Алгоритмы

Кафедра ИТУС ТУ

Исаева Г.Н.

Понятие алгоритма

Под **алгоритмом** понимают постоянное и точное предписание (указание) исполнителю совершить определенную последовательность действий, направленных на достижение указанной цели или решение поставленной задачи.

История алгоритма

- Слово алгоритм происходит от *algorithmi* – латинской формы написания имени великого математика IX в. Аль Хорезми, который сформулировал правила выполнения арифметических действий.
- В дальнейшем это понятие стали использовать для обозначения последовательности действий, приводящих к решению поставленной задачи.

Исполнитель алгоритма - это абстрактная или реальная (техническая, биологическая или биотехническая) система, способная выполнять действия,

Характеристики исполнителя

- среда;
- элементарные действия;
- система команд;
- отказы.

Характеристики исполнителя

- **Среда** (или обстановка) - это «место обитания» исполнителя.
- Каждый исполнитель может выполнять команды только из некоторого строго заданного списка - **системы команд** исполнителя.

Характеристики исполнителя

- После вызова команды исполнитель совершает соответствующее **элементарное действие.**
- **Отказы** исполнителя возникают, если команда вызывается при недопустимом для нее состоянии среды.

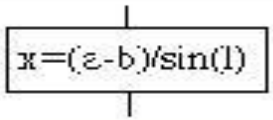
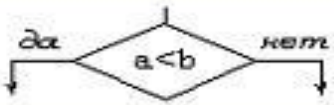
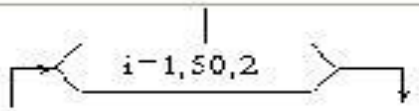
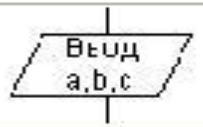
Способы записи алгоритмов

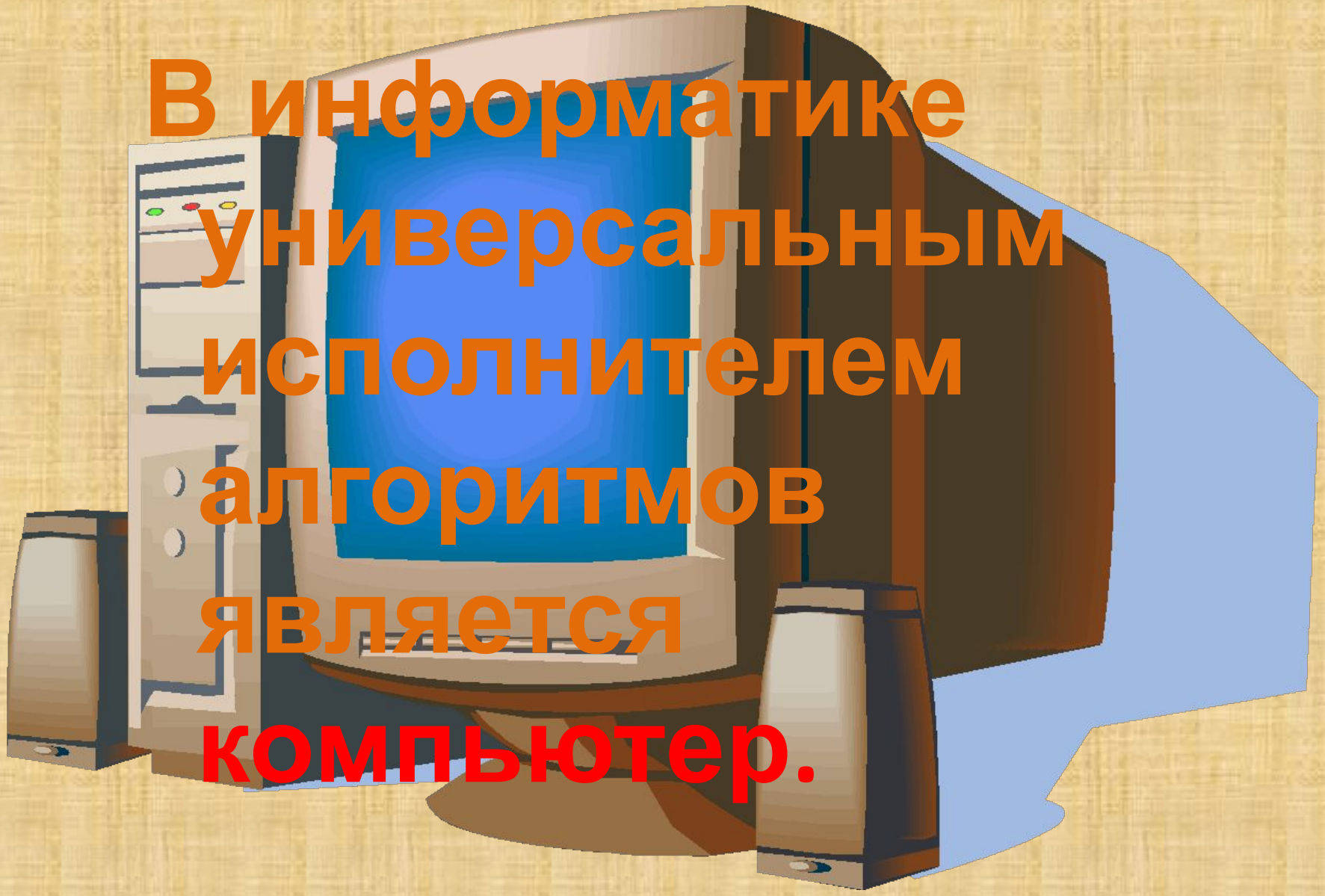
- словесный, (недостаток– многословность, возможна неоднозначность–«он встретил ее на поле с цветами»)
- графический (блок-схемы)
- на псевдокоде
- с помощью языка программирования (программа)

Свойства алгоритма

- Дискретность. Последовательное выполнение простых шагов
- Определенность. Каждое правило алгоритма должно быть четким, однозначным.
- Результативность. Алгоритм должен приводить к решению за конечное число шагов.
- Массовость. Применим для некоторого класса задач, различающихся лишь исходными данными.
- Правильность. Алгоритм правильный, если его выполнение дает правильные результаты решения поставленной задачи.

Графический способ записи алгоритма

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условий
Модификация		Начало цикла
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод-вывод		Ввод-вывод в общем виде
Пуск-останов		Начало, конец алгоритма, вход и выход в подпрограмму
Документ		Вывод результатов на печать



В информатике универсальным исполнителем алгоритмов является компьютер.

Этапы решения задач на ЭВМ

Постановка задачи;

Построение модели (математическая формализация);

Построение алгоритма;

Составление программы на языке программирования;

Отладка и тестирование программы;

Анализ полученных результатов

Этапы решения задач на ЭВМ

Технологическая цепочка решения задачи на ЭВМ предусматривает возможность возвратов на предыдущие этапы после анализа полученных результатов

Базовые алгоритмические структуры

Алгоритмы можно представить как некоторые структуры , состоящие из отдельных базовых (основных) элементов:

Следование

Ветвление

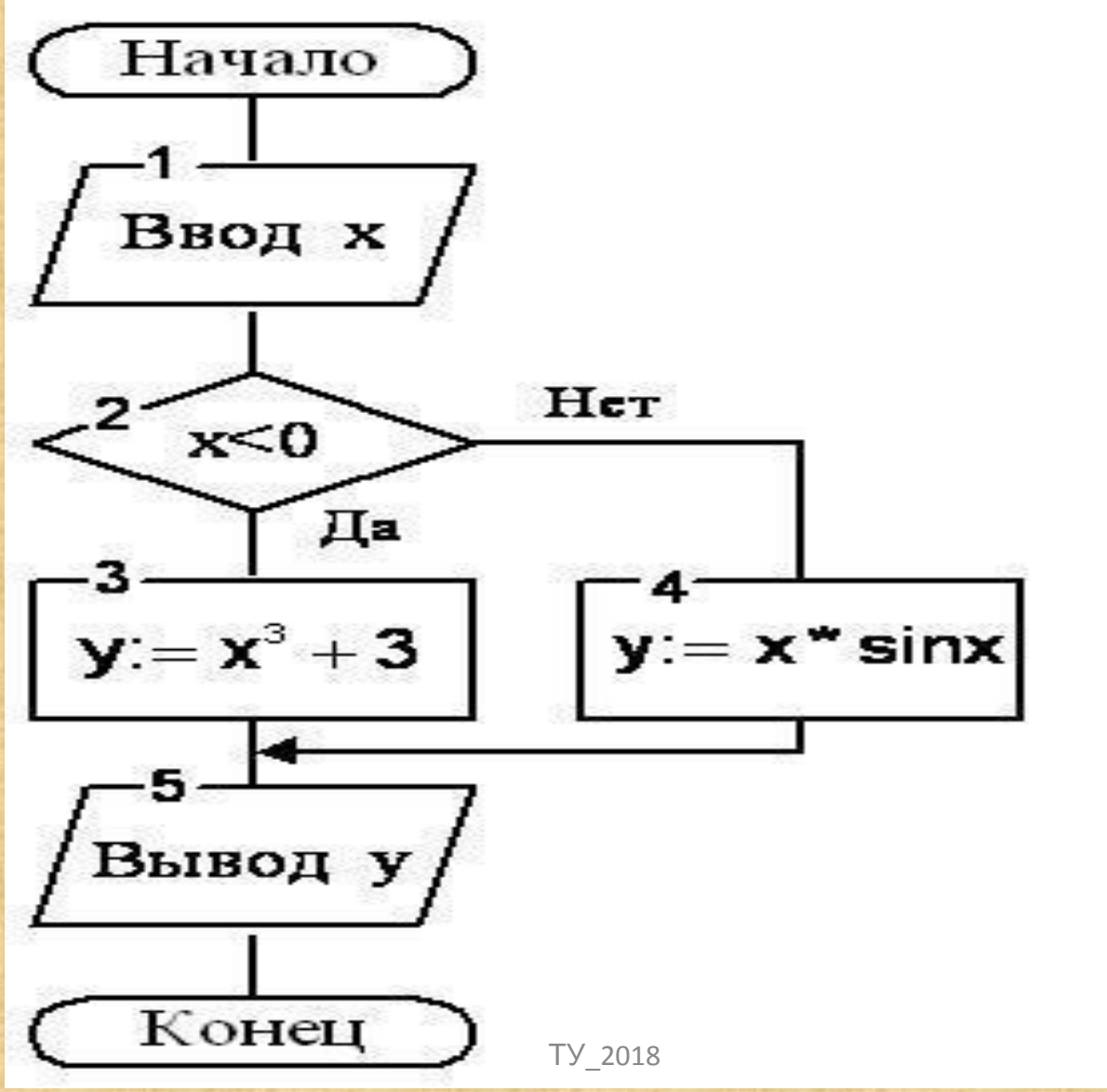
Цикл

РАЗВЕТВЛЯЮЩИЯСЯ ПРОГРАММЫ

Требуется решить систему неравенств:

$$y = \begin{cases} x^3 + 3, \text{если } x > 0, \\ x * \sin x, \text{если } x \le 0. \end{cases}$$

Блок-схема решения задачи



УСЛОВНЫЙ ОПЕРАТОР

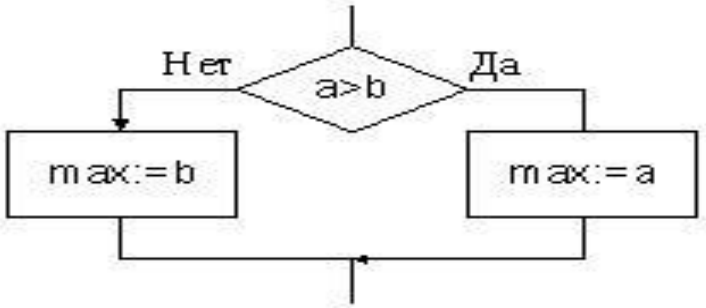
Условный оператор служит для ветвлений в программе и имеет следующий синтаксис:

if <условие> then <оператор1> else <оператор2>.

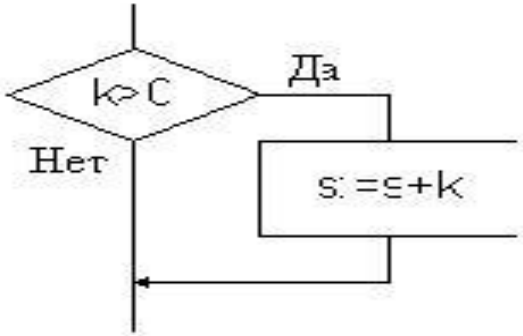
Здесь *if, then, else* – ключевые слова (перев. с *англ.* если, то, иначе соответственно);

<условие> – логическое выражение типа сравнения (например, *a>b, c<=d, f=1*),

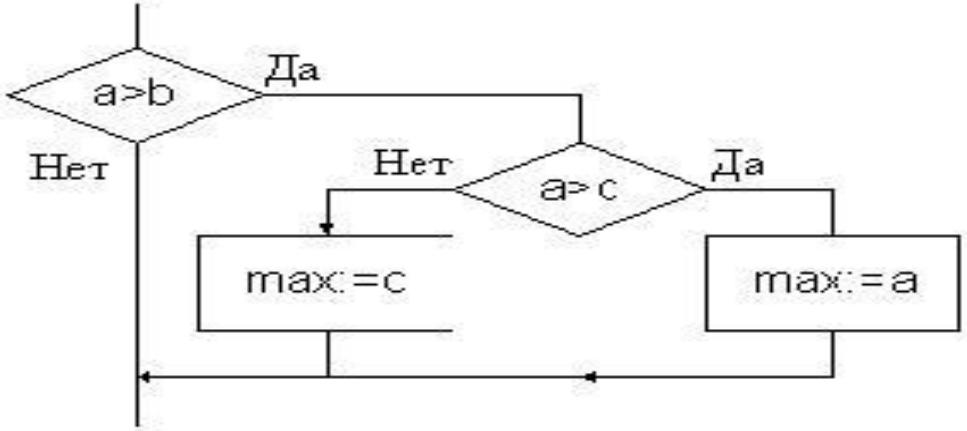
Фрагменты схем алгоритмов




```
if a>b then
    max:=a
  else
    max:=b;
```




```
if k>0 then
    s:=s+k;
```




```
if a>b then
  if a>c then
    max:=a
    else max:=c;
```

Данные и типы данных

Данные — это любая информа*ция*, представленная в формализованном виде и пригодная для обработки алгоритмом.

Данные делятся на переменные и константы.

Переменные — это такие данные, значения которых могут изменяться в процессе выполнения алгоритма.

Константы — это данные, значения которых не меняются в процессе выполнения алгоритма.

Каждая переменная и константа должна иметь свое уникальное *имя. Имена переменных и констант задаются идентификаторами.*

Идентификатор (по определению) представляет собой последовательность букв и цифр.

Тип данных

это множество допустимых значений, которые может принимать тот или иной *объект*, а также множество допустимых операций, которые применимы к нему

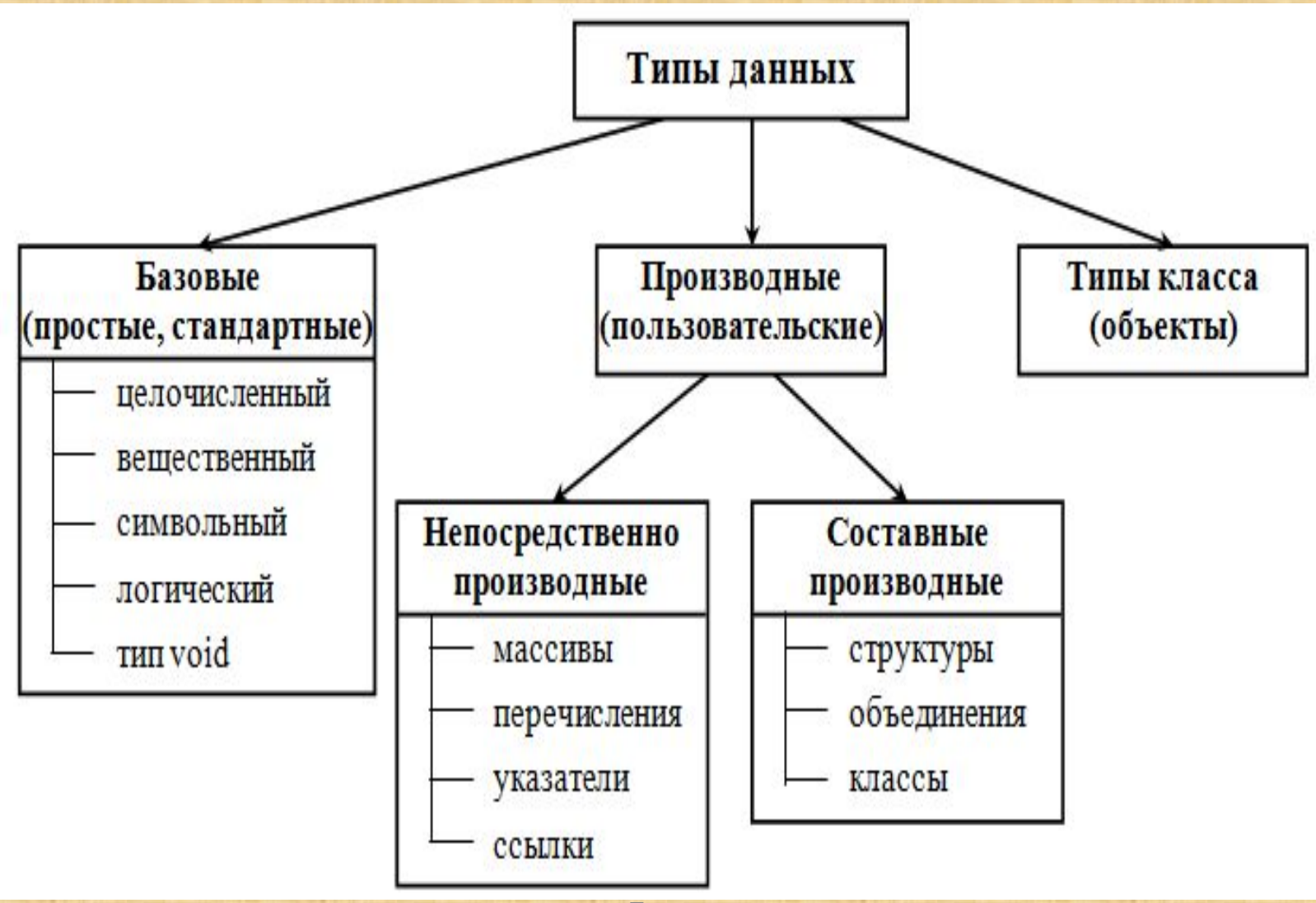
Тип данных

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- объем памяти, выделяемый под данные;
- множество (диапазон) значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к данным этого типа.

Классификация типов данных

Типы данных Си++



Множества

В Турбо-Паскале множества – это наборы однотипных объектов, каким-либо образом связанных друг с другом. Характер связей между объектами подразумевается программистом. Максимальное количество объектов в множестве – 256.

Определение множества производится в два этапа. Сначала определяется базовый для него тип, а затем с помощью оборота *set of* – само

```
•type
•  digch='0'..'9';
•  digitch = set of digch;
•  dig= 0..9;
•  digit = set of dig;
•  sport=(football,hockey,tennis,rugby);
•  hobby=set of sport;
• var s1,s2,s3:digitch;
•     s4,s5,s6:digit;
•     hobby1:hobby;
```

```
•begin
•s1:=['1','2','3'];
•s2:=['3','2','1'];
•s3:=['2','3'];
•s4:=[0..3,6];
•s5:=[4,4];
• s6:=[3..9];
• hobby1:=[football,hockey,tennis,rugby];
• if tennis in hobby1 then writeln('Теннис!');
• end
```

Файлы

Под файлом понимается именованная область внешней памяти или логическое устройство – потенциальный источник или приемник информации

Любой сколько-нибудь развитый язык программирования должен содержать средства для организации хранения информации на внешних запоминающих устройствах и доступа к этой информации

Характерные особенности ФАЙЛОВ

1) у файла есть имя, это дает возможность работать с несколькими файлами одновременно;
2) содержит компоненты одного типа (типом может быть любой тип, кроме файлового);
3) длина вновь создаваемого файла никак не ограничена при объявлении и ограничивается лишь емкостью внешних устройств памяти.

Классификация файлов

В зависимости от способа описания можно выделить:

- текстовые *(text)* файлы
- компонентные (двоичные или типизированные)(*file of*)
- бестиповой (нетипизированные) (*file*).

Вид файла определяет способ хранения информации в файле.

Обращение к файлу производится через файловую переменную:

type

< имя > = file of < тип >;

< имя > = text;

< имя > = file;

где < имя > – имя файлового типа или файловой переменной (правильный идентификатор);

file, of, text – ключевые слова

< тип > – любой тип языка Турбо-Паскаль, кроме файлового.

Последовательный доступ

- Текстовый файл является файлом последовательного доступа, и его можно представить как набор строк произвольной длины.
- Последовательный файл отличается от файлов с другой организацией тем, что чтение (или запись) из файла (в файл) ведутся байт за байтом от начала к концу.

Прямой доступ

Типизированные файлы содержат компоненты строго постоянной длины, что дает возможность организовать прямой доступ к каждому компоненту. Для этой цели служит встроенная процедура *seek*:

seek(<ф.п.>,<n компонента>)

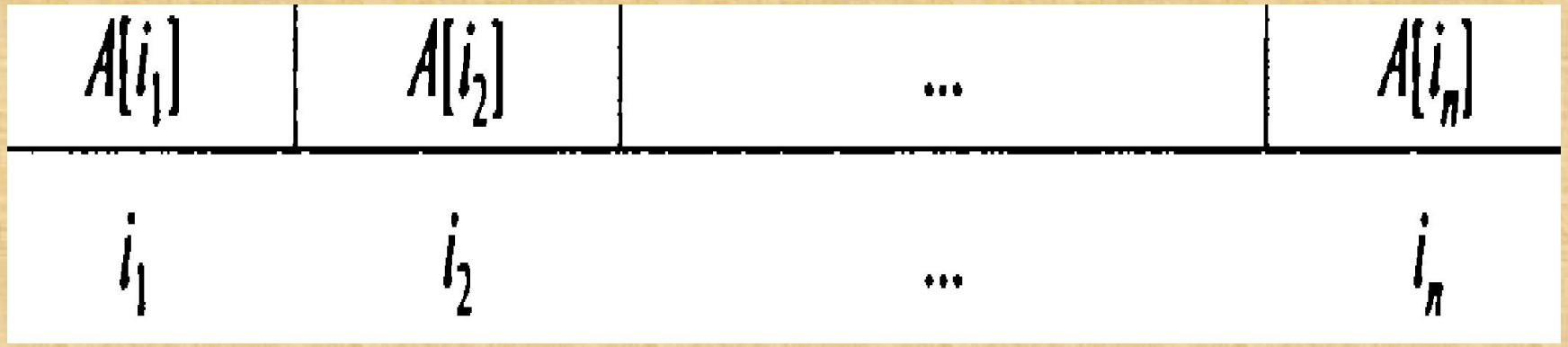
Здесь *<n компонента>* – выражение типа *longint*, указывающее номер компонента.

Организация ввода-вывода при прямом доступе

- Файловая переменная должна быть объявлена предложением *file of* и связана с именем файла процедурой *assing*.
- Файл необходимо открыть процедурой *rewrite* или *reset*.
- Для чтения и записи в типизированный файл используются известные процедуры *read* и *write*.

Массивы

Массив представляет собой некоторое количество расположенных в определенном порядке элементов одного типа.
Индекс предназначен для обеспечения возможности указания на элементы массива.



Массивы

Алгоритм сортировки пузырьком сводится к повторению проходов по элементам сортируемого массива.

Проход по элементам массива выполняет внутренний цикл. За каждый проход сравниваются два соседних элемента, и если порядок неверный элементы меняются местами.

Внешний цикл будет работать до тех пор, пока массив не будет отсортирован.

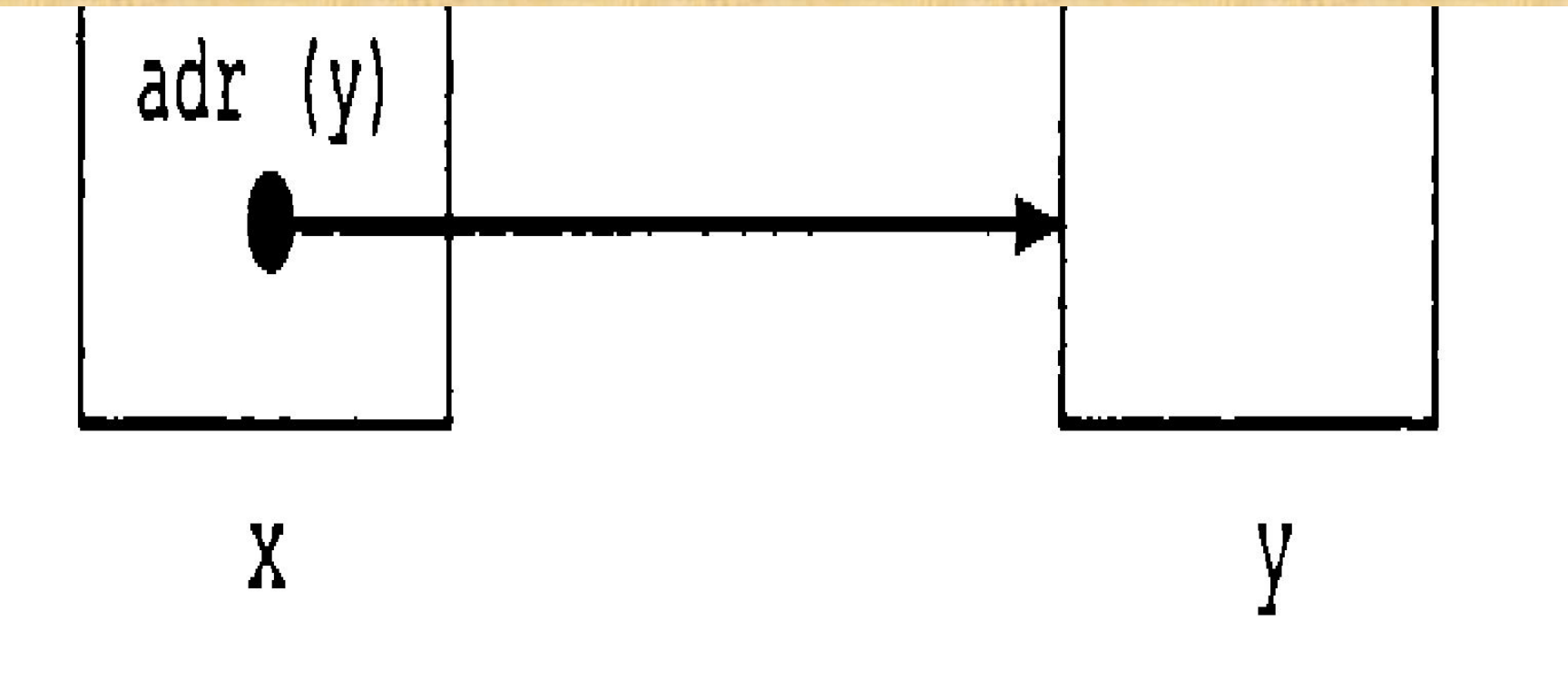
```
int main()
{    setlocale(LC_ALL, "Russian");
    //зашиваем размерность массива
 int size ;
    //флаг для выхода из сортировки
   bool flag = false;
    //создание массива
   double mass[50];
   cout<<"\nВведите "<< size <<" элементов массива:\n";
    for(int i = 0; i < size; i++)
   {       cout<<"A [ "<< i <<" ]= ";
      cin>>mass[i];
   }     //вывод неотсортированного массива
   cout<<"\n\nВведенный массив:\n\n";
    for(int i = 0; i < size; i++)
      //setw - установка расстояния между элементами массива в выводу( и именно для него iomanip )
      cout<<setw(4)<<mass[i];
         while(!flag)
   {       flag = true;
       for(int i = 0; i < size-1; i++)
         //по возрастанию - знак > , по убыванию - знак <
         if(mass[i] > mass[i+1])
         {              //выполняем перестановку соседних элементов с помощью функции swap(а то с буфером как-то
моветон)               swap(mass[i],mass[i + 1]);
            flag = false;
         }    }
   //вывод отсортированного массива
   cout<<"\n\nОтсортированный массив:\n\n";
    for(int i = 0; i < size; i++)
      cout<<setw(4)<<mass[i];
    _getch();
   return 0;
}
```

Указатели

Ссылочный тип данных является средством организации и обработки сложных изменяющихся структур данных.

Этот тип данных предназначен для обеспечения возможности указания на данные других типов и называется *указателем (ссылкой).*

Ссылочный тип данных



Статические переменные

Статические переменные представляют собой переменные, которые объявляются в некоторых процедурах или блоках.

Такие переменные формируются автоматически при передаче управления процедуре и уничтожаются при выходе из нее.

Время существования таких переменных соответствует времени выполнения данной процедуры.

Динамические переменные

Образование динамической переменной, содержащей непосредственно ссылочное значение, осуществляется в результате выполнения специальной процедуры

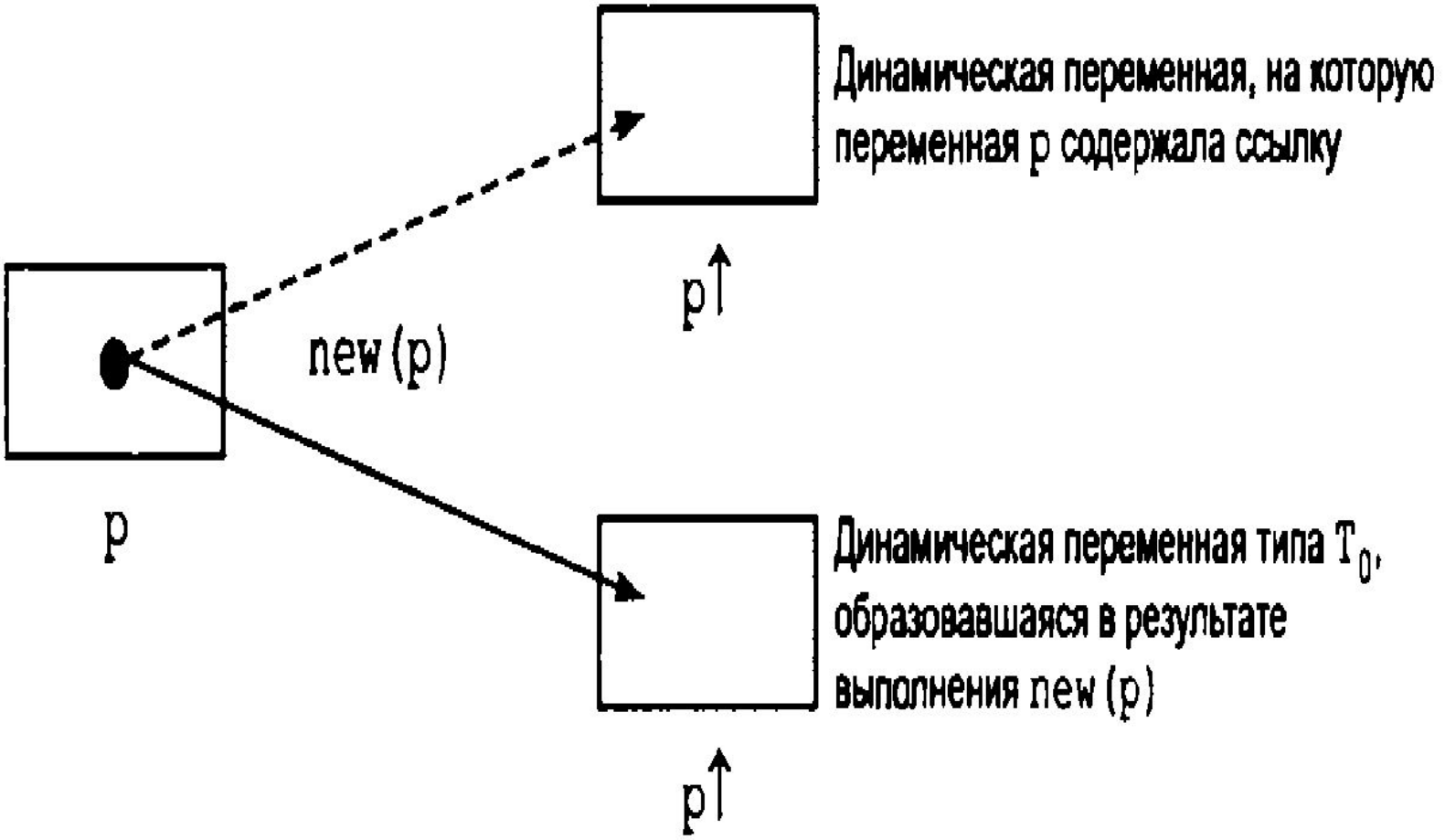
n e w (p) .

Процедура new(p) **обеспечивает**:

1) размещение переменной динамического типа To в памяти;

2) присваивание переменной **p** ссылки на размещенную переменную.

Динамические переменные



Классификация структур данных

1. По характеру взаимосвязи элементов структуры (с точки зрения порядка их размещения/выборки) виды структур можно разделить на **линейные** и **нелинейные**.

2. По характеру информации, представляемой структурой (с точки зрения однородности и «элементарности» типов данных), — на **однородные** структуры, где все элементы имеют один тип данных, и **неоднородные** (композиционные), где элементы относятся к разным типам данных.

Линейные структуры

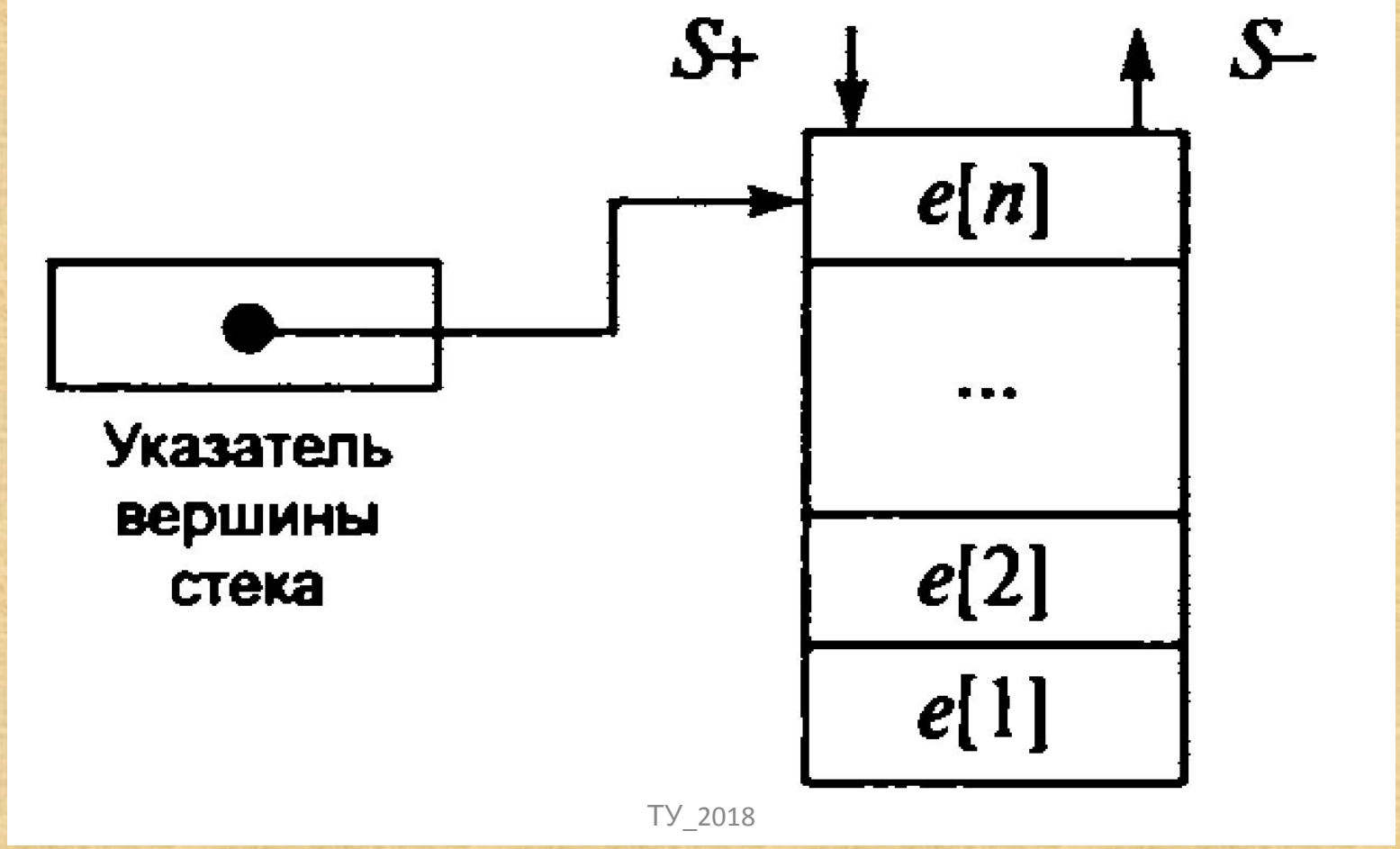
- **Последовательные файлы**
- **Стеки**
- **Очереди**
- **Массивы**

Последовательность так же, как и массив, представляет собой совокупность *однотипных элементов*.

Однако число элементов до размещения неизвестно.

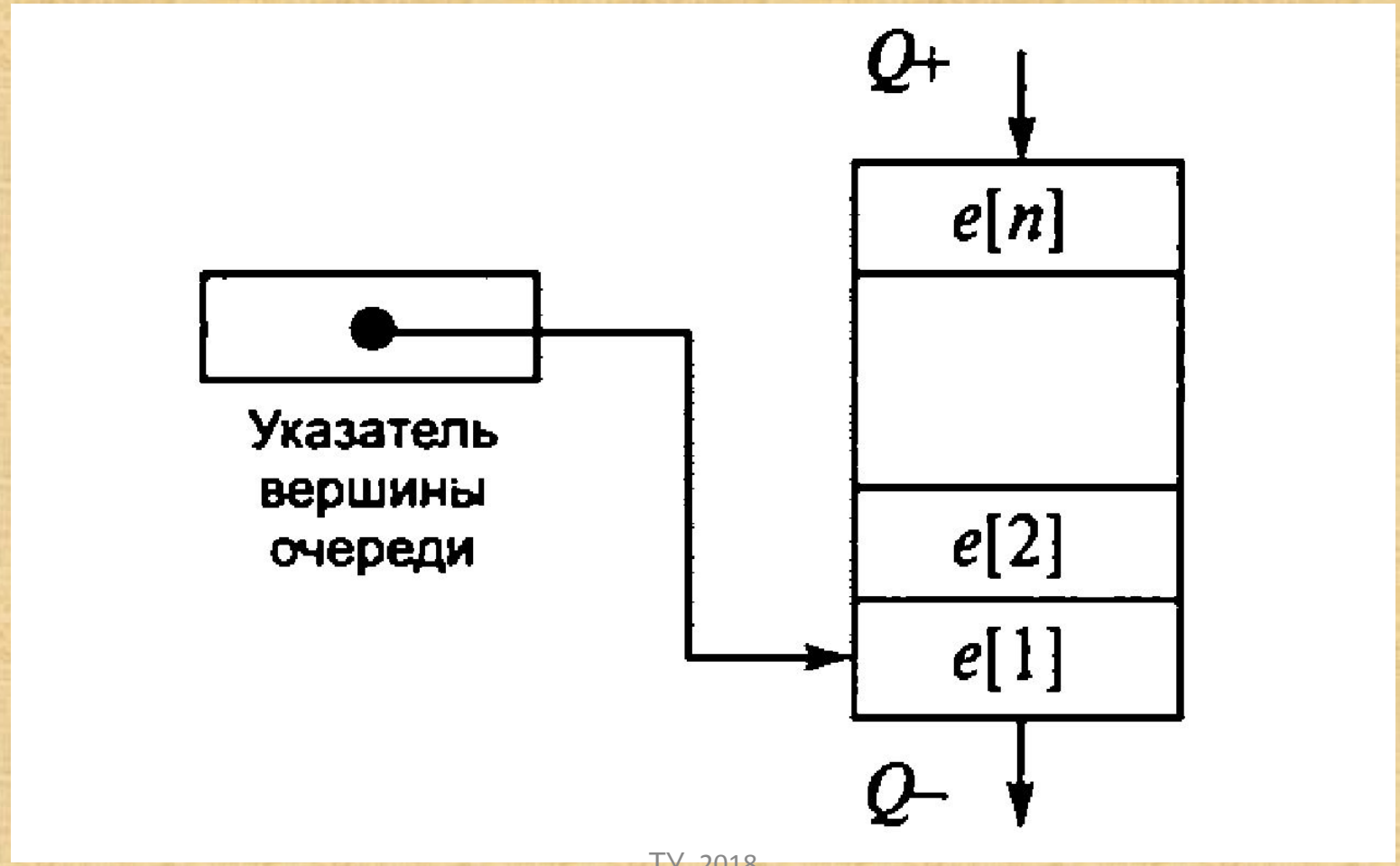
Стек

Last In First Out — LIFO



Очередь

First In First Out — FIFO



Нелинейные структуры

- **Списки**
- **Деревья**
- **Графы**

Порядок следования (и, соответственно, выборки) элементов таких структур может не соответствовать порядку расположения элементов в памяти.

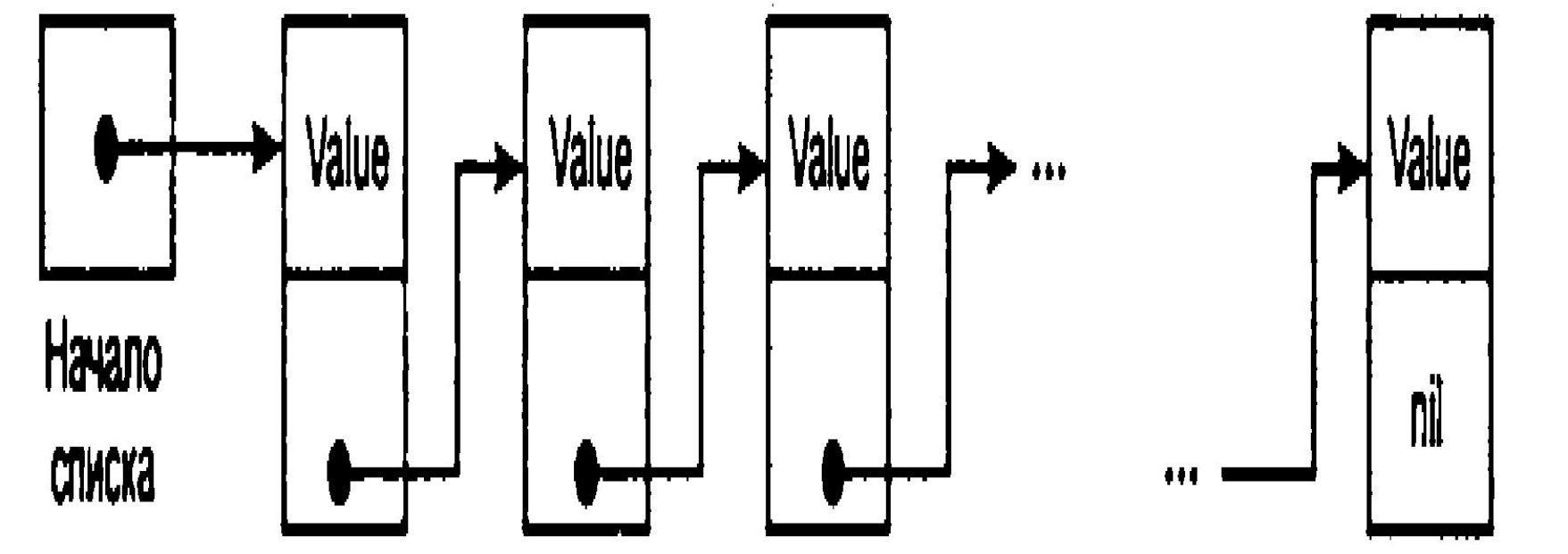
Списки

В зависимости от способа построения списка и предполагаемых путей доступа к элементам различают следующие виды списков:

- **однонаправленные;**
- **двунаправленные;**
- **циклические.**

Однонаправленный список

каждый элемент содержит обязательно только одну ссылку — на следующий по порядку элемент.



Однонаправленные списки

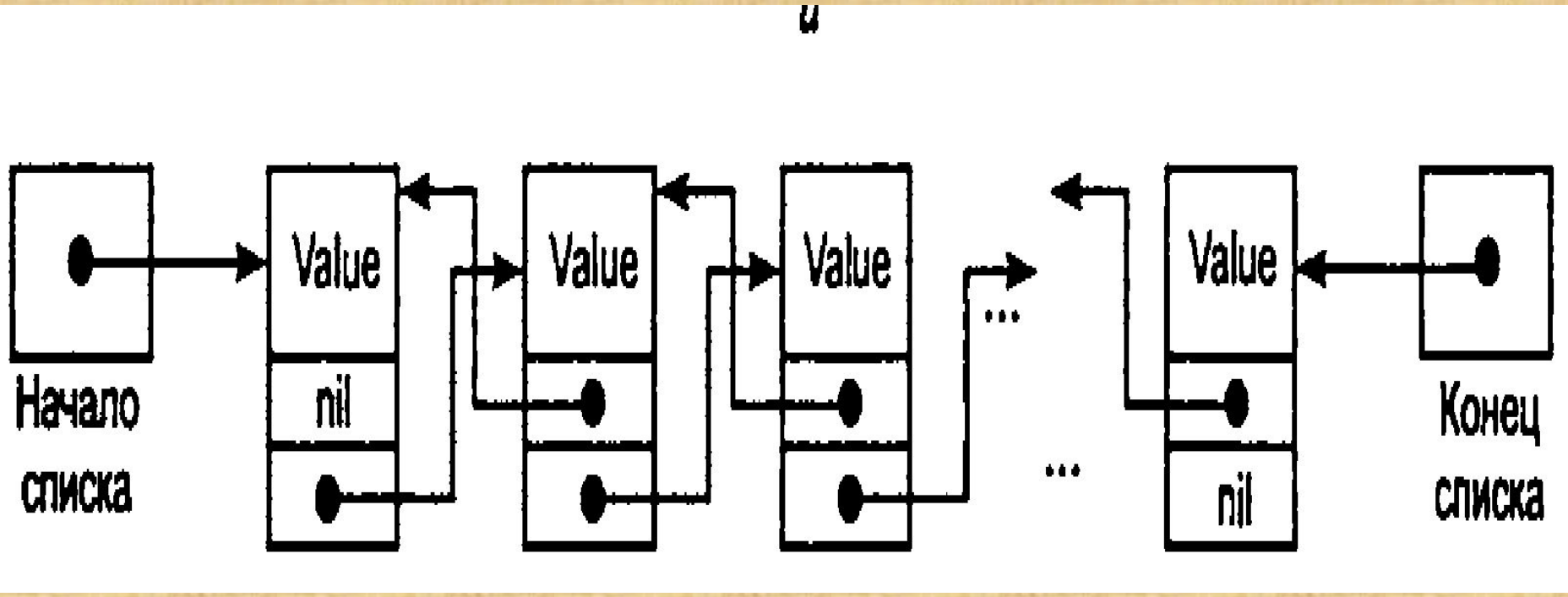
предусматривают жесткий порядок перебора элементов — только в одном направлении, от первого к последнему.

Двунаправленный список

представляет собой цепочку элементов, в которой каждый элемент содержит ссылку не только на следующий, но и на предыдущий.

Для таких списков нужна дополнительная переменная — указатель на последний элемент списка.

Двунаправленный список

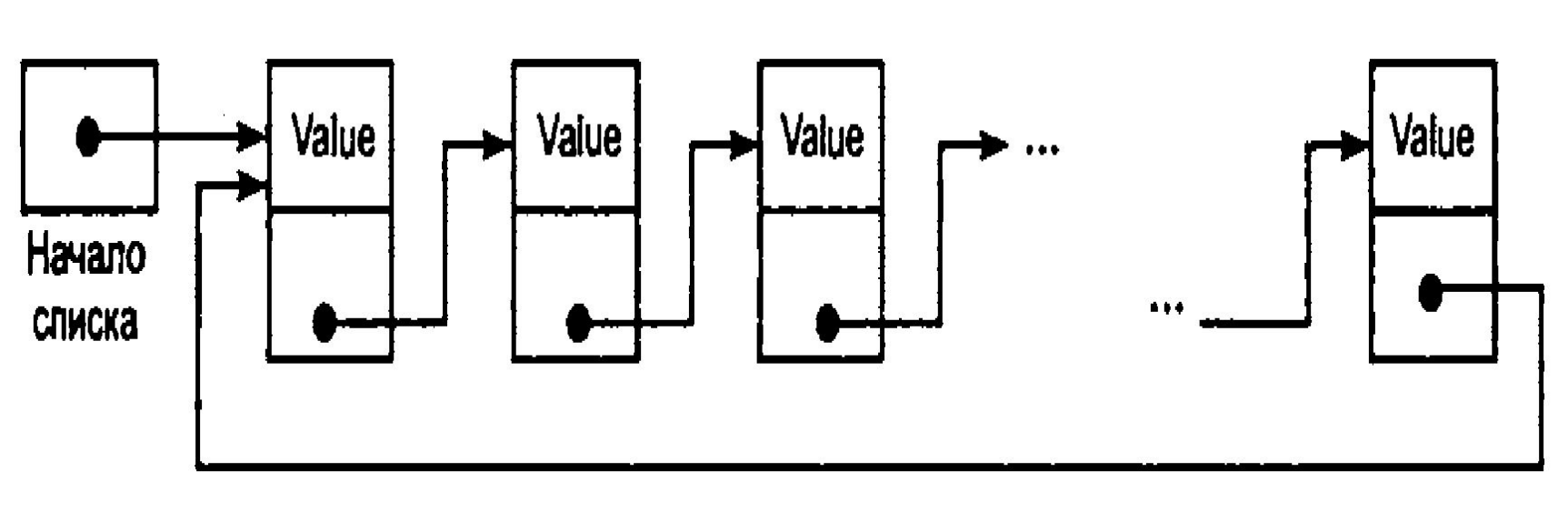


Циклические списки

В циклических или кольцевых списках порядок следования элементов зацикливается следующим образом: в ***однонаправленном кольцевом списке*** последний элемент ссылается на первый

как на следующий, а в **двунаправленном кольцевом списке** последний ссылается на первый как на следующий, а первый — на последний как на предыдущий.

Циклические списки



Деревья

Дерево представляет собой иерархию элементов, называемых узлами.

На самом верхнем уровне иерархии имеется только один узел — корень. Каждый узел, кроме корня, связан с одним узлом на более высоком уровне, называемым исходным узлом для данного узла.

Каждый элемент имеет только один исходный.

Деревья

Каждый элемент может быть связан с одним или несколькими элементами на более низком уровне, которые называются порожденными.

Элементы, расположенные в конце ветви, т. е. не имеющие порожденных, называются листьями.

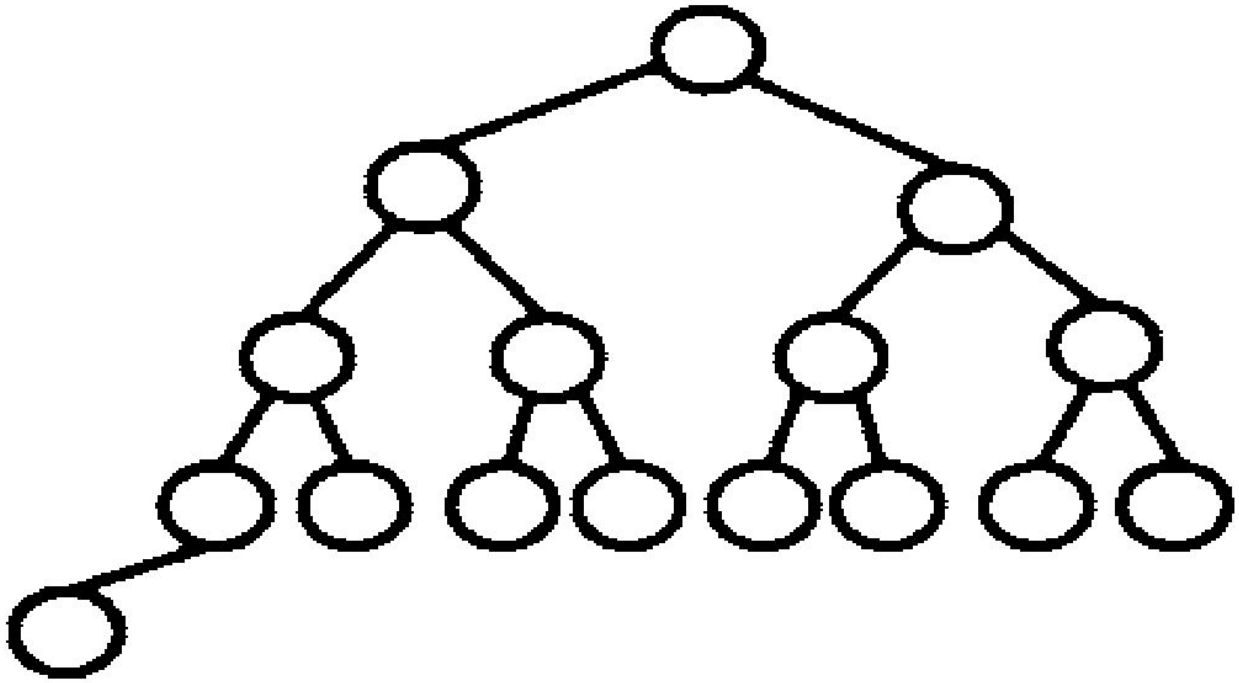
Деревья

- *Высота дерева* определяется **количеством уровней**, на которых располагаются его узлы
- В соответствии со структурой заполнения деревья подразделяются

на :

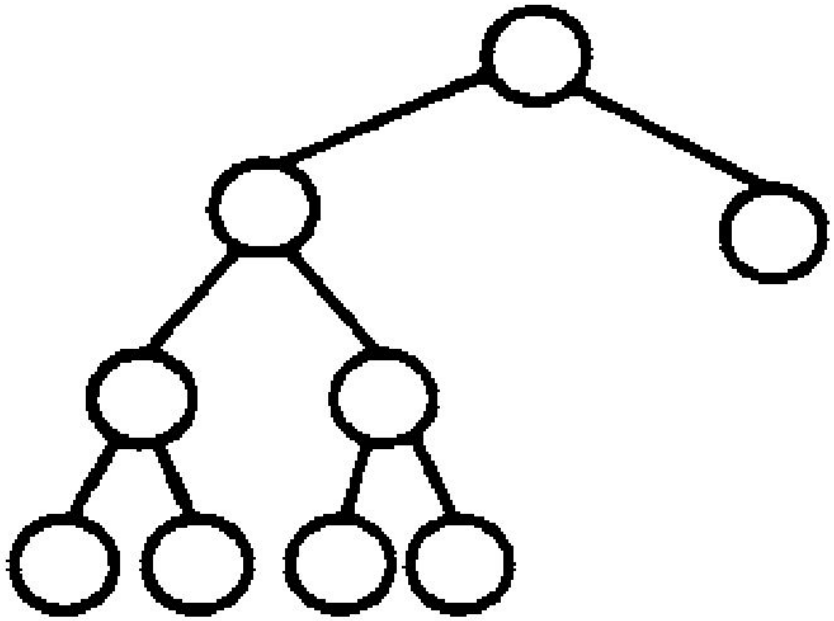
- сбалансированные;
- несбалансированные.

Деревья



Сбалансированное двоичное дерево

Деревья



Несбалансированное двоичное дерево

Двоичные деревья

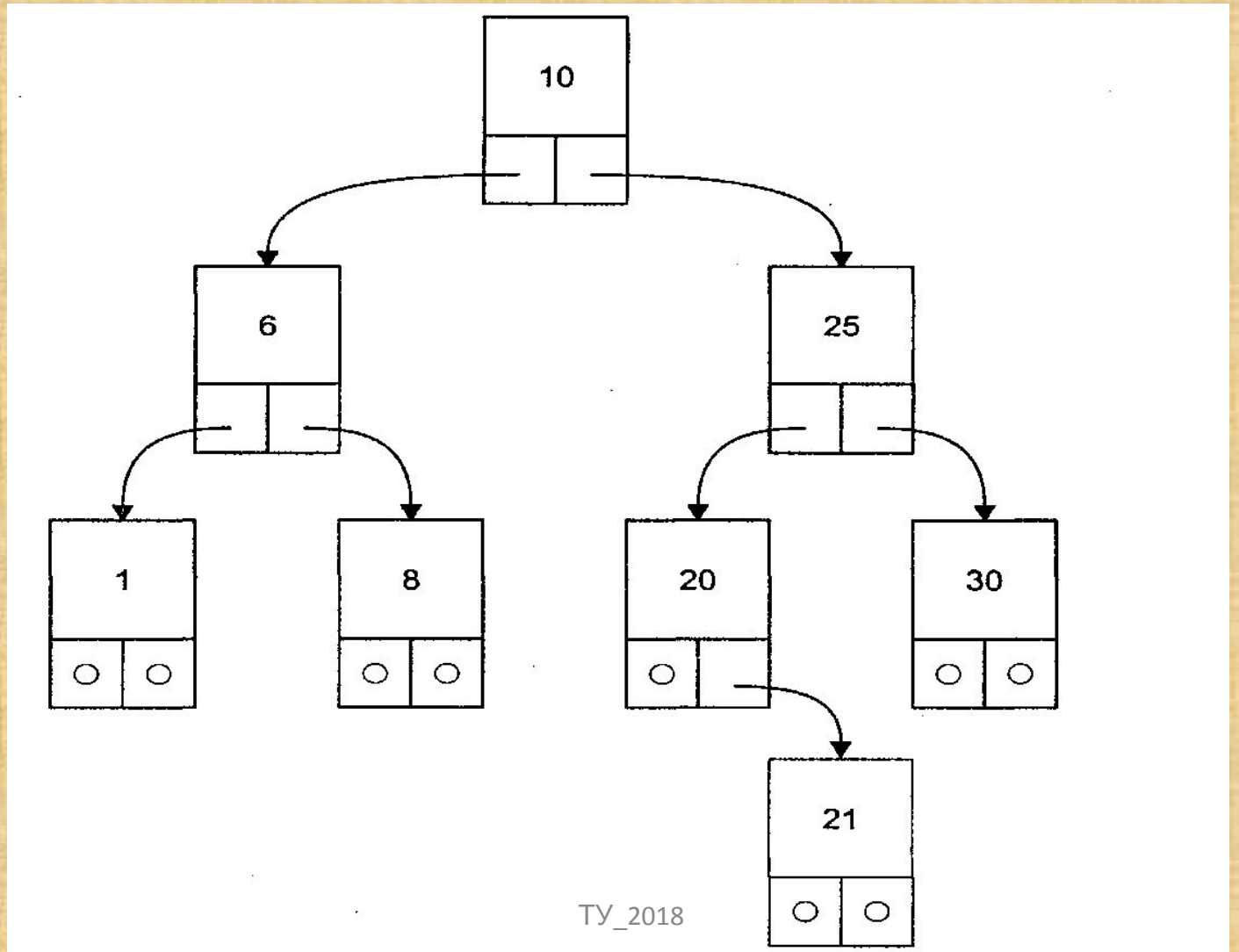
— это древовидная структура, в которой допускается не более двух ветвей для одного узла.

Любые связи в дереве с любым количеством ветвей можно представить в виде двоичных древовидных структур.

Двоичные деревья

Если дерево организовано таким образом, что для каждого узла все ключи его **левого поддерева** меньше ключа этого узла, а все ключи его **правого поддерева** — больше, оно называется *деревом поиска*

ДВОИЧНОЕ ДЕРЕВО



Двоичные деревья

- Дерево является рекурсивной структурой данных, поскольку каждое поддерево также является деревом.
- Действия с такими структурами нагляднее всего описываются *с помощью рекурсивных алгоритмов*

Двоичные деревья

function way__around (дерево){
way_around (левое поддерево)
посещение корня
way_around (правое поддерево)
}

Можно обходить дерево и в другом порядке, например, сначала корень, потом поддеревья

Двоичные деревья

Если дерево организовано таким образом, что для каждого узла все ключи его **левого поддерева** меньше ключа этого узла, а все ключи его **правого поддерева** — больше, оно называется *деревом поиска*

Формирование дерева из массива целых чисел

```
#include <iostream.h>
struct Node {
int d;
Node *left;
Node *right;
};
Node * f i r s t (int d); / / Формирование первого элемента дерева
Node * search_insert(Node *root, int d); // Поиск с включением
void print_tree(Node *root, int l);
```

Графовые структуры

Графовая структура представляет собой наиболее общий (произвольный) случай размещения и связей отдельных элементов

в памяти.

Списковые структуры и деревья – это частные случаи графа

Графовые структуры

Один из способов представления графовых структур в памяти ЭВМ— представление графа в виде *совокупности узлов и дуг*.

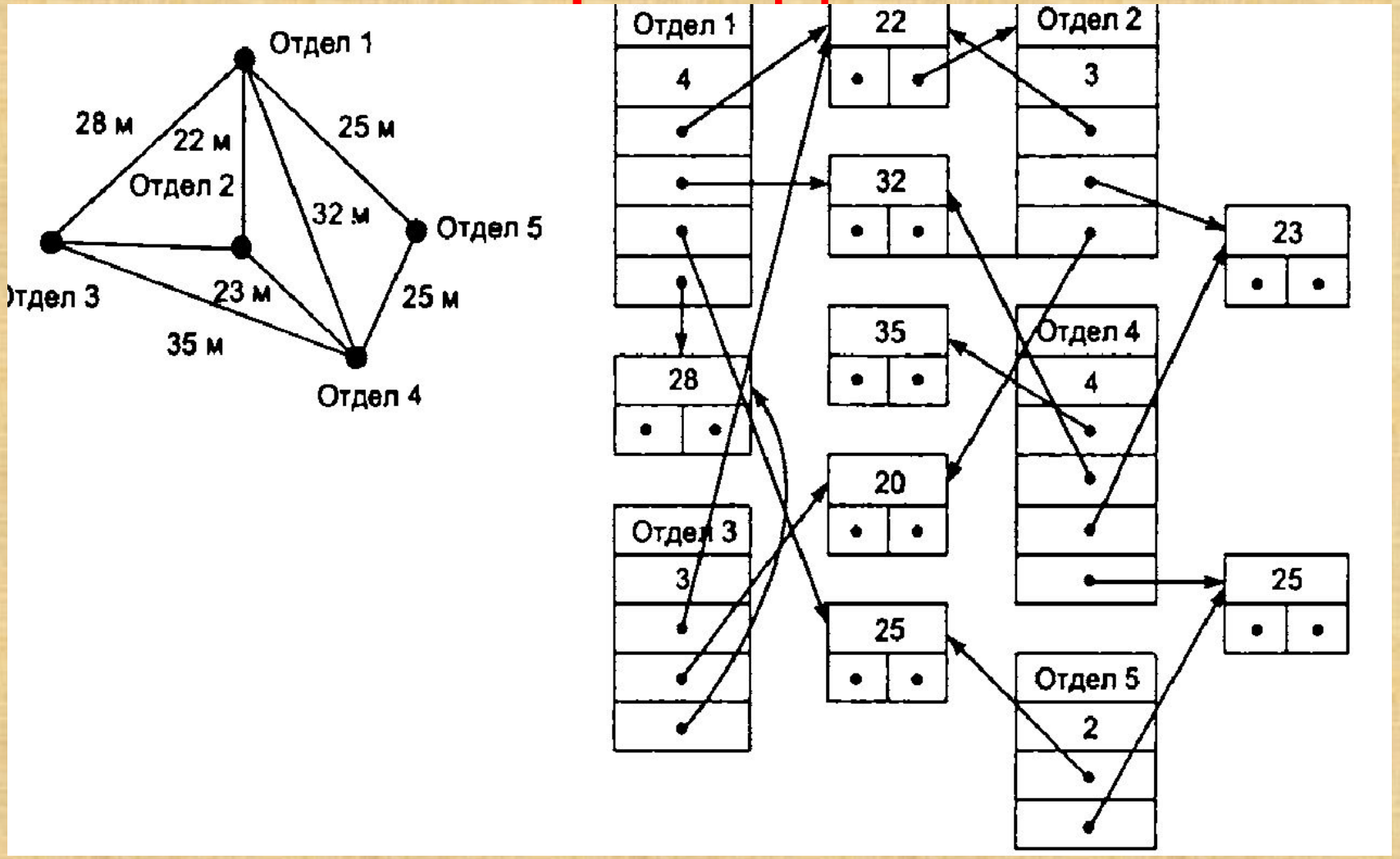
Дуги при этом представляют собой однотипные структуры, состоящие из двух частей: *данные и пара указателей*, соответственно, на *левый и правый узлы*.

Графовые структуры

Один из способов представления графовых структур в памяти ЭВМ— представление графа в виде *совокупности узлов и дуг*.

Дуги при этом представляют собой однотипные структуры, состоящие из двух частей: *данные и пара указателей*, соответственно, на *левый и правый узлы*.

Граф для схемы сетевого провода



Алгоритмы сортировки

- https://www.intuit.ru/studies/courses/648/504/lecture/11466
- https://prog-cpp.ru/algorithm-sort/
- https://ppt-online.org/95398
- https://ppt4web.ru/informatika/metody-sortirovki-dannykh.html