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. Для печати используйте спецификатор `%.2f`.
- 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 на вход треугольник и возвращать периметр этого треугольника.
- Функцию `get_triangle_area`, которая будет принимать на вход треугольник и возвращать его площадь. Можно использовать формулу Герона.
- Функцию `moved_triangle`, которая будет принимать на вход треугольник 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 (обязательно по указателю) и одну точку(она будет играть роль вектора-перемещения). Функция должна менять передаваемый по указателю треугольник.
- Функцию `void rotated_triangle(const Triangle* t, const Point* origin, double alpha)`, которая будет создавать новый треугольник, который будет являться результатом поворота треугольника `*t` вокруг точки `*origin` на угол `alpha` радиан против часовой стрелки. Функция должна возвращать этот новый треугольник.
- Функцию `void rotate_triangle(Triangle* t, const Point* origin, double alpha)`, которая будет вращать треугольник `*t` вокруг точки `*origin` на угол `alpha` радиан против часовой стрелки.

  Формула новых координат при повороте на угол α:

$$x' = x \cdot \cos(\alpha) - y \cdot \sin(\alpha)$$
$$y' = x \cdot \sin(\alpha) + y \cdot \cos(\alpha)$$

Проверьте все эти функции в `main`. Для передачи структур в функции лучше использовать передачу по указателю либо по указателю на константу.

Задача 2: 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).

Задачи:

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 3: 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 50 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, ...)
```

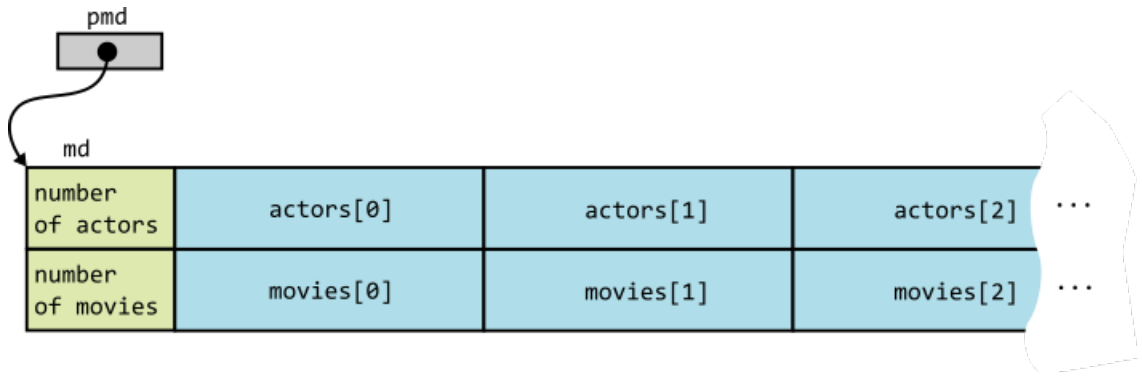
очень плохая идея. Ведь при передаче в функцию всё копируется и это означает, что при каждом вызове такой функции будет происходить копирование всей базы фильмов. Решение – использование указателей:

```
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.



Задачи:

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`
   - `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).

5. **Фильмы по городу:** На вход подаётся название города. Нужно найти все фильмы в которых играет хотя бы один актёр, родившийся в этом городе. Все эти фильмы нужно отсортировать по дате выхода (от старых к новым) и сохранить в файл `movies_of_actors_from_<название города>.txt`. При записи в файл нужно напечатать не только информацию о фильме, но и информацию о всех актёрах, которые в нём играют.