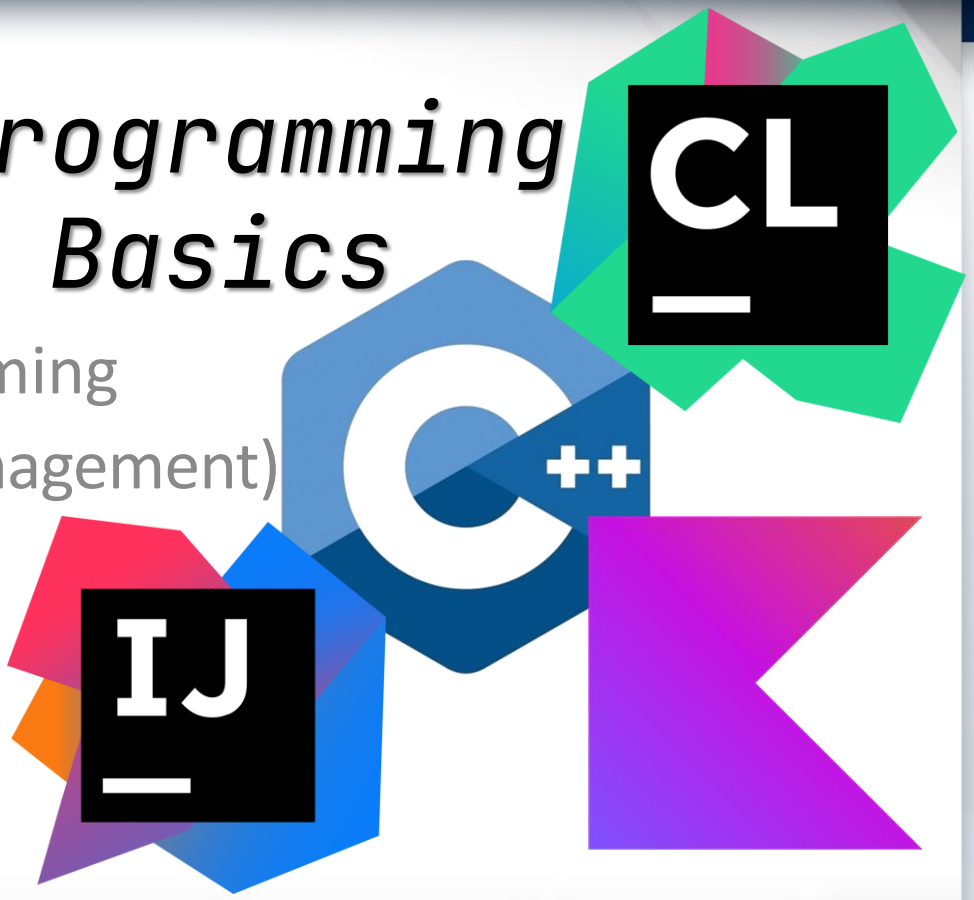
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.8 – Dynamic Memory Management)



Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Why memory management?

- We can create arrays of the desired size by specifying the size in square brackets
- You can always create an array "with a margin“ For example:

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

Or:

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

Why memory management?

- One or two arrays – it's OK, but what if … 1000?
- But most of them will be empty!

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

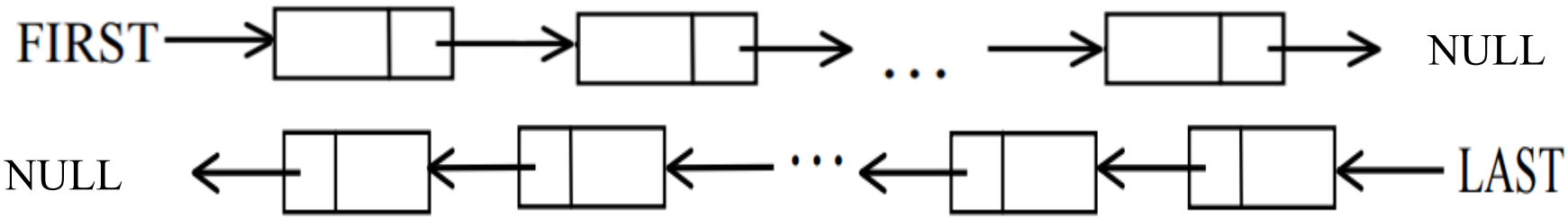
What can we do?

- You can use dynamic memory allocation.
- For example, you can use the abstract data structure "linear linked list"

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

What can we do?

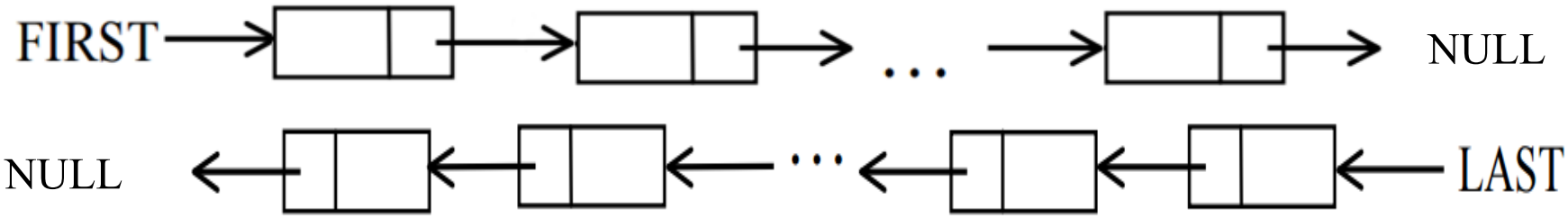
- You can use dynamic memory allocation.
- For example, you can use the abstract data structure "linear linked list"



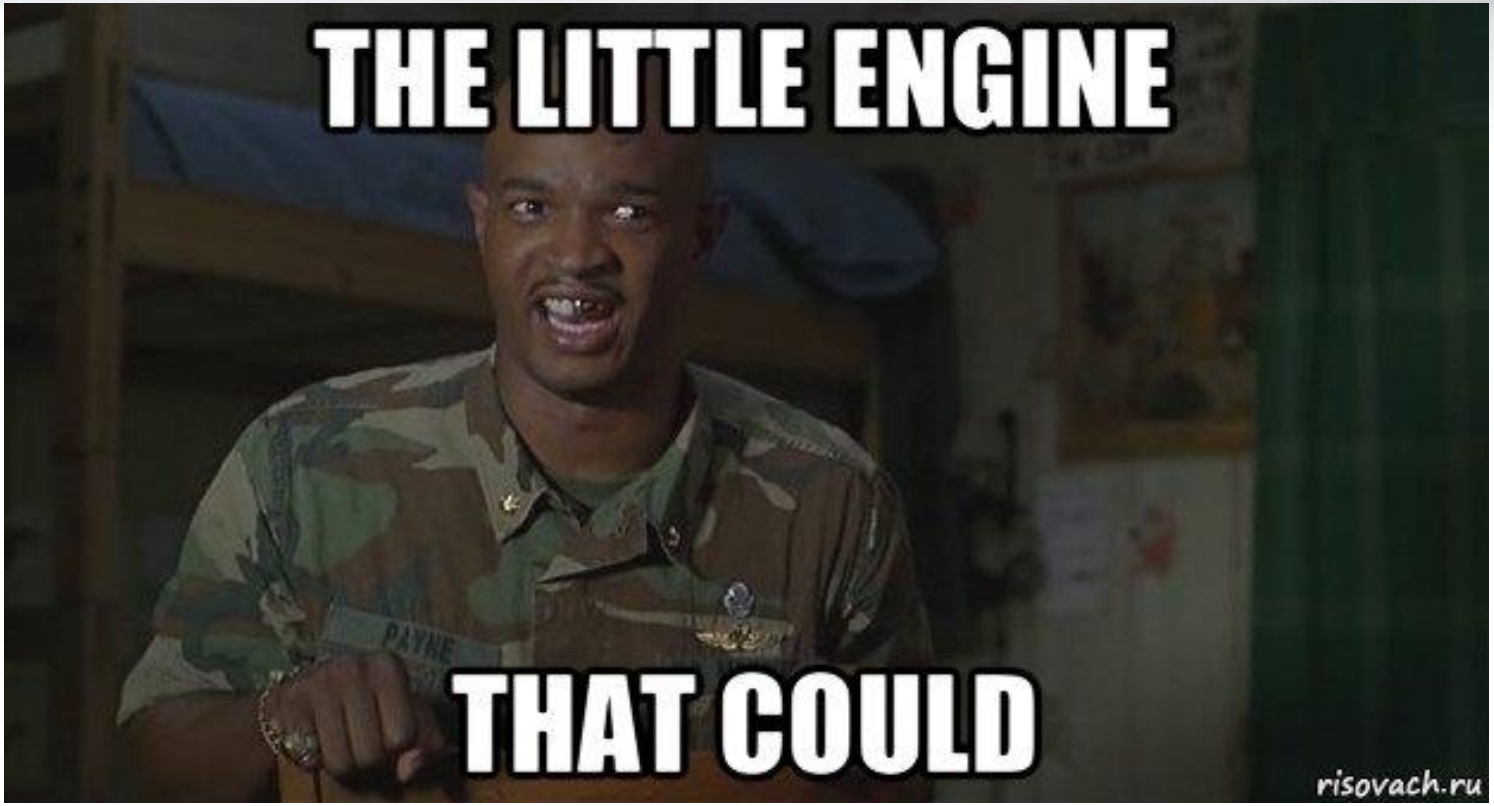
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Linear linked list

- You can use dynamic memory allocation.
- For example, you can use the abstract data structure "linear linked list"



What does it look like?



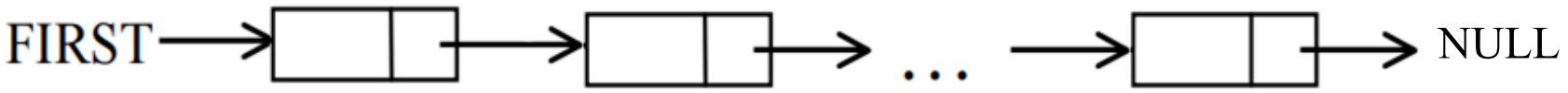
This was supposed to be a picture of a train.
But, I think, this one is better☺

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

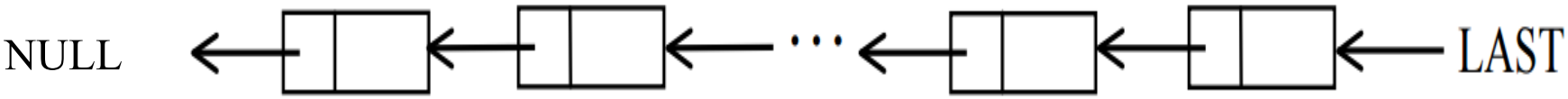
Linear linked list

Types of Linked Lists:

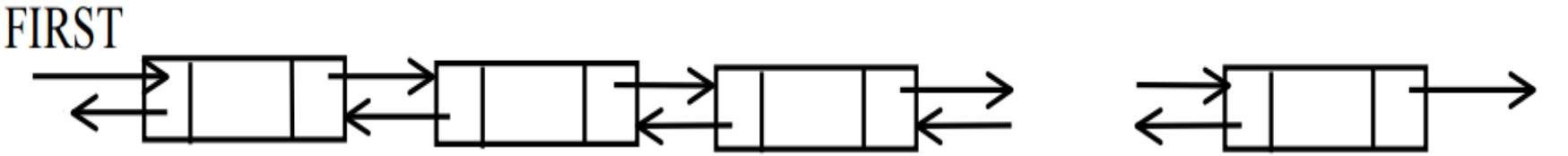
Every element contains a pointer to next



Every element contains a pointer to previous

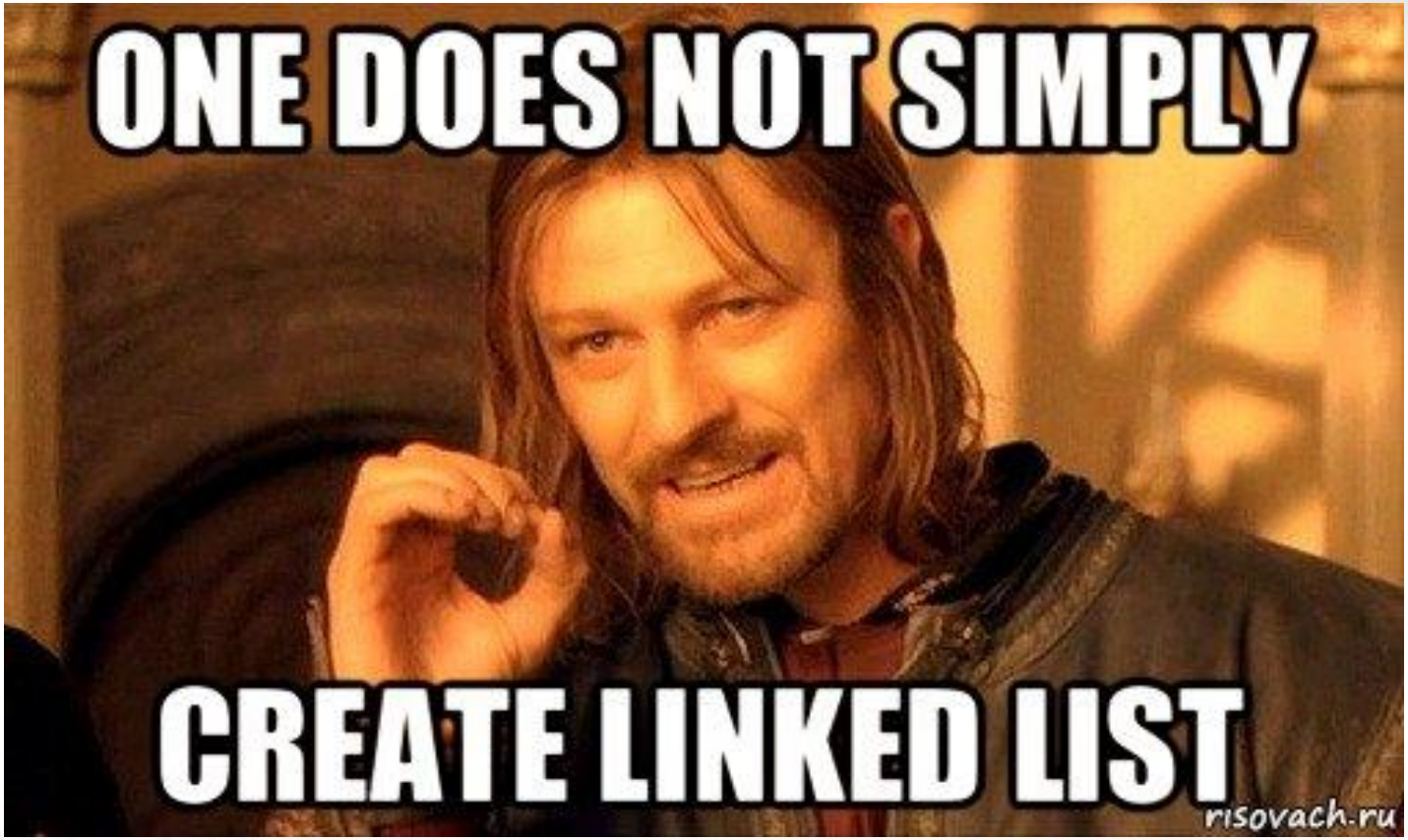


Doubly linked list



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Did you got it?



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

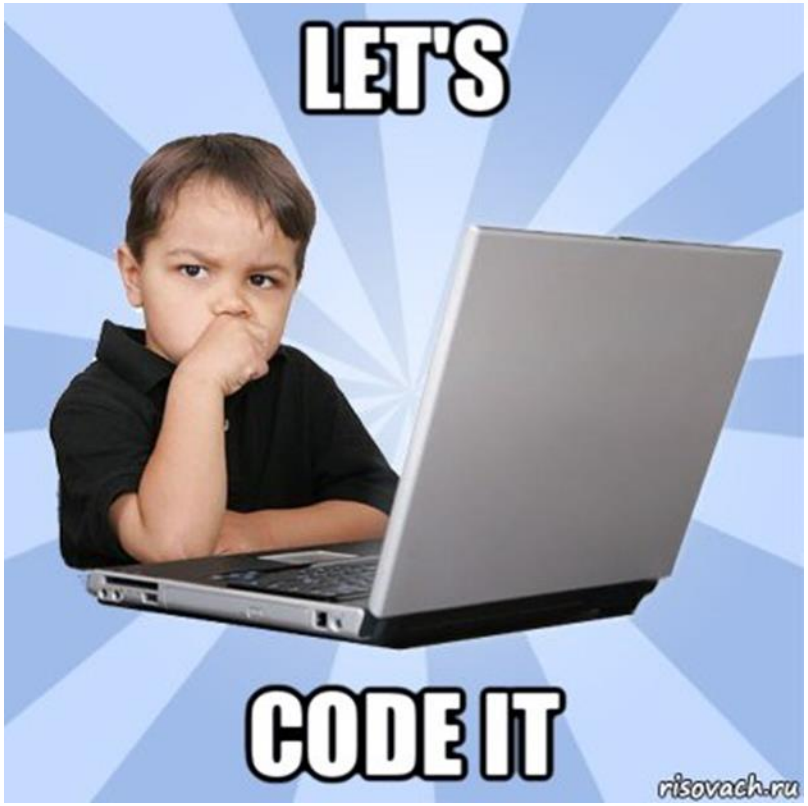
Let’s code!

Let's declare a new type - the structure of the list element:

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

Now, to build a linear list, we can use a variable of this type:

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

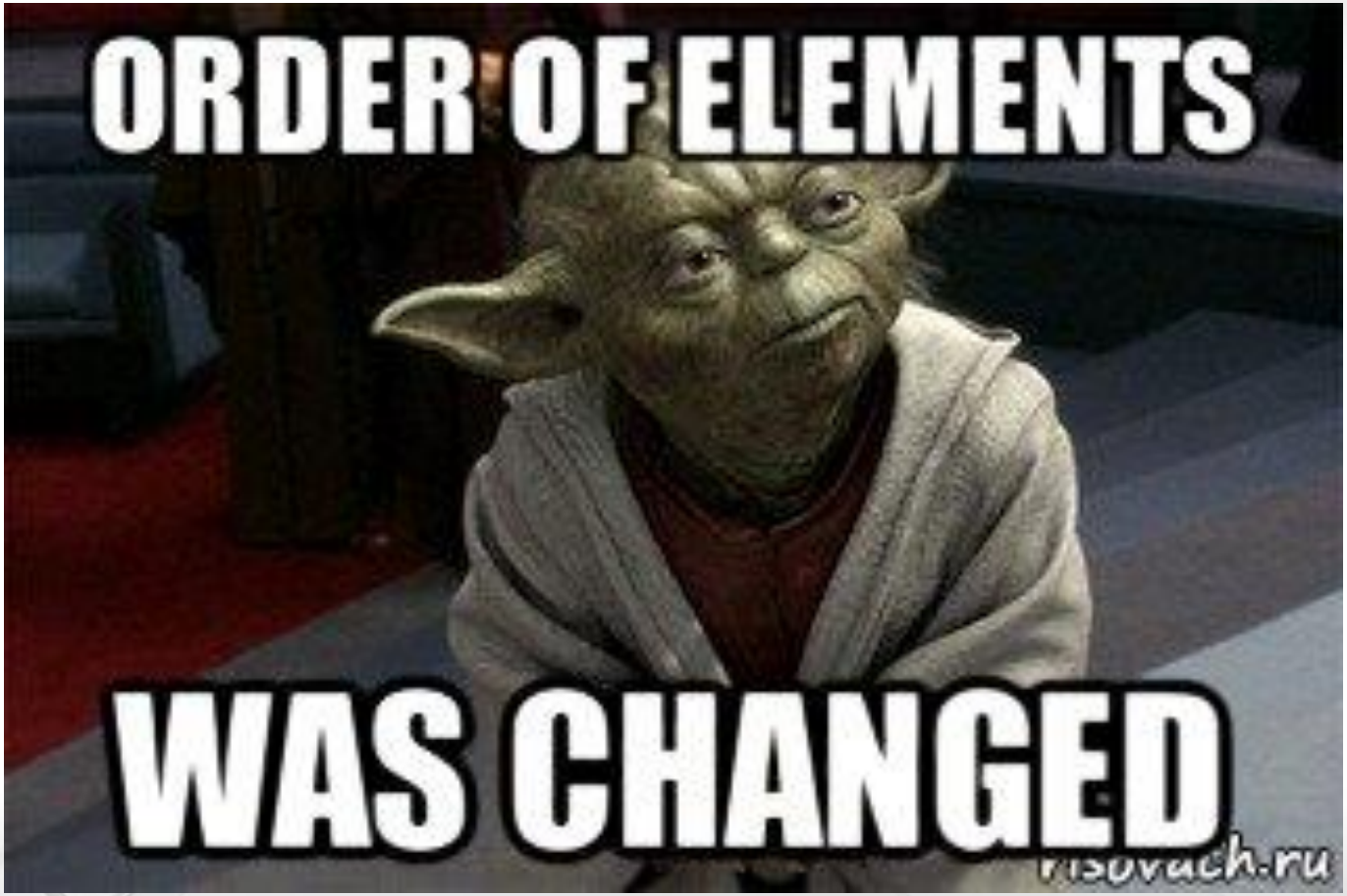

LET'S
CODE IT
risovach.ru

Stack

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

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

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


ORDER OF ELEMENTS
WAS CHANGED
risovach.ru

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Types of Lists

- **A stack is a linear data structure** that follows the Last-In-First-Out (**LIFO**) principle, where the last element inserted into the stack is the first one to be removed.
- Most often, the principle of the stack is compared with a stack of plates: in order to take the second from the top, you need to remove the top one.

Stack

A stack can be thought of as a collection of elements, with two main operations:

- **push()**: Adds an element to the top of the stack.
- **pop()**: Removes the element at the top of the stack.

Additional operations that may be supported by a stack include:

- **peek()/top()**: Returns the element at the top of the stack without removing it.
- **size()**: Returns the number of elements in the stack.
- **empty()**: Returns whether the stack is empty or not.

Types of Lists

- **A queue is a linear data structure** that follows the First-In-First-Out (**FIFO**) principle, where the first element inserted into the queue is the first one to be removed.

Adding an element is possible only to the end of the queue, selecting only from the beginning of the queue, while the selected element is removed from the queue.

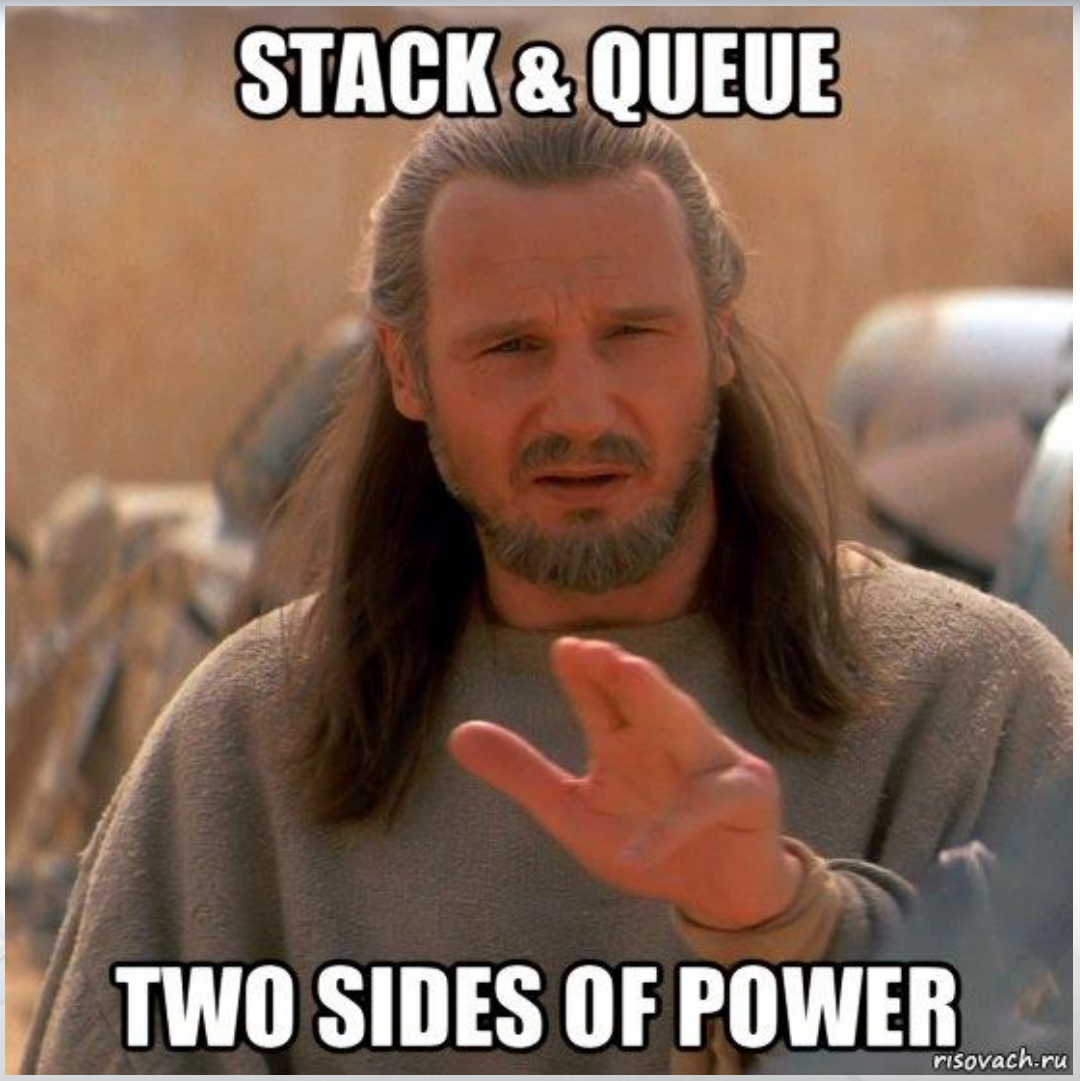
Queue

A queue can be thought of as a collection of elements, with two main operations:

- **offer()**: Adds an element to the back of the queue.
- **pull()**: Removes the element at the front of the queue.

Additional operations that may be supported by a queue include:

- **front()**: Returns the element at the front of the queue without removing it.
- **size()**: Returns the number of elements in the queue.
- **empty()**: Returns whether the queue is empty or not.


STACK & QUEUE
TWO SIDES OF POWER
risovach.ru

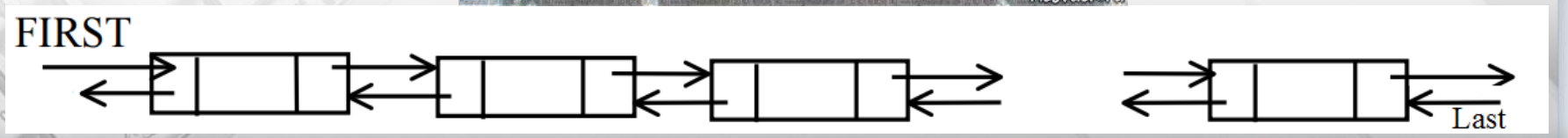
Queue

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

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

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

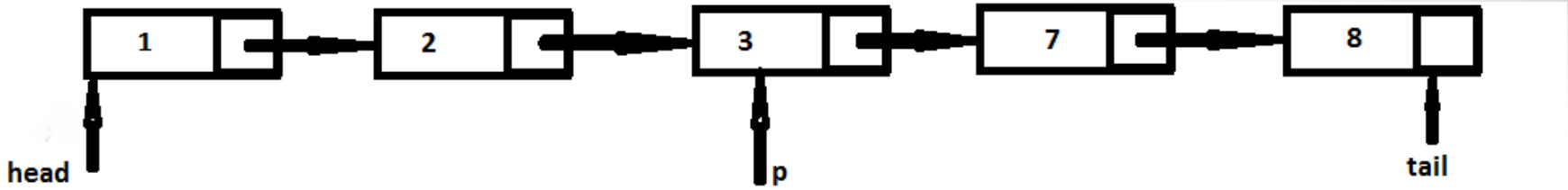
Linear linked lists



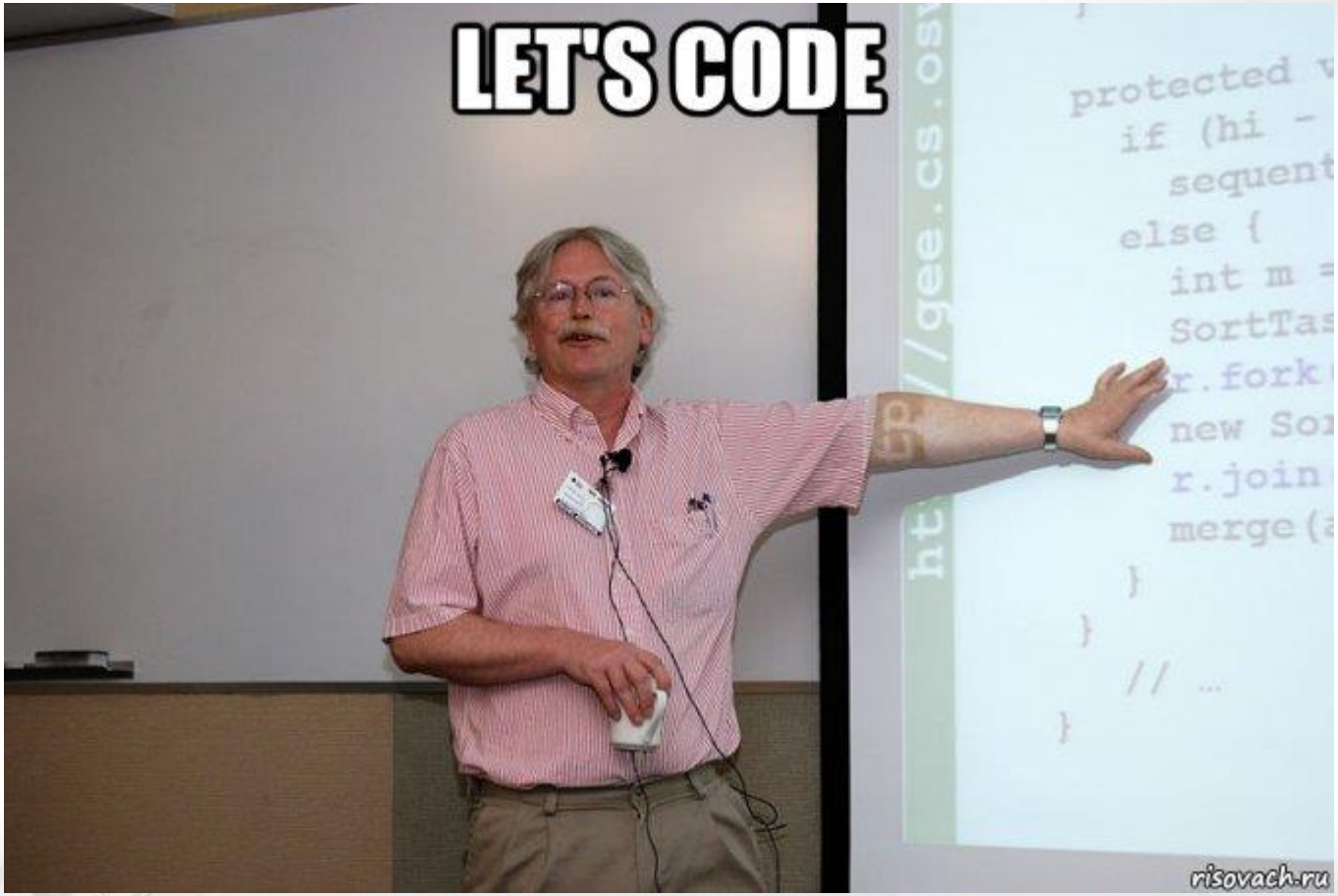
Basic list tasks

1. Find the given element (pointer to it)
2. Add element after the given one
3. Add element before the given one
4. Delete the given element
5. Remove element after given one

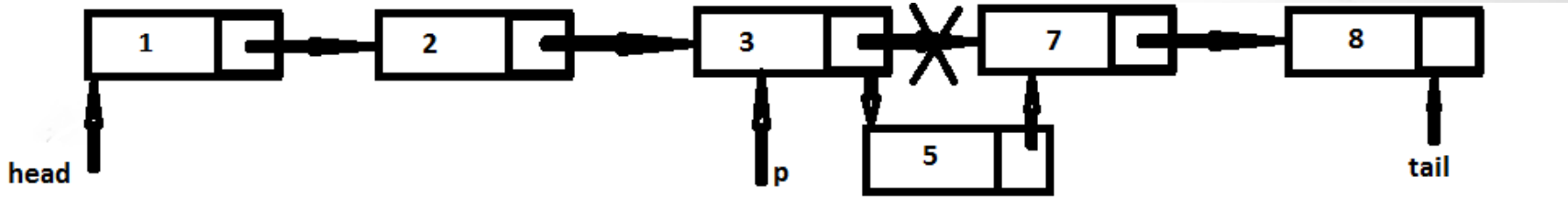
1. Find the given element (pointer to it)



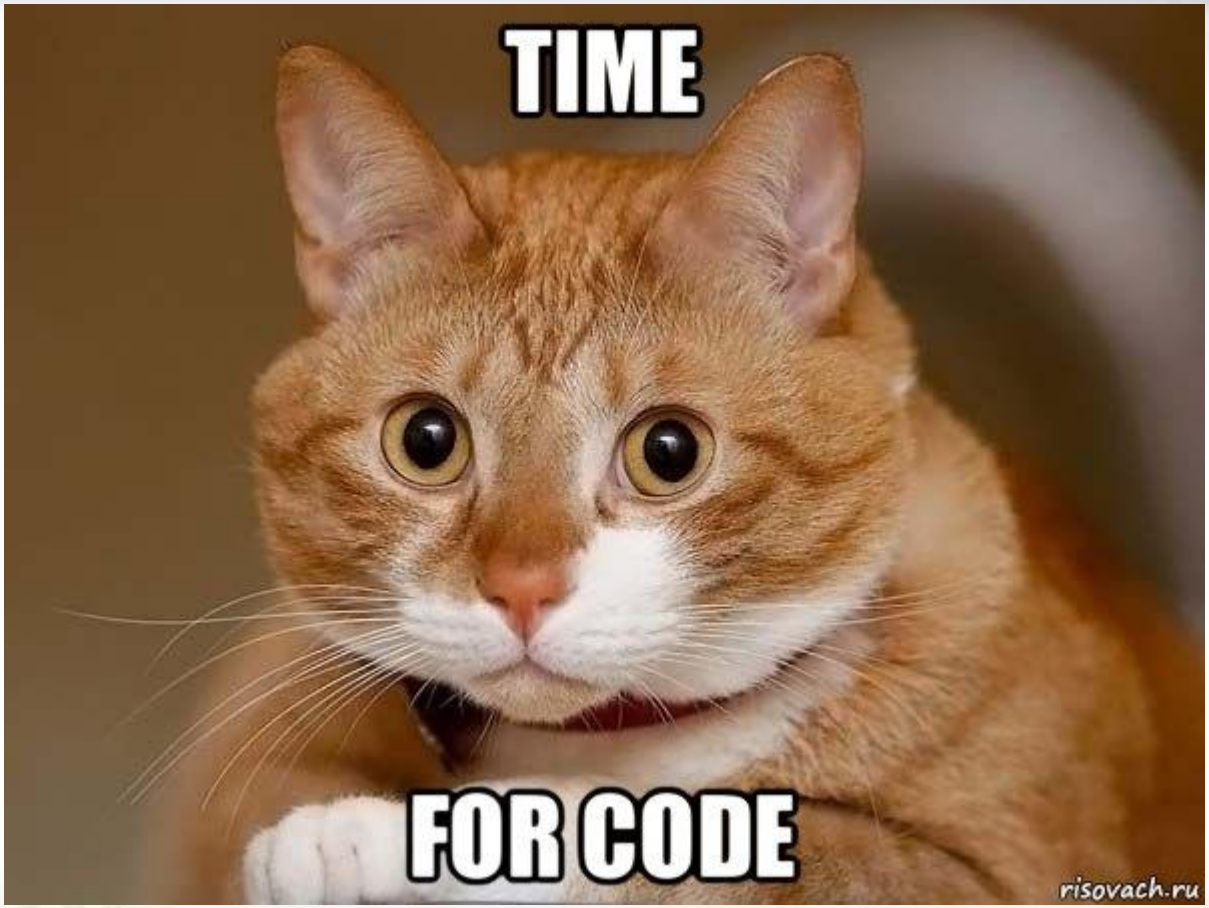

```
element * find(element * head, int value) {
    element * t;
    t = head;
    while (t) {
        if (t->x == value) {
            break;
        } else {
            t = t->next;
        }
    }
    return t;
}
```


LET'S CODE
risovach.ru

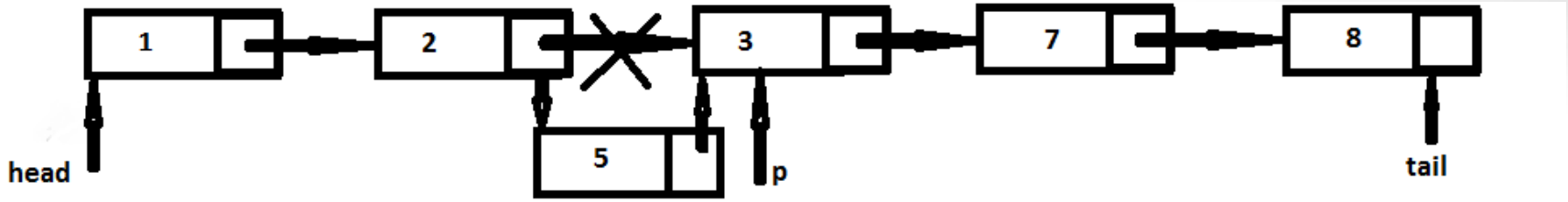
2. Add element after given one



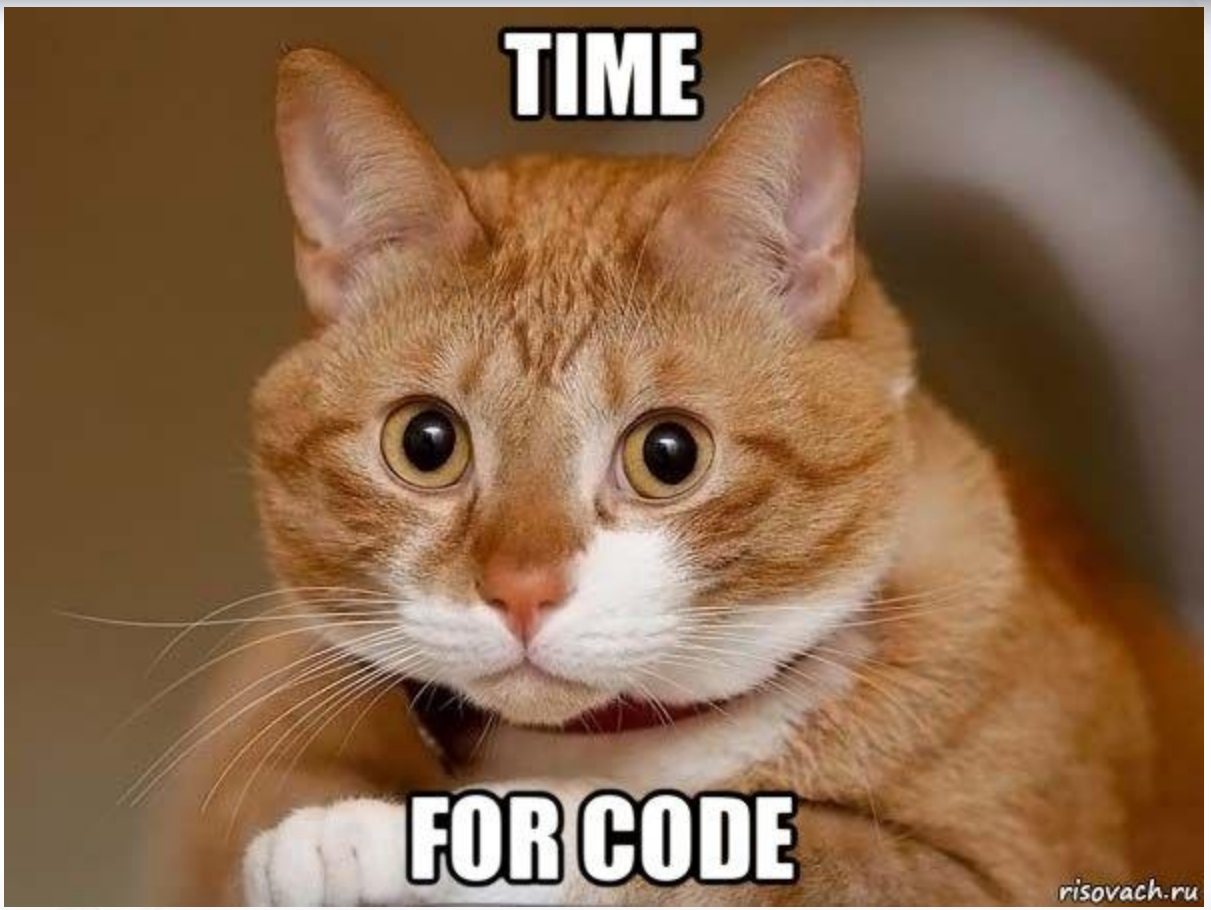

```
void addAfter(element * f, int value) {
    //1
    element * t = new element;
    t->x = value;
    //2
    t->next = f->next;
    //3
    f->next = t;
}
```


TIME
FOR CODE
risovach.ru

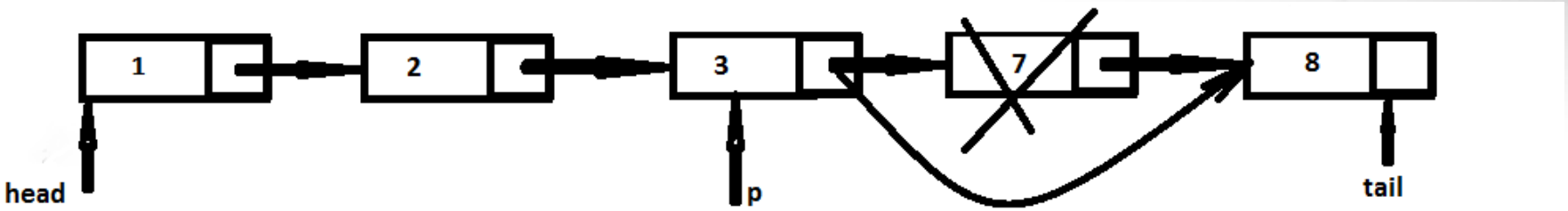
3. Add element before the given one



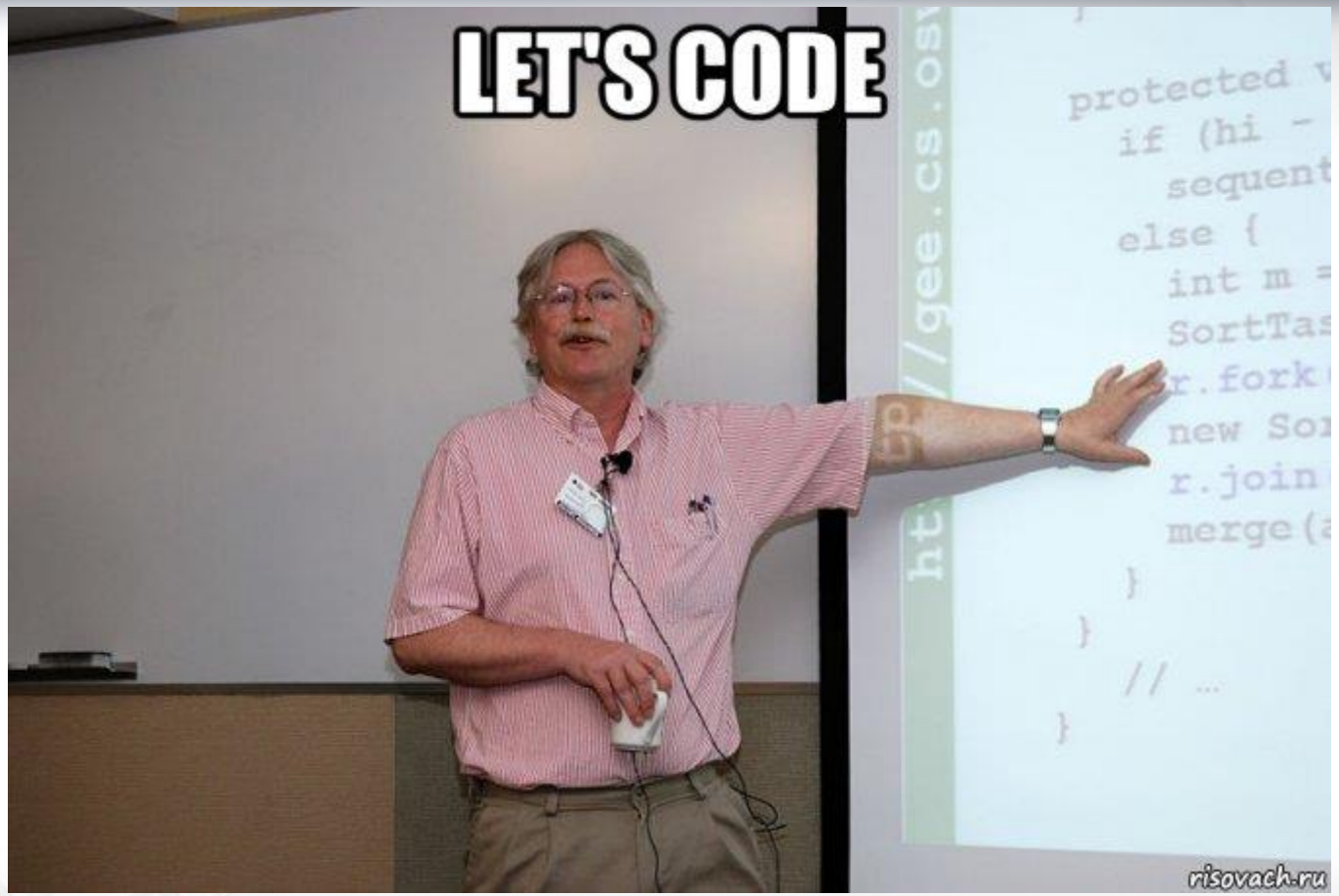

```
void addBefore(element * f, int value) {
    //1
    element * t = new element;
    //2
    t->next = f->next;
    //3
    f->next = t;
    //4
    t->x = f->x;
    f->x = value;
}
```

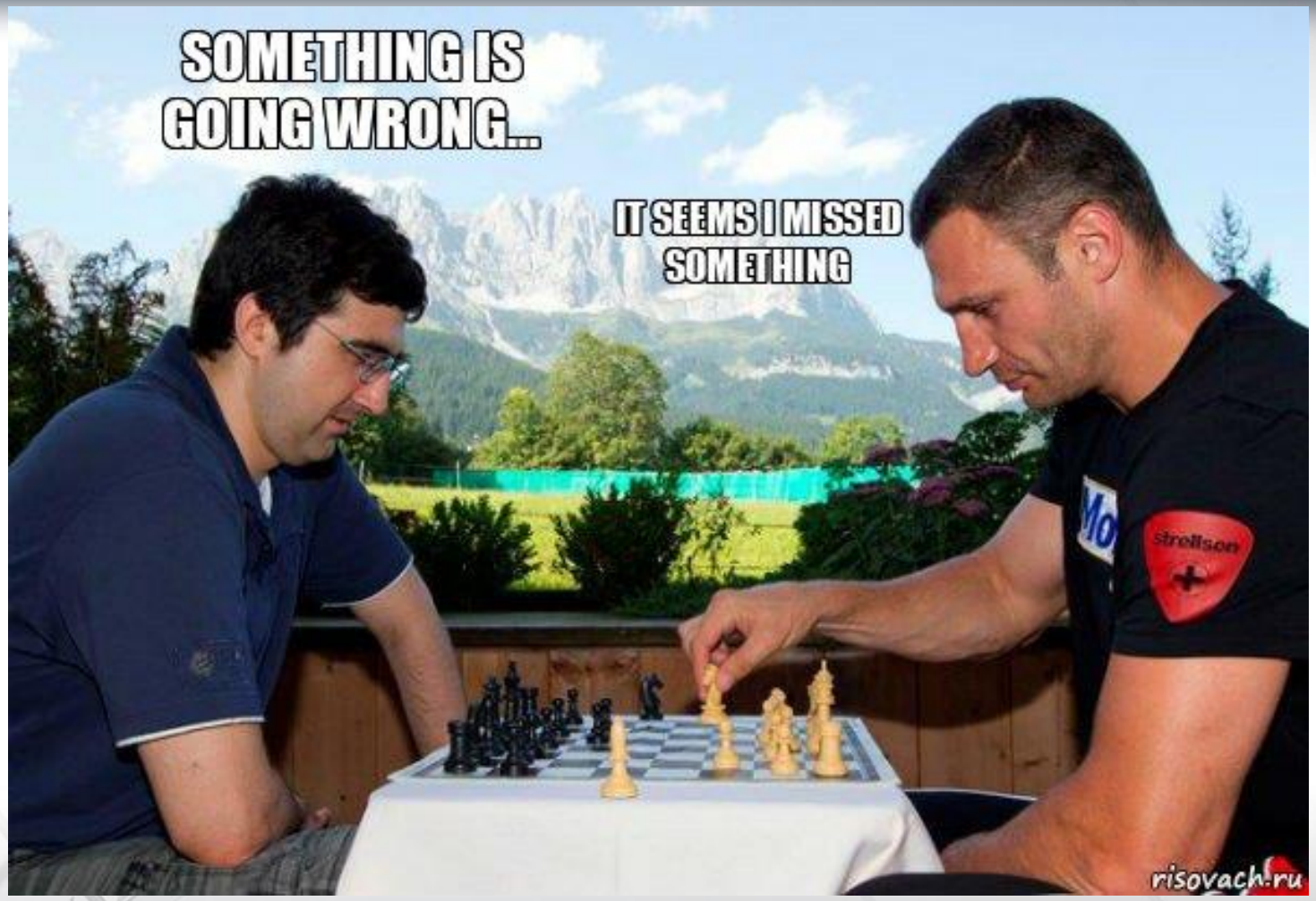

TIME
FOR CODE
risovach.ru

5. Remove element after given one



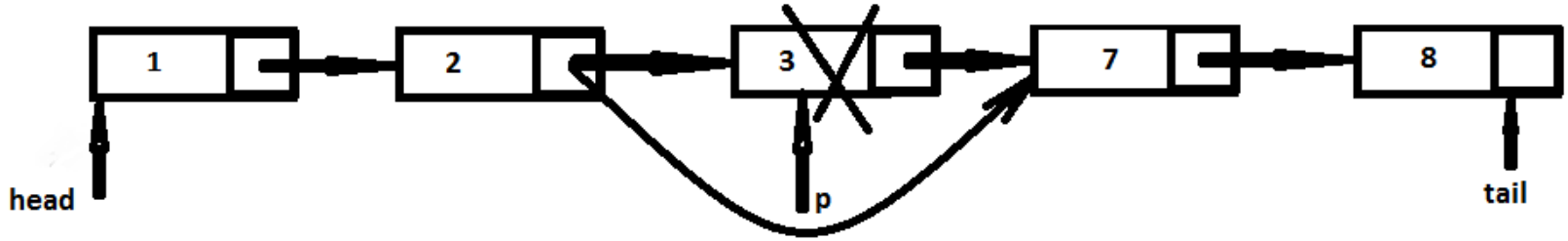

```
void deleteAfter(element * f) {
    element * t = f->next;
    f->next = f->next->next; //t->next
    delete t;
}
```


LET'S CODE
risovach.ru



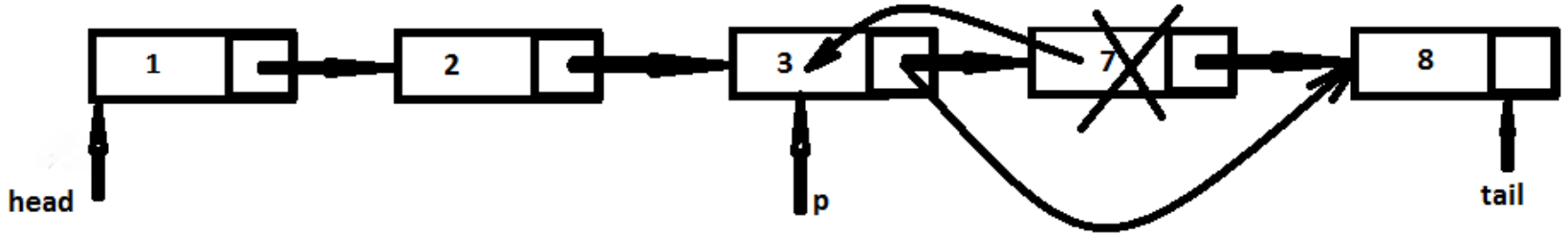
O… ok, after 3 should be 4, maybe…

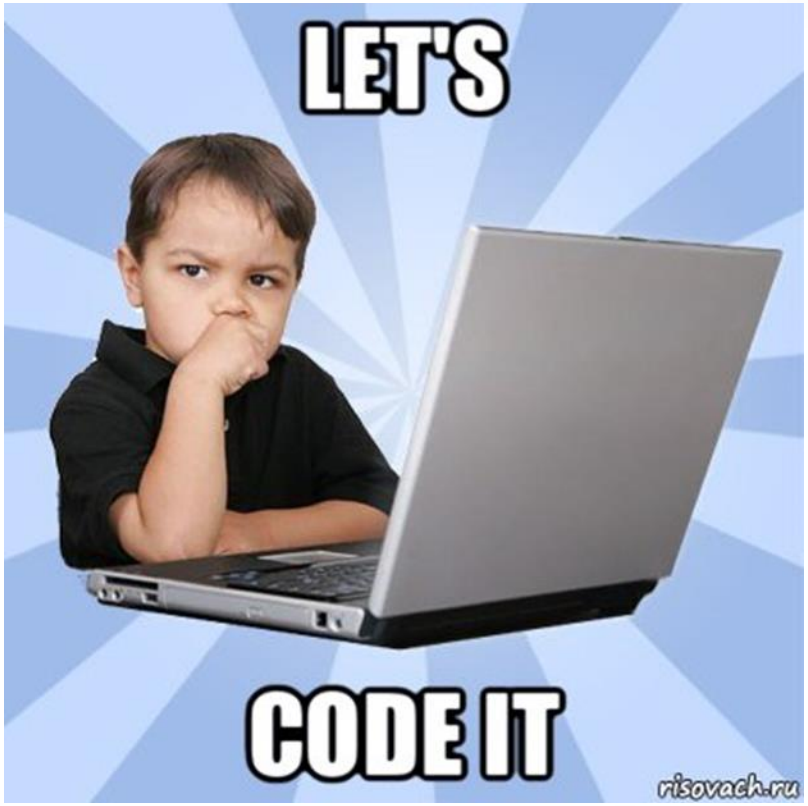
4. Delete the given element



?

А если так попробовать?




LET'S
CODE IT
risovach.ru

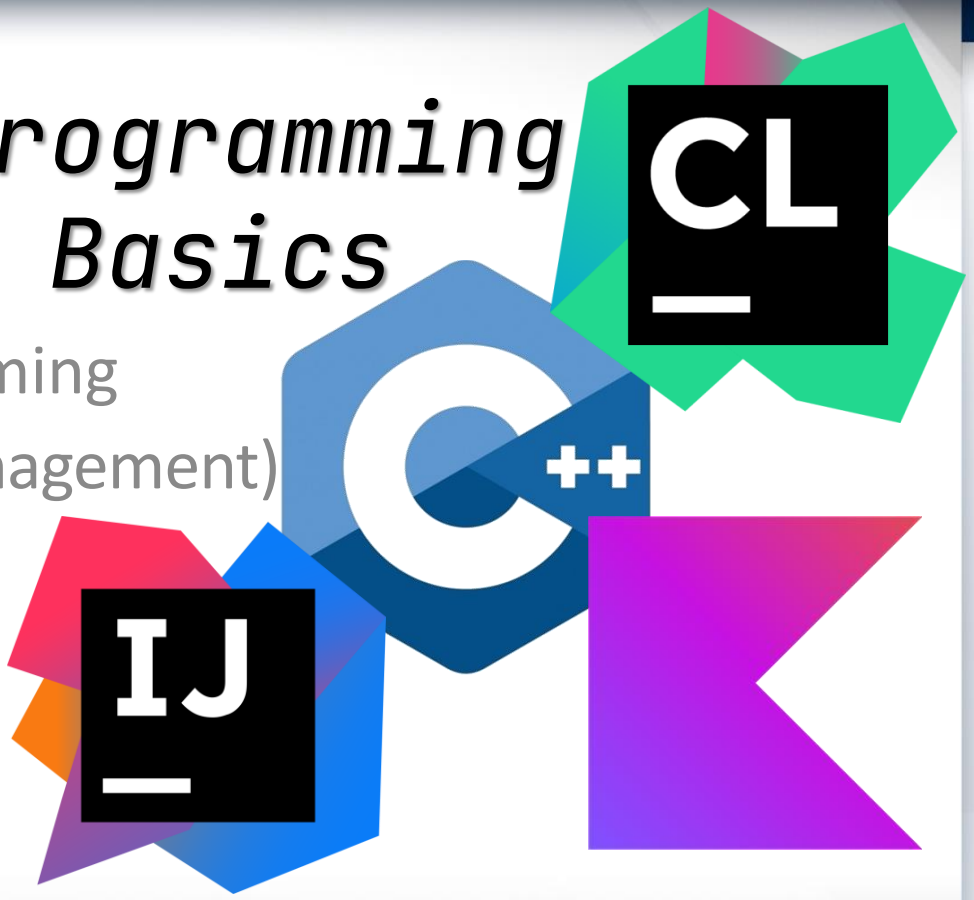
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.8 – Dynamic Memory Management)



Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua