

МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Пояснительная записка к курсовой работе

по дисциплине: «Программирования»

на тему: «Разработка системы управления базой данных на языке С++»

Выполнил студент группы М3О-211Б-11

Плоцкий Богдан Андреевич

Проверила: Дмитриева Елена Анатольевна

Москва 2022 г.

Содержание

[Задание 4](#_Toc122348654)

[Схема классов 5](#_Toc122348655)

[Описание классов 6](#_Toc122348656)

[MENU 6](#_Toc122348657)

[COMMAND 7](#_Toc122348658)

[DATABASEMANAGER 8](#_Toc122348659)

[MUSICSTUFF 9](#_Toc122348660)

[FIELDSTYPE 10](#_Toc122348661)

[MY\_LIST 11](#_Toc122348662)

[NODE 12](#_Toc122348663)

[Код программы 13](#_Toc122348664)

[command.cpp 13](#_Toc122348665)

[command.h 15](#_Toc122348666)

[music\_stuff.cpp 16](#_Toc122348667)

[music\_stuff.h 20](#_Toc122348668)

[music\_stuff\_list.cpp 22](#_Toc122348669)

[music\_stuff\_list.h 27](#_Toc122348670)

[menu.cpp 29](#_Toc122348671)

[menu.h 42](#_Toc122348672)

[support\_func.hpp 44](#_Toc122348673)

[constants.h 48](#_Toc122348674)

[kursovaya-2022.cpp 51](#_Toc122348675)

[Результат работы 52](#_Toc122348676)

[Файлы 52](#_Toc122348677)

[Скриншоты работы программы 53](#_Toc122348678)

[Вывод 55](#_Toc122348679)

# Задание

Разработать систему управления базой данных на языке С++.

Тематики базы данных выбираются согласно варианту.

Для хранения данных использовать текстовый файл.

Разработать структуры данных для хранения данных и функции для их обработки.

Разработать функции:

1. для ввода данных,
2. выборки значения по заданному условию,
3. вывода данных на экран,
4. удаления,
5. записи новых данных,
6. сортировки по определенному пользователем значению.

Возможно наличие нескольких узлов списков с одинаковыми значениями полей. Обеспечить выдачу запросов по любым полям данных. Реализовать систему меню.

**Схема организации данных -** Линейные двусвязные списки.

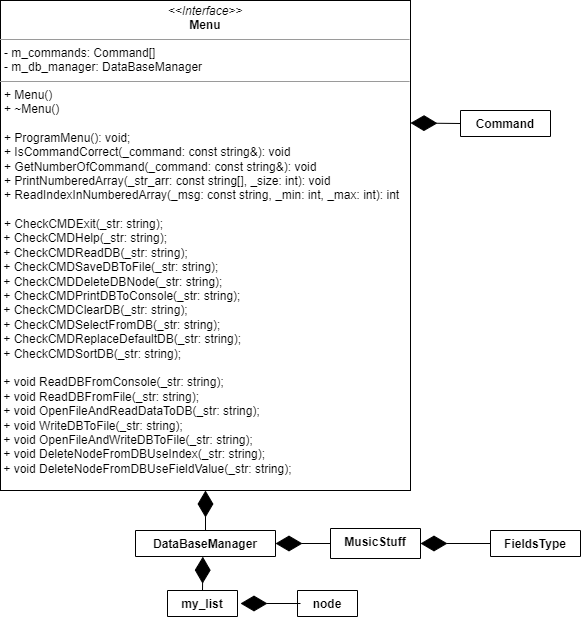
Вариант 10.

«Музыкальный товар»: носитель (грампластинка, аудиокассета, лазерный диск);

порядковый номер в каталоге; название; исполнитель (фамилия, имя); время звучания;

количество произведений; цена по каталогу.

# Схема классов

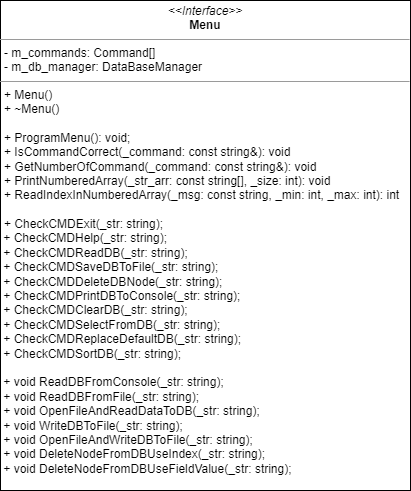


# Описание классов

## MENU

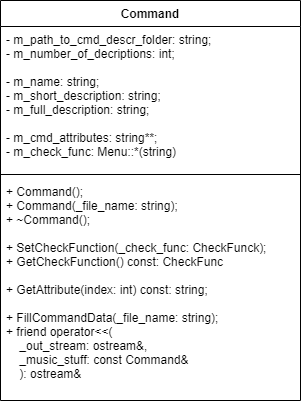
Класс меню является основным классом этой СУБД. Именно в нем прописаны все диалоги между пользователем и программой. Именно этот класс позволяет заполнять, удалять, сортировать, и многие другие операции, которые можно совершать с базой данных.

Графическое представление в виде прямоугольника дает явное представление о полях, которые есть в этом классе.



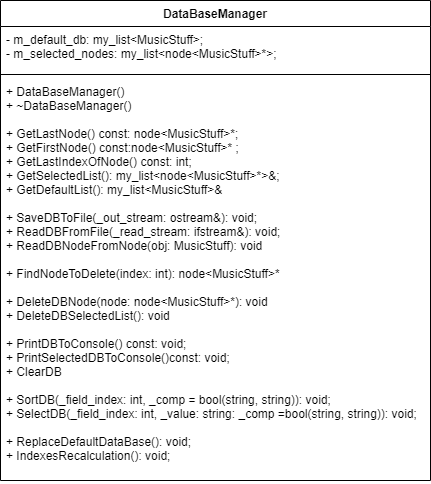
## COMMAND

Этот класс позволяет распознавать команды, которые вводит пользователь в консоль. Он хранит краткое описание и полное описание каждой команды, а также ее название. Также этот класс хранит функцию, которой должна обрабатываться каждая команда. То есть, при вводе команды в консоль, именно этот класс, а именно его поле **m\_name**, позволяет программе понять, что такая команда есть, после чего позволяет вызвать ее функцию обработки, указатель на которую хранится в **m\_check\_func**.



## DATABASEMANAGER

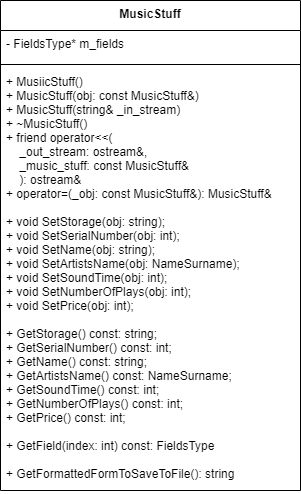
Этот класс является управляющим всей базой данных. Он хранит саму базу данных в виде двусвязного списка в переменной **m\_default\_db**, а также вспомогательную базу данных **m\_selected\_nodes**, состоящую из списка с указателями на элементы из основной базы данных. В основной базе данных хранится вся база данных, которая, на текущий момент, была получена программой от пользователя путем вызова различных функций этого же класса. Вспомогательная база данных нужна выборки определенных элементов и для дальнейшего взаимодействия с ними.



Класс **DataBaseManager** имеет множество методов, позволяющих с легкостью манипулировать базой данных так, как надо пользователю.

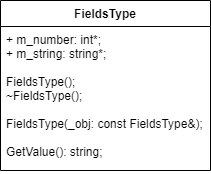
## MUSICSTUFF

Этот класс является элементом базы данных. Из объектов типа **MusicStuff** состоит список в **DataBaseManager**. Класс имеет массив полей, которые позволяют хранить информацию двух типов данных: **string**, **int**. Множество методов обеспечивает создание, удаление, вывод этого класса в консоль и огромное количество других возможностей, которые необходимы для манипулирования данными в полях и не только.



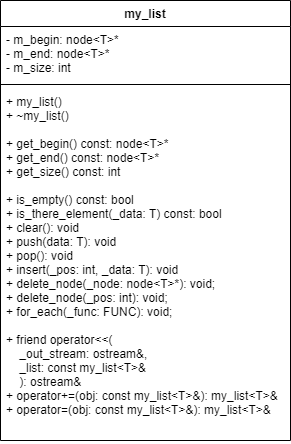
## FIELDSTYPE

Этот структура, хранящая указатели на два типа данных: **int** и **string**. Она определяет тип поля в базе данных. Из таких полей состоит **MusicStuff**. При создании объекта **FieldsType**, определяется тип данных, который будет хранить только что созданный объект.

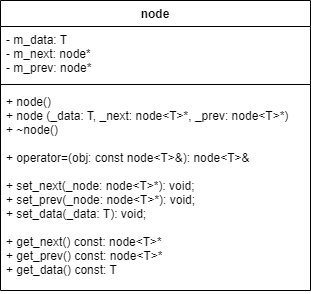


## MY\_LIST

Этот шаблонный класс является двусвязным списком. На его основе создаются объекты базы данных в **DataBaseManager**. **My\_list** хранит указатель на первый элемент списка, указатель на последний элемент списка и размер самого списка. Множество методов создают интерфейс, благодаря которому список можно создавать, удалять, искать в нем элементы и множество других действий.



## NODE



Этот класс является элементом списка. На его основе класс **my\_list** создает двусвязный список. Этот класс позволяет хранить какие–либо данные и иметь доступ к следующим элементам или предыдущим.

# Код программы

Код программы включает в себя множество файлов, которые будут указаны ниже. Каждый файл будет относится к какому-то классу или выполнять вспомогательную роль.

## command.cpp

#include "command.h"

Command::Command(string \_file\_name)

:m\_path\_to\_cmd\_descr\_folder("assets/messages/"),

m\_number\_of\_descriptions(3),

m\_cmd\_attributes(new string\* [m\_number\_of\_descriptions]),

m\_check\_func(nullptr)

{

m\_cmd\_attributes[0] = &m\_name;

m\_cmd\_attributes[1] = &m\_short\_description;

m\_cmd\_attributes[2] = &m\_full\_description;

FillCommandData(\_file\_name);

}

Command::Command()

:m\_path\_to\_cmd\_descr\_folder("assets/messages/"),

m\_number\_of\_descriptions(3),

m\_cmd\_attributes(new string\* [m\_number\_of\_descriptions]),

m\_check\_func(nullptr)

{

m\_cmd\_attributes[0] = &m\_name;

m\_cmd\_attributes[1] = &m\_short\_description;

m\_cmd\_attributes[2] = &m\_full\_description;

}

Command::~Command()

{

if (m\_cmd\_attributes)

delete[] m\_cmd\_attributes;

}

void Command::SetCheckFunction(CheckFunc \_check\_func)

{

m\_check\_func = \_check\_func;

}

Command::CheckFunc Command::GetCheckFunction() const

{

return m\_check\_func;

}

string Command::GetAttribute(int \_index) const

{

if (0 <= \_index && \_index <= m\_number\_of\_descriptions)

{

return \*m\_cmd\_attributes[\_index];

}

return "";

}

void Command::FillCommandData(string \_file\_name)

{

string file\_path = m\_path\_to\_cmd\_descr\_folder + \_file\_name;

ifstream fin(file\_path);

if (!fin)

{

cout << "\tFillCommandData: файл " << file\_path << " не открылся\n";

return;

}

string temp;

int ind = 0; // индекс текущего заполняемого поля

// считывание данных из файла

while (fin.peek() != EOF && ind < m\_number\_of\_descriptions)

{

// считывание строки

getline(fin, temp);

// если последний символ - ;

if (temp[temp.size() - 1] == ';')

{

// удаляем этот символ

temp = temp.substr(0, temp.size() - 1);

// сохраняем полученную строку

\*m\_cmd\_attributes[ind] += temp;

// переходим к следующей команде

ind++;

}

// если это не конец всей фразы

else

{

\*m\_cmd\_attributes[ind] += temp + '\n';

}

}

}

ostream& operator<<(ostream& \_out\_stream, const Command& \_cmd)

{

\_out\_stream << \_cmd.m\_name << endl

<< \_cmd.m\_short\_description << endl

<< \_cmd.m\_full\_description << endl;

return \_out\_stream;

}

## command.h

#pragma once

#include <iostream>

#include <fstream>

#include "../support\_func/support\_func.hpp"

using namespace std;

// объявление струкутры меню

class Menu;

// структура, содержащая описание каждой команды

class Command

{

string m\_path\_to\_cmd\_descr\_folder; // путь до папки c описанием команд

int m\_number\_of\_descriptions; // количество команд

string m\_name; // имя команды

string m\_short\_description; // краткое описание

string m\_full\_description; // полное описание

string\*\* m\_cmd\_attributes; // массив с указателями на поля структуры

// указатель на функцию обработки

typedef void (Menu::\* CheckFunc)(string);

CheckFunc m\_check\_func;

public:

// конструкторы

Command(string \_file\_name);

Command();

// деструктор

~Command();

// установить функцию обработки команды

void SetCheckFunction(CheckFunc);

// вернуть функцию обработки команды

// нужно для вызова функции обработки извне

CheckFunc GetCheckFunction() const;

// получение атрибутов команды

// (имя, короткое описание, полное описание)

string GetAttribute(int \_index) const;

// заполнение команды данными из файла

void FillCommandData(string \_file\_name);

// оператор вывода

friend ostream& operator<<(ostream& \_out\_stream, const Command& \_cmd);

};

## music\_stuff.cpp

В файле хранится реализация методов класса **MusicStuff**, а также структуры **FieldsType**.

#include "music\_stuff.h"

MusicStuff::MusicStuff()

{

m\_fields = new FieldsType[NUMBER\_OF\_FIELDS];

m\_fields[0].m\_string = new string; // носитель

m\_fields[1].m\_number = new int; // порядковый номер

m\_fields[2].m\_string = new string; // название

m\_fields[3].m\_string = new string; // имя исполнителя

m\_fields[4].m\_string = new string; // фамилия исполнителя

m\_fields[5].m\_number = new int; // время звучания в минутах

m\_fields[6].m\_number = new int; // количество воспросизведений

m\_fields[7].m\_number = new int; // цена

}

MusicStuff::MusicStuff(const MusicStuff& \_obj)

:MusicStuff()

{

\*this = \_obj;

}

MusicStuff::MusicStuff(string& \_input\_string)

:MusicStuff()

{

// заполнение класса MusicStuff

\*SERIAL\_NUMBER = GET\_INT\_DATA(\_input\_string);

\*STORAGE = GET\_DATA(\_input\_string);

\*NAME = GET\_DATA(\_input\_string);

\*ARTIST\_NAME = GET\_DATA(\_input\_string);

\*ARTIST\_SURNAME = GET\_DATA(\_input\_string);

\*SOUND\_TIME = GET\_INT\_DATA(\_input\_string);

\*NUMBER\_OF\_PLAYS = GET\_INT\_DATA(\_input\_string);

\*PRICE = GET\_INT\_DATA(\_input\_string);

}

MusicStuff::~MusicStuff()

{

delete[] m\_fields;

}

ostream& operator<<(ostream& \_out\_stream, const MusicStuff& \_music\_stuff)

{

\_out\_stream

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[0]) << \*\_music\_stuff.SERIAL\_NUMBER

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[1]) << \*\_music\_stuff.STORAGE

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[2]) << \*\_music\_stuff.NAME

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[3]) << \*\_music\_stuff.ARTIST\_NAME

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[4]) << \*\_music\_stuff.ARTIST\_SURNAME

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[5]) << \*\_music\_stuff.SOUND\_TIME

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[6]) << \*\_music\_stuff.NUMBER\_OF\_PLAYS

<< "|" << OUT\_W(' ', FIELDS\_WIDTH[7]) << \*\_music\_stuff.PRICE

<< "|";

return \_out\_stream;

}

MusicStuff& MusicStuff::operator=(const MusicStuff& \_obj)

{

\*STORAGE = \*\_obj.STORAGE;

\*SERIAL\_NUMBER = \*\_obj.SERIAL\_NUMBER;

\*NAME = \*\_obj.NAME;

\*ARTIST\_NAME = \*\_obj.ARTIST\_NAME;

\*ARTIST\_SURNAME = \*\_obj.ARTIST\_SURNAME;

\*SOUND\_TIME = \*\_obj.SOUND\_TIME;

\*NUMBER\_OF\_PLAYS = \*\_obj.NUMBER\_OF\_PLAYS;

\*PRICE = \*\_obj.PRICE;

return \*this;

}

void MusicStuff::SetStorage(string \_str)

{

\*STORAGE = \_str;

}

void MusicStuff::SetSerialNumber(int \_num)

{

\*SERIAL\_NUMBER = \_num;

}

void MusicStuff::SetName(string \_str)

{

\*NAME = \_str;

}

void MusicStuff::SetArtistsName(string \_str)

{

\*ARTIST\_NAME = \_str;

}

void MusicStuff::SetArtistsSurname(string \_str)

{

\*ARTIST\_SURNAME = \_str;

}

void MusicStuff::SetSoundTime(int \_num)

{

\*SOUND\_TIME = \_num;

}

void MusicStuff::SetNumberOfPlays(int \_num)

{

\*NUMBER\_OF\_PLAYS = \_num;

}

void MusicStuff::SetPrice(int \_num)

{

\*PRICE = \_num;

}

string MusicStuff::GetStorage() const

{

return \*STORAGE;

}

int MusicStuff::GetSerialNumber() const

{

return \*SERIAL\_NUMBER;

}

string MusicStuff::GetName() const

{

return \*NAME;

}

string MusicStuff::GetArtistsName() const

{

return \*ARTIST\_NAME;

}

string MusicStuff::GetArtistsSurname() const

{

return \*ARTIST\_SURNAME;

}

int MusicStuff::GetSoundTime() const

{

return \*SOUND\_TIME;

}

int MusicStuff::GetNumberOfPlays() const

{

return \*NUMBER\_OF\_PLAYS;

}

int MusicStuff::GetPrice() const

{

return \*PRICE;

}

FieldsType MusicStuff::GetField(int index) const

{

if (0 <= index && index <= NUMBER\_OF\_FIELDS)

{

return m\_fields[index];

}

return FieldsType();

}

string MusicStuff::GetFormattedFormToSaveToFile() const

{

return string(

to\_string(\*SERIAL\_NUMBER) + ':' +

\*STORAGE + ':' +

\*NAME + ':' +

\*ARTIST\_NAME + ':' +

\*ARTIST\_SURNAME + ':' +

to\_string(\*SOUND\_TIME) + ':' +

to\_string(\*NUMBER\_OF\_PLAYS) + ':' +

to\_string(\*PRICE)

);

}

FieldsType::FieldsType()

:m\_number(NULL), m\_string(NULL)

{

}

FieldsType::~FieldsType()

{

if(m\_number != NULL)

delete m\_number;

if(m\_string != NULL)

delete m\_string;

}

FieldsType::FieldsType(const FieldsType& \_obj)

:FieldsType()

{

if (\_obj.m\_number)

{

if (m\_number) delete m\_number;

m\_number = new int;

\*m\_number = \*\_obj.m\_number;

}

if (\_obj.m\_string)

{

if (m\_string) delete m\_string;

m\_string = new string;

\*m\_string = \*\_obj.m\_string;

}

}

string FieldsType::GetValue()

{

if (m\_number)

return to\_string(\*m\_number);

if (m\_string)

return \*m\_string;

return "";

}

## music\_stuff.h

Файл хранит описание класса **MusicStuff** и структуры **FieldsType**.

#pragma once

#include "../../support\_func/support\_func.hpp"

#include "../../../list/my\_list/my\_list.hpp"

// поле базы данных

struct FieldsType

{

int\* m\_number;

string\* m\_string;

FieldsType();

~FieldsType();

FieldsType(const FieldsType&);

string GetValue();

};

class MusicStuff

{

private:

// носитель

// порядковый номер

// название

// имя исполнителя

// время звучания в минутах

// количество воспросизведений

// цена

// поля базы данных

FieldsType\* m\_fields;

// для упрощенного доступа к полям

#define STORAGE m\_fields[0].m\_string

#define SERIAL\_NUMBER m\_fields[1].m\_number

#define NAME m\_fields[2].m\_string

#define ARTIST\_NAME m\_fields[3].m\_string

#define ARTIST\_SURNAME m\_fields[4].m\_string

#define SOUND\_TIME m\_fields[5].m\_number

#define NUMBER\_OF\_PLAYS m\_fields[6].m\_number

#define PRICE m\_fields[7].m\_number

public:

// конструктор без параметров

MusicStuff();

// конструктор копирования

MusicStuff(const MusicStuff&);

// конструктор считывающий данные из строки

MusicStuff(string& \_input\_string);

// деструктор

~MusicStuff();

// оператор вывода в консоль

friend ostream& operator<<(ostream& \_out\_stream, const MusicStuff& \_music\_stuff);

// оператор присваивания

MusicStuff& operator= (const MusicStuff& \_obj);

// Set методы

void SetStorage(string);

void SetSerialNumber(int);

void SetName(string);

void SetArtistsName(string);

void SetArtistsSurname(string);

void SetSoundTime(int);

void SetNumberOfPlays(int);

void SetPrice(int);

// Get методы

string GetStorage() const;

int GetSerialNumber() const;

string GetName() const;

string GetArtistsName() const;

string GetArtistsSurname() const;

int GetSoundTime() const;

int GetNumberOfPlays() const;

int GetPrice() const;

FieldsType GetField(int index) const;

string GetFormattedFormToSaveToFile() const;

};

## music\_stuff\_list.cpp

Файл содержит реализацию методов класса **DataBaseManager**.

#include "music\_stuff\_list.h"

DataBaseManager::DataBaseManager()

{

}

DataBaseManager::~DataBaseManager()

{

}

node<MusicStuff>\* DataBaseManager::GetLastNode() const

{

return m\_default\_db.get\_end();

}

node<MusicStuff>\* DataBaseManager::GetFirstNode() const

{

return m\_default\_db.get\_begin();

}

int DataBaseManager::GetLastIndexOfNode() const

{

node<MusicStuff>\* el = GetLastNode();

int number = 1;

if (el != NULL)

{

number = el->get\_data().GetSerialNumber();

}

return number;

}

my\_list<node<MusicStuff>\*>& DataBaseManager::GetSelectedList()

{

return m\_selected\_nodes;

}

my\_list<MusicStuff>& DataBaseManager::GetDefaultList()

{

return m\_default\_db;

}

my\_list<string>\* DataBaseManager::GetDataInField(int \_index) const

{

// выходной список

my\_list<string>\* out = new my\_list<string>;

// элемент для прохода всего списка

node<MusicStuff>\* el = m\_default\_db.get\_begin();

while (el)

{

string str = el->get\_data().GetField(\_index).GetValue();

if (!out->is\_there\_element(str))

{

out->push(str);

}

el = el->get\_next();

}

return out;

}

void DataBaseManager::SaveDBToFile(ostream& \_out\_stream)

{

node<MusicStuff>\* el = m\_default\_db.get\_begin();

while (el)

{

// вывод форматированной строки в файловый поток

\_out\_stream <<

el->get\_data().GetFormattedFormToSaveToFile()

<< endl;

// переход к следующему элементу

el = el->get\_next();

}

}

void DataBaseManager::ReadDBNodeFromNode(MusicStuff \_obj)

{

m\_default\_db.push(\_obj);

IndexesRecalculation();

}

node<MusicStuff>\* DataBaseManager::FindNodeToDelete(int \_index)

{

// если индекс больше или равен размеру списка

if (\_index >= m\_default\_db.get\_size())

{

FUNC\_INFO(

"индекс " + to\_string(\_index + 1) +

" больше или равен размеру списка " +

to\_string(m\_default\_db.get\_size())

);

return NULL;

}

// поиск элемента для удаления

node<MusicStuff>\* temp = m\_default\_db.get\_begin();

int index = 0;

while (temp && index < \_index)

{

index++;

temp = temp->get\_next();

}

return temp;

}

void DataBaseManager::DeleteDBNode(node<MusicStuff>\* \_node)

{

m\_default\_db.delete\_node(\_node);

IndexesRecalculation();

}

void DataBaseManager::DeleteDBSelectedList()

{

node<node<MusicStuff>\*>\* el = m\_selected\_nodes.get\_begin();

while (el)

{

node<node<MusicStuff>\*>\* next = el->get\_next();

m\_default\_db.delete\_node(el->get\_data());

m\_selected\_nodes.delete\_node(el);

el = next;

}

IndexesRecalculation();

}

void DataBaseManager::ReadDBFromFile(ifstream& \_read\_stream)

{

// очистка базы данных перед чтением

m\_default\_db.clear();

// считывание данных

while (\_read\_stream.peek() != EOF)

{

// пропуск ненужных переводов строк

if (\_read\_stream.peek() == '\n')

{

\_read\_stream.get();

}

// если не нашли переход строки

// (строка не пустая)

else

{

// считываем строку с информацией

string data\_str;

getline(\_read\_stream, data\_str);

// записываем эту информацию в список

m\_default\_db.push(data\_str);

}

}

}

void DataBaseManager::PrintDBToConsole() const

{

// печатаем список, если он не пуст

if (m\_default\_db.get\_size() != 0)

{

cout << TABLE\_CAP;

cout << m\_default\_db;

}

else

{

FUNC\_INFO("Список пуст");

}

}

void DataBaseManager::PrintSelectedDBToConsole() const

{

// печатаем список, если он не пуст

if (m\_default\_db.get\_size() != 0)

{

cout << TABLE\_CAP;

// создаем элемент для чтения данных из list'а

node<node<MusicStuff>\*>\* cur\_el = m\_selected\_nodes.get\_begin();

// идем по list'у, пока на наткнемся на конечный элемент

while (cur\_el != nullptr)

{

// вывод данных элемента

cout << cur\_el->get\_data()->get\_data() << '\n';

// переход к следующему элементу

cur\_el = cur\_el->get\_next();

}

}

else

{

FUNC\_INFO("Список пуст");

}

}

void DataBaseManager::ClearDB()

{

m\_default\_db.clear();

}

void DataBaseManager::SortDB(int \_field\_index, bool \_comp(string, string))

{

node<MusicStuff>\* el = m\_default\_db.get\_begin();

while (el)

{

node<MusicStuff>\* el2 = m\_default\_db.get\_begin();

while (el2)

{

if (

\_comp(

el->get\_data().GetField(\_field\_index).GetValue(),

el2->get\_data().GetField(\_field\_index).GetValue()

)

)

{

MusicStuff temp = el->get\_data();

el->set\_data(el2->get\_data());

el2->set\_data(temp);

}

el2 = el2->get\_next();

}

el = el->get\_next();

}

IndexesRecalculation();

}

void DataBaseManager::SelectDB(int \_field\_index, string \_value, bool \_comp(string, string))

{

// очистка выборочной базы данных

m\_selected\_nodes.clear();

// элемент для прохождения по списку

node<MusicStuff>\* temp = m\_default\_db.get\_begin();

// проходимся по всей базе данных и

// отбираем элементы со значением \_value

// в поле с индексом \_field\_index

while (temp)

{

if (

\_comp(

temp->get\_data().GetField(\_field\_index).GetValue(),

\_value

)

)

{

m\_selected\_nodes.push(temp);

}

temp = temp->get\_next();

}

}

void DataBaseManager::ReplaceDefaultDataBase()

{

node<MusicStuff>\* elem = m\_default\_db.get\_begin();

while (elem)

{

node<MusicStuff>\* next = elem->get\_next();

if (!m\_selected\_nodes.is\_there\_element(elem))

{

m\_default\_db.delete\_node(elem);

}

elem = next;

}

IndexesRecalculation();

}

void DataBaseManager::IndexesRecalculation()

{

node<MusicStuff>\* temp = m\_default\_db.get\_begin();

int index = 1;

while (temp)

{

MusicStuff temp\_ms = temp->get\_data();

temp\_ms.SetSerialNumber(index);

temp->set\_data(temp\_ms);

index++;

temp = temp->get\_next();

}

}

## music\_stuff\_list.h

#pragma once

#include "../music\_stuff/music\_stuff.h"

#include "../../../list/my\_list/my\_list.hpp"

class DataBaseManager

{

// база данных со всеми данными

my\_list<MusicStuff> m\_default\_db;

// списоск с выбранными элеметами

my\_list<node<MusicStuff>\*> m\_selected\_nodes;

public:

// Конструктор по умолчанию

DataBaseManager();

// Деструктор по умолчанию

~DataBaseManager();

// Get Методы

// получить последний элемент

node<MusicStuff>\* GetLastNode() const;

// получить первый элемент

node<MusicStuff>\* GetFirstNode() const;

// получение индекса последнего элемента из списка

int GetLastIndexOfNode() const;

// получение списка элементов с полем определенного значения

my\_list<node<MusicStuff>\*>& GetSelectedList();

// получение основного списка элементов

my\_list<MusicStuff>& GetDefaultList();

// печать значений поля по индексу

my\_list<string>\* GetDataInField(int \_index) const;

// Функции менеджера

// сохранение базы данных в файл

void SaveDBToFile(ostream& \_out\_stream);

// считать базу данных из файла

void ReadDBFromFile(ifstream& \_read\_stream);

// считать элемень из уже существующего

void ReadDBNodeFromNode(MusicStuff);

// поиск элемента для удаления

node<MusicStuff>\* FindNodeToDelete(int \_index);

// удалить элемент из базы данных

void DeleteDBNode(node<MusicStuff>\* \_node);

void DeleteDBSelectedList();

// печать в поток

void PrintDBToConsole() const;

// печать выборочной БД в поток

void PrintSelectedDBToConsole() const;

// очистить базу данных

void ClearDB();

// сортировать базу данных

void SortDB(int \_field\_index, bool(string, string) = COMPARE::IsLower);

// выборка элементов

void SelectDB(int \_field\_index, string \_value

, bool(string, string) = COMPARE::IsEqual);

// замена основной базы данных той, которая в выборке

void ReplaceDefaultDataBase();

// перерасчет индексов

void IndexesRecalculation();

};

## menu.cpp

Файл содержит описание методов класса **Menu.**

#include "menu.h"

Menu::Menu()

{

// выделение памяти под массив с описанием функций

m\_command = new Command[NUMBER\_OF\_COMMANDS];

// чтение команд из файлов

for (int i = 0; i < NUMBER\_OF\_COMMANDS; i++)

{

m\_command[i].FillCommandData(COMMAND\_DESCRIPTION\_FILES[i]);

}

// заполнение массива функций обработки команд

m\_command[0].SetCheckFunction(&Menu::CheckCMDExit);

m\_command[1].SetCheckFunction(&Menu::CheckCMDHelp);

m\_command[2].SetCheckFunction(&Menu::CheckCMDReadDB);

m\_command[3].SetCheckFunction(&Menu::CheckCMDPrintDBToConsole);

m\_command[4].SetCheckFunction(&Menu::CheckCMDSaveDBToFile);

m\_command[5].SetCheckFunction(&Menu::CheckCMDDeleteDBNode);

m\_command[6].SetCheckFunction(&Menu::CheckCMDClearDB);

m\_command[7].SetCheckFunction(&Menu::CheckCMDSelectFromDB);

m\_command[8].SetCheckFunction(&Menu::CheckCMDReplaceDefaultDB);

m\_command[9].SetCheckFunction(&Menu::CheckCMDSortDB);

}

Menu::~Menu()

{

delete[] m\_command;

}

void Menu::ProgramMenu()

{

// вывод справки о программе

cout << DBMS\_DESCRIPTION;

// строка с введенными командами

string input\_all\_command;

// строка с введенными командами

string input\_first\_command;

// строка с введенными аттрибутами команды

string input\_attributes;

// цикл выполнения программы

do

{

// вывод сообщения запроса команды

cout << DBMS\_CONSOLE\_REQUEST\_COMMAND;

// считывание команды

getline(cin, input\_all\_command);

// если строка не пустая,

// тогда можно проверять ее на корректность

if (input\_all\_command.length() != 0)

{

// подготовка строки для получения слов из нее

RemoveUnnecessarySpaces(input\_all\_command);

// перевод строки в нижний регистр

input\_all\_command = ToLowerCase(input\_all\_command);

// получение только команды (без атрибутов)

input\_first\_command = GetToken(input\_all\_command, ' ');

// если введенное слово является командой

// и не был введен выход

if (

IsCommandCorrect(input\_first\_command) &&

input\_first\_command != CMD\_NAME(0)

)

{

//вызов необходимой функции для команды

CMD\_CHK\_FUNC(input\_first\_command, input\_all\_command);

}

// если была введена не команда

else if (input\_first\_command != CMD\_NAME(0))

{

INFO("\"" + input\_first\_command + "\"" + NOT\_CORRECT\_COMMAND);

}

}

} while (input\_first\_command != CMD\_NAME(0));

}

bool Menu::IsCommandCorrect(const string& \_command)

{

// проходимся по массиву команд

for (int i = 0; i < NUMBER\_OF\_COMMANDS; i++)

{

// если нашли совпадающую команду

// выводим 1

if (CMD\_NAME(i) == \_command)

{

return true;

}

}

// иначе выводим 0

return false;

}

int Menu::GetNumberOfCommand(const string& \_command)

{

// проходимся по массиву команд

for (int i = 0; i < NUMBER\_OF\_COMMANDS; i++)

{

// если нашли совпадающую команду

// выводим i

if (CMD\_NAME(i) == \_command)

{

return i;

}

}

// иначе -1

return -1;

}

void Menu::PrintNumberedArray(const string \_arr[], int \_size)

{

for (int i = 0; i < \_size; i++)

{

cout << "\t" << i + 1 << ") " << \_arr[i] << "\n";

}

}

int Menu::ReadIndexInNumberedArray(const string \_str, int \_min, int \_max)

{

return atoi(

CheckableRead(

"\t[" + \_str + "]> ",

[\_min, \_max](string num)

{

return IsThereANumber(num) &&

\_min <= atoi(num.c\_str()) &&

atoi(num.c\_str()) <= \_max;

}

).c\_str()) - 1;

}

// проверка команды выход

void Menu::CheckCMDExit(string \_str)

{

// параметров у нее нет, так что их

// не надо проверять

}

// проверка команды ПОМОЩЬ

void Menu::CheckCMDHelp(string \_str)

{

// если аргументы не были переданы

// тогда выводим все команды и информацию по ним

if (!\_str.length())

{

for (int i = 0; i < NUMBER\_OF\_COMMANDS; i++)

{

cout << ToUpperCase(CMD\_NAME(i))

<< "\t\t" << CMD\_SH\_DECR(i)

<< endl;

}

}

// если есть какие-то аргументы

else

{

// если этот аргумент - команда

// выводим подробную инфу по ней

if (IsCommandCorrect(\_str))

{

cout << CMD\_FL\_DESCR(GetNumberOfCommand(\_str)) << endl;

}

// если такого аргумента не существует

else

{

cout << "\t\"" << \_str << "\"" << NOT\_CORRECT\_COMMAND;

}

}

}

// проверка команды ЧТЕНИЕДАННЫХ

void Menu::CheckCMDReadDB(string \_str)

{

// место считывания информации

string temp = GetToken(\_str);

// проверка места считывания информации

// из файла

if (!temp.length())

{

// запуск режима диалога

cout << "\n\tВыберете способ ввода информации:\n";

// Вывод названий полей в базе данных

PrintNumberedArray(INFO\_ENTER, INFO\_ENTER\_SIZE);

// номер типа считывания

int read\_type = ReadIndexInNumberedArray(

"Откуда считывать информацию",

1, INFO\_ENTER\_SIZE

);

// если считывание с консоли

if (read\_type == 0)

{

ReadDBFromConsole("");

}

// если считывание из файла

else if (read\_type == 1)

{

ReadDBFromFile("");

}

}

// из стандартного файла

else if (temp == "-ф")

{

// получение файлового пути

temp = GetToken(\_str);

// Считывание из файла

ReadDBFromFile(temp);

}

// если введенная информация - не ключ

else

{

FUNC\_INFO("неизвестный ключ: \"" + temp + "\"");

}

}

// сохранение базы данных в файл

void Menu::CheckCMDSaveDBToFile(string \_str)

{

RETURN\_IF\_LIST\_IS\_EMPTY;

// считывание ключа

string temp = GetToken(\_str);

// если ключей нет

// запуск диалога

if (!temp.length())

{

INFO("\n\tВыберете место вывода информации:");

// печать типов

PrintNumberedArray(FILE\_PATH, FILE\_PATH\_SIZE);

// ввод номера варианта

int read\_type = ReadIndexInNumberedArray(

"Как считывать файл",

1, FILE\_PATH\_SIZE

);

// использование файла по умолчанию

if (read\_type == 0)

{

WriteDBToFile("");

}

// если надо ввести путь из консоли

else if (read\_type == 1)

{

INFO("\n\tФайл должен быть в папке " + DB\_FOLDER\_PATH);

WriteDBToFile(CheckableRead("\t[Введите имя файла]> "));

}

}

// если введен ключ -ф

else if (temp == "-ф")

{

WriteDBToFile(GetToken(\_str));

}

else

{

FUNC\_INFO("неизвестный ключ: \"" + temp + "\"");

}

}

// удаление элеемнта из базы данных

void Menu::CheckCMDDeleteDBNode(string \_str)

{

RETURN\_IF\_LIST\_IS\_EMPTY;

// как удалять: с помощью индекса

// или с помощью поля

string temp = GetToken(\_str);

// если ключей нет

if (!temp.length())

{

// запус диалога

INFO("\n\tВыберете тип удаления информации:");

// печать типов

PrintNumberedArray(DELETE\_NODE, DELETE\_NODE\_SIZE);

// ввод номера варианта

int type = ReadIndexInNumberedArray(

"Как удалять",

1, DELETE\_NODE\_SIZE

);

// удаление по индексу

if (type == 0)

{

DeleteNodeFromDBUseIndex("");

}

// удаление по значению поля

else if (type == 1)

{

DeleteNodeFromDBUseFieldValue("");

}

}

// если было введено удаление по индексу

else if (temp == "-и")

{

DeleteNodeFromDBUseIndex(\_str);

}

// если было введено удаление по полю

else if (temp == "-п")

{

DeleteNodeFromDBUseFieldValue(\_str);

}

else

{

FUNC\_INFO("неизвестный ключ: \"" + temp + "\"");

}

}

// проверка команды ПЕЧАТЬДАННЫХ

void Menu::CheckCMDPrintDBToConsole(string \_str)

{

RETURN\_IF\_LIST\_IS\_EMPTY;

m\_db\_manager.PrintDBToConsole();

}

// очистка базы данных

void Menu::CheckCMDClearDB(string \_str)

{

m\_db\_manager.ClearDB();

INFO("База данных очищена");

}

// выбрать из базы данных определенны элементы

void Menu::CheckCMDSelectFromDB(string \_str)

{

RETURN\_IF\_LIST\_IS\_EMPTY;

// взятие ключа

string key = GetToken(\_str);

// диалоговое сообщение

cout << "\n\tВозможные поля для выборки:\n";

// Вывод названий полей в базе данных

PrintFieldsOfDataBase();

// индекс поля

int number\_of\_field =

atoi(

CheckableRead(

"\t[По какому полю осуществлять выбор]> ",

[](string num)

{

return

IsThereANumber(num) &&

1 <= atoi(num.c\_str()) &&

atoi(num.c\_str()) <= NUMBER\_OF\_FIELDS;

}

).c\_str()) - 1;

// получение списка значений поля

my\_list<string> out = \*m\_db\_manager.GetDataInField(number\_of\_field);

// печать списка

cout << "\n\tУникальные значения этого поля:\n";

int ind = 1;

out.for\_each([&ind](node<string>\* el)

{

cout << "\t" << ind++ << ") " << el->get\_data() << "\n";

}

);

// какое значение искать в поле

int number\_of\_value =

atoi(

CheckableRead(

"\t[Какое значение использовать в поле]> ",

[&out](string num)

{

return

IsThereANumber(num) &&

1 <= atoi(num.c\_str()) &&

atoi(num.c\_str()) <= out.get\_size();

}

).c\_str()) - 1;

// значение поля под индексом number\_of\_value

// с которым будем сравнивать

string field\_value = out.get\_element\_by\_index(number\_of\_value)->get\_data();

cout << "\n\tЕсли неравенство верно, происходит выбор. Типы сравнений:\n";

// вывод типов сравнений

for (int i = 0; i < NUMBER\_OF\_COMPARISONS; i++)

{

cout << "\t" << i + 1 << ") " << "элементы " <<

NAMES\_OF\_COMPARISONS[i] << ' ' << field\_value << "\n";

}

// выбор типа сравнения элементов

int comp\_type =

atoi(

CheckableRead(

"\t[Какой тип сравнения использовать]> ",

[](string num)

{

return

IsThereANumber(num) &&

1 <= atoi(num.c\_str()) &&

atoi(num.c\_str()) <= NUMBER\_OF\_COMPARISONS;

}

).c\_str()) - 1;

// выборка элементов списка с определенным

// значением определенного поля

m\_db\_manager.SelectDB

(

number\_of\_field,

field\_value,

COMPARE::COMPARISONS[comp\_type]

);

cout << "\n[Выбраны следующие элементы]> \n";

// Вывод шапки таблицы

cout << TABLE\_CAP;

// печать базы данных

if (!m\_db\_manager.GetSelectedList().is\_empty())

{

m\_db\_manager.GetSelectedList().for\_each([](auto \_el)

{

cout << \_el->get\_data()->get\_data() << endl;

}

);

}

else

{

INFO("\t Такие элементы не найдены");

}

}

// заменить исходную бд полученной из выборки

void Menu::CheckCMDReplaceDefaultDB(string \_str)

{

RETURN\_IF\_LIST\_IS\_EMPTY;

// вызов функции выборки

if (m\_db\_manager.GetSelectedList().is\_empty())

{

CheckCMDSelectFromDB("");

}

// иначе печать базы данных

else

{

m\_db\_manager.PrintSelectedDBToConsole();

}

// получение ответа

string answ = CheckableRead(

"\t[Готовы ли вы оставить только эти элементы? (да/нет)]> ",

[](string str)

{

return ToLowerCase(str) == "да" || ToLowerCase(str) == "нет";

}

);

if (answ == "да")

{

INFO("Замена текущей базы");

m\_db\_manager.ReplaceDefaultDataBase();

}

else

{

INFO("Элемнеты не заменены")

}

}

void Menu::CheckCMDSortDB(string \_str)

{

RETURN\_IF\_LIST\_IS\_EMPTY;

// Вывод названий полей в базе данных

PrintFieldsOfDataBase();

// индекс поля

int number\_of\_field =

atoi(

CheckableRead(

"\t[По какому полю осуществлять сортировку]> ",

[](string num)

{

return

IsThereANumber(num) &&

1 <= atoi(num.c\_str()) &&

atoi(num.c\_str()) <= NUMBER\_OF\_FIELDS;

}

).c\_str()) - 1;

INFO("\n\tТипы сортировки:");

// вывод типов сортировок

for (int i = 0; i < NUMBER\_OF\_SORTS; i++)

{

cout << "\t" << i + 1 << ") " << NAMES\_OF\_SORTS[i] << endl;

}

// взятие типа сортировки

int sort\_type =

atoi(

CheckableRead(

"\t[Каким образом сортировать]> ",

[](string num)

{

return

IsThereANumber(num) &&

1 <= atoi(num.c\_str()) &&

atoi(num.c\_str()) <= NUMBER\_OF\_SORTS;

}

).c\_str()) - 1;

// сортировка

m\_db\_manager.SortDB(number\_of\_field, COMPARE::COMPARISONS[sort\_type]);

INFO("Список отсортирован");

}

void Menu::ReadDBFromConsole(string \_str)

{

INFO("\n\tВвод с консоли:");

MusicStuff ms;

// считывание места хранения

ms.SetStorage(

CheckableRead(

"\t[Введите НОСИТЕЛЬ]> "

)

);

// запись порядкового номера

ms.SetSerialNumber(

m\_db\_manager.GetLastIndexOfNode() + 1

);

// считывание названия трека

ms.SetName(

CheckableRead(

"\t[Введите НАЗВАНИЕ ТРЕКА]> "

)

);

// считывание имени исполнителя

ms.SetArtistsName(

CheckableRead(

"\t[Введите ИМЯ исполнителя]> "

)

);

// считывание фамилии исполнителя

ms.SetArtistsSurname(

CheckableRead(

"\t[Введите ФАМИЛИЮ исполнителя]> "

)

);

// считывание времени проигрывания трека

ms.SetSoundTime(

atoi(

CheckableRead(

"\t[Введите ВРЕМЯ ЗВУЧАНИЯ (минуты)]> ",

IsThereANotNegativeNumber

).c\_str()

)

);

// считывание количества воспроизведений

ms.SetNumberOfPlays(

atoi(

CheckableRead(

"\t[Введите КОЛИЧЕСТВО ВОСПРОИЗВЕДЕНИЙ]> ",

IsThereANotNegativeNumber

).c\_str()

)

);

// считывание цены

ms.SetPrice(

atoi(

CheckableRead(

"\t[Введите ЦЕНУ (рубли)]> ",

IsThereANotNegativeNumber

).c\_str()

)

);

// запись считанного элемента в список

m\_db\_manager.ReadDBNodeFromNode(ms);

}

void Menu::ReadDBFromFile(string \_str)

{

INFO("\n\tЧтение из файла");

// проверка на наличие файлового пути

string temp = GetToken(\_str);

// если путь до файла не указан

// запуск диалога

if (!temp.length())

{

// печать типов

PrintNumberedArray(FILE\_PATH, FILE\_PATH\_SIZE);

// ввод номера варианта

int read\_type = ReadIndexInNumberedArray(

"Как считывать файл",

1, FILE\_PATH\_SIZE

);

// использование файла по умолчанию

if (read\_type == 0)

{

INFO("\n\tИспользование стандартного файла " + DB\_FILE\_PATH);

OpenFileAndReadDataToDB(DB\_FILE\_PATH);

}

// если надо ввести путь из консоли

else if (read\_type == 1)

{

INFO("\n\tФайл должен быть в папке " + DB\_FOLDER\_PATH);

OpenFileAndReadDataToDB(

DB\_FOLDER\_PATH + CheckableRead("\t[Введите имя файла]> ")

);

}

}

// если указан путь до файла

else

{

temp = GetToken(temp);

temp = DB\_FOLDER\_PATH + temp;

INFO("Использование файла " + temp);

OpenFileAndReadDataToDB(temp);

}

}

void Menu::OpenFileAndReadDataToDB(string \_str)

{

// создаем файловый поток и

// читаем информацию оттуда

ifstream fin(\_str);

// если не удалось открыть файл

if (!fin.is\_open())

{

// вывод сообщение об ошибке

INFO("Файл \"" + \_str + "\" не был открыт");

}

// иначе записываем информацию из файлв в консоль

else

{

m\_db\_manager.ReadDBFromFile(fin);

INFO("Чтение выполнено");

}

// закрытие файла

fin.close();

}

void Menu::WriteDBToFile(string \_str)

{

INFO("\n\tПечать в файл");

// получение файлового пути

string temp = GetToken(\_str);

// если путь до файла не указан

// запуск диалога

if (!temp.length())

{

INFO("Использование стандартного файла " + OUT\_DB\_FILE\_PATH);

OpenFileAndWriteDBToFile(OUT\_DB\_FILE\_PATH);

}

// если указан путь до файла

else

{

temp = GetToken(temp);

temp = DB\_FOLDER\_PATH + temp;

INFO("Использование файла " + temp);

OpenFileAndWriteDBToFile(temp);

}

}

void Menu::OpenFileAndWriteDBToFile(string \_str)

{

// создаем файловый поток

ofstream fout(\_str);

// если не удалось открыть файл

if (!fout.is\_open())

{

// вывод сообщение об ошибке

INFO("Файл \"" + \_str + "\" не был открыт");

}

// иначе записываем информацию из файлв в консоль

else

{

m\_db\_manager.SaveDBToFile(fout);

INFO("Запись выполнена");

}

// закрытие файла

fout.close();

}

void Menu::DeleteNodeFromDBUseIndex(string \_str)

{

// получение индекса

string temp = GetToken(\_str);

// если нет индекса

if (!temp.length())

{

// запуск диалога

INFO("\n\tВыберете индекс элемента:");

// печать списка

CheckCMDPrintDBToConsole("");

// ввод номера варианта

int index = ReadIndexInNumberedArray(

"Индекс удаляемого элемента",

1, m\_db\_manager.GetDefaultList().get\_size()

);

// поиск элемента для удаления

node<MusicStuff>\* nd = m\_db\_manager.FindNodeToDelete(index);

// если такого индекса нет

if (!nd)

{

FUNC\_INFO("элемент не найден");

return;

}

// уточнение у пользователя

// готов ли он удалить эти элементы

INFO("\n\tЭлемент для удаления:");

cout << TABLE\_CAP << endl;

cout << nd->get\_data() << endl;

// получение ответа

string answ = CheckableRead(

"\t[Готовы ли вы удалить его? (да/нет)]> ",

[](string str)

{

return ToLowerCase(str) == "да" || ToLowerCase(str) == "нет";

}

);

// если ответ да

if (answ == "да")

{

m\_db\_manager.DeleteDBNode(nd);

}

else

{

INFO("Элементы не удалены")

}

}

// если был введен индекс и он - число

else if (IsThereANotNegativeNumber(temp))

{

// поиск элемента для удаления

node<MusicStuff>\* nd =

m\_db\_manager.FindNodeToDelete(atoi(temp.c\_str()) - 1);

// если такого индекса нет

if (!nd)

{

FUNC\_INFO("Элемент не найден");

return;

}

m\_db\_manager.DeleteDBNode(nd);

}

else

{

FUNC\_INFO("неизвестный ключ: \"" + temp + "\"");

}

}

void Menu::DeleteNodeFromDBUseFieldValue(string \_str)

{

// процедура выборки

CheckCMDSelectFromDB("");

// получение ответа

string answ = CheckableRead(

"\t[Готовы ли вы удалить их? (да/нет)]> ",

[](string str)

{

return ToLowerCase(str) == "да" || ToLowerCase(str) == "нет";

}

);

if (answ == "да")

{

// удаление выбранных элементов из списка

m\_db\_manager.DeleteDBSelectedList();

}

else

{

INFO("Элементы не удалены")

}

}

## menu.h

Файл содержит описание класса **Menu**.

#pragma once

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* M E N U \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../support\_func/support\_func.hpp"

#include "../constants.h"

#include "../command/command.h"

#include "../db\_mng\_sys/music\_stuff\_list/music\_stuff\_list.h"

#include <fstream>

#include <Windows.h>

class Menu

{

/\* массив с командами

\* возможные коды управления СУБД

\* 0 выход

\* 1 помощь

\* 2 чтение

\* 3 печать

\* ...

\* ...

\*\*/

Command\* m\_command;

// Список музыкальных товаров

DataBaseManager m\_db\_manager;

// Для быстрого доступа к полям структуры

#define CMD\_NAME(num) m\_command[num].GetAttribute(0)

#define CMD\_SH\_DECR(num) m\_command[num].GetAttribute(1)

#define CMD\_FL\_DESCR(num) m\_command[num].GetAttribute(2)

#define CMD\_CHK\_FUNC(in\_name, in\_arg) (this->\*m\_command[GetNumberOfCommand(in\_name)].GetCheckFunction())(in\_arg)

// Выход из функции, если пуст лист

#define RETURN\_IF\_LIST\_IS\_EMPTY \

if (m\_db\_manager.GetDefaultList().is\_empty())\

{\

FUNC\_INFO("база данных пуста");\

return;\

}

public:

// конструктор и деструктор

Menu();

~Menu();

// функция меню

void ProgramMenu();

// Проверяет, является ли введенная строка командой СУБД

bool IsCommandCorrect(const string& \_command);

// Получение номера команды в массиве

int GetNumberOfCommand(const string& \_command);

// печать нумерованного массива параметров

void PrintNumberedArray(const string[], int);

// ввод номера параметра из нумерованного

// масива параметров

int ReadIndexInNumberedArray(const string \_msg, int \_min, int \_max);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// ФУНКЦИИ ПРОВЕРОК //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// проверка команды ВЫХОД

void CheckCMDExit(string \_str);

// проверка команды ПОМОЩЬ

void CheckCMDHelp(string \_str);

// проверка команды ЧТЕНИЕДАННЫХ

void CheckCMDReadDB(string \_str);

// сохранить базу данных в файл

void CheckCMDSaveDBToFile(string \_str);

// удалить элемент из базы данных

void CheckCMDDeleteDBNode(string \_str);

// печать базы данных в консоль в читаемом виде

void CheckCMDPrintDBToConsole(string \_str);

// очистка базы данных

void CheckCMDClearDB(string \_str);

// сортировка базы данных

void CheckCMDSelectFromDB(string \_str);

// замена исходной базы данных той,

// которая была получена из выборки

void CheckCMDReplaceDefaultDB(string \_str);

// сортировка базы данных

void CheckCMDSortDB(string \_str);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// ФУНКЦИИ ДЛЯ ФУНКЦИЙ ПРОВЕРОК //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// чтение из консоли

void ReadDBFromConsole(string \_str);

// чтение из файла

void ReadDBFromFile(string \_str);

// открытие файла по пути \_str

// и запись данных в базу данных

void OpenFileAndReadDataToDB(string \_str);

// запись БД в файл

void WriteDBToFile(string \_str);

// запись БД в файл

void OpenFileAndWriteDBToFile(string \_str);

// удаление элемента по индексу

void DeleteNodeFromDBUseIndex(string \_str);

// удаление элемента по значению поля

void DeleteNodeFromDBUseFieldValue(string \_str);

};

## support\_func.hpp

Файл содержит вспомогательные функции, необходимые для корректной обработки данных и множественных проверок.

#pragma once

#include "../constants.h"

using namespace std;

// Булевая функция по умолчанию

inline bool DefaultTrueFunc(string)

{

return true;

}

// Проверка на число

inline bool IsItANumber(string \_str)

{

// итератор на начало строки

std::string::const\_iterator it = \_str.begin();

// идем до символа, который окажется не цифрой

while (it != \_str.end() && std::isdigit(\*it)) it++;

// если строка не пустая и мы дошли до конца

// значит это число

return (\_str.end() == it);

}

// Проверка на наличие числа в строке

inline bool IsThereANumber(string \_str)

{

// итератор на начало строки

std::string::const\_iterator it = \_str.begin();

// идем до символа, который окажется не цифрой

while (it != \_str.end() && std::isdigit(\*it)) it++;

// если строка не пустая и мы дошли до конца

// значит это число

return !\_str.empty() && (\_str.begin() - it != 0);

}

// проверка на не отрицательное число

inline bool IsThereANotNegativeNumber(string \_str)

{

// если это число и оно не отрицательно

if (IsThereANumber(\_str) && atoi(\_str.c\_str()) >= 0)

return true;

return false;

}

// ввод и проверка значений

template<class FUNC = bool(string)>

inline string CheckableRead(

const string \_welcome\_str, // строка с запросом ввода

const FUNC& \_comp // функция сравнения

= DefaultTrueFunc,

const string \_err\_str // строка с ошибкой

= "Было введено некорректное значение"

)

{

// считываемый символ

string str;

// вывод сообщения

cout << \_welcome\_str;

// считывание из консоли

getline(cin, str);

// если было введено не то, что нужно было

if (!\_comp(str))

{

FUNC\_INFO(\_err\_str);

// рекурсивный запрос значения

str = CheckableRead<FUNC>(\_welcome\_str, \_comp, \_err\_str);

}

return str;

}

// является ли символ кириллическим

inline bool IsCyrillic(char \_symb)

{

return 'А' <= \_symb && \_symb <= 'я'

|| \_symb == 'ё' || \_symb == 'Ё';

}

// получение подстроки отделенной с помощью delim

// и удаление этой подстроки из изначальной строки

inline string GetToken(string& \_str, char \_delim = ' ')

{

// позиция делителя

size\_t delim\_pos = \_str.find(\_delim);

// если делитель не был найден, переносим всю строку

if (delim\_pos == -1)

{

delim\_pos = \_str.size();

}

// строка с нужной подстрокой

// копирование нужной строки

string new\_str = \_str.substr(0, delim\_pos);

// сдвиг всех символов в начало

\_str.erase(0, delim\_pos + 1);

return new\_str;

}

// печать всех названий полей

inline void PrintFieldsOfDataBase()

{

for (int i = 0; i < NUMBER\_OF\_FIELDS; i++)

{

cout << "\t" << i + 1 << ") " << NAMES\_OF\_FIELDS[i] << "\n";

}

}

// проверка на корректность данных

inline string IsStringNotEmpty(const string& \_str)

{

// если строка не пустая, тогда возвращаем ее

// иначе вернем ошибочное сообщение

return (\_str.length() ? \_str : NOT\_CORRECT\_DATA);

}

/// <summary>

/// Подготовка строки для получения из нее слов

/// </summary>

/// <param name="\_str"> строка, которую надо подготовить </param>

/// <param name="\_delim"> символ, который делит строку на слова</param>

inline void RemoveUnnecessarySpaces(string& \_str)

{

// добавление \_delim в конец

// нужно для корректной работы алгоритма удаления

\_str += ' ';

// уничтожение ненужных символов \_delim

size\_t begin\_del\_pos = -1;

size\_t end\_del\_pos = 0;

do

{

// поиск первого разделяющего символа

begin\_del\_pos = \_str.find(' ', begin\_del\_pos + 1);

// поиск конца последовательности из разделяющих символов

end\_del\_pos = \_str.substr(begin\_del\_pos).find\_first\_not\_of(' ');

// удаление этих пробелов

\_str.erase(begin\_del\_pos, end\_del\_pos - 1);

} while (

end\_del\_pos != string::npos &&

begin\_del\_pos != string::npos

);

// если первый символ - пробел

if (\_str[0] == ' ')

{

\_str.erase(0, 1);

}

}

/// <summary>

/// Перевод строки в нижний регистр

/// </summary>

/// <param name="\_str"> Строка, которую нужно перевести в нижний регистр</param>

/// <returns>Вернет строку в нижнем регистре</returns>

inline string ToLowerCase(const string \_str)

{

string out;

for (int i = 0; i < \_str.length(); i++)

{

if (IsCyrillic(\_str[i]) && \_str[i] < 'а')

{

out += (char(\_str[i] + 32));

}

else

{

out += tolower(\_str[i]);

}

}

return out;

}

/// <summary>

/// Перевод строки в верхний регистр

/// </summary>

/// <param name="\_str"> Строка, которую нужно перевести в верхний регистр</param>

/// <returns>Вернет строку в верхнем регистре</returns>

inline string ToUpperCase(const string \_str)

{

string out;

for (int i = 0; i < \_str.length(); i++)

{

if (IsCyrillic(\_str[i]) && \_str[i] >= 'а')

{

out += (char(\_str[i] - 32));

}

else

{

out += toupper(\_str[i]);

}

}

return out;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// ФУНКЦИИ СРАВНЕНИЯ СТРОК //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

namespace COMPARE

{

inline bool IsLower(string s1, string s2)

{

if (IsItANumber(s1) && IsItANumber(s2))

return atoi(s1.c\_str()) < atoi(s2.c\_str());

return s1 < s2 && s1.length() <= s2.length();

}

inline bool IsGreater(string s1, string s2)

{

if (IsItANumber(s1) && IsItANumber(s2))

return atoi(s1.c\_str()) > atoi(s2.c\_str());

return s1 > s2 && s1.length() >= s2.length();

}

inline bool IsEqual(string s1, string s2)

{

if (IsItANumber(s1) && IsItANumber(s2))

return atoi(s1.c\_str()) == atoi(s2.c\_str());

return s1 == s2;

}

inline bool IsLowerEqual(string s1, string s2)

{

return IsLower(s1, s2) || IsEqual(s1, s2);

}

inline bool IsGreaterEqual(string s1, string s2)

{

return IsGreater(s1, s2) || IsEqual(s1, s2);

}

// Массив функций сравнения

static bool (\*COMPARISONS[])(string, string) =

{

IsGreater,

IsLower,

IsEqual,

IsLowerEqual,

IsGreaterEqual

};

}

## constants.h

Файл содержит константы, необходимый, по большей части, для корректной работы диалога пользователя с программой.

#pragma once

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* C O N S T \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

#include <fstream>

#include <iomanip>

using namespace std;

// вывод в консоль сообщения

#define INFO(str) cout<<"\t"<<str<<"\n";

// вывод сообщения от имени функции

#define FUNC\_INFO(str) cout<<"\t" + string(\_\_FUNCTION\_\_) + ": " + str + "\n";

// логотип

const string PATH\_TO\_LOGO\_FOLDER = "assets/logo\_slides/";

const string PATH\_TO\_LOGO = PATH\_TO\_LOGO\_FOLDER + "logo.txt";

// строка слева от введенной команды

const string DBMS\_CONSOLE\_REQUEST\_COMMAND = "\n[СУБД v0.1]> ";

// справка о программе

const string DBMS\_DESCRIPTION =

"СУБД была разработана Плоцким Богданом в качестве Курсовой Работы\n\

После появления строки " +

DBMS\_CONSOLE\_REQUEST\_COMMAND +

"начинайте вводить команды\n\

Если неизвестны команды, введите \"помощь\"\n";

// сообщение об ошибочном вводе команды

const string NOT\_CORRECT\_COMMAND = " такой команды не существует\n\

Если неизвестны команды, введите: помощь\n\

Если не знаете, как пользоваться командой, \

введите: помощь <неизвестная команда>\n";

// сообщение о некорректных данных

const string NOT\_CORRECT\_DATA = "-";

// путь до ппаки с базами данных

const string DB\_FOLDER\_PATH = "assets/databases/";

// путь до файла с базой данных

const string DB\_FILE\_PATH = DB\_FOLDER\_PATH + "db.txt";

// путь жо файла с выводом БД

const string OUT\_DB\_FILE\_PATH = DB\_FOLDER\_PATH + "db\_out.txt";

// заполнение len элементов элементом symb

#define OUT\_W(symb, len) fixed << setfill(symb) << setw(len)

// получение токена из строки и проерка этой строки

#define GET\_DATA(str) IsStringNotEmpty(GetToken(str,':'))

#define GET\_INT\_DATA(str) atoi(GET\_DATA(str).c\_str());

// количество команд

const int NUMBER\_OF\_COMMANDS = 10;

// ПРИ ДОБАВЛЕНИИ КОМАНДЫ НАДО ЕЩЕ ДОБАВИТЬ

// В COMMAND\_CHECK\_FUNCTIONS УКАЗАТЕЛИ НА СОТВЕТСТВУЮЩИЕ

// ФУНКЦИИ В КОНСТРУКТОРЕ Menu()

// массив с файлами, в которых лежит описание команд

const string COMMAND\_DESCRIPTION\_FILES[NUMBER\_OF\_COMMANDS] =

{

"exit.txt",

"help.txt",

"read.txt",

"print.txt",

"save.txt",

"delete.txt",

"clear.txt",

"select.txt",

"replace.txt",

"sort.txt"

};

// Количество полей в базе данных

const int NUMBER\_OF\_FIELDS = 8;

// названия полей

const string NAMES\_OF\_FIELDS[NUMBER\_OF\_FIELDS] =

{

"НОСИТЕЛЬ",

"ПОРЯДКОВЫЙ НОМЕР",

"НАЗВАНИЕ ПЕСНИ",

"ИМЯ ИСПОЛНИТЕЛЯ",

"ФАМИЛИЯ ИСПОЛНИТЕЛЯ",

"ВРЕМЯ ЗВУЧАНИЯ",

"КОЛИЧЕСТВО ВОСПРОИЗВЕДЕНИЙ",

"ЦЕНА",

};

// шапка таблицы

const string TABLE\_CAP =

"| " + NAMES\_OF\_FIELDS[1] +

" | " + NAMES\_OF\_FIELDS[0] +

" | " + NAMES\_OF\_FIELDS[2] +

" | " + NAMES\_OF\_FIELDS[3] +

" | " + NAMES\_OF\_FIELDS[4] +

" | " + NAMES\_OF\_FIELDS[5] +

" | " + NAMES\_OF\_FIELDS[6] +

" | " + NAMES\_OF\_FIELDS[7] +

" |\n";

// ширина полей таблицы при выводе

const int FIELDS\_WIDTH[9] =

{

NAMES\_OF\_FIELDS[1].length() + 6,

NAMES\_OF\_FIELDS[0].length() + 6,

NAMES\_OF\_FIELDS[2].length() + 7,

NAMES\_OF\_FIELDS[3].length() + 10,

NAMES\_OF\_FIELDS[4].length() + 6,

NAMES\_OF\_FIELDS[5].length() + 6,

NAMES\_OF\_FIELDS[6].length() + 6,

NAMES\_OF\_FIELDS[7].length() + 6,

};

// количество функций сортировки

const int NUMBER\_OF\_SORTS = 2;

// названия сортировок

const string NAMES\_OF\_SORTS[2] =

{

"по убыванию",

"по возрастанию"

};

// количество функций сравнения

const int NUMBER\_OF\_COMPARISONS = 5;

// названия функций сравнения

const string NAMES\_OF\_COMPARISONS[NUMBER\_OF\_COMPARISONS] =

{

">",

"<",

"==",

"<=",

">=",

};

// размер массива для типов ввода информации

const int INFO\_ENTER\_SIZE = 2;

// способ ввода информации

const string INFO\_ENTER[INFO\_ENTER\_SIZE] =

{

"консоль",

"файл"

};

// размера массива с инфой для файлового ввода/вывода

const int FILE\_PATH\_SIZE = 2;

// для функции ввода информации из файла

// и печати в файл

const string FILE\_PATH[FILE\_PATH\_SIZE] =

{

"использование файла по умолчанию",

"ввести свой путь до файла"

};

// размер массива с сообщениями для DELETE\_NODE

const int DELETE\_NODE\_SIZE = 2;

// для функции удаления элементов из списка

const string DELETE\_NODE[DELETE\_NODE\_SIZE] =

{

"удаление по индексу (один элемент)",

"удаление по значению поля (несколько элементов)"

};

## kursovaya-2022.cpp

Главный файл проекта. Здесь находится функция main, откуда все запускается.

#include "menu/menu.h"

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include <Windows.h> // для считывания кириллицы

int main()

{

setlocale(LC\_ALL, "ru");

// для считывания кириллицы

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Menu mn;

mn.ProgramMenu();

return 0;

}

# Результат работы

## Файлы

1. Файл использованный в качестве исходной базы данных содержал следующие данные:

1:интернет:Плакала:Леша:Савик:2:122:25

2:интернет:Все мимо:Artik & Asti::1:255:12

3:диск:MoLoko:LOBODA::3:1:0

4:диск:Frendly Fire:Boulvard:Depo:2:500:3

5:жесткий диск:права:ooes::2:600:45

6:флешка:МАГИЯ:3TERNITY::1:698:56

7:интернет:Антигерой:Mnogoznaal::2:456:32

8:диск:Рассвет:КАКАЯ:РАЗНИЦА:2:8971:54

9:флешка:Ни много ни мало:Boulvard:Depo:2:78651:123

10:флешка:YuNg BrAtZ:XXXTentacion::1:3215:254

11:диск:Cause You Are Young:C.C.:Catch:3:456:12345

12:интернет:Lonely World:Brennan:Savage:1:12346:321

1. Файл, в который база данных выгрузила полученный результат после выполнения работы СУБД выглядел так:

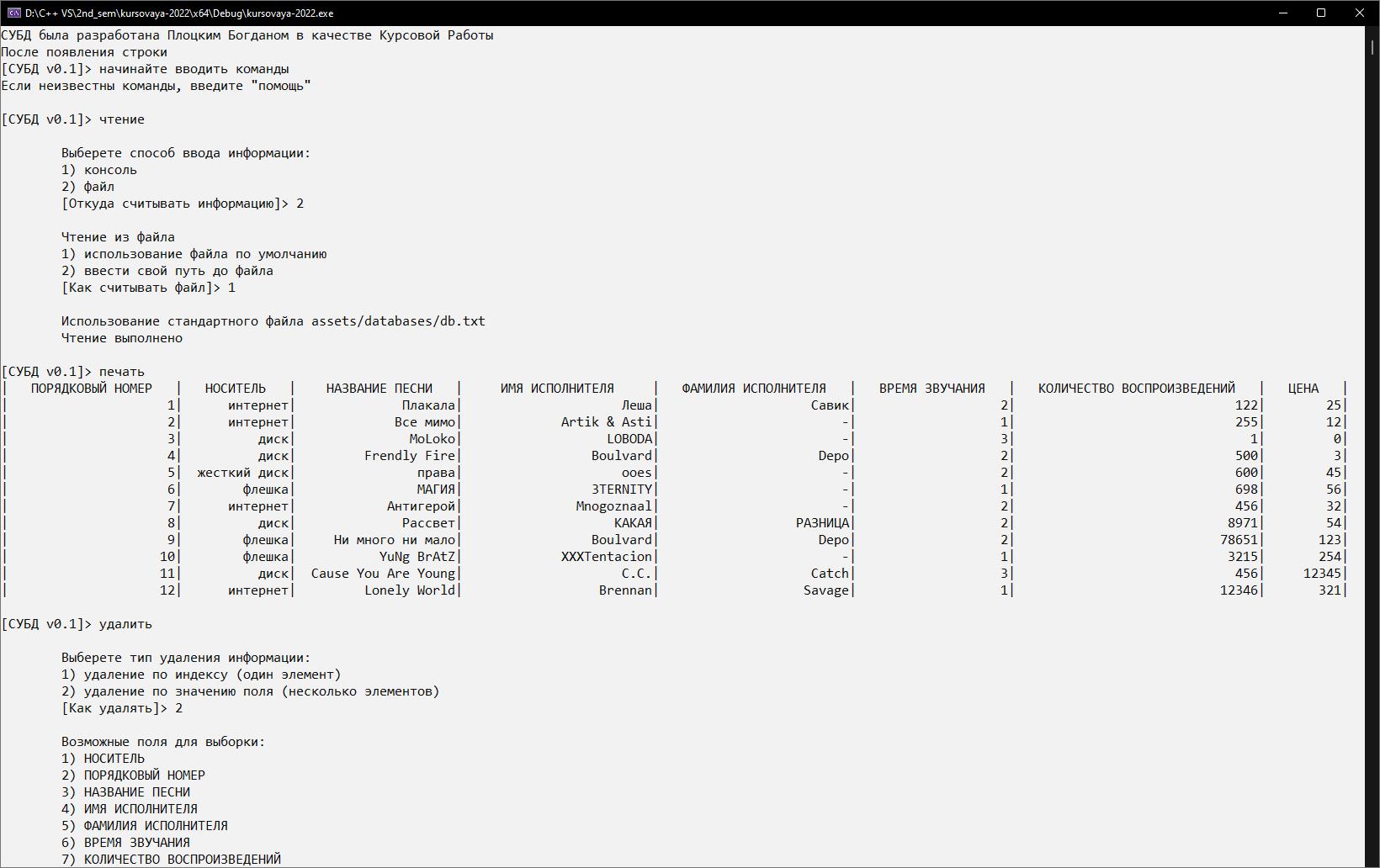
1:интернет:Антигерой:Mnogoznaal:-:2:456:32

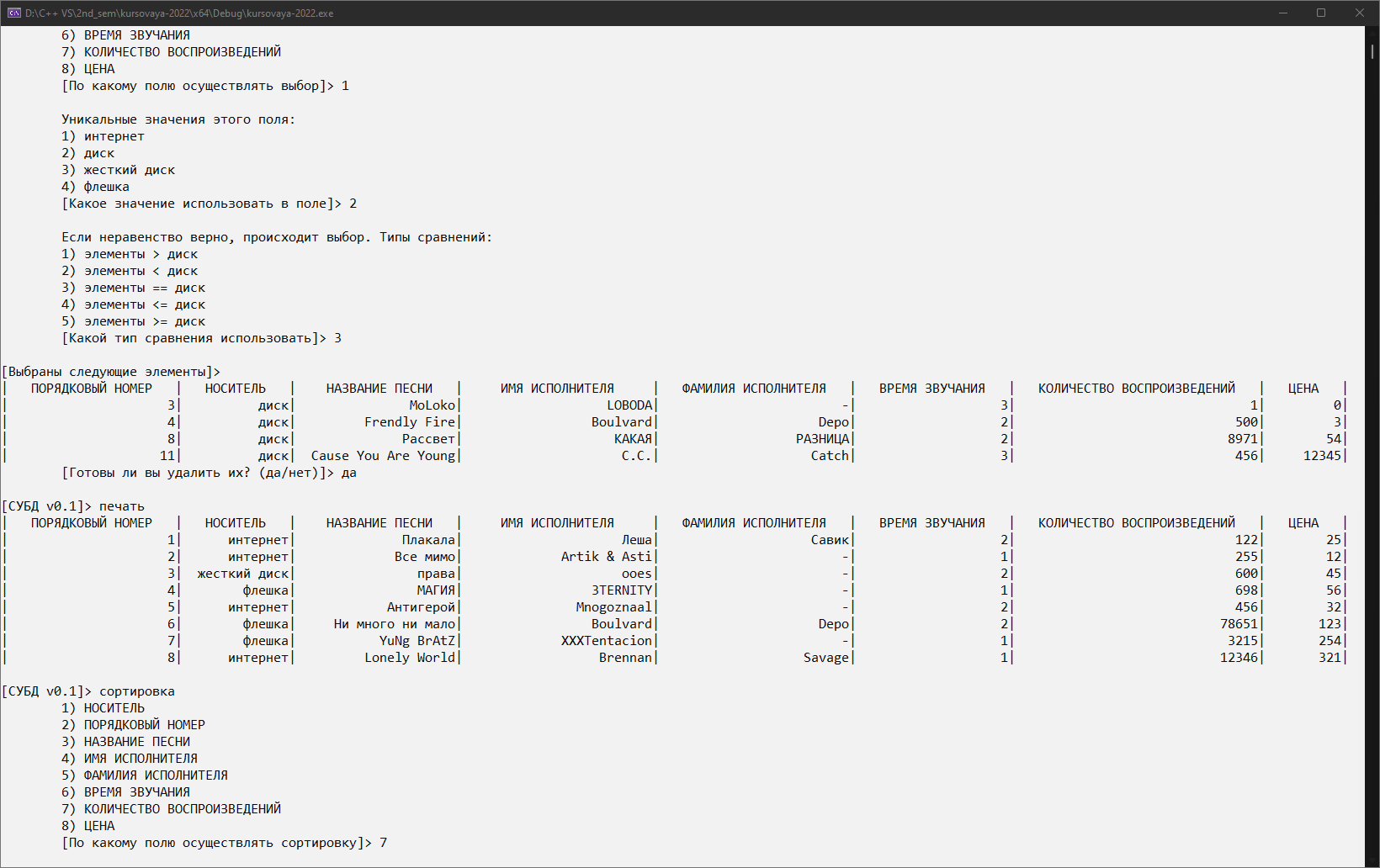
2:флешка:МАГИЯ:3TERNITY:-:1:698:56

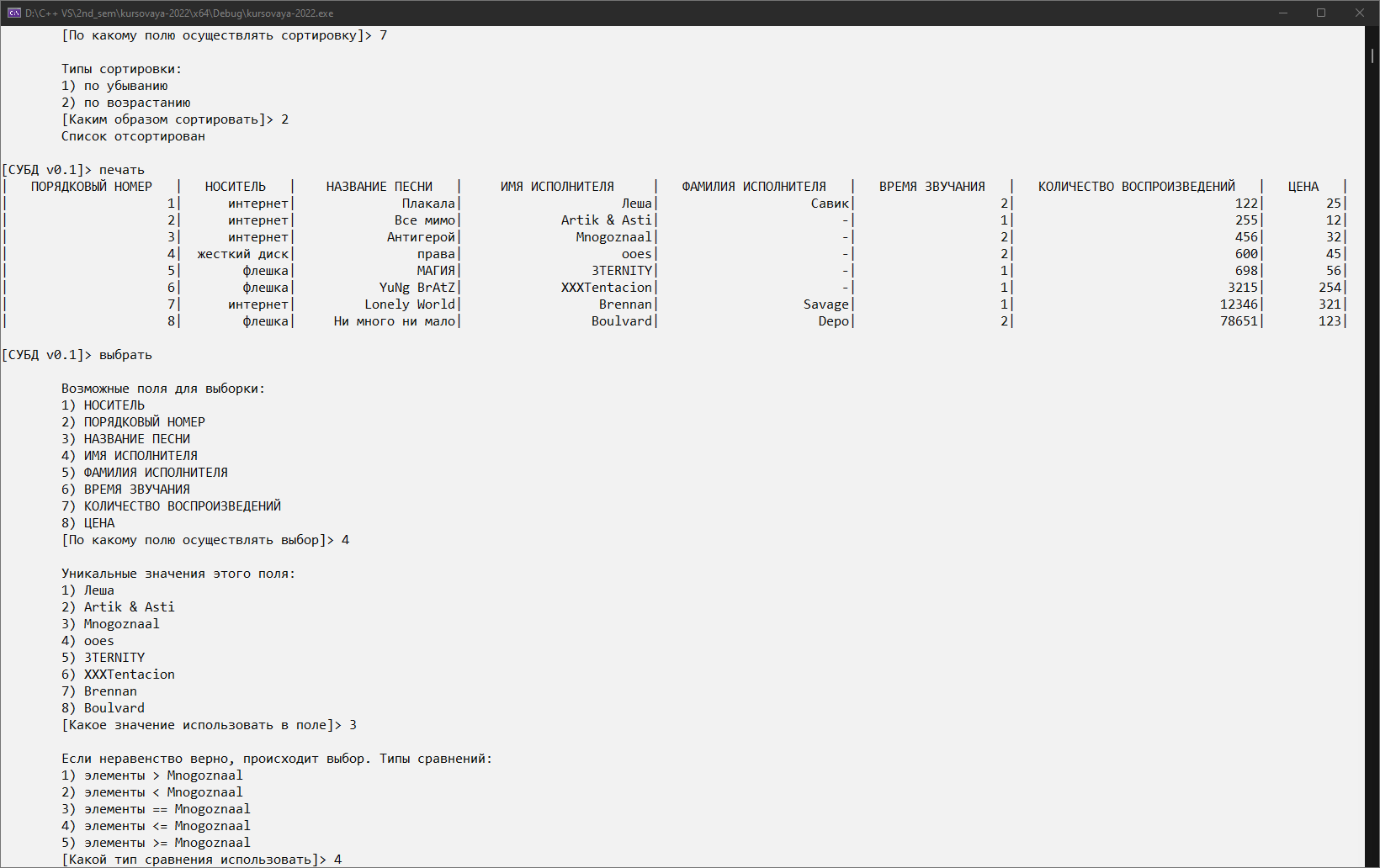
3:интернет:Lonely World:Brennan:Savage:1:12346:321

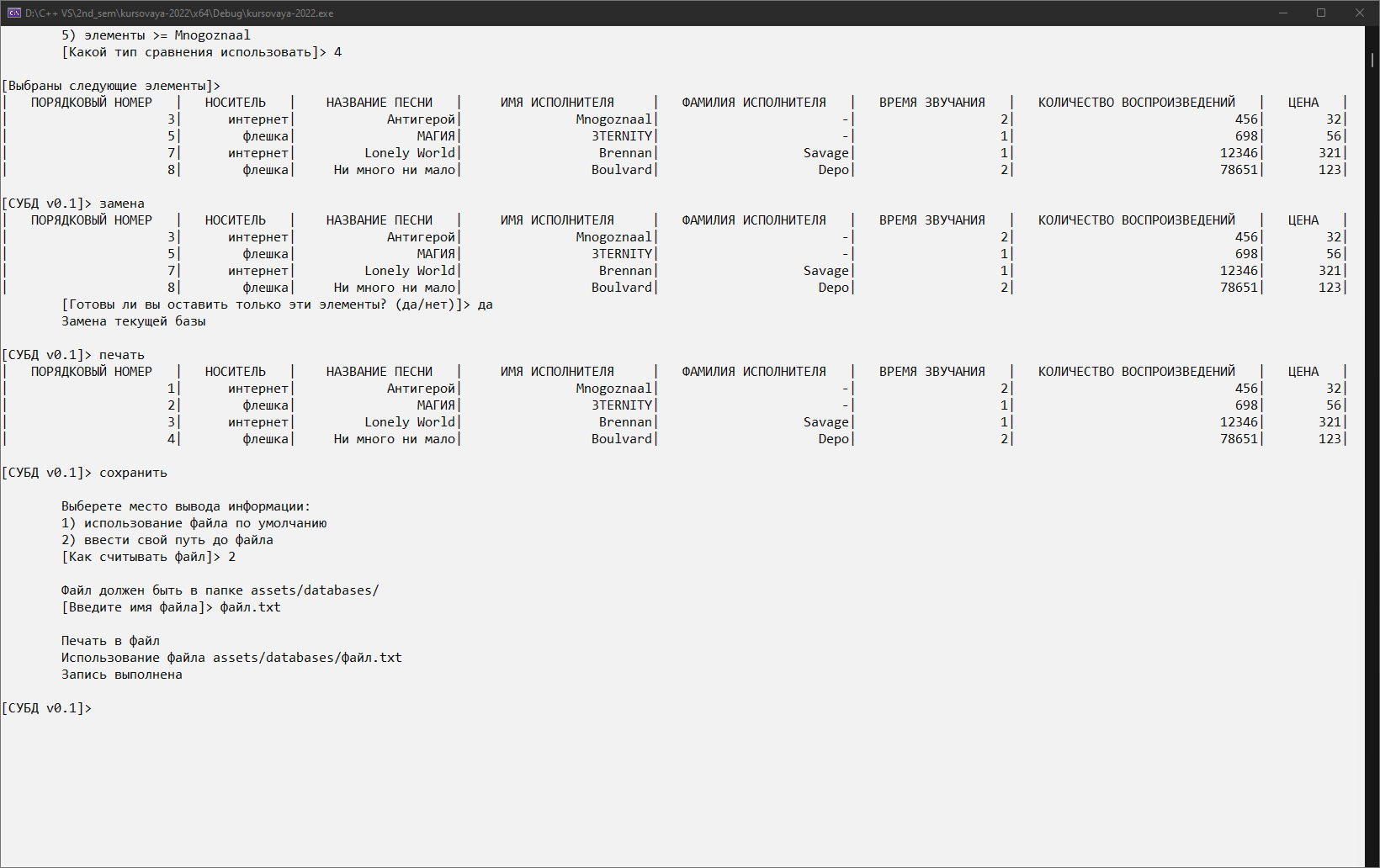
4:флешка:Ни много ни мало:Boulvard:Depo:2:78651:123

## Скриншоты работы программы









# Вывод

В ходе разработки программы было использовано ООП, что сильно упростило задачу. С точки зрения понимания строения программы, благодаря ООП программы делится на отдельные «блоки», каждый из которых выполняет свою задачу, что в совокупности делает возможным существование этой СУБД. После длительного процесса разработки, данная СУБД научилась считывать данные из файла и из консоли, выводить их на консоль, выгружать базу данных в файл, редактировать саму базу данных, то есть, удалять какие-то элементы из нее по одному или сразу несколько, выбирая по значению поля, выбирать элементы базы данных по определенному полю, а также определенному значению этого поля. База данных была разработана в соответствии с вариантом номер 10.