                              else if (mode ==
                                    SortMode::LOGIN)
     SortEntities((Entity**)tem
                                                   char** fields =
p, size, order, mode);
                                    new char* [size] {};
                                                   for (Iteration i
          else if (mode ==
                                    = 0; i < size; ++i)
SortMode::CONTENT || mode ==
SortMode::OWNER ID)
                                                        fields[i] =
                                    temp[i]->GetLogin();
     SortTextEntities((TextEnti
ty**) temp, size, order, mode);
                                         SortByShaker((Entity**)tem
                                    p, fields, size, order);
                                              }
                                              else if (mode ==
         return temp;
                                    SortMode::PASSWORD)
     }
     else
                                                   char** fields =
         return themes;
                                    new char* [size] {};
     }
                                                   for (Iteration i
                                    = 0; i < size; ++i)
}
User** KMK::Sort(User** users,
                                                        fields[i] =
size t size, OrderMode order,
                                   temp[i]->GetPassword();
SortMode mode)
```

if (size > 1)

```
MenuItem** m items =
     SortByShaker((Entity**)tem
                                     nullptr;
p, fields, size, order);
                                         } ;
                                          std::ostream&
                                     operator << (std::ostream& out,
          return temp;
     }
                                     const Menu& menu);
     else
                                          std::istream&
                                     operator>>(std::istream& in,
     {
          return users;
                                     const Menu& menu);
     }
}
                                     #endif // !MENU H
     Menu.h
                                          Menu.cpp
#ifndef MENU H
                                     #include "Menu.h"
#define MENU H
#include "AbstractMenuItem.h"
                                     #include "Constants.h"
                                     #include <iostream>
#include <iostream>
                                     #include "TypeDefinitions.h"
namespace KMK
{
                                     using namespace KMK;
    class Menu
                                     Menu::Menu(char* title,
     public:
                                     MenuItem** items, size t count)
         Menu(char* title,
MenuItem** items, size t
                                          m title = new
                                     char[LENGTH OF FIELD];
count);
                                          strcpy s(m title,
          int GetSelect();
                                     LENGTH OF FIELD, title);
          bool GetRunning();
                                          m items = items;
          char* GetTitle();
                                          m count = count;
          size t GetCount();
          MenuItem**
GetItems();
                                     int Menu::GetSelect() { return
                                     m select; }
          void Print() const;
                                     bool Menu::GetRunning() {
          int RunCommand()
                                     return m running; }
                                     char* Menu::GetTitle() { return
const;
                                     m title; }
          friend std::ostream&
                                     size t Menu::GetCount() {
operator<<(std::ostream& out,
                                     return m count; }
const Menu& menu);
                                     MenuItem** Menu::GetItems() {
          friend std::istream&
                                     return m items; }
operator>>(std::istream& in,
const Menu& menu);
                                     void Menu::Print() const
                                          std::cout << '\t' <<
    private:
                                     m title << "\n\n";</pre>
          int m select = -1;
          bool m running =
                                          for (Iteration i = 0; i <
                                     m count; i++)
false;
          char* m_title =
                                          {
                                               std::cout << i << ".
nullptr;
                                     ";
          size t m count{};
```

```
m items[i]-
                                     #include "DialogueListItem.h"
>PrintItemName();
                                     #include "InterestListItem.h"
                                     #include "ReminderListItem.h"
          std::cout << '\n';</pre>
                                     #include "ThemeListItem.h"
     }
                                     #include <iostream>
}
int Menu::RunCommand() const
                                     using namespace KMK;
     Print();
                                     int main()
     std::cout << m count << ".</pre>
                                          UserListItem users =
     std::cout << "Enter
                                     UserListItem((char*)"User
command: ";
                                     list", (char*)"User
     unsigned short command;
                                     database.dat", (char*)"User
     std::cin >> command;
                                     IDs.dat");
     system("cls");
                                           DialogueListItem dialogs =
     if (command != m count)
                                     DialogueListItem((char*)"Dialog
                                     ue list", (char*) "Dialogue
                                     database.dat", (char*) "Dialoque
          return
                                     IDs.dat");
m items[command]->Run();
                                           InterestListItem interests
     else
                                     InterestListItem((char*)"Intere
                                     st list", (char*) "Interst
          return 1;
                                     database.dat", (char*)"Interst
     }
                                     IDs.dat");
}
                                          ReminderListItem reminders
std::ostream&
KMK::operator<<(std::ostream&</pre>
                                     ReminderListItem((char*) "Remind
out, const Menu& menu)
                                     er list", (char*) "Reminder
                                     database.dat", (char*) "Reminder
                                     IDs.dat");
    menu.Print();
                                          ThemeListItem themes =
                                     ThemeListItem((char*)"Theme
    return out;
}
                                     list", (char*) "Theme
                                     database.dat", (char*) "Theme
std::istream&
                                     IDs.dat");
KMK::operator>>(std::istream&
in, const Menu& menu)
                                          Menu menu =
                                     Menu((char*) "Chat Bot", new
{
     int code = 0;
                                     MenuItem*[5] { &users,
     while (code == 0)
                                     &dialogs, &interests,
                                     &reminders, &themes }, 5);
          code =
menu.RunCommand();
                                          std::cin >> menu;
          system("cls");
     }
                                          return 0;
                                     }
     return in;
}
     Main.cpp
#include "Menu.h"
#include "UserListItem.h"
```

Демонстрация:

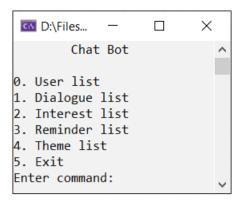


Рисунок 2. Главное меню

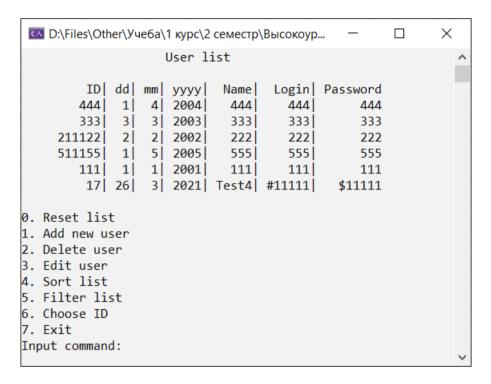


Рисунок 3. База данных пользователей

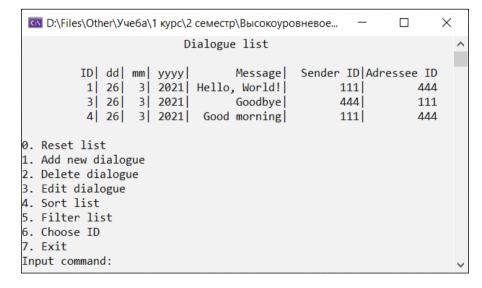


Рисунок 4. База данных диалогов

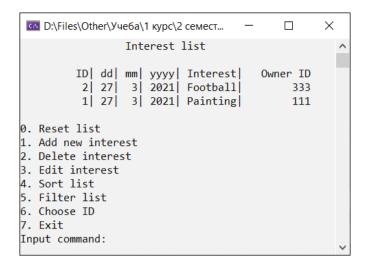


Рисунок 5. База данных интересов

```
Reminder list
       ID | dd | mm | yyyy |
                         Owner ID
                                     Reminder: dd | mm | yyyy
        1 27
              3 2021
                           111 | Do homework: 28
                                                 3
                                                    2021
                             333 Go to doctor: 1 4 2021
       25 | 27 | 3 | 2021 |
                             444
       26 | 27 | 3 | 2021 |
                                      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

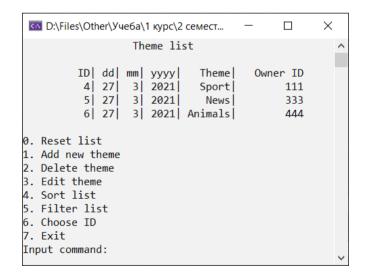


Рисунок 7. База данных тем

Вывод: в ходе выполнения лабораторной работы были получены практические навыки перегрузки операторов "<<", ">>>" и "()" через дружественные функции, создания функторов, работы с базами данных, сортировки и фильтрации массивов данных, создания, удаления и изменения данных.