



**ГУАП**

Государственный университет  
аэрокосмического приборостроения

[www.guap.ru](http://www.guap.ru)

---

# Информатика

**Рождественская Ксения  
Николаевна**

Кафедра 14

[ksu.khramenkova@gmail.com](mailto:ksu.khramenkova@gmail.com)



# Сложные структуры данных: массивы, последовательности, стеки, очереди, деки, деревья

## Данные

- Вычислительные процессы происходят над данными, которые изменяют входные и выходные данные
- Детали работы с данными скрыты в некоторых абстракциях
- Абстракции позволяют опускать детали
  - Это некоторый **интерфейс** для доступа к функциональности сложных объектов
- В программировании абстракции скрывают сложности реализации процесса обработки данных

# Сложные структуры данных

## Данные

- **Абстрактные типы данных** – это подробное описание группы операций, применимых к конкретному типу данных.
  - **Скрывают**: детали хранения данных в памяти и управления ими
- В алгоритмах используем команды, которые описаны для работы с выбранным абстрактным типом данных.

# Сложные структуры данных

## Данные

- **Простота.** Код становится доступнее для понимания и изменения. Проще сосредоточиться на алгоритмах решения задачи
- **Гибкость.** Данные можно представить в виде разных структур, выбирать надо такую, которая лучше всего соответствует задаче. Разные реализации одной и той же структуры хранения данных предлагают одни и те же действия по обработке данных.
- **Повторное использование.** Можно использовать одни и те же структуры для работы с разными данными.
- **Организация.** Иногда требуется создать разные структуры данных для конкретного типа данных, для удобства работы с ним. Это разделение функциональности (часть кода имеет дело с одним и тем же логическим аспектом и должны быть представлены отдельным модулем).

# Сложные структуры данных

## Данные

- **Организация.** Иногда требуется создать разные структуры данных для конкретного типа данных, для удобства работы с ним. Это разделение функциональности (часть кода имеет дело с одним и тем же логическим аспектом и должны быть представлены отдельным модулем).
- **Удобство.** Вы можете использовать готовую структуру данных для работы и последующего создания алгоритма работы программы.
- **Устранение программных ошибок.** При обнаружении ошибки в структуре хранения и обработки данных, вы можете исправить ее единожды, отладить и успешно использовать.

# Сложные структуры данных

## Данные

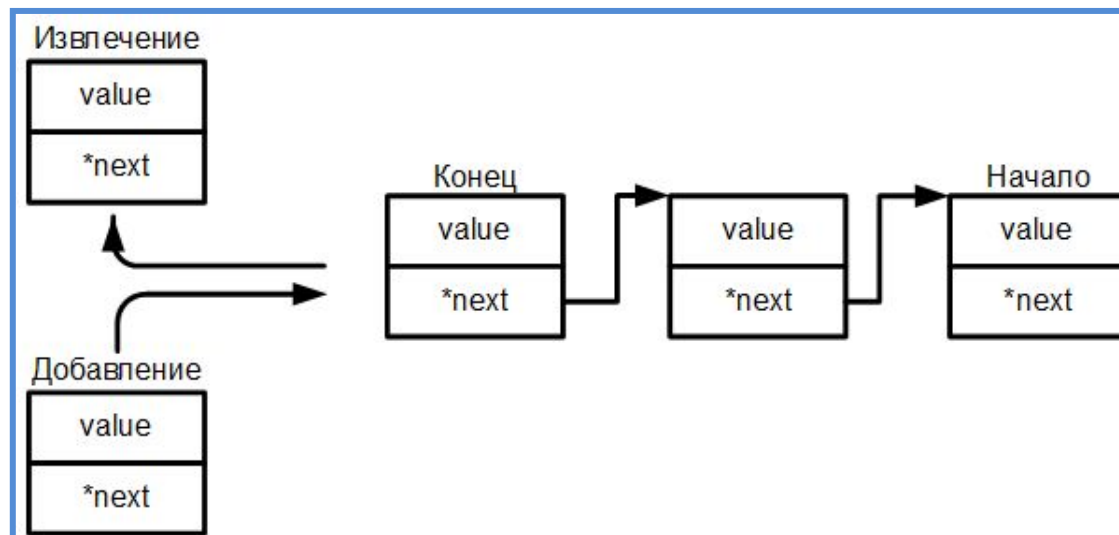
**Примитивные типы данных** – это типы данных со встроенной поддержкой в языке программирования, который вы используете.

Например: целое число, с плавающей точкой, операции с ними (сложение, вычитание и пр.)

# Сложные структуры данных

## Данные. Стек

- **Стек** (stack) позволяет работать только с ее верхним элементом.
- Элемент на вершине стека – это **всегда** элемент, который был добавлен **последним**.

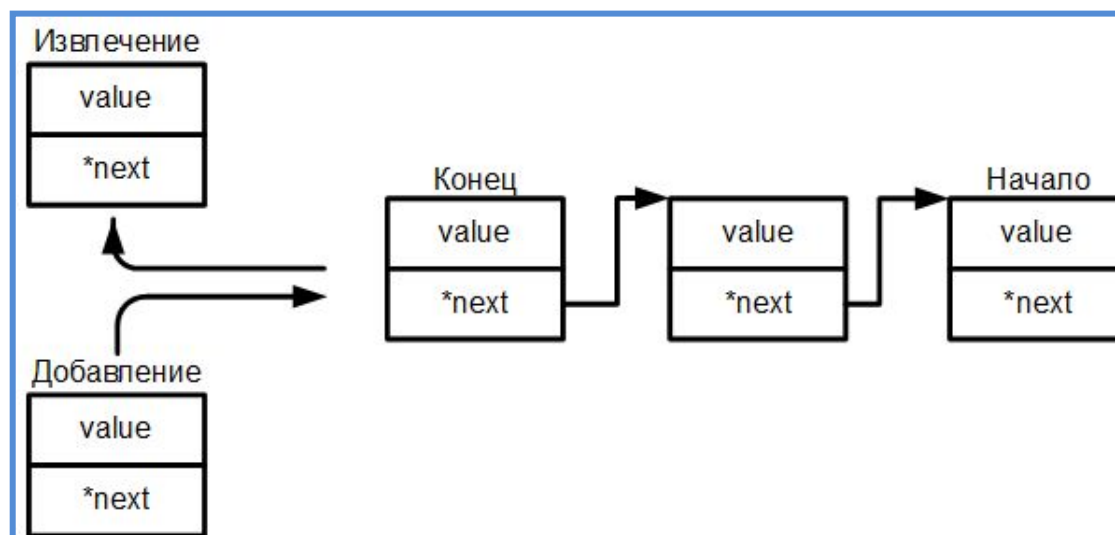


# Сложные структуры данных

## Данные. Стек

- Минимальный набор операций это: добавление и извлечение элемента.
  - Дополнительно может быть: проверка элементов в стеке, получение текущего кол-ва элементов и пр.

**LIFO**  
Last-In, First-Out





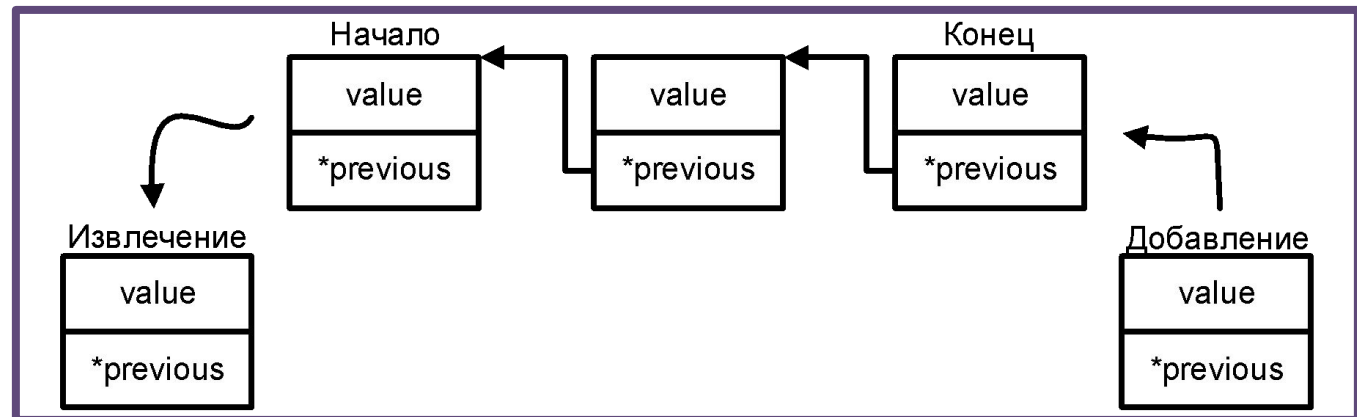
# Сложные структуры данных

## Данные. Очередь

- **Очередь** (queue) позволяет извлекать элементы только из начала очереди, помещать элементы можно только в конец очереди.
- Основные операции: добавить элемент и извлечь элемент.

**FIFO**

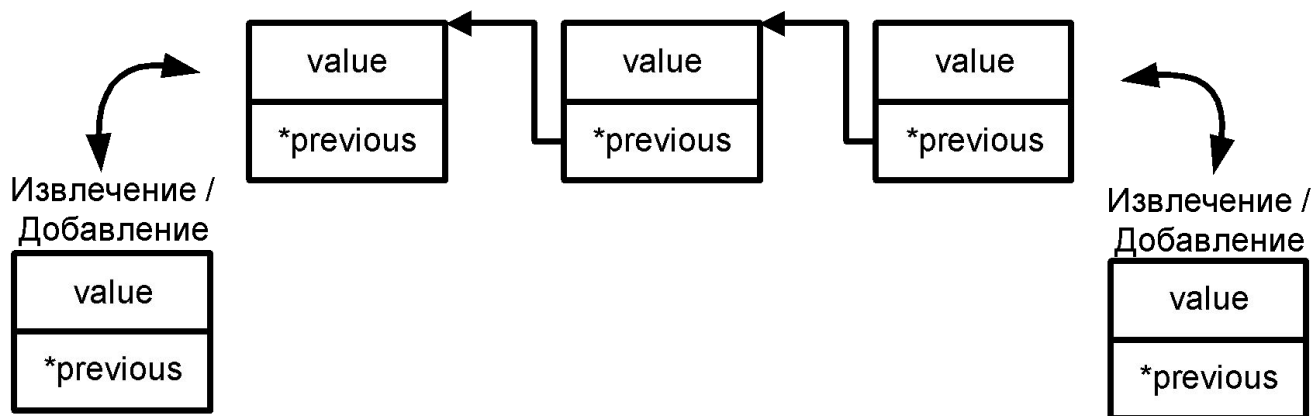
First-In, First-Out



# Сложные структуры данных

## Данные. Дэк

- Дэк (двусторонняя очередь) расширяет поведение обычной очереди. В дек можно извлекать и добавлять элементы, как с начала, так и с конца очереди.

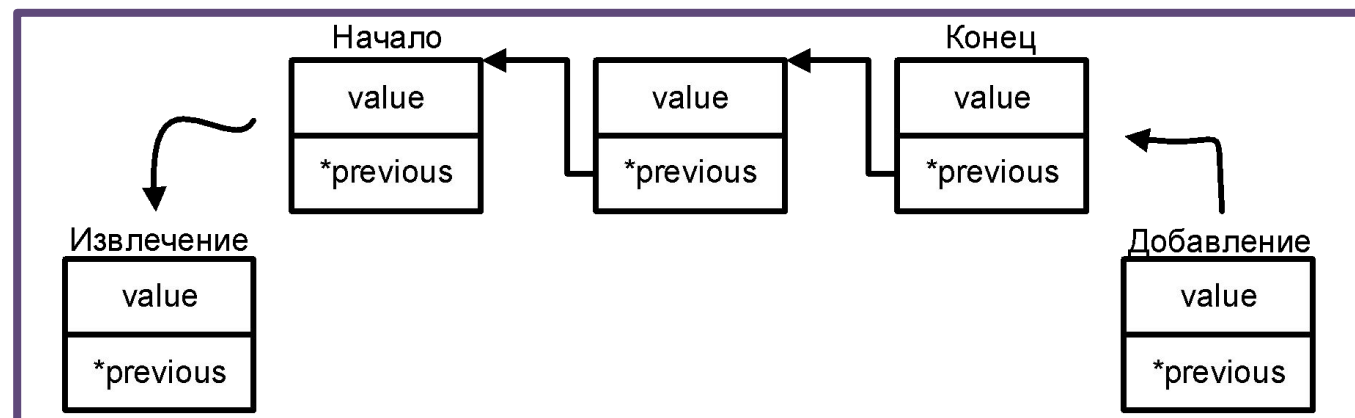


# Сложные структуры данных

## Данные. Очередь с приоритетом

- **Очередь с приоритетом** (priority queue) аналогична обычной очереди с той лишь разницей, что помещенным в нее элементам присваивается **приоритет**. Элементы с высоким приоритетом помещаются в начало очереди, а незначительные добавляются в конец.
- Добавление элемента потребует ввод значение и приоритета для размещения его в такой очереди.

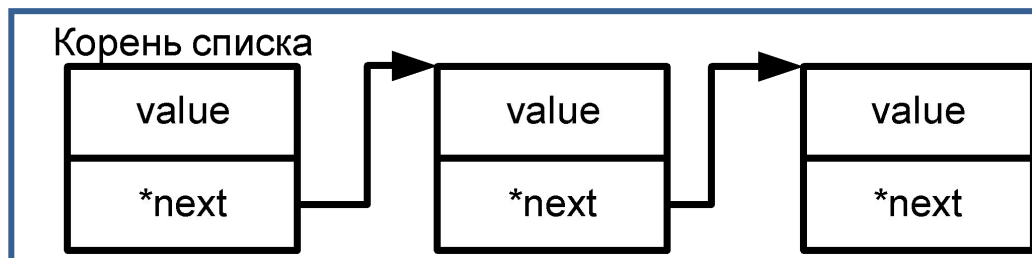
**FIFO**  
First-In, First-Out



# Сложные структуры данных

## Данные. Список

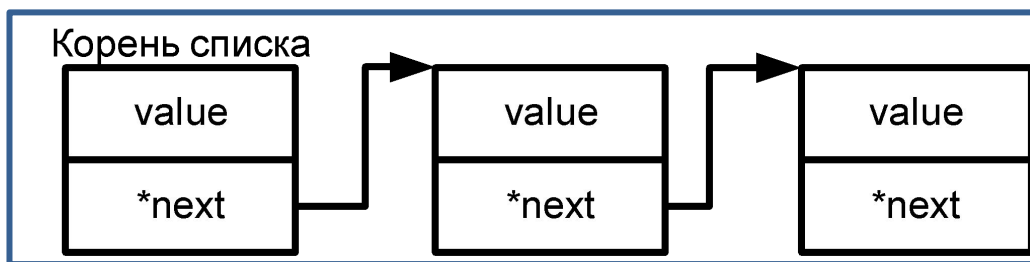
- **Список** (list) позволяет переупорядочивать, извлекать, вставлять, удалять элементы в произвольном порядке.
- Основные операции: вставить элемент и удалить элемент, получить значение элемента, отсортировать элементы, вернуть подсписок, изменить порядок следования элементов (операции требуют указание позиции).



# Сложные структуры данных

## Данные. Сортированный список

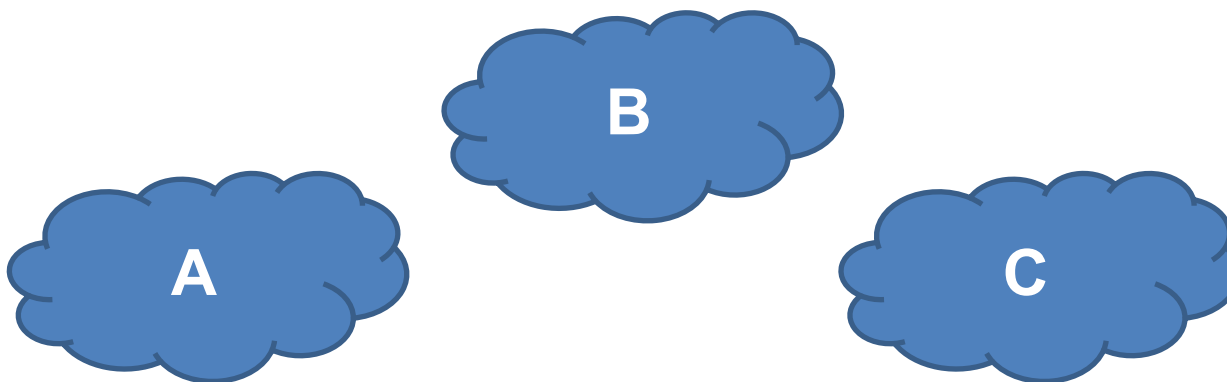
- **Сортированный список** (list) нужен, когда необходима постоянная упорядоченность элементов.
- Основные операции: вставить элемент (позиция автоматически определяется) и удалить элемент, получить значение элемента.



# Сложные структуры данных

## Данные. Множество

- **Множество** (set) представляет неупорядоченные группы **уникальных** элементов.
- Основные операции: добавить элемент (такого элемента быть в множестве еще не должно), перечислить все элементы, удалить элемент.



# Сложные структуры данных

## Структуры

- **Структура данных** описывает как данные организованы и как в них получить доступ в памяти ПК.

**Массив**

**Двусвязный  
список**

**Связный  
список**

**Двоичная  
куча**

**Двоичное дерево  
поиска**

**Дерев  
о**

**Хеш-  
таблица**

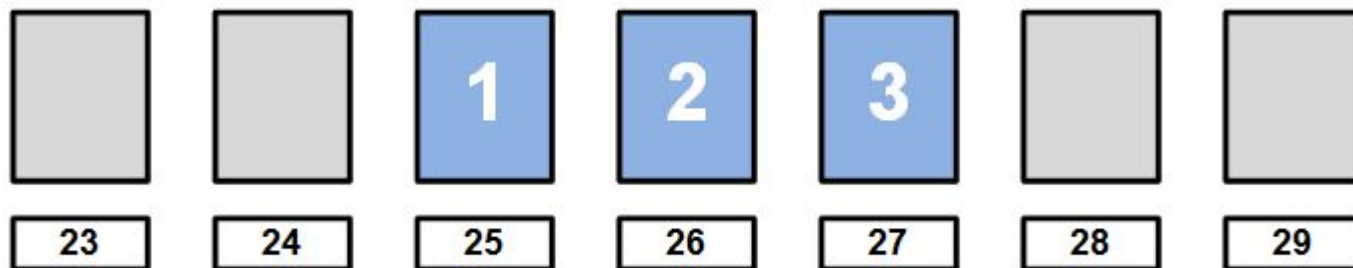
**Гра  
ф**

# Сложные структуры данных

## Структуры. Массив

- **Массив** (array) – самый простой способ хранения набора элементов в памяти компьютера.
- Происходит выделение единого пространства в памяти и последовательной записи элементов.

Содержимое памяти



Адреса ячеек памяти



# Сложные структуры данных

## Структуры. Массив

- Каждый элемент в массиве занимает такой же объем памяти, что и любой другой.
- Можно напрямую обращаться к любому элементу массива.
- Полезен для реализации стека, списков и очередей.

---

### Но

- Может оказаться нецелесообразно выделять большие и непрерывные блоки памяти.
- При использовании динамического массива может оказаться, что рядом достаточной свободной памяти нет.
- Удаление или вставка элемента из середины массива требует сдвига всех последующих элементов на одну позицию.

# Сложные структуры данных

## Структуры. Связный список

- Связный список (linked list) позволяет хранить элементы в цепи ячеек, которые не обязательно должны находиться в последовательных адресах памяти.
- Память выделяется по мере необходимости.
- Каждая ячейка имеет адрес, сообщающий об адресе следующей ячейки в цепи.
- Ячейка с пустым указателем отмечает конец такой цепочки.
- При наращивании списка не возникает никаких проблем: любая ячейка может храниться в любой части памяти.
  - т.е. размер списка ограничен только объемом имеющейся свободной памяти

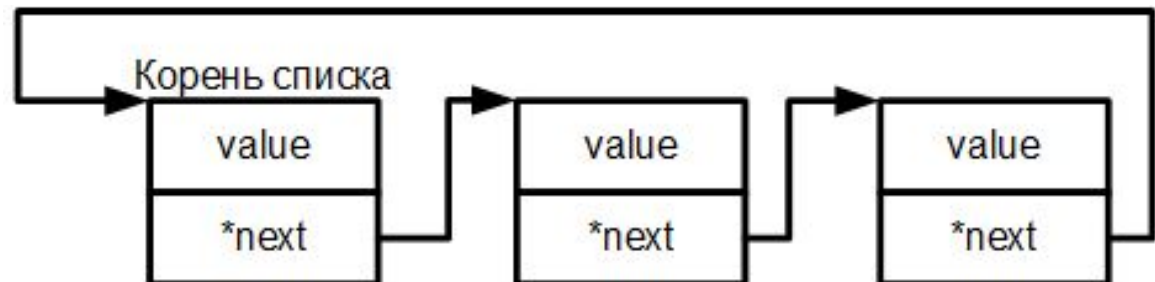
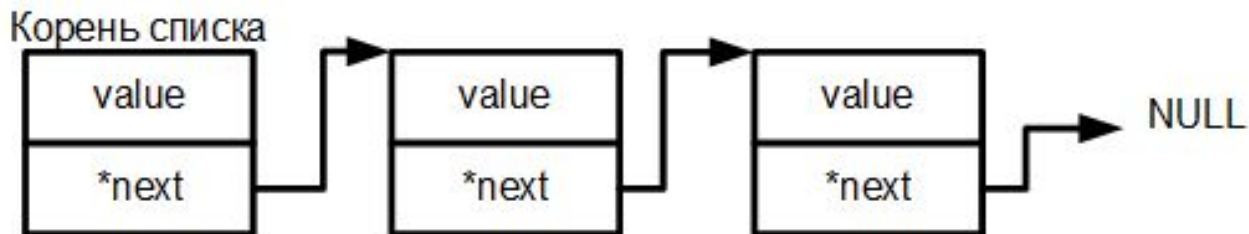
# Сложные структуры данных

## Структуры. Связный список

Но

!

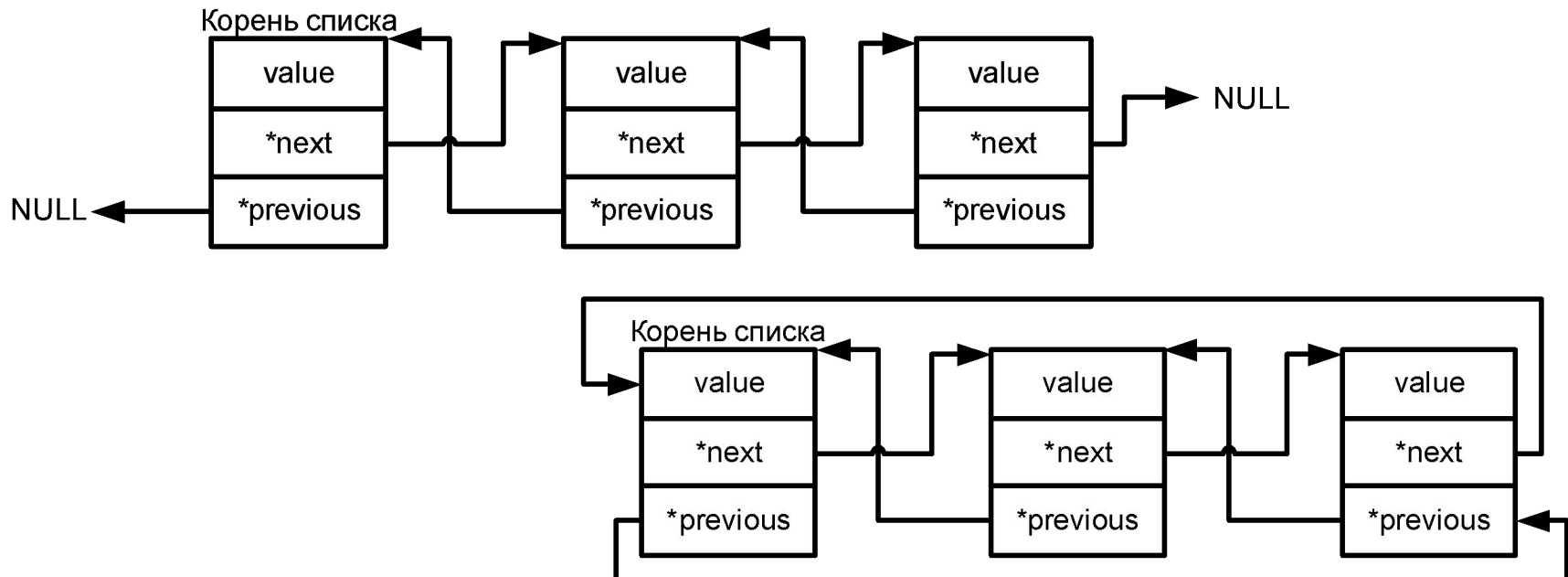
- Не можем получить сразу  $i$ -тый элемент. Нужно последовательно пройти по цепочке элементов до искомого элемента.



# Сложные структуры данных

## Структуры. Двусвязный список

- Двусвязный список (double linked list) – связный список, где ячейки имеют два указателя: на предыдущую ячейку и на следующую.



# Сложные структуры данных

## Структуры. Двусвязный список

- Преимущества те же, что и у связного списка.
- При этом есть возможность передвигаться как «вперед» по списку, так и «назад».
- Но по прежнему сразу же доступ к  $i$ -тому элементу не получить.

# Сложные структуры данных

## Структуры. Массив VS Связный список

- В языках программирования как правило есть библиотеки, которые уже включают в себя реализацию списка, очереди, стека и т.д.
- Но если в языке программирования нет таких возможностей, то:
  - Связные списки предпочтительнее когда: операции вставки и удаления выполнялись быстро; не требуется произвольный доступ к данным; нужно вставлять и удалять элементы внутрь последовательности
  - Массивы предпочтительнее когда: нужен произвольный доступ к данным; нужен быстрый доступ к элементам

# Сложные структуры данных

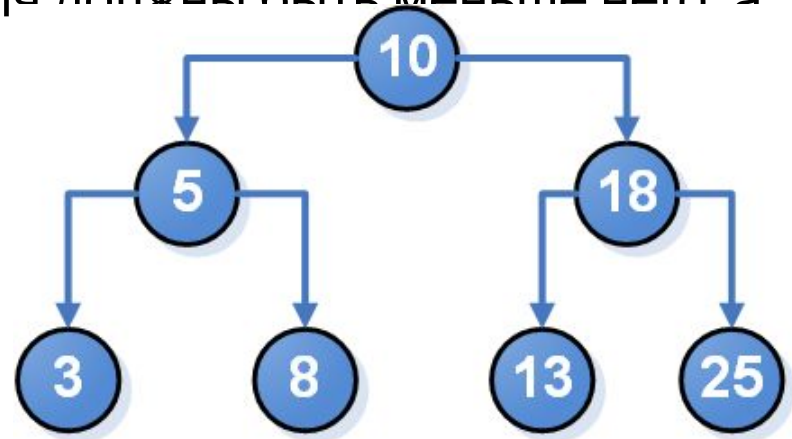
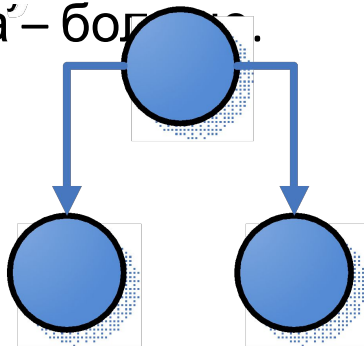
## Структуры. Дерево

- **Дерево** (tree) использует элементы, которым для хранения объектов не нужно располагаться в физической памяти непрерывно.
- Деревья удобны для иерархических данных.
- Ячейка называется **узлом**, указатель из одной ячейки на другую – **ребром**. Самая первая ячейка – это **корневой узел**.
- Узлы, не имеющие дочерних узлов – **листья**.
- **Путь** между двумя узлами определяется множеством узлов и ребер.
- **Уровень узла** – это длина пути от узла до корневого узла в дереве.
- Множество деревьев называется **лесом**.

# Сложные структуры данных

## Структуры. Двоичное дерево поиска

- **Двоичное дерево поиска** (binary search tree) – тип дерева, поиск в котором выполняется особенно эффективно.
- Узлы в двоичном дереве могут иметь не более двух дочерних узлов.
- Дочерние узлы слева от родителя должны быть меньше него, а справа – больше.



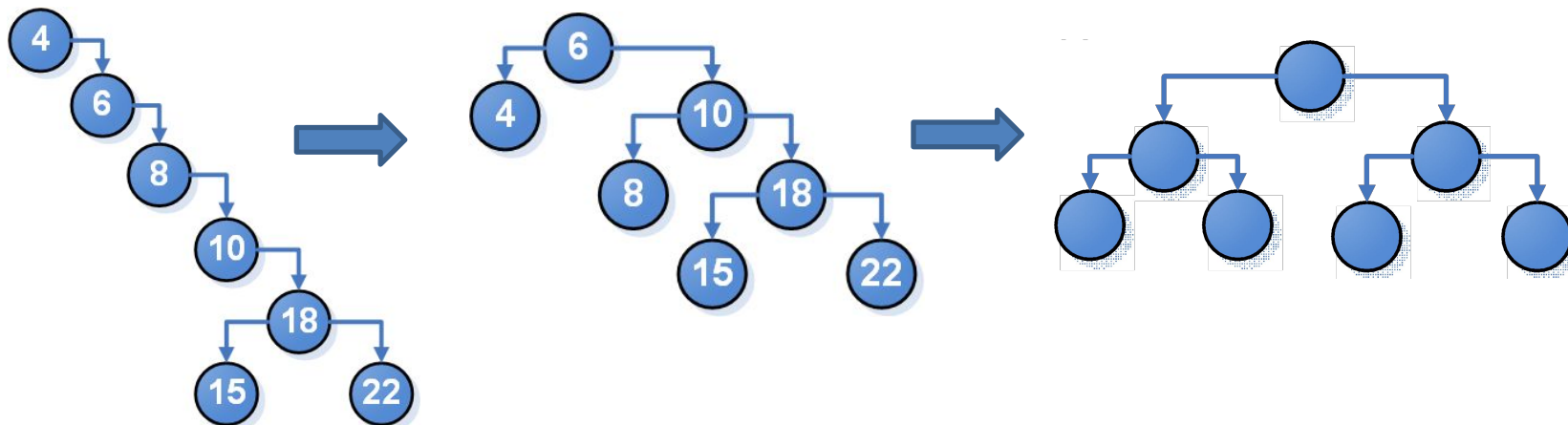


# Сложные структуры данных

## Структуры. Двоичное дерево поиска

### Балансировка дерева

- Путем добавления элементов можно получить некое подобие связного списка.
- **Но!** Такое дерево можно перестроить. Эта процедура называется **балансировкой дерева**.

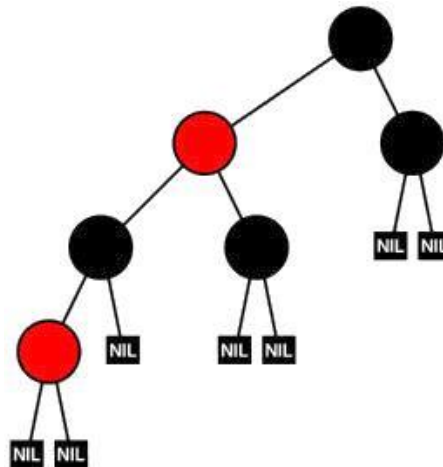


# Сложные структуры данных

## Структуры. Двоичное дерево поиска

### Сбалансированные двоичные деревья

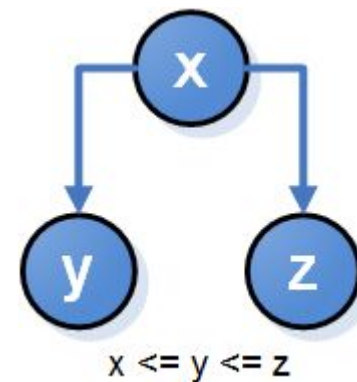
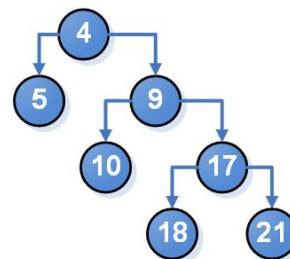
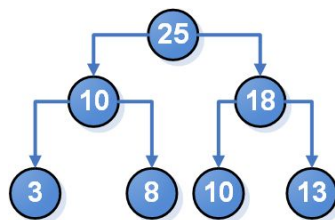
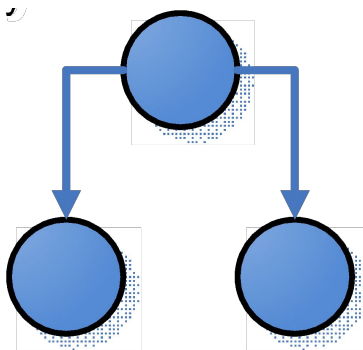
- Процедуры вставки или удаления элементов гарантируют, что дерево остается сбалансированным.
- AVL-дерево, Красно-черное дерево – примеры сбалансированных деревьев.



# Сложные структуры данных

## Структуры. Двоичная куча

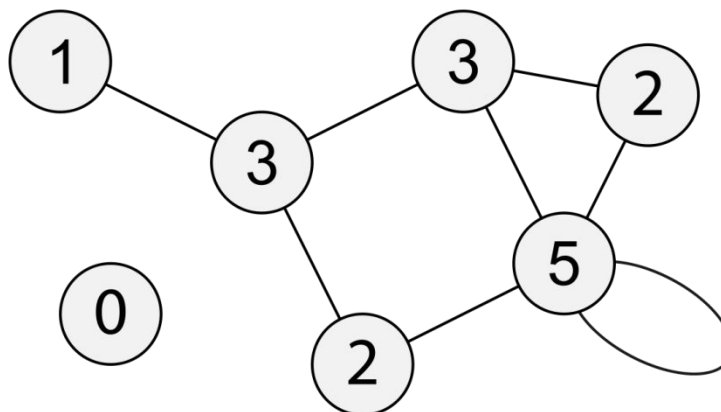
- **Двоичная куча** (binary heap) - особый тип двоичного дерева поиска, в котором можно мгновенной найти самый маленький (или самый большой) элемент.
- Правила размещения элементов те же, что и двоичные деревья поиска, но есть одно ограничение: **родительский узел должен быть меньше (либо больше) обоих своих дочерних узлов.**



# Сложные структуры данных

## Структуры. Граф

- **Граф** (graph) аналогичен дереву.
- Данные организованы в виде узлов (вершин) и дуг (ребер) так, что любой узел может иметь произвольное число входящих и исходящий ребер.



# Сложные структуры данных

## Структуры. Хеш-таблица

- **Хеш-таблица** (hash table) – структура данных, которая позволяет находить элементы со сложностью алгоритма  $O(1)$ .
- Требуется предварительное выделение большого блока последовательной памяти. Но в отличие от массива, позиция элемента определяется **хеш-функцией**.
- Хеш-функция – это специальная функция, которая на входе получает данные, предназначенные для хранения, и возвращает число, которое интерпретируется как позиция в памяти, куда будет помещен элемент.
- **НО!** Иногда хеш-функция возвращает одинаковую позицию для разных входных данных. Это **хеш-коллизия**. В этом случае необходимо такие элементы должны быть сохранены в одной позиции хеш-таблицы.