Факультатив по программированию на языке C

Занятие 7
Аллокация памяти

План занятий

№	Тема	Описание
1	Введение в курс	Языки программирования. Основы работы с Linux.
2	Основы языка С	Написание и компиляция простейших программ с использованием gcc. Правила написания кода.
3	Компиляция	Разбиение программы на отдельные файлы. Make файлы. Компиляция.
4	Ввод данных. Библиотеки	Работа со вводом/выводом. Статические и динамические библиотеки.
5	Хранение данных. Память	Хранение процесса в памяти компьютера. Виртуальная память, сегментация. Секции программы.
6	Устройство памяти.	Elf файлы. Указатели и массивы. Типы данных. Gdb и отладка
7	Аллокация памяти	Аллокация памяти. Битовые операции – сдвиги, логические операции. Битовые поля.
8	Обработка данных	Безопасные функции. Перечисления. Static переменные. Inline функции. Макросы
9	Язык ассемблера	Основы анализа программ на языке ассемблер.

Дерево языка

Напоминание

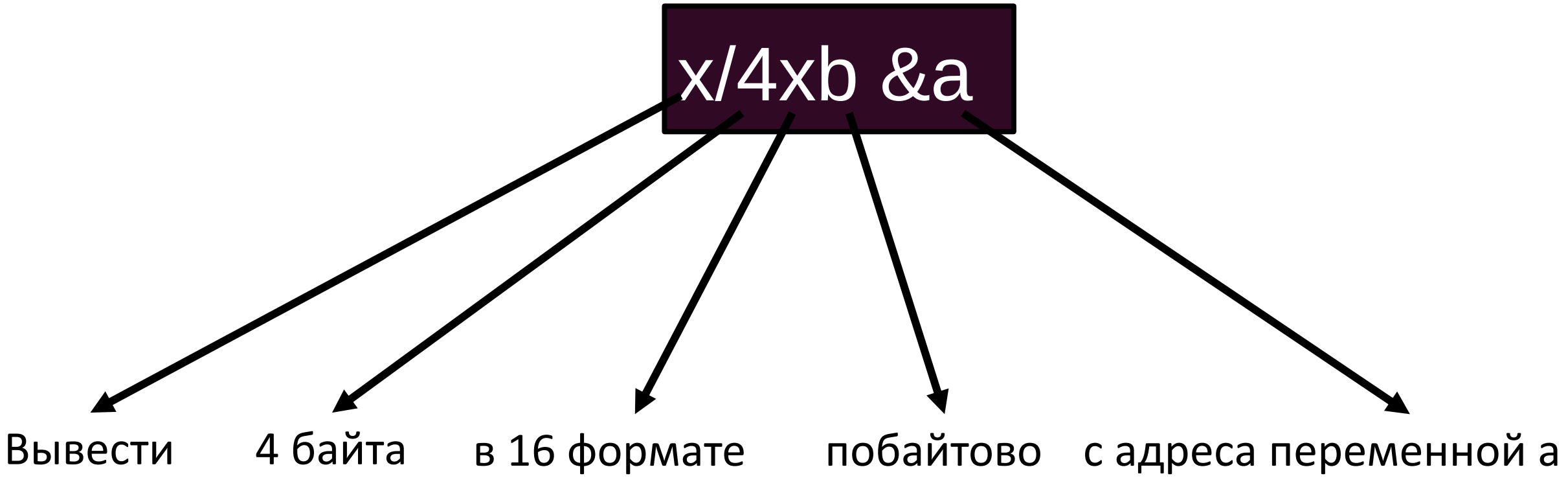
```
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from gdb_vals...
(gdb)
```

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

GDB

```
list
break 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

GDB

```
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;
}
```

x/80xb &b

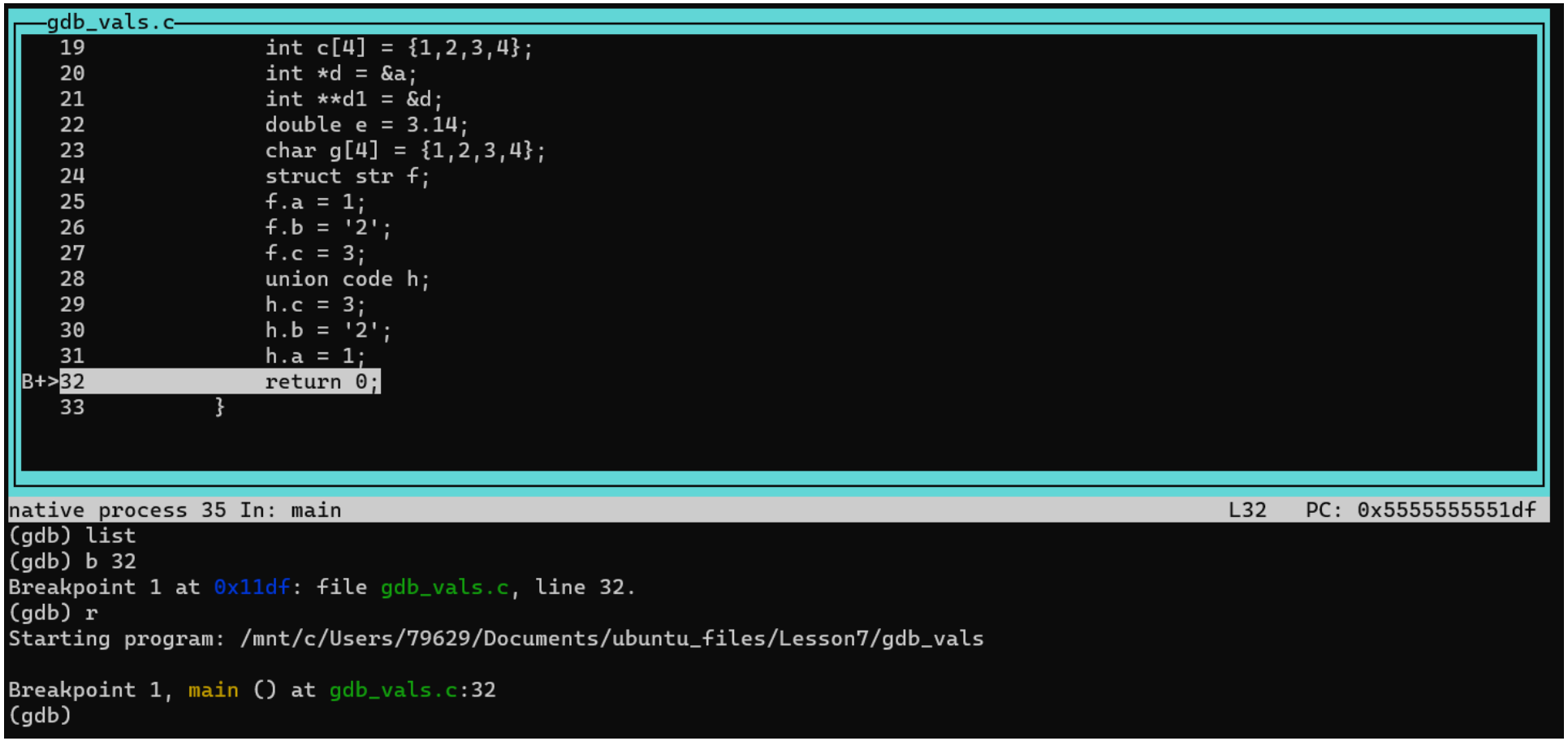
0x								
0x								
0x								
0x								
0x								
0x								
0x								
0x								

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



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

Некоторые общие вопросы о памяти

```
struct S
{
    char a;
    int b;
};
```

Какой размер данной структуры?

```
struct S
{
    char a;
    short c;
    int b;
};
```

Какой размер данной структуры?

Как уменьшить размер?

Можно ли оптимизировать?

Оценки школьника

```c
typedef struct Student {
    char name[20];
    int Russian;
    int Literature;
    int Physics;
    int English;
    int Science;
    int Mathematics;
    int History;
    int Geography;
    int Social_Studies;
    int Chemistry;
    int Biology;
    int Economics;
    int Art;
    int IT;
} St;
```

Какой размер данной структуры?

```
sab@LAPTOP-B03PIUAN:.../Lesson7$ ./bit_field
76
```

Битовые поля

```
struct {
    type member_name  : width;
};
```

```
struct device {
    unsigned active : 1;
    unsigned ready : 1;
    unsigned error : 1;
} dev_code;
```

Type -> signed, unsigned, int or _Bool

Оценки школьника – оптимальное решение

```c
#include <stdio.h>
typedef struct Student {
    char name[20];
    int Russian : 3;
    int Literature : 3;
    int Physics : 3;
    int English : 3;
    int Science : 3;
    int Mathematics : 3;
    int History : 3;
    int Geography : 3;
    int Social_Studies : 3;
    int Chemistry : 3;
    int Biology : 3;
    int Economics : 3;
    int Art : 3;
    int IT : 3;
} St;
```

Какой размер данной структуры?

```
sab@LAPTOP-B03PIUAN:.../Lesson7$ ./bit_field_2
28
```

Битовые операции

Операция	**Реализация**
ИЛИ	A | B
И	A & B
НЕ	~A
Исключающее ИЛИ	A ^ B
Логический сдвиг	A << 1 A >> 1

Операция выставления бита

b = 00**0**00000

```
b = b | 1 << 5;
```

```
b |= 1 << 5;
```

b = 00**1**00000

Небольшая задачка

Оформите в виде функций:

1. Выставление n-го бита в единицу
2. Выставление n-го бита в ноль
3. Инвертирование n-го бита
4. Возвращение значения n-го бита

Возможное решение

```
int setbit(const int value, const int position) {
    return (value | (1 << position));
}
int unsetbit(const int value, const int position) {
    return (value & ~(1 << position));
}
int switchbit(const int value, const int position) {
    return (value ^ (1 << position));
}
int checkbit(const int value, const int position) {
    return ((value & (1 << position)) != 0);
}
```

Утечки памяти

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

```c
#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

Копирование указателей

```c
#include <stdio.h>
#include <stdlib.h>
const int N = 10;

int main() {
    int* ptr = (int*)malloc(N * sizeof(N));
    for (size_t i = 0; i < N; i++)
    {
        ptr[i] = i;
    }
    int* ptr2 = ptr;
    free(ptr);
    for (size_t i = 0; i < N; i++)
    {
        printf("%d\n",ptr2[i]);
    }
    free(ptr2);
    return 0;
}
```

Что будет выведено на экран?

```
-572662307
-572662307
-572662307
-572662307
-572662307
-572662307
-572662307
-572662307
-572662307
-572662307
```

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)
```

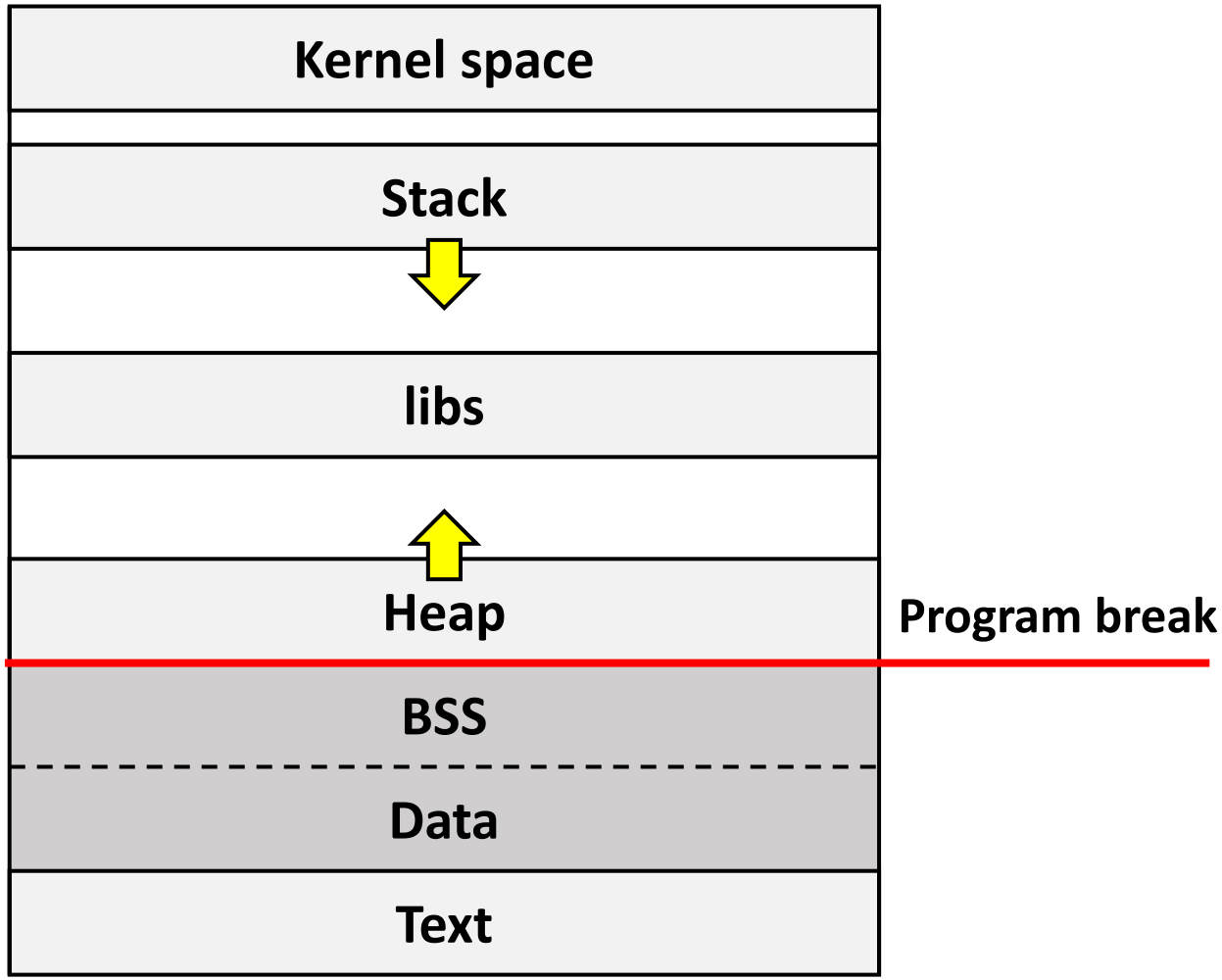
Аллокация памяти

Системные вызовы для работы с памятью:

brk()/sbrk() — используется для изменения объема памяти, выделенной процессу.

mmap() – позволяет отображать память из любого места процесса

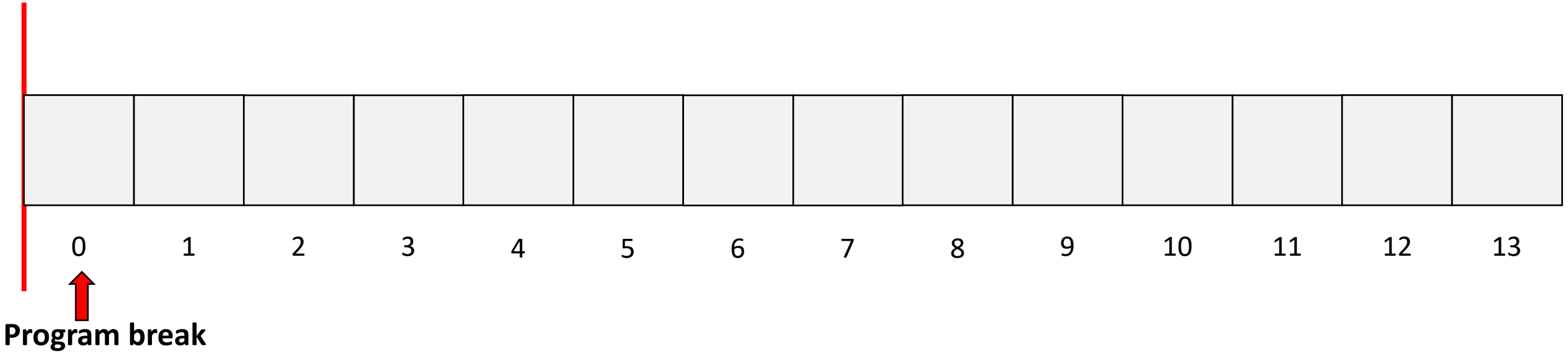
Аллокация памяти



С помощью системного вызова ***sbrk()*** возможно изменять положение **Program break**

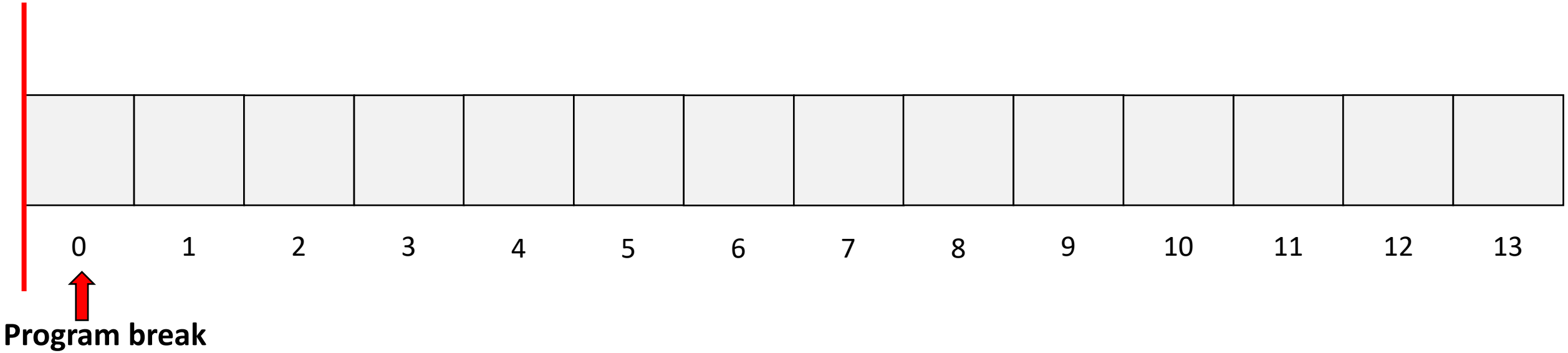
Аллокация памяти

```
int* x = (int*)malloc(N);
```



Аллокация памяти

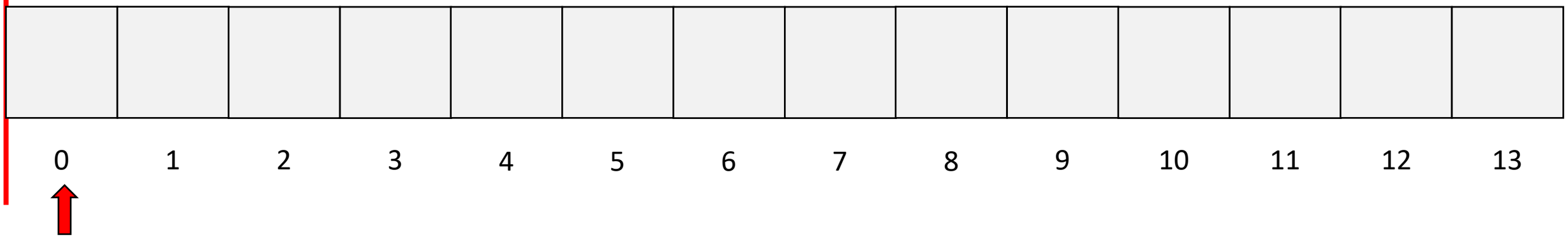
```
int* x = (int*)malloc(N);
```




```
struct mem_control_block {
        int is_available;
        int size;
};
```

Аллокация памяти

```
int* x = (int*)malloc(3);
```



Program break

current_location

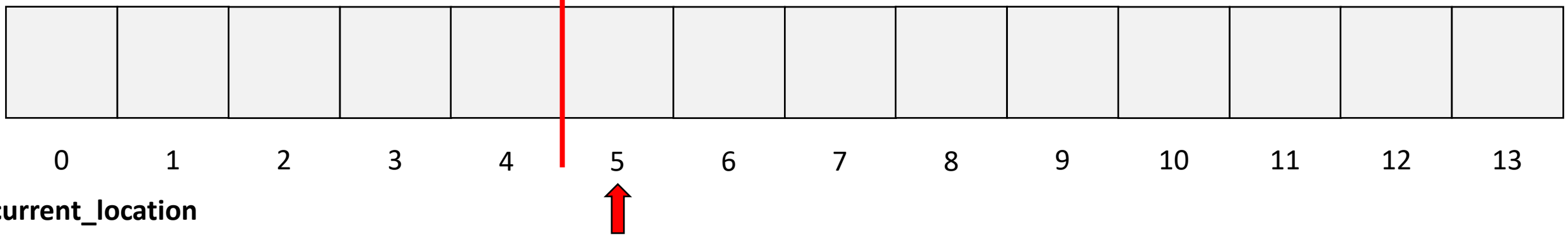
managed_memory_start

last_valid_address

`last_valid_address = sbrk(0);`	`last_valid_address = 0`
`managed_memory_start = last_valid_address;`	`managed_memory_start = 0`
`current_location = managed_memory_start;`	`current_location = 0`
`numbytes = numbytes + sizeof(struct mem_control_block);`	`numbytes = 3 + 2 = 5`

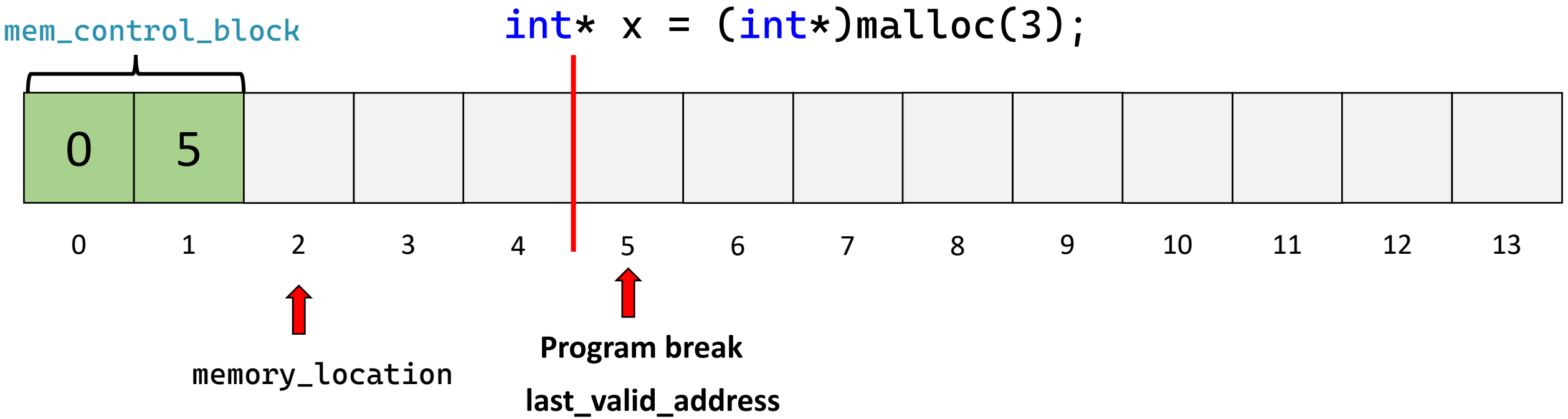
Аллокация памяти

```
int* x = (int*)malloc(3);
```



`sbrk(numbytes);`	`sbrk(5);`
`memory_location = last_valid_address;`	`memory_location = 0`
`last_valid_address = last_valid_address + numbytes;`	`last_valid_address = 0 + 5`
`current_location_mcb = memory_location;`	`current_location_mcb = 0`
`current_location_mcb->is_available = 0;`	`clm->is_available = 0`
`current_location_mcb->size = numbytes;`	`clm->size = 5`

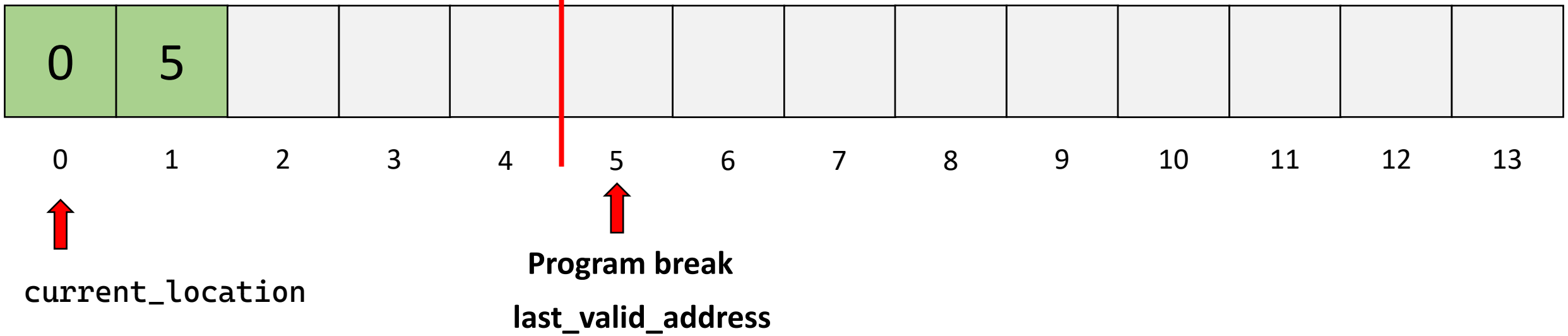
Аллокация памяти




```
memory_location = memory_location + sizeof(struct mem_control_block);

memory_location = 0 + 2 = 2;

return memory_location;
```

Аллокация памяти

```
int* y = (int*)malloc(5);
```

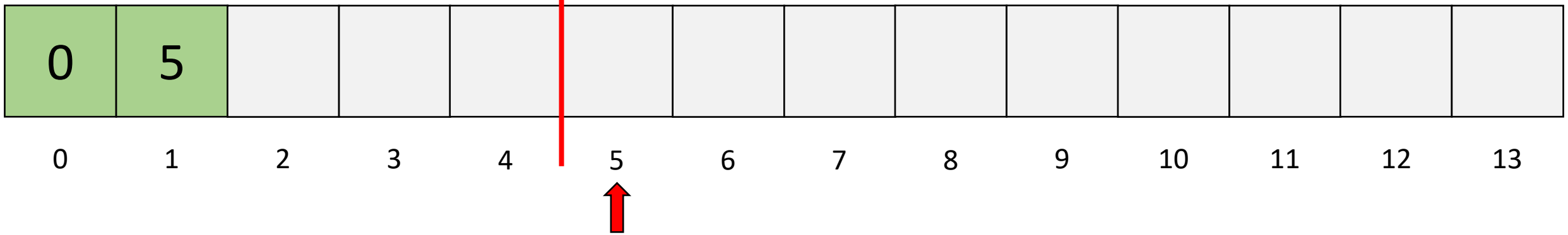



```
while (current_location != last_valid_address){
        if (current_location_mcb->is_available) {//выделяем память}
        current_location = current_location + current_location_mcb->size;
}
```

Ячейка занята, поэтому двигаемся дальше `current_location = 0 + 5 = 5`

Аллокация памяти

```
int* y = (int*)malloc(5);
```



Program break

last_valid_address

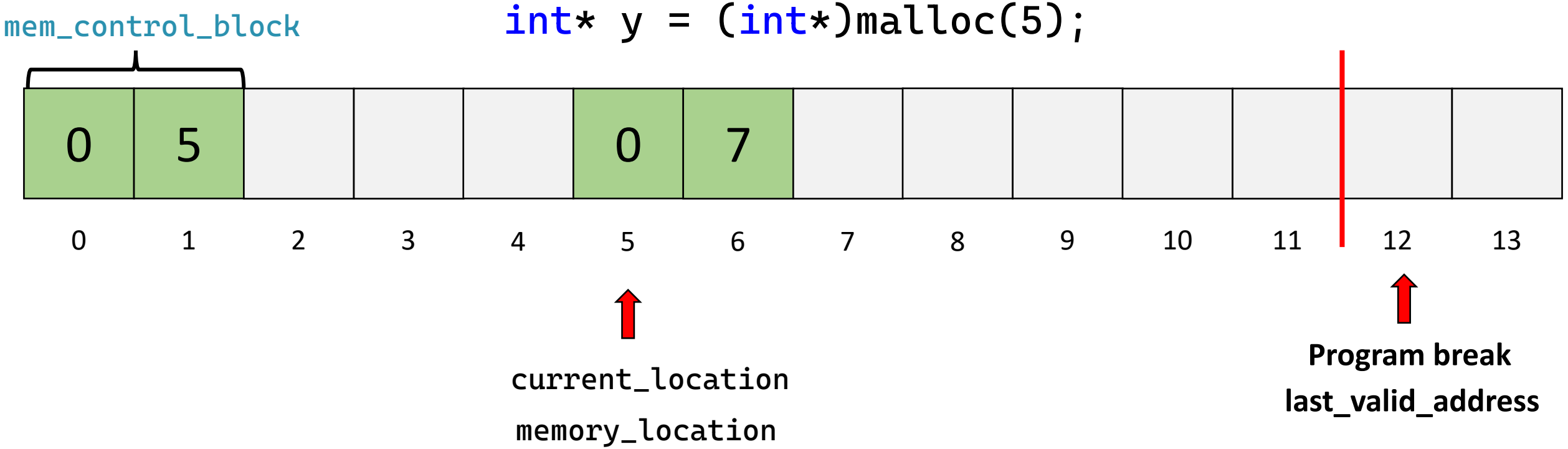
`current_location`

`memory_location`

Цикл закончился -> снова необходимо просить памяти

Аллокация памяти

```
int* y = (int*)malloc(5);
```

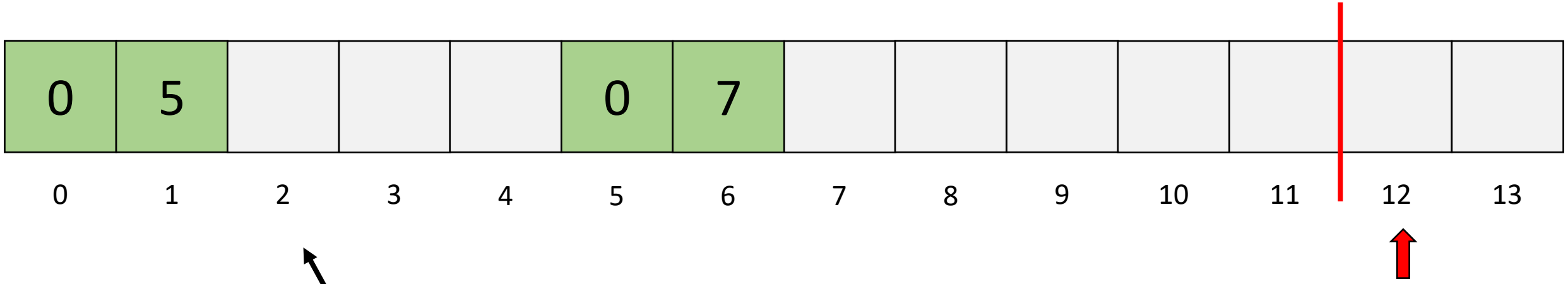



```
memory_location = memory_location + sizeof(struct mem_control_block);
memory_location = 5 + 2 = 7;
```

```
return memory_location;
```

Аллокация памяти

```
free(x);
```

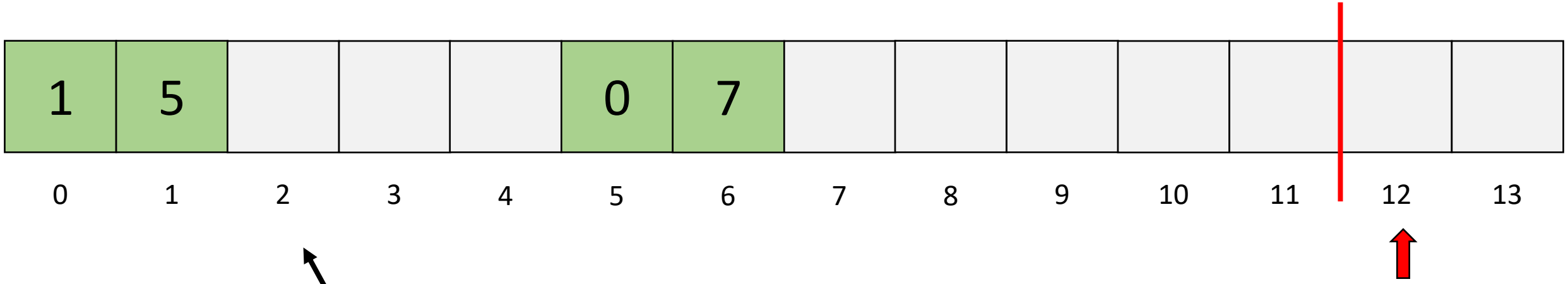


Адрес начала выделенной памяти

```
mcb = x - sizeof(struct mem_control_block);
mcb->is_available = 1;
```

Аллокация памяти

```
free(x);
```

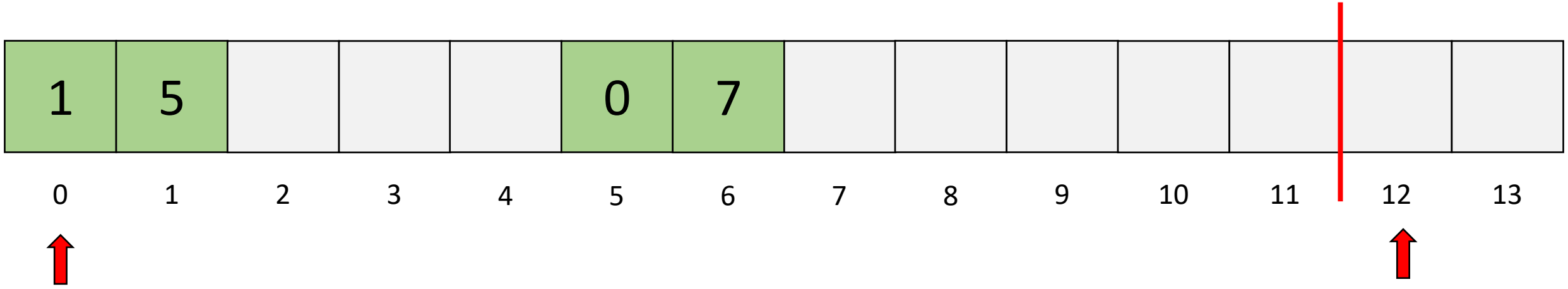


Адрес начала выделенной памяти

```
mcb = x - sizeof(struct mem_control_block);
mcb->is_available = 1;
```

Аллокация памяти

```
int* y = (int*)malloc(2);
```

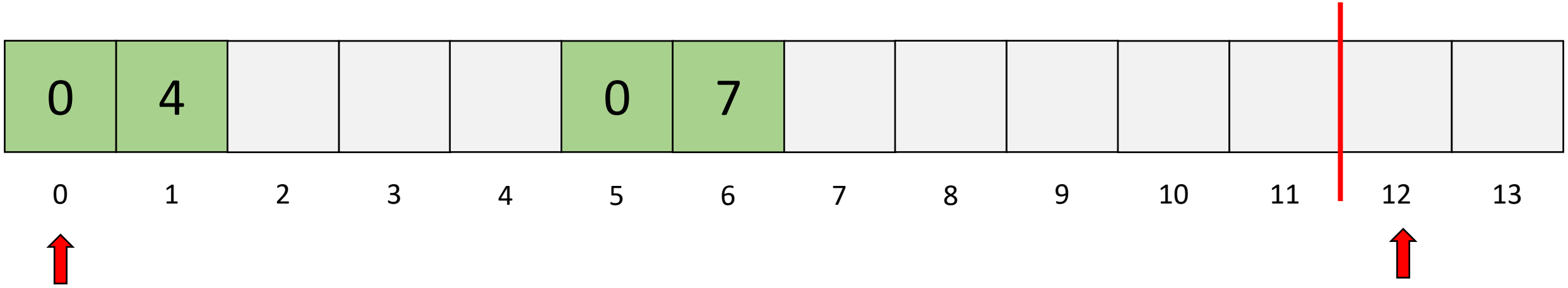



```
while (current_location != last_valid_address){
        if (current_location_mcb->is_available) {//выделяем память}
        current_location = current_location + current_location_mcb->size;
}
```

Ячейка свободна, поэтому используем её

Аллокация памяти

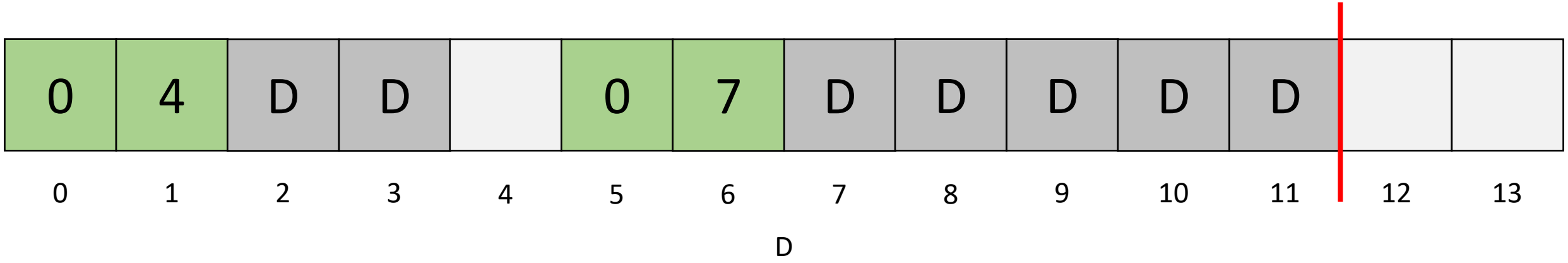
```
int* y = (int*)malloc(2);
```




```
while (current_location != last_valid_address){
        if (current_location_mcb->is_available) {//выделяем память}
        current_location = current_location + current_location_mcb->size;
}
```

Ячейка свободна, поэтому используем её

Недостатки реализации



1. Неэффективное использование памяти:
    1.1. Пустые ячейки (например, 4)
    1.2. Положение структур невозможно изменить
2. Медленная работа

Перечисления

```
enum Numbers
{
    Zero,
    One,
    Two,
    Three,
    Four
};
```

Перечисление — это пользовательский тип, состоящий из набора целочисленных констант, называемых перечислителями.

Перечисления

```
enum Numbers
{
    Zero,
    One,
    Two,
    Three,
    Four
};
```

Каждый параметр в *enumerator-list* присваивает имя значению набора перечисления. По умолчанию первый параметр *enumeration-constant* связан со значением 0. Следующий параметр *enumeration-constant* в списке связывается со значением (*constant-expression* + 1), если явно не указано другое значение. Имя параметра *enumeration-constant* эквивалентно его значению.

Inline функции

```c
#include <stdio.h>

int summ(int a, int b)
{
    return a + b;
}

int main(void)
{
    int x = summ(5, 5);
    return 0;
}
```

```
0000000000001129 <summ>:
    1129:       f3 0f 1e fa             endbr64
    112d:       55                      push   %rbp
    112e:       48 89 e5                mov    %rsp,%rbp
    1131:       89 7d fc                mov    %edi,-0x4(%rbp)
    1134:       89 75 f8                mov    %esi,-0x8(%rbp)
    1137:       8b 55 fc                mov    -0x4(%rbp),%edx
    113a:       8b 45 f8                mov    -0x8(%rbp),%eax
    113d:       01 d0                   add    %edx,%eax
    113f:       5d                      pop    %rbp
    1140:       c3                      retq

0000000000001141 <main>:
    1141:       f3 0f 1e fa             endbr64
    1145:       55                      push   %rbp
    1146:       48 89 e5                mov    %rsp,%rbp
    1149:       48 83 ec 10             sub    $0x10,%rsp
    114d:       be 05 00 00 00          mov    $0x5,%esi
    1152:       bf 05 00 00 00          mov    $0x5,%edi
    1157:       e8 cd ff ff ff          callq  1129 <summ>
    115c:       89 45 fc                mov    %eax,-0x4(%rbp)
    115f:       b8 00 00 00 00          mov    $0x0,%eax
    1164:       c9                      leaveq
    1165:       c3                      retq
    1166:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
    116d:       00 00 00
```

Inline функции

```c
#include <stdio.h>

__attribute__((always_inline))
inline int summ(int a, int b)
{
    return a + b;
}

int main(void)
{
    int x = summ(5, 5);
    return 0;
}
```

```
0000000000001129 <main>:
    1129:       f3 0f 1e fa             endbr64
    112d:       55                      push   %rbp
    112e:       48 89 e5                mov    %rsp,%rbp
    1131:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%rbp)
    1138:       c7 45 fc 05 00 00 00    movl   $0x5,-0x4(%rbp)
    113f:       8b 55 f8                mov    -0x8(%rbp),%edx
    1142:       8b 45 fc                mov    -0x4(%rbp),%eax
    1145:       01 d0                   add    %edx,%eax
    1147:       89 45 f4                mov    %eax,-0xc(%rbp)
    114a:       b8 00 00 00 00          mov    $0x0,%eax
    114f:       5d                      pop    %rbp
    1150:       c3                      retq
    1151:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
    1158:       00 00 00
    115b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

Ещё пару слов о разбиении на файлы

main.c

```c
#include <stdio.h>
#include "lib.h"

int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

lib.c

```c
int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.h

```c
#pragma once
int summ(int a, int b);
```

Как в main.c увидеть a из lib.c ?

Ещё пару слов о разбиении на файлы

main.c

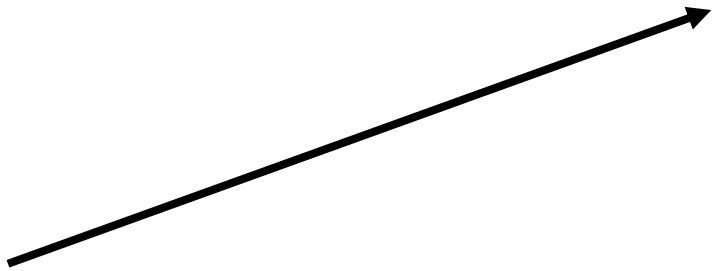
```c
#include <stdio.h>
#include "lib.h"

int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

lib.c

```c
int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.h

```c
#pragma once
int summ(int a, int b);
extern int a;
```



Ключевое слово extern позволяет объявить переменную, но не определить

Ещё пару слов о разбиении на файлы

main.c

```c
#include <stdio.h>
#include "lib.h"

int a = 20;
int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

lib.c

```c
int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.h

```c
#pragma once
int summ(int a, int b);
```

Как обойти двойное определение?

Статические глобальные переменные

main.c

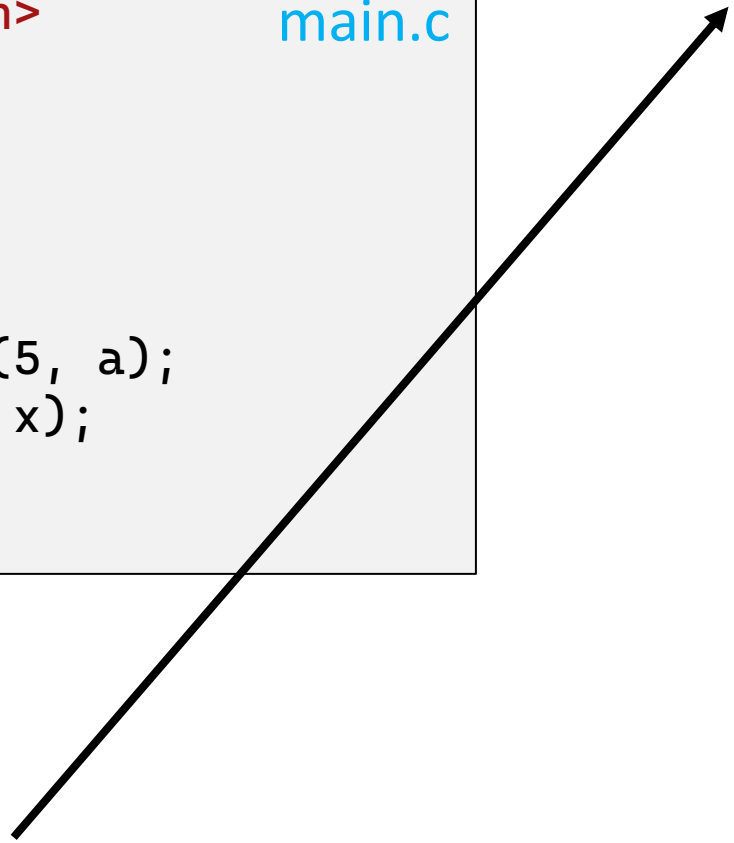
```c
#include <stdio.h>
#include "lib.h"

int a = 20;
int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

lib.c

```c
static int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.h

```c
#pragma once
int summ(int a, int b);
```



Ключевое слово static позволяет сузить область видимости переменной до файла

Статические локальные переменные

```c
#include <stdio.h>

int summ(int a)
{
    int b = 5;
    b++;
    return a + b;
}

int main(void)
{
    int x = summ(5);
    printf("%d\n", x);
    x = summ(5);
    printf("%d\n", x);
    return 0;
}
```

```
X1 = 11
X2 = 11
```

Статические локальные переменные

```c
#include <stdio.h>

int summ(int a)
{
    static int b = 5;
    b++;
    return a + b;
}

int main(void)
{
    int x = summ(5);
    printf("%d\n", x);
    x = summ(5);
    printf("%d\n", x);
    return 0;
}
```

```
X1 = 11
X2 = 12
```

Спасибо за внимание!