МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибернетики

Кафедра высшей математики

КУРСОВАЯ РАБОТА

по дисциплине
«Языки и методы программирования»

Тема курсовой работы

«Информационно-поисковая система (ИПС) - «Продажа музыкальной аудио-видео продукции»

Студент группы КМБО-01-18 *Соловьева А.Б.*

Руководитель курсовой работы *Шерстнев Е.В.*

Работа представлена к защите «__»_____201___ г. *(подпись студента)*

«Допущен к защите» «__»_____201___ г. *(подпись руководителя)*

МОСКВА — 2019

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибернетики

Кафедра высшей математики

Утверждаю
Заведующий
кафедрой_____*Ю.И.Худак*

«____» _____2019г.

ЗАДАНИЕ
на выполнение курсовой работы
по дисциплине «Языки и методы программирования»

Студент *Соловьева А.Б.* Группа *КМБО-01-18*

1. **Тема: «Информационно- поисковая система (ИПС) - «Продажа музыкальной аудио-видео продукции»**

2. **Исходные данные:**
   Разработать клиент-сервер систему с базой данных (базой может выступать текстовый файл).

3. **Перечень вопросов, подлежащих разработке, и обязательного графического материала:**
   1) Система должна обеспечивать возможность ввода, редактирования, удаления, поиска, фильтрации данных через меню или интерфейс, а так же вывода отчетов по заданным критериям (не менее 3х отчетов).

4. **Срок представления к защите курсовой работы: до** « » _____ 2019 г.

Задание на курсовую работу выдал «___»_____2019 г. _____ *(_____)*
Задание на курсовую работу получил «___»_____2019 г. _____ *(_____)*

Оглавление

Введение

Целью курсовой работы является реализация Информационно-поисковой системы – «Продажа музыкальной аудио-видео продукции». Необходимо реализовать клиент-сервер систему с базой данных. Система должна поддерживать стандартный функционал (CRUD – create, read, update, delete), то есть обеспечивать: создание, чтение, редактирование, удаление записей. Помимо этого будет реализован поиск и фильтр непосредственно для базы данных. Также, что немало важно для таких систем, необходимо обеспечить поддержку отчетности – выдачу отчета по критерию. Для реализации был выбран язык программирования C++.

Основная часть

Определение архитектуры «клиент – сервер»

Не все компьютеры и программы, находящиеся в рамках одной информационной сети обладают одинаковыми правами. Какие-то из них владеют ресурсами, а другие могут всего лишь обращаться к этим ресурсам. Компьютер (программу), управляющий ресурсами, определяют как сервер данного ресурса. Клиент и сервер какого-либо ресурса могут находиться в рамках одной вычислительной системы или сети.

Основной принцип технологии «клиент-сервер» заключается в разделении функций приложения на такие группы как:

- Функции управления ресурсами (файловой системой, базой данных).
- Ввод и отображения данных (взаимодействие с пользователем).
- Прикладные функции, обеспечивающие работоспособность первых двух групп.

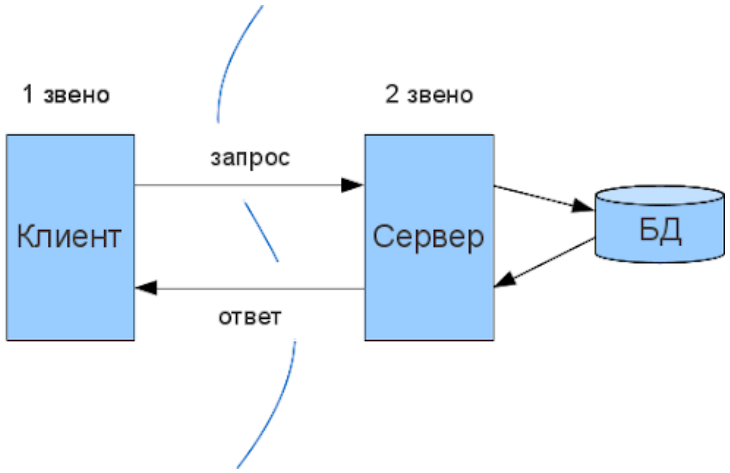

Рисунок 1 – иллюстрация архитектуры «клиент-сервер»

Основной функционал

Программа реализована с помощью функций и классов, разбитых на отдельные файлы.

Работа программы организована при помощи взаимодействия 3-х классов:

- class Data – объекты класса определяют информацию хранящуюся в базе
- class Server – объект класса осуществляет непосредственное управление базой дынных.
- class Client – объект класса позволяет взаимодействовать с программой – «сервером» с помощью набора команд.

Функционал класса Data

Класс имеет набор приватных полей, которые отвечают за хранящуюся информацию. Так как для реализации была выбрана информационно-поисковая система (ИПС) – «Продажа музыкальной аудио-видео продукции», уместны такие значения как:

- id – идентификатор позиции в базе данных;
- name – наименование позиции;
- author – автор аудио-видео продукции;
- genre – жанр продукции;
- extension – разрешение реализуемой продукции;
- size – размер (в килобайтах) продукции;
- cost – цена товара;

Также реализован набор сеттеров*, геттеров* (прим. setName(…), getName()), которые отвечают за изменение и выдачу информации, хранящейся в приватных полях соответственно. Совершенно стандартный функционал для большинства типов хранения данных.

Определены перегрузки операторов:

- присваивания (=);
- потокового ввода (>>);
- потокового вывода (<<);

Перегрузка оператор присваивания была реализована для изменения полей дынных вовремя работы с базой. К примеру, во время удаления значения из

базы, необходимо произвести сдвиг элементов в структуре хранения данных, то есть изменить значения идентификаторов позиций.

Перегрузки операторов ввода/вывода необходимы для взаимодействия с элементами в текстовой базе. В зависимости от функции базы данных, нужно вводить/выводить элементы в консоль/файл, соответственно для сокращения объема кодовой базы, были введены данные перегрузки операторов.

Функционал класса Server

Данный класс является «основой» проекта. Он осуществляет прямое взаимодействие с базой данных, то есть является программой – «сервером». Класс имеет возможность перезаписи, добавления, удаления данных. Говоря простым языком, имеет «полную власть» над базой данных. Server универсален и подходит для управления базами, хранящими любые типы данных.

Данный класс имеет такие приватные поля как:

- baseName – имя файла базы данных
- currentId – указатель на последний идентификатор в базе
- base – основа для хранения базы данных в оперативной памяти.

В качестве структуры данных для base был выбран вектор (массив). Выбрана именно эта структура данных потому, что она поддерживает доступ по индексу элемента, нет никаких ограничений на хранимые данные, быстрое добавление элементов в конец структуры.

Определены конструктор и деструктор класса. Конструктор принимает имя файла базы данных в системе.

```
Server::Server(std::string _baseName)
{
    baseName = _baseName;
    loadBase(baseName);
}
```

Код 1. Конструктор класса Server

Конструктор вызывает функцию обеспечения loadBase(…), которая в свою очередь загружает текстовую базу данных в оперативную память, и выставляет указатель на последний элемент.

Реализован функционал CRUD – create, read, update, delete; посредством функций create(…), read(…), update(…), del(…).

- create – принимает на вход объект класса Data, осуществляет одновременную запись элемента в структуру данных базы во временной памяти и запись элемента в текстовую базу данных.
- read – принимает на вход целочисленную переменную id, функция возвращает строку базы данных по соответствующему идентификатору.
- update - принимает на вход целочисленную переменную id, а также переменную типа Data, заменяет строку с соответствующим id на введенную.
- del – принимает на вход целочисленную переменную id, удаляет из базы элемент с соответствующим идентификатором и сдвигает все элементы с большим значением id на одну позицию.

```
for (int i = id-1; i < base.size()-1; ++i) {
    base[i+1].setId(i+1);
    base[i] = base[i+1];
}
base.pop_back();
```

Код 2. Часть функции del, показывающая сдвиг

Организована фильтрация и поиск, по заранее указанным полям.

Поиск ведется по полю name у данных. Это соответственно позволяет пользователю узнать, есть ли данная продукция в базе. Чтобы найти произведение, сервер обходит подгруженную в оперативную память базу посредством алгоритма линейного поиска.

Фильтрация ведется по полю cost. Ориентируясь на техническое задание, это поле является одним из самых релевантных для фильтрации.

```
std::vector<Data>  Server::filter(int  cost,  bool  (*compareFunction)(int,
int)){

   std::vector<Data> filtered;
   for (const auto& item: base) {
```

```
        if (compareFunction(cost, item.getCost())) {
                filtered.push_back(item);
        }
    }
    return filtered;
}
```

Код 3. Функция filter

Механизмы C++ позволяют передавать в качестве параметра функции другую функцию. Соответственно это помогает сократить кодовую базу. При организации клиента в функцию filter передается лямбда-выражение. Это помогает избежать создания лишних функции или блоков с условиями.

```
filtered = base.filter(bufferNum, [](int lhs, int rhs) -> bool { return lhs < rhs; });
```

Код 4. Пример использования функции filter

filter возвращает вектор с элементами типа Data, для последующего вывода в консоль.

Функции-отчеты представлены как:

- report10MaxCost() - выводит десять самых дорогих произведений в базе.
- report10MinSize() - выводит десять самых минимальных по размеру элементов реализуемой продукции.
- reportChosenGenre(...) - на вход принимает строку - выбранный жанр для отчета, далее находит все позиции с указанным жанром и выводит сформированный отчет в консоль.

Также у данного класса есть прикладная функция writeHat(…), которая принимает на вход поток вывода «шапки» базы. Эта функция помогает избежать повторения кода, к примеру, при выводе отчетов или перезаписи базы при удалении элемента.

Рисунок 2 - «шапка» базы

Функционал класса Client

Посредством данного класса происходит «общение» пользователя с сервером.

Приватными полями данного класса выступают структуры данных типа map (C++). Выбрана такая структура данных в связи с тем, что структура switch не может принимать тип данных string. Поэтому ввод осуществляется посредством подмены команд соответствующими значениями словаря.

- commandMap – содержит основные команды первичного пользовательского ввода ({“CREATE”, 1});
- updateMap – содержит команды выбора поля при редактировании ({“NAME”, 1});
- filterKeyMap – содержит команды выбора направления фильтрации ({“MORE”, 1});

Еще одно приватное поле – baseName. В него записывается имя базы данных, с которой будет работать пользователь. Запись осуществляется либо внутри функции main посредством конструктора, либо во время работы функции userInterface. Если имя базы нигде не было указано, и пользователь отказался вводить его внутри функции userInterface, выдается сообщение об ошибке и работа программы прекращается.

userInterface() является основной функцией класса Client в ней осуществлен механизм пользовательского ввода. Внутри нее создается объект класса Server, с помощью которого происходит взаимодействие пользователя с базой данных.

Постоянный ввод команд обеспечивается с помощью бесконечного цикла do-while из которого можно выйти с помощью команды EXIT. Внутри цикла определены буферные переменные, в которые производится запись пользовательского ввода, в которые выводятся результаты работы функций класса Server. С каждой итерацией цикла система просит пользователя ввести команду для работы с базой. Далее, после введения команды, она передается в оператор switch, который ищет соответствие с доступными командами в

своем теле. В случае отсутвия команды условие уходит в default, нде пользователя предупреждают о том, что такой команды нет.

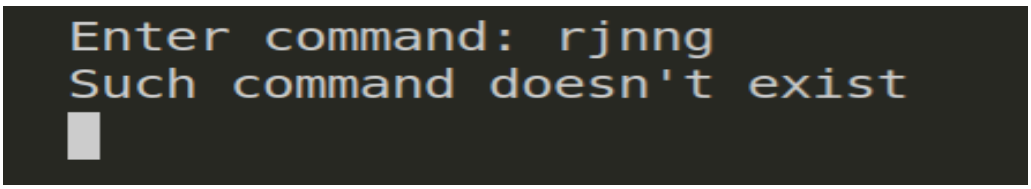

Рисунок 3 – иллюстрация обработки неправильного ввода

В случае успеха оператор switch переносит пользователя в следующий блок, где ему доступны дальнейшие действия для работы с базой. Внутри блоков также есть механизмы, которые перехватывают неправильный ввод и выводят пользователю сообщение об ошибке.

Система выстроена так, чтобы у пользователя была возможность создавать новые команды, путем быстрого добавления желаемой команды в тело оператора switch.

Тесты и работа программы

База имеет вид таблицы, в столбцах которой уже есть некоторая информация.

Id	Name	Author	Genre	Extension	Size(Kbyte)	Cost($)
1	Enter Sandman	Metallica	rock	wav	4341	3
2	Lazy	Deep purple	rock	wav	6473	1
3	Seven Devil	FaTM	pop	mp3	5242	2
4	Dance monkey	Tones and I	indie	mp3	346545	5
5	Drop That Low	Tujamo	hip-hop	mp3	67384	4
6	Lalala	Y2K & bbno$	rap	ogc	572419	12
7	WTF	HUGEL	hip-hop	wav	6753842	9
8	Lordly	Feder	dance	mp3	14532342	15
9	Party Train	Redfoo	electro	ogc	751441	4
10	Sunflower	Post Malone	hip-hop	wav	6853429	7
11	Lean On	Major Lazer	soul	ogc	5724831	10
12	NO LIE	Sean Paul	pop	mp3	564714	4
13	Good Day	DNCE	indie-pop	wav	756437	7
14	Gangam Style	PSY	kpop	mp3	4134554	20

Рисунок 4 – вид базы

Далее в зависимости от выбранных пользователем команд, вид базы меняется. По мере ввода команд система дает пользователю подсказки, что именно нужно вводить, какие поля нужны для фильтрации, какие для поиска и т.п.

Команда CREATE

Данная команда посредством пользовательского ввода создает новую позицию в базе данных

```
Enter command: CREATE
Enter data :
Enter name :-----------------: Ride It
Enter author of production :-: Regard
Enter genre :----------------: dance
Enter extension :------------: wma
Enter size in kbytes :-------: 13542
Enter cost in $ :------------: 11
```

Рисунок 5 — пример пользовательского ввода команды CREATE

После ввода данных в базе появляется новая позиция с автоматически выставленным Id. Пользователь не может самостоятельно ввести Id во избежание появления двух идентичных записей в базе.

```
|   12|        NO LIE|      Sean Paul|         pop|       mp3|   11647|      4|
|   13|      Good Day|           DNCE|    indie-pop|       wav|    3564|      7|
|   14|  Gangam Style|            PSY|        kpop|       mp3|    1345|     20|
|   15|       Ride It|         Regard|       dance|       wma|   13542|     11|
```

Рисунок 6 — появление новой записи в базе

Команда READ

Посредством данной команды происходит чтение записей из базы. Пользователь вводит Id и выводится полная информация по позиции с соответствующим идентификатором. Для чтения выбран именно Id, потому что шансы ошибки ввода меньше. Так как Name и Author у продукции может повторяться, а Id — нет.

```
Enter command: READ
Choose id of row: 6
Choosen row:

|   Id|          Name|        Author|       Genre|  Extension|  Size(Kbyte)|  Cost($)|
|    6|        Lalala|   Y2K & bbno$|         rap|        ogc|         6724|       12|
```

Рисунок 7 — применение команды READ

Команда UPDATE

С помощью команды UPDATE пользователь может произвести замену полей в любой записи. Программа предлагает пользователю выбор определенного поля, соответственно полностью запись менять не нужно.

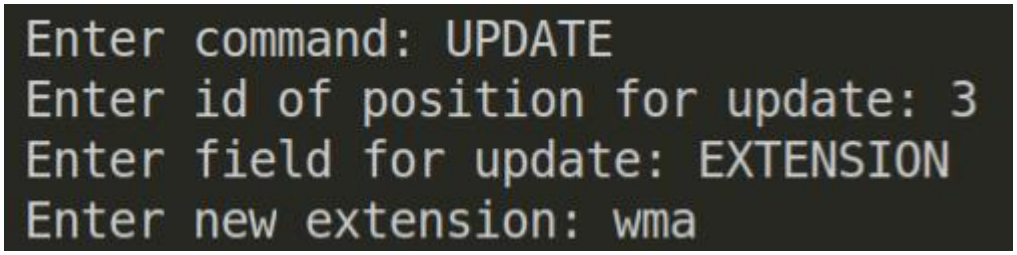

Рисунок 8 — применение команды UPDATE

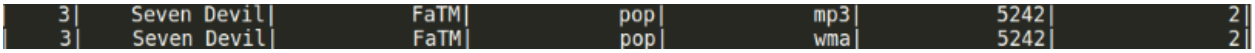

Рисунок 9 — поле до/после

Команда DELETE

Данная команда позволяет удалить любую запись из базы по ее уникальному идентификатору — Id. После выполнения данной операции все Id сдвигаются, и у каждой записи появляется новый Id.

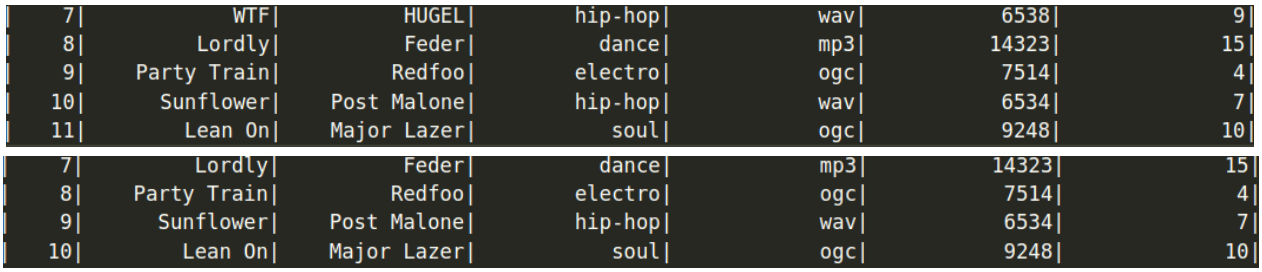

Рисунок 10 — база до и после удаления 7-й записи

Команда SEARCH

Посредством команды SEARCH производится поиск позиции в базе данных по полю NAME. В случае, если такой продукции нет, система выдает предупреждение об отсутствии, в случае успеха — id найденной позиции в базе.

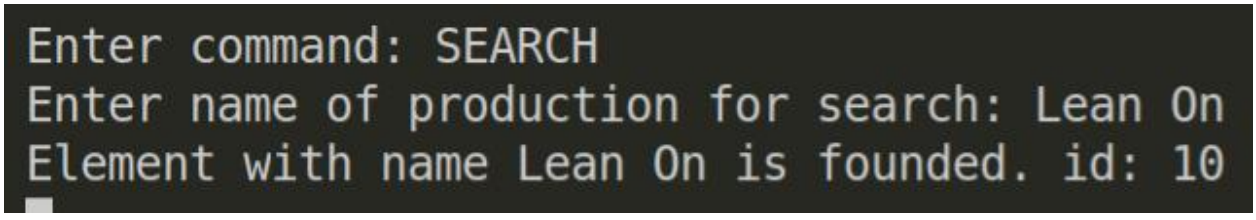

Рисунок 11 - применение команды SEARCH

Команда FILTER

Данная команда выводит таблицу записей в консоль в зависимости от ключа фильтра — {MORE, EQUAL, LESS}. То есть создает таблицу значений из элементов: больших, равных, меньших указанного. Фильтр работает по полю COST, так как предполагается то, что это одно из самых релевантных полей для системы продаж.

```
Enter command: FILTER
Enter filter key: MORE
Enter filter edge: 5

|   Id|          Name|        Author|         Genre|     Extension|   Size(Kbyte)|       Cost($)|
|    6|        Lalala|   Y2K & bbno$|           rap|           ogc|          6724|            12|
|    7|        Lordly|         Feder|         dance|           mp3|         14323|            15|
|    9|     Sunflower|   Post Malone|       hip-hop|           wav|          6534|             7|
|   10|       Lean On|   Major Lazer|          soul|           ogc|          9248|            10|
|   12|      Good Day|          DNCE|      indie-pop|           wav|          3564|             7|
|   13|  Gangam Style|           PSY|          kpop|           mp3|          1345|            20|
|   14|       Ride It|        Regard|         dance|           wma|         13542|            11|
```

Рисунок 12 — пример работы команды FILTER

В данном случае выводятся все позиции, стоимость которых, больше 5 у.е. за штуку.

Команда MAX COST

Также в базе представлены команды отчетов, одна из них — MAX COST. Пользователю необходимо ввести количество позиций для вывода, далее система с помощью стандартной библиотеки C++ - <algorithm> сортирует и выводит n первых позиций удовлетворяющих требованию — самые дорогие позиции в базе.

```
sort(report.begin(), report.end(), [](Data lhs, Data rhs) -> bool { return lhs.getCost() > rhs.getCost(); });
```

Код 5. Сортировка базы

В случае, если в базе меньше n позиций, система выводит предупреждение и отчет содержащий все отсортированные позиции в базе.

```
Enter command: MAX COST
Enter number of positions for report: 6
Report with 6 max costed items in base

|   Id|          Name|        Author|         Genre|     Extension|   Size(Kbyte)|       Cost($)|
|   13|  Gangam Style|           PSY|          kpop|           mp3|          1345|            20|
|    7|        Lordly|         Feder|         dance|           mp3|         14323|            15|
|    6|        Lalala|   Y2K & bbno$|           rap|           ogc|          6724|            12|
|   14|       Ride It|        Regard|         dance|           wma|         13542|            11|
|   10|       Lean On|   Major Lazer|          soul|           ogc|          9248|            10|
|    9|     Sunflower|   Post Malone|       hip-hop|           wav|          6534|             7|
```

Рисунок 13 — отчет — n позиций с максимальной стоимостью

Команда MIN SIZE

Команда работает по такому же принципу как и MAX COST. Разница только в направлении сортировки, так как создается отчет из n минимальных по размеру позиций.

```
Enter command: MIN SIZE
Enter number of positions for report: 5
Report with 5 min sized items in base
```

Id	Name	Author	Genre	Extension	Size(Kbyte)	Cost($)
13	Gangam Style	PSY	kpop	mp3	1345	20
12	Good Day	DNCE	indie-pop	wav	3564	7
1	Enter Sandman	Metallica	rock	wav	4341	3
3	Seven Devil	FaTM	pop	wma	5242	2
2	Lazy	Deep purple	rock	wav	6473	1

Рисунок 14 — отчет — n позиций с минимальным размером

Команда CHOSEN GENRE

Посредством команды CHOSEN GENRE создается отчет по выбранному жанру. То есть происходит полный обход базы, загруженной в оперативную память, и в отдельный вектор загружаются значения, удовлетворяющие условию — соответствующий жанр. В случае, если в базе нет такого жанра, выводится предупреждение об отсутствии.

```
Enter command: CHOSEN GENRE
Choose genre for report: pop
```

Id	Name	Author	Genre	Extension	Size(Kbyte)	Cost($)
3	Seven Devil	FaTM	pop	wma	5242	2
11	NO LIE	Sean Paul	pop	mp3	11647	4

Рисунок 15 — отчет — все позиции жанра pop

Заключение

В ходе выполнения курсовой работы была реализована информационно-поисковая система (ИПС) - «Продажа музыкальной аудио-видео продукции» на языке C++. При разработке были продемонстрированы принципы построения приложений такого рода.

Выполнение курсовой работы помогло закрепить изученный материал и навыки, полученные в ходе лекций и на лабораторных работах, систематизировать знания языка и научиться применять их в поставленных задачах.

Список литературы

1. Дьюхарст Программирование на C++ / Дьюхарст, Старк Стефан; , Кэти. - М.: ДиаСофт, 2015. - 272 с.
2. Страуструп, Б. Язык программирования C++: Специальное издание / Б. Страуструп; Пер. с англ. Н.Н. Мартынов. — М.: БИНОМ, 2012. — 1136 с.
3. Раскин, Д. Интерфейс. Новые направления в проектировании компьютерных систем.: Символ-Плюс, 2007. - 272 с.

Листнинг с кодом

Data.hpp

```
#pragma once

#include <string>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <vector>

class Data
{
private:
    int id;
    std::string name;
    std::string author;
    std::string genre;
    std::string extension;
    size_t size;
    int cost;

public:
    Data(
        int _id = 0,
        std::string _name = "",
        std::string _author = "",
        std::string _genre = "",
        std::string _extension = "",
        int _size = 1,
        int _cost = 0);

    Data &operator=(const Data&);

    int getId() const { return id; }
    std::string getName() const { return name; }
    std::string getAuthor() const { return author; }
    std::string getGenre() const { return genre; }
    std::string getExtension() const { return extension; }
    int getSize() const { return size; }
    int getCost() const { return cost; }


    void setId(const int& _id) { id = _id; }
    void setName(const std::string& _name) { name = _name; }
    void setAuthor(const std::string& _author) { author = _author; }
    void setGenre(const std::string& _genre) { genre = _genre; }
    void setExtension(const std::string& _extension) { extension = _extension; }
    void setSize(const int& _size) { size = _size; }
    void setCost(const int& _cost) { cost = _cost; }

    friend std::ostream &operator<<(std::ostream &, const Data &);
    friend std::istream &operator>>(std::istream &, Data &);
};
```

Data.cpp

```cpp
#include "Data.hpp"


Data::Data(
            int _id,
            std::string _name,
            std::string _author,
            std::string _genre,
            std::string _extension,
            int _size,
            int _cost
        )
{
    id = _id;
    name = _name;
    author = _author;
    genre = _genre;
    extension = _extension;
    size = _size;
    cost = _cost;
}

Data &Data::operator=(const Data &data)
{
    if (this == &data)
        return *this;

    setId(data.getId());
    setName(data.getName());
    setAuthor(data.getAuthor());
    setGenre(data.getGenre());
    setExtension(data.getExtension());
    setSize(data.getSize());
    setCost(data.getCost());

    return *this;
}

std::ostream &operator<<(std::ostream &out, const Data &sample)
{
    out << '|' << std::setw(5) << sample.getId() << '|'
        << std::setw(15) << sample.getName() << '|'
        << std::setw(15) << sample.getAuthor() << '|'
        << std::setw(15) << sample.getGenre() << '|'
        << std::setw(15) << sample.getExtension() << '|'
        << std::setw(15) << sample.getSize() << '|'
        << std::setw(15) << sample.getCost() << '|' << std::endl;
    return out;
}

std::istream &operator>>(std::istream &in, Data &sample)
{
    std::vector<std::string> parced;
    std::string parce;

    getline(in, parce);
```

```
    if (parce == "")
        return in;

    std::string newItem = "";
    int stickCounter = 1;

    for (int i = 1; i < parce.size(); ++i) {

        if ((newItem != "" && stickCounter == 2) || i == parce.size()-1) {
            parced.push_back(newItem);
            newItem = "";
            stickCounter = 1;
        } else if ((parce[i] != '|' && parce[i] != ' ') ||
                   ((parce[i-1] != '|' && parce[i-1] != ' ') &&
                   (parce[i+1] != '|' && parce[i+1] != ' '))) {
            newItem += parce[i];
        } else if (parce[i] == '|') {
            ++stickCounter;
        }
    }

    sample.setId(std::stoi(parced[0]));
    sample.setName(parced[1]);
    sample.setAuthor(parced[2]);
    sample.setGenre(parced[3]);
    sample.setExtension(parced[4]);
    sample.setSize(std::stoi(parced[5]));
    sample.setCost(std::stoi(parced[6]));

    return in;
}
```

Server.hpp

```
#pragma once

#include "Data.hpp"
#include <vector>
#include <algorithm>
#include <map>
#include <unistd.h>

class Server
{
private:
    std::string baseName;
    int currentId;
    std::vector <Data> base;

public:
    Server(std::string _baseName = ""); // +
    ~Server();                          // +

    void loadBase(std::string); // +
    void printBase(std::ostream &); // +
```

```
    void create(Data); // +
    Data read(int); // +
    void update(int, Data); // +
    Data del(int); // +


    Data search(std::string); // +
    std::vector<Data> filter(int, bool (*compareFunction)(int, int)); // +

    void writeHat(std::ostream &); // +

    void reportMaxCost(int);
    void reportMinSize(int);
    void reportChosenGenre(std::string);

    Data getId(int id) { return base[id]; }

    void userInterface();
};
```

Server.cpp

```
#include "Server.hpp"


Server::Server(std::string _baseName)
{
    baseName = _baseName;
    loadBase(baseName);
}

Server::~Server()
{
    base.clear();
}

void Server::loadBase(std::string _baseName)
{
    currentId = 1;
    std::ifstream fin(_baseName);
    std::string strForParce;

    getline(fin, strForParce);
    getline(fin, strForParce);


    while (fin.good()) {
        Data newRecord;
        fin >> newRecord;

        currentId = base.size()+1;
        if (newRecord.getId() == 0)
            return;

        base.push_back(newRecord);
    }
}
```

```
void Server::printBase(std::ostream &out)
{
    writeHat(out);

    for (const auto &item : base) {
        out << item;
    }
    out << std::endl;
}

void Server::create(Data data)
{
    data.setId(currentId);
    base.push_back(data);

    std::ofstream fin(baseName, std::ios::app);
    fin << data;

    ++currentId;
}

Data Server::read(int id)
{
    return base[id-1];
}

void Server::update(int id, Data data)
{
    if (id < 1 || id > base.size()) {
        std::cout << "Such id doesn't exist" << std::endl;
        return;
    }

    data.setId(id);
    base[id-1] = data;

    std::ofstream fout(baseName);
    printBase(fout);
}

Data Server::del(int id)
{
    Data data;
    if (id < 1 || id > base.size()) {
        std::cout << "Such id doesn't exist" << std::endl;
        return data;
    }
    data = base[id];

    for (int i = id-1; i < base.size()-1; ++i) {
        base[i+1].setId(i+1);
        base[i] = base[i+1];
    }
    base.pop_back();

    std::ofstream fout(baseName);
    printBase(fout);
```

```
    return data;
}

Data Server::search(std::string name)
{
    for (const auto& item: base) {
        if (name == item.getName()){
            std::cout << "Element with name" << ' '
                    << item.getName() << ' '
                    << "is founded. id:" << ' '
                    << item.getId() << std::endl;
            return item;
        }
    }
    std::cout << "Element with name" << ' '
              << name << ' '
              << "isn't founded in base" << std::endl;
    return *(new Data);
}

std::vector<Data> Server::filter(int cost, bool (*compareFunction)(int, int))
{
    std::vector<Data> filtered;

    for (const auto& item: base) {
        if (compareFunction(cost, item.getCost())) {
            filtered.push_back(item);
        }
    }
    return filtered;
}

void Server::reportMaxCost(int value)
{
    std::vector<Data> report = base;

    sort(report.begin(), report.end(), [](Data lhs, Data rhs) -> bool { return
lhs.getCost() > rhs.getCost(); });

    if (report.size() < value) {
        std::cout << "Unfortunately, only " << report.size() << " records in base"
<< std::endl;
    }

    writeHat(std::cout);

    for (int i = 0; i < value; ++i) {
        if (i == report.size())
            return;
        std::cout << report[i];
    }
}
void Server::reportMinSize(int value)
{
    std::vector<Data> report = base;

    sort(report.begin(), report.end(), [](Data lhs, Data rhs) -> bool { return
lhs.getSize() < rhs.getSize(); });
```

```
    if (report.size() < value) {
        std::cout << "Unfortunately, only " << report.size() << " records in base"
<< std::endl;
    }

    writeHat(std::cout);

    for (int i = 0; i < value; ++i) {
        if (i == report.size())
            return;
        std::cout << report[i];
    }
}
void Server::reportChosenGenre(std::string key)
{
    std::vector<Data> report;

    for (auto iter = base.begin(); iter != base.end(); ++iter) {
        if (iter->getGenre() == key) {
            report.push_back(*iter);
        }
    }

    if (report.empty()) {
        std::cout << "No such genre in base" << std::endl;
        return;
    }
    writeHat(std::cout);

    for (int i = 0; i < report.size(); ++i)
    {
        std::cout << report[i];
    }
}

void Server::writeHat(std::ostream &out)
{
    for (int i = 0; i < 103; ++i) {
        out << '_';
    }
    out << std::endl;
    out << '|' << std::setw(5) << "Id" << '|'
        << std::setw(15) << "Name" << '|'
        << std::setw(15) << "Author" << '|'
        << std::setw(15) << "Genre" << '|'
        << std::setw(15) << "Extension" << '|'
        << std::setw(15) << "Size(Kbyte)" << '|'
        << std::setw(15) << "Cost($)" << '|' << std::endl;
}
```

Client.hpp

```
#include "Server.hpp"


class Client
{
private:
    std::map<std::string, int> commandMap = {{"CREATE", 1},
                                             {"READ", 2},
                                             {"UPDATE", 3},
                                             {"DELETE", 4},
                                             {"SEARCH", 5},
                                             {"FILTER", 6},
                                             {"MAX COST", 7},
                                             {"MIN SIZE", 8},
                                             {"CHOSEN GENRE", 9},
                                             {"EXIT", 10}};

    std::map<std::string, int> updateMap = {{"NAME", 1},
                                            {"AUTHOR", 2},
                                            {"GENRE", 3},
                                            {"EXTENSION", 4},
                                            {"SIZE", 5},
                                            {"COST", 6}};

    std::map<std::string, int> filterKeyMap = {{"MORE", 1},
                                               {"EQUAL", 2},
                                               {"LESS", 3}};

    std::string baseName;

public:
    Client(std::string _baseName="");

    void userInterface();
};
```

Client.cpp

```
#include "Client.hpp"


Client::Client(std::string _baseName)
{
    baseName = _baseName;
}

void Client::userInterface()
{

    char ans;

    std::cout << "Do you want to change name of data base (y/n)? ";
    std::cin >> ans;
```

```
if (ans == 'y') {
    std::cout << "Enter name of data base: ";
    std::cin >> baseName;
} else if (ans == 'n' && baseName == "") {
    std::cout << "No base name" << std::endl;;
    return;
}
std::cin.ignore(32767, '\n');
printf("\e[1;1H\e[2J");

Server base(baseName);

do
{
    std::string command;
    std::string fieldForUpdate;
    std::string filterKey;
    std::vector<Data> filtered;

    std::cout << "Enter command: ";
    std::getline(std::cin, command);

    int bufferNum;
    std::string buffer;
    Data data;
    int id;
    const int showTime = 5;

    switch (commandMap[command])
    {
    case 1:

        std::cout << "Enter data : " << std::endl;
        std::cout << "Enter name :-----------------: ";
        std::getline(std::cin, buffer);
        data.setName(buffer);

        std::cout << "Enter author of production :-: ";
        std::getline(std::cin, buffer);
        data.setAuthor(buffer);

        std::cout << "Enter genre :----------------: ";
        std::getline(std::cin, buffer);
        data.setGenre(buffer);

        std::cout << "Enter extension :------------: ";
        std::getline(std::cin, buffer);
        data.setExtension(buffer);

        std::cout << "Enter size in kbytes :-------: ";
        std::cin >> bufferNum;
        data.setSize(bufferNum);

        std::cout << "Enter cost in $ :------------: ";
        std::cin >> bufferNum;
        data.setCost(bufferNum);

        base.create(data);
```

```
    break;
case 2:
    std::cout << "Choose id of row: ";
    std::cin >> id;

    std::cout << "Chosen row: " << std::endl;
    base.writeHat(std::cout);
    std::cout << base.getId(id-1) << std::endl;

    std::cout << "Press enter for continue...";
    sleep(showTime);
    break;
case 3:
    std::cout << "Enter id of position for update: ";
    std::cin >> id;
    std::cout << "Enter field for update: ";
    std::cin >> fieldForUpdate;
    std::cin.ignore(32767, '\n');

    data = base.getId(id-1);

    switch (updateMap[fieldForUpdate])
    {
    case 1:
        std::cout << "Enter new name: ";
        std::getline(std::cin, buffer);
        data.setName(buffer);
        break;
    case 2:
        std::cout << "Enter new author: ";
        std::getline(std::cin, buffer);
        data.setAuthor(buffer);
        break;
    case 3:
        std::cout << "Enter new genre: ";
        std::getline(std::cin, buffer);
        data.setGenre(buffer);
        break;
    case 4:
        std::cout << "Enter new extension: ";
        std::getline(std::cin, buffer);
        data.setExtension(buffer);
        break;
    case 5:
        std::cout << "Enter new size: ";
        std::cin >> bufferNum;
        data.setSize(bufferNum);
        break;
    case 6:
        std::cout << "Enter new cost: ";
        std::cin >> bufferNum;
        data.setCost(bufferNum);
        break;
    default:
        std::cout << "Such field doesn't exist" << std::endl;
        return;
        break;
    }
    base.update(id, data);
```

```
            break;
        case 4:
            std::cout << "Enter id of row for delete: ";
            std::cin >> id;

            base.del(id);
            break;
        case 5:
            std::cout << "Enter name of production for search: ";
            std::getline(std::cin, buffer);

            base.search(buffer);
            sleep(showTime);
            break;
        case 6:
            std::cout << "Enter filter key: ";
            std::cin >> filterKey;

            std::cout << "Enter filter edge: ";
            std::cin >> bufferNum;
            std::cin.ignore(32767, '\n');

            switch (filterKeyMap[filterKey])
            {
            case 1:
                filtered = base.filter(bufferNum, [](int lhs, int rhs) -> bool {
return lhs < rhs; });
                break;
            case 2:
                filtered = base.filter(bufferNum, [](int lhs, int rhs) -> bool {
return lhs == rhs; });
                break;
            case 3:
                filtered = base.filter(bufferNum, [](int lhs, int rhs) -> bool {
return lhs > rhs; });
                break;
            default:
                std::cout << "Such filter key doesn't exist" << std::endl;
                break;
            }

            if (filtered.empty())
            {
                std::cout << "No items matching the condition" << std::endl;
            }
            else
            {
                base.writeHat(std::cout);
                for (const auto &item : filtered)
                {
                    std::cout << item;
                }
            }
            sleep(showTime+2);
            break;
        case 7:
            std::cout << "Enter number of positions for report: ";
            std::cin >> bufferNum;
```

```
            std::cout << "Report with" << ' ' << bufferNum << ' ' << "max costed
items in base" << std::endl;
            base.reportMaxCost(bufferNum);
            sleep(showTime+2);
            break;
        case 8:
            std::cout << "Enter number of positions for report: ";
            std::cin >> bufferNum;

            std::cout << "Report with" << ' ' << bufferNum << ' ' << "min sized
items in base" << std::endl;
            base.reportMinSize(bufferNum);
            sleep(showTime+2);
            break;
        case 9:
            std::cout << "Choose genre for report: ";
            getline(std::cin, buffer);

            base.reportChosenGenre(buffer);
            sleep(showTime+2);
            break;
        case 10:
            return;
            break;
        default:
            std::cout << "Such command doesn't exist" << std::endl;
            sleep(3);
            break;
        }
        printf("\e[1;1H\e[2J");
    } while (true);
}
```