Семинар #8: Указатели и динамическое выделение памяти. Домашнее задание.

Память

Как выглядит память, инициализируемая при создании следующих переменных (в системе с порядком байт Little Endian):

1. `int a = 0x11223344;`
2. `int b = 123456789;`
3. `int array[3] = {10, 2000, 65535};`
4. `char str[8] = ”Hello“;`
5. `float x = -15.91;`
6. `double y = -15.91;`
7.
```
struct data
{
    char str[5];
    int number;
};
struct data c = {"Cat", 100000};
```

Память представить в виде последовательности 2-значных шестнадцатиричных чисел. Например число `int a = 757004;` будет храниться в памяти как `0x54, 0x82, 0x73, 0x00`.

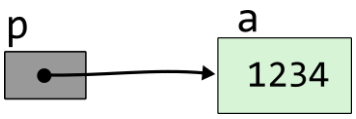
Подсказка: Чтобы проверить, как будет выглядеть память, можно создать указатель типа `char*` на эту память и распечатать каждый байт в виде шестнадцатиричного числа:

```
char* p = (char*)&a;
for (int i = 0; i < sizeof(a); ++i)
{
    printf("0x%02hhx ", p[i]);
}
```

Указатели

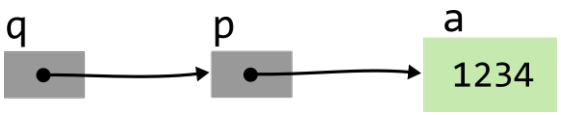
Указатель на `int`

```
int a = 1234;
int* p = &a;
```



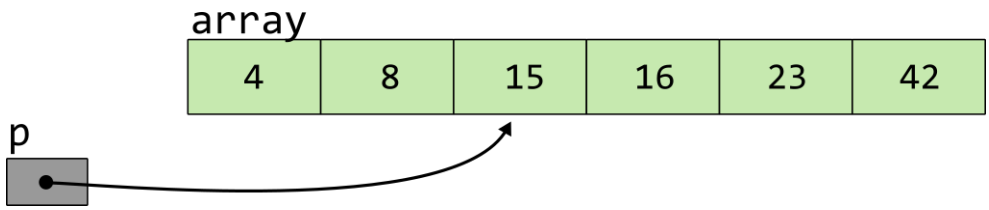
Указатель на указатель на `int`

```
int a = 1234;
int* p = &a;
int** q = &p;
```



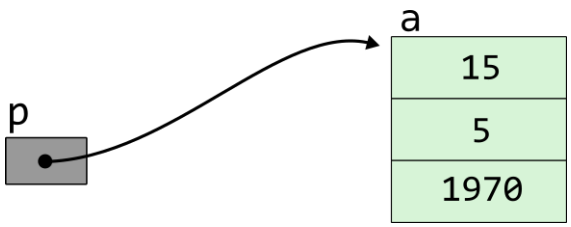
Указатель на элемент массива

```
int array[5] = ;
int* p = &a[1];
```



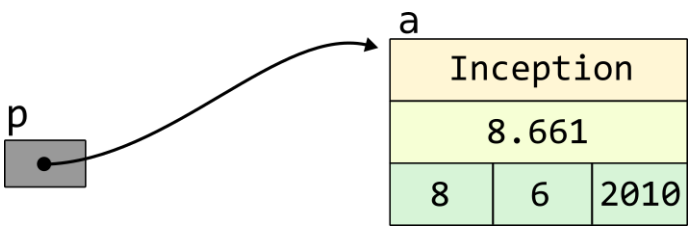
Указатель на элемент структуру

```
struct date
{
        int day, month, year;
};
struct date a = {};
struct date* p = &a;
```



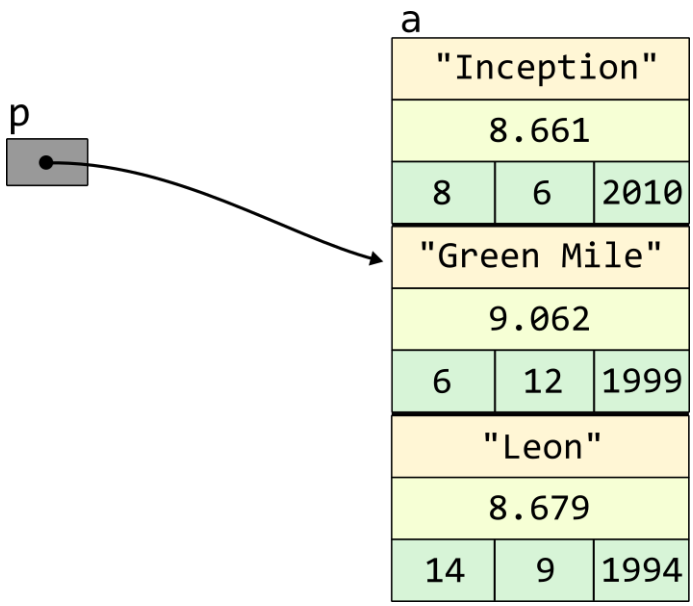
Указатель на элемент структуру Movie

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

Movie a = {"Inception", 8.661, {8, 6, 2010}};
Movie* p = &a;
```



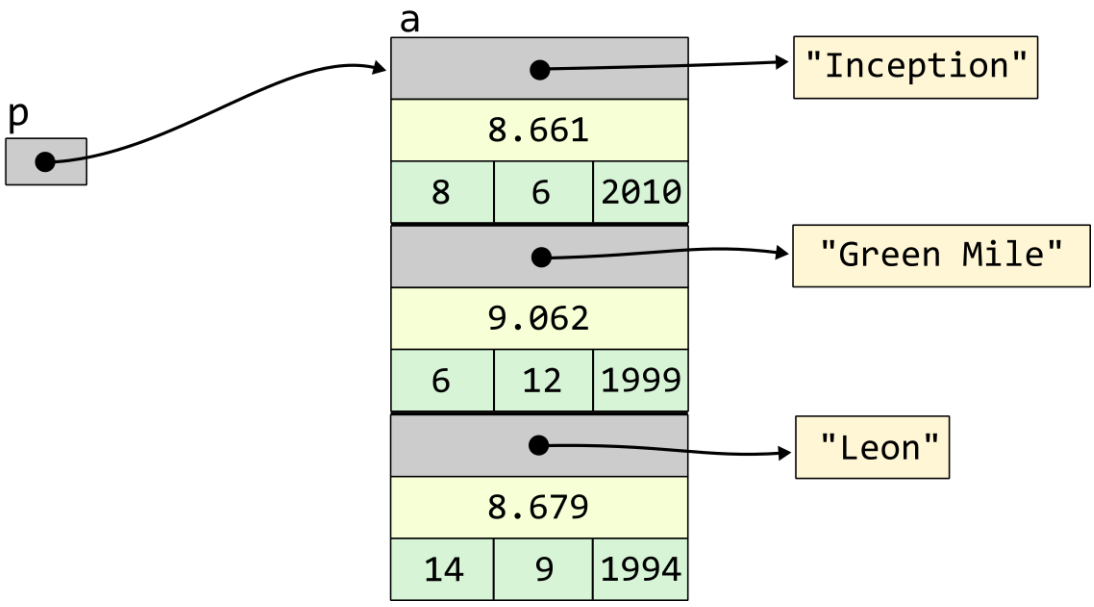
Указатель на массив структур Movie

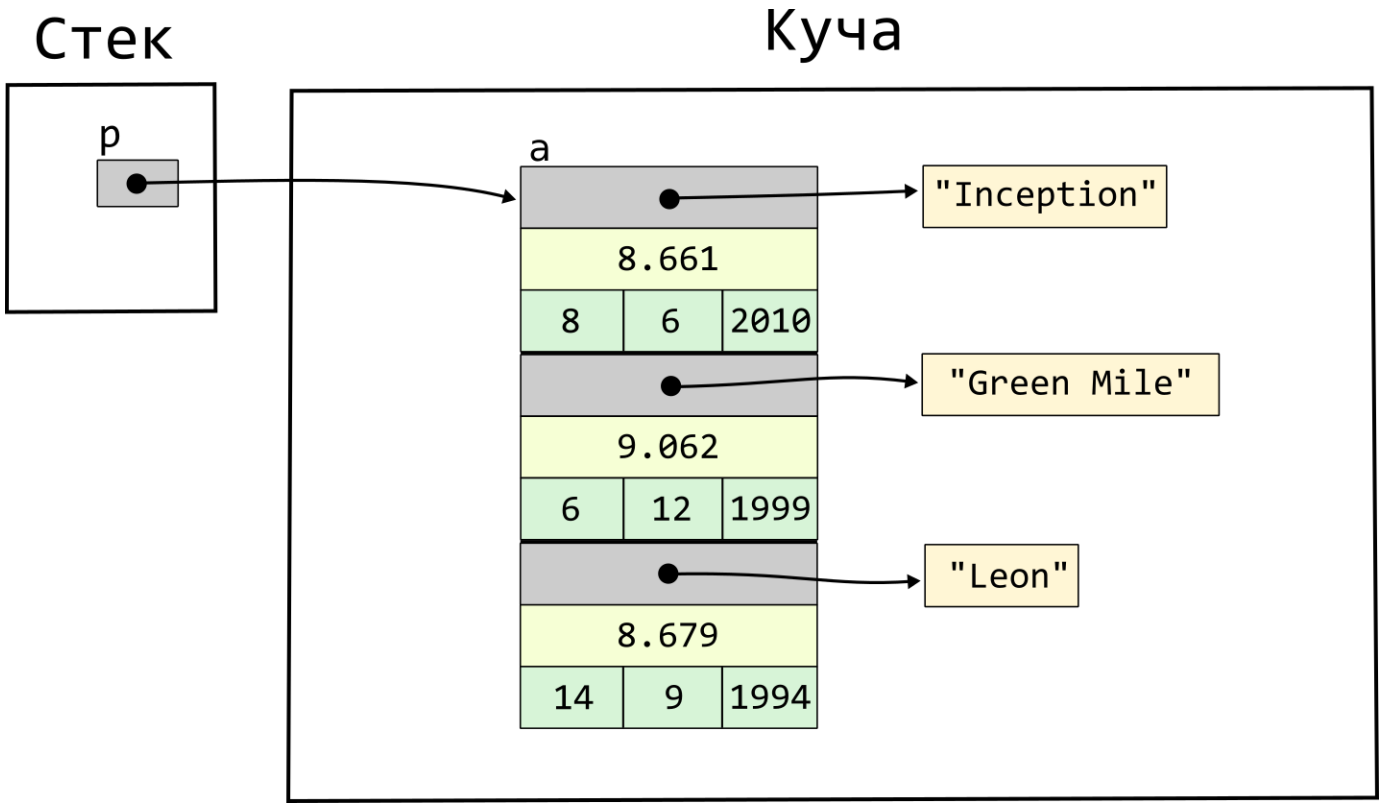
```
struct movie
{
        char title[50];
        float rating;
        struct date release_date;
};
typedef struct movie Movie;

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];
```



Указатель на массив структур, выделенный в куче

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

int main()
{
        Movie* p = (Movie*)malloc(3 * sizeof(Movie));
        strcpy(p[0].title, "Inception");
        p[0].rating = 8.661;
        p[0].release_date
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];
```





У каждой переменной есть адрес. Адрес - это номер байта, начиная с которого лежит эта переменная в памяти. Чтобы найти адрес переменной, нужно перед ней поставить знак амперсанда `&`.
Для хранения адресов в языке C введены специальные переменные, которые называются указатели. Тип переменной указателя = тип той переменной, на которую он ‘указывает‘ + звёздочка(`*`) на конце. Например, указатель, который будет хранить адреса переменных типа `int` должен иметь тип `int*`.
Чтобы по указателю получить саму переменную, нужно перед указателем поставить звёздочку(`*`).

```c
#include <stdio.h>
int main()
{
    int a = 7;
    printf("Value = %d. Address = %p\n",  a,  &a);
```

```
    int* pa = &a;
    printf("Value = %d. Address = %p\n", *pa, pa);
}
```

Выполните задание из файла `0pointer_basics.c`.

Арифметика указателей

С указателями можно производить следующие операции:

- Прибавить или отнять число `p + 2`
- Вычитать 2 указателя `p - q`
- Разыменование (получить то, на что указывает указатель) `*p`
- Квадратные скобки (прибавить число + разыменование): `p[i] == *(p+i)`

Пусть есть одномерный статический массив и указатель на 4-й элемент этого массива:

```
int numbers[6] = {4, 8, 15, 16, 23, 42};
int* p = &numbers[3];
```

Чему равны следующие выражения:

1. `numbers[5]`
2. `*p`
3. `*(p+1)`
4. `*(p-2)`
5. `p[0]`
6. `p[1]`
7. `p[-2]`
8. `*numbers`
9. `*(numbers+5)`
10. `p - numbers`
11. `(short*)p - (short*)numbers`
12. `(char*)p - (char*)numbers`

Подсказка: имя массива во многих случаях ведёт себя как указатель на первый элемент массива.
Выполните задание из файла `1pointerarith.c`.

Malloc и free:

Основные функции для динамического выделения памяти:

- `void* malloc(size_t n)` – выделяет n байт и возвращает указатель `void*` на начало этой памяти
- `void free(void* p)` – освобождает выделенную память
- `void* realloc(void* p, size_t new_n)` – перевыделяет выделенную память

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
        // Выделяем 50 байт памяти, адрес первого байта будет храниться в указателе p
        void* p = malloc(50);

        // Освободим только - что выделенные 50 байт
        // Память можно освободить в люой момент выполнения, экономя память
        free(p);
```

```
        // Выделяем 12 байт памяти, с указателем p1 теперь
        //       можно обращаться как с массивом размера 3
        int* p1 = malloc(12);

        // Выделяем объём памяти достаточный для хранения 15 - ти int - ов
        int* p2 = malloc(15 * sizeof(int));

        // Теперь с p1 и p2 можно работать также как и с массивами типа int
        // То есть можно применять операции типа p1[2]
        // И p1 и p2 будут вести себя как массива размера 3 и 15 соответственно
        for (int i = 0; i < 3; ++i)
                scanf("%d", &p1[i]);
        printf("%d", p1[0] + p1[2]);

        // Увеличим размер нашего массива с 15 до ти25-
        p2 = realloc(p2, 25 * sizeof(int));

        // Не забывайте освобождать ненужную память!
        free(p1);
        free(p2);
}
```

Задачи:

- Выделить 123 байта памяти и записать адрес на эту память в указатель типа `void*`.
- Выделить память для хранения 10 элементов типа `unsigned long long`.
- Выделить память для хранения 100 элементов типа `float*`.
- Выделить память для хранения 10 элементов типа `double`. Изменить размер этого динамического массива с 10 до 50, используя `realloc`.
- Освободить всю память, которую вы выделили.