*Институт Транспорта и связи*

**Курсовая работа**

По дисциплине

«Программирование»

## Тема: «База данных для игрового магазина»

Студент:

Орлов Денис

Группа: 4501BD

Рига

2016 г.

**Содержание**

Оглавление

[Тема: База данных 1](#_Toc451192490)

[**Введение** 3](#_Toc451192491)

[**1.** **Задание на курсовую работу** 4](#_Toc451192492)

[**2.** **Анализ предметной области** 5](#_Toc451192493)

[**3.** **Проектирование** 7](#_Toc451192494)

[**4.** **Кодирование** 8](#_Toc451192495)

[**5.** **Тестирование** 9](#_Toc451192496)

[**6.** **Результаты работы программы** 10](#_Toc451192497)

[**7.** **Заключение** 14](#_Toc451192498)

[**8.** **Список литературы** 16](#_Toc451192499)

[**9.** **Приложение (код программы)** 17](#_Toc451192500)

**Введение**

Данная курсовая работа посвящена разработке программы на языке с++ для игрового магазина, программа будет отображать данные об игре и издателе, а также программа даёт возможность редактировать, добавлять и удалять информацию. Курсовая работа содержит девять этапов разработки, для данного программного обеспечения они указаны ниже:

*Первый*: содержит задание на курсовую работу, в которой указаны условия для выполнения поставленной задачи.

*Второй*: содержит анализ предметной области, где была проанализирован предметная область, в которой нужно работать.

*Третий*: Содержит проектирование, в котором объясняется что делает тот или иной пункт меню.

*Четвертый*: содержит кодирование где объясняется принцип основных функций, а именно: функции добавления, удаления, сортировки и поиска.

*Пятый*: в данном разделе идёт тестирование, в нем показан анализ ошибок и проблем, которые возникли во время написания программы.

*Шестой*: Результаты работы программы, этот раздел содержит рисунки работы программы.

*Седьмой*: Заключение, в этом разделе подводится итог всей проведённой работы

*Восьмой*: Список литературы, в данном разделе находится ссылки, на веб-сайты, которые были использованы в процессе разработки данной программы.

*Девятый*: Приложение (код программы), в этом разделе содержится исходный код программы.

1. **Задание на курсовую работу**

Задание заключается в следующем, надо будет создать приложение, которое позволит хранить и обрабатывать информацию для конкретной предметной области. Надо будет определить данных которые будут храниться, а также определить их типы. Нужно будет написать такие функции как добавление, сортировка, отображение данных, удаление данных и фильтрация данных методом пузырька. Требуется разработать структуру приложения, а также пользовательский интерфейс – представление того, каким образом будут отображаться данные. Нужно разработать алгоритмы, которые будут выполнять основные функции программы.

1. Выбрать предметную область и проанализировать ее. Определить объекты, о которых требуется хранить информацию и их характеристики. Определить данные, которые будут сохраняться, определить их типы, ограничения и т.д. Примеры предметных областей и объектов см. в п. 7. В разрабатываемой системе должно быть 2-3 объекта.

2. Определить функции, которыми должна быть наделена система. Основными функциями системы должны быть:

• добавление данных;

• отображение данных (по одному объекту, по несколько объектов, в виде таблицы);

• удаление данных об объектах (удаление по указанному критерию);

• поиск и фильтрация данных (возможность задания разных критериев, минимум 3 критерия);

• подсчёт каких-либо суммарных характеристик, например, количество записей с определённым значением какого-либо поля (минимум 2 характеристики);

• сортировка (по нескольким критериям). Определить критерии поиска, сортировки, удаления.

3.Разработать структуру приложения. Разработать структуры данных – struct. Для каждого объекта своя структура. Для каждого объекта создаётся отдельный бинарный файл.

4. Разработать пользовательский интерфейс – представление того, каким образом будут отображаться данные, каким образом будет выбираться то или иное действие с данными, меню и т.п.

5. Разработать алгоритмы, выполняющих основные функции программы (поиск, суммарные характеристики, удаление, сортировка и т.д.)

Написать код программы

Протестировать программу

Написать отчёт

1. **Анализ предметной области**

В качестве предметной области была выбрана база данных игр. В программе должны быть реализованы следующие функции: Поиск по названию игры, удаление по названию игры, поиск по имени издателя, фильтр по цене, общая сумма всех игр в базе, сортировка.

Проанализировав данную область, стало ясно что разработка данной программы потребует много времени и сил, так как содержит очень много информации об играх и об издателях этих игр. Для удобства работы, программа должна иметь возможность хранить следующие данные:

* «Название игры»
* «цена»
* «Издатель»
* «Игровая платформа»
* «Количество игроков»
* «Дата выхода»
* «Жанр»
* «Количество на складе»

Каждой игре и издателю будет автоматически присвоен уникальный код товара (далее идентификатор). При поиске по идентификатору, сразу же будет выдана вся информация согласно веденному коду. Если пользователь захочет вести новую игру в базу, то он должен позаботиться о том, если ли идентификатор издателя той игры, которую он хочет добавить.

1. **Проектирование**

**Предварительное проектирование.**

На данном этапе разработки, было продуманно:

* «Количество необходимых бинарных файлов»
* «Оформление пользовательского меню»

Данные с которыми должна работать программа, должны обязательно хранится в разных файлах:

* «База игр»
* «База издателей»

База игр должна хранить данные об играх, которые приведены ниже:

* «Название игры»
* «цена»
* «Идентификатор издателя»
* «Игровая платформа»
* «Количество игроков»
* «Дата выхода»
* «Жанр»
* «Количество на складе»

Поскольку в базе игр хранится идентификатор издателя, то не будет необходимости водить всю информацию об, так как идентификатор содержит всю ранее веденные информацию в базе данных издателей, а именно:

* «Имя издателя»
* «Дата основания компании»
* «Страна издателя»

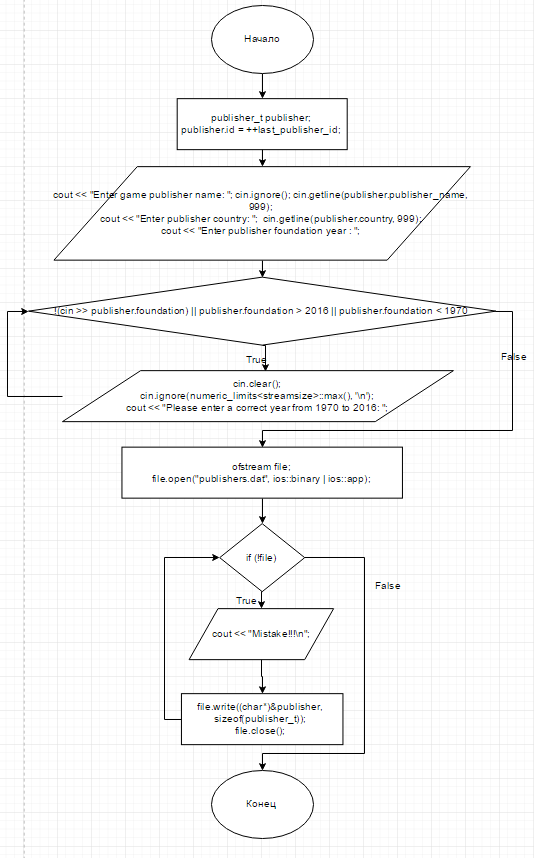
Пользовательский интерфейс представляет из себя строчное меню.

Главное меню должно содержать:

1. «Добавление издателя»
2. «Показ издателя»
3. «Удаление издателя»
4. «Поиск»
5. «Добавление игры»
6. «Показ игры»
7. «Удаление игры»
8. «Выход из программы»

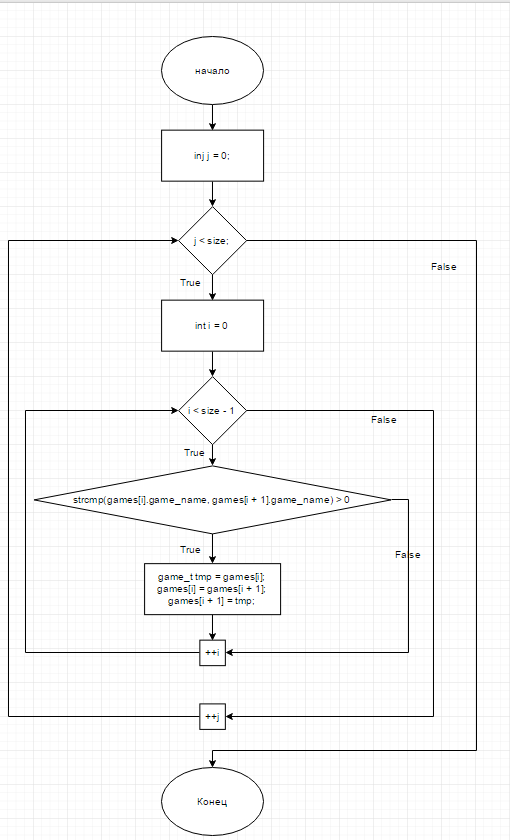
**Детальное проектирование.**

На данном этапе были разработаны различные функции, всего программа содержит 24 функции, но самые основные из них это добавление, удаление, сортировка и поиск. Именно о них и пойдёт речь. Ниже приведены их алгоритмы.



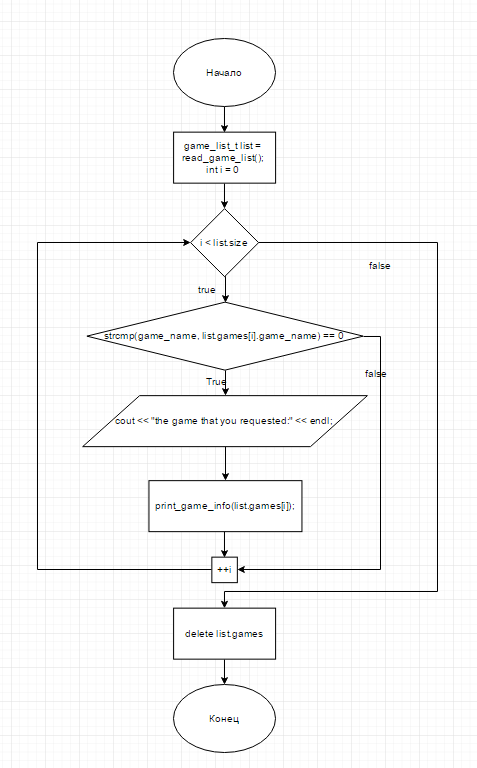
Алгоритм 1 Добавление

Далее приведен алгоритм сортировки методом пузырька(Алгоритм 2)



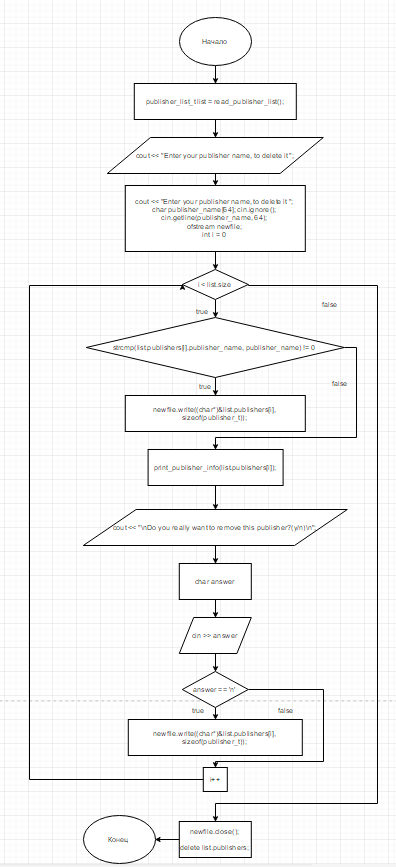
Алгоритм 2 сортировка

На алгоритме 3 можно увидеть поиск



Алгоритм 3 Поиск

На алгоритме 4 мы видим удаление



Алгоритм 4 удаление

1. **Кодирование**

Для использования некоторых функций для приложения требуется подключить библиотеки, которые указаны ниже:

* <iostream> - Операции ввода и вывода
* <fstream> - работа с файлами
* <limits> - Проверяет корректность ввода значения от пользователя
* <algorithm> - Включает набор функций для выполнения алгоритмических операций.

# Для реализации программы использовались два бинарных файла. Оба файла создаются вовремя добавления пользователем данных в базу. И все последующие, обращения происходят к уже существующим файлам.

# Было создано 2 структуры. Структура game\_t содержит в себе информацию о 9 данных, а структура publisher\_t о 4 данных. Ниже рассмотрены обе структуры с названиями данных и типов.

Структура game\_t:

* char game\_name[64];
* float price;
* int publisher\_id;
* char platform[16];
* uint players;
* uint release\_date;
* char genre[64];
* uint amount;
* int id;

Структура publisher\_t:

* char publisher\_name[64];
* uint foundation;
* char country[64];
* int id;

Была объявлены две глобальные переменные:

* int last\_game\_id = 0;

# int last\_publisher\_id = 0;

А также тип unsigned int был заменен на uint для большего удобства работы с программой. (typedef unsigned uint;)

**Список функций**

В данной работе использовано 24 функции, главной которой стала конечно же функция «main». Функция «main» начинает свою работу самая первая и её задача объединить все функции в одно приложение. Ниже в таблице 1. Представлен весь перечень функций.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип функции | Название | Параметры функции | Возвращаемый результат | значение |
| void | add\_game() |  |  | Записывает данные игры в games.dat |
| void | add\_publisher() |  |  | Записывает данные издателя в publishers.dat |
| void | bubble\_sort\_name() | game\_t games[], int size |  | Сортировка методом пузырька по имени игры |
| void | bubble\_sort\_price() | game\_t games[], int size |  | Сортировка методом пузырька по цене игры |
| void | count\_copies\_by\_publisher() | int publisher\_id |  | Показывает оставшиеся копии игр от данного издателя |
| void | count\_price\_by\_genre() | char genre[64] |  | Считает общую сумму выручки по жанру игры |
| void | delete\_game() |  |  | Удаляет игру по веденному имени |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| void | delete\_publisher() |  |  | Удаляет издателя по веденному имени |
| void | filter\_by\_price() | Float max\_price |  | Сортирует игры по цене |
| void | greetings() |  |  | Приветствие программы |
| void | handle\_events() |  |  | Основное меню программы |
| int | main() |  | 0 | Главная функция |
| bool | ok\_publisher\_id() | int publisher\_id | Ok\_id | Проверяет есть ли веденный id в базе |
| void | print\_game\_info() | game\_t game |  | Вывод на экран информацию об игре |
| void | print\_publisher\_info() | publisher\_t publisher |  | Вывод на экран информацию об издателе |
| game\_list\_t | read\_game\_list() |  | result | Считывает файл games.dat |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| publisher\_list\_t | read\_publisher\_list() |  | result | Считывает файл publishers.dat |
| record\_list\_t | read\_record\_list() | char\* file\_name, int record\_size | result |  |
| void | search\_by\_name() | char game\_name[64] |  | Поиск по имени игры |
| void | searcg\_by\_publisher() | int publisher\_id |  | Поиск по id издателя |
| void | search\_menu() |  |  | Меню поиска |
| void | show\_games() |  |  | Показ всех игр |
| void | show\_publishers() |  |  | Показ всех издателей |
| void | sort\_by\_name\_in\_show() |  |  | Сортировка по имени в функции show |
| void | sort\_by\_price\_in\_show() |  |  | Сортировка по цене в функции show |

1. **Тестирование**

Тестирование проводилось во время написания программы, а также и после. Ниже будут приведены наиболее часто выявляемые ошибки.

**Add game**🡪Проводились проверки по добавлению данных, серьёзных проблем замечено не было, была лишь одна проблема с getline, программа никак не хотела читать строку через пробел, но решение нашлось, вперед getline надо было поставить cin.ignore(); тогда программа заработала корректно.

**Show games**🡪 Данные во все случаях отображались верно, но было одно но. В getline было выделено очень мало памяти, поэтому программа выводила помимо строки которой вел пользователь, ещё и непонятные символы, которые как предполагаются, это был мусор. Решение было очень простое, нужно было просто увеличить количество памяти, и всё прекрасно заработало.

**Remove**🡪Проводилось тестирование удаление игры по имени. Всё работало верно, никаких проблем выявлено не было.

**Search**🡪Проводились поиски по 5-ти критериям, а именно: по «имени игры», «издателю», «максимальной цене», «по копиям на складе», «и по общей сумме игр». Все поиски работали верно, проблем выявлено не было.

**Show publishers**🡪Программа показывает всех добавленных издателей, а также присваивает им все уникальный id, начиная с цифры 1. Проблем с показом данных не возникло.

**Add publishers**🡪Было проведено множество проверок добавления издателя, Никаких проблем с добавлением, обнаружено не было.

**Exit**🡪 Функция работает корректно ошибок выявлено не было.

**Другие ошибки🡪** Когда пользователь пытался ввести в поле где надо водить цифры букву, программа на это никак не реагировала. Ошибка устранена, теперь программа проверяет то что вводит пользователь. В программе не было задано ограничения для года, то есть пользователь мог ввести как 900 год, так и 3000 год, ошибка устранена.

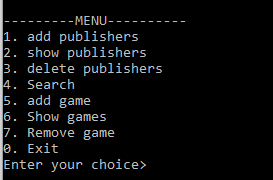
1. **Результаты работы программы**

При запуске программы пользователь увидит надпись, показанную на рис. 1



**Рис. 1 Приветствие программы**

В программе представлен очень просто интерфейс для большей ясности работы, интерфейс представлен на рис. 2



**Рис. 2 Главное меню программы**

Прежде чем начать работать в программе, пользователь должен ввести в базу данных информацию об издателе, как это делается показано на рис. 3

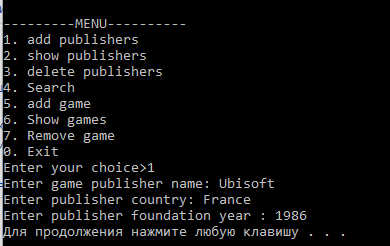


Рис. 3 добавление издателя.

После этого пользователь может начать добавлять игры, указывая просто id издателя (рис. 4)



Рис. 4 добавление игры

После того как игра была добавлена, мы можем отсортировать её по трем критериям, а именно: По алфавиту, по цене, и показать без сортировки. (рис .5)

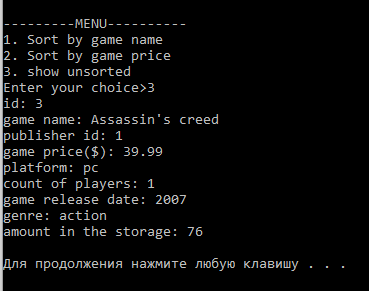


Рис. 5 показ игры

Удаление игры происходит по её имени, после того как ввели имя, программа попросит точно, нужно ли удалить данную игру (рис. 6)

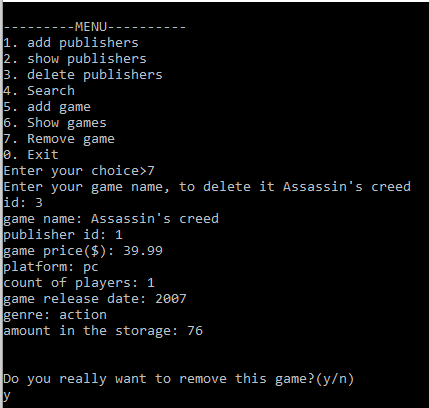


Рис. 6 удаление игры

Если нам нужно найти игру или игры от определенного издателя, то достаточно просто вести его id и мы получим весь список игр от нужного нам издателя (рис. 7)

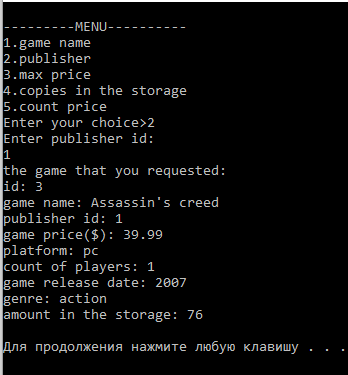


Рис. 7 поиск по id издателя

На рис. 8 представлено окно показа издателя.

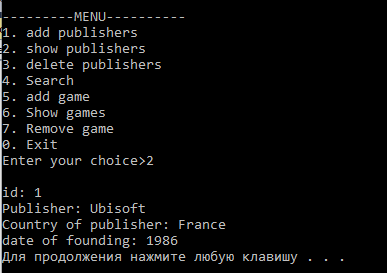


Рис. 8 показ издателя

На рис. 9 показан выход из программы.

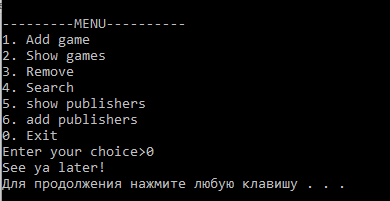


Рис. 9 выход из программы

1. **Заключение**

Данная работа была посвящена разработке базе данных игр.

В ходе разработки системы было сделано следующее: Была написана база данных, с двумя 2 объектами. Первый объект — это сами игры, точнее информация о них. Вторым объектом является информация об издателе. В ходе работы над системой столкнулись с рядом проблем, а именно: Когда пользователь пытался ввести в поле где надо водить цифры букву, программа на это никак не реагировала. Ошибка устранена, теперь программа проверяет то что вводит пользователь. В программе не было задано ограничения для года, то есть пользователь мог ввести как 900 год, так и 3000 год, ошибка устранена.

В итоге получилась программа, которая реализует все поставленные перед ней задачи. Программа учитывает ошибки ведёные пользователем, если пользователь ведет неправильное значение, например в поле так где надо водить цифру, ведет букву, то программа сообщит об этом и попросит пользователя исправить свою ошибку, также в программе проверяется правильность года, то есть нельзя написать год который ещё не наступил, например 2087. Учитывая то что каждая игра и издатель имеют уникальный идентификатор, повторное ведение идентификатора который уже существует в базе данных не будет позволено программой. Также была ведена система поиска которая ищет по нескольким критериям, например, по «имени игры», «издателю», «максимальной цене», «по копиям на складе», «и по общей сумме игр».

В ходе разработки программы, было написано 25 функций, большинство из них нужны просто что бы сократить код программы и сделать его более читабельным, например у нас была строка кода которая использовалась несколько раз в программе, а писать каждый раз строку кода состоящую из 30 строк не очень хотелось, было решено вынести эту строку в отдельную функции и когда она будет нужна, просто обращаться к ней через функцию. В ходе работы был ряд проблем, которые были устранены в дальнейшем.

Были сделаны следующие выводы: данная работа позволила получить новые знания и закрепить старые, полученные на протяжении двух семестров. Также стало совсем неожиданным, что, казалось бы, легкая программа, а написать её понадобилось столько времени. Было исправлено очень много ошибок, а бывало даже, что если исправляешь одну ошибку, то появляется другая, было очень сложно, но за это был получен бесценный опыт, который поможет в дальнейшем.

1. **Список литературы**

За основу была взята третья лабораторная работа.

Были использованы следующие сайты🡪

http://stackoverflow.com/

http://www.cplusplus.com/

1. **Приложение (код программы)**

#include<iostream>

#include<fstream>

#include<limits>

#include <algorithm>

using namespace std;

typedef unsigned uint;

struct publisher\_t {

char publisher\_name[64];

uint foundation;

char country[64];

int id;

};

struct game\_t

{

char game\_name[64];

float price;

int publisher\_id;

char platform[16];

uint players;

uint release\_date;

char genre[64];

uint amount; // amount of copies in shop storage

int id;

};

// for internal use

struct record\_list\_t {

void\* records = 0;

int size = 0;

};

// for internal use

struct game\_list\_t {

game\_t\* games = 0;

int size = 0;

};

// for internal use

struct publisher\_list\_t {

publisher\_t\* publishers = 0;

int size = 0;

};

void greetings();

void add\_game();

void show\_publishers();

void show\_games();

void delete\_game();

void delete\_publisher();

void handle\_events();

void search\_menu();

game\_list\_t read\_game\_list();

publisher\_list\_t read\_publisher\_list();

// Global variables.

int last\_game\_id = 0;

int last\_publisher\_id = 0;

int main()

{

greetings();

handle\_events();

system("pause");

return 0;

}

void print\_publisher\_info(publisher\_t publisher)

{

cout << "\n";

cout << "id: " << publisher.id << "\n"

<< "Publisher: " << publisher.publisher\_name << "\n"

<< "Country of publisher: " << publisher.country << "\n"

<< "date of founding: " << publisher.foundation << endl;

}

void print\_game\_info(game\_t game)

{

cout << "id: " << game.id << "\n"

<< "game name: " << game.game\_name << "\n"

<< "publisher id: " << game.publisher\_id << "\n"

<< "game price($): " << game.price << "\n"

<< "platform: " << game.platform << "\n"

<< "count of players: " << game.players << "\n"

<< "game release date: " << game.release\_date << "\n"

<< "genre: " << game.genre << "\n"

<< "amount in the storage: " << game.amount << endl;

cout << endl;

}

void greetings()

{

cout << "Database of games version 1.0 " << endl;

system("pause");

}

void add\_publisher()

{

publisher\_t publisher;

publisher.id = ++last\_publisher\_id;

cout << "Enter game publisher name: "; cin.ignore(); cin.getline(publisher.publisher\_name, 999);

cout << "Enter publisher country: "; cin.getline(publisher.country, 999);

cout << "Enter publisher foundation year : ";

while (!(cin >> publisher.foundation) || publisher.foundation > 2016 || publisher.foundation < 1970) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Please enter a correct year from 1970 to 2016: ";

}

ofstream file;

file.open("publishers.dat", ios::binary | ios::app);

if (!file) cout << "Mistake!!!\n";

else

{

file.write((char\*)&publisher, sizeof(publisher\_t));

file.close();

}

}

// prints the info about publisher and asks

// user to confirm it is the publisher he wants

bool ok\_publisher\_id(int publisher\_id)

{

publisher\_list\_t list = read\_publisher\_list();

bool ok\_id = false;

for (int i = 0; i < list.size; ++i) {

publisher\_t publisher = list.publishers[i];

if (publisher\_id == publisher.id)

{

print\_publisher\_info(publisher);

ok\_id = true;

}

}

delete list.publishers;

return ok\_id;

}

void add\_game()

{

game\_t game;

game.id = ++last\_game\_id;

cout << "Enter publisher id: ";

while (!(cin >> game.publisher\_id) || !ok\_publisher\_id(game.publisher\_id)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input. Try again: ";

}

cout << "Enter your game name: "; cin.ignore(); cin.getline(game.game\_name, 999);

cout << "Enter game price: ";

while (!(cin >> game.price)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid price. Try again: ";

}

cout << "Enter game platform: "; cin.ignore(); cin.getline(game.platform, 999);

cout << "Enter count of players: ";

while (!(cin >> game.players)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input. Try again: ";

}

cout << "Enter game release date: ";

while (!(cin >> game.release\_date) || game.release\_date > 2016 || game.release\_date < 1947) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Please enter a correct year from 1947 to 2016: ";

}

cout << "Enter game genre: "; cin.ignore(); cin.getline(game.genre, 999);

cout << "Enter your game copies amount in storage: ";

while (!(cin >> game.amount)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input. Try again: ";

}

ofstream file;

file.open("games.dat", ios::binary | ios::app);

if (!file) cout << "Mistake!!!\n";

else

{

file.write((char\*)&game, sizeof(game\_t));

file.close();

}

}

void delete\_game()

{

game\_list\_t list = read\_game\_list();

cout << "Enter your game name, to delete it ";

char game\_name[64]; cin.ignore(); cin.getline(game\_name, 64);

ofstream newfile;

newfile.open("games.dat", ios::binary);

for (int i = 0; i < list.size; i++)

{

if (strcmp(list.games[i].game\_name, game\_name) != 0)

newfile.write((char\*)&list.games[i], sizeof(game\_t));

else

{

print\_game\_info(list.games[i]);

cout << "\nDo you really want to remove this game?(y/n)\n";

char answer; cin >> answer;

if (answer == 'n')

newfile.write((char\*)&list.games[i], sizeof(game\_t));

}

}

newfile.close();

delete list.games;

}

void delete\_publisher()

{

publisher\_list\_t list = read\_publisher\_list();

cout << "Enter your publisher name, to delete it ";

char publisher\_name[64]; cin.ignore(); cin.getline(publisher\_name, 64);

ofstream newfile;

newfile.open("publishers.dat", ios::binary);

for (int i = 0; i < list.size; i++)

{

if (strcmp(list.publishers[i].publisher\_name, publisher\_name) != 0)

newfile.write((char\*)&list.publishers[i], sizeof(publisher\_t));

else

{

print\_publisher\_info(list.publishers[i]);

cout << "\nDo you really want to remove this publisher?(y/n)\n";

char answer; cin >> answer;

if (answer == 'n')

newfile.write((char\*)&list.publishers[i], sizeof(publisher\_t));

}

}

newfile.close();

delete list.publishers;

}

void handle\_events()

{

int choice = -1;

while (choice)

{

system("cls");

cout << "\n---------MENU----------\n";

cout << "1. add publishers\n2. show publishers\n3. delete publishers \n4. Search\n5. add game\n6. Show games\n7. Remove game\n0. Exit\n";

cout << "Enter your choice>";

cin >> choice;

switch (choice) {

case -100:

break; case 1: add\_publisher();

break; case 2: show\_publishers();

break; case 3: delete\_publisher();

break; case 4: search\_menu();

break; case 5: add\_game();

break; case 6: show\_games();

break; case 7: delete\_game();

break; case 0: cout << "See ya later!\n";

}

system("pause");

}

}

// need to free memory, after using it!

record\_list\_t read\_record\_list(char\* file\_name, int record\_size)

{

record\_list\_t result;

ifstream file;

file.open(file\_name, ios::binary);

if (!file) cout << "\n\nThere is some error with your file, please try again!\n\n";

else

{

file.seekg(0, ios::end);

int size = file.tellg();

size = size / record\_size;

file.seekg(0, ios::beg);

void\* records = malloc(size \* record\_size);

file.read((char\*)records, size \* record\_size);

file.close();

result.records = records;

result.size = size;

}

return result;

}

// need to free memory, after using it!

game\_list\_t read\_game\_list()

{

game\_list\_t result;

record\_list\_t records = read\_record\_list("games.dat", sizeof(game\_t));

result.games = (game\_t\*)records.records;

result.size = records.size;

for (int i = 0; i < result.size; ++i) {

game\_t game = result.games[i];

last\_game\_id = max(last\_game\_id, game.id);

}

return result;

}

// need to free memory, after using it!

publisher\_list\_t read\_publisher\_list()

{

publisher\_list\_t result;

record\_list\_t records = read\_record\_list("publishers.dat", sizeof(publisher\_t));

result.publishers = (publisher\_t\*)records.records;

result.size = records.size;

for (int i = 0; i < result.size; ++i) {

publisher\_t game = result.publishers[i];

last\_game\_id = max(last\_game\_id, game.id);

}

return result;

}

void search\_by\_name(char game\_name[64])

{

game\_list\_t list = read\_game\_list();

for (int i = 0; i < list.size; ++i) {

if (strcmp(game\_name, list.games[i].game\_name) == 0) {

cout << "the game that you requested:" << endl;

print\_game\_info(list.games[i]);

}

}

delete list.games;

}

void search\_by\_publisher(int publisher\_id)

{

game\_list\_t list = read\_game\_list();

for (int i = 0; i < list.size; ++i) {

if (publisher\_id == list.games[i].publisher\_id) {

cout << "the game that you requested:" << endl;

print\_game\_info(list.games[i]);

}

}

delete list.games;

}

void filter\_by\_price(float max\_price)

{

game\_list\_t list = read\_game\_list();

for (int i = 0; i < list.size; ++i) {

game\_t game = list.games[i];

if (game.price <= max\_price) {

cout << "the game that you requested: " << endl;

print\_game\_info(game);

}

}

delete list.games;

}

void count\_copies\_by\_publisher(int publisher\_id)

{

game\_list\_t list = read\_game\_list();

int sum = 0;

for (int i = 0; i < list.size; ++i) {

game\_t game = list.games[i];

if (publisher\_id == game.publisher\_id) {

sum += game.amount;

}

}

cout << "the remaining copies " << sum << endl;

delete list.games;

}

void count\_price\_by\_genre(char genre[64])

{

game\_list\_t list = read\_game\_list();

float sum = 0;

for (int i = 0; i < list.size; ++i) {

game\_t game = list.games[i];

if (strcmp(genre, game.genre) == 0) {

sum += game.price \* game.amount;

}

}

cout << "your count price($): " << sum << endl;

delete list.games;

}

// sorts {games} array

void bubble\_sort\_price(game\_t games[], int size)

{

for (int j = 0; j < size; ++j) {

for (int i = 0; i < size - 1; ++i) {

if (games[i].price > games[i + 1].price) {

game\_t tmp = games[i];

games[i] = games[i + 1];

games[i + 1] = tmp;

}

}

}

}

// sorts {games} array

void bubble\_sort\_name(game\_t games[], int size)

{

for (int j = 0; j < size; ++j) {

for (int i = 0; i < size - 1; ++i) {

if (strcmp(games[i].game\_name, games[i + 1].game\_name) > 0) {

game\_t tmp = games[i];

games[i] = games[i + 1];

games[i + 1] = tmp;

}

}

}

}

void sort\_by\_name\_in\_show()

{

game\_list\_t list = read\_game\_list();

cout << endl;

cout << "Sorted by game name: " << endl;

bubble\_sort\_name(list.games, list.size);

cout << endl;

for (int i = 0; i < list.size; i++) {

print\_game\_info(list.games[i]);

}

delete list.games;

}

void sort\_by\_price\_in\_show()

{

game\_list\_t list = read\_game\_list();

cout << endl;

cout << "Sorted by game price: " << endl;

bubble\_sort\_price(list.games, list.size);

cout << endl;

for (int i = 0; i < list.size; i++) {

print\_game\_info(list.games[i]);

}

delete list.games;

}

void show\_publishers()

{

publisher\_list\_t list = read\_publisher\_list();

for (int i = 0; i < list.size; i++) {

print\_publisher\_info(list.publishers[i]);

}

delete list.publishers;

}

void show\_games()

{

game\_list\_t list = read\_game\_list();

int choice = -1;

bool done = false;

while (!done)

{

system("cls");

cout << "\n---------MENU----------\n";

cout << "1. Sort by game name\n2. Sort by game price\n3. show unsorted\n";

cout << "Enter your choice>";

cin >> choice;

switch (choice)

{

case 1:

sort\_by\_name\_in\_show();

done = true;

case 2:

sort\_by\_price\_in\_show();

done = true;

case 3:

for (int i = 0; i < list.size; i++) {

print\_game\_info(list.games[i]);

}

done = true;

}

}

delete list.games;

}

void search\_menu()

{

int choice = -1;

char name[64];

int id;

while (true)

{

system("cls");

cout << "\n---------MENU----------\n";

cout << "1.game name \n2.publisher\n3.max price\n4.copies in the storage\n5.count price\n";

cout << "Enter your choice>";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter game name\n";

cin >> name;

search\_by\_name(name);

return;

case 2:

cout << "Enter publisher id: \n";

cin >> id;

search\_by\_publisher(id);

return;

case 3:

float max\_price;

cout << "Enter max price\n";

cin >> max\_price;

filter\_by\_price(max\_price);

return;

case 4:

cout << "Enter publisher id: \n";

cin >> id;

count\_copies\_by\_publisher(id);

return;

case 5:

char genre[64];

cout << "Enter game genre\n";

cin >> genre;

count\_price\_by\_genre(genre);

return;

}

}

}