



Пример

```
gcc -g gdb_vals.c -o gdb_vals  
gdb gdb_vals
```

https://github.com/SergeyBalabaev/Elective-C-Programming-Language/blob/main/lesson4/gdb_vals.c



GDB

```
list
break 32
clear 32
run
info registers
info locals
ptype a
print a
print &a
print sizeof(a)
set var a = 512
print a
```



GDB

`x/4xb &a`

Вывести

4 байта

в 16 формате

побайтово

с адреса переменной a



x/80xb &b

[illegible]



GDB – возможный вариант решения

```
int main()
{
    int a = 1024;
    char b = 'b';
    int c[4] = {1,2,3,4};
    int *d = &a;
    int **d1 = &d;
    double e = 3.14;
    char g[4] = {1,2,3,4};
    struct str f;
    f.a = 1;
    f.b = '2';
    f.c = 3;
    union code h;
    h.a = 1;
    h.b = '2';
    h.c = 3;
    return 0;
}
```

0x7fffffffde3b:	0x62	0x00	0x04	0x00	0x00	0x3c	0xde	0xff
0x7fffffffde43:	0xff	0xff	0x7f	0x00	0x00	0x00	0x00	0x00
0x7fffffffde4b:	0x00	0x00	0x00	0x08	0x40	0x40	0xde	0xff
0x7fffffffde53:	0xff	0xff	0x7f	0x00	0x00	0x1f	0x85	0xeb
0x7fffffffde5b:	0x51	0xb8	0x1e	0x09	0x40	0x01	0x00	0x00
0x7fffffffde63:	0x00	0x32	0x7f	0x00	0x00	0x00	0x00	0x00
0x7fffffffde6b:	0x00	0x00	0x00	0x08	0x40	0x01	0x00	0x00
0x7fffffffde73:	0x00	0x02	0x00	0x00	0x00	0x03	0x00	0x00
0x7fffffffde7b:	0x00	0x04	0x00	0x00	0x00	0x80	0xdf	0xff
0x7fffffffde83:	0xff	0x01	0x02	0x03	0x04	0x00	0xb5	0xcd



Графический интерфейс

```
gdb_vals.c
19      int c[4] = {1,2,3,4};
20      int *d = &a;
21      int **d1 = &d;
22      double e = 3.14;
23      char g[4] = {1,2,3,4};
24      struct str f;
25      f.a = 1;
26      f.b = '2';
27      f.c = 3;
28      union code h;
29      h.c = 3;
30      h.b = '2';
31      h.a = 1;
B->32      return 0;
33      }
```

native process 35 In: main L32 PC: 0x555555551df

(gdb) list

(gdb) b 32

Breakpoint 1 at 0x11df: file gdb_vals.c, line 32.

(gdb) r

Starting program: /mnt/c/Users/79629/Documents/ubuntu_files/Lesson7/gdb_vals

Breakpoint 1, main () at gdb_vals.c:32

(gdb)

tui enable



Ответьте на следующие вопросы

1. Какой тип у переменной `c`?
2. Какой тип у переменной `&c`?
3. Какой тип у переменной `c[1]`?
4. Выполните команды: `print sizeof(d)` и `print sizeof(*d)`
5. Какой размер у структуры? Совпадает ли он с суммой размеров всех переменных?
6. Какой размер у объединения?
7. Положите в объединение значение 97 в ячейку `h.a` и выведите `h.b`



Динамическое выделение памяти



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



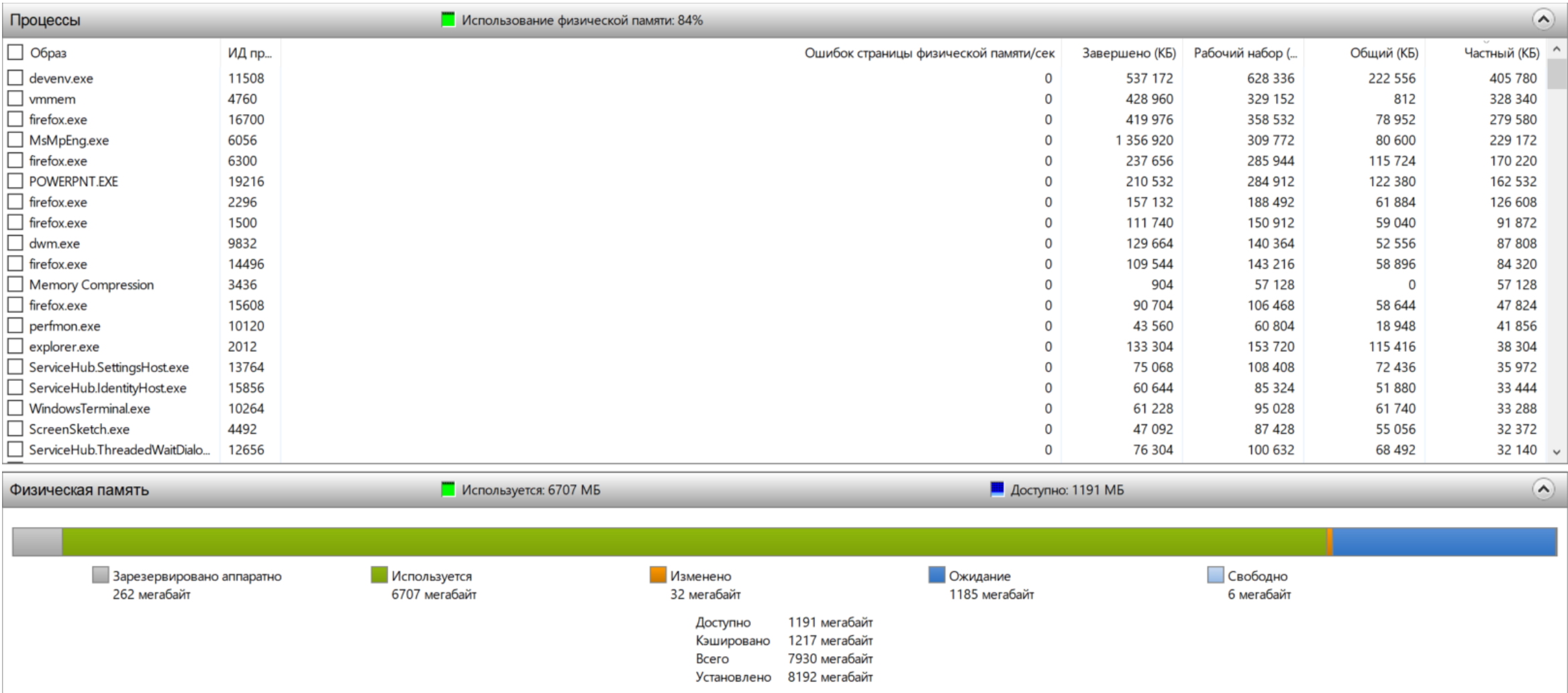


Устройство памяти процесса



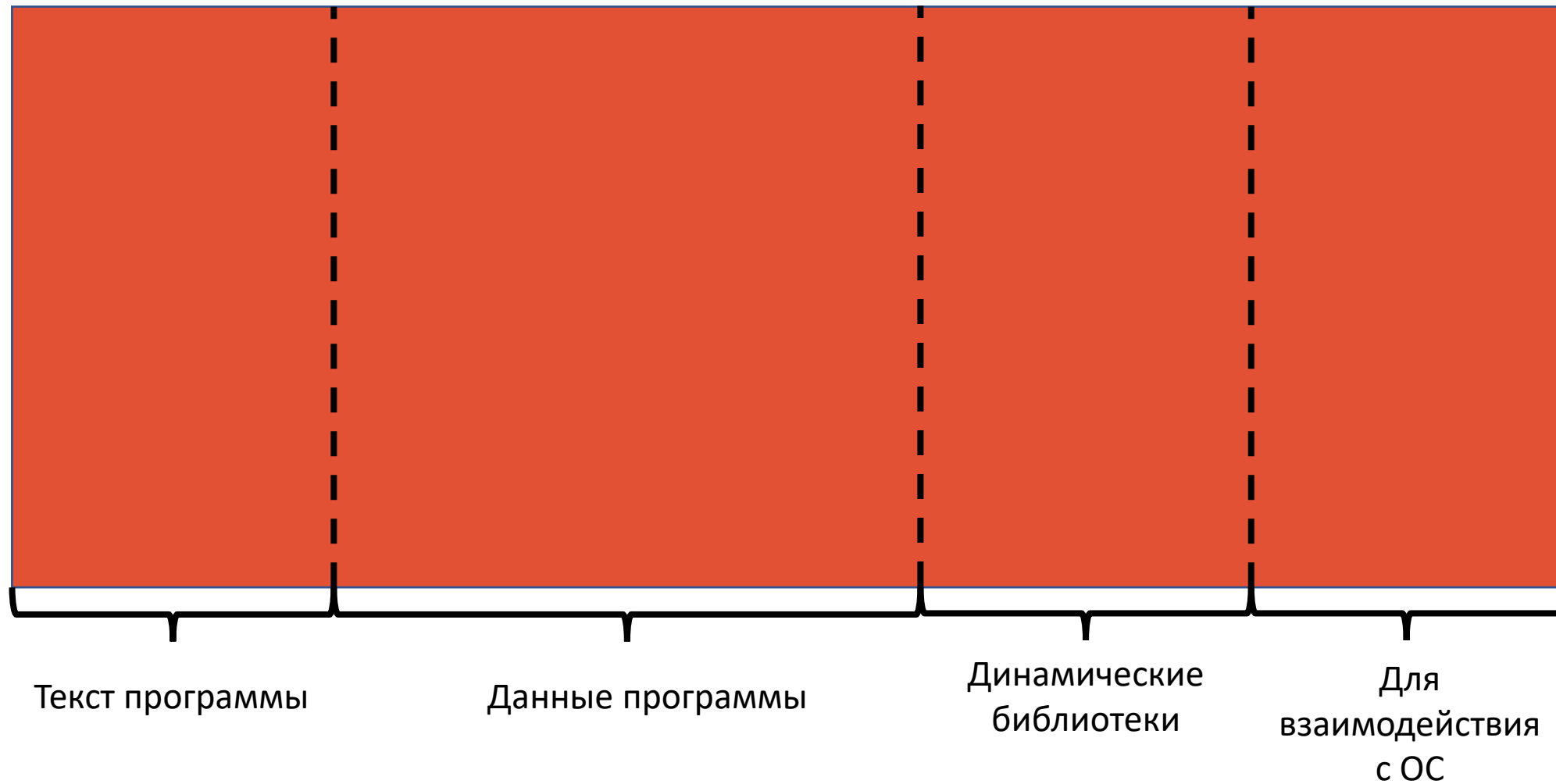


Монитор ресурсов



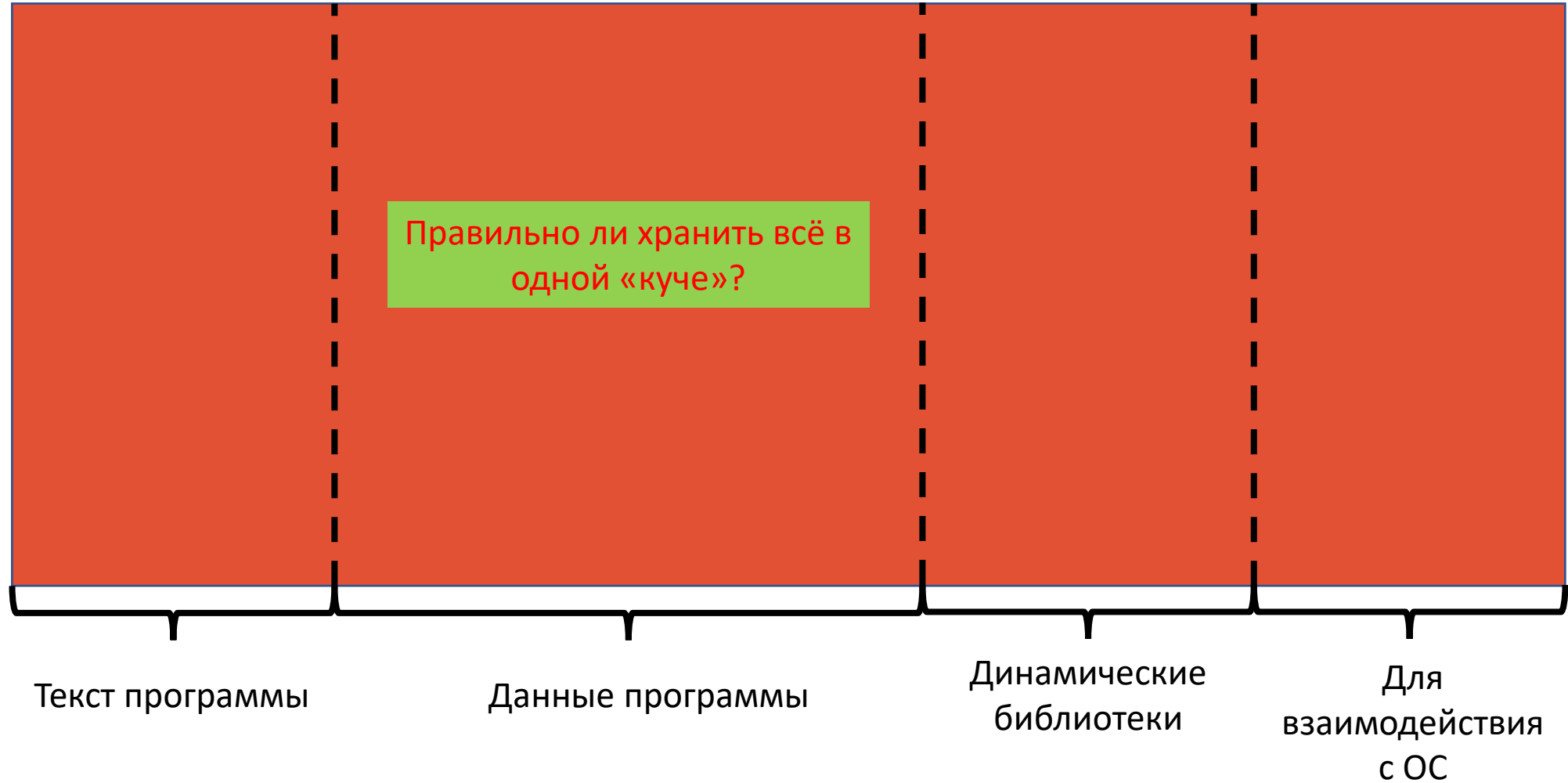


Устройство памяти процесса



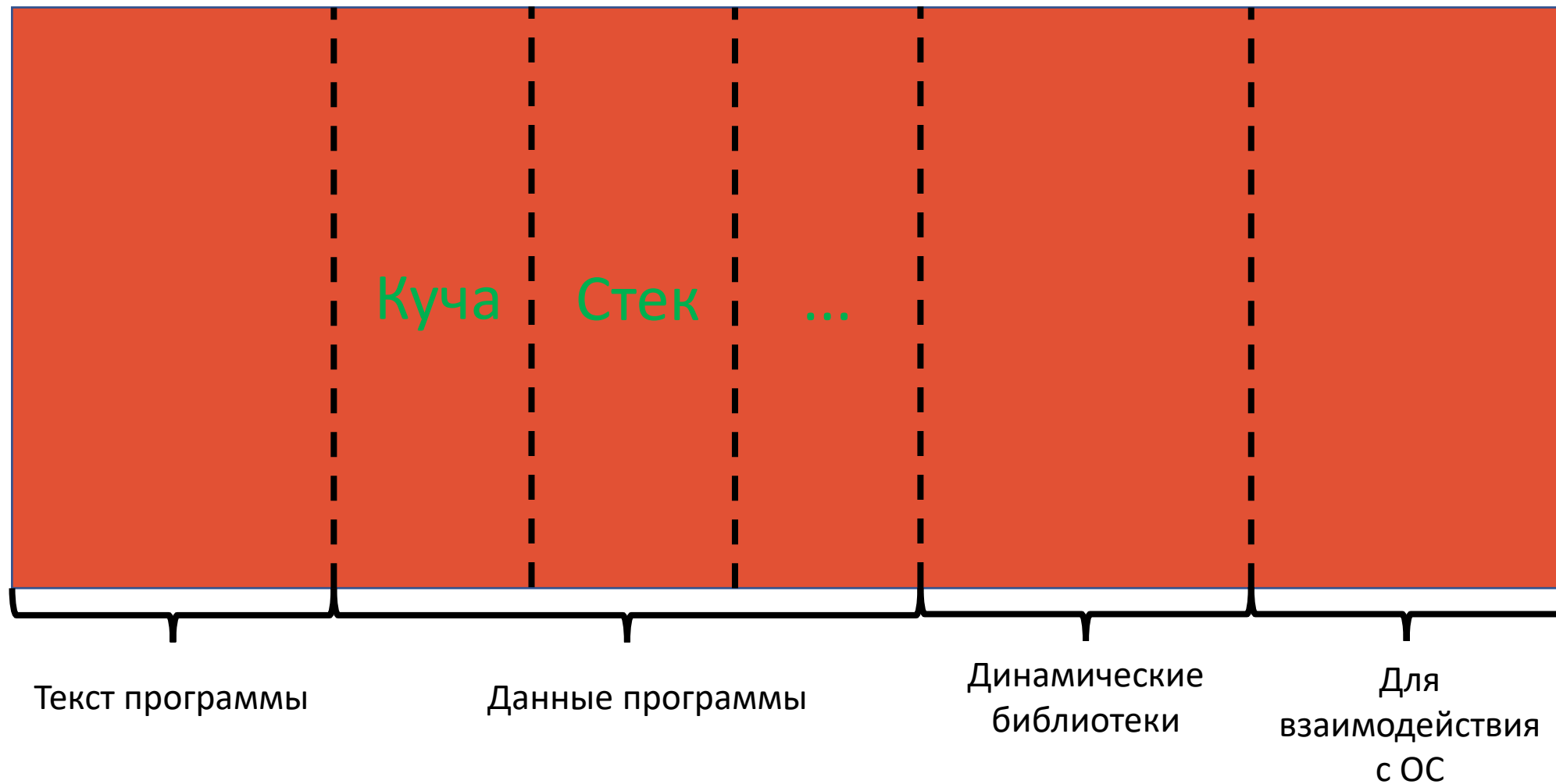


Устройство памяти процесса





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]
7f16bfc000-7f16bfd06000 r--p 00000000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfd06000-7f16bfe7e000 r-xp 00025000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfe7e000-7f16bfec8000 r--p 0019d000 08:10 11116 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f16bfec8000-7f16bfec9000 ---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]
```



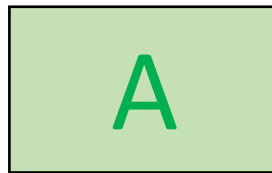
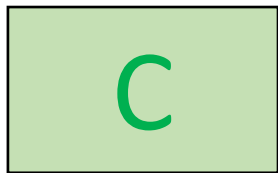
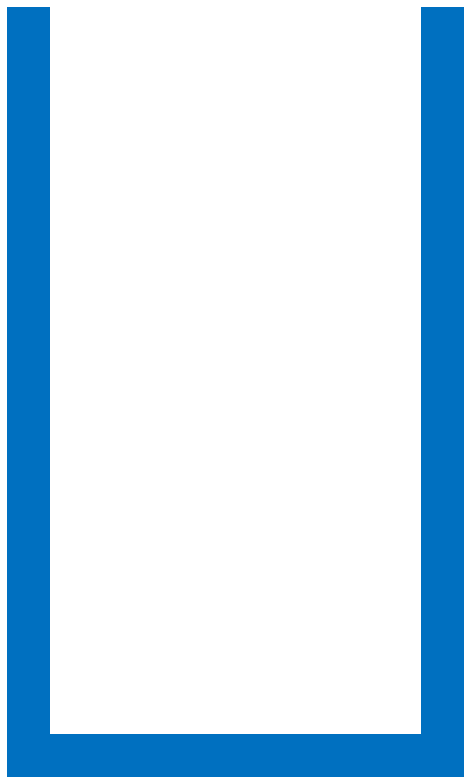

Стек



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

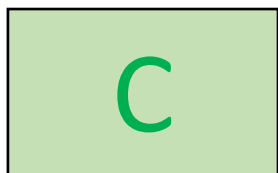
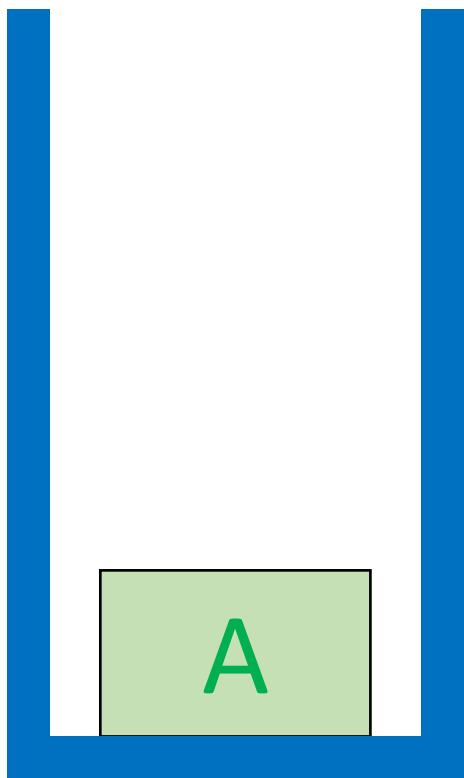


Стек



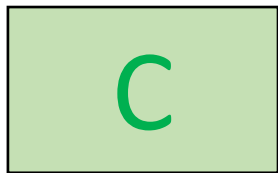
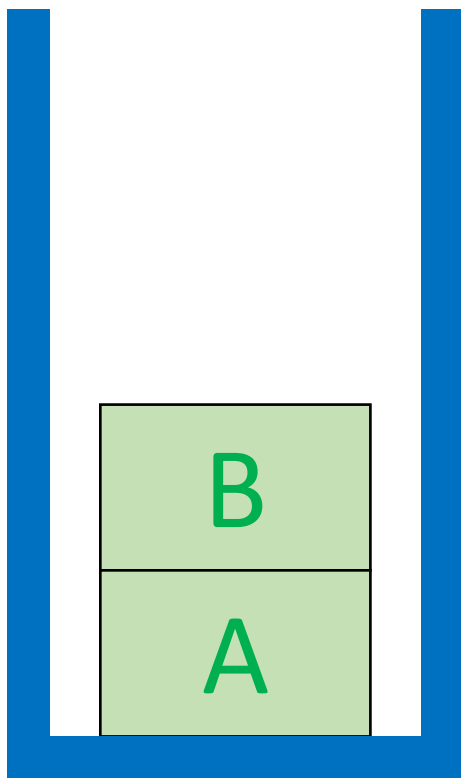


Стек



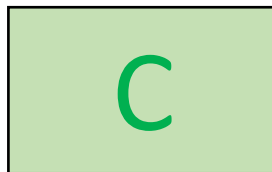
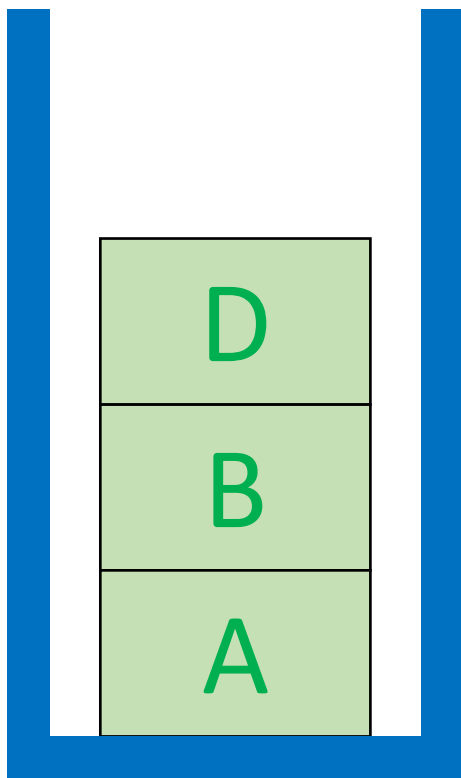


Стек



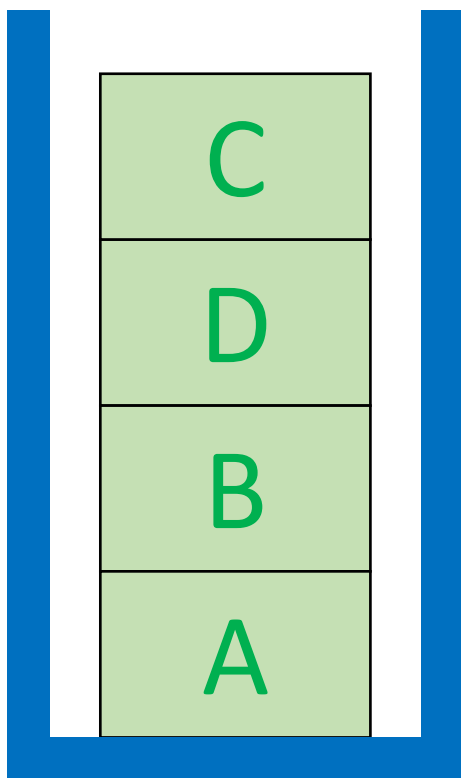


Стек



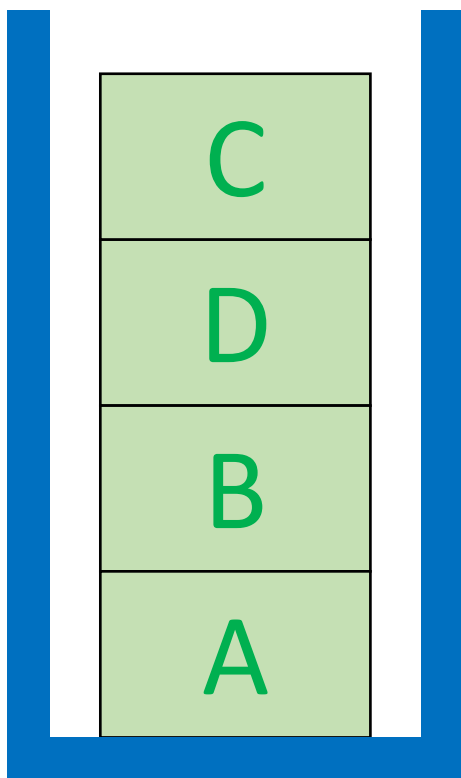


Стек





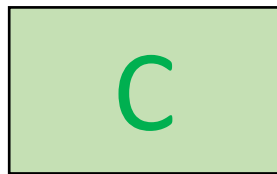
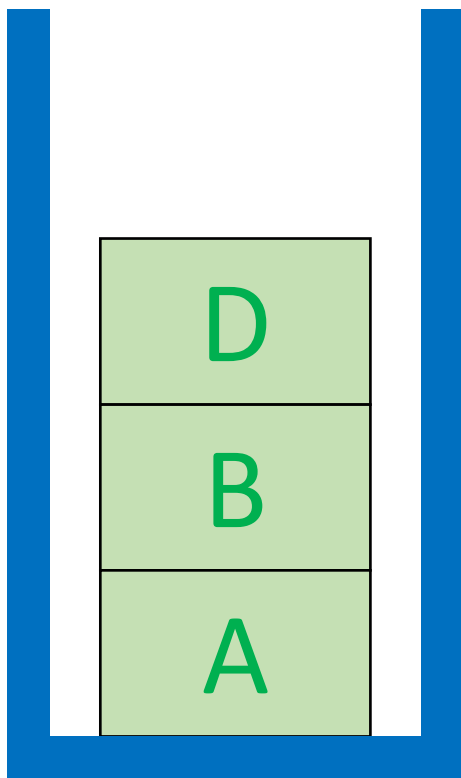
Стек



Как достать «B»?

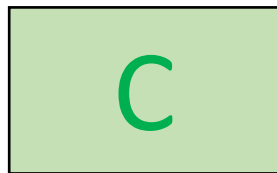
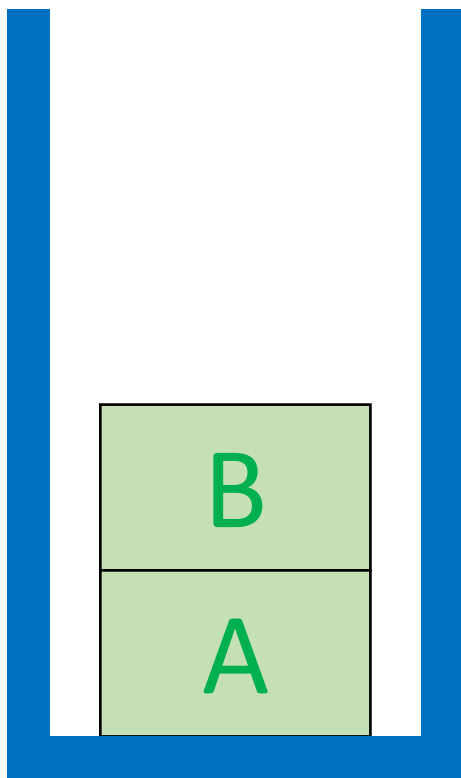


Стек



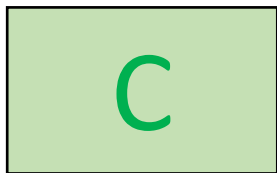
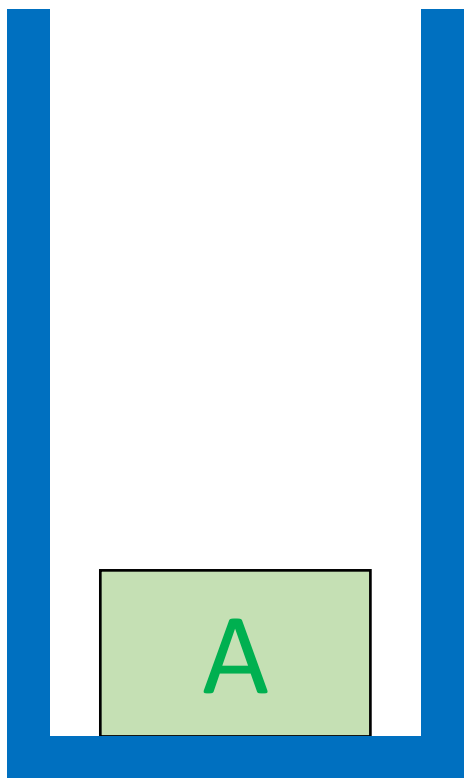


Стек



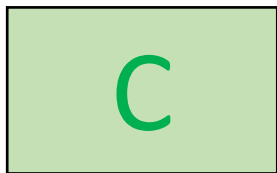
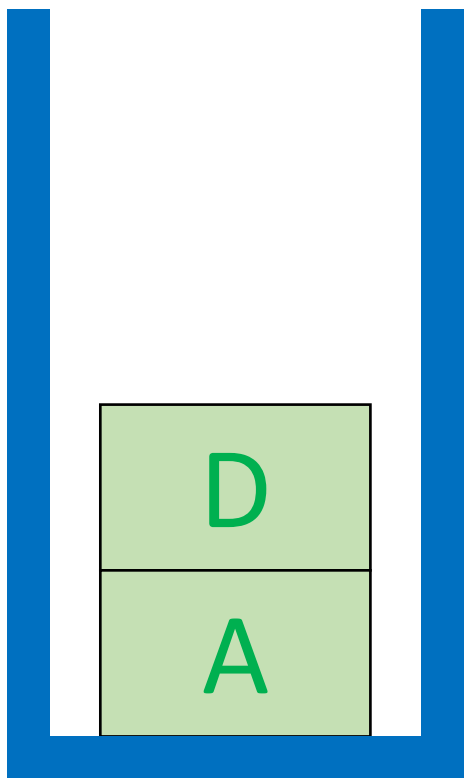


Стек



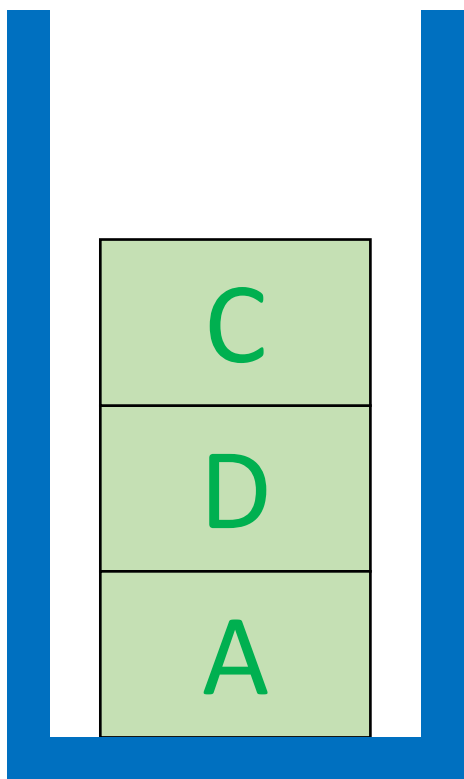


Стек





Стек





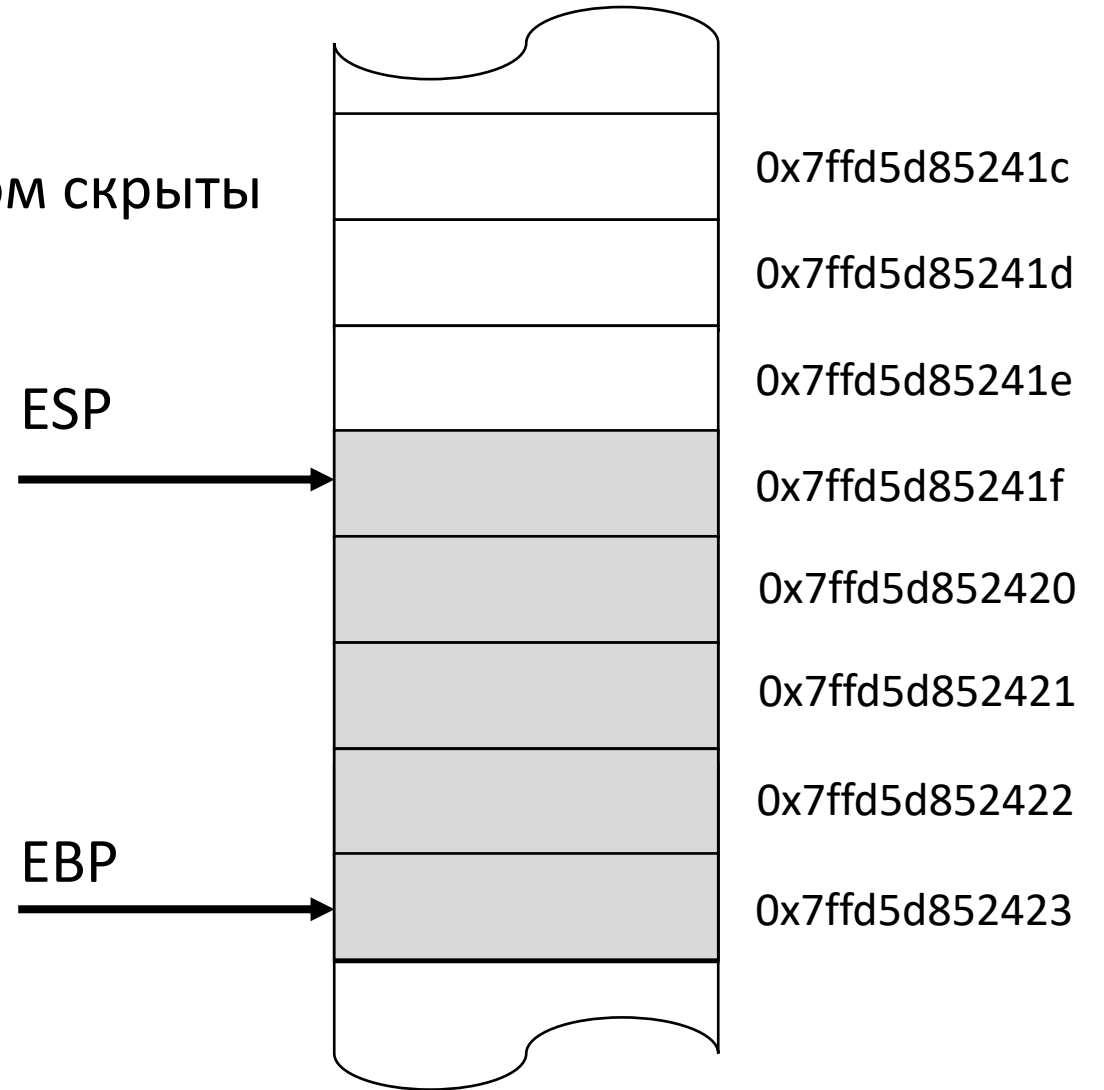
Стек

Для программиста операции работы со стеком скрыты

На языке Ассемблер (x86) используются специальные регистры

ESP – указатель на вершину стека

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

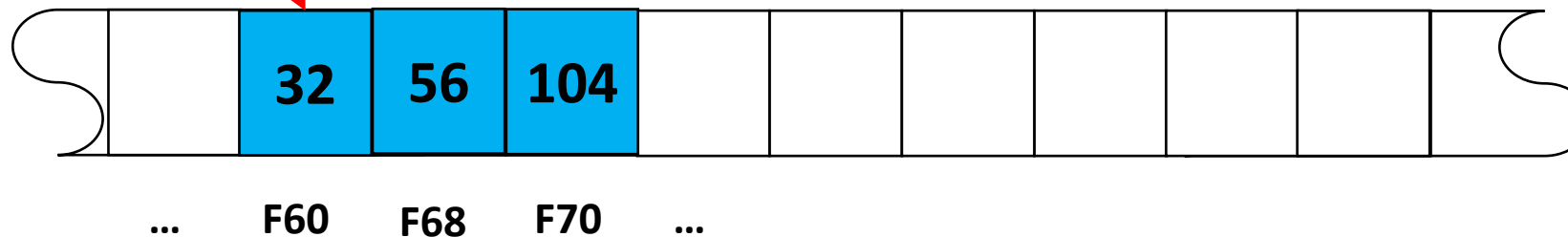
```
double* arr = (double*)malloc(N * sizeof(double));
```

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

Стек

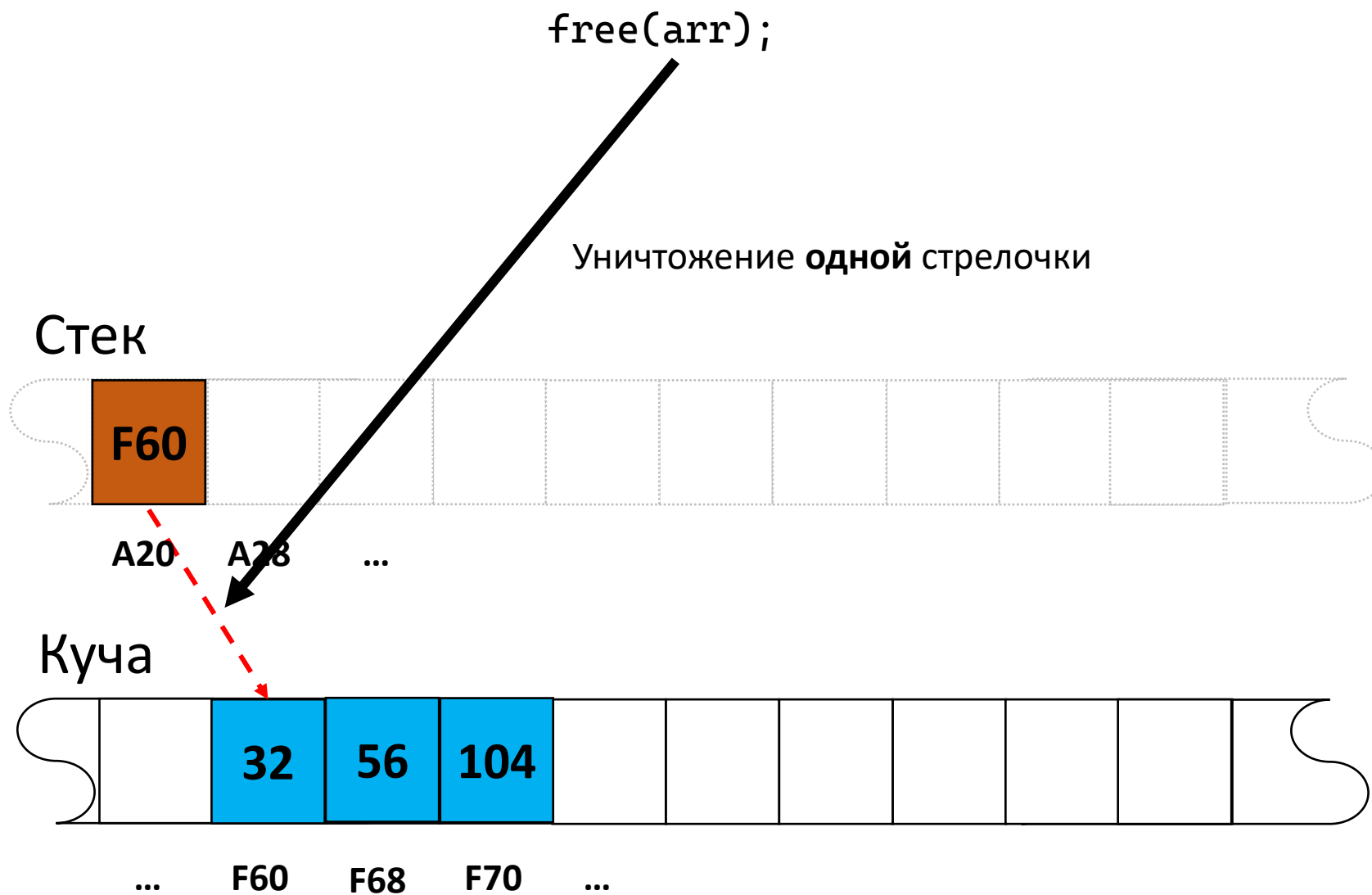


Куча





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);
}

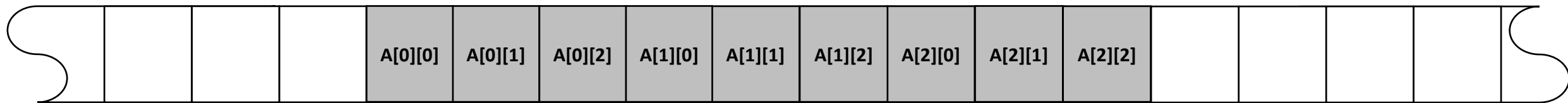
void print_array(double* x, int N)
{
    for (int i = 0; i < N; i++)
        printf("%lf ", x[i]);
    printf("\n ");
}

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

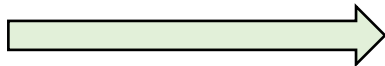
```
int A = 10;  
int* p1 = &A;  
int** p2 = &p1;  
printf("A = \t%d\n", A);  
printf("p1 = \t%p\n", p1);  
printf("p2 = \t%p\n", p2);  
printf("*p2 = \t%p\n", *p2);  
printf("*p1 = \t%d\n", *p1);  
printf("**p2 = \t%d\n", **p2);
```

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



Работа с двумерными массивами

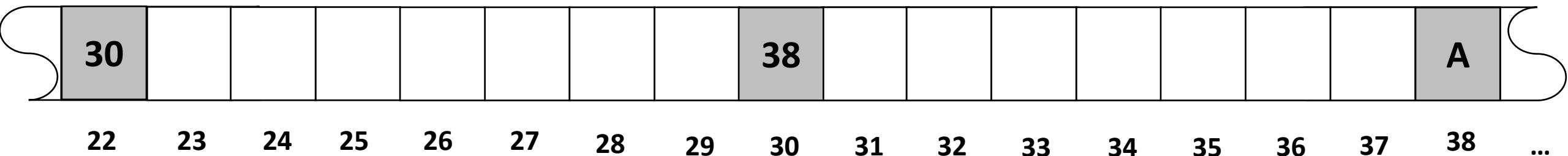
`int **X`



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);
```

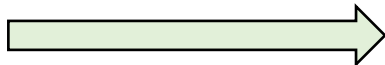
```
A =      10
p1 =      0000002457CFFD38
p2 =      0000002457CFFD30
*p2 =      0000002457CFFD38
*p1 =      10
**p2 =      10
```





Работа с двумерными массивами

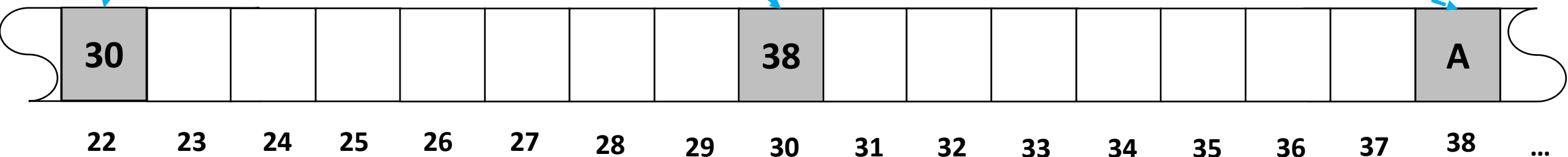
`int **X`



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
```



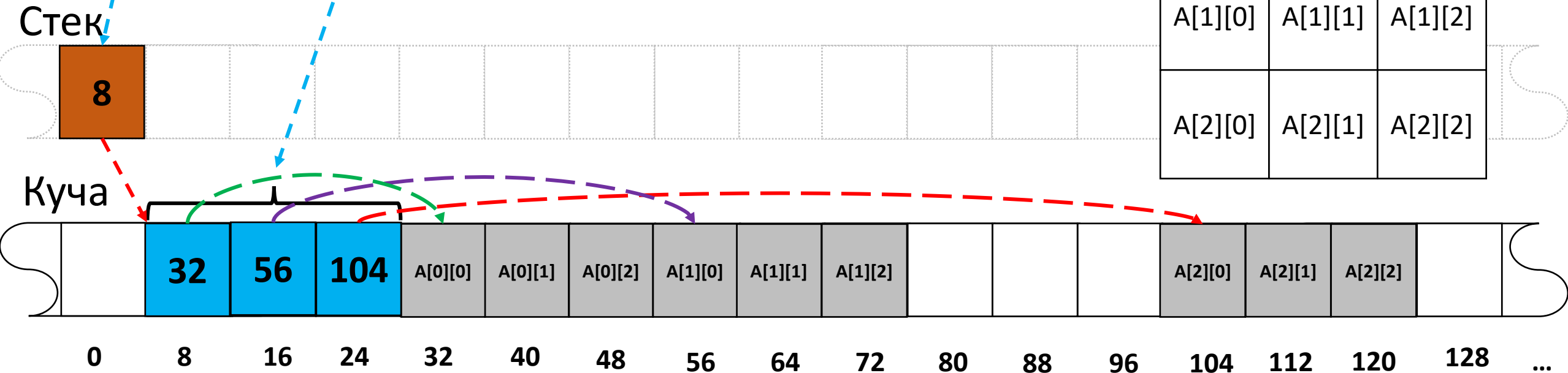


Выделение памяти под двумерный массив

```
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





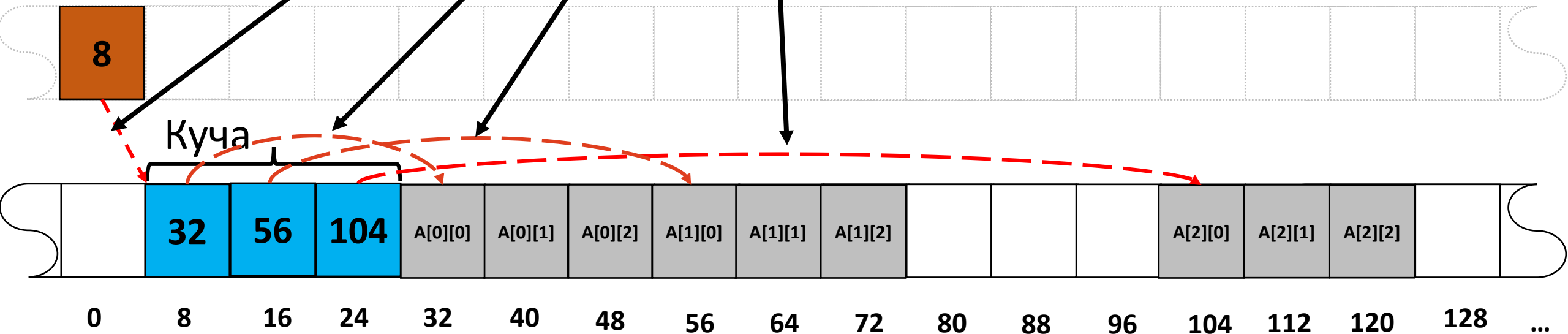
Выделение памяти под двумерный массив

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

Необходимо удалить **все** стрелочки

Стек

Куча





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



Утечки памяти

```
gcc mem_leaks.c -o mem_leaks  
valgrind -s ./mem_leaks
```

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

```
int main()  
{  
    int N_mas = 10;  
    double** A = (double**)malloc(N_mas * sizeof(double*));  
    for (int i = 0; i < N_mas; i++) {  
        A[i] = (double*)malloc(N_mas * sizeof(double));  
    }  
  
    free(A);  
    return 0;  
}
```

Что не так?



Утечки памяти

```
==390== Memcheck, a memory error detector
==390== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==390== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==390== Command: ./mem_leaks
==390==
==390==
==390== HEAP SUMMARY:
==390==   in use at exit: 800 bytes in 10 blocks
==390== total heap usage: 11 allocs, 1 frees, 880 bytes allocated
==390==
==390== LEAK SUMMARY:
==390==   definitely lost: 800 bytes in 10 blocks
==390==   indirectly lost: 0 bytes in 0 blocks
==390==   possibly lost: 0 bytes in 0 blocks
==390==   still reachable: 0 bytes in 0 blocks
==390==     suppressed: 0 bytes in 0 blocks
==390== Rerun with --leak-check=full to see details of leaked memory
==390==
==390== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0
```




Утечки памяти

```
for (int i = 0; i < N_mas; i++) {  
    free(A[i]);  
}  
free(A);
```

```
==397== Memcheck, a memory error detector  
==397== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  
==397== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info  
==397== Command: ./mem_leaks  
==397==  
==397==  
==397== HEAP SUMMARY:  
==397==   in use at exit: 0 bytes in 0 blocks  
==397== total heap usage: 11 allocs, 11 frees, 880 bytes allocated  
==397==  
==397== All heap blocks were freed -- no leaks are possible  
==397==  
==397== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```



New VS malloc

В чем разница между new
и malloc

new

1. C++
2. Оператор -> можно перегрузить
3. Алгоритм работы:
 1. Выделение памяти
 2. Вызов конструктора

malloc

1. C
2. Функция
3. Алгоритм работы:
 1. Выделение памяти

Код см. в репозитории



Что будет выведено на экран?

[illegible]



Копирование указателей

```
==318== Invalid free() / delete / delete[] / realloc()
==318==   at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==318==   by 0x109244: main (in /mnt/c/Users/79629/Documents/ubuntu_files/C++/copyptr)
==318== Address 0x4a48040 is 0 bytes inside a block of size 40 free'd
==318==   at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==318==   by 0x109238: main (in /mnt/c/Users/79629/Documents/ubuntu_files/C++/copyptr)
==318== Block was alloc'd at
==318==   at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==318==   by 0x1091A7: main (in /mnt/c/Users/79629/Documents/ubuntu_files/C++/copyptr)
==318==
==318==
==318== HEAP SUMMARY:
==318==   in use at exit: 0 bytes in 0 blocks
==318== total heap usage: 2 allocs, 3 frees, 1,064 bytes allocated
==318==
==318== All heap blocks were freed -- no leaks are possible
==318==
==318== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```



Передача динамического массива в функцию

```
void init_arr(int* arr, int N){  
    for (size_t i = 0; i < N; i++)  
    {  
        arr[i] = rand() % 100;  
    }  
}
```



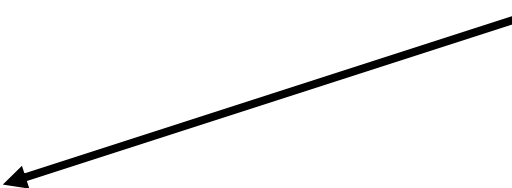
Тут все хорошо, работаем как с
обычным массивом

Попробуем написать функцию,
которая увеличит память на N ячеек



Вызов realloc в функции

Передаем адрес указателя



```
void increase_array_size(int **a, int n, int n_new) {  
    int *q;  
    q = (int*) realloc(*a, n_new*sizeof(int));  
    if (q==NULL) { /* проверка успешности увеличения памяти */  
        printf("Error increase array size");  
    }  
    else {  
        *a = q;  
    }  
}
```

```
increase_array_size(&ptr, N, N + 20);
```