ЗАДАНИЕ №3.2

1.

```
#include <stdio.h>

#define N 2
#define M 3
#define K 4

int main(void)
{
    int arr[N][M][K] = { };

    for (size_t i = 0; i < N; ++i)
        for (size_t j = 0; j < M; ++j)
            for (size_t k = 0; k < K; ++k)
                arr[i][j][k] = i * 100 + j * 10 + k;

    return 0;
}
```

Объявим трехмерный массив размеров 2, 3, 4. Для наглядности вывода дампа памяти проинициализируем элементы массива какими-то значениями

2.

С помощью gdb выведем дамп памяти:

```
(gdb) x/24w arr
0x7fffffffdf50:     0       1       2       3
0x7fffffffdf60:     10      11      12      13
0x7fffffffdf70:     20      21      22      23
0x7fffffffdf80:     100     101     102     103
```

```
0x7fffffffdf90:  110  111  112  113
0x7fffffffdfa0:  120  121  122  123
```

По этому выводу можно заметить, что в одном сегменте памяти, который состоит из 16 байт хранится 4 элемента массива (размер элемента равен 4 байта). Соответственно, массив лежит в памяти последовательно, без дырок.

3.

Вывод массива целиком

```
(gdb) x/24 arr
0x7fffffffdf50:  0  1  2  3
0x7fffffffdf60:  10  11  12  13
0x7fffffffdf70:  20  21  22  23
0x7fffffffdf80:  100  101  102  103
0x7fffffffdf90:  110  111  112  113
0x7fffffffdfa0:  120  121  122  123
```

Выводится соответственно 6 сегментов памяти (24 элемента массива)

Фиксируем N

```
(gdb) x/12 arr[0]
0x7fffffffdf50:  0  1  2  3
0x7fffffffdf60:  10  11  12  13
0x7fffffffdf70:  20  21  22  23
(gdb) x/12 arr[1]
0x7fffffffdf80:  100  101  102  103
0x7fffffffdf90:  110  111  112  113
0x7fffffffdfa0:  120  121  122  123
```

Видим, что при фиксированном N выводятся 3 строчки дампа памяти (12 элементов массива), соответствующие arr[0] и arr[1] соответственно

Фиксируем N и M

```
(gdb) x/4 arr[0][0]
0x7fffffffdf50:	0	1	2	3
(gdb) x/4 arr[0][1]
0x7fffffffdf60:	10	11	12	13
(gdb) x/4 arr[0][2]
0x7fffffffdf70:	20	21	22	23
(gdb) x/4 arr[1][0]
0x7fffffffdf80:	100	101	102	103
(gdb) x/4 arr[1][1]
0x7fffffffdf90:	110	111	112	113
(gdb) x/4 arr[1][2]
0x7fffffffdfa0:	120	121	122	123
```

Видим, что при фиксированных N и M выводится 1 строчка дампа памяти, состоящая из 4 элементов, которые являются соответствующими элементами массива

Вывод:

Трехмерный массив записывается в память следующим образом: arr[0][0][0], arr[0][0][1] … arr[0][0][K], arr[0][1][0] … arr[0][1][K], arr[0][2][0] … arr[0][M][K], arr[1][0][0] … arr[N][M][K].

Соответственно рекурентная формула записи массива размерности R в память следующая:

Если R > 1, то последовательно обращаемся к элементам массива (то есть массивам размерности R - 1), иначе просто записываем одномерный массив в память.

4.

```
int (*p0)[3][4] = arr;  // массив из 2 элементов типа int [3][4]
int (*p1)[4] = arr[0];  // массив из 3 элементов типа int [4]
```

```
int *p2 = arr[0][0];  // массив из 4 элементов типа int
int s = arr[0][0][0]; // элемент массива типа int
```

Размер массива = количество элементов массива * тип элементов массива

Соответственно:

Размер arr = 2 * 3 * 4 * sizeof(int) = 2 * 3 * 4 * 4 = 96

Размер arr[0] = 3 * 4 * sizeof(int) = 3 * 4 * 4 = 48

Размер arr[0][0] = 4 * sizeof(int) = 4 * 4 = 16

Размер arr[0][0][0] = sizeof(int) = 4

Проверим с помощью следующего кода:

```
printf("%ld %ld %ld %ld", sizeof(arr), sizeof(arr[0]), sizeof(arr[0][0]), sizeof(arr[0][0][0]));
```

Output:

```
96 48 16 4
```

Соответствующая проверка через gdb выглядит следующим образом:

```
(gdb) p sizeof(arr)
$5 = 96
(gdb) p sizeof(arr[0])
$6 = 48
(gdb) p sizeof(arr[0][0])
$7 = 16
(gdb) p sizeof(arr[0][0][0])
$8 = 4
(gdb)
```

5.

void check_mas3d(int arr[][M][K], size_t n, size_t m, size_t k); // прототип функции, которая обрабатывает трехмерный массив

void check_mas2d(int arr[][K], size_t m, size_t k); // прототип функции, которая обрабатывает матрицу трехмерного массива

void check_mas1d(int arr[], size_t k); // прототип функции, которая обрабатывает одномерный массив матрицы трехмерного массива

void check_elem(int *el); // прототип функции, которая обрабатывает элемент трехмерного массива (передаем по указателю на случай, если захотим поменять его значение)