

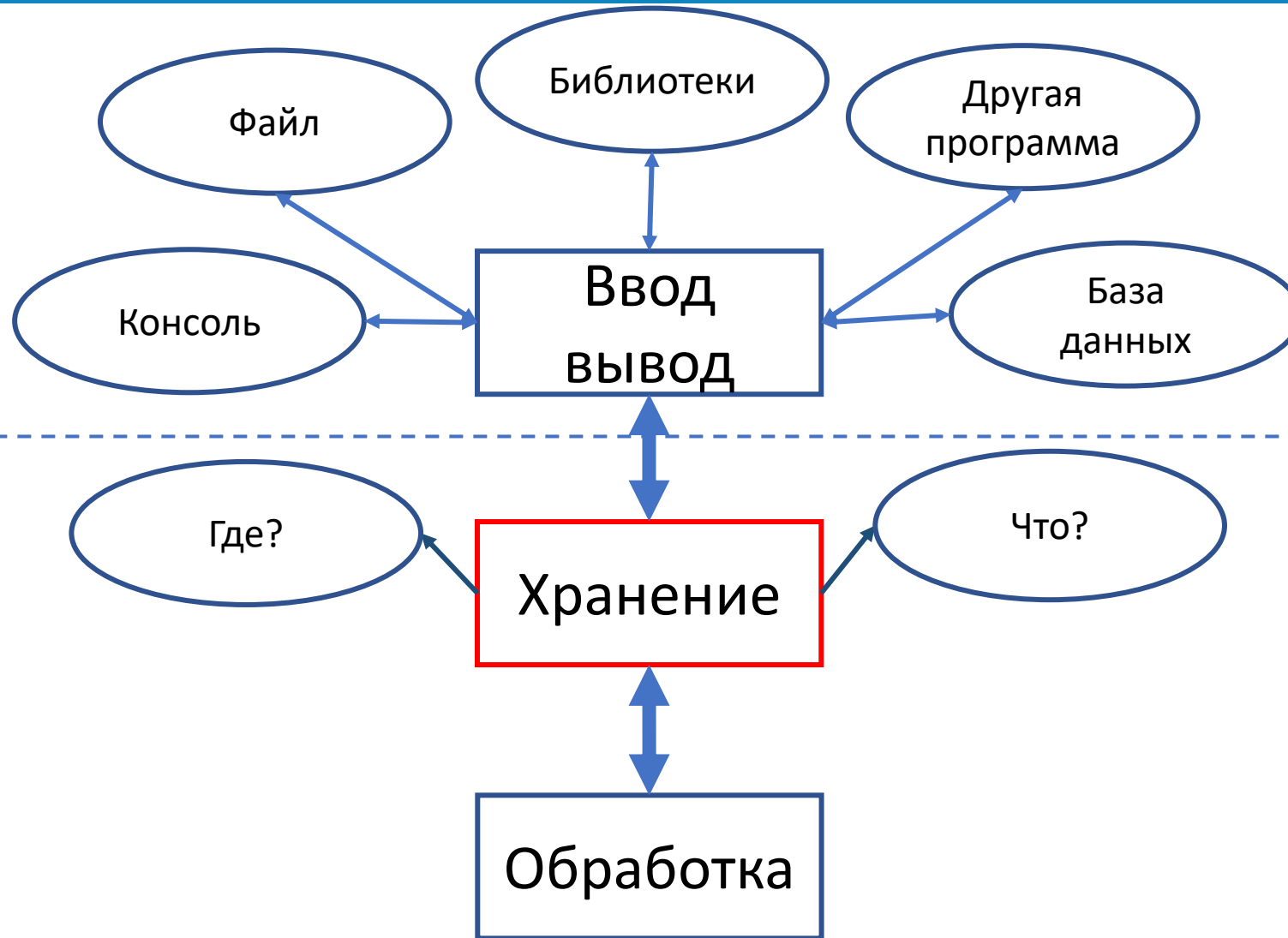


# Основы программирования на C++

## Занятие 6. Динамическое выделение памяти



# Дерево языка

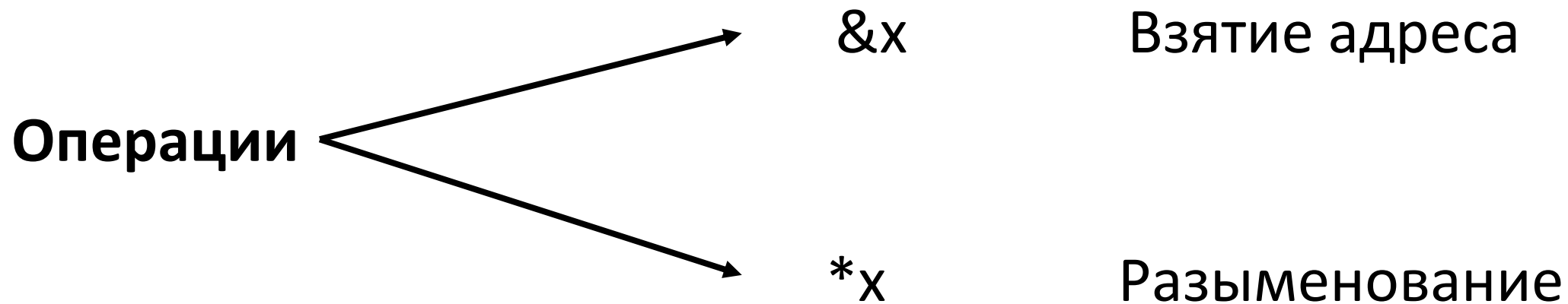




## В предыдущей лекции...

**Указатель** - это адрес переменной в памяти.

`int*`  храним адрес 'int'





## В предыдущей лекции...

```
#include <stdio.h>
int main()
{
    int x = 10;
    int* y = &x;
    printf("x = %d\n", x);
    *y = *y + 1;
    printf("x = %d\n", x);
    return 0;
}
```

Результат

```
x = 10
x = 11
```



## В предыдущей лекции...

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

```
void swap(int *a, int *b)
{
    int tmp = *b;
    *b = *a;
    *a = tmp;
}
```

```
int main()
{
    int a = 10, b = 15;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Результат

```
a = 10, b = 15
a = 15, b = 10
```



# Предисловие

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

const int N = 10;

void input_array(double *x)
{
    int arrMax = 100, arrMin = 0;
    for (int i = 0; i < N; i++)
        x[i] = arrMin + (arrMax - arrMin) * ((double) rand() / RAND_MAX);
}

void print_array(double* x)
{
    for (int i = 0; i < N; i++)
        printf("%lf ", x[i]);
    printf("\n ");
}

int main()
{
    srand(time(NULL));
    double array[N];
    input_array(array);
    print_array(array);
    return 0;
}
```

Существует ли ограничение  
на значение N?



# Устройство памяти процесса

**Процесс** – программа  
во время исполнения  
и выделенные ей  
ресурсы







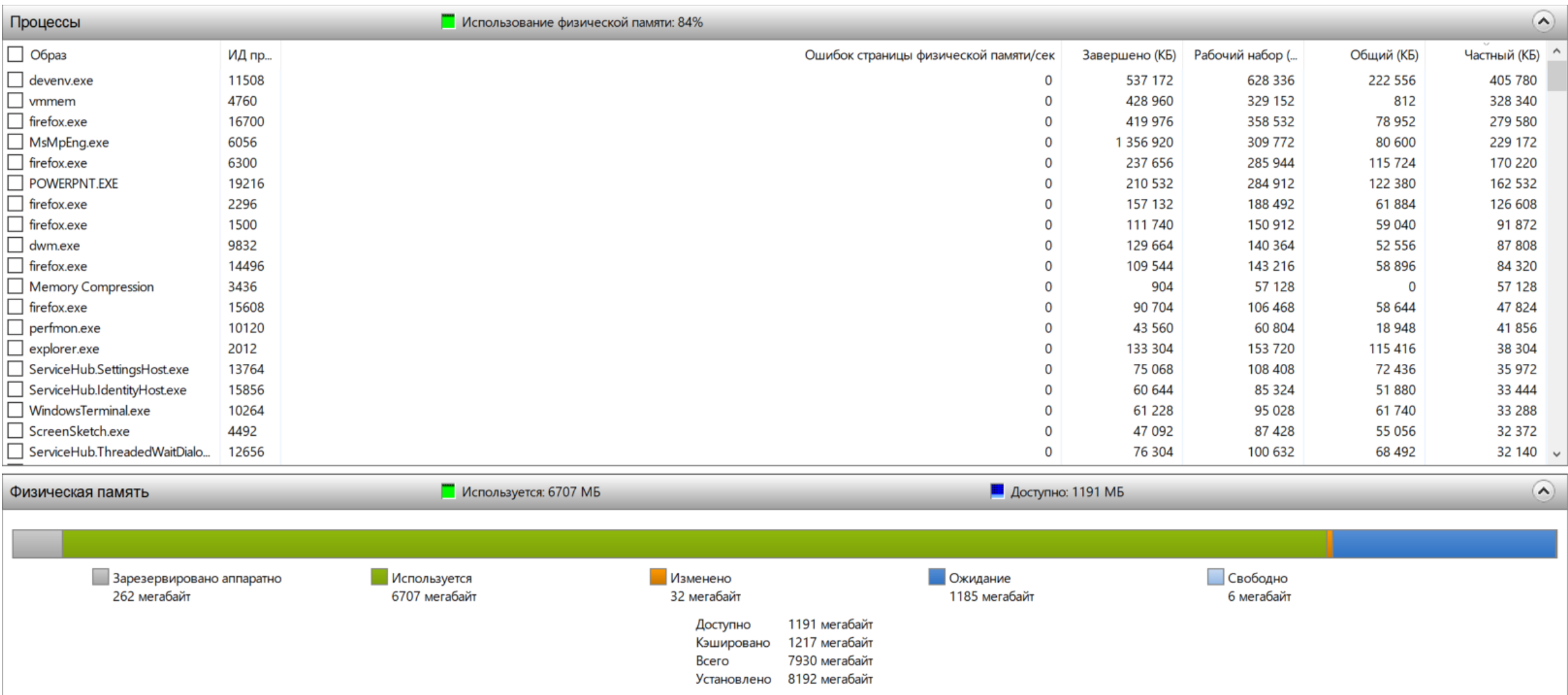
# Устройство памяти процесса





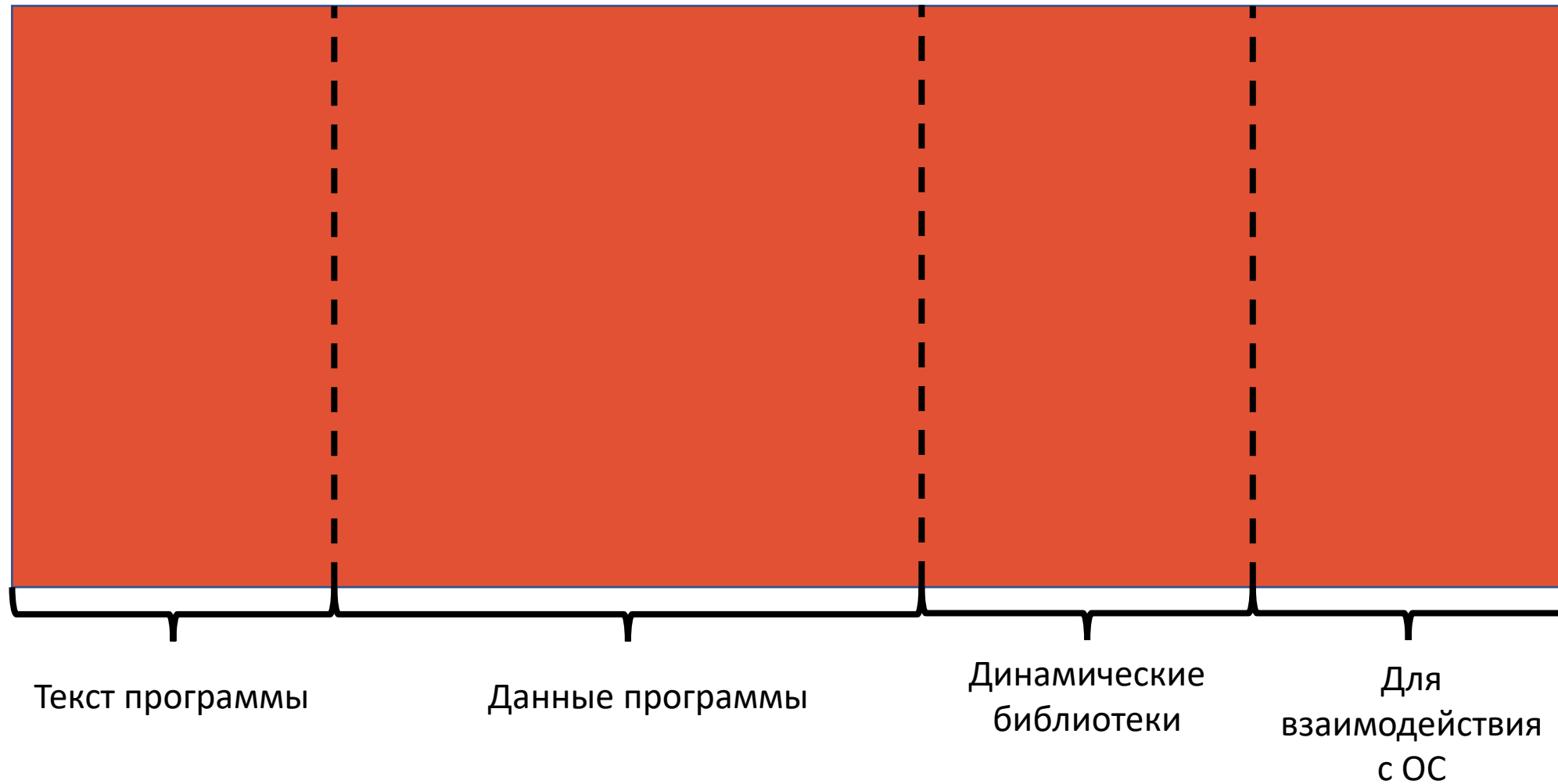


# Монитор ресурсов



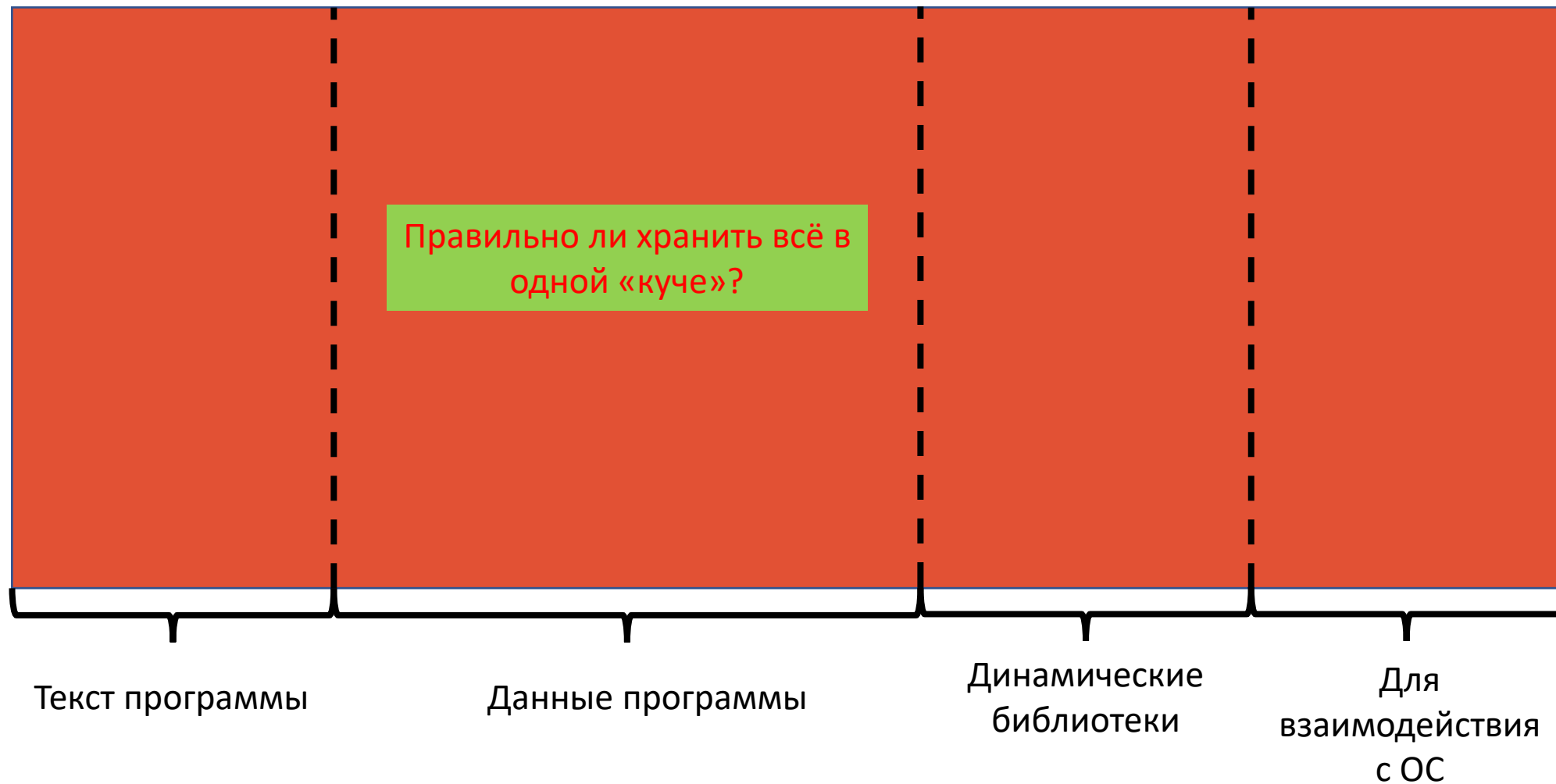


# Устройство памяти процесса



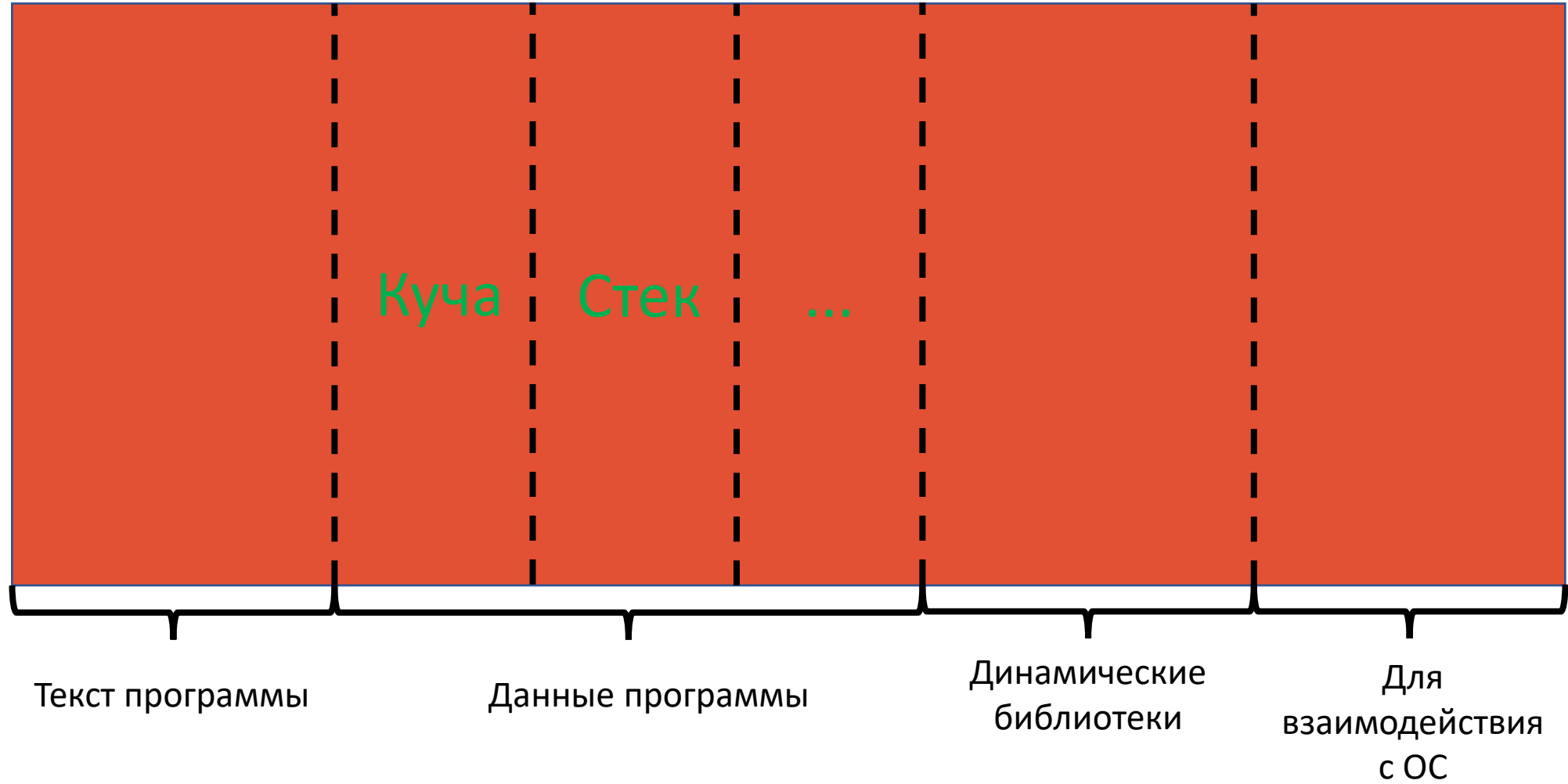


# Устройство памяти процесса





# Устройство памяти процесса





```
55e233729000-55e23372a000 r--p 00000000 00:2e 10696049115018075 /mnt/c/Users/79629/Documents/ubuntu_files/memory
55e23372a000-55e23372b000 r-xp 00001000 00:2e 10696049115018075 /mnt/c/Users/79629/Documents/ubuntu_files/memory
55e23372b000-55e23372c000 r--p 00002000 00:2e 10696049115018075 /mnt/c/Users/79629/Documents/ubuntu_files/memory
55e23372c000-55e23372d000 r--p 00002000 00:2e 10696049115018075 /mnt/c/Users/79629/Documents/ubuntu_files/memory
55e23372d000-55e23372e000 rw-p 00003000 00:2e 10696049115018075 /mnt/c/Users/79629/Documents/ubuntu_files/memory
55e2350fc000-55e23511d000 rw-p 00000000 00:00 0 [heap]
7f16bfc000-7f16bfd000 r--p 00000000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfd000-7f16bfe000 r-xp 00025000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfe000-7f16bfec000 r--p 0019d000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfec000-7f16bfec9000 ---p 001e7000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfec9000-7f16bfec9000 r--p 001e7000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfec9000-7f16bfecf000 rw-p 001ea000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfecf000-7f16bfed5000 rw-p 00000000 00:00 0
7f16bfed5000-7f16bfede000 r--p 00000000 08:10 11000 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f16bfede000-7f16bff01000 r-xp 00001000 08:10 11000 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f16bff01000-7f16bff09000 r--p 00024000 08:10 11000 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f16bff09000-7f16bff0b000 r--p 0002c000 08:10 11000 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f16bff0b000-7f16bff0c000 rw-p 0002d000 08:10 11000 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f16bff0c000-7f16bff0d000 rw-p 00000000 00:00 0
7ffc901b0000-7ffc901d1000 rw-p 00000000 00:00 0 [stack]
7ffc901e6000-7ffc901ea000 r--p 00000000 00:00 0 [vvar]
7ffc901ea000-7ffc901eb000 r-xp 00000000 00:00 0 [vdso]
```



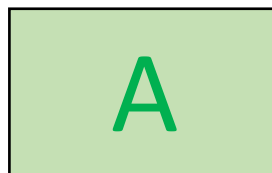
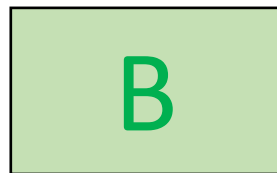
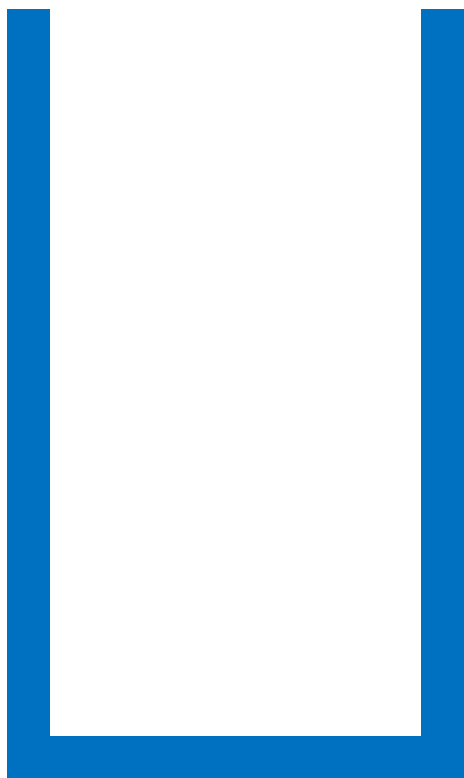
# Стек



**Стек** — это структура данных, которая работает по принципу FILO (first in — last out; первый пришел — последний ушел).



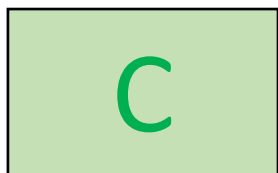
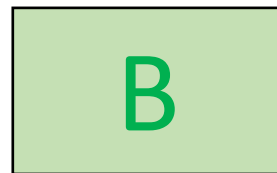
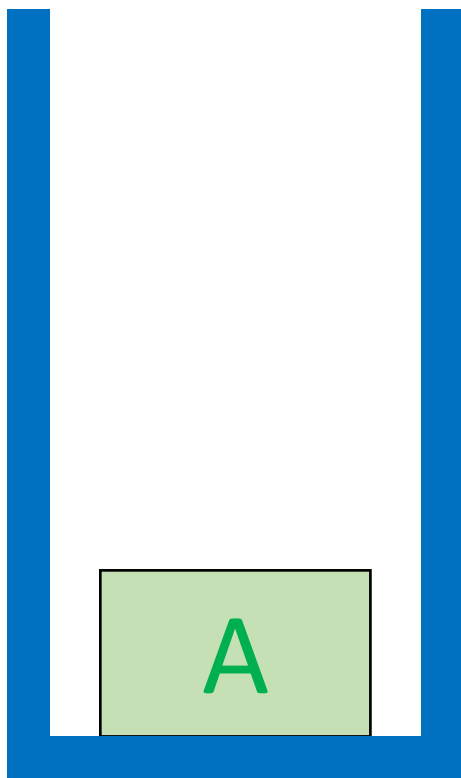
# Стек





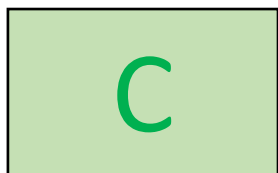
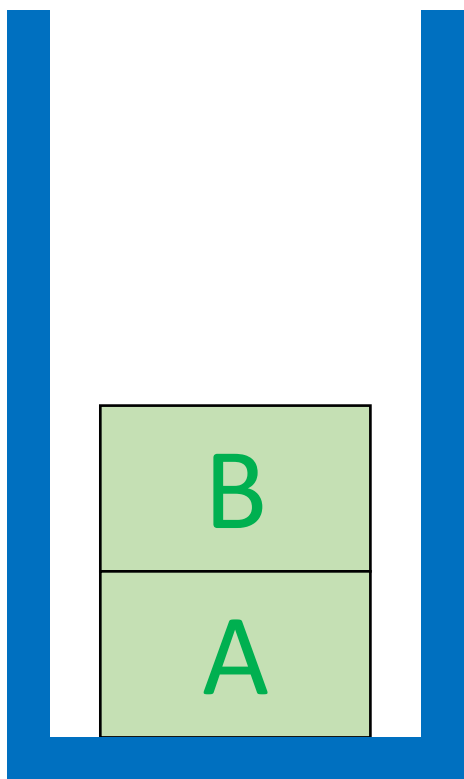


# Стек



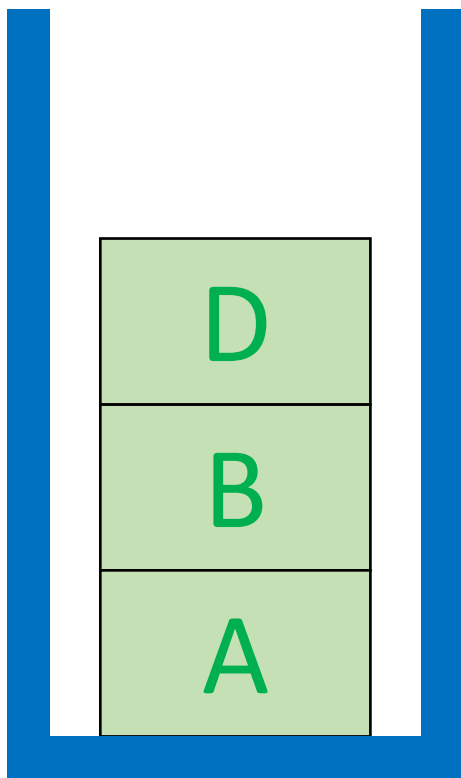


# Стек



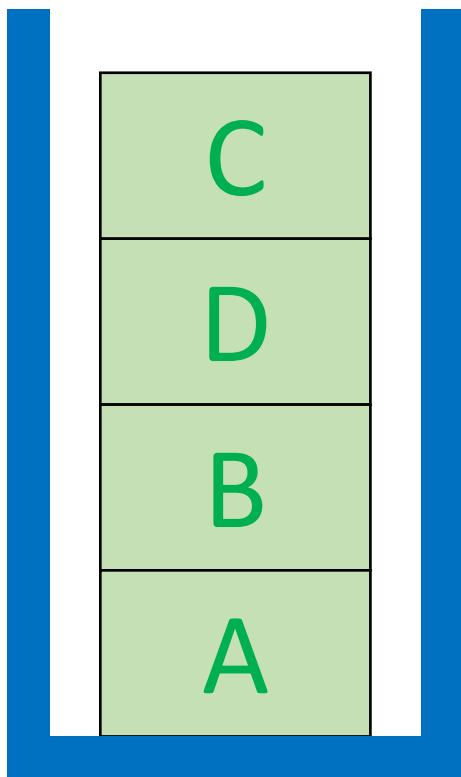


# Стек



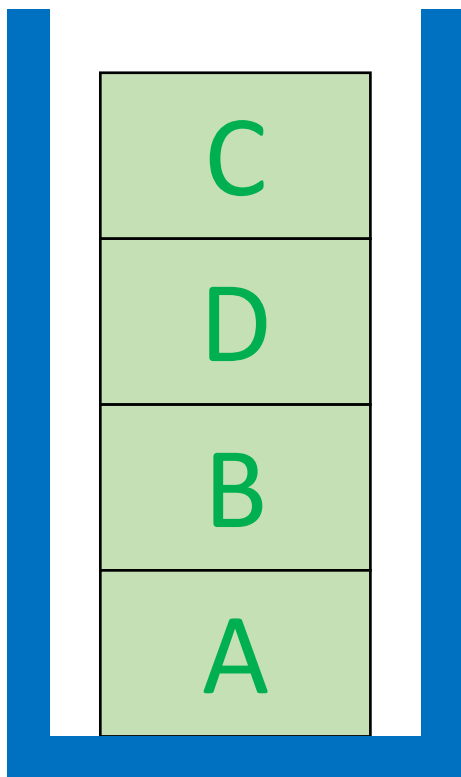


# Стек





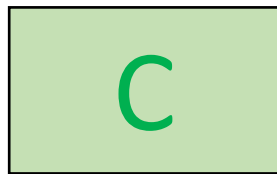
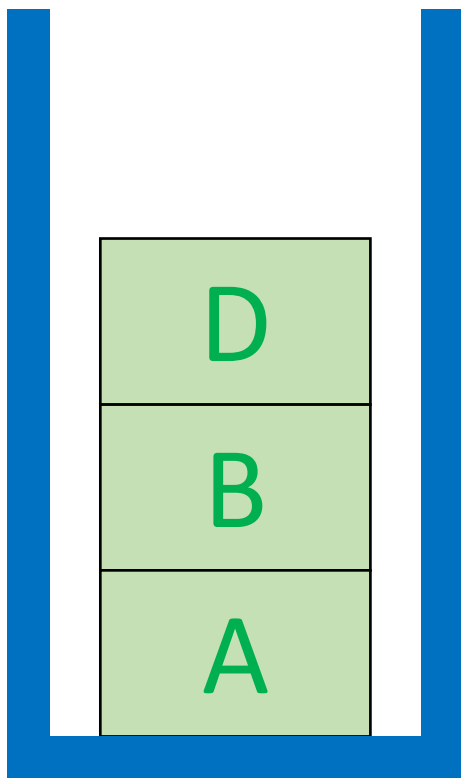
# Стек



Как достать «B»?

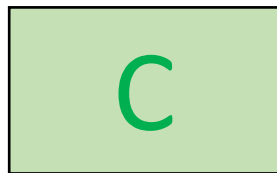
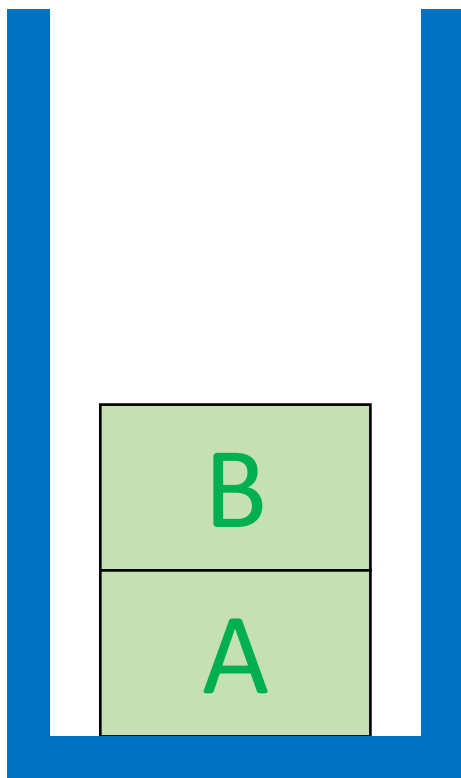


# Стек





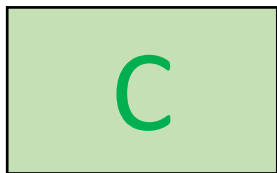
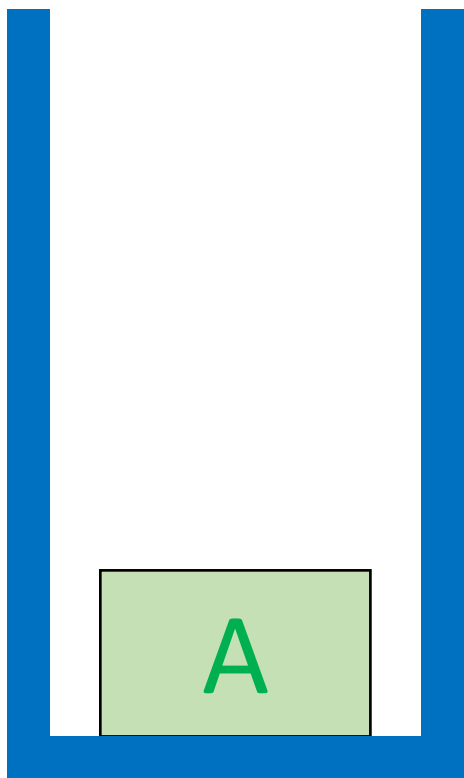
# Стек





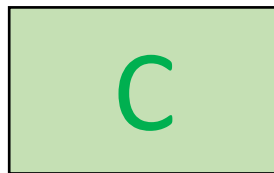
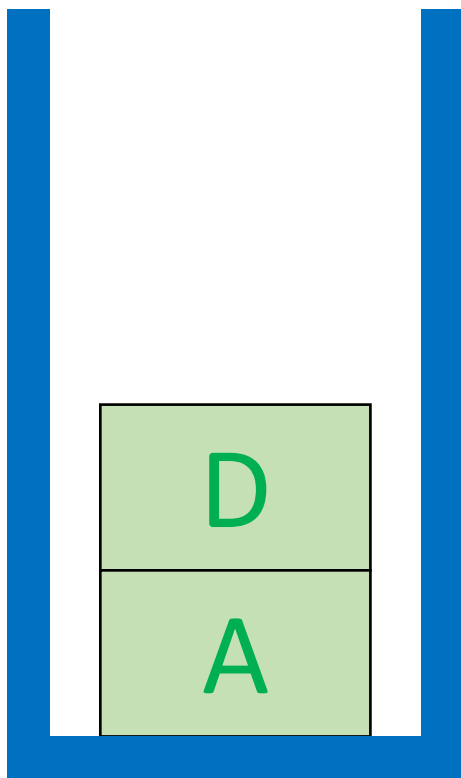


# Стек



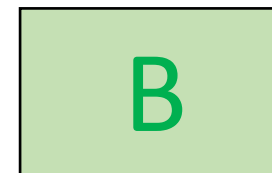
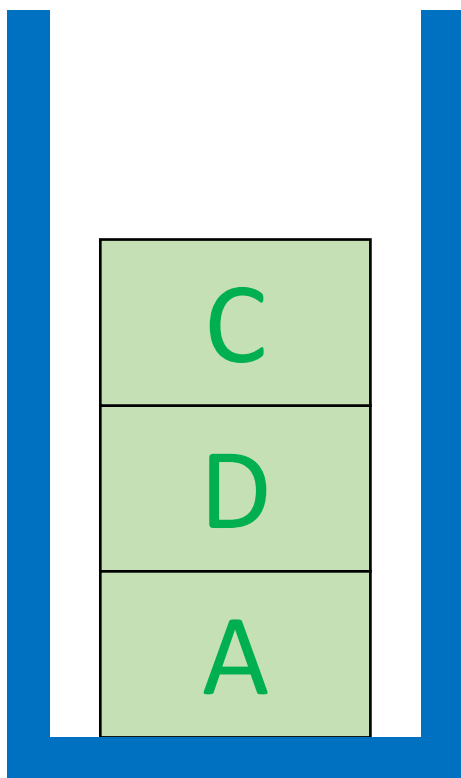


# Стек



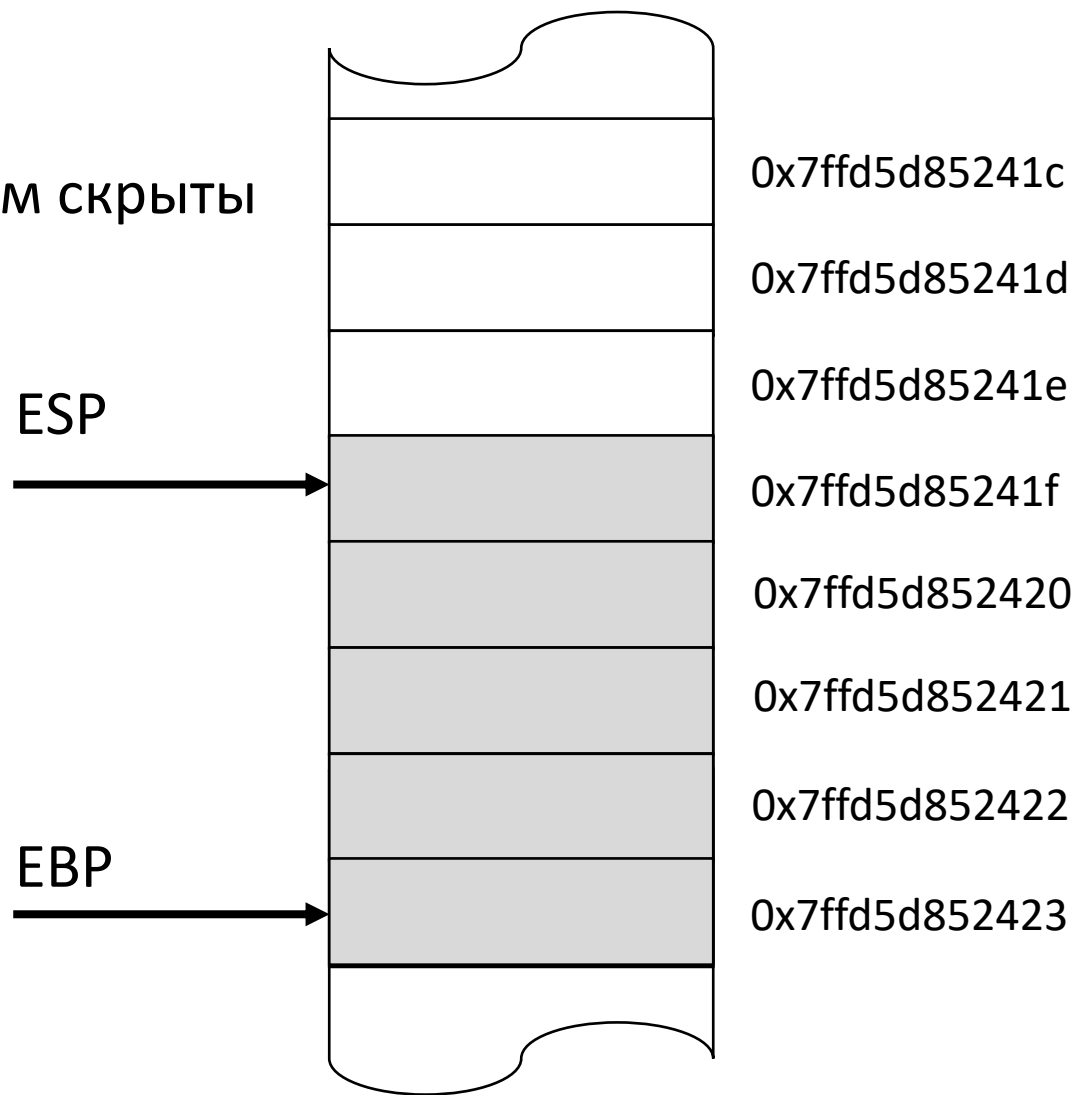


# Стек





EBP – указатель на начало стека





# Куча

**Куча** - название структуры данных, с помощью которой реализована динамически распределяемая память приложения.





## Выделение памяти

### Стек

Память распределяется по методу (LIFO)

Нет необходимости освобождения памяти

Размер: **1 МБ** (по умолчанию Windows)

### Куча

Память распределяется в случайном порядке

Необходимо освобождать память

Размер: **4 ГБ и более**



# Выделение памяти

```
void *malloc(size_t size)
```

Размещение блоков памяти

```
void *realloc(void *memblock, size_t size);
```

Повторное выделение блоков памяти.

```
void *calloc(size_t number, size_t size);
```

Выделяет массив в памяти и инициализирует его элементы значением 0.

```
void free(void *ptr)
```

Освобождает блок памяти.





# Выделение памяти

Тип `void *`

`void *malloc(size t size)` Размер выделенной памяти

`void free(void *ptr)`

Размера памяти Нет



# Выделение памяти

```
int N = 3;
int arrMax = 100, arrMin = 0;

double* arr = (double*)malloc(N * sizeof(double));

for(int i = 0; i < N; ++i)
    arr[i] = arrMin + (arrMax - arrMin) * ((double)rand() / RAND_MAX);

printf("&arr = %p\n", &arr);

for (int i = 0; i < N; ++i)
    printf("&arr[%d] = %p\n", i, &arr[i]);
for (int i = 0; i < N; ++i)
    printf("%lf\n", arr[i]);

free(arr);
```

```
&arr =      00000037405CFA20
&arr[0] =   0000023344359F60
&arr[1] =   0000023344359F68
&arr[2] =   0000023344359F70
0.125126
56.358531
19.330424
```



# Выделение памяти

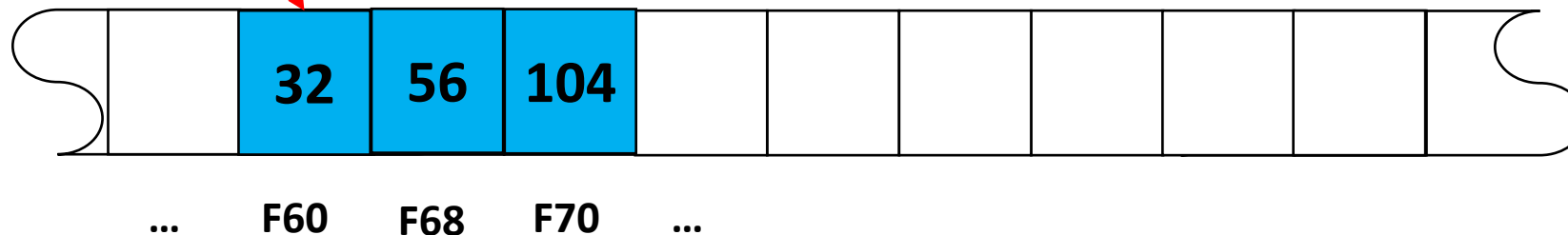
```
double* arr = (double*)malloc(N * sizeof(double));
```

```
&arr =          00000037405CFA20  
&arr[0] =       0000023344359F60  
&arr[1] =       0000023344359F68  
&arr[2] =       0000023344359F70
```

Стек

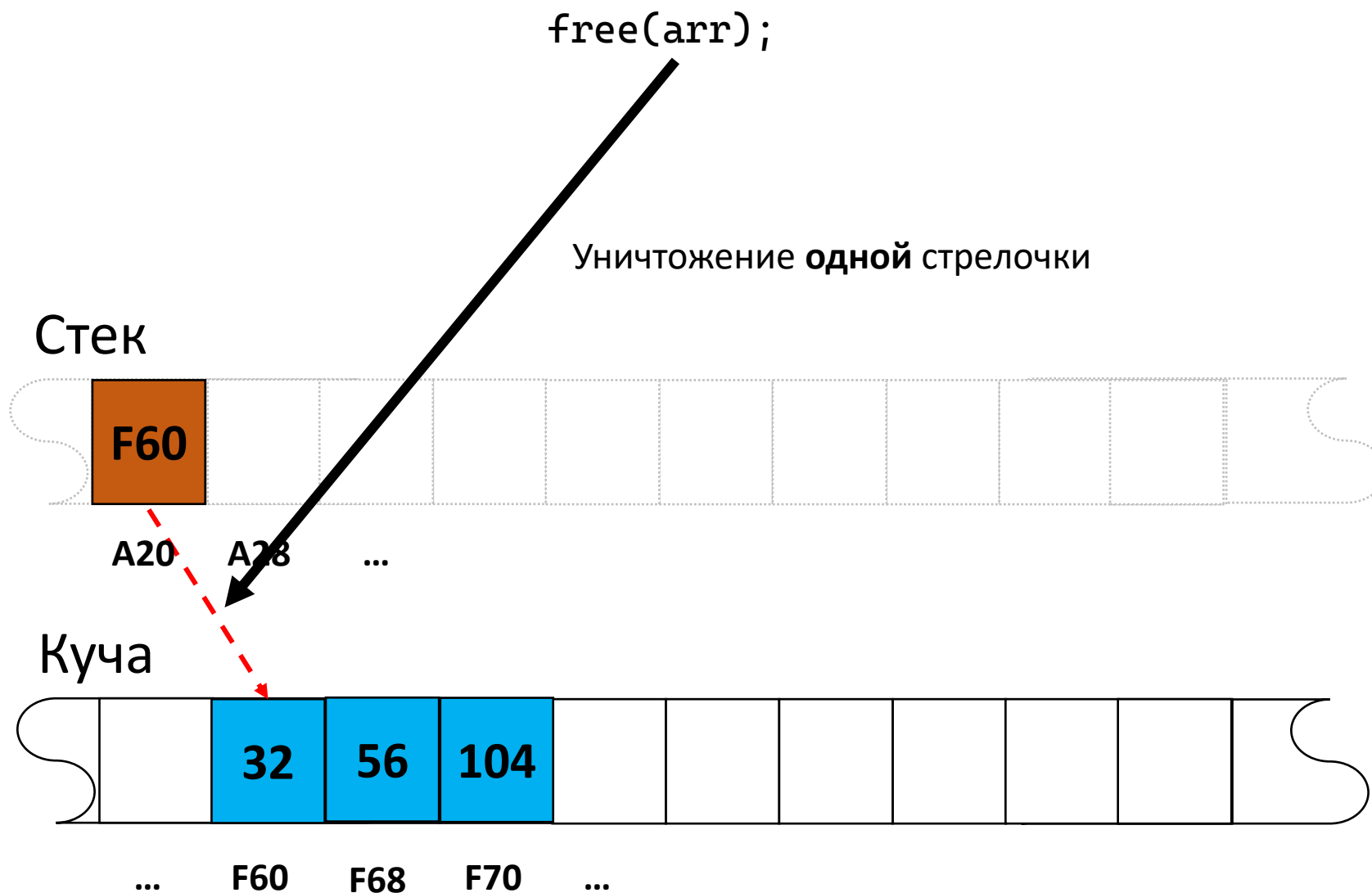


Куча





# Выделение памяти





# Пример

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

const int N = 10;

void input_array(double *x)
{
    int arrMax = 100, arrMin = 0;
    for (int i = 0; i < N; i++)
        x[i] = arrMin + (arrMax - arrMin) * ((double) rand() / RAND_MAX);
}

void print_array(double* x)
{
    for (int i = 0; i < N; i++)
        printf("%lf ", x[i]);
    printf("\n ");
}

int main()
{
    double* array_heap = (double*)malloc(N * sizeof(double));
    input_array(array_heap);
    print_array(array_heap);
    free(array_heap);
    return 0;
}
```



# Теперь можно!

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

void input_array(double *x, int N)
{
    int arrMax = 100, arrMin = 0;
    for (int i = 0; i < N; i++)
        x[i] = arrMin + (arrMax - arrMin) * ((double) rand() / RAND_MAX);
}

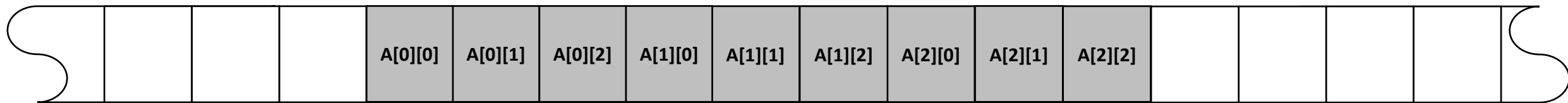
void print_array(double* x, int N)
{
    for (int i = 0; i < N; i++)
        printf("%lf ", x[i]);
    printf("\n ");
}

int main()
{
    int arr_len;
    scanf_s("%d", &arr_len); //размер массива указываем уже после запуска программы
    double* array_heap = (double*)malloc(arr_len * sizeof(double));
    input_array(array_heap, arr_len);
    print_array(array_heap, arr_len);
    free(array_heap);
    return 0;
}
```



# Работа с двумерными массивами

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]







# Работа с двумерными массивами

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

void input_array(double *x, int X_arr, int Y_arr)
{
    int arrMax = 100, arrMin = 0;
    for (int i = 0; i < X_arr; i++)
        for (int j = 0; j < Y_arr; j++)
            x[j + i * X_arr] = arrMin + (arrMax - arrMin) * ((double) rand() / RAND_MAX);
}

void print_array(double* x, int X_arr, int Y_arr)
{
    for (int i = 0; i < X_arr; i++) {
        for (int j = 0; j < Y_arr; j++)
            printf("%lf ", x[j + i * X_arr]);
        printf("\n ");
    }
}

int main()
{
    int arr_X, arr_Y;
    scanf_s("%d%d", &arr_X, &arr_Y);
    double* array_heap = (double*)malloc(arr_X * arr_Y * sizeof(double));
    input_array(array_heap, arr_X, arr_Y);
    print_array(array_heap, arr_X, arr_Y);
    free(array_heap);
    return 0;
}
```

Работаем с двумерным массивом  
как с одномерным



# Работа с двумерными массивами

`int **X`  Указатель на указатель

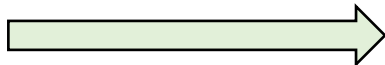
```
int A = 10;
int* p1 = &A;
int** p2 = &p1;
printf("A = \t%d\n", A);
printf("p1 = \t%p\n", p1);
printf("p2 = \t%p\n", p2);
printf("*p2 = \t%p\n", *p2);
printf("*p1 = \t%d\n", *p1);
printf("**p2 = \t%d\n", **p2);
```

```
A =      10
p1 =      0000002457CFFD38
p2 =      0000002457CFFD30
*p2 =      0000002457CFFD38
*p1 =      10
**p2 =      10
```



# Работа с двумерными массивами

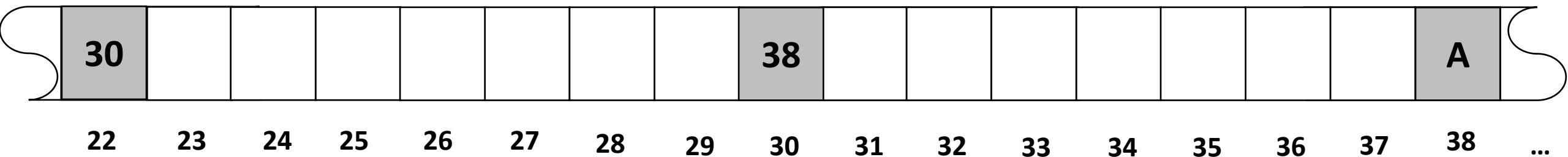
`int **X`



Указатель на указатель

```
int A = 10;
int* p1 = &A;
int** p2 = &p1;
printf("A = \t%d\n", A);
printf("p1 = \t%p\n", p1);
printf("p2 = \t%p\n", p2);
printf("*p2 = \t%p\n", *p2);
printf("*p1 = \t%d\n", *p1);
printf("**p2 = \t%d\n", **p2);
```

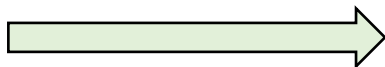
```
A =      10
p1 =      0000002457CFFD38
p2 =      0000002457CFFD30
*p2 =      0000002457CFFD38
*p1 =      10
**p2 =      10
```





# Работа с двумерными массивами

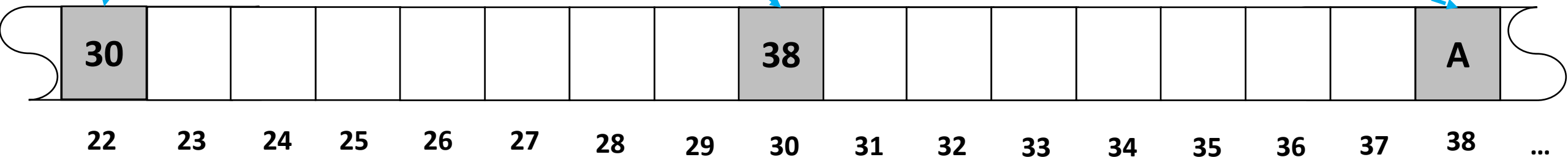
`int **X`



Указатель на указатель

```
int A = 10;
int* p1 = &A;
int** p2 = &p1;
printf("A = \t%d\n", A);
printf("p1 = \t%p\n", p1);
printf("p2 = \t%p\n", p2);
printf("*p2 = \t%p\n", *p2);
printf("*p1 = \t%d\n", *p1);
printf("**p2 = \t%d\n", **p2);
```

```
A =      10
p1 =     0000002457CFFD38
p2 =     0000002457CFFD30
*p2 =    0000002457CFFD38
*p1 =     10
**p2 =    10
```





# Выделение памяти под двумерный массив

```
double** array_heap = (double**)malloc(arr_X * sizeof(double*));  
for (int i = 0; i < arr_X; ++i)  
{  
    array_heap[i] = (double*)malloc(arr_Y * sizeof(double));  
}
```

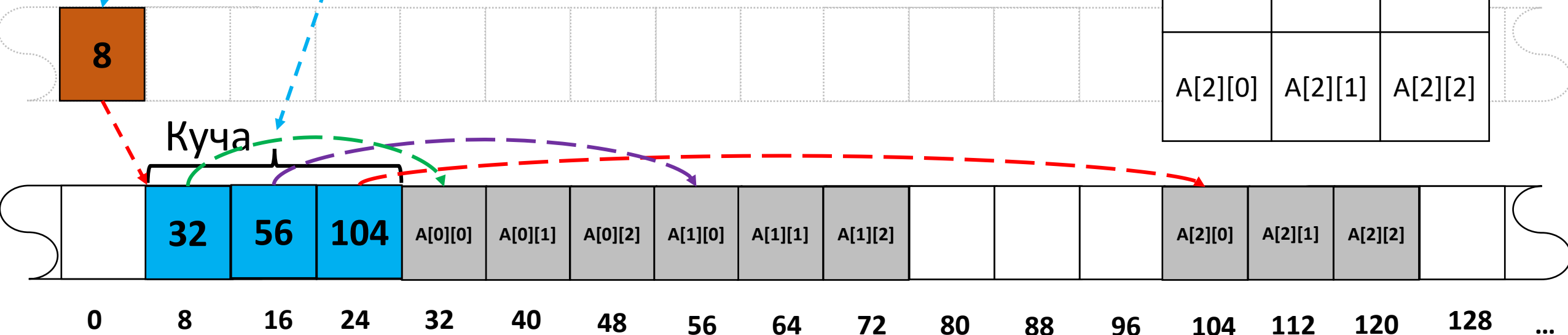
Число строк

Число столбцов

Стек

Куча

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]

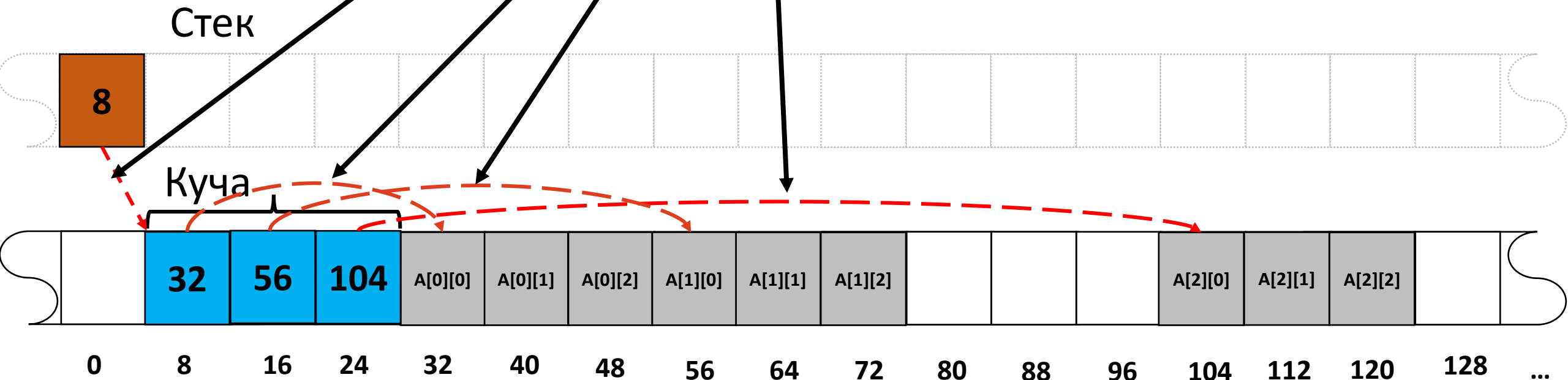




# Выделение памяти под двумерный массив

```
for (int i = 0; i < arr_X; i++)  
    free(array_heap[i]);  
free(array_heap);
```

Необходимо удалить **все** стрелочки





# Работа с двумерными массивами

```
#include <stdio.h>
#include <stdlib.h>
void input_array(double** x, int X_arr, int Y_arr){
    int arrMax = 100, arrMin = 0;
    for (int i = 0; i < X_arr; i++)
        for (int j = 0; j < Y_arr; j++)
            x[i][j] = arrMin + (arrMax - arrMin) * ((double)rand() / RAND_MAX);
}
void print_array(double** x, int X_arr, int Y_arr){
    for (int i = 0; i < X_arr; i++) {
        for (int j = 0; j < Y_arr; j++)
            printf("%lf ", x[i][j]);
        printf("\n ");
    }
}
int main(){
    int arr_X, arr_Y;
    scanf_s("%d%d", &arr_X, &arr_Y);
    double** array_heap = (double**)malloc(arr_X * sizeof(double*));
    for (int i = 0; i < arr_X; ++i){
        array_heap[i] = (double*)malloc(arr_Y * sizeof(double));
    }
    input_array(array_heap, arr_X, arr_Y);
    print_array(array_heap, arr_X, arr_Y);
    for (int i = 0; i < arr_X; i++) // цикл по строкам
        free(array_heap[i]);      // освобождение памяти под строку
    free(array_heap);
    return 0;
}
```

Работаем как с обычным  
двумерным массивом



## Домашнее задание 6

Доработать программу калькулятор следующим образом:

1. Сделать матрицы динамическими. Обеспечить корректное выделение и освобождение памяти. Размер матриц возможен произвольный.

Срок выполнения: 28.03.22