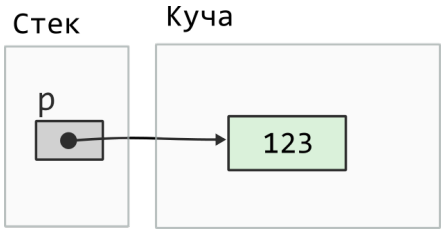
Семинар #5: Сегменты памяти. Домашнее задание.

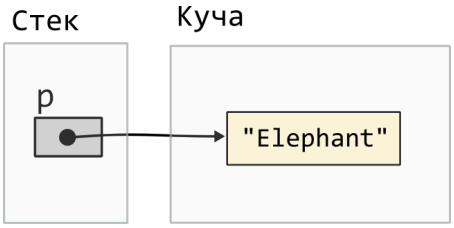
Задача 1. Создание объектов в куче

Напишите код, который будет создавать в куче объекты, соответствующие следующим рисункам. В каждой задаче напечатайте созданные в куче объекты. В каждой задаче освободите всю память, которую вы выделили.

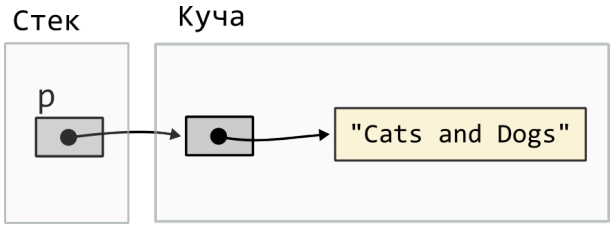
1. Одно число типа `size_t`.



2. Строка (массив из элементов типа `char`).

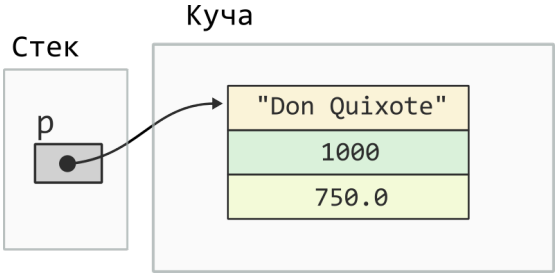


3. Указатель, указывающий на строку.

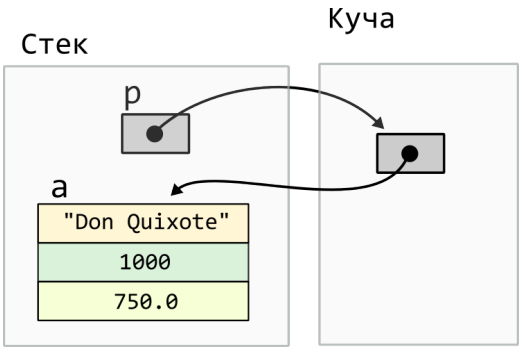


4. Структура типа `Book` из семинара про структуры. Для печати такой структуры можете использовать функцию `print_book` из семинара про структуры.

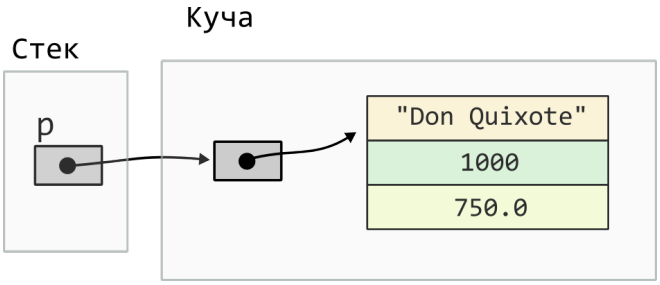
```c
struct book
{
    char title[50];
    int pages;
    float price;
};
typedef struct book Book;
```



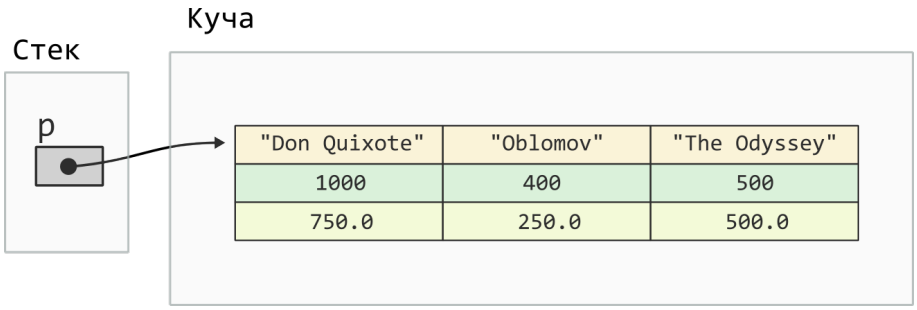
5. Указатель, указывающий на структуру на стеке.



6. Указатель, указывающий на структуру в куче.

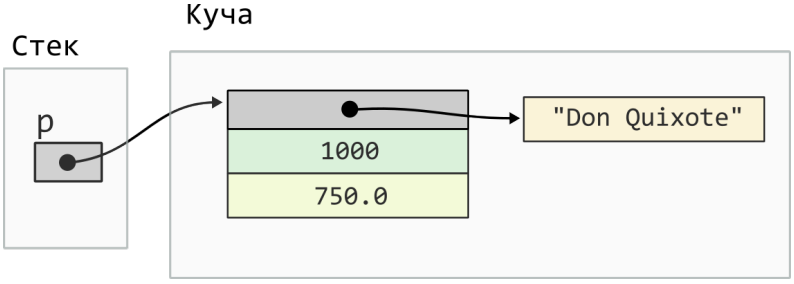


7. Массив структур.

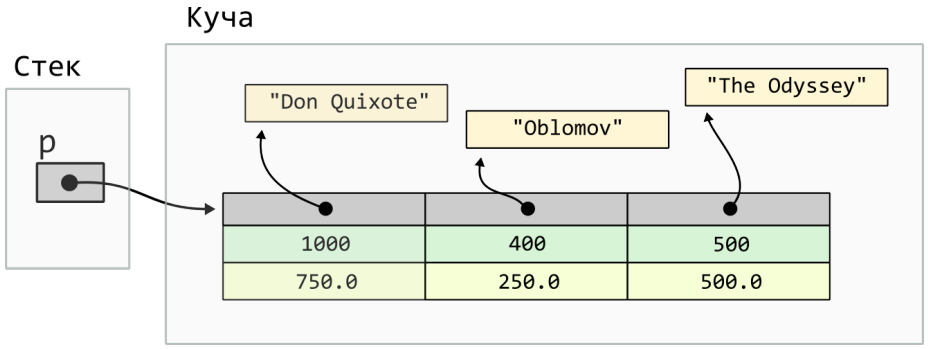


8. Видоизмените структуру `Book`, чтобы она, вместо строки, хранила указатель на строку в куче. Таким образом у нас не будет ограничения на длину названия. Создайте такую структуру в куче. Функцию `print_book` для такой структуры потребуется немного изменить.

```
struct book
{
    char* title;
    int pages;
    float price;
};
typedef struct book Book;
```

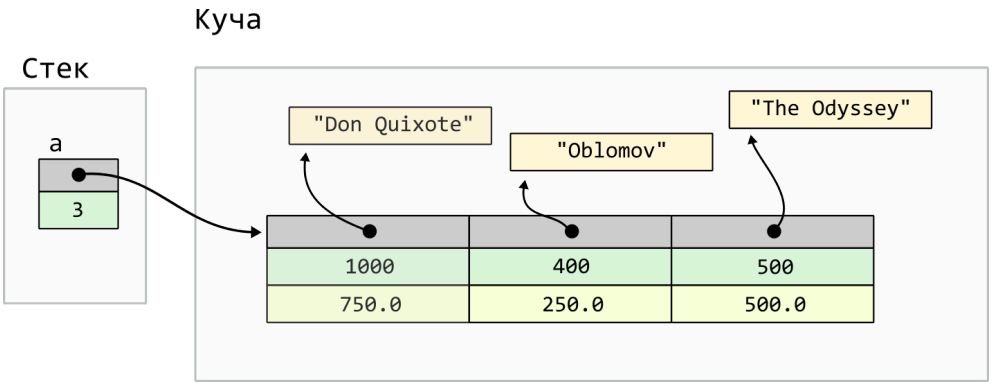


9. Массив таких структур в куче.



10. Создадим структуру `Library`, которая сможет хранить информацию о произвольном количестве книг.

```
struct library
{
    Book* books;
    int number_of_books;
};
typedef struct library Library;
```



Напишите функции для удобной работы с такой структурой:

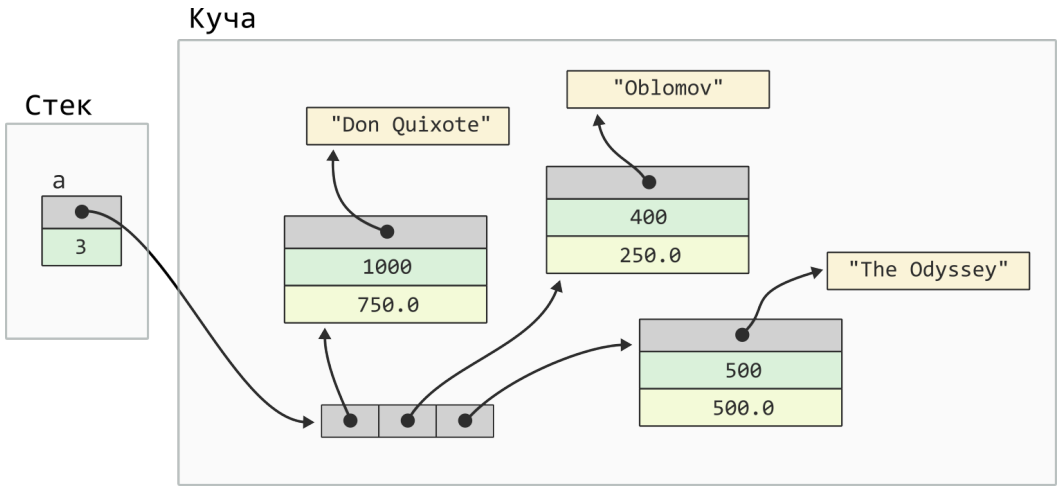
- `library_create` - функция должна задавать поля `books` и `number_of_books` структуры `Library`. При этом данная функция должна выделять необходимое количество памяти.
- `library_set` - должна задавать значение `i`-й книги.
- `library_get` - принимает на вход индекс `i` и возвращает указатель на `i`-ю книгу.
- `library_print` - должна печатать все книги библиотеки на экран.
- `library_destroy` - должна освобождать всю память и устанавливать значения полей в `NULL` и `0` соответственно.

```
Library a;
library_create(&a, 3);
library_set(a, 0, "Don Quixote", 1000, 750.0);
library_set(a, 1, "Oblomov", 400, 250.0);
library_set(a, 2, "The Odyssey", 500, 500.0);
library_print(a);
print_book(library_get(a, 1));
library_destroy(&a);
```

Обратите внимание, что функции `library_create` и `library_destroy` должны принимать указатель на структуру `Library` так как они должны менять поля этой структуры.

11. Изменим структуру `Library` так, чтобы она хранила указатель на массив из указателей на структуры.

```
struct library
{
    Book** books;
    int number_of_books;
};
typedef struct library Library;
```



Напишите функции для удобной работы с такой структурой:

- `library_create` - функция должна задавать поля `books` и `number_of_books` структуры `Library`. При этом данная функция должна выделять память под массив из указателей и задавать все значения элементов массива значением `NULL`.
- `library_set` - должна задавать значение `i`-й книги.
- `library_get` - принимает на вход индекс `i` и возвращает указатель на `i`-ю книгу. Если структура с таким индексом ещё не создана, то функция должна вернуть `NULL`.
- `library_print` - должна печатать все книги библиотеки на экран.
- `library_destroy` - должна освобождать всю память (память под структуры и память под массив указателей) и устанавливать значения полей в `NULL` и `0` соответственно.

Работа с такой структурой не должна отличаться от работы со структурой из предыдущей подзадачи:

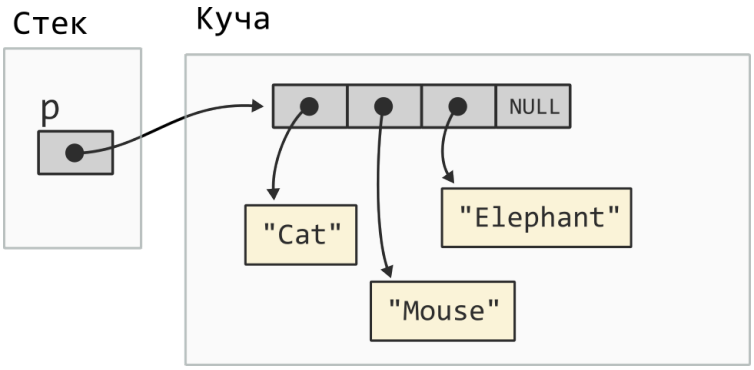
```
Library a;
library_create(&a, 3);
library_set(a, 0, "Don Quixote", 1000, 750.0);
library_set(a, 1, "Oblomov", 400, 250.0);
library_set(a, 2, "The Odyssey", 500, 500.0);
library_print(a);
print_book(library_get(a, 1));
library_destroy(&a);
```

Задача 2. Геометрическая прогрессия

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

Задача 3. 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 мы можем не хранить размер каждой строки.
Мы можем хранить количество строк в таком массиве в отдельной переменной, а можем просто выделить в массиве указателей на один элемент больше и хранить в этом элементе значение `NULL`. Таким образом мы можем найти количество строк в массиве.



1. Написать функцию `char** get_test_strings()` который будет создавать массив строк, представленный на рисунке выше, и возвращать его.
2. Написать функцию `void print_strings(FILE* stream, char** string_array)` который будет печатать переданный массив строк `string_array` в поток `stream`. С помощью этой функции напечатайте массив строк, представленный на рисунке, на экран.
3. Написать функцию `char** load_lines(const char* filename)`, которая будет создавать динамический массив строк, считывать из файла все строки в этот массив строк, и возвращать его. Один из способов как это можно сделать:
   - Пройдите по файлу с помощью `fgetc` и посчитайте количество строк в файле (количество `\n` + 1).
   - Выделите необходимую память в куче под массив указателей (учтите нулевой указатель в конце).
   - Вернитесь в начало файла с помощью `fseek`.
   - Пройдите файл заново и считайте количество символов в каждой строке. Эту информацию нужно где-то хранить, поэтому выделите в куче временный массив в котором будет храниться длина каждой строки.
   - Выделите необходимую память для каждой строки (учтите нулевой символ в конце строк).
   - Вернитесь в начало файла с помощью `fseek`.
   - Пройдите файл заново и считайте каждую строку из файла в соответствующее место в массиве строк.
   - Освободите память под временный массив, закройте файл и верните из функции динамический массив строк.
4. Написать функцию `void destroy_strings(char*** p_string_array)`, которая будет уничтожать динамический массив строк. Эта функция должна освобождать всю память (память под каждую строку и память под массив указателей). Также эта функция должна присваивать указателю `p` значение `NULL`.

   ```
   char** p = load_lines("three_little_pigs.txt");
   destroy_strings(&p);
   ```

5. Написать функцию `void sort_strings(char** words)`, которая будет сортировать все строки по алфавиту. Используйте функцию `strcmp`.

Задача 4. Сортировщик строк

Напишите программу `line_sorter`, которая будет считывать текстовый файл, сортировать строки этого файла по алфавиту и записывать результат в другой файл. Названия файлов функция должна принимать через аргументы командной строки. Пример использования такой программы:

```
./line_sorter invisible_man.txt result.txt
```

Задача 5. Разные сегменты

Задача 6. Сумматор

Создайте функцию `adder`, которая будет принимать на вход число и возвращать сумму всех чисел, которые приходили на вход этой функции за время выполнения программы.

```c
#include <stdio.h>
// Тут нужно написать функцию adder

int main()
{
    printf("%i\n", adder(10));  // Напечатает 10
    printf("%i\n", adder(15));  // Напечатает 25
    printf("%i\n", adder(70));  // Напечатает 95
}
```