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

Память

Задачи #1-7: 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 шестнадцатеричных чисел. Например число `int a = 757004;` будет храниться в памяти как `0x54, 0x82, 0x73, 0x00`.

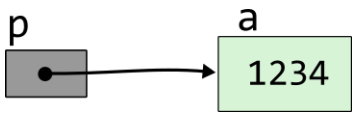
Подсказка: Чтобы проверить, как будет выглядеть память, можно создать указатель типа `char*` на эту память и распечатать каждый байт в виде шестнадцатеричного числа:

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

Указатели

Указатель на `int`

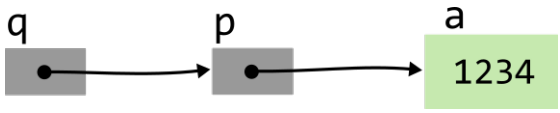
```
int a = 1234;
int* p = &a;
```



Задача #8: Удвойте значение переменной `a`, используя только указатель `p`.

Указатель на указатель на `int`

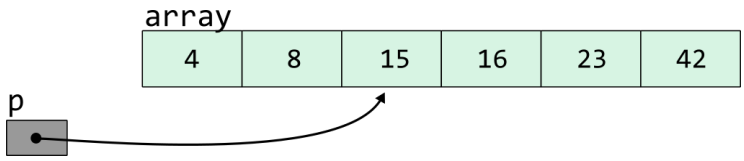
```
int a = 1234;
int* p = &a;
int** q = &p;
```



Задача #9: Удвойте значение переменной `a`, используя только указатель `q`.

Указатель на элемент массива

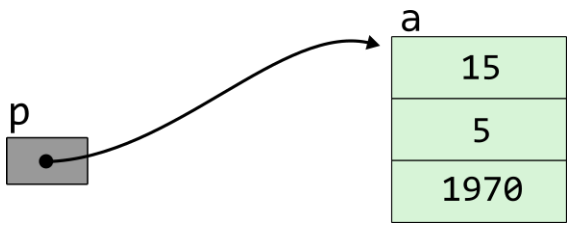
```c
int main()
{
  int array[6] = {4, 8, 15, 16, 23, 42};
  int* p = &array[2];
}
```



Задача #10: Удвойте значение `array[1]`, используя только указатель `p` (не меняя `p`, он должен указывать на `array[2]`).

Указатель на структуру

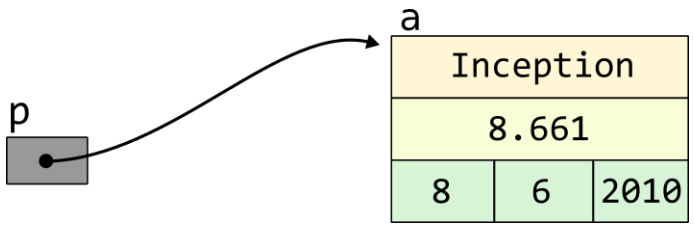
```c
#include <stdio.h>
struct date
{
        int day;
        int month;
        int year;
};

int main()
{
        struct date a = {15, 5, 1970};
        struct date* p = &a;
        printf("%d %d %d\n", a.day, (*p).day, p->day);
}
```



Задача #11: Удвойте значение поля `year`, используя только указатель `p`.

Указатель на структуру немного сложнее

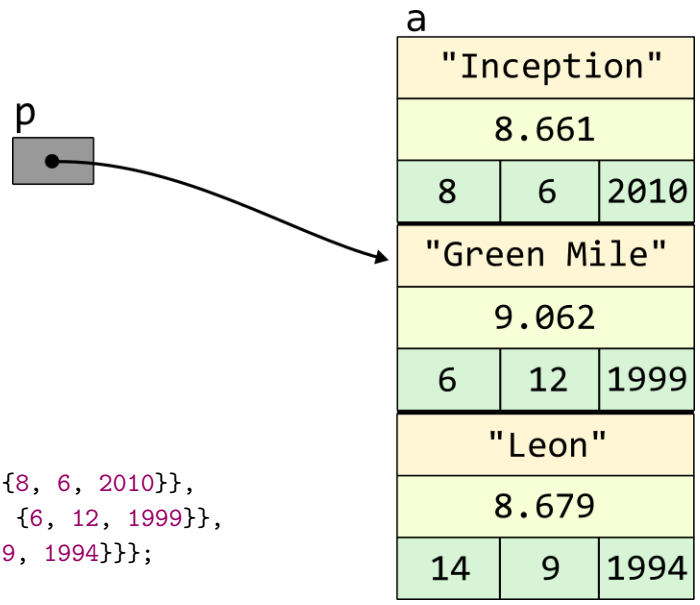
```c
#include <stdio.h>
struct movie
{
    char title[50];
    float rating;
    struct date release_date;
};
typedef struct movie Movie;


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



Задача #12: Удвойте значение поля `rating`, используя только указатель `p`.
Задача #13: Удвойте значение поля месяца выхода фильма, используя только указатель `p`.

Указатель на массив структур

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

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

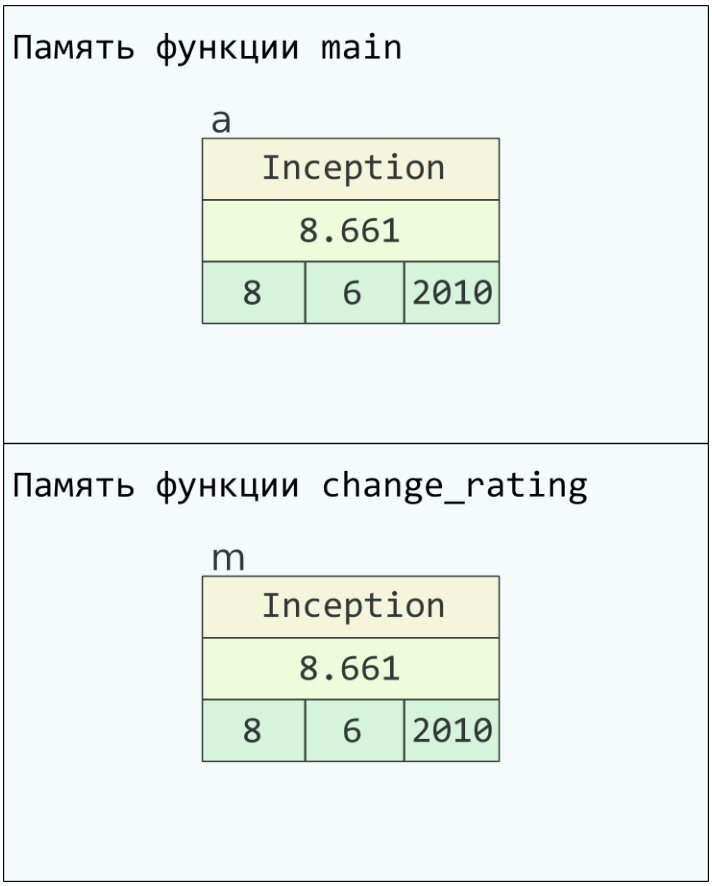


Задача #14: Удвойте значение рейтинга фильма `Inception`, используя только указатель `p`. (не меняя `p`, он должен указывать на `a[1]`)

Задача #15: Удвойте значение года выхода фильма `Leon`, используя только указатель `p`. (не меняя `p`, он должен указывать на `a[1]`).

Передача в функцию по значению

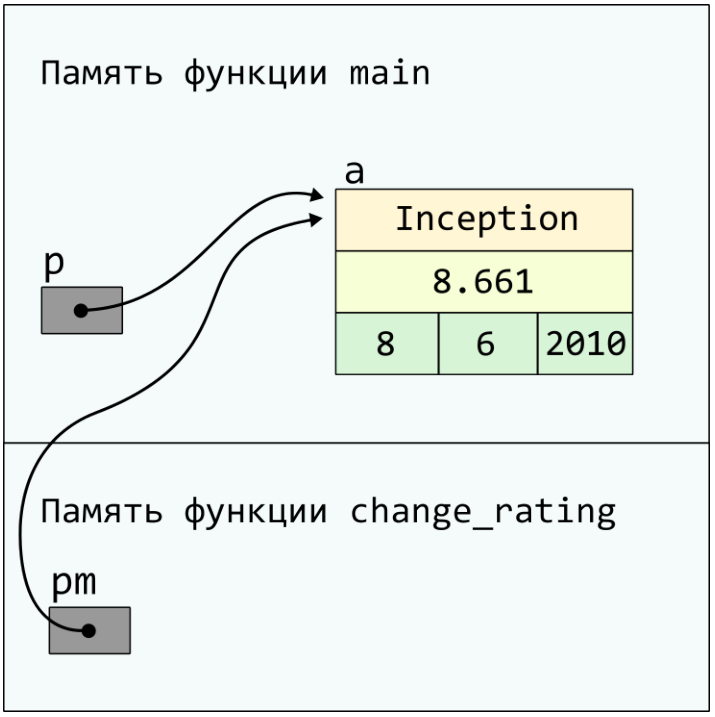
```c
#include <stdio.h>

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. 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);
}
```



Всё, что передаётся в функцию, копируется. Но теперь туда копируется указатель, который содержит адрес стурктуры `a`. Используя этот указатель, мы можем изменить изначальную структуру. Более того, так как указатель занимает меньше память, его копирования в функцию происходит быстрее, чем копирование всей структуры.

Задача #16: Напишите функцию `change_day(struct date* pd)`, которая будет увеличивать день на `1`. Используйте эту функцию, чтобы увеличить день даты выхода на `1` у структуры `a`.

Memory allocation (Выделение памяти)

Основные функции для динамического выделения памяти в сегменте памяти Куча (библиотека `stdlib.h`):

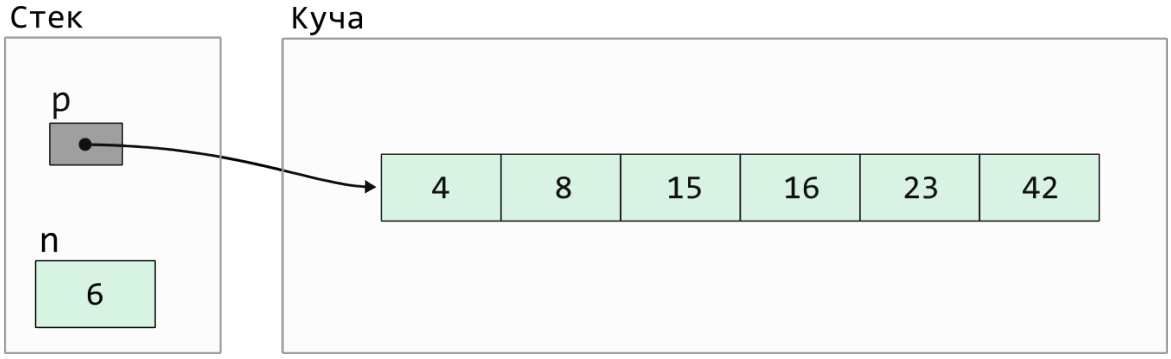
- `void* malloc(size_t n)` – выделяет `n` байт и возвращает указатель `void*` на начало этой памяти. Если память выделить не получилось (например памяти не хватает), то функция вернёт значение `NULL`. (`NULL` – это просто константа равная нулю).

- `void free(void* p)` – освобождает выделенную память. Если ненужную память вовремя не освободить, то она останется помеченной, как занятая до момента завершения программы. Произойдёт так называемая утечка памяти.

- `void* realloc(void* p, size_t new_n)` – перевыделяет выделенную память. Указатель `p` должен указывать на ранее выделенную память. Память, на которую ранее указывал `p`, освободится.
  Если память перевыделить не получилось (например памяти не хватает), то функция вернёт значение `NULL`. При этом указатель `p` будет продолжать указывать на старую память, она не освободится.

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

int main()
{
        int n = 6;
        int* p = (int*)malloc(n * sizeof(int));
        // Работаем с p как будто это массив
        p[0] = 4;
        p[1] = 8;
        p[2] = 15;
        p[3] = 16;
        p[4] = 23;
        p[5] = 42;
        // К сожалению инициализировать такой массив через скобочки нельзя

        // После того, как массив становится не нужен, память нужно освободить:
        free(p);
}
```

Схематическое изображение памяти, выделенной в коде выше:

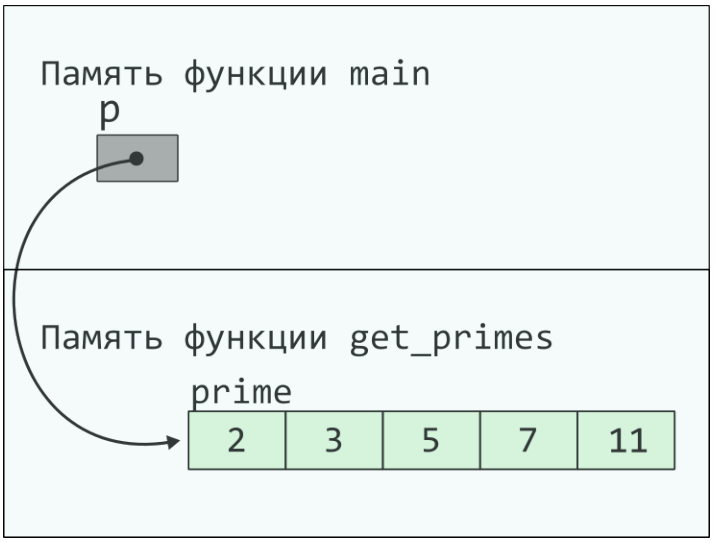


Задача #17: Создайте массив из 10^7 чисел типа `double` и инициализируйте его корнями целых чисел (т.е. `p[i] = sqrt(i)`). Напечатайте последний элемент этого массива. Можно ли создать такой большой массив на стеке?

Возврат массива из функции (ошибочный способ)

Допустим мы хотим создать функцию, которая должна будет возвращать массив. Известно, что при передаче массива в функцию он всегда передаётся по указателю. Можно попытаться создать статический массив внутри функции и вернуть указатель на него как это сделано в примере ниже. Однако это является грубой ошибкой. Дело в том, что память на стеке выделяется в начале выполнения функции и освобождается по выходу из функции. Таким образом, возвращаемый указатель будет ссылаться на уже освобождённую память. Ошибка!

```c
#include <stdio.h>

int* get_primes()
{
        int primes[5] = {2, 3, 5, 7, 11};
        int* result = &primes[0];
        return result;
}
int* other_function()
{
        int arr[5] = {55, 66, 77, 88, 99};
        int* result = &arr[0];
        return result;
}
int main()
{
        int* p = get_primes();
        // other_function();
        printf("%d\n", p[4]);
}
```

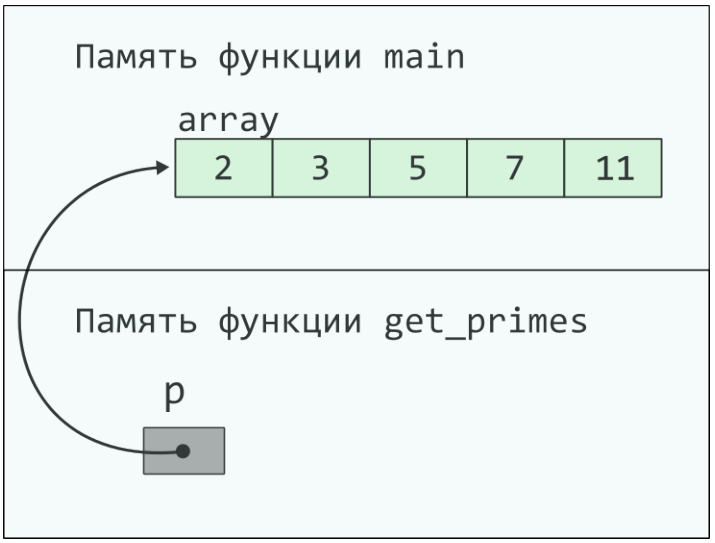


Что напечатает программа, если раскомментить строку?

Возврат массива из функции (передавая указатель как аргумент)

Другой способ “возврата” массива: передадим функции указатель на уже созданный массив и попросим его заполнить. При этом нужно следить, чтобы функция не вышла за пределы массива. Так мы передавали массивы в функции в предыдущих семинарах.

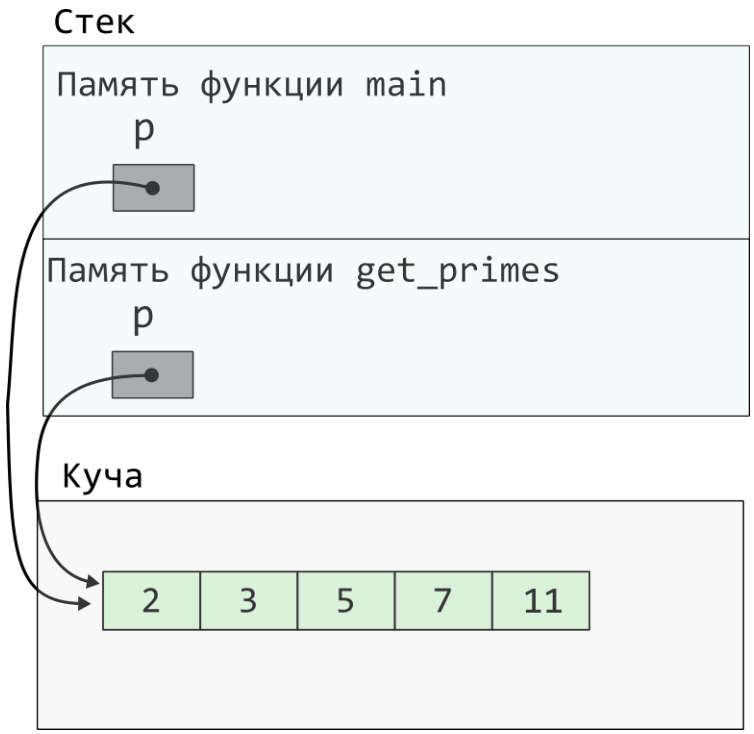
```c
#include <stdio.h>

void get_primes(int* p)
{
        p[0] = 2;
        p[1] = 3;
        p[2] = 5;
        p[3] = 7;
        p[4] = 11;
}
int main()
{
        int array[100];
        get_primes(array);
        for (int i = 0; i < 5; ++i)
                printf("%d ", array[i]);
}
```



Возврат массива из функции (через динамическую память)

Наконец, ещё один, новый способ возврата массива из функции – это создание массива в куче с помощью `malloc` и возвращение указателя на него. При завершении функции, выделенная в куче память не освобождается и этот массив можно использовать. Только нужно не забыть вызвать `free`, когда массив станет не нужен.

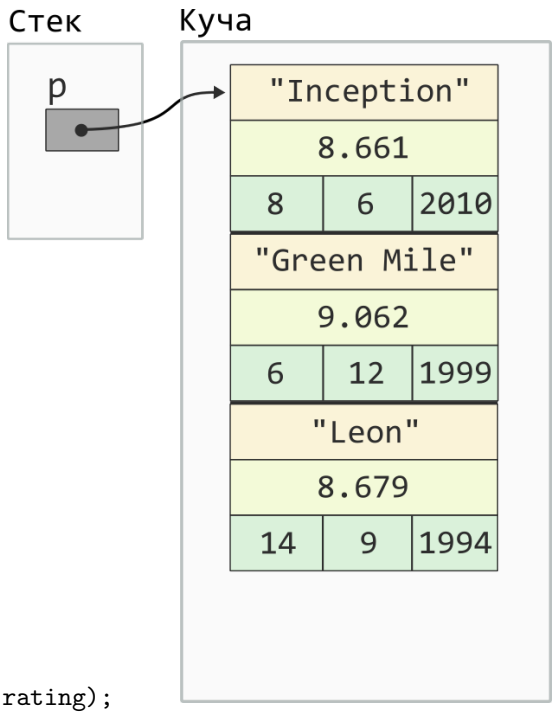
```c
#include <stdlib.h>
#include <stdio.h>

int* get_primes()
{
        int* p = (int*)malloc(5*sizeof(int));
        p[0] = 2;
        p[1] = 3;
        p[2] = 5;
        p[3] = 7;
        p[4] = 11;
        return p;
}

int main()
{
        int* p = get_primes();
        for (int i = 0; i < 5; ++i)
                printf("%d ", p[i]);
        free(p);
}
```



Задача #18: Напишите функцию `float* get_geometric_progression(float a, float r, int n)`, которая возвращает указатель на динамический массив, содержащий геометрическую прогрессию из n чисел: $a, ar, ar^2, \ldots$ Память должна выделяться динамически. Вызовите эту функцию из `main` и напечатайте первые 10 степеней тройки.

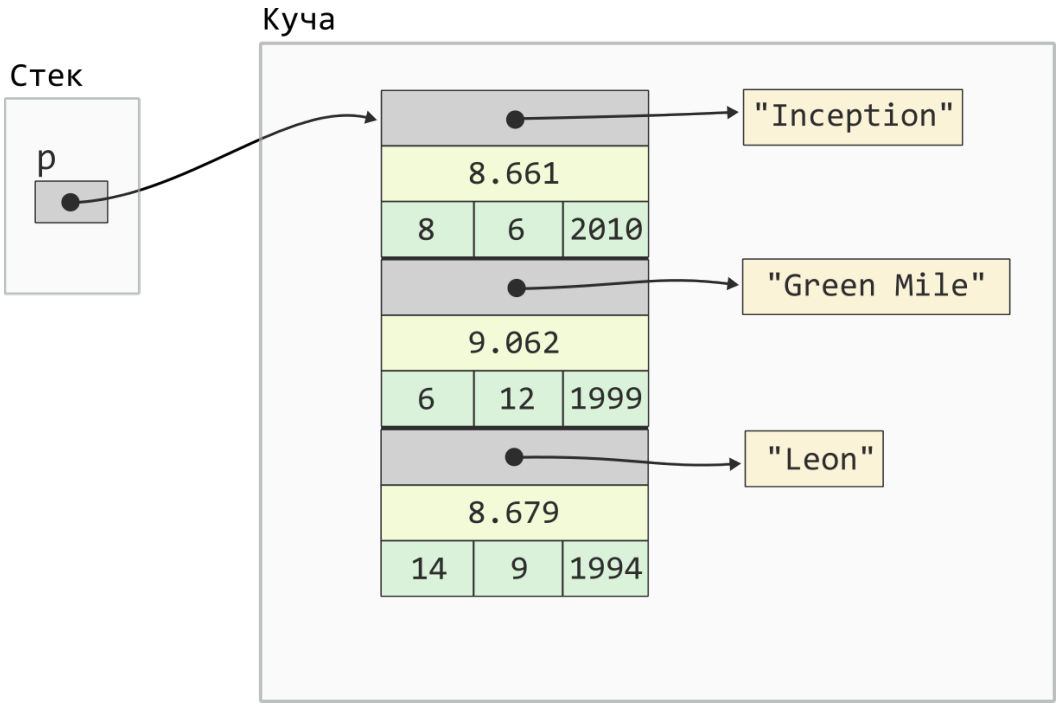
Указатель на массив структур, выделенный в куче

Пример программы, которая динамически выделяет массив структур:




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

void print_movies(const Movie* pm, int number_of_movies)
{
    for (int i = 0; i < number_of_movies; ++i)
    {
        printf("%s. Rating: %.3f. ", pm[i].title, pm[i].rating);
        printf("Release Date: %d/%d/%d\n", pm[i].release_date.day,
               pm[i].release_date.month, pm[i].release_date.year);
    }
}
void set_movie(Movie* pm, char* title, float rating, int day, int month, int year)
{
    strcpy(pm->title, title);
    pm->rating = rating;
    pm->release_date.day = day;
    pm->release_date.month = month;
    pm->release_date.year = year;
}
int main()
{
    Movie* p = (Movie*)malloc(3 * sizeof(Movie));
    set_movie(p, "Inception", 8.661, 8, 6, 2010);
    set_movie(p + 1, "Green Mile", 9.062, 6, 12, 1999);
    set_movie(p + 2, "Leon", 8.679, 14, 9, 1994);
    print_movies(p, 3);
    free(p);
}
```

Задача #19: Одна из проблем в коде выше заключается в том, что для поля `title` всегда выделяется `50` байт на стеке. Название фильма обычно значительно меньше, чем `50` байт, так что много памяти выделяется зря. Более того, если вдруг появится фильм с длиной названия более `50` символов, то такая структура не сможет его обработать. Решение – выделять память под строки динамически, как это показано на рисунке ниже.
Измените код программы так, чтобы он выделял память под строки динамически. Также нужно добавить функцию `void delete_movie_array(Movie* pm)`, которая будет освобождать всю выделенную под массив структур память (и под строки и под сам массив).

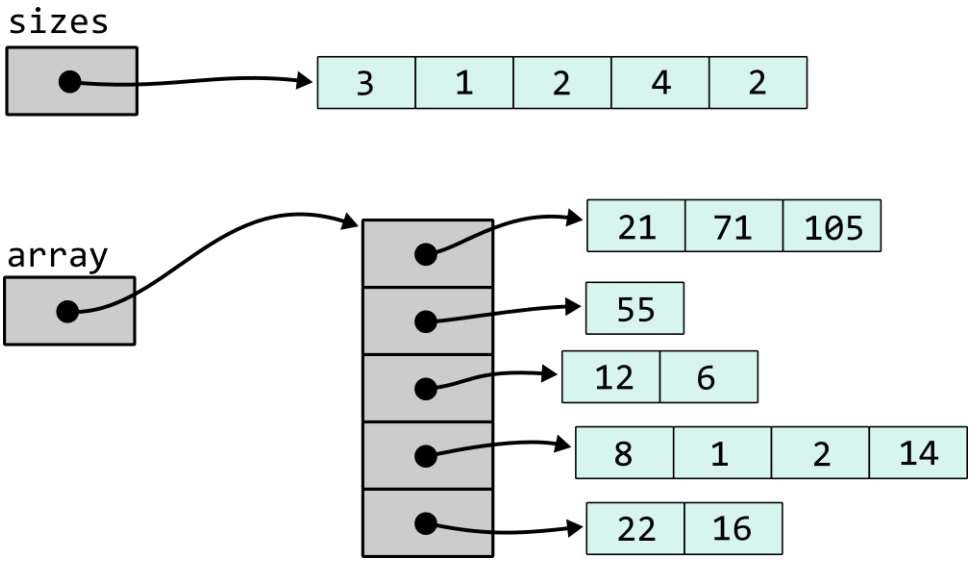


Двумерный динамический массив

В стандартной библиотеке нет специальных средств по созданию двумерных динамических массивов. Есть 2 варианта для создания такого массива:

1. Создать одномерный динамический массив размера `n * m` и работать с ним. Это хороший вариант, когда длины всех строк массива равны или примерно равны и не меняются.

2. Создать динамический массив из указателей, каждый указатель будет соответствовать строке. Затем, для каждой строки динамически выделить столько памяти, сколько нужно. При этом нам нужно будет создать отдельный массив(`sizes`), который будет хранить размеры каждой строки.



Задача #20: Напишите код, который будет выделять память и инициализировать её в соответствии со схемой на рисунке. Также напишите и проверьте функции:
`void print_two_dim_array(int n, int* sizes, int** array)` - печать такого массива и
`void delete_two_dim_array(int n, int* sizes, int** array)` - освобождение памяти.

Динамический массив строк

Динамический массив строк – это двумерный динамический массив элементов типа `char`. Но с одной особенностью: размер строки задаётся не числом в отдельной переменной, а специальным нулевым символом на конце строки. Поэтому от массива `sizes` из прошлой задачи можно отказаться.

Задача #21: Пусть есть файл `words.txt` с примерно следующим содержанием:

```
5
Hello
OK
Cat
Antidisestablishmentarianism
Programming
```

- Написать функцию `void read_words(char* filename, int* p_number_of_words, char*** p_words)`, которая будет считывать из файла такого формата все слова, выделять в куче память под эти слова и сохранять слова в этой памяти. Вызов этой функции должен происходить таким образом:

  ```
  int number_of_words;
  char** words;
  read_words("words.txt", &number_of_words, &words);
  ```

  Считайте, что каждое слово не превышает `10000` символов. Также учтите, что при выделении памяти на строку нужно не забывать нулевой символ на конце строки (функция `strlen` возвращает длину строки без учёта этого символа).

- Написать функцию `void write_words(FILE* stream, int number_of_words, char** words)`, которая будет печатать все слова.

- Написать функцию `void sort_words(int number_of_words, char** words)`, которая будет сортировать все слова по адфавиту.

- Считайте все слова, отсортируйте их и запишите всё в файл `sorted_words.txt`. (не забудьте освободить всю память в конце).