# Структуры и алгоритмы обработки данных

*Лекция 4*

Базовые типы данных языков программирования высокого уровня (ч.2)

# Основные принципы концепции типа данных

*Тип данных определяет*

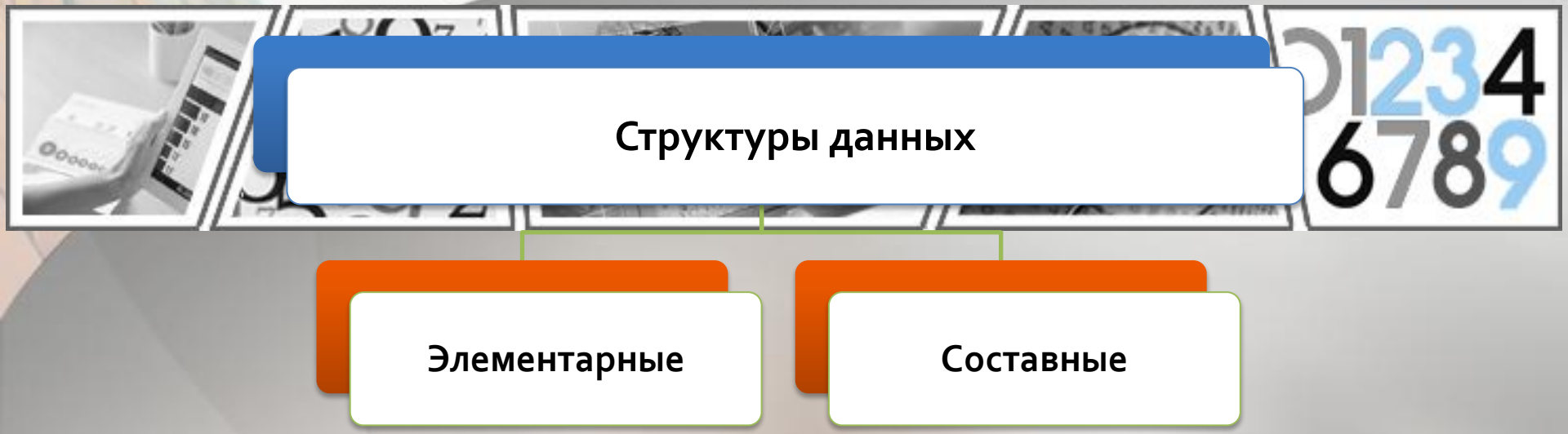
- *множество допустимых значений;*
- *множество операций, которые могут выполняться над значением;*
- *структуру значения (скаляр, вектор и т.д.);*
- *способ машинного представления значения*

# Основные принципы концепции типа данных

- *Тип константы, переменной или выражения может быть определен по внешнему виду (по изображению) или по описанию без выполнения каких-либо вычислений*

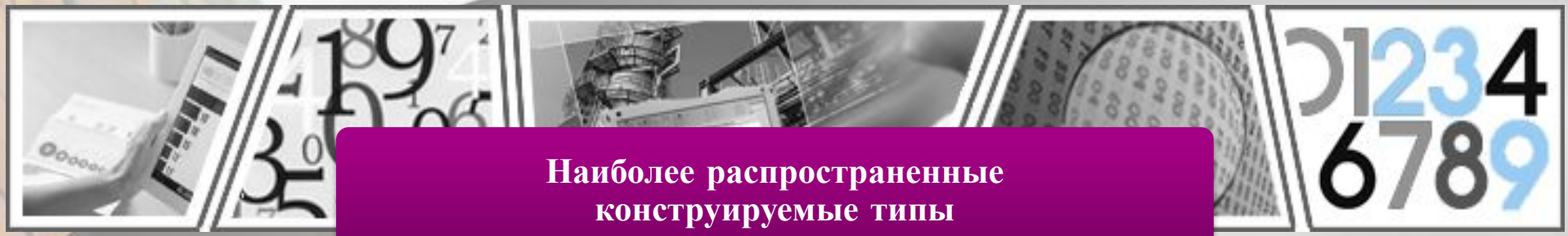
- *Любая операция или функция требует аргументов и возвращает результат вполне определенного типа. Типы аргументов и результатов операций определяется по вполне определенным правилам языка*

# Конструируемые типы данных




- *Рассмотрим группы разновидностей типов данных, которые в литературе часто называют "составными", т.к. любое значение любого из этих типов состоит из значений одного или нескольких других типов*

# Конструируемые типы данных



**Наиболее распространенные конструируемые типы**

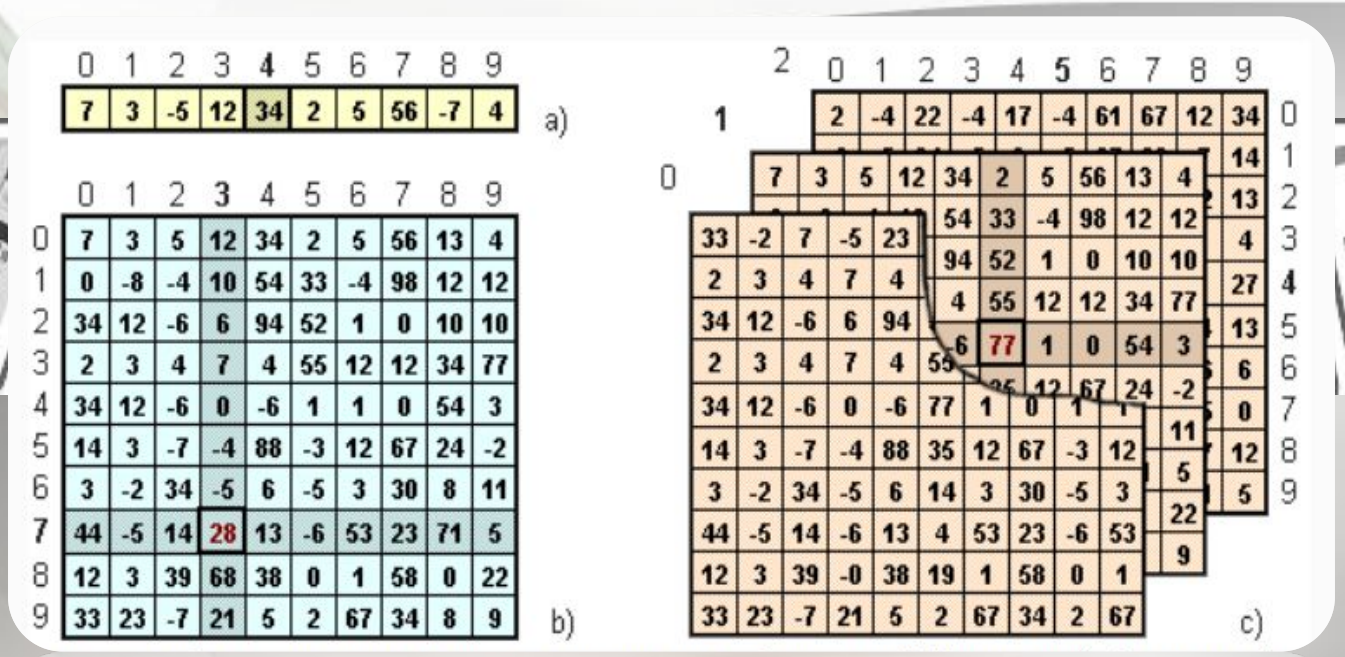
*тип массива*

*тип строки*

*тип записи*

*тип множества*

# Массивы



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 3 | -5 | 12 | 34 | 2 | 5 | 56 | -7 | 4 |

a)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7 | 3 | 5 | 12 | 34 | 2 | 5 | 56 | 13 | 4 |
| 1 | 0 | -8 | -4 | 10 | 54 | 33 | -4 | 98 | 12 | 12 |
| 2 | 34 | 12 | -6 | 6 | 94 | 52 | 1 | 0 | 10 | 10 |
| 3 | 2 | 3 | 4 | 7 | 4 | 55 | 12 | 12 | 34 | 77 |
| 4 | 34 | 12 | -6 | 0 | -6 | 1 | 1 | 0 | 54 | 3 |
| 5 | 14 | 3 | -7 | -4 | 88 | -3 | 12 | 67 | 24 | -2 |
| 6 | 3 | -2 | 34 | -5 | 6 | -5 | 3 | 30 | 8 | 11 |
| 7 | 44 | -5 | 14 | 28 | 13 | -6 | 53 | 23 | 71 | 5 |
| 8 | 12 | 3 | 39 | 68 | 38 | 0 | 1 | 58 | 0 | 22 |
| 9 | 33 | 23 | -7 | 21 | 5 | 2 | 67 | 34 | 8 | 9 |

b)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | -4 | 22 | -4 | 17 | -4 | 61 | 67 | 12 | 34 |
| 1 | 7 | 3 | 5 | 12 | 34 | 2 | 5 | 56 | 13 | 4 |
| | 33 | -2 | 7 | -5 | 23 | 54 | 33 | -4 | 98 | 12 |
| | 2 | 3 | 4 | 7 | 4 | 94 | 52 | 1 | 0 | 10 |
| | 34 | 12 | -6 | 6 | 94 | 4 | 55 | 12 | 12 | 34 |
| | 2 | 3 | 4 | 7 | 4 | 55 | 6 | 77 | 1 | 0 |
| | 34 | 12 | -6 | 0 | -6 | 77 | 1 | 0 | 1 | 1 |
| | 14 | 3 | -7 | -4 | 88 | 35 | 12 | 67 | -3 | 12 |
| | 3 | -2 | 34 | -5 | 6 | 14 | 3 | 30 | -5 | 3 |
| | 44 | -5 | 14 | -6 | 13 | 4 | 53 | 23 | -6 | 53 |
| | 12 | 3 | 39 | -0 | 38 | 19 | 1 | 58 | 0 | 1 |
| | 33 | 23 | -7 | 21 | 5 | 2 | 67 | 34 | 2 | 67 |

c)

Структуры аналогичные векторам и матрицам в информатике принято называть *массивами*

Все элементы массива должны быть *одного и того же типа*

$$\bar{x} = \{x_1, x_2, x_3\}; \quad \bar{w} = (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9);$$

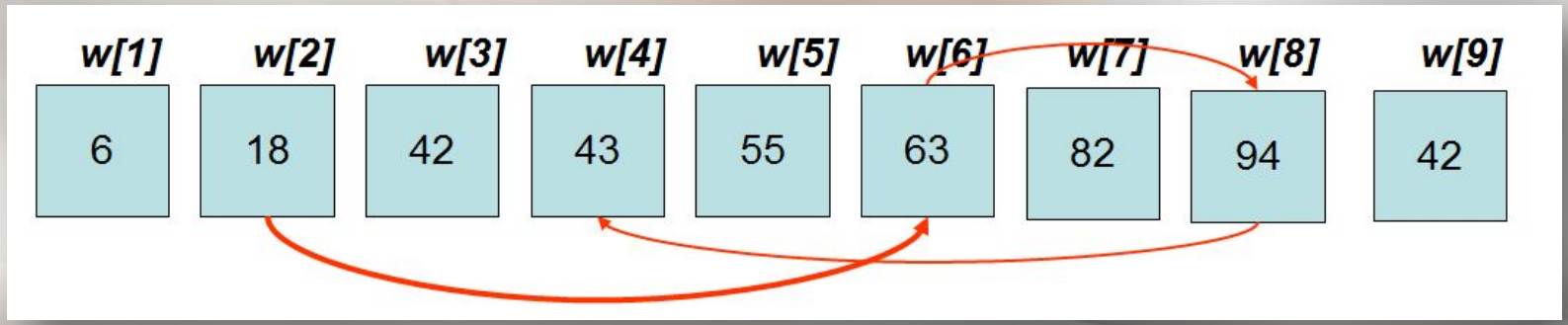
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

# Массивы

**Массив** – это последовательность однотипных данных, объединенная общим именем, элементы (компоненты) которой отличаются (идентифицируются) индексами

**Индекс** элемента указывает место (номер) элемента в массиве

**Количество элементов массива** фиксировано и определено в его описании и называется его **размером**

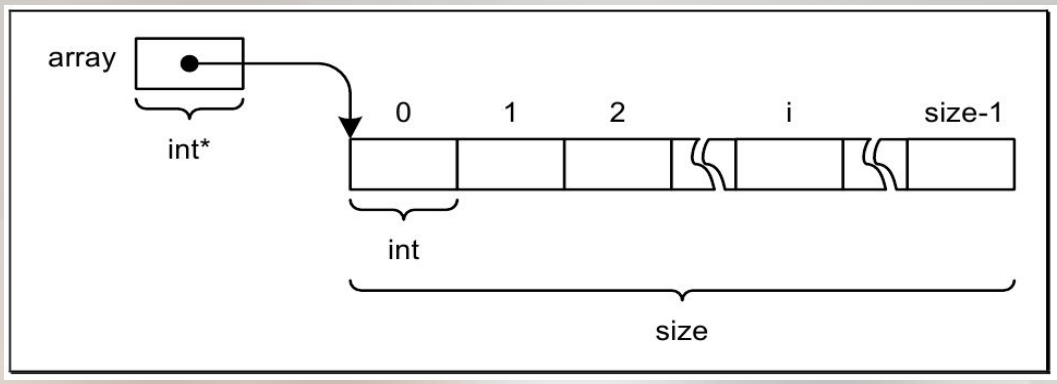


# Массивы

- К элементам массива можно обращаться только по их номеру (индексу)
- Все компоненты массива являются одинаково доступными
- Значения элементам массива присваиваются также как и другим переменным с учетом типа массива
- Для доступа (обращения) к отдельному элементу массива используется индекс или несколько индексов (w[5];  w[i+2]; A[1,2])
- Индексы могут быть выражениями, значения которых могут произвольным образом изменяться в заранее заданных границах. Поэтому говорят, что к элементам массивов имеется *прямой доступ*

# Массивы

- **В языке C++ под индексом элемента массива понимают смещение относительно адреса начала массива в оперативной памяти, выраженное в элементах**




- **описание массива сводится к описанию адресной переменной (указателя), хранящей адрес первого элемента массива (с индексом 0)**

В **C++** индекс может изменяться от **0** до **size-1,**
где **size –** число элементов в массиве

# Массивы

**Массив – это группа однотипных элементов, имеющих общее имя и расположенных в памяти рядом**

**Особенности:**

- все элементы имеют один тип
- весь массив имеет одно имя
- все элементы расположены в памяти рядом

**Примеры:**

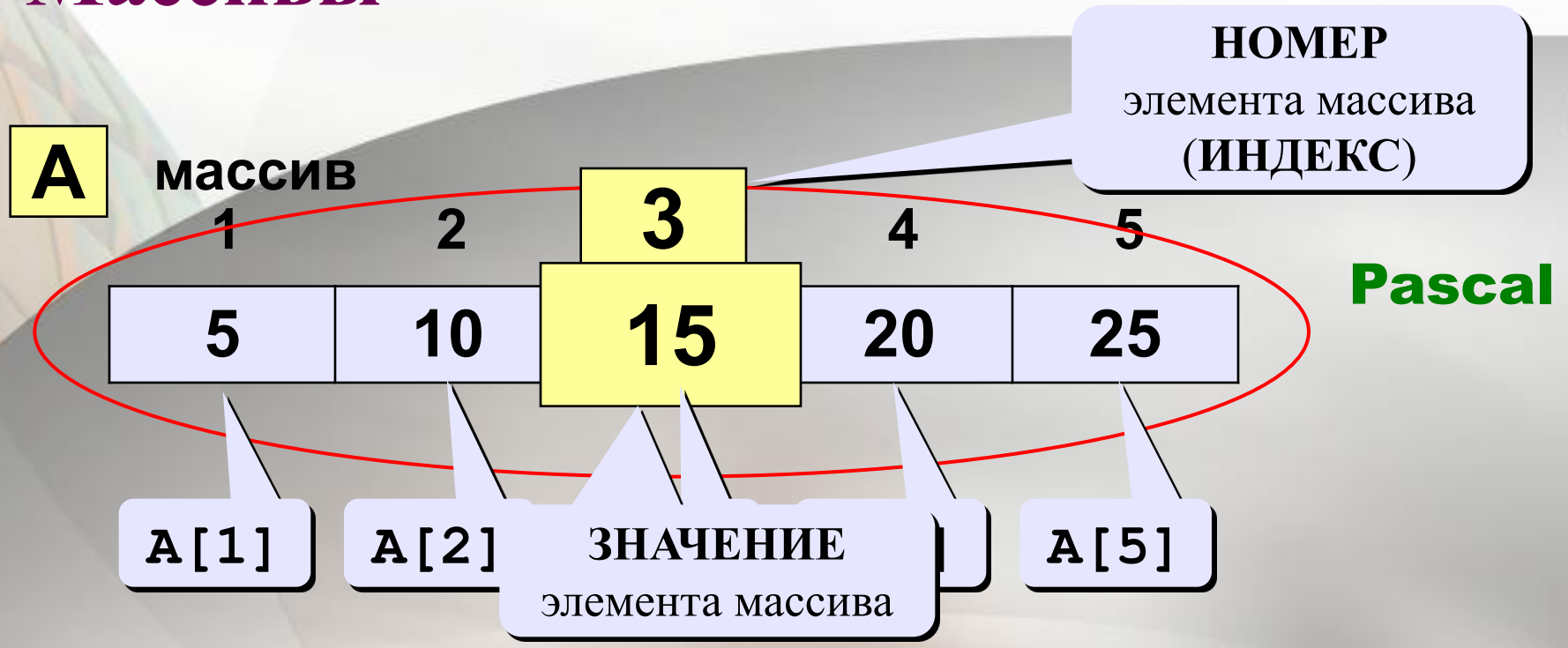
- список студентов в группе
- квартиры в доме
- школы в городе
- данные о температуре воздуха за год

# Массивы

**Массив – это группа однотипных элементов, имеющих общее имя и расположенных в памяти рядом**

**Примеры:** **Накормить кота из 3 коробки**. Сравните с командой **Накормить Мурчика**. Чувствуете разницу?

# Массивы


НОМЕР
элемента массива
(ИНДЕКС)
A
массив
1
2
3
4
5
5
10
15
20
25
Pascal
A[1]
A[2]
ЗНАЧЕНИЕ
элемента массива
A[5]

НОМЕР (ИНДЕКС)
элемента массива: 2
A[2]
ЗНАЧЕНИЕ
элемента массива: 10

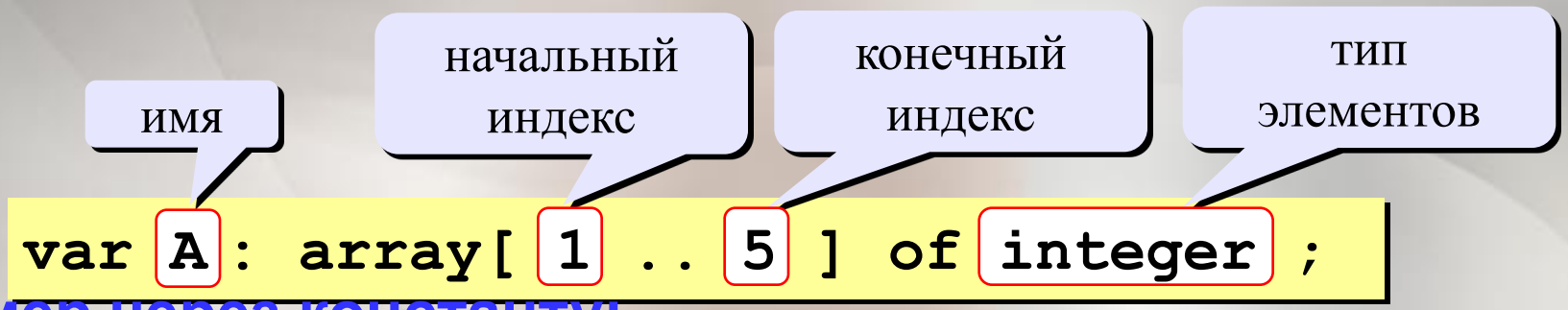
# Объявление массивов

**Зачем объявлять?**

- определить **имя** массива
- определить **тип** массива
- определить **число элементов**
- выделить **место в памяти**

**Pascal**

**Массив целых чисел:**




**Размер через константу:**

```
const N=5;
var A: array[1..N] of integer;
```

# Объявление массивов

Массивы других типов:

```
var X, Y: array [1..10] of real;
    C: array [1..20] of char;
```

Pascal

Другой диапазон индексов:

```
var Q: array [0..9] of real;
    C: array [-5..13] of char;
```

Ин

```
var A: array ['A'..'Z'] of real;
    B: array [False..True] of integer;
...
   A['C'] := 3.14259*A['B'];
   B[False] := B[False] + 1;
```

# Тип компонент

**Тип компонент массива может быть любым:**

Pascal

```
var a4: array[10..20] of real;
```
- массив из компонент простого типа

```
a5: array[0..100] of record1;
```
- массив из записей

```
a6: array[-10..10] of ^string;
```
- массив из указателей на строки

```
a7: array[-1..1] of file;
```
- массив из имен файловых переменных

```
a8: array[1..100] of array[1..100] of char;
```
- двумерный массив (массив векторов)

# Что неправильно?

```
var a: array [1..10] of integer;
...
  A[5] := 4.5;
```

Pascal

```
var a: array ['a'..'z'] of integer;
...
  A['b'] := 15;
```

```
var a: array [0..9] of integer;
...
  A[10] := 'x';
```

# Массивы

| | | | | | |
|---|---|---|---|---|---|
| | arr[0][0] | arr[0][1] | arr[0][2] | arr[0][3] | |
| | arr[1][0] | arr[1][1] | arr[1][2] | arr[1][3] | |
| | arr[2][0] | arr[2][1] | arr[2][2] | arr[2][3] | |

purecodecpp.com

**Двумерные массивы в C++**

| | столбец 0 | столбец 1 | столбец 2 | столбец 3 |
|---|---|---|---|---|
| строка 0 | arr[0][0] | arr[0][1] | arr[0][2] | arr[0][3] |
| строка 1 | arr[1][0] | arr[1][1] | arr[1][2] | arr[1][3] |
| строка 2 | arr[2][0] | arr[2][1] | arr[2][2] | arr[2][3] |

# Массивы

*Краткие итоги*

- Массив является представителем *структурированного типа данных* в языке C++
- Элементы массива имеют одинаковые имя, тип и располагаются в памяти последовательно
- Элементы массива характеризуются индексами, значениями и адресуемой памятью
- Для массивов нельзя выполнить операцию прямого присваивания
- Адресация элементов массива осуществляется с помощью индексированного имени. Обращаться к элементам массива можно также посредством механизма указателей
- Массивы используются для решения прикладных задач

# Строки

- Строкой называется последовательность символов
- Строка – это динамическая структура
- В процессе выполнения программы количество элементов строки (ее длина) может изменяться от нуля до максимального размер памяти, выделенной под эту строку
- Строка представляет собой массив символов из элементов типа char

# Строки

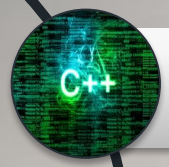
**Чем плох массив символов?**

- каждый символ – отдельный объект;
- массив имеет длину N, которая задана при объявлении

**Что нужно?**

- обрабатывать последовательность символов как единое целое
- строка должна иметь переменную длину

# Операции над строками




присваивание

сравнение

конкатенация

# Операции над строками

*Операция присваивания*

- ***Операндами могут быть символы, строки, символьные массивы***
- ***Результатом операции является строка равная значению операнда***
- ***Если строковой переменной присваивается значение, превышающее её длину, то перед присваиванием происходит усечение присваиваемого значения***

# Операции над строками

*Операция сравнения*

- ***Переменные строкового типа можно сравнивать между собой***
- ***Из двух строк является та большей, у которой первый из неравных символов больше (по ASCII-коду)***
- ***Иначе они равны***

***Система кодировки символов ASCII (American Standard Code for Information Interchange)***

# Операции над строками

*Операция конкатенации*

- ***Операндами могут быть символы, строки, символьные массивы***
- ***Результатом является строка, полученная дописыванием в конец первого операнда второго операнда***

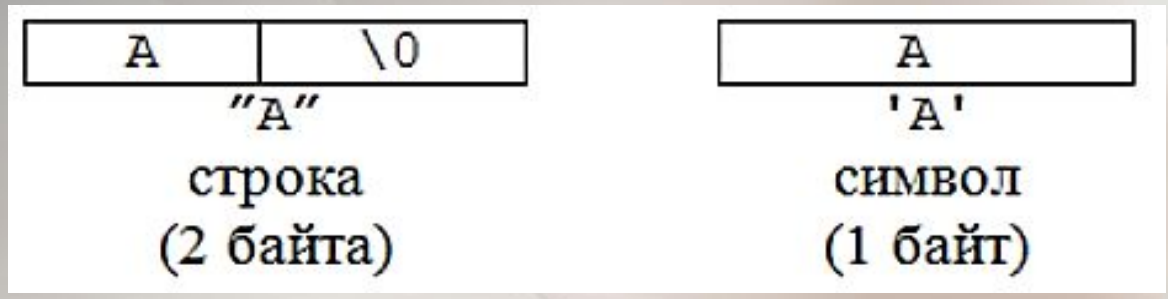
# Операции над строками

*В языке C++ строка – это пронумерованная последовательность символов (массив символов), она всегда имеет тип char[]*

- *Все символы строки нумеруются, начиная с нуля*
- *Символ конца строки также нумеруется – ему соответствует наибольший из номеров*
- *Строка считывается значением типа «массив символов»*
- *Количество элементов в таком массиве на 1 больше, чем изображение соответствующей строки, так как в конец строки добавлен нулевой символ '\0'*

# Операции над строками

*В языке C++ строка – это пронумерованная последовательность символов (массив символов), она всегда имеет тип char[]*




*Представление символа и строки*

**Необходимо отметить, что один *символ* и *строка* длиной в один символ совершенно не эквивалентны друг другу.**

**Вне зависимости от своей реальной длины, строка относится к конструируемым структурированным типам данных, а не к базовым порядковым**

# Записи (Структуры)

*Структура* – это составной объект, в который входят элементы любых типов, за исключением функций

В отличие от массива, который является однородным объектом (все элементы относятся к одному типу данных), структура может быть неоднородной

*Структура* – это тип данных, сформированный из объектов однородных либо разнообразных типов данных

*Структуру* можно представить себе как запись, состоящую из нескольких полей или элементов

*Структуры* обеспечивают удобный способ организации связанных по смыслу переменных

# Записи (Структуры)

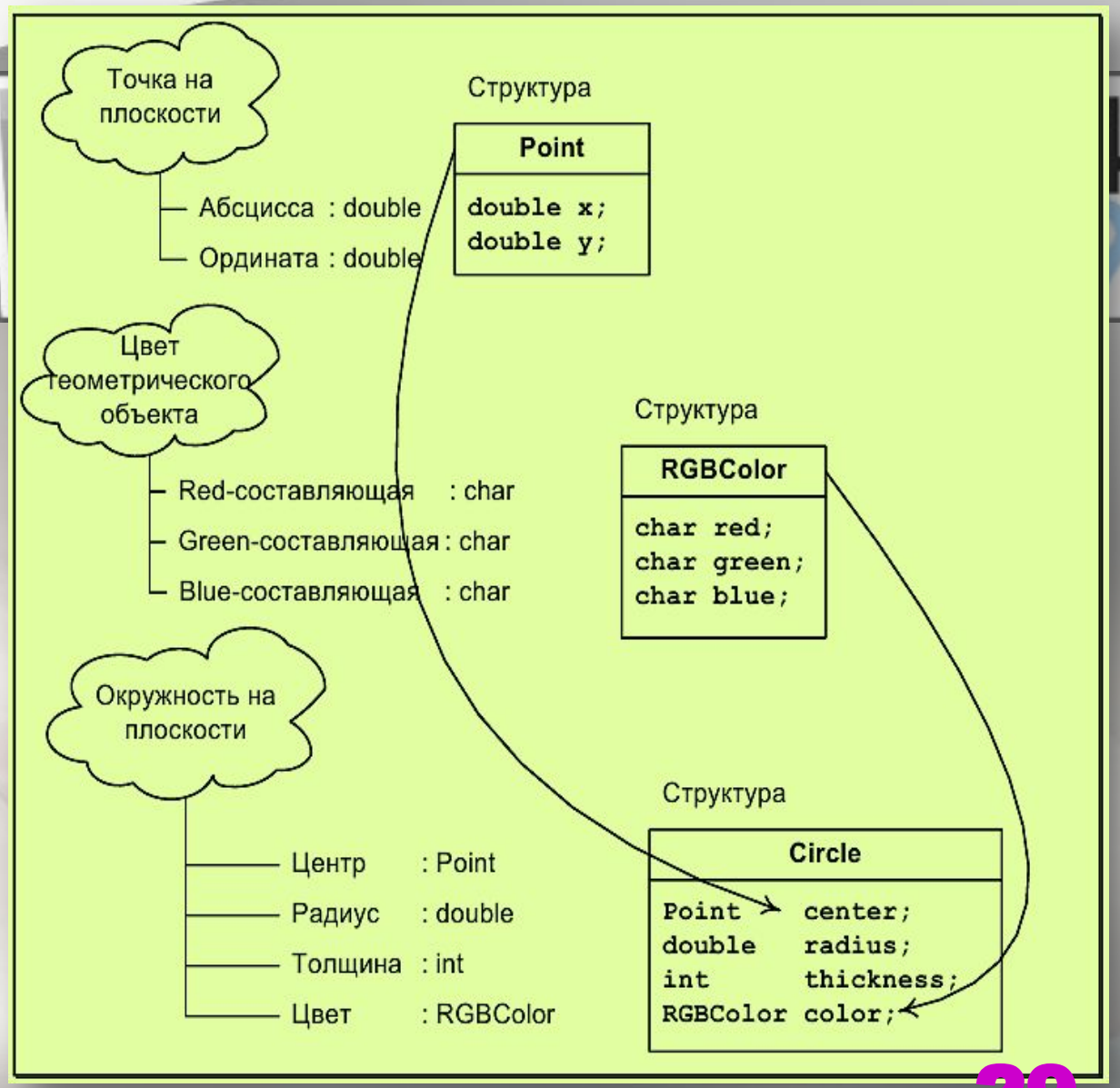
*Традиционный пример структуры – строка платежной ведомости:*

- *содержит сведения о служащем:*
  - *полное имя*
  - *адрес*
  - *номер карточки социального страхования*
  - *зарплата и т.д.*
- *некоторые из этих характеристик сами могут быть структурами:*
  - *полное имя состоит из нескольких компонент (фамилии, имени и отчества)*
  - *адрес*
  - *зарплата*

# Записи (Структуры)

*Традиционный пример структуры из области графики :*

- *точка на плоскости есть пара вещественных координат,*
- *шар в пространстве моделируется четырьмя вещественными числами и т. д*



# Записи (Структуры)

| | День | Месяц | Год |
|---|---|---|---|
| День Победы: | 9 | май | 1945 |
| Полёт Гагарина: | 12 | апрель | 1961 |

*Записи* - *структуры, аналогичные строкам таблицы*

- *Компоненты записей принято называть полями*
- *Различные поля (столбцы таблицы) могут быть разных типов*
- *Для доступа к отдельным полям записи используются их фиксированные и неизменные имена*
  - *Например: День Победы. Месяц := май*
- *Поля могут выбираться для обработки в произвольном порядке, поэтому говорят, что доступ к компонентам записи прямой*

# Множества

**Множество - ограниченный , неупорядоченный набор различных элементов одного типа**

**Примеры множеств: Множество арабских цифр.** A

0 1 2 3 4
5 6 7 8 9

**Множество знаков арифметических операций.** B

+ - / *

**У множества есть имя и тип.**
**Тип элементов множества называется базовым типом.**

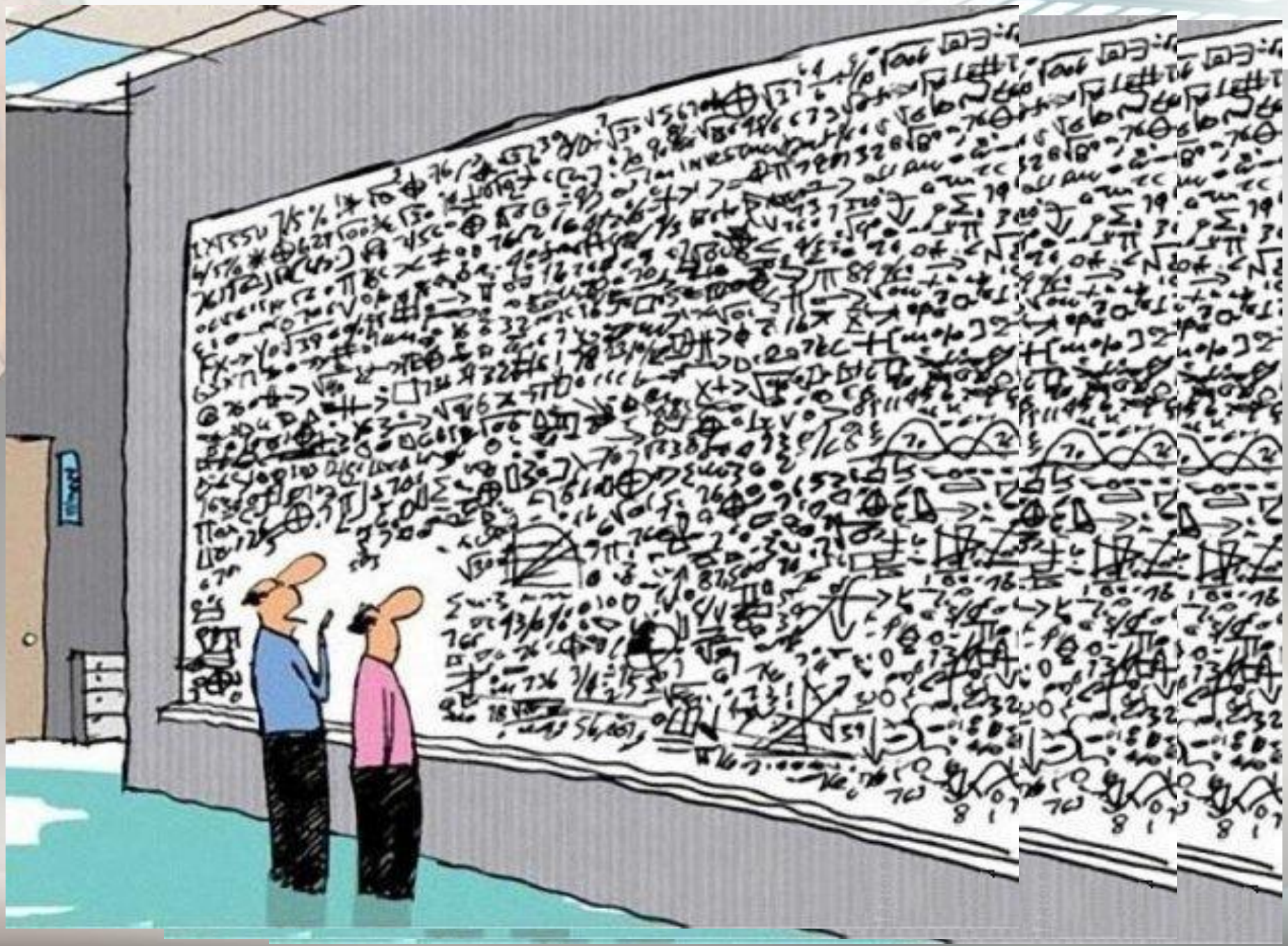
**Примеры: В множестве A базовым типом является интервальный, который включает цифры от 0 до 9.**

**В множестве B базовым типом является символьный тип**

# Множества

**Набор однородных объектов**

| Массив | Множество |
| --- | --- |
| *Основная операция доступа к элементу - операция обращения по индексу* | *Основная операция - операция проверки принадлежности элемента множеству* |

# Операции над множествами

- количество элементов в множестве заранее не определяется, и с течением времени оно может изменяться
- все элементы множества должны быть одного и того же типа
- доступа к отдельным элементам множества нет; можно только узнать принадлежит элемент множеству или нет, включить элемент в множество или исключить его из множества
- предусмотрены стандартные операции над множествами: *объединение*, *пересечение*, *вычитание* и т.д.