Семинар #6: Структуры. Домашнее задание.

Задача 1. Треугольник

Для описания треугольников на плоскости были определены структуры `Point` и `Triangle`:

```
struct point
{
    double x, y;
};
typedef struct point Point;

struct triangle
{
    Point a, b, c;
};
typedef struct triangle Triangle;
```

Напишите следующие функции для работы с этими структурами:

- Функцию `print_point`, которая будет принимать точку и печатать её в формате `(1.23, 4.56)`. То есть в круглых скобках, через запятую и с двумя знаками после запятой.
- Функцию `print_triangle`, которая будет принимать на вход треугольник и печатать координаты треугольника в следующем формате: `{(1.00, 0.00), (0.50, 2.00), (0.00, 1.50)}`.
- Функцию `distance`, которая будет принимать на вход 2 точки и возвращать расстояние между ними.
- Функцию `get_triangle_perimeter`, которая будет принимать треугольник по константному указателю и возвращать его периметр.
- Функцию `get_triangle_area`, которая будет принимать на вход треугольник по константному указателю и возвращать его площадь. Можно использовать формулу Герона.
- Функцию `moved_triangle`, которая будет принимать на вход треугольник по константному указателю и одну точку (она будет играть роль вектора-перемещения). Функция должна возвращать новый треугольник, у которого все координаты будут передвинуты на вектор-перемещение.
- Функцию `move_triangle`, которая будет принимать на вход треугольник по указателю и одну точку (она будет играть роль вектора-перемещения). Функция должна менять передаваемый ей треугольник.

Задача 2. Рецензии на компьютерные игры

На вход программе приходит информация о рецензиях компьютерных игр. В первой строке содержится число `n` - количество игр. Далее идут `n` строк. В каждой строке содержится название игры, заканчивающееся двоеточием сразу после идёт целое число `k` – количество оценок, которые эта игра получила, затем идут `k` оценок. Оценка, это число от 1 до 10. Нужно отсортировать все игры по средней оценке и напечатать название игр и их среднюю оценку.

вход	выход
5 Need For Speed: 6 6 1 2 7 5 4 Sector: 3 1 4 2 The Cube: 7 9 8 7 9 8 10 7 Principle Of Chaos 2: 5 4 3 6 5 7 Metal Power: 10 8 5 3 9 6 2 6 7 5 8	The Cube, 8.286 Metal Power, 5.900 Principle Of Chaos 2, 5.200 Need For Speed, 4.667 Sector, 2.333

Протестировать программу можно на файле `videogames.txt`.

Задача 3. Создание указателей

Решения всех подзадач этой части – одна строка. Результат выполнения задания – `.txt` файл, который содержит все эти строки.

1. В следующей программе создаётся переменная `a` типа `int`:

```
int main()
{
    int a = 1234;
    // Тут нужно написать 1 строку кода
}
```

   Создайте указатель `p` и инициализируйте его адресом переменной `a`.

2. В следующей программе создаётся переменная `a` типа `double`:

```
int main()
{
    double a = 12.34;
    // Тут нужно написать 1 строку кода
}
```

   Создайте указатель `p` и инициализируйте его адресом переменной `a`.

3. В следующей программе создаётся переменная `a` типа `char`:

```
int main()
{
    char a = ')';
    // Тут нужно написать 1 строку кода
}
```

   Создайте указатель `p` и инициализируйте его адресом переменной `a`.

4. В следующей программе создаётся массив `array` из элементов типа `int`:

```
int main()
{
    int array[5] = {10, 20, 30, 40, 50};
    // Тут нужно написать 1 строку кода
}
```

   Создайте указатель `p` и сделайте так, чтобы он указывал на первый элемент массива (индекс `0`).

5. В следующей программе создаётся строка – массив `str` из элементов типа `char`:

```
int main()
{
    char str[20] = "Sapere Aude";
    // Тут нужно написать 1 строку кода
}
```

   Создайте указатель `p` и сделайте так, чтобы он указывал на символ `'A'` из строки `str`.

6. В следующей программе создаётся структура `Book` из семинара про структуры:

```
struct book
{
    char title[50];
    int pages;
    float price;
};
typedef struct book Book;

int main()
{
    Book b = {"Fahrenheit 451", 400, 700.0};

}
```

   (a) Создайте указатель `pb` и сделайте так, чтобы он указывал на структуру `b`.

   (b) Создайте указатель `pprice` и сделайте так, чтобы он указывал на поле `price` структуры `b`.

   (c) Создайте указатель `pc` и сделайте так, чтобы он указывал символ `'t'` поля `title` структуры `b`.

7. В следующей программе создаётся переменная `a` типа `float` и `p` указатель, который хранит её адрес:

```
int main()
{
    float a = 1.2;
    float* p = &a;
    // Тут нужно написать 1 строку кода
}
```

   Создайте указатель `pp` и сделайте так, чтобы он указывал на указатель `p`.

8. В следующей программе создаётся структура `Book` из семинара про структуры и указатель на неё:

```
struct book
{
    char title[50];
    int pages;
    float price;
};
typedef struct book Book;

int main()
{
    Book b = {"Fahrenheit 451", 400, 700.0};
    Book* pb = &b;
    // Тут нужно написать 1 строку кода
}
```

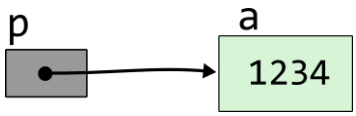
   Создайте указатель `ppb` и сделайте так, чтобы он указывал на указатель `pb`.

Задача 4. Использование указателей

Решения всех подзадач этой части – одна строка. Результат выполнения задания – `.txt` файл, который содержит все эти строки.

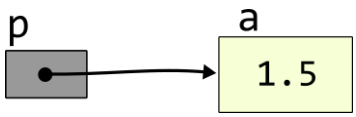
1. В следующей программе была создана переменная `a` и указатель на неё `p`. Удвойте значение переменной `a`, используя только указатель `p`. Нужно использовать указатель `p`, саму переменную `a` использовать нельзя.

```c
#include <stdio.h>
int main()
{
    int a = 1234;
    int* p = &a;
    // Тут нужно написать 1 строку кода

    printf("%i\n", a);
}
```



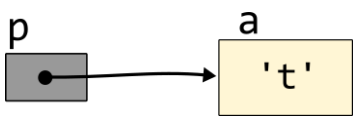
2. В следующей программе была создана переменная `a` типа `float` и указатель на неё `p`. Возведите значение переменной `a` в квадрат, используя только указатель `p`. Нужно использовать только указатель `p`, саму переменную `a` использовать нельзя.

```c
#include <stdio.h>
int main()
{
    float a = 1.5;
    float* p = &a;
    // Тут нужно написать 1 строку кода

    printf("%f\n", a);
}
```



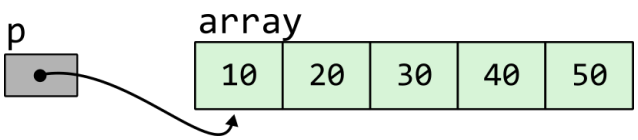
3. В следующей программе была создана переменная `a` типа `char` и указатель на неё `p`. Переведите символ, хранящийся в переменной `a` в верхний регистр, используя только указатель `p`. Нужно использовать только указатель `p`, саму переменную `a` использовать нельзя.

```c
#include <stdio.h>
int main()
{
    char a = 't';
    char* p = &a;
    // Тут нужно написать 1 строку кода

    printf("%c\n", a);
}
```



4. В следующей программе был создан массив `array` переменных типа `int` и указатель `p` на первый элемент массива.

```c
#include <stdio.h>
int main()
{
    int array[5] = {10, 20, 30, 40, 50};
    int* p = &array[0];

    for (int i = 0; i < 5; ++i)
        printf("%i ", array[i]);
}
```



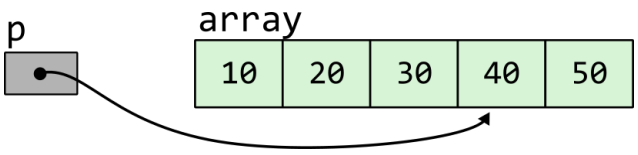
   (a) Добавьте `1` к первому элементу массива (`array[0]`), используя только указатель `p`. Нужно использовать только указатель `p`, сам массив `array` использовать нельзя. Решение – 1 строка.

   (b) Добавьте `1` к четвёртому элементу массива (`array[3]`), используя только указатель `p`. Нужно использовать только указатель `p`, сам массив `array` использовать нельзя. Менять `p` тоже нельзя. Решение – 1 строка.

   (c) Добавьте `1` ко всем элементам массива. Нужно использовать только указатель `p`, сам массив `array` использовать нельзя. Решение – 1 цикл.

5. В следующей программе был создан массив `array` переменных типа `int` и указатель `p` на четвёртый элемент массива (`array[3]`).

```c
#include <stdio.h>
int main()
{
    int array[5] = {10, 20, 30, 40, 50};
    int* p = &array[3];

    for (int i = 0; i < 5; ++i)
        printf("%i ", array[i]);
}
```



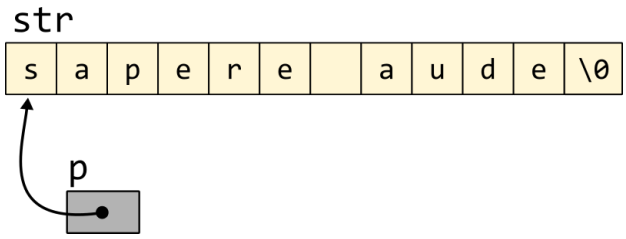
   (a) Добавьте `1` к первому элементу массива (`array[0]`), используя только указатель `p`. Нужно использовать только указатель `p`, сам массив `array` использовать нельзя. Менять `p` тоже нельзя. Решение – 1 строка.

   (b) Добавьте `1` к пятому элементу массива (`array[4]`), используя только указатель `p`. Нужно использовать только указатель `p`, сам массив `array` использовать нельзя. Менять `p` тоже нельзя. Решение – 1 строка.

   (c) Добавьте `1` ко всем элементам массива. Нужно использовать только указатель `p`, сам массив `array` использовать нельзя. Решение – 1 цикл.

6. В следующей программе была создана строка `str` и указатель `p` на первый символ строки.

```c
#include <stdio.h>
int main()
{
    int str[] = "sapere aude";
    int* p = &str[0];

    printf("%s\n", str);
}
```



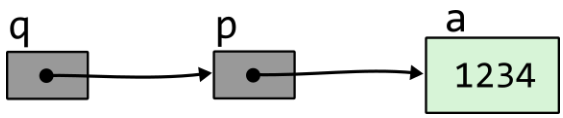
(a) Переведите в верхний регистр первую букву строки, используя только указатель `p`. Нужно использовать только указатель `p`, саму строку `str` использовать нельзя. Решение – 1 строка.

(b) Переведите в верхний регистр первую букву второго слова строки, используя только указатель `p`. Нужно использовать только указатель `p`, саму строку `str` использовать нельзя. Решение – 1 строка.

(c) Переведите в верхний регистр все буквы строки, используя только указатель `p`. Нужно использовать только указатель `p`, саму строку `str` использовать нельзя. Решение – 1 цикл

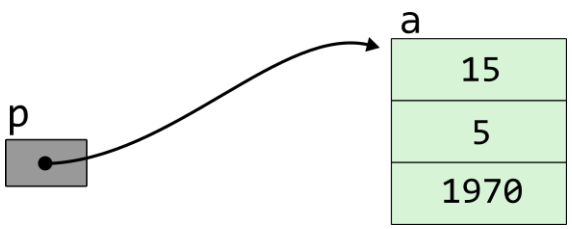
7. В следующей программе есть переменная `a` типа `int`, указатель `p` на эту переменную и указатель `q` на `p`. Удвойте значение переменной `a`, используя только указатель `q`. Нужно использовать только указатель `q`.

```c
#include <stdio.h>
int main()
{
    int a = 1234;
    int* p = &a;
    int** q = &p;
    // Тут нужно написать 1 строку кода
    printf("%i\n", a);
}
```



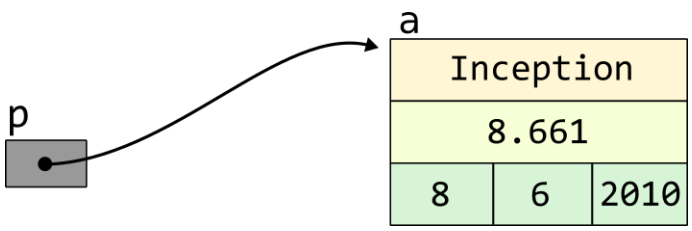
8. В следующей программе была создана структура `a` типа `Date` и указатель на эту структуру. Добавьте `1` к значению поля `year`, используя только указатель `p`.

```c
#include <stdio.h>
struct date
{
    int day, month, year;
};
typedef struct date Date;
int main()
{
    Date a = {15, 5, 1970};
    Date* p = &a;
    // Тут нужно написать 1 строку кода
    printf("%d %d %d\n",
           a.day, a.month, a.year);
}
```



9. В следующей программе была создана структура `a` типа `Movie` и указатель на неё.

```c
#include <stdio.h>
struct date
{
    int day, month, year;
};
typedef struct date Date;

struct movie
{
    char title[50];
    float rating;
    Date release_date;
};
typedef struct movie Movie;
int main()
{
    Movie a = {"Inception", 8.661, {8, 6, 2010}};
    Movie* p = &a;
    // Тут нужно написать 1 строку кода
}
```

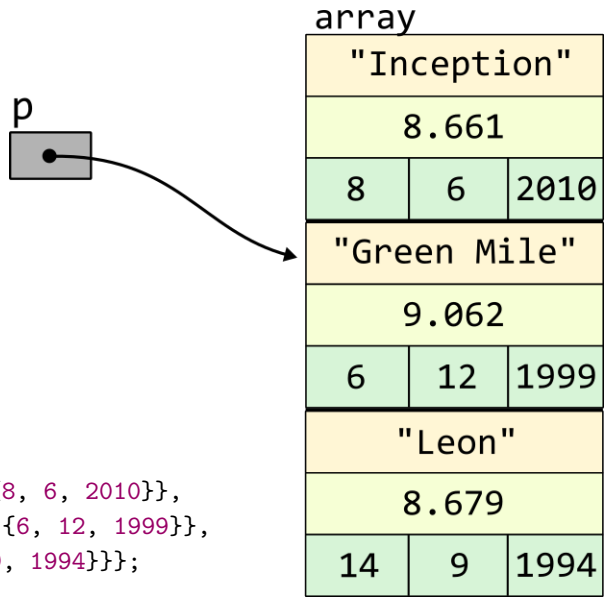


(a) Увеличьте на 1 значение поля `rating`, используя только указатель `p`.

(b) Увеличьте на 1 значение поля месяца выхода фильма, используя только указатель `p`.

10. В следующей программе был создан массив `array` из структур типа `Movie` и указатель `p`, который указывает на второй элемент массива (`array[1]`).

```c
#include <stdio.h>
struct date
{
    int day, month, year;
};
typedef struct date Date;

struct movie
{
    char title[50];
    float rating;
    struct date release_date;
};
typedef struct movie Movie;

int main()
{
    Movie array[3] = {{"Inception", 8.661, {8, 6, 2010}},
                      {"Green Mile", 9.062, {6, 12, 1999}},
                      {"Leon", 8.679, {14, 9, 1994}}};
    Movie* p = &array[1];
}
```



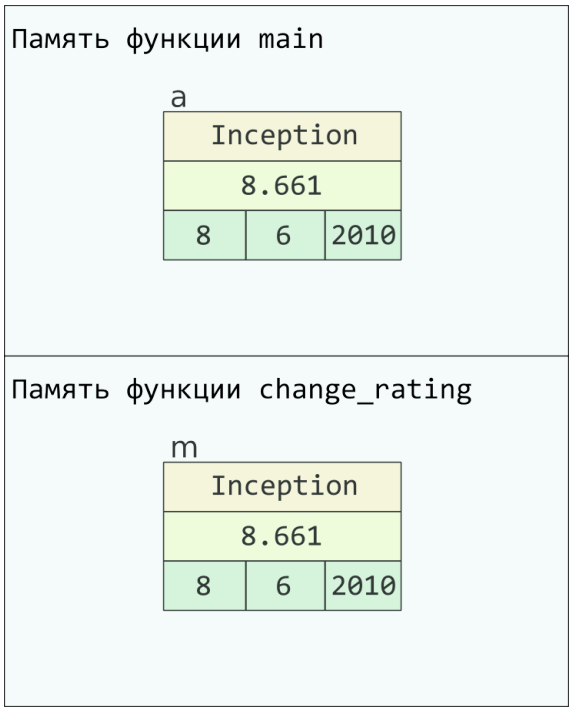
(a) Увеличьте на 1 значение рейтинга фильма `Inception`, используя только указатель `p`. При этом менять `p` нельзя, он должен указывать на `array[1]`.

(b) Удвойте значение года выхода фильма `Leon`, используя только указатель `p`. При этом менять `p` нельзя, он должен указывать на `array[1]`.

Задача 5. Передача в функцию по указателю

Передача в функцию по значению

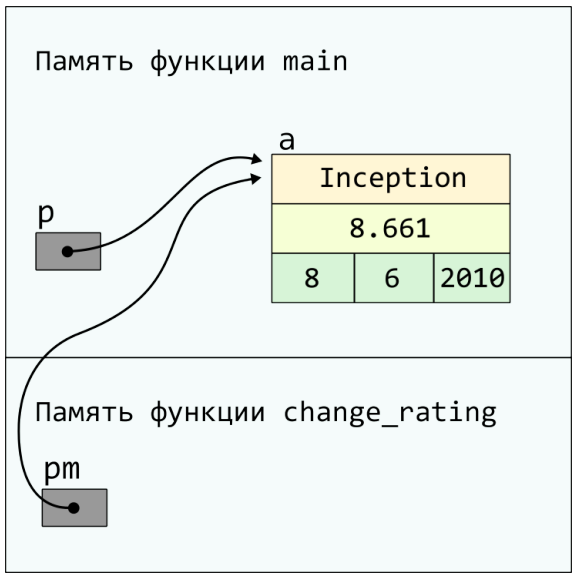
```c
#include <stdio.h>
struct movie
{
    char title[50];
    float rating;
    struct date release_date;
};
typedef struct movie Movie;

void change_rating(Movie m)
{
    m.rating += 1;
}
int main()
{
    Movie a = {"Inception", 8.661,
                {8, 6, 2010}};
    change_rating(a);
}
```



Всё, что передаётся в функцию, копируется (кроме массивов). Поэтому функция `change_rating` будет менять поле `rating` у копии структуры `a`, а изначальная структура не изменится.

Передача в функцию по указателю:

```c
#include <stdio.h>
struct movie
{
    char title[50];
    float rating;
    struct date release_date;
};
typedef struct movie Movie;

void change_rating(Movie* pm)
{
    pm->rating += 1;
}
int main()
{
    Movie a = {"Inception", 8.661,
                {8, 6, 2010}};
    Movie* p = &a;
    change_rating(&a);
}
```



Теперь в функцию копируется указатель, который содержит адрес стурктуры `a`. Используя этот указатель, мы можем изменить изначальную структуру. Более того, так как указатель занимает меньше памяти, его копирования в функцию происходит быстрее, чем копирование всей структуры.

Подзадачи:

1. Напишите функцию `void increase_rating(Movie* p)`, которая будет принимать указатель типа `Movie*` и увеличивать рейтинг фильма, на которой указывает `p`, на 1.

2. Напишите функцию `void change_year_of_movies(Movie* p, int size)`, которая принимает на вход указатель на первый элемент массива структур типа `Movie` и размер этого массива. Функция должна увеличивать год выхода всех фильмов на 1. Протестируйте функции, вызвав их из функции `main` с помощью следующего кода:

```
#include <stdio.h>
struct date
{
    int day, month, year;
};
typedef struct date Date;
struct movie
{
    char title[50];
    float rating;
    struct date release_date;
};
typedef struct movie Movie;

void print_date(const Date* pd)
{
    printf("%02d.%02d.%04d", pd->day, pd->month, pd->year);
}
void print_movie(const Movie* pm)
{
    printf("Title: %s\nRating: %.2f\nDate: ", pm->title, pm->rating);
    print_date(&pm->release_date);
    printf("\n");
}
// Тут вам нужно написать функции increase_rating и change_year_of_movies

int main()
{
    Movie a[3] = {{"Inception", 8.661, {8, 6, 2010}},
                  {"Green Mile", 9.062, {6, 12, 1999}},
                  {"Leon", 8.679, {14, 9, 1994}}};
    increase_rating()
    change_year_of_movies(a, 3);
    for (int i = 0; i < 3; ++i)
        print_movie(&a[i]);
}
```

Задача 6. Структура Актёр

Напишем структуру `Actor`, в которой будем хранить информацию об одном актёре:

```
struct actor
{
    char name[32];
    char surname[32];
    int gender;
    int height;
    Date birth_date;
    Address birth_address;
};
typedef struct actor Actor;
```

Поля структуры `Actor`:

- `name` – имя актёра
- `surname` – фамилия
- `gender` – пол (`0`, если это мужчина; `1`, если это женщина)
- `height` – рост в сантиметрах
- `birth_date` – дата рождения (структура, содержащая 3 числа)
- `birth_address` – место рождения (структура, содержащая 3 строки: страна, регион и город)

Файл `actors.csv`:

В файле `actors.csv` содержится информация о 2000 актёрах (все данные сгенерированы случайным образом). Файл имеет следующий вид:

```
2000
Abel,Garifullin,0,189,16/2/1992,Russia,Rostovskaya Oblast,Rostov-na-Donu
Viktor,Shchyotkin,0,162,28/6/1992,Russia,Samarskaya Oblast,Samara
Sophia,Sigayeva,1,148,30/1/1963,Russia,Kurskaya Oblast,Zheleznogorsk
Vlada,Solodnikova,1,163,16/7/2004,Russia,Sverdlovskaya Oblast,Polevskoy
... (всего 2000 записей) ...
```

Файлы формата `.csv` можно открывать как обычным текстовым редактором, так и с помощью программы для работы с табличными данными (например, Excel).

Подзадачи:

В файле `actors.c` содержится начальный код, нужный для решения следующих задач.

1. **Заданный рост:** Напишите функцию, которая будет принимать на вход массив из актёров и заданный рост и будет печатать всех актёров, которые имеют этот рост. Прототип функции:
   `void print_all_actors_by_height(const Actor* actors, int number_of_actors, int height)`
2. **Заданный город:** Напишите функцию, которая будет принимать на вход массив из актёров и название города и будет печатать всех актёров, которые родились в этом городе. Прототип функции:
   `void print_all_actors_by_city(const Actor* actors, int number_of_actors, char city[])`
   Для сравнения строк используйте функцию `strcmp` из библиотеки `string.h`.

Задача 7. Структуры Фильм и структура База Фильмов

Напишем структуру `Movie`, в которой будет хранить всю информацию об одном фильме:

```
struct movie
{
    char title[50];
    Date release_date;
    double rating;
    int crew_size;
    int crew[20];
};
typedef struct movie Movie;
```

Поля структуры `Movie`:

- `title` – название фильма (не более 49 символов)
- `release_date` – дата выхода фильма (структура `Date`)
- `rating` – рейтинг фильма
- `crew_size` – количество актёров, задействованных в этом фильме
- `crew` – индексы актёров в массиве `actors` структуры `MovieDatabase`. Нумерация начинается с `0`.

Также напишем структуру `MovieDatabase` в котором будем хранить информацию о базе данных фильмов. Она будет содержать информацию о множестве актёров и фильмов. При этом массив актёров и фильмов будут иметь фиксированный размер (потому что создавать массив произвольной длины мы пока не умеем – научимся только на следующих семинарах).

```
struct movie_database
{
    int number_of_actors;
    Actor actors[5000];
    int number_of_movies;
    Movie movies[5000];
};
typedef struct movie_database MovieDatabase;
```

Поля структуры `MovieDatabase`:

- `number_of_actors` – количество актёров в базе данных (не более 5000)
- `actors` – массив из всех актёров
- `number_of_movies` – количество фильмов в базе данных (не более 5000)
- `movies` – массив из всех фильмов

Это, конечно, не самый лучший способ для работы с базой данных. Гораздо лучший способ – использование систем управления базами данных и библиотек для работы с ними, но это выходит за рамки данного курса.

Файл `movies.csv`:

В файле `movies.csv` содержится информация о 4000 фильмах (все данные сгенерированы случайным образом). Файл имеет следующий вид:

```
4000
Dingy King,14/1/1980,7.402,2,1485 1932
Admire The Home,28/9/1973,6.504,9,673 814 1087 926 38 1378 629 1080 71
Egocentric Airport,24/7/1983,4.773,11,116 1747 958 40 892 1403 1752 338 62 590 1861
Stuff And The Heat,27/12/1995,6.013,9,1574 53 692 210 908 463 705 232 1582
... всего 4000 записей ...
```

Передача структур в функции:

Видно, что структура `MovieDatabase` имеет очень большой размер (`1680016` байт!). Передавать такой размер в функцию по значению вот так:

```
void some_function(MovieDatabase md, ...)
```

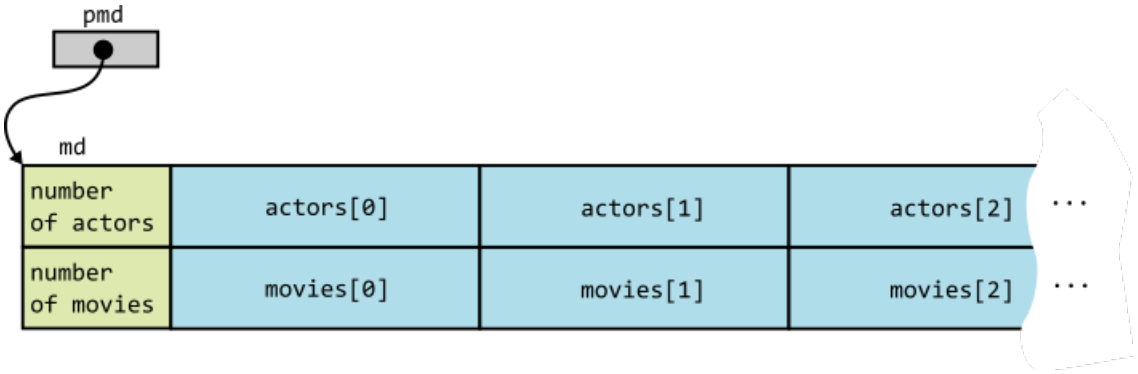
очень плохая идея. Ведь при передаче в функцию всё копируется и это означает, что при каждом вызове такой функции будет происходить копирование всей базы фильмов. Решение – использование указателей:

```
void some_function(MovieDatabase* pmd, ...)
```

Теперь при вызове функции копироваться будет только указатель (всего 8 байт) и, зная адрес структуры, мы сможем получать доступ ко всем её элементам как и раньше. Однако, передавая так структуру в неизвестную нам функцию (например, функцию, которую написал другой программист), мы не можем гарантировать, что она не изменится внутри. Это ведёт к усложнению программирования, так как теперь нам нужно следить за всеми структурами при их передаче в функции (а это не так просто, ведь функции могут вызывать другие функции, а исходный код многих библиотечных функций может быть вообще неизвестен). Решение этой проблемы – использование модификатора (`const`):

```
void some_function(const MovieDatabase* pmd, ...)
```

Теперь структуру на которую указывает `pmd` нельзя поменять внутри функции.



Подзадачи:

В файле `movies.c` содержится начальный код, нужный для решения следующих задач.

1. **Лучший фильм x4:** Напишите 4 функции, каждая из которых будет находить лучший фильм, при этом возвращая результат разными путями.

   - `Movie find_best_movie_value(const MovieDatabase* pmd)`
     Возвращает структуру
   - `int find_best_movie_index(const MovieDatabase* pmd)`
     Возвращает номер фильма – индекс в массиве `pmd->movies`
   - `const Movie* find_best_movie_pointer(const MovieDatabase* pmd)`
     Возвращает указатель на нужную структуру
   - `void find_best_movie_argument(const MovieDatabase* pmd, Movie* p_best_movie)`
     Записывает лучший фильм в структуру по адресу `p_best_movie`.

   Вызовите все эти функции из `main`.

2. **Фильмография:** На вход подаётся 2 строки: имя и фамилия актёра. Напечатайте все фильмы с его участием.

3. **Лучший актёр:** Напишите функцию, которая будет находить лучшего актёра (актёра с самым большим средним рейтингом фильмов с его/её участием). Вызовите эту функцию из `main` и напечатайте этого актёра на экран.

4. **Фильмы года:** Напечатайте на экран все фильмы, вышедшие в определённый год. все фильмы должны быть отсортированы по рейтингу (от лучшего к худшему).