НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Алгоритмизация и программирование

Программирование на C/C++ и Kotlin

(ч.8 – динамические списки)

Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

Зачем нужны динамические списки?

- Мы можем создавать массивы нужного размера указав размер в квадратных скобках
- Всегда можно создать массив «с запасом»

Например, так:

```
char c[1000];
```

Или так:

```
int z[100000];
```

Зачем нужны динамические списки?

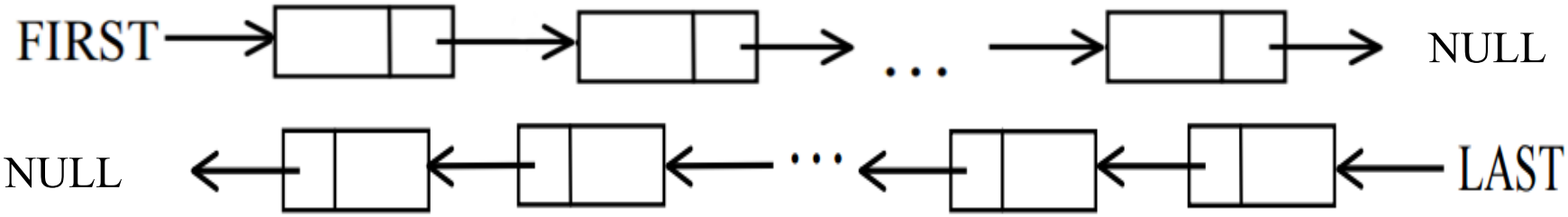
- Если один-два таких массива – не страшно, а если 1000?
- А ведь большая часть их будет не заполнена!

Что делать?

- Можно использовать динамическое распределение памяти.
- Например, можно использовать абстрактную структуру данных «линейный список»

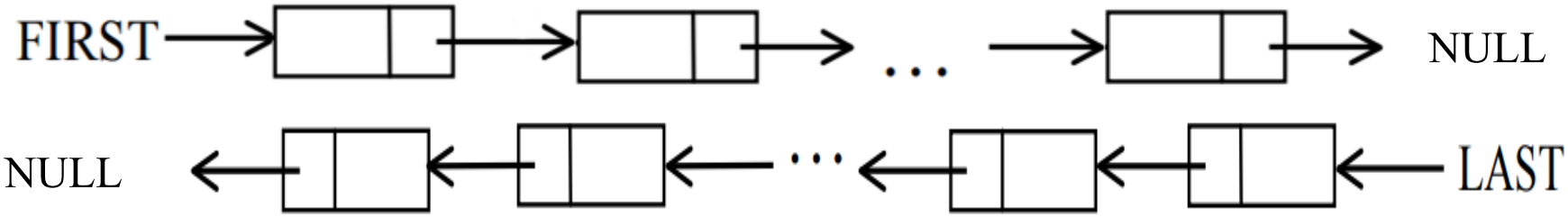
Что делать?

- Можно использовать динамическое распределение памяти.
- Например, можно использовать абстрактную структуру данных «линейный список»

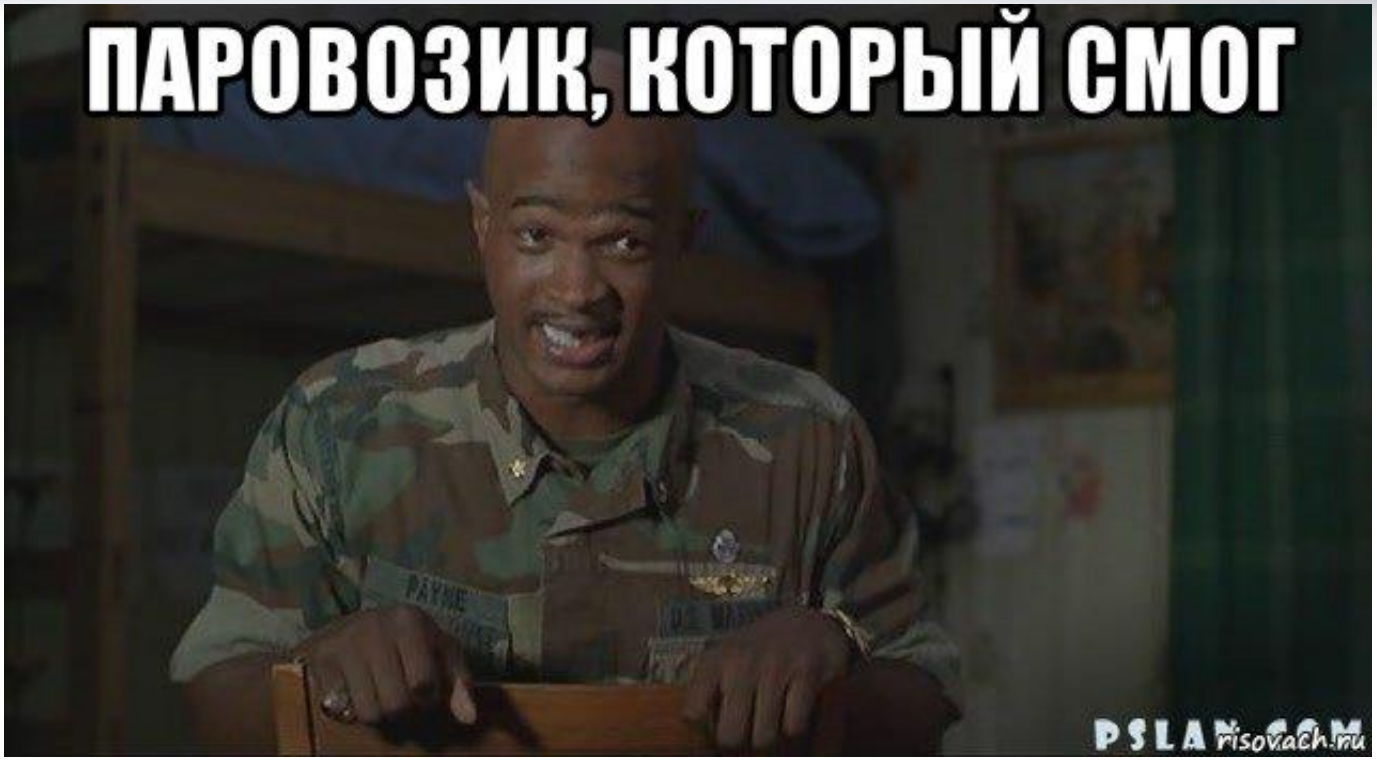


Что делать?

- Можно использовать динамическое распределение памяти.
- Например, можно использовать абстрактную структуру данных «линейный список»



На что это похоже?

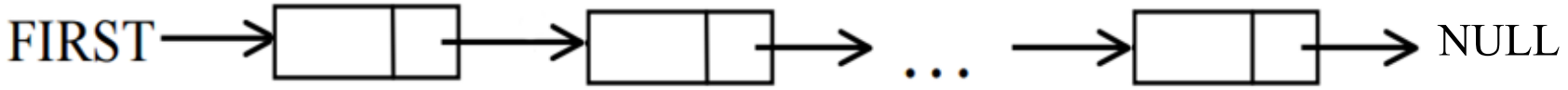


Здесь должна была находиться картинка с поездом.
Но эта мне нравится больше ☺

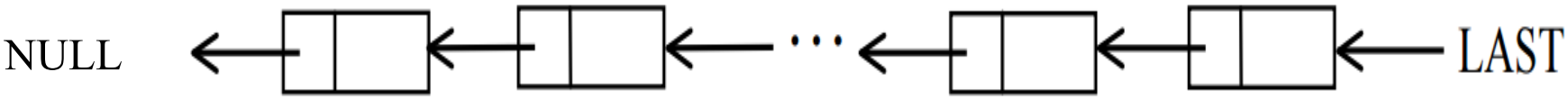
Линейный список

Разновидности линейного списка

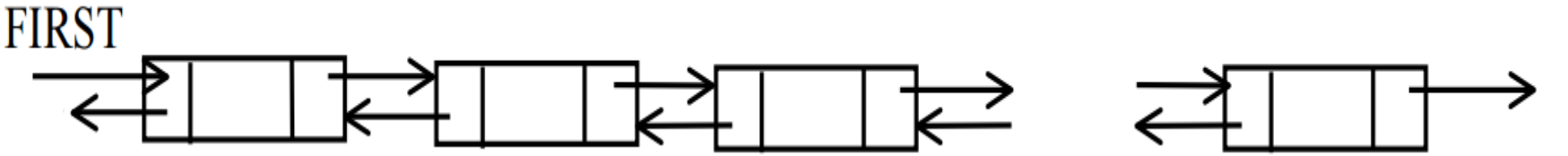
Каждый элемент содержит указатель на следующий



Каждый элемент содержит указатель на предыдущий



Двунаправленный список



Ну вы же понимаете. Да?

Что-то давно не было кода!

Опишем новый тип - структуру элемента списка:

```
struct element {
    int data;
    element * next;
};
```

Теперь, для построения линейного списка можно использовать переменную этого типа:

```
element * head = nullptr;
```

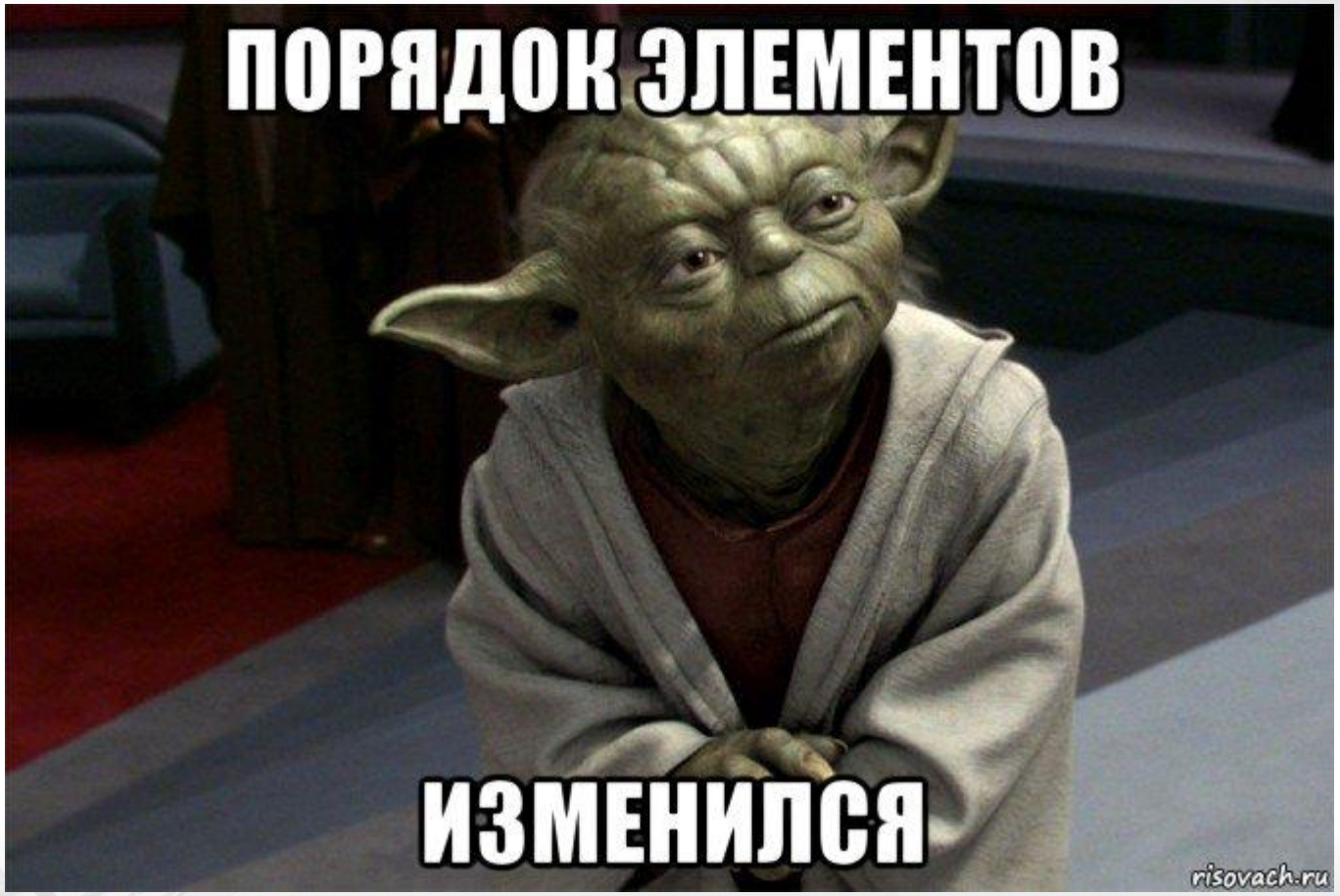

НУ ЭТО ВСЕ ТЕОРИЯ
ПОРА НАПИСАТЬ КОД

Стек

```
void push(int value) {
    auto * t = new element;
    t->data = value;
    t->next = head;
    head = t;
}

int pop() {
    element * t = head;
    head = head->next;
    int v = t->data;
    delete t;
    return v;
}

bool isEmpty() {
    return head == nullptr;
}
```


ПОРЯДОК ЭЛЕМЕНТОВ
ИЗМЕНИЛСЯ
risovach.ru

Виды списков

- **Стек — структура данных**, представляющая собой **список элементов**, организованных по принципу ***LIFO*** (*last in — first out*, «последним пришёл — первым вышел»).

Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Виды списков

- **Очередь** — **структура данных**, представляющая собой **список элементов** , организованных по принципу ***FIFO*** (first in — first out) «первый пришёл — первый вышел».

Добавление элемента возможно лишь в конец очереди, выборка — только из начала очереди, при этом выбранный элемент из очереди удаляется.

Не пора ли написать код?

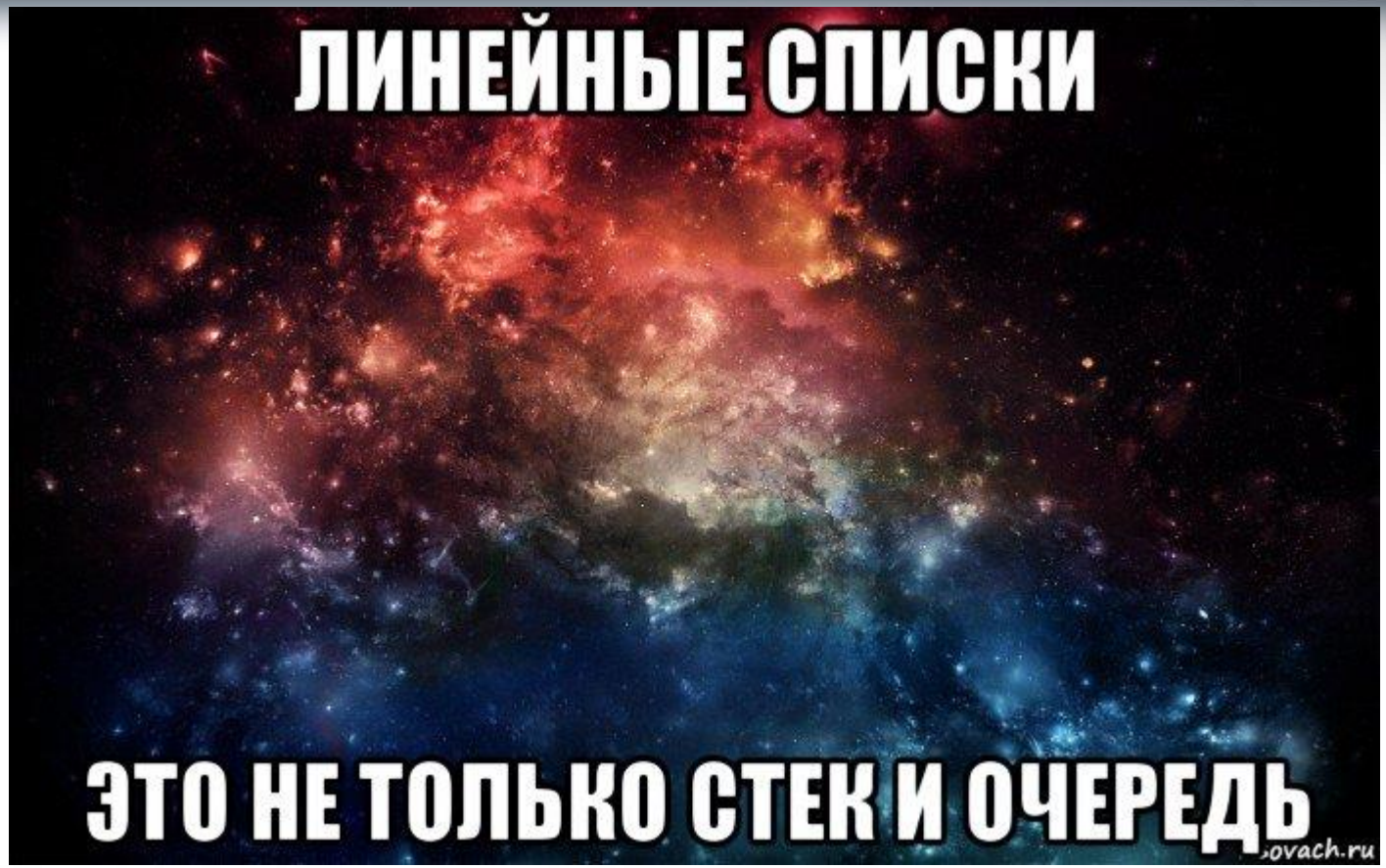
Очередь

```
int pull() {
    element * t = head;
    head = head->next;
    int v = t->data;
    delete t;
    return v;
}

void offer(int value) {
    auto * t = new element;
    t->data = value;
    t->next = nullptr;
    if (head == nullptr) {
        head = t;
    } else
        tail->next = t;
    tail = t;
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Линейные списки

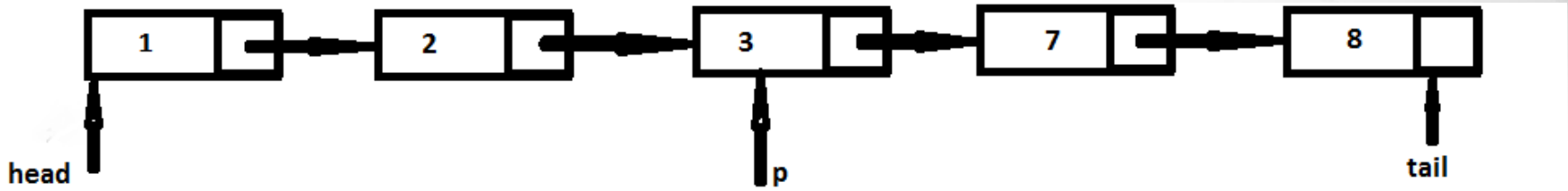


А про такой список не забыли?

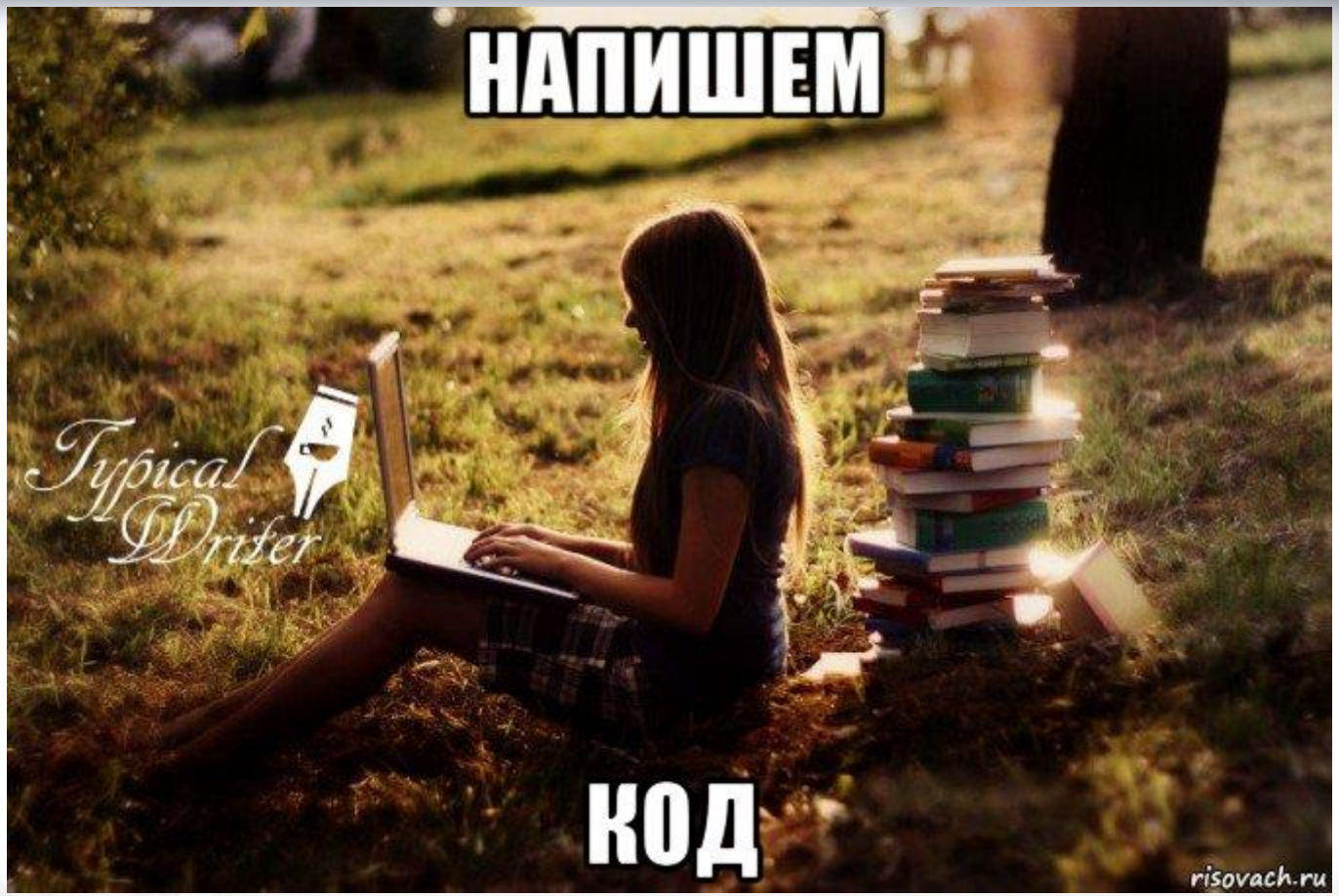
Базовые задачи на списках

1. Найти заданный элемент (указатель на него)
2. Добавить элемент после заданного
3. Добавить элемент перед заданным
4. Удалить заданный элемент
5. Удалить элемент после заданного

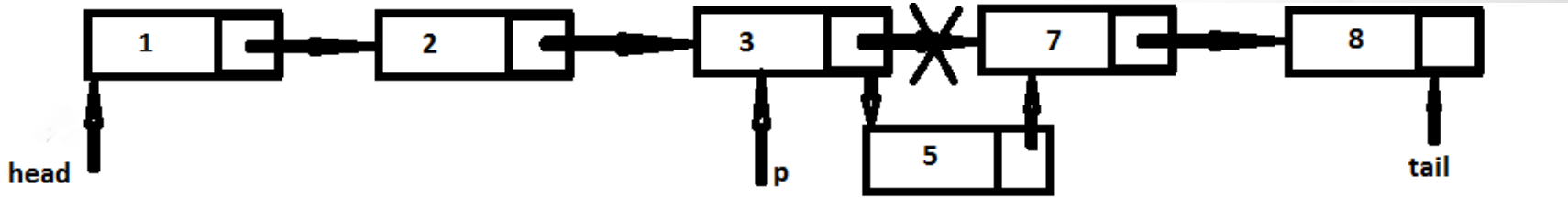
1. Найти заданный элемент (указатель на него)



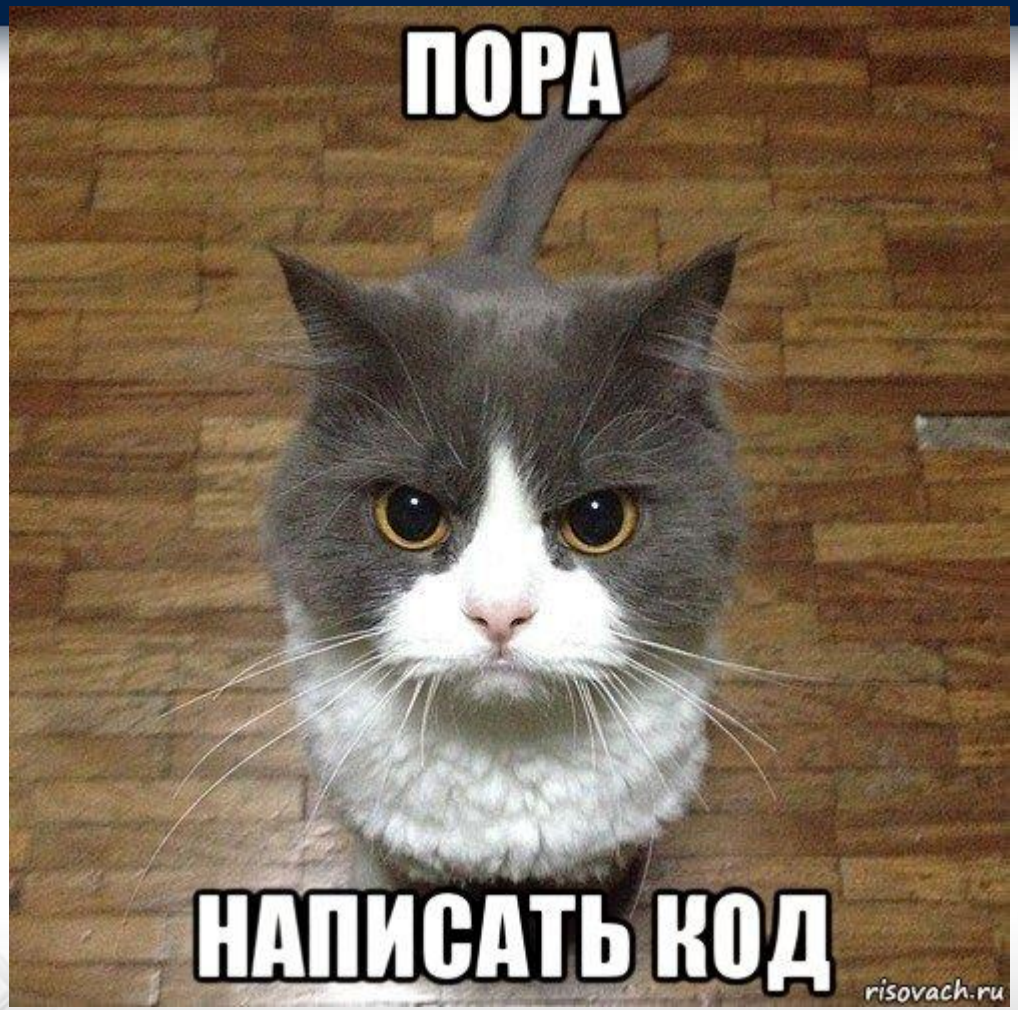

```
element * find(element * head, int value) {
    element * t;
    t = head;
    while (t) {
        if (t->data == value) {
            break;
        } else {
            t = t->next;
        }
    }
    return t;
}
```


НАПИШЕМ
Typical Writer
КОД
risovach.ru

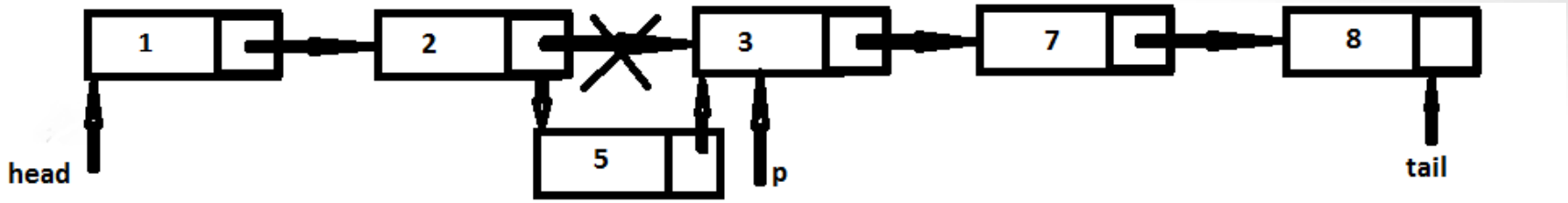
2. Добавить элемент после заданного




```
void addAfter(element * f, int value) {
    //1
    element * t = new element;
    t->data = value;
    //2
    t->next = f->next;
    //3
    f->next = t;
}
```


ПОРА
НАПИСАТЬ КОД
risovach.ru

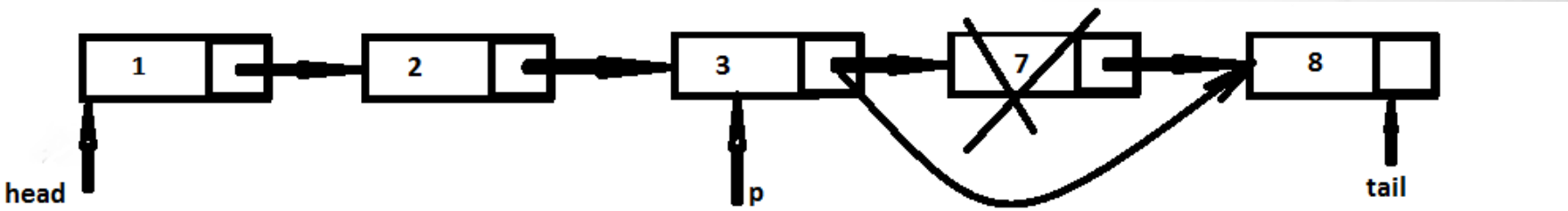
3. Добавить элемент перед заданным



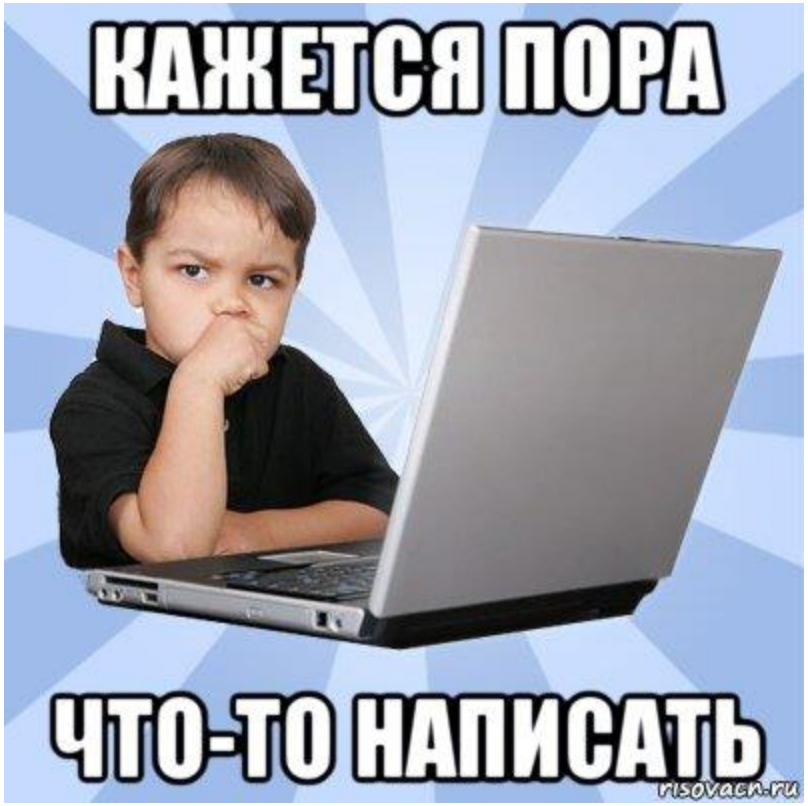

```
void addBefore(element * f, int value) {
    //1
    element * t = new element;
    //2
    t->next = f->next;
    //3
    f->next = t;
    //4
    t->data = f->data;
    f->data = value;
}
```


НЕМНОГО
КОДА?
risovach.ru

5. Удалить элемент после заданного



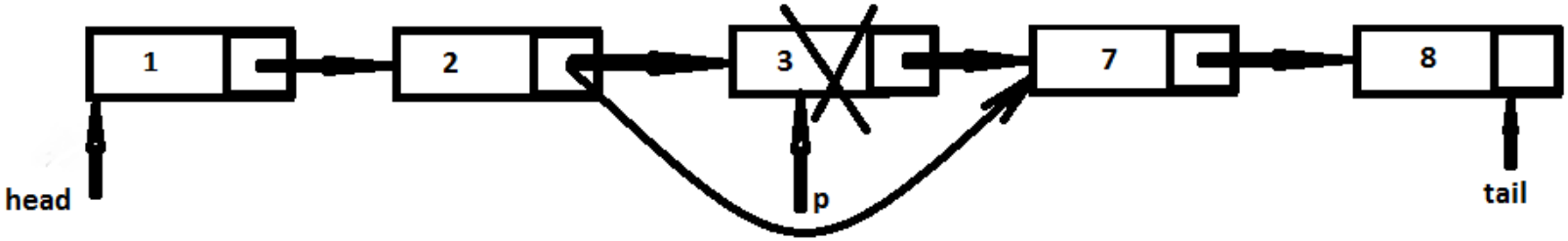

```
void deleteAfter(element * f) {
    element * t = f->next;
    f->next = f->next->next; //t->next
    delete t;
}
```


КАЖЕТСЯ ПОРА
ЧТО-ТО НАПИСАТЬ
risovach.ru


КАЖЕТСЯ, Я ЧТО-ТО
ПРОПУСТИЛ
risovach.ru

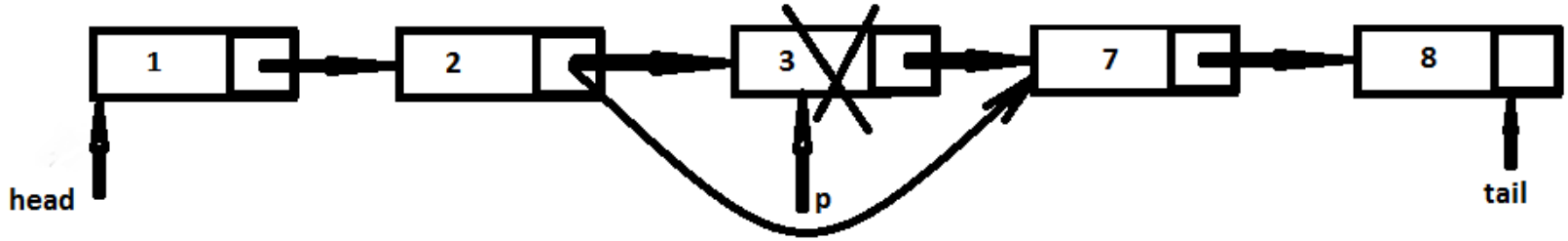
Ну да, после 3 должно быть 4, кажется…

4. Удалить заданный элемент



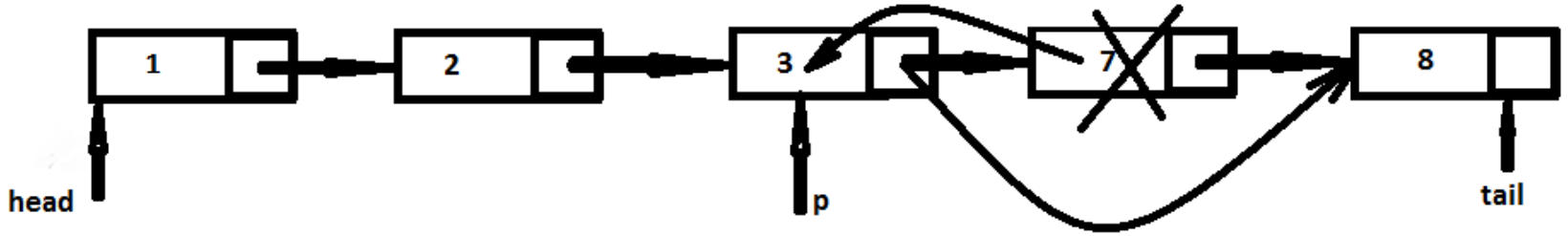
?

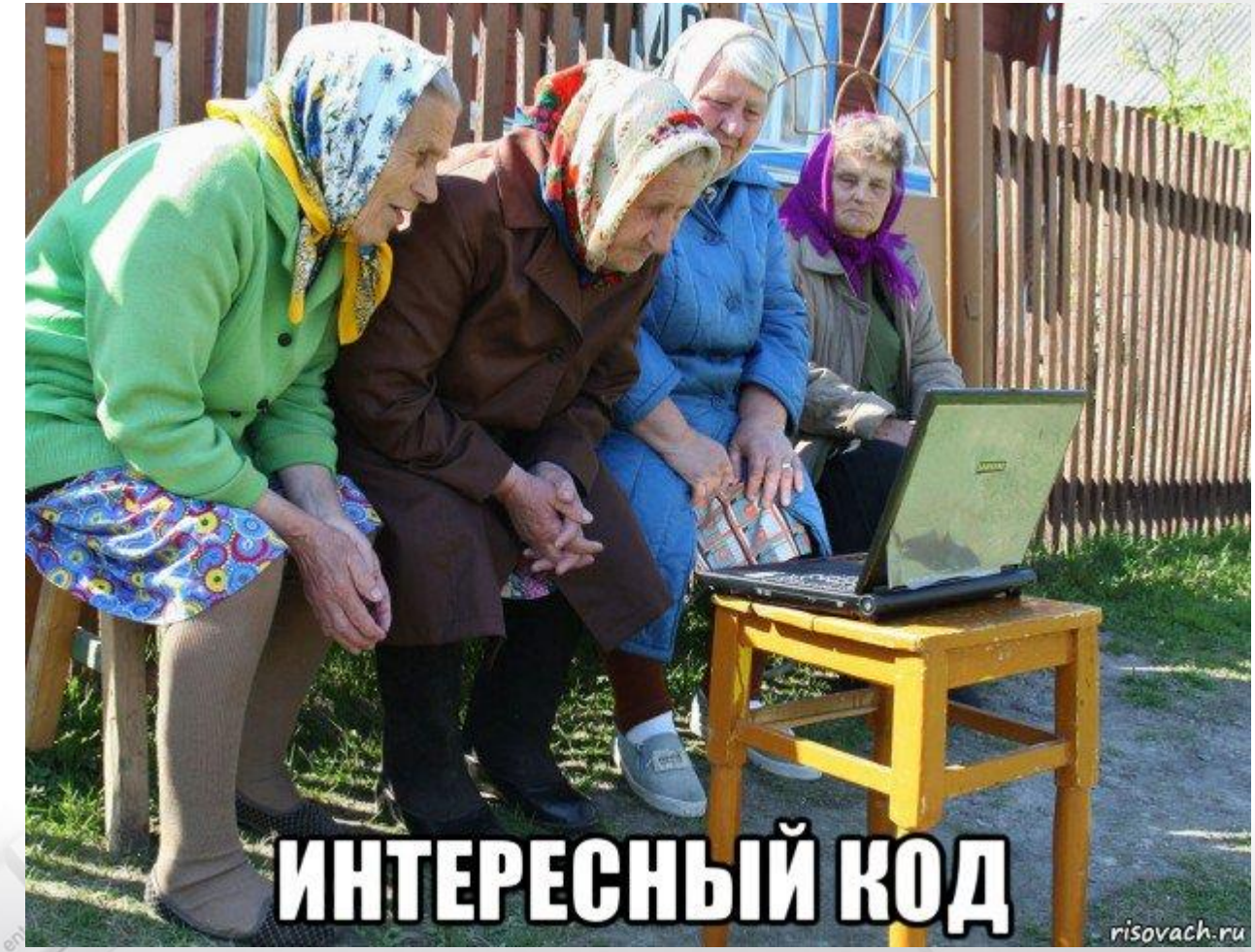
4. Удалить заданный элемент



?

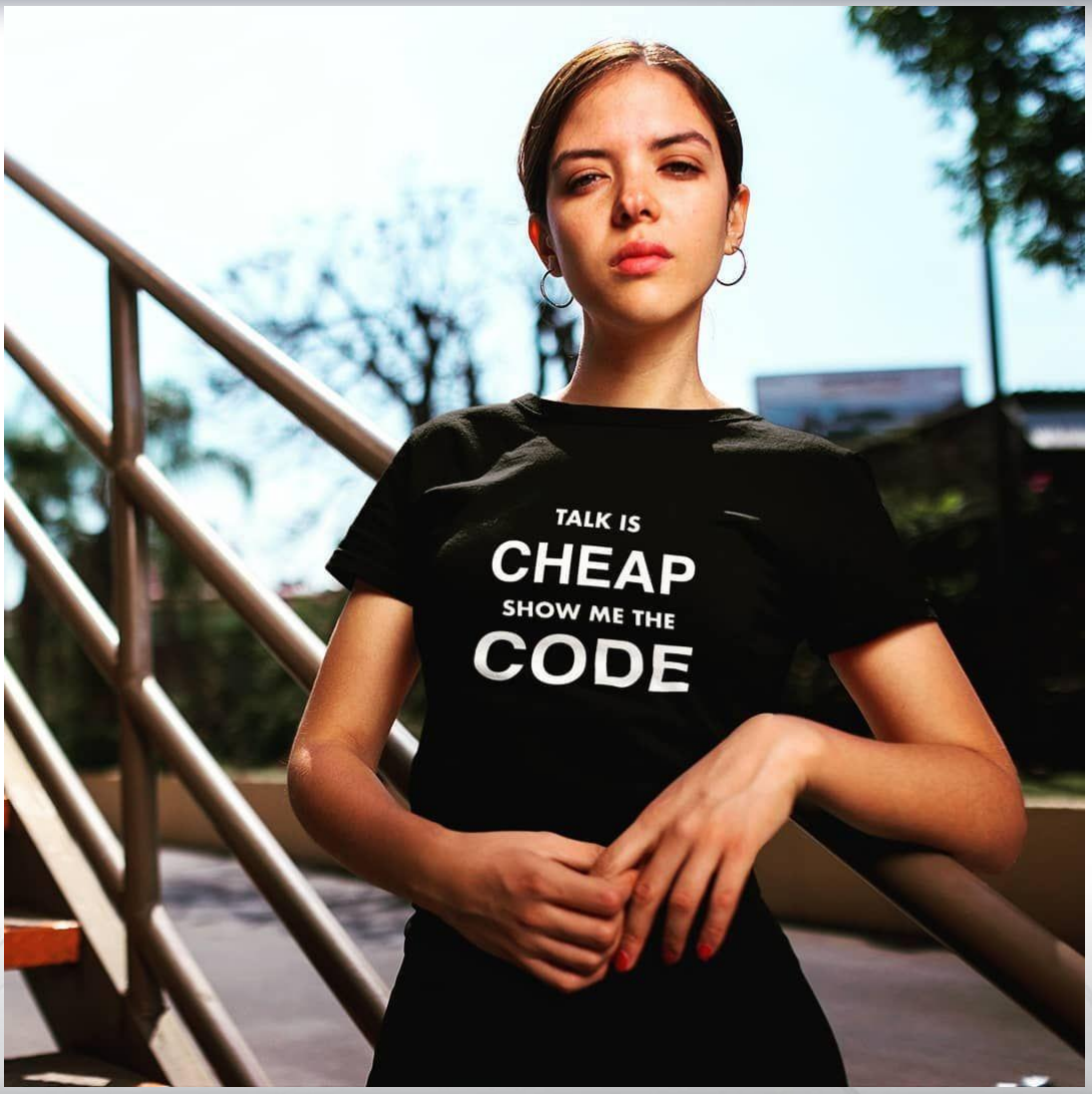
А если так попробовать?




ИНТЕРЕСНЫЙ КОД
risovach.ru

А как же Kotlin ?

- Отсутствие указателей – вместо них безопасные ссылки
- Объектная ориентированность – можно «скрыть» детали реализации
- Автоматическая сборка мусора решает много проблем
- Встроенная null-safety – код с логическими ошибками доступа к нулевым ссылкам часто не компилируется


TALK IS
CHEAP
SHOW ME THE
CODE

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Алгоритмизация и программирование

Программирование на C/C++ и Kotlin

(ч.8 – динамические списки)

Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua