Практикум на ЭВМ
Динамические многомерные массивы.
Динамические структуры.

Баев А.Ж.

Казахстанский филиал МГУ

06 октября 2018

Статические матрицы

Массив `int a[2]` — указатель типа `int *`.
Матрица типа `int [2][3][4]` — указатель типа `int (*)[2][3]`.
Почему именно первая размерность?

Следующие прототипы одинаковы:

```
void f(int array[2][3][4]);
void f(int (*array)[3][4]);
```

Статические матрицы

Обратите внимание на круглые скобки.

```
int (*array)[3][4]
```

```
int *array[3][4]
```

Статические матрицы — плотно упакованные

```
void *memset(void *s, int c, size_t n);
void *memcpy(void *dest, void *src, size_t n);
```

Инициализация всей матрицы.

```
#include <string.h>

const int n = 10, m = 20;

int A[n][m];
memset(A, 0, n * m * sizeof(int));

int B[n][m];
memcpy(B, A, n * m * sizeof(int));

int C[n * m];
memcpy(C, A, n * m * sizeof(int));
```

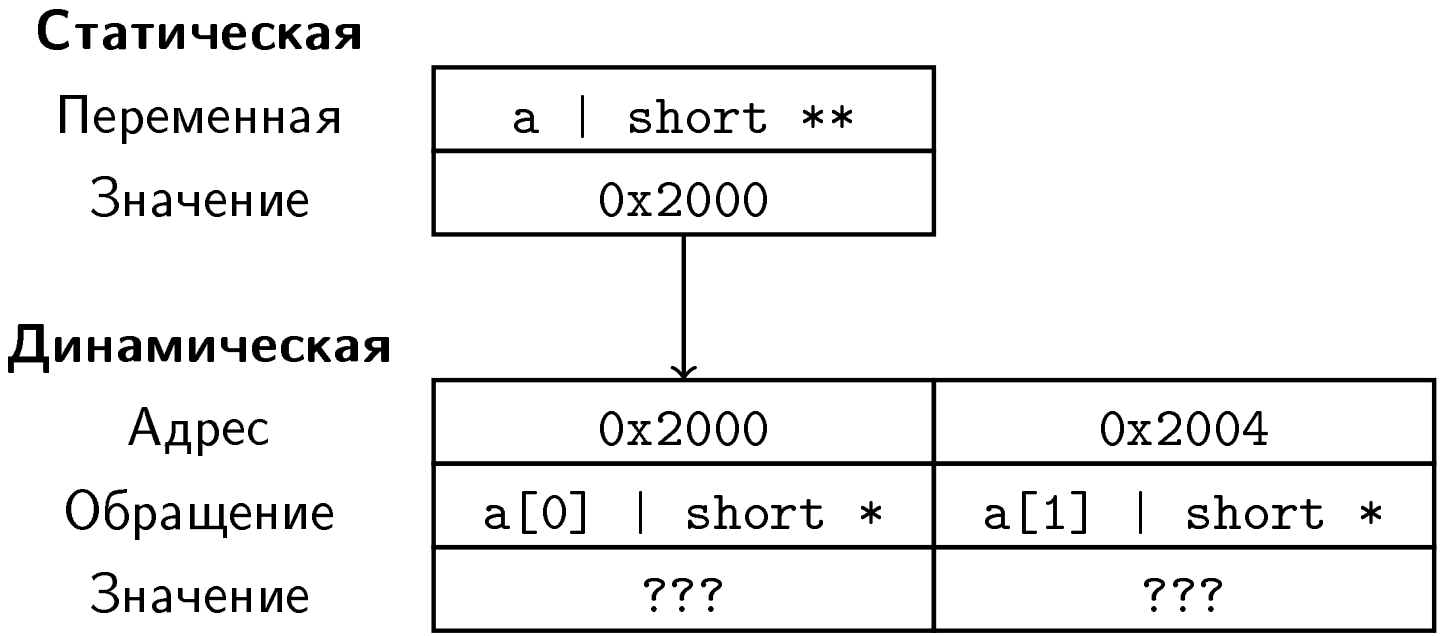
Динамические матрицы

1) уложить всю матрицу построчно в виде одного динамического массива (плотная упаковка);
2) создать дополнительный массив указателей на динамические массивы (неплотная упаковка).
Смотрим второе. Почему?

Динамические матрицы. Матрица 2x3

Каждая строка матрицы — динамический массив. Где хранить все эти указатели? Еще в одном динамическом массиве!

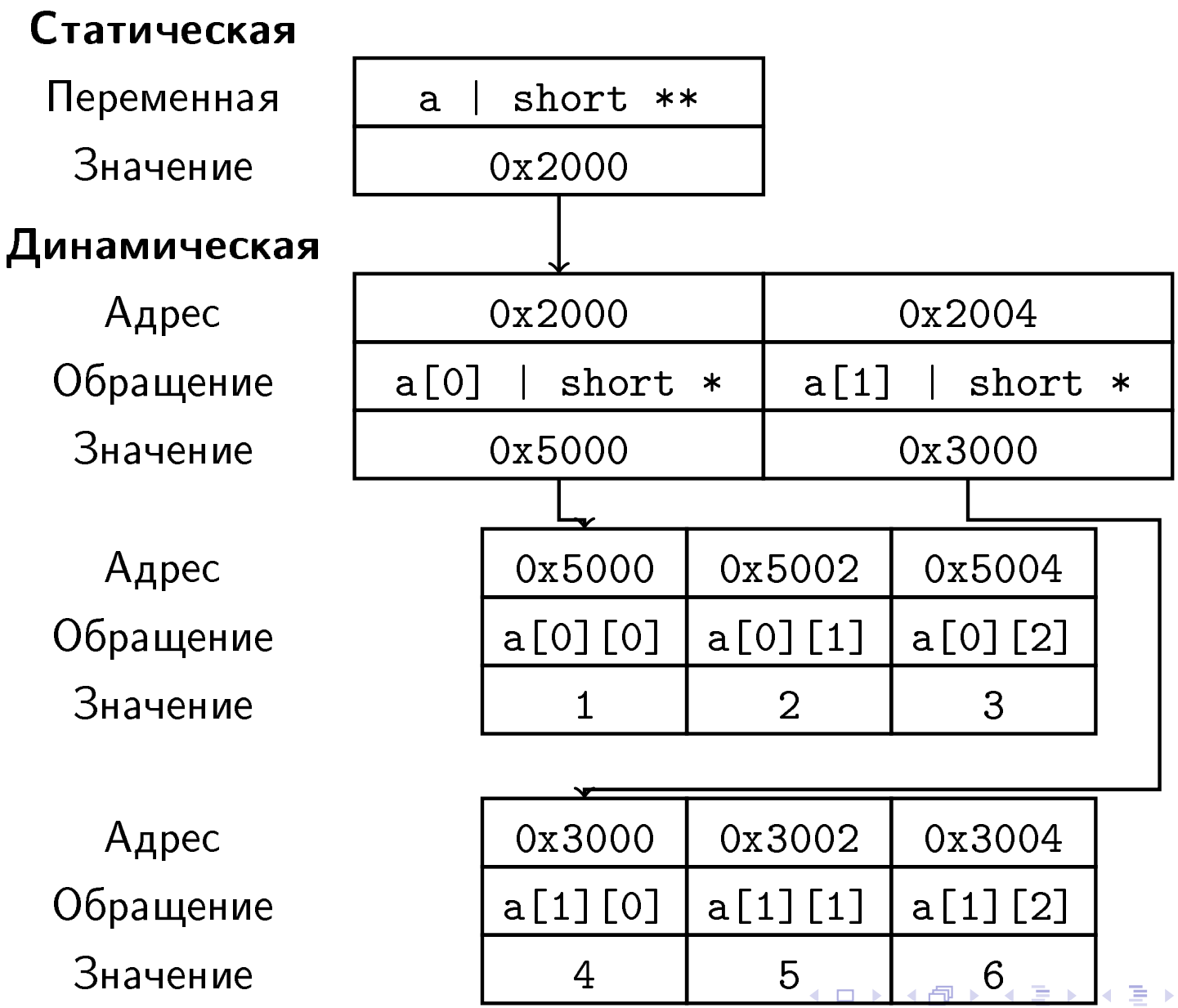
```
int **a = malloc(2 * sizeof(int *));
```



Динамические матрицы. Матрица 2x3

```
    a[0] = malloc(3 * sizeof(int));
    a[1] = malloc(3 * sizeof(int));
```

Динамические матрицы. Матрица 2x3



Динамические матрицы. Матрица 2x3

Очистка

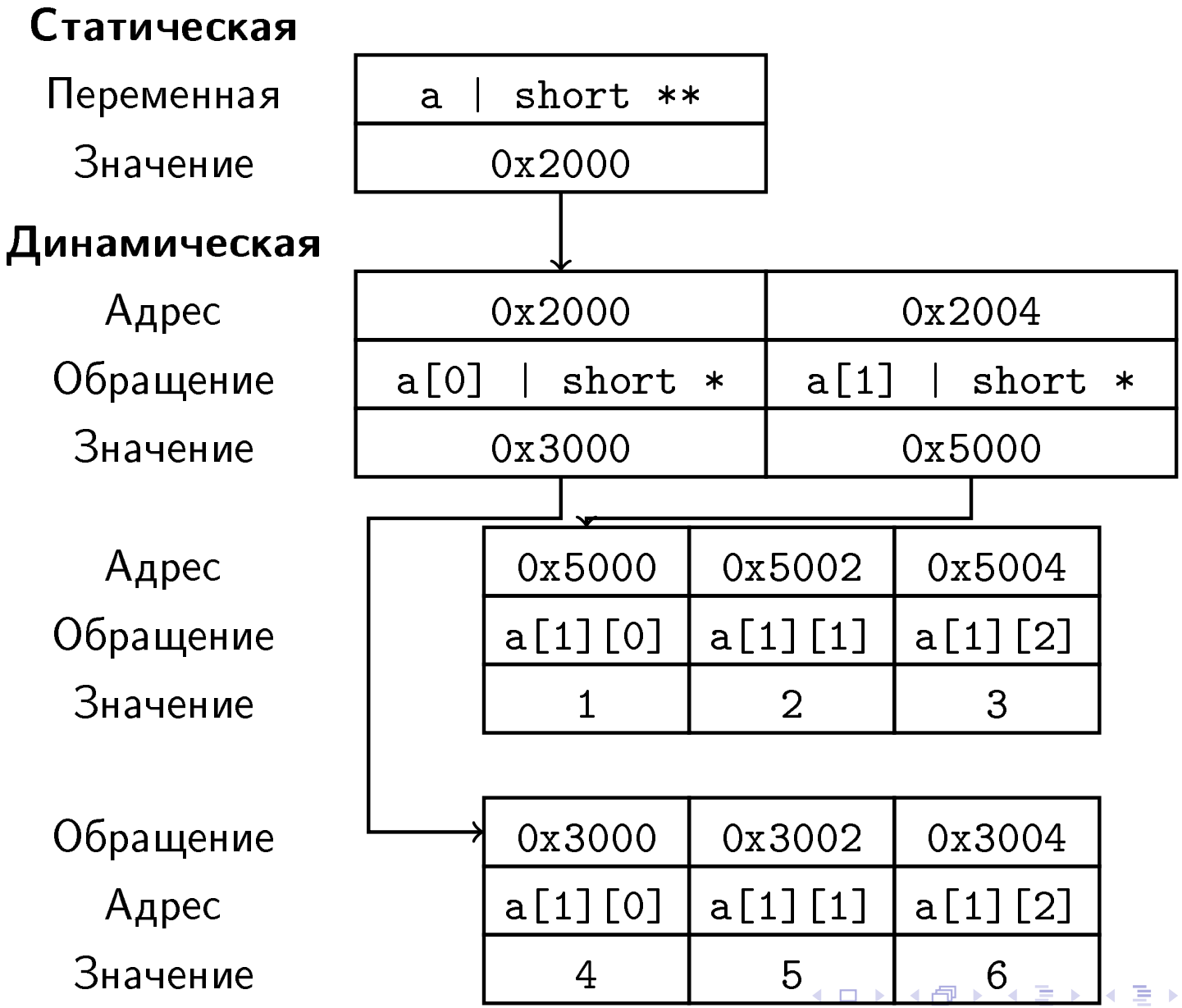
```
free(a[0]);
free(a[1]);
free(a);
```

Динамические матрицы. Любимая плюшка линала

Перестановка строк!

```
short *tmp = a[0];
a[0] = a[1];
a[1] = tmp;
```

Динамические матрицы. Матрица 2x3



Отличия динамической и статической матрицы.

1. матрица размера $n \times m$, где каждый элемент имеет размер s, а указатель — размер p. В случае со статической памятью: $n \cdot m \cdot s$. В случае с динамической памятью: $(1 + n) \cdot p$ для указателей и $n \cdot m \cdot s$ для самой матрицы.
2. динамические матрицы хранятся кусками по строкам, а статические — цельном блоком. Перестановка строк O(1).
3. Размеры разных строк могут быть разными.

Аргументы командной строки.

```
$ gcc prog.c -o prog -Wall
```

Здесь 4 аргумента:

```
int main(int argc, char** argv);
```

При запуске вашей программы, параметр argc будет содержать количество параметров командной строки, включая название программы (5).
Аргумент argv — это массив строк, которые сформированы из аргументов, включая название программы:

```
argv[0] == "gcc"
argv[1] == "prog.c"
argv[2] == "-o"
argv[3] == "prog"
argv[4] == "Wall"
```

Головоломки.

0. Какие присваивания допустимы между указанными переменными (среди всех возможных пар)?

```
int a[2][3];
int (*b)[3];
int *c[3];
int **d;
```

1. Динамическая матрица из 3×4. Строки и столбы — динамические.
2. Динамическая матрица из 3×4. Строки — статические, столбы — динамические.
3. Динамическая матрица из 3×4. Строки — динамические, столбы — статические.
4. Написать программу, которая печатает свой исходный код, при условии, что он лежит рядом и имеет такое же название (prog и prog.c).
5. Написать программу, которая печатает свой бинарный код.

Указатель на структуру.

Оператор звезда и точка или оператор стрелка.

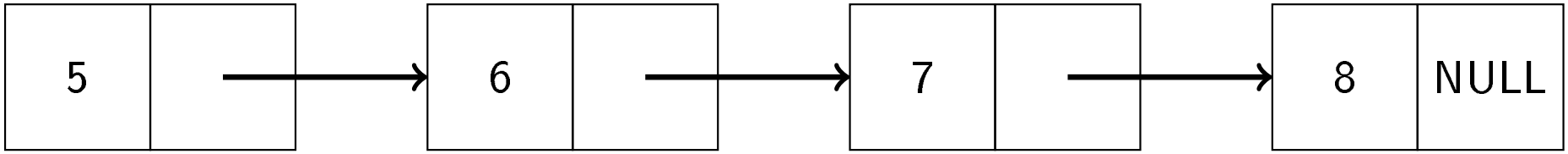
```
struct Point {
    int x, y;
};
...
    struct Point point = {2, 3};
    struct Point *ptr = &point;
    (*ptr).x = 5;
    ptr->y = 6;
```

Указатель на структуру.

Оператор звезда и точка или оператор стрелка.

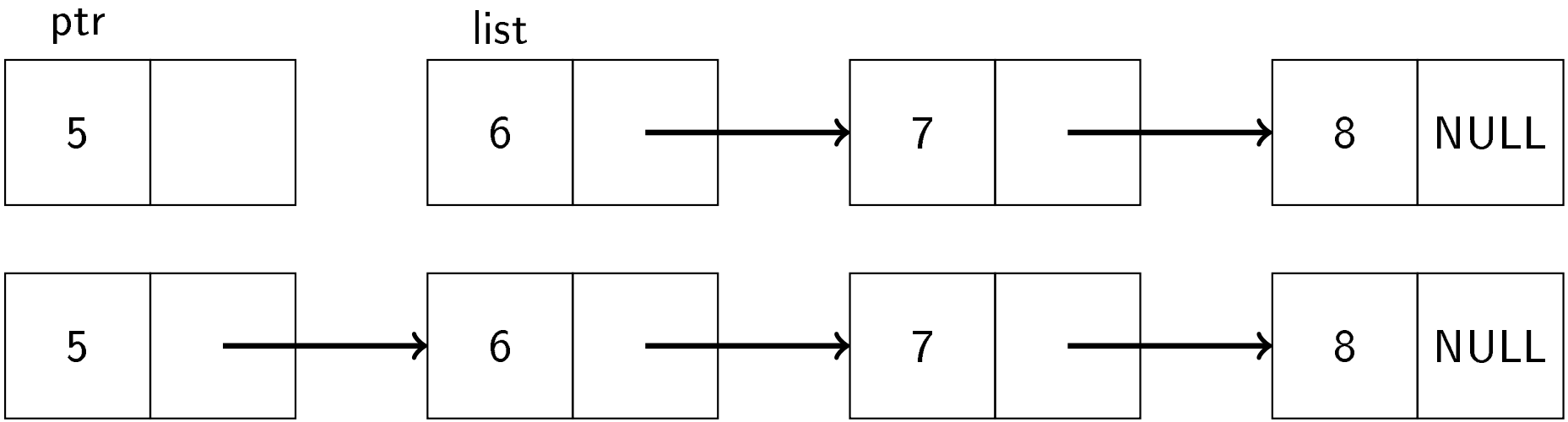
```
struct Point {
    int x, y;
};
...
    struct Point point = malloc(sizeof(struct Point));
    (*ptr).x = 5;
    ptr->y = 6;
```

Динамическая структура — список.



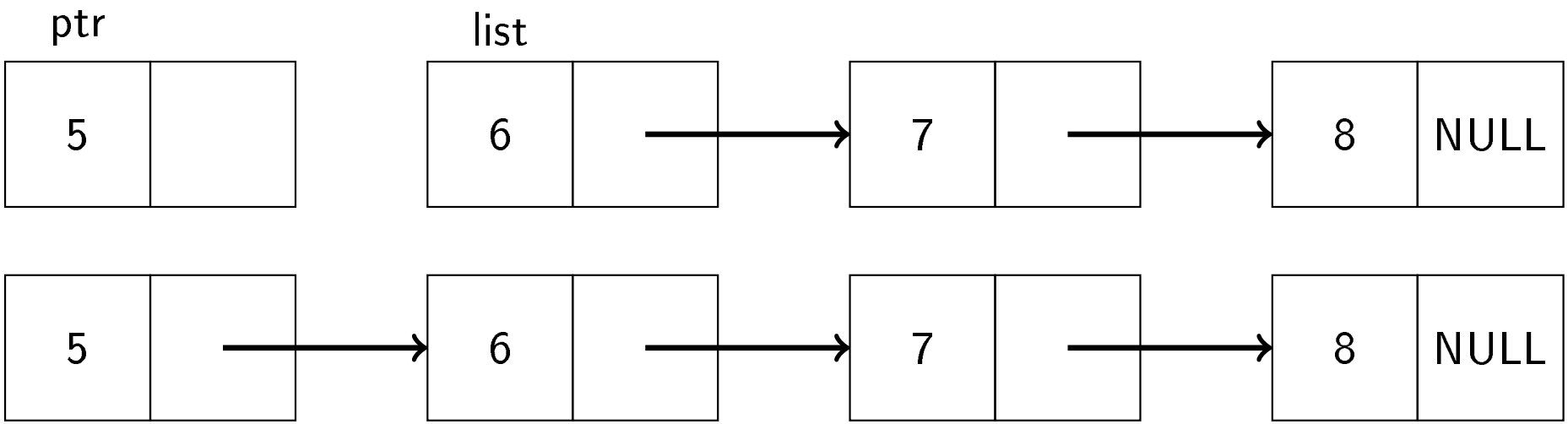

```
typedef struct node *link;

struct node {
    int elem;
    link next;
};

int main() {
    link list = NULL;
    list = push_front(list, 6);
    list = push_front(list, 5);
    list = push_back(list, 7);
    list = push_back(list, 8);
    return 0;
}
```

Список. Добавить в начало.



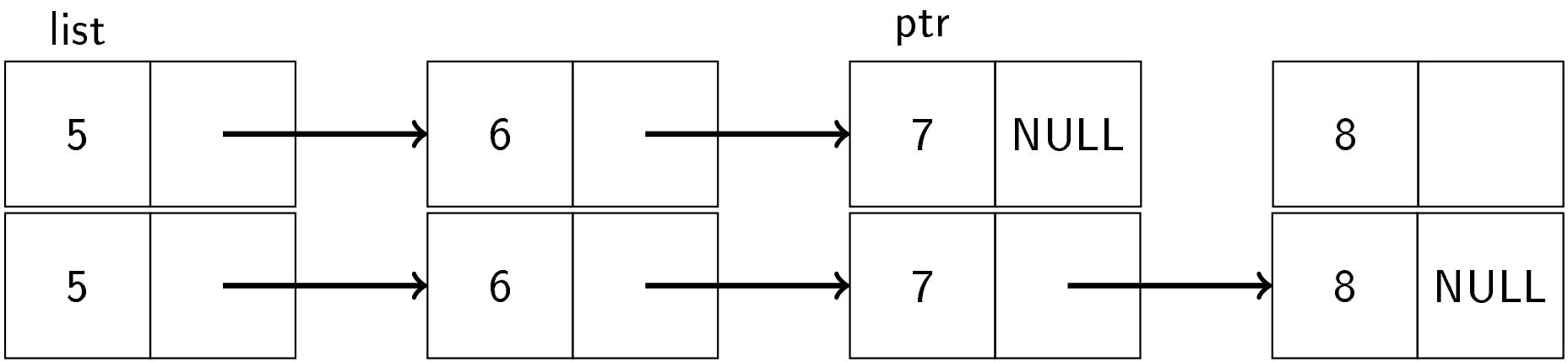

```
link push_front(link list, int elem) {
    link ptr = malloc(sizeof(struct node));
    ptr->elem = elem;
    ptr->next = list;
    return ptr;
}
```

Список. Типичная ошибка.



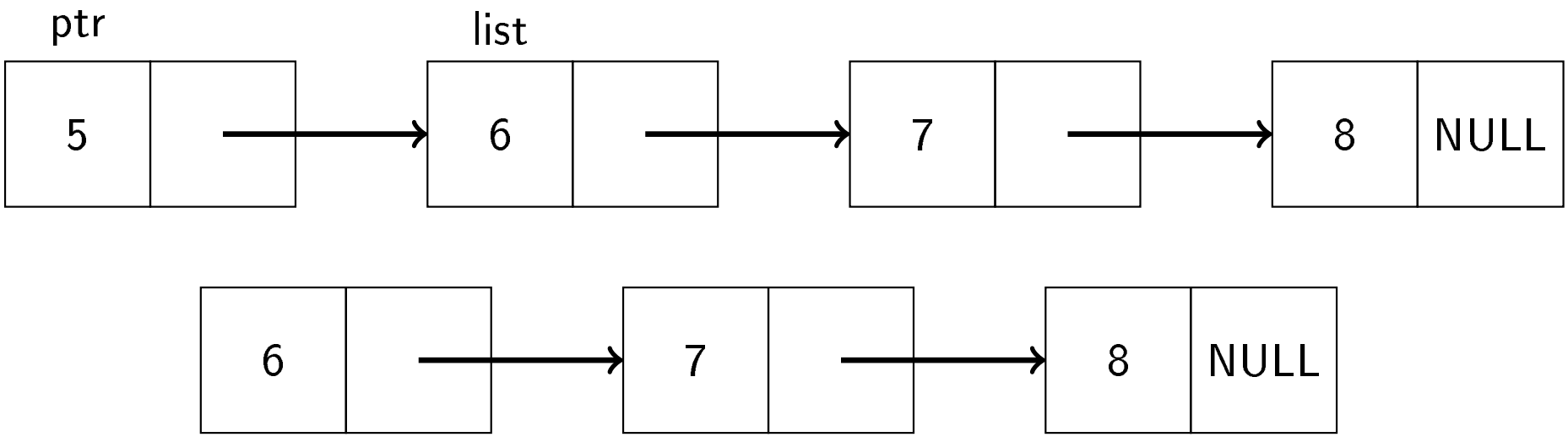

```
void push_front(link list, int elem) {
    link ptr = malloc(sizeof(struct node));
    ptr->elem = elem;
    ptr->next = list;
    list = ptr;
}
```

Список. Добавить в конец.



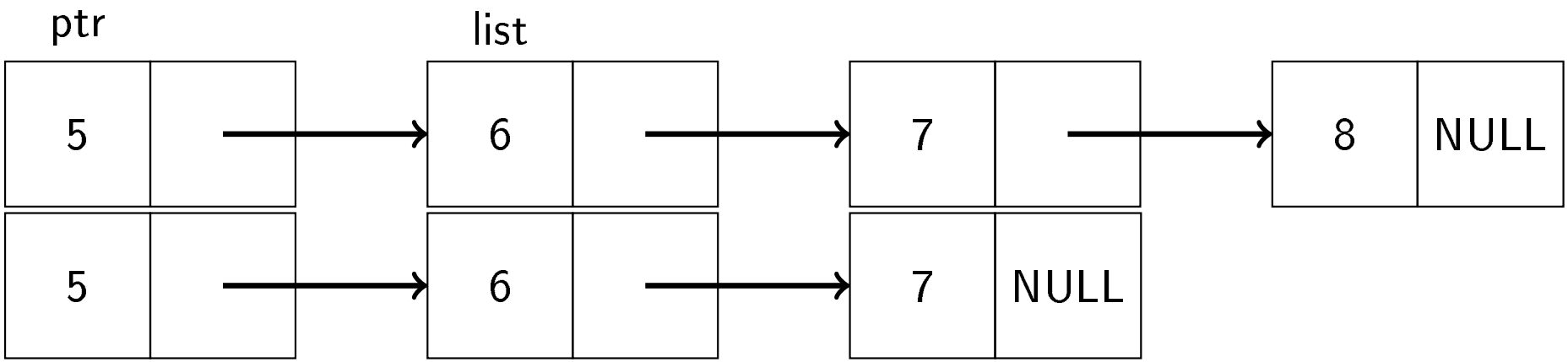

```
link push_back(link list, int elem) {
    if (list == NULL) {
        return push_front(list, elem);
    }
    link ptr = list;
    while (ptr -> next != NULL) {
        ptr = ptr -> next;
    }
    ptr->next = push_front(NULL, elem);
    return list;
}
```

Список. Убрать из начала.




```
link pop_front(link ptr) {
    link list = ptr->next;
    free(ptr);
    return list;
}
```

Список. Убрать из конца.




```
link pop_back(link list) {
    if (list -> next == NULL) {
        return pop_front(list);
    }
    link ptr = list;
    while (ptr -> next -> next != NULL) {
        ptr = ptr -> next;
    }
    ptr->next = pop_front(ptr->next);
    return ptr;
}
```

Список. Вывод. Очистка.

```
void print(link list) {
    putchar('[');
    while (list != NULL) {
        printf("%d␣", list->elem);
        list = list -> next;
    }
    puts("]");
}
```

```
link clear(link list) {
    link ptr;
    while (list != NULL) {
        ptr = list -> next;
        free(list);
        list = ptr;
    }
    return NULL;
}
```

Список. Генерируем список.

```
link range(int from, int to) {
    link list = NULL;
    int elem = to - 1;
    while (elem >= from) {
        list = push_front(list, elem);
        elem--;
    }
    return list;
}
```

Отладка в gdb

Компилируем

```
gcc prog.c -o prog -Wall -Werror -lm -g
```

Запускаем отладчик

```
gdb prog
```

Посмотрим код

```
list 1
```

Ставим точку останова (до которой программа будет выполняться в обычном режиме). Лучше ставить сразу после ввода.

```
break 6
```

Запускаем

```
run
```

Добавляем переменную наблюдения (можно несколько переменных)

```
display n
display ans
```

Делаем построчное выполнение (первый раз надо набрать команду целиком, потом просто Enter).

```
next
```

Выход

```
quit
```

Отладка в gdb

```
79      int main() {
80          link list = NULL;
81          list = push_front(list, 5);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb) break 80
Breakpoint 1 at 0x93e: file prog.c, line 80.
```

```
(gdb) run
Starting program: prog
Breakpoint 1, main () at prog.c:80
80          link list = NULL;
```

```
(gdb) display list
1: list = (link) 0x0
```

Отладка в gdb

```
79      int main() {
80          link list = NULL;
81          list = push_front(list, 5);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb) next
81          list = push_front(list, 5);
1: list = (link) 0x0
```

```
(gdb)
82          print(list);
1: list = (link) 0x555555756260
```

Отладка в gdb

```
79      int main() {
80          link list = NULL;
81          list = push_front(list, 5);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb)
[5 ]
83          list = clear(list);
1: list = (link) 0x555555756260
```

```
(gdb)
84          return 0;
1: list = (link) 0x0
```

Отладка в gdb. Без ошибок.

```
79      int main() {
80          link list = NULL;
81          list = push_front(list, 5);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb) break 80
Breakpoint 1 at 0x93e: file prog.c, line 80.
```

```
(gdb) run
Starting program: prog
Breakpoint 1, main () at prog.c:80
80          link list = NULL;
```

```
(gdb) display list
1: list = (link) 0x0
```

Отладка в gdb. Без ошибок.

```
79      int main() {
80          link list = NULL;
81          list = push_front(list, 5);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb) next
81          list = push_front(list, 5);
1: list = (link) 0x0
```

```
(gdb)
82          print(list);
1: list = (link) 0x555555756260
```

Отладка в gdb. Без ошибок.

```
79      int main() {
80          link list = NULL;
81          list = push_front(list, 5);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb)
[5 ]
83          list = clear(list);
1: list = (link) 0x555555756260
```

```
(gdb)
84          return 0;
1: list = (link) 0x0
```

Отладка в gdb. Ошибки.

```
79      int main() {
80          link list = NULL;
81          list = pop_front(list);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb) break 80
Breakpoint 1 at 0x93e: file prog.c, line 80.
```

```
(gdb) run
Starting program: prog
Breakpoint 1, main () at prog.c:80
80          link list = NULL;
```

```
(gdb) display list
1: list = (link) 0x0
```

Отладка в gdb. Ошибки.

```
79      int main() {
80          link list = NULL;
81          list = pop_front(list);
82          print(list);
83          list = clear(list);
84          return 0;
85      }
```

```
(gdb) next
81          list = pop_front(list);
1: list = (link) 0x0
```

```
(gdb)
Program received signal SIGSEGV, Segmentation fault.
0x0000555555554762 in pop_front (ptr=0x0) at prog.c:19
19          link list = ptr->next;
```

Санитайзер

```
gcc prog.c -o prog -fsanitize=address
```

```
==11053==ERROR: AddressSanitizer: SEGV on unknown
    address 0x000000000008 (pc 0x55db755ccbfb
    bp 0x7fff74e0dfc0 sp 0x7fff74e0dfa0 T0)
==11053==The signal is caused by a READ memory access.
==11053==Hint: address points to the zero page.
    #0 0x55db755ccbfa in pop_front ./prog.c:19
    #1 0x55db755ccf91 in main ./prog.c:81
    #2 0x7f5fc8a69b96 in __libc_start_main
        (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
    #3 0x55db755cca59 in _start
        (./prog+0xa59)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV ./prog.c:19 in pop_front
==11053==ABORTING
```