Лекция VIII

11 матра 2017

Абстрактные типы данных

Абстракция - выявление существенных **признаков** какого-либо объекта или группы объектов.

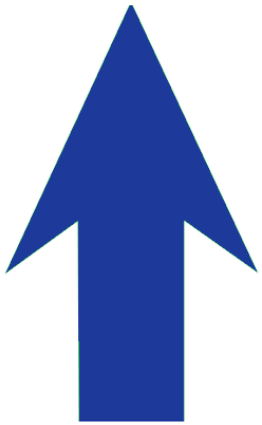
Списки - набор элементов одинакого типа, связанных между собой

Абстрактные типы данных. Списки


ЭЛЕМЕНТ СПИСКА
здесь храним данные
информация о соседях

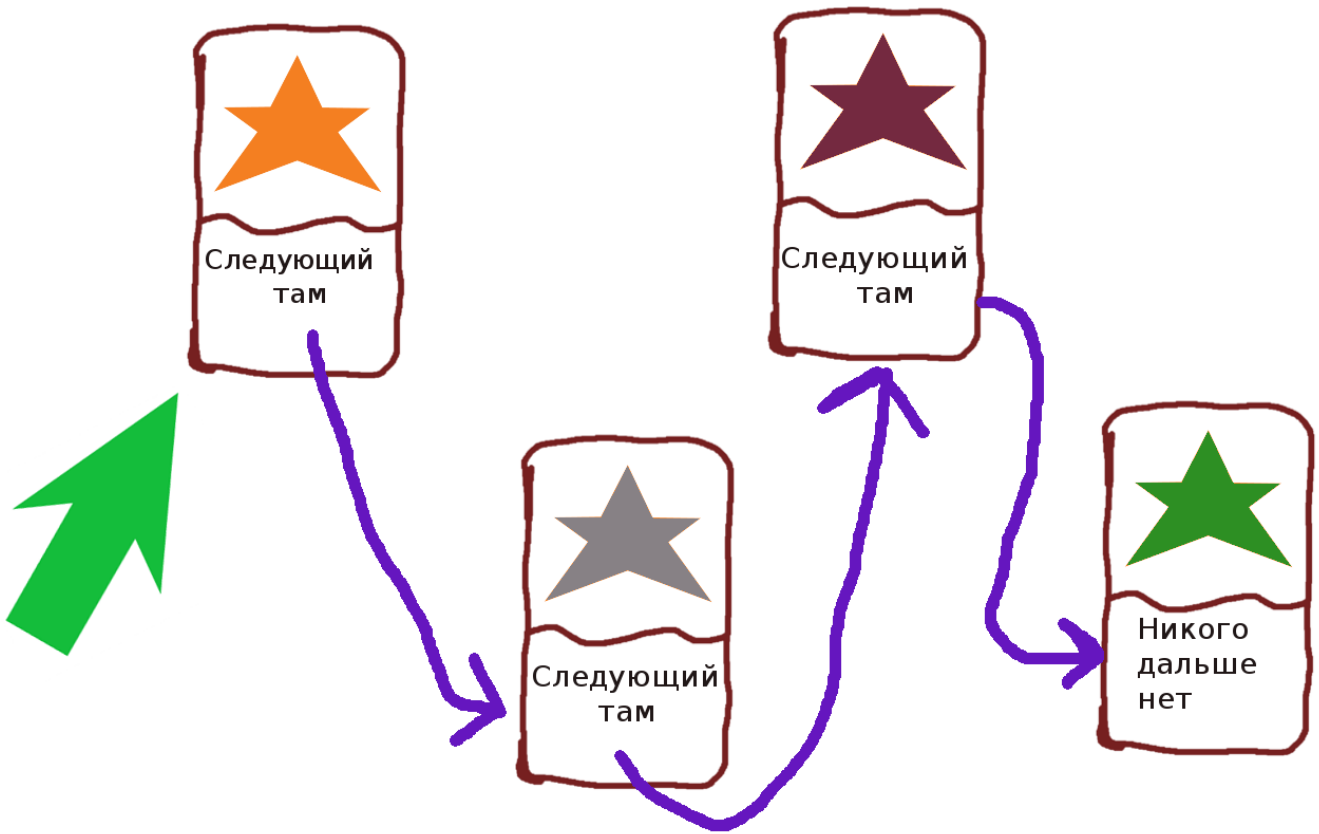
ОБОЗНАЧЕНИЯ

← НАЧАЛО СПИСКА

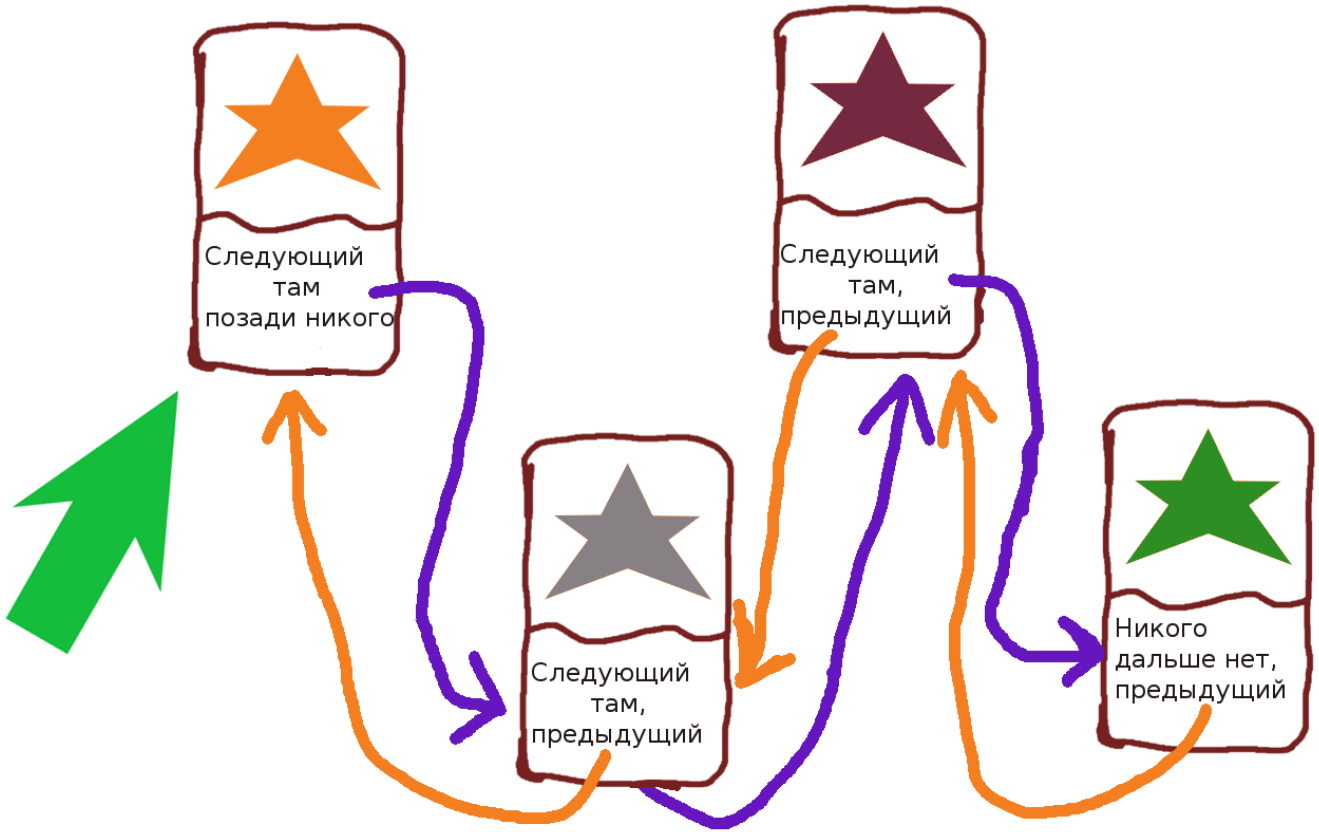


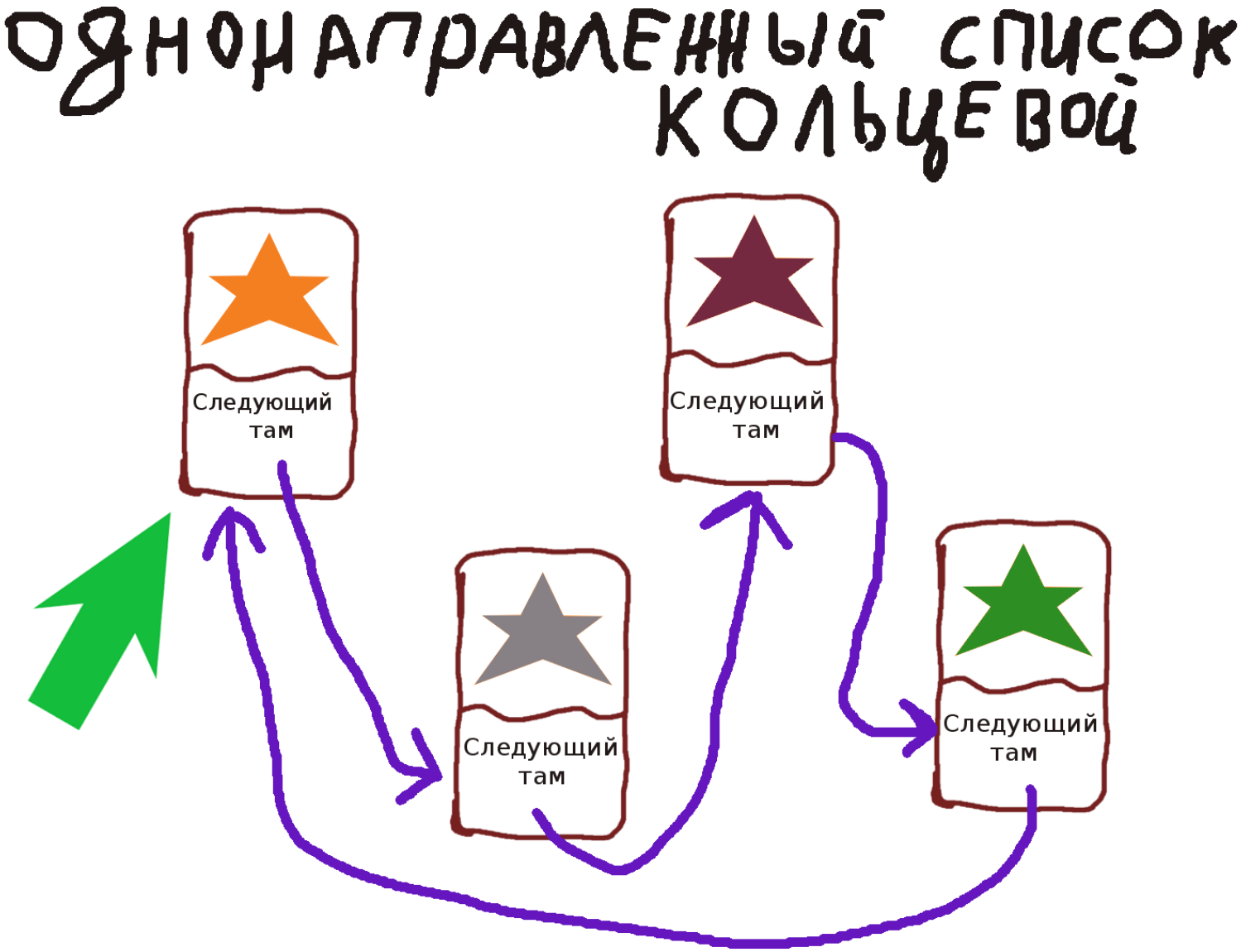
← Вспомогательный указатель на элемент списка

Односвязный список



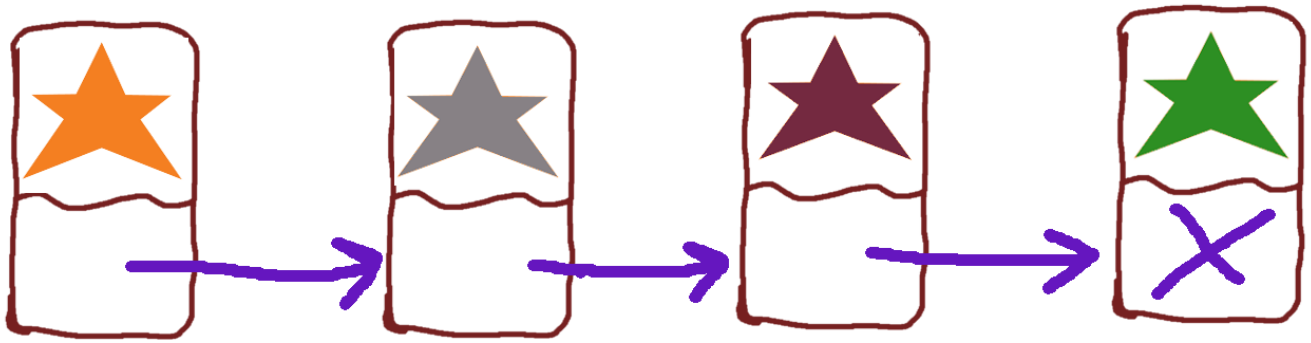
Двунаправленный список

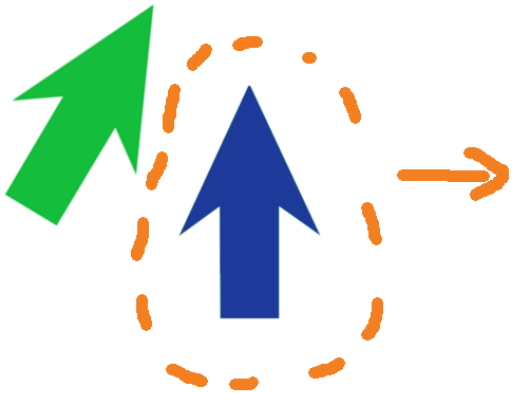



Односвязный список
кольцевой
Следующий там
Следующий там
Следующий там
Следующий там

Операции со списком

Поиск зелёной звезды

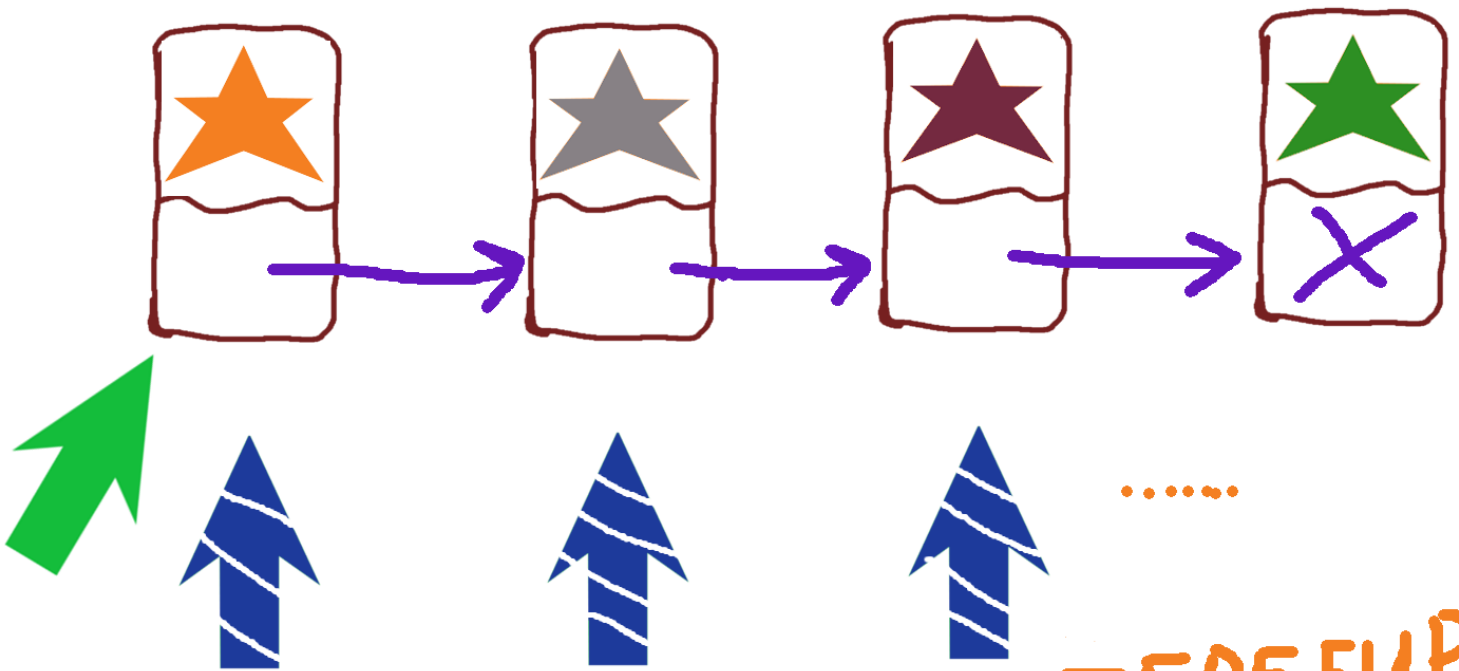




Начинаем с первого элемента списка

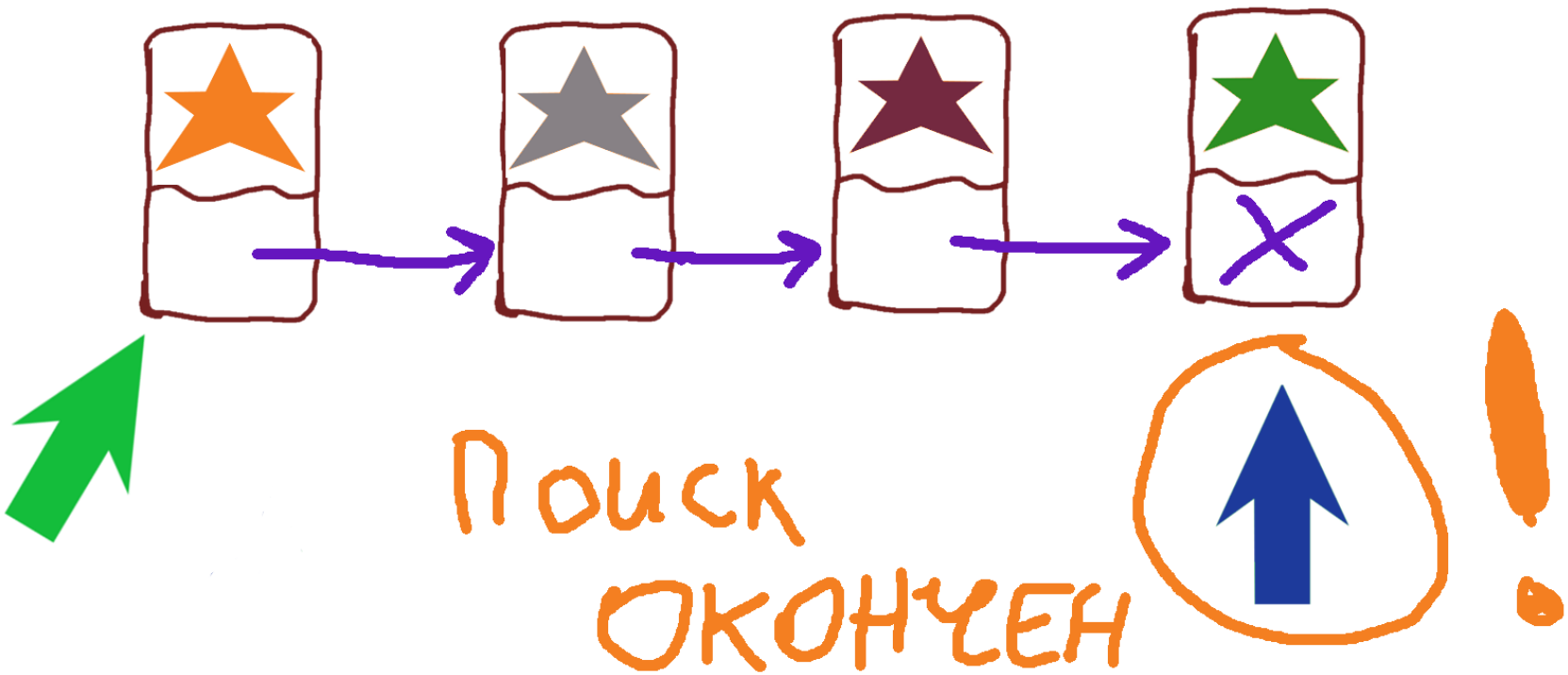
Операции со списком

Поиск зелёной звезды



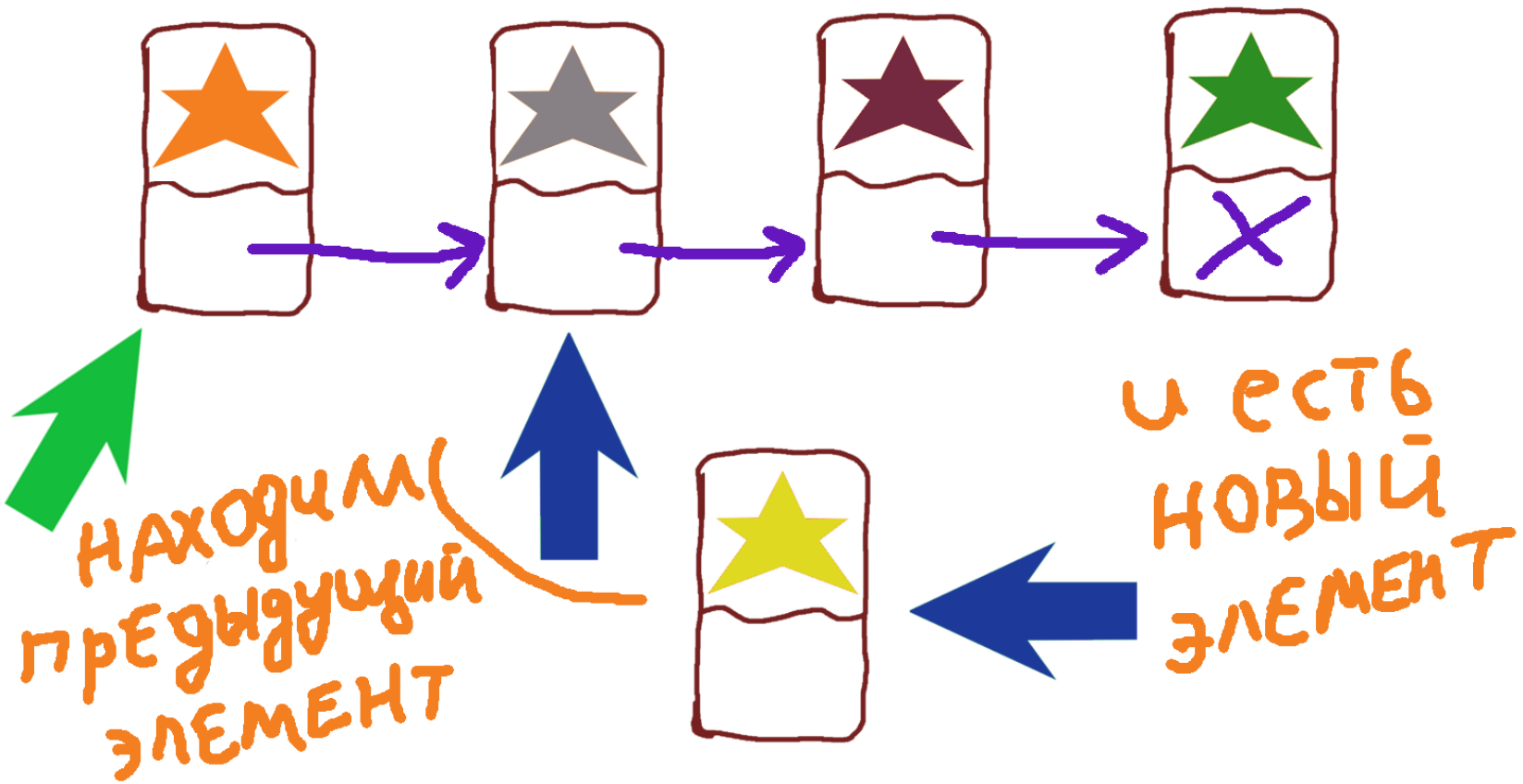
Операции со списком

Поиск зелёной звезды



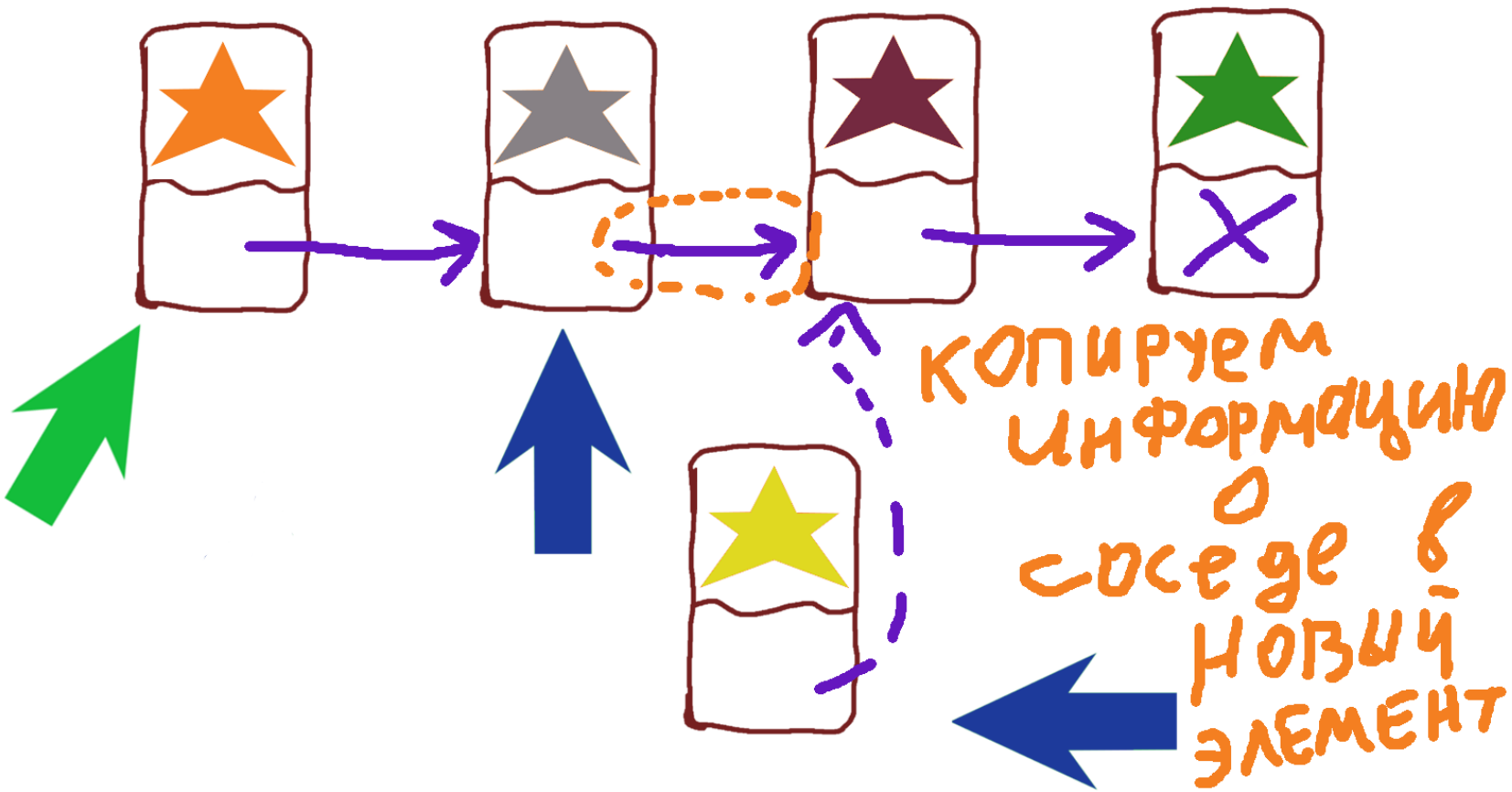
Операции со списком

Вставка на произвольное место: 3



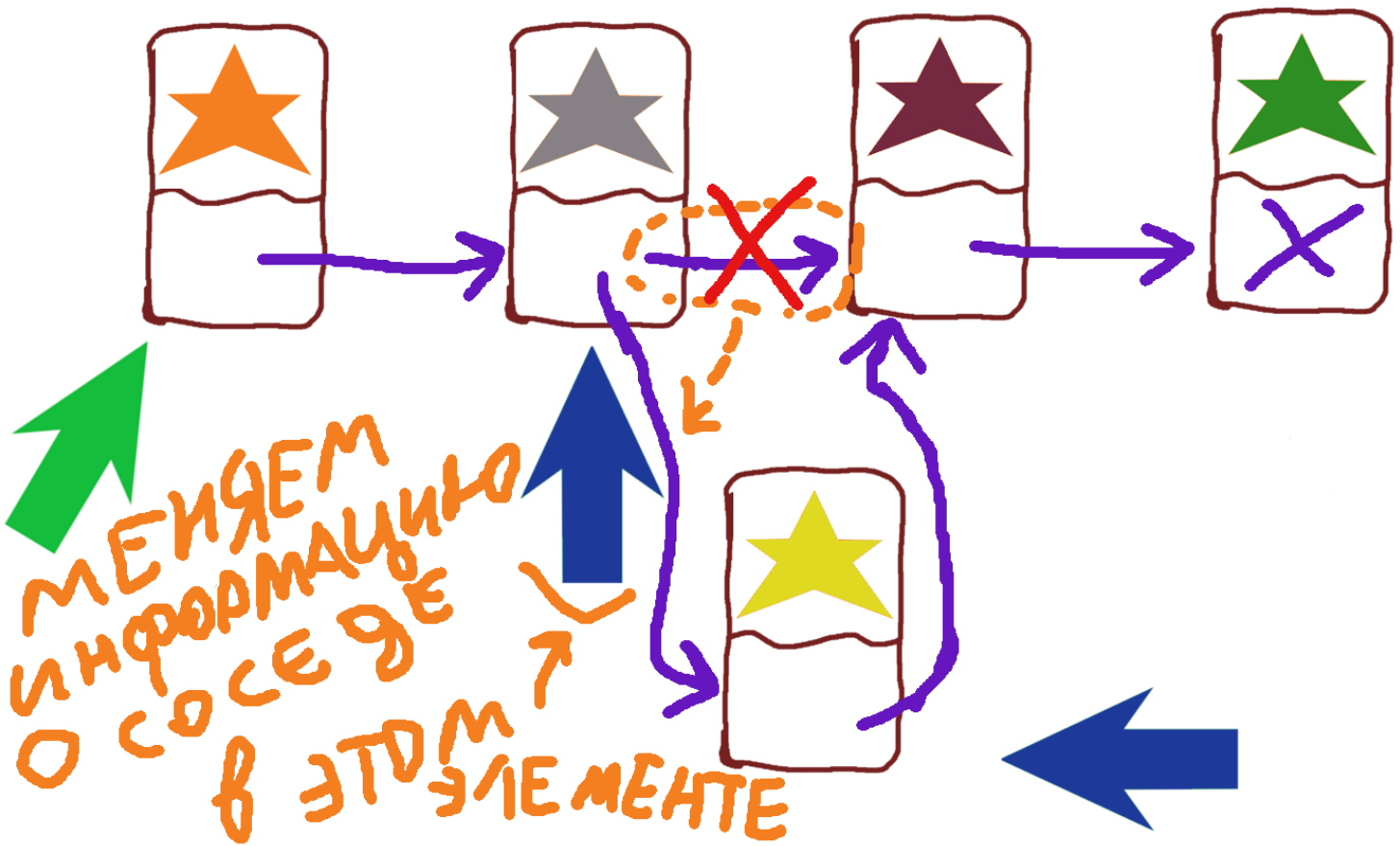
Операции со списком

Вставка на произвольное место: 3



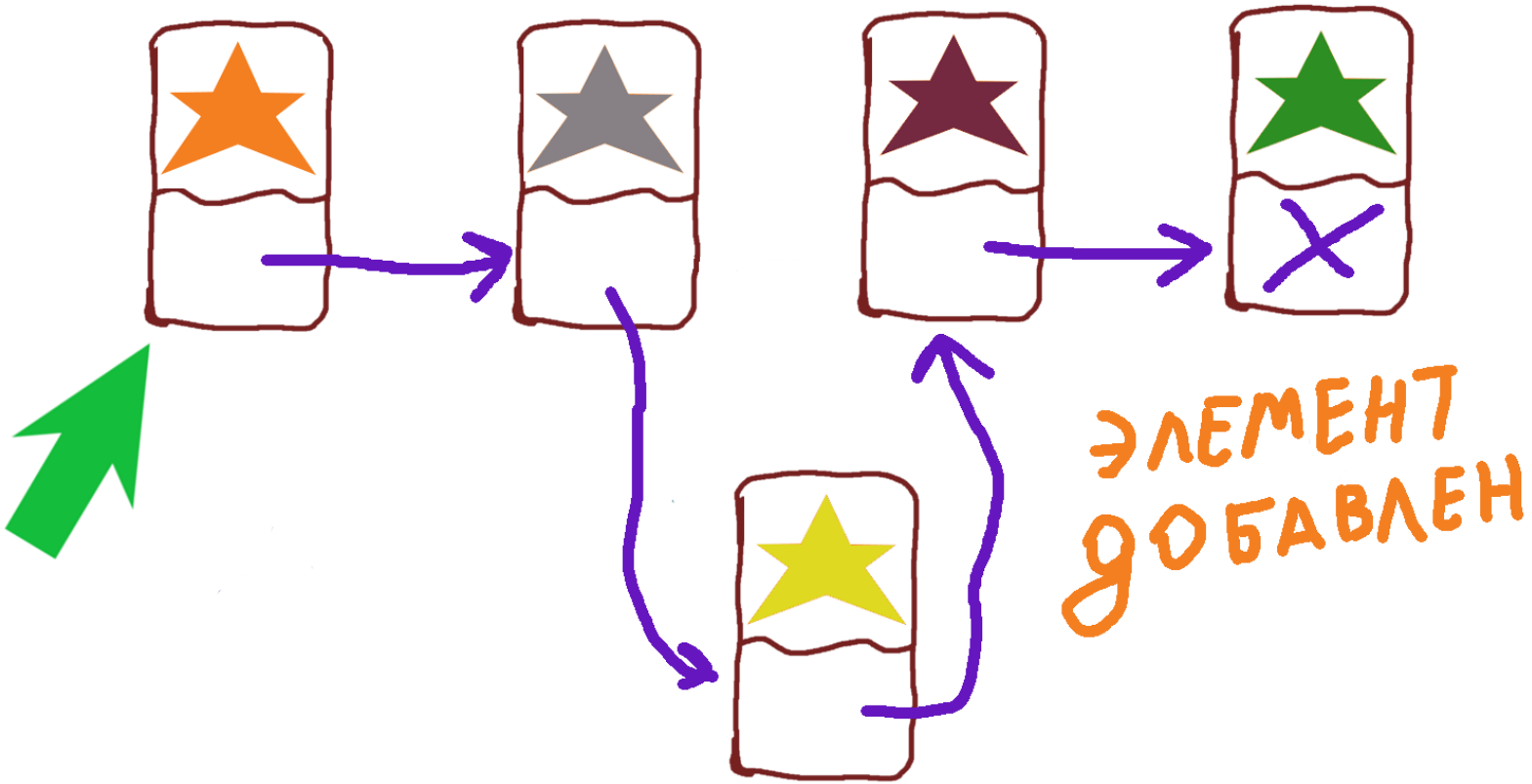
Операции со списком

Вставка на произвольное место: 3



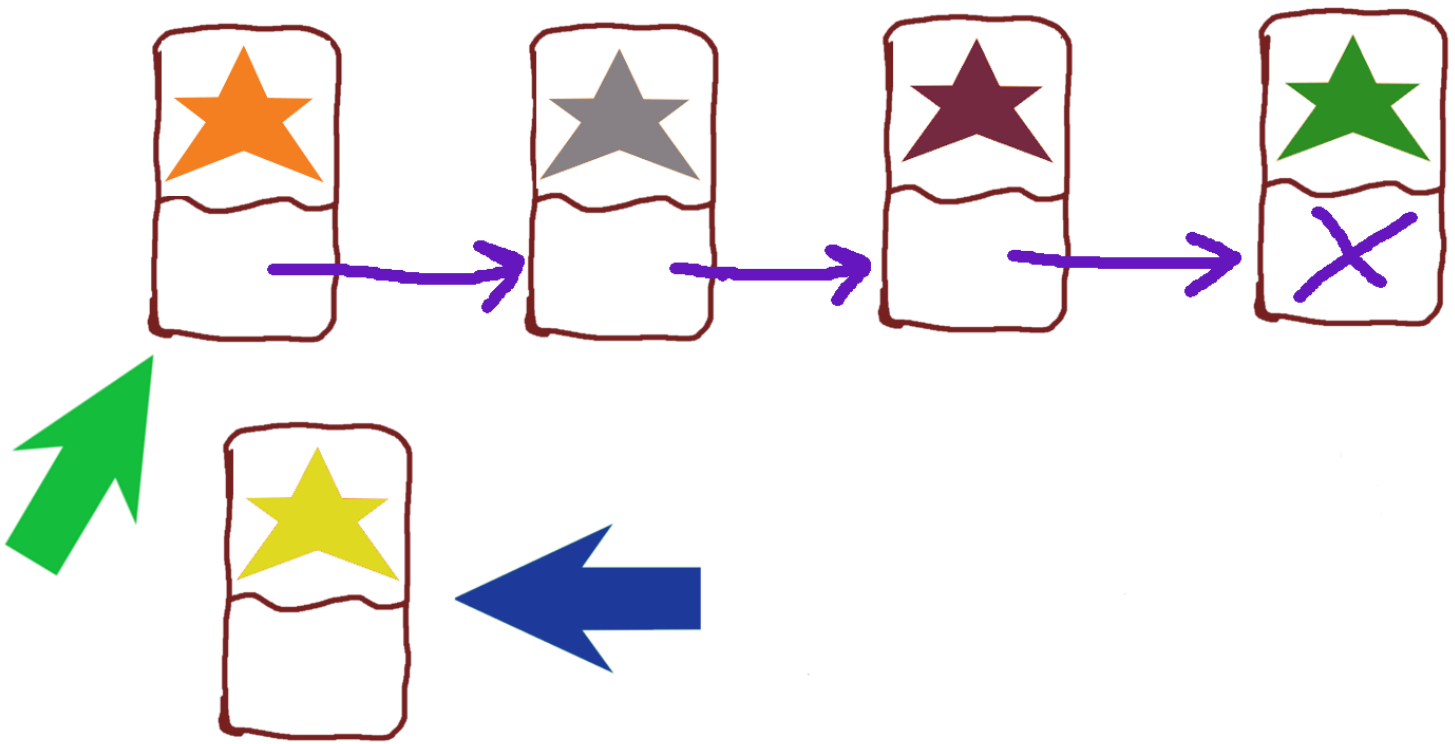
Операции со списком

Вставка на произвольное место: 3



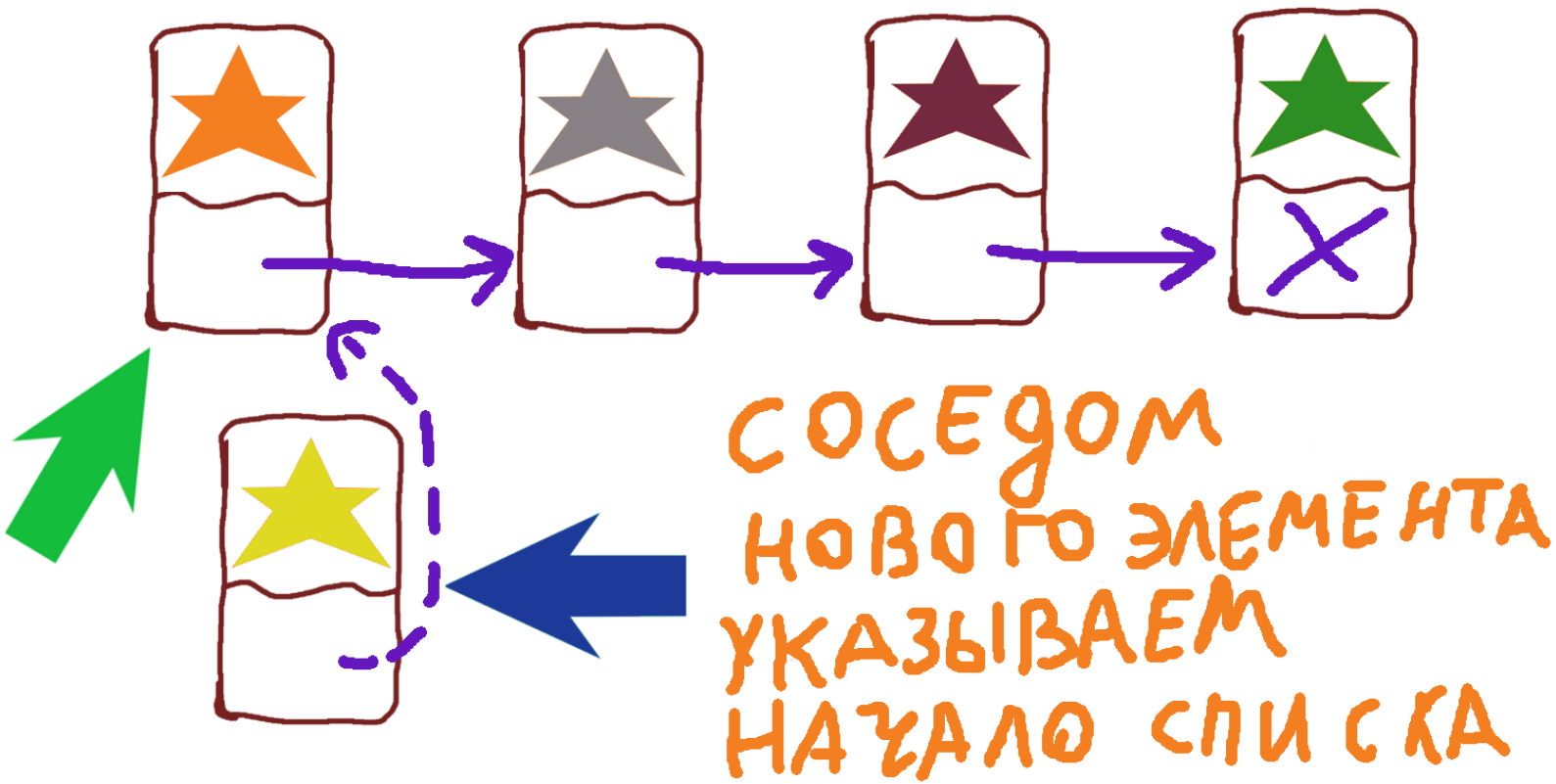
Операции со списком

Вставка на первое место



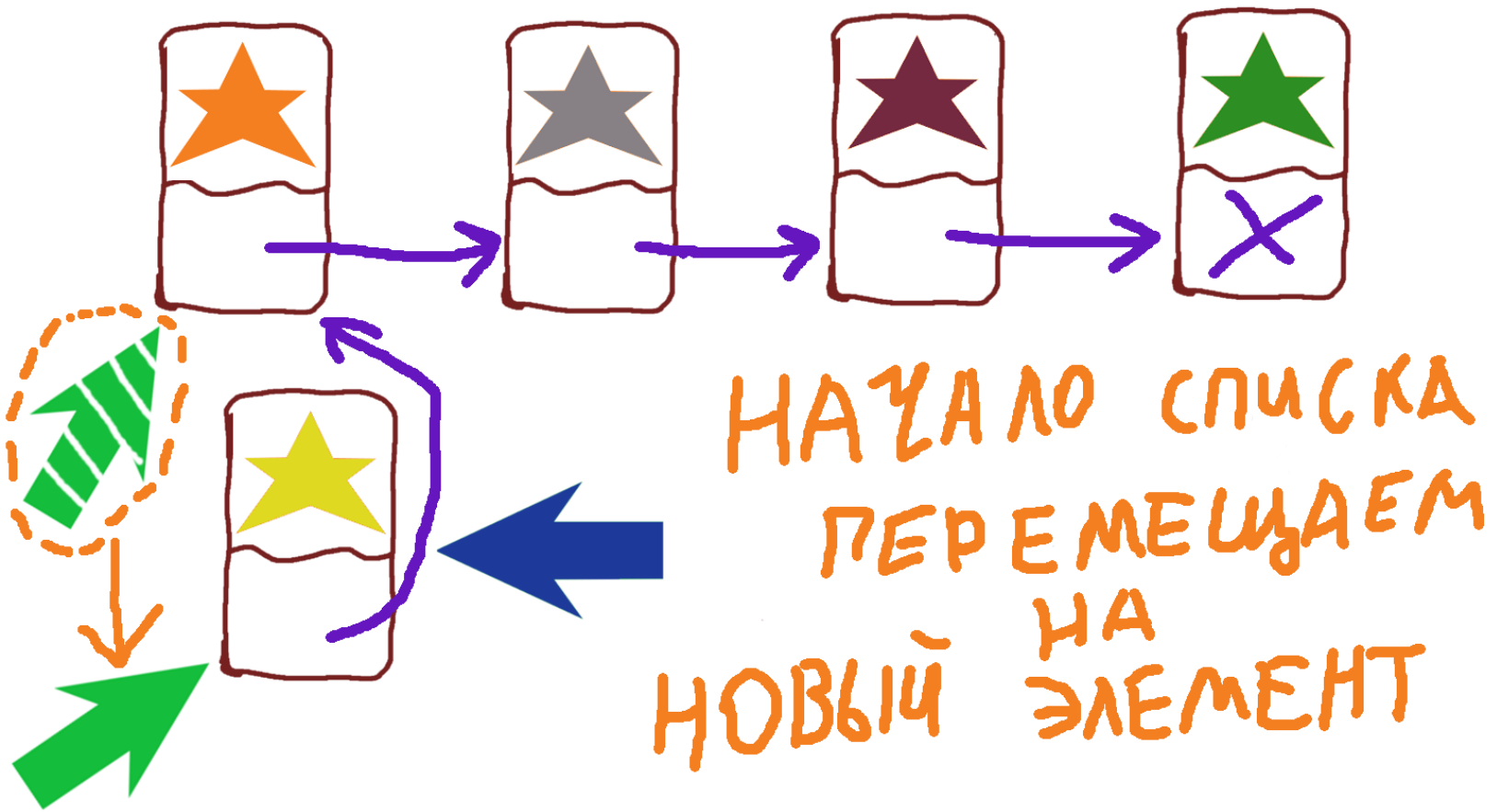
Операции со списком

Вставка на первое место



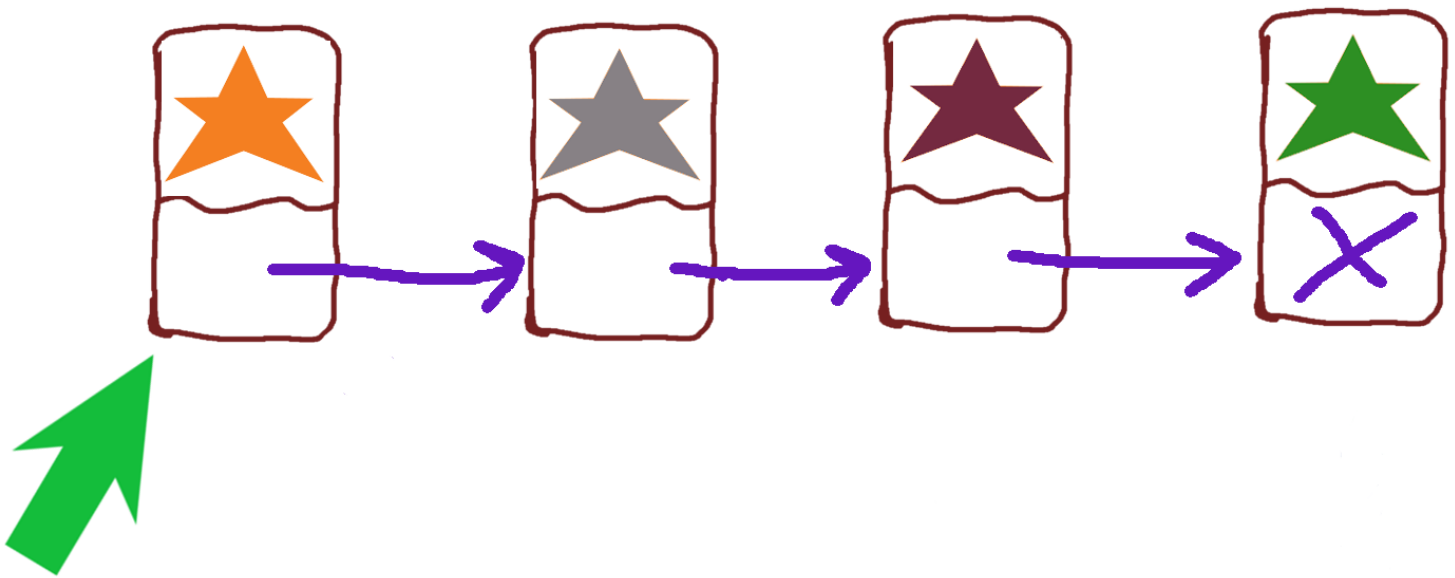
Операции со списком

Вставка на первое место



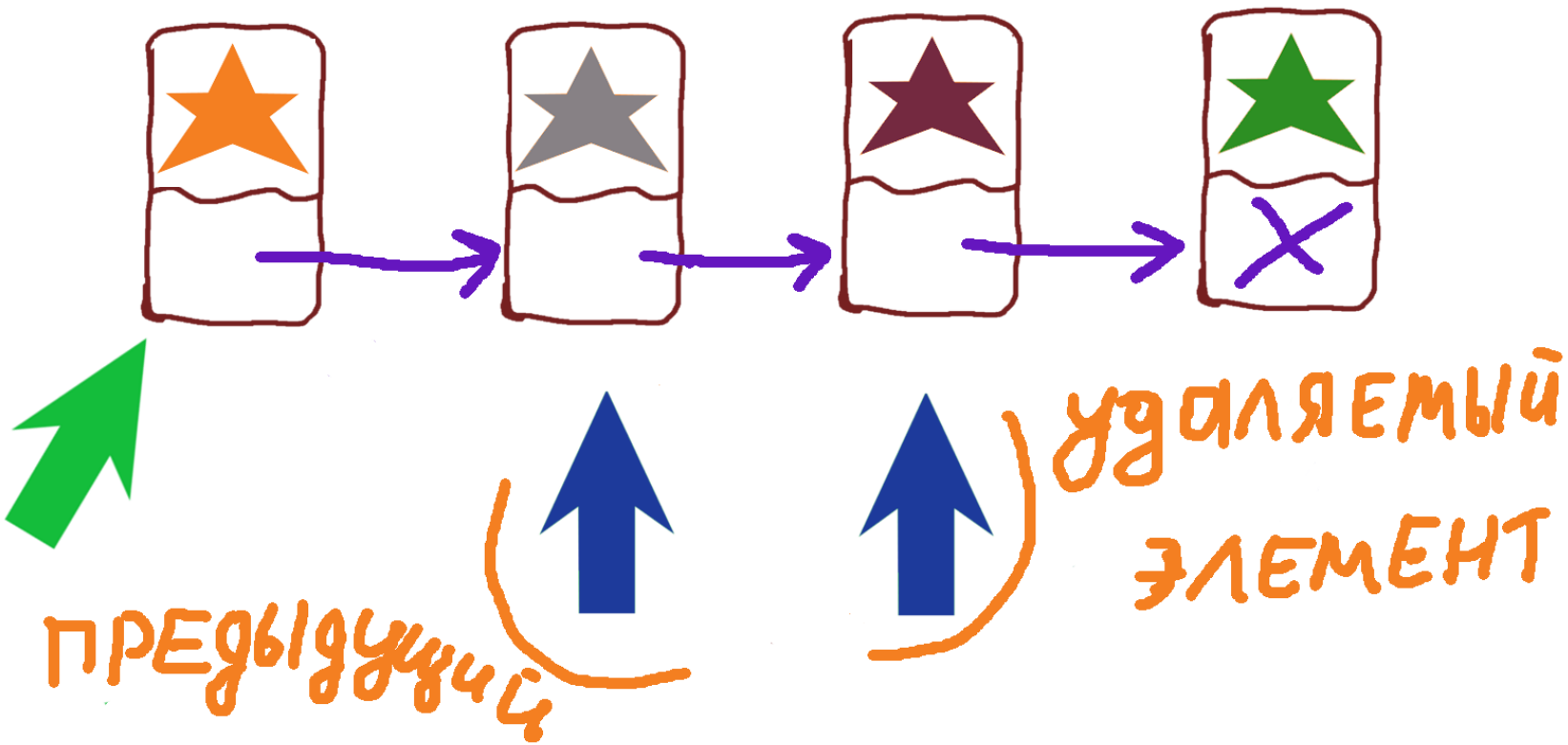
Операции со списком

Удаление элемента: №3



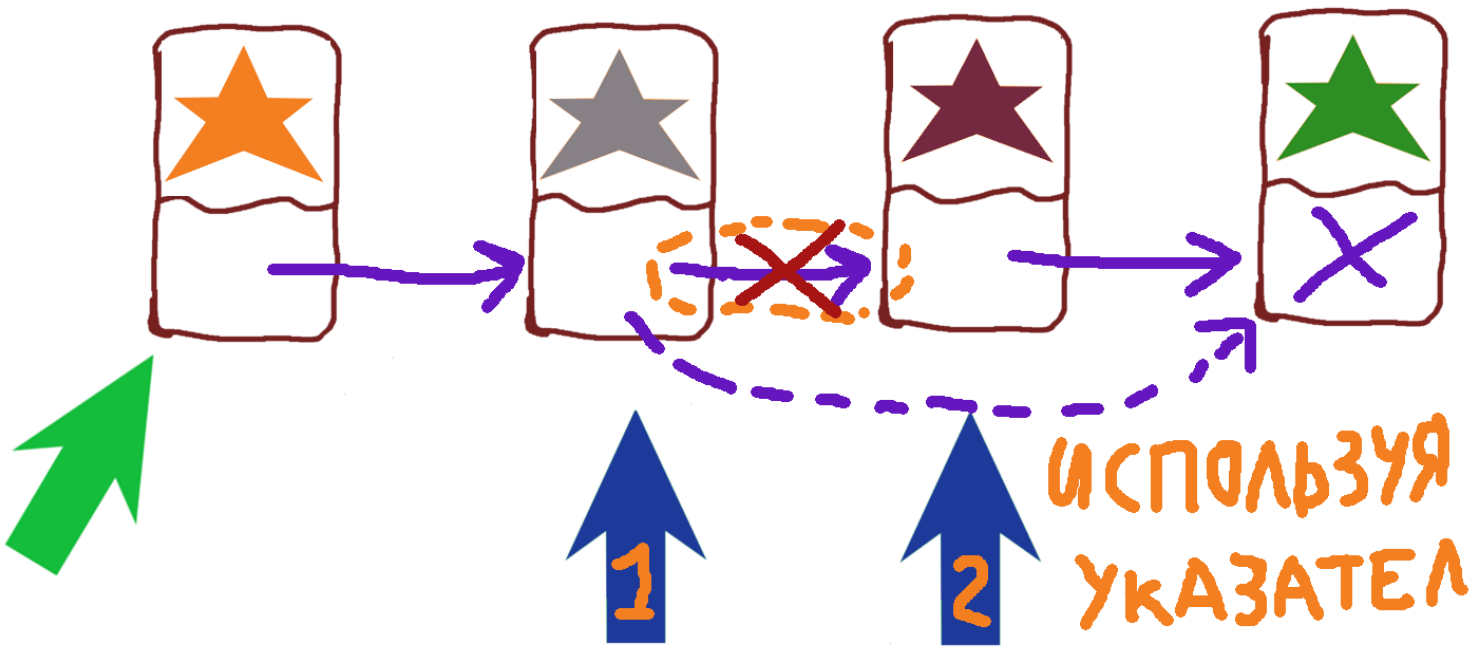
Операции со списком

Удаление элемента: №3



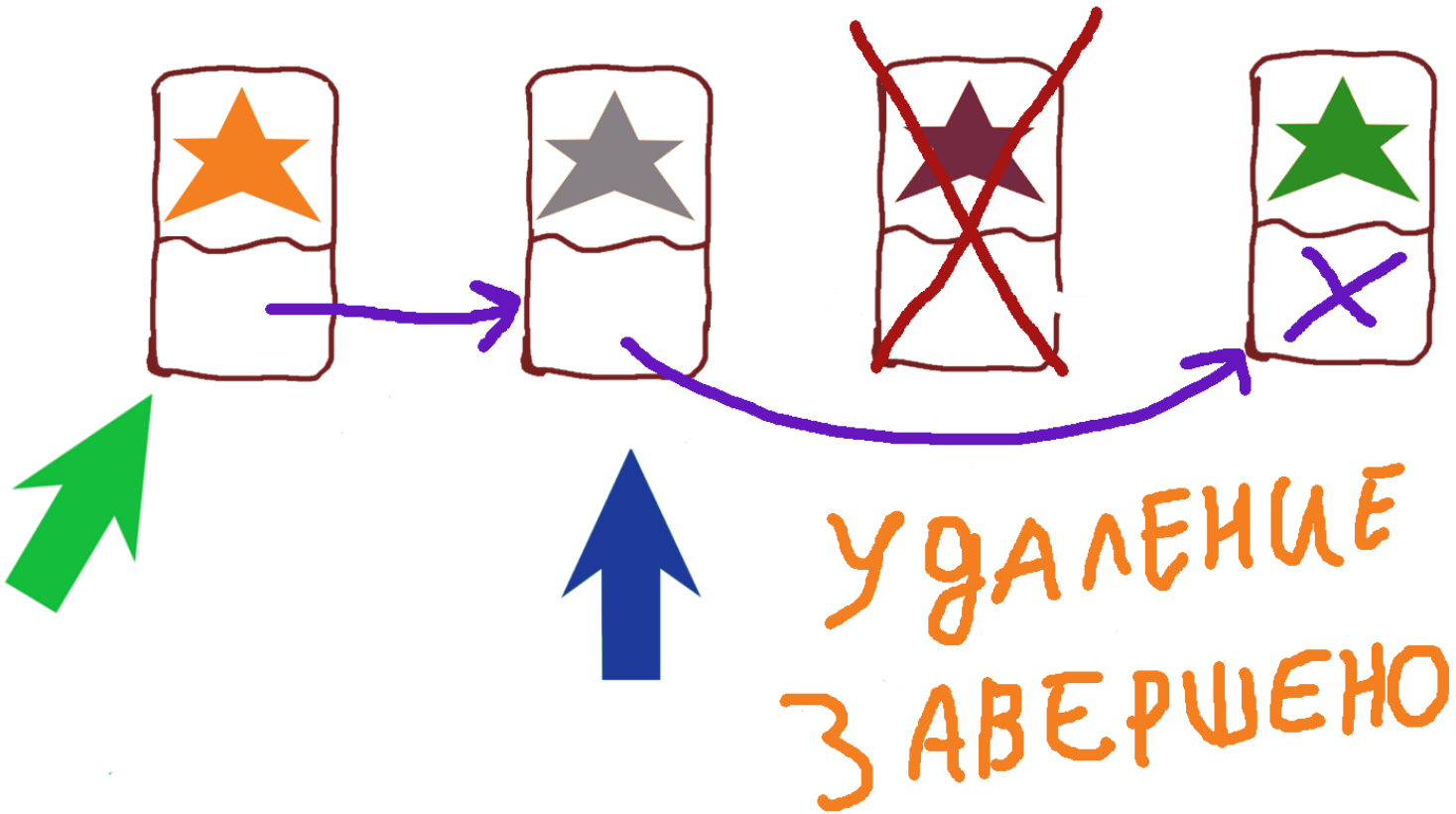
Операции со списком

Удаление элемента №3



Операции со списком

Удаление элемента :№3



Абстрактные типы данных. Очередь

Очередь - является набором элементов, доступ к которым осуществляется по принципу "первый пришёл, последний ушёл"(он же - "first in, last out"или "FILO"). При работе с очередью допустимы следующие две операции:

1. Добавление нового элемента в конец списка (известна на просторах интернета как *enqueue*, а иногда и как *push*)
2. Удаление первого элемента из списка (*dequeue*, реже - *pop*)

Абстрактные типы данных. Стек

Стек - является совокупностью элементов, доступ к которым осуществляется по принципу "первый пришёл, первый ушёл"(он же - "first in, first out"или "FIFO"). Допустимыми являются следующие две операции:

1. Добавление нового элемента в начало списка (*push*)
2. Удаление первого элемента из списка (*pop*)

Список. Пример реализации

В качестве примера далее рассмотрим пример реализации списка на C++, который имеет следующие свойства и действия:

- список является однонаправленным;
- вставка осуществляется на заданную позицию (1, 2, 3, ...). Если позиция превышает размер списка - вставляем элемент как последний;
- удаление также присходит по заданному номеру элемента;
- предоставляется поиск данных элемента по номеру.

Список. Пример реализации

Шаг 1. Выбираем данные, которые будут храниться в каждом элементе списка. Сделаем для них простую структуру.

```
1 struct Star
2 {
3   int height, width;
4   char name[130];
5   char color[40];
6 };
```

Список. Пример реализации

Шаг 2. Описываем элемент списка. Так же используем структуру

```
1 struct Node
2 {
3   Star star;
4
5   // Для хранения следующего элемента списка
6   // используем указатель на объект той же ↩
        структуры
7   Node *next; // <--- !!!
8 };
```

Список. Пример реализации

Шаг 3. Определяем, как будут осуществлятся операции со списком. Выбора тут два - либо делать всё через набор функций (спосов в стиле C), либо использовать определение класса из C++. Несмотря ни на что, второй способ легче использовать, следовательно выбираем его

```
1  class StarList
2  {
3  public:
4    bool insert_elem(Star elem, unsigned position);
5    Star* delete_elem(unsigned position);
6    Star* find_at_position(unsigned position);
7
8  private:
9    Node *begin_list;
10 };
```

Список. Пример реализации

Шаг 4. Написание нужных методов

```
1  bool StarList::insert_elem(Star elem, unsigned position) {
2    if (position == 0) { return false; }
3    if (begin_list == nullptr) { postition = 1; }
4
5    Node *new_node = new(std::nothrow) Node{elem, nullptr};
6    if (new_node == nullptr) { return false; }
7
8    if (position == 1) {
9      new_node->next = begin_list;
10     begin_list = new_node;
11   } else {
12     Node *walker = begin_list;
13     unsigned counter = 1;
14
15     while ( (counter != position - 1) &&  (walker->next != ↩
           nullptr) )
16     { walker = walker->next; ++counter; }
17
18     new_node->next = walker->next;
19     walker->next = new_node;
20   }
21   return true;
22 }
```

Шаг 4. Написание нужных методов

```
1  Star* StarList::delete_elem(unsigned position) {
2    if (position == 0) { return nullptr; }
3
4    Node *walker = begin_list;
5    if (position == 1) {
6      begin_list = walker->next;
7      Star* star_elem = new Star;
8      *star_elem = walker->star;
9      delete walker;
10     return star_elem;
11   } else {
12     unsigned counter = 1;
13
14     while ( (counter != position - 1) &&
15             (walker->next != nullptr) ) {
16       walker = walker->next; ++counter;
17     }
18
19     if (counter == position - 1) {
20       Node *deleted_elem = walker->next;
21       Star* star_elem = new Star;
22       *star_elem = deleted_elem->star;
23
24       walker->next = deleted_elem->next;
25       delete deleted_elem;
26       return star_elem;
27     }
28   }
29
30   return nullptr;
31 }
```

Список. Пример реализации

Шаг 4. Написание нужных методов

```
1  Star* StartList::find_at_position(unsigned position) {
2    if (position == 0) { return nullptr; }
3
4    Node *walker = begin_list;
5    unsigned counter = 1;
6
7    while ( (counter != position) &&
8            (walker->next != nullptr) ) {
9      walker = walker->next; ++counter;
10   }
11
12   if (counter == position) {
13     Star* star_elem = new Star;
14     *star_elem = walker->star;
15
16     return star_elem;
17   }
18
19   return nullptr;
20 }
```

Список. Пример реализации

Шаг 5. Технические дополнения: инициализация нулевым указателем начала списка при создании объекта; автоматическое удаление всех элементов, когда объект больше не нужен

```
1  class StarList
2  {
3  public:
4    StarList() { begin_list = nullptr; }
5    ~StarList() { remove_recursive(begin_list); }
6
7    bool insert_elem(Star elem, unsigned position);
8    Star* delete_elem(unsigned position);
9    Star* find_at_position(unsigned position);
10
11 private:
12   Node *begin_list;
13
14   void remove_recursive(Node *elem);
15 };
```

Список. Пример реализации

Шаг 5. Технические дополнения: реализация рекурсивного удаления списка

```
1  class StarList
2  {
3  public:
4    StarList() { begin_list = nullptr; }
5    ~StarList() { remove_recursive(begin_list); }
6
7    ...
8
9  private:
10   Node *begin_list;
11   void remove_recursive(Node *elem);
12 };
13
14 void StarList::remove_recursive(Node *elem)
15 {
16   if (elem == nullptr) { return; }
17   Node *next_elem = elem->next;
18   delete elem;
19   remove_recursive(next_elem);
20 }
```

Список. Миссия пройдена!

```
1  struct Star
2  {
3    int height, width;
4    char name[130];
5    char color[40];
6  };
7
8  struct Node
9  {
10   Star star;
11   Node *next;
12 };
13
14 class StarList
15 {
16 public:
17   StarList() { begin_list = nullptr; }
18   ~StarList() { remove_recursive(begin_list); }
19
20   bool insert_elem(Star elem, unsigned position);
21   Star* delete_elem(unsigned position);
22   Star* find_at_position(unsigned position);
23
24 private:
25   Node *begin_list;
26   void remove_recursive(Node *elem);
27 };
```

Список. Пример использования

```
1  Star one   = { 4, 6, "Полярная", "зелёный" };
2  Star two   = { 10, 8, "Неизвестная", "красный" };
3  Star three = { 7, 11, "Надежды", "белый" };
4  Star four  = { 3, 2, "Абракадабра", "жёлтый" };
5
6  StarList my_list;
7  my_list.insert_elem(one, 1);
8  my_list.insert_elem(two, 2);
9  my_list.insert_elem(three, 3);
10 my_list.insert_elem(four, 4);
11 my_list.insert_elem(one, 5);
12
13 Star *deleted = my_list.delete_elem(3);
14 if ( deleted != nullptr ) {
15   cout << deleted->name << " была удалена\n";
16 }
17
18 Star *found = my_list.find_at_position(3);
19 if ( found != nullptr ) {
20   cout << "На 3-ей позиции звезда с именем "
21        << found->name;
22 }
```