

# Семинар #8: Сегменты памяти. Домашнее задание.

## Выделение памяти в Куче

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

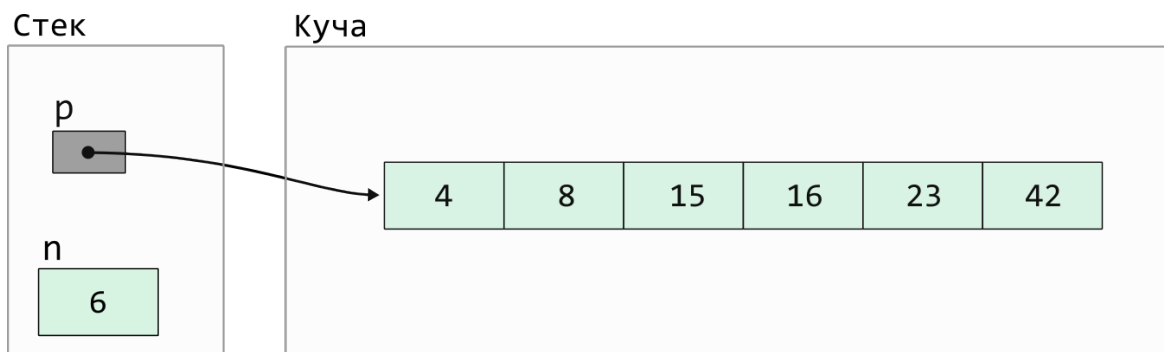
- `void* malloc(size_t n)` – выделяет `n` байт и возвращает указатель `void*` на начало этой памяти. Если память выделить не получилось (например памяти не хватает), то функция вернёт значение `NULL`. (`NULL` – это просто константа равная нулю).
- `void free(void* p)` – освобождает выделенную память.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

```
#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)

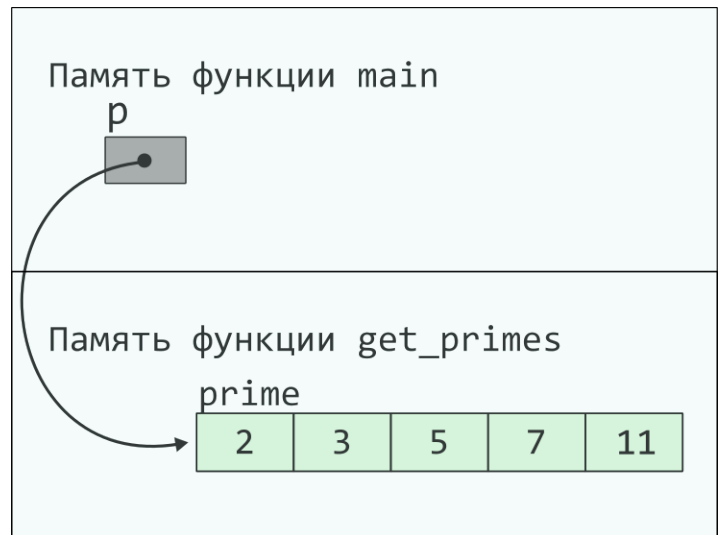
Допустим мы хотим создать функцию, которая должна будет возвращать массив. Известно, что при передаче массива в функцию он всегда передаётся по указателю. Можно попытаться создать статический массив внутри функции и вернуть указатель на него как это сделано в примере ниже. Однако это является грубой 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!

```
#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 раскомментировать строку?

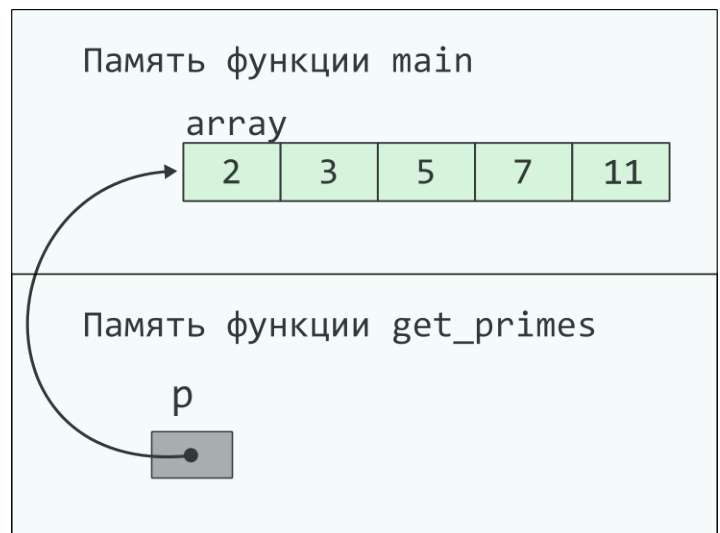
## Возврат массива из функции (передавая указатель как аргумент)

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.

```
#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]);
}
```



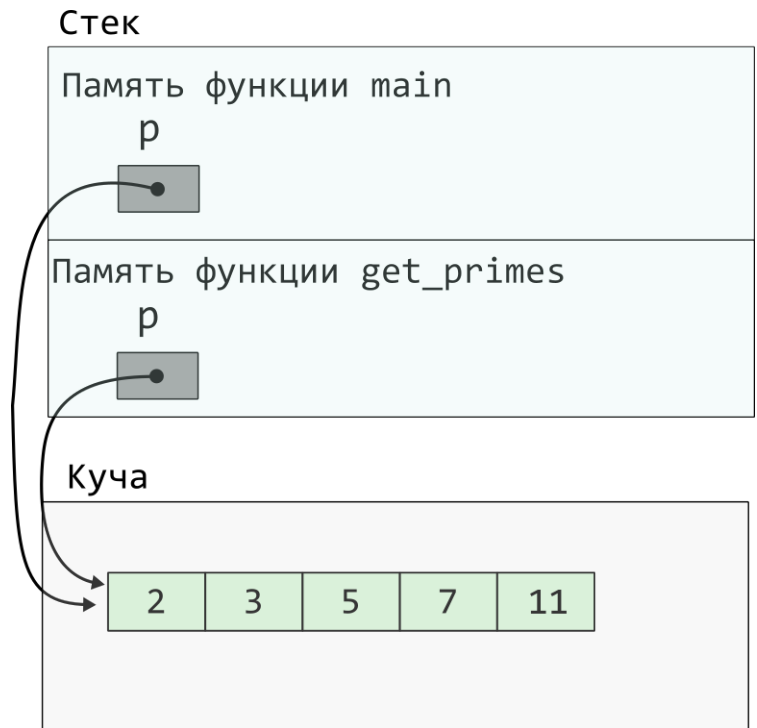
## Возврат массива из функции (через динамическую память)

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.

```
#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:  $a, ar, ar^2, \dots$ . Память должна выделяться динамически. Вызовите эту функцию из `main` и напечатайте первые 10 степеней тройки.

## Указатель на массив структур, выделенный в куче

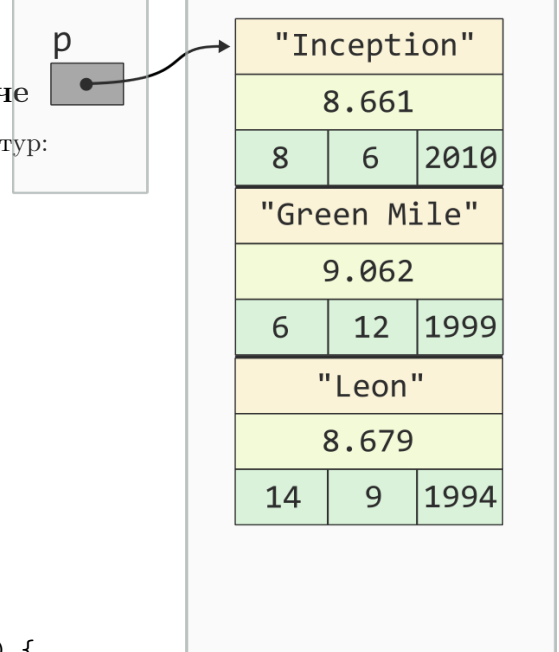
Пример программы, которая динамически выделяет массив структур:

```
#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;

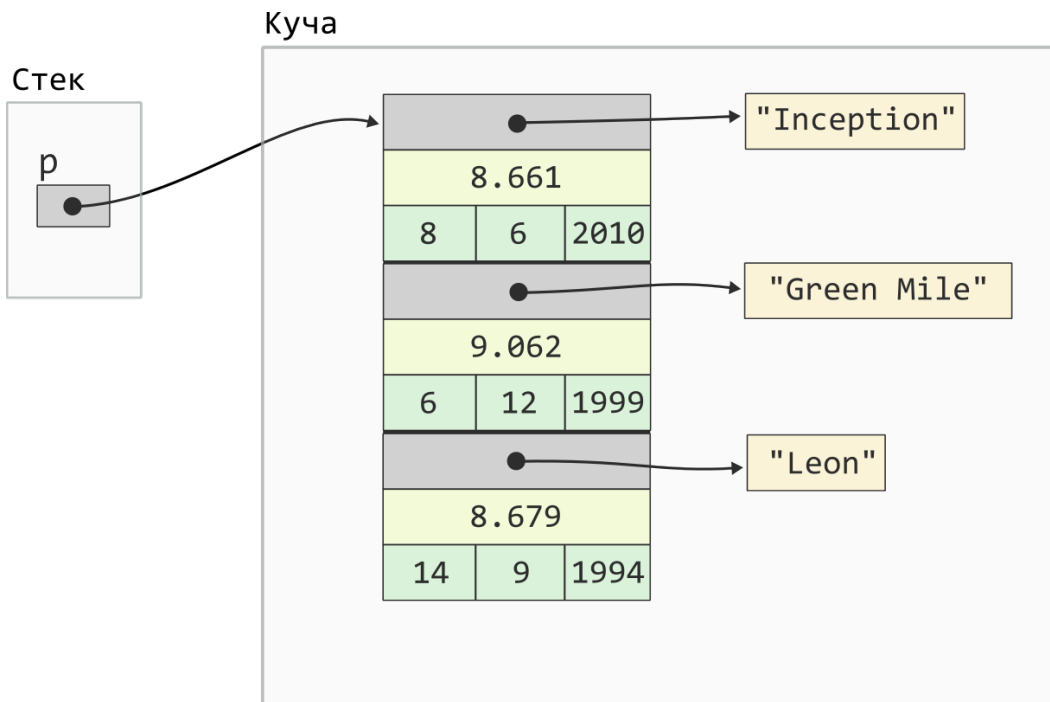
void print_movies(const Movie* pm, int number_of_movies) {
    for (int i = 0; i < number_of_movies; ++i) {
        printf("%s. Rating: %.3f. ", pm[i].title, pm[i].rating);
        printf("Release Date: %d/%d/%d\n", pm[i].release_date.day,
            pm[i].release_date.month, pm[i].release_date.year);
    }
}

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);
}
```



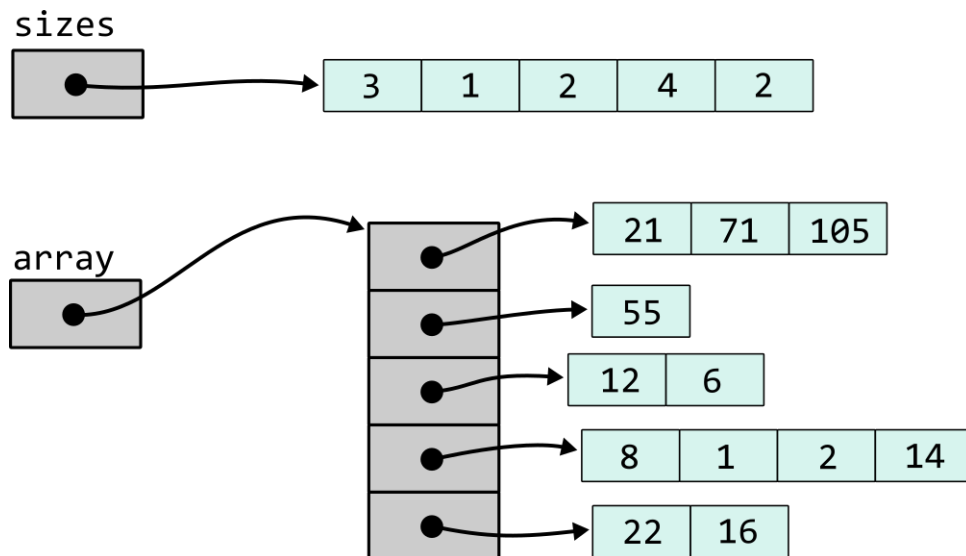
**Задача #19:** Одна из проблем в коде выше заключается в том, что для поля `title` всегда выделяется 50 байт на стеке. Название фильма обычно значительно меньше, чем 50 байт, так что много памяти выделяется зря. Более того, если вдруг появится фильм с длиной названия более 50 символов, то такая структура 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 алфавиту.
- Считайте все слова, отсортируйте их и запишите всё в файл `sorted_words.txt`. (не забудьте освободить всю память в конце).