Типы и структуры данных

Лекции

Силантьева Александра Васильевна
ИУ7

Структура курса

- Лекции – еженедельно
- Лабораторные – еженедельно

Итоговая оценка (сумма баллов):

- 1) Л.р. 42 – 70 балл. - в срок с отчетами
  1) РК 1 (5 нед.) баллы по ЭУ (35?)
  2) РК 2 (10 нед.) баллы по ЭУ (35?)
  - +
- 2)теория РК 3 (16 нед.) 18-30 балл.
- 3) зачет 18-30 балл. (теория) + возможно задача

Лабораторные работы

всего 8 л.р.

- Баллы за каждую разные, в зав-ти от сложности и объема.
- Макс. баллы при сдаче в срок (обычно 2 недели после выдачи) работающей и протестированной программы + отчет + ответы на вопросы.
- После срока баллы снижаются.
- **Без отчета задача не принимается.**
- Первые **5** человек в группе, сдавшие задачу + доп. балл. Первые **3** сдавшие все задачи + доп. баллы

Цель курса

Научить:

- эффективно решать задачи выбора структур данных и представления их в ЭВМ в зависимости от решаемой задачи и доступных вычислительных ресурсов;
- конструировать средствами используемого языка программирования новые типы данных, соответствующие специфике решаемой задачи.

Литература

Основная учебная литература

- **Кормен Т., Лейзерсон Ч., Риверс Р.** Алгоритмы: построение и анализ: Пер. с англ. М.: Издат. дом «Вильямс»
- **Ахо А., Хопкрофт Д., Ульман Д.** Структуры данных и алгоритмы.: Пер. с англ. М.: Издат. дом «Вильямс».
- **Вирт Н**. Алгоритмы и структуры данных: Пер. с англ. СПб.: Невский диалект, 2007. - 352с.
- **Иванова Г.С**. Основы программирования. Учебник для ВУЗов. − М.: Изд-во МГТУ им. Н.Э. Баумана
- **Иванова Г.С.** Технология программирования: Учебник для ВУЗов. − М.:Изд-во МГТУ им.Н.Э.Баумана

Дополнительная учебная литература.

- **Кнут Д.** Искусство программирования, Т. 3. Сортировка и поиск: Пер. с англ. М.: Издат. дом «Вильямс»
- **Седжвик Р.** Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск/Алгоритмы на графах: Пер. с англ. СПб.: «ДиаСофтЮП»
- **Тассел В.** Стиль, разработка, эффективность, отладка и испытание программ: Пер. с англ. М.:Мир
- **Кент Б**. Экстремальное программирование: Пер. с англ. СПб.: Питер
- **Писсанецки С.** Технология разреженных матриц.: Пер. с англ. - М.: Мир
- **Керниган Б., Пайк Р.** Практика программирования: Пер. с англ. М.: Издат. дом «Вильямс»
- **Баунси Кен** Основные концепции структур данных и реализация в С++: Пер. с англ. М.: Издат. Дом «Вильямс»

Электронные ресурсы

- Сайт центра дистанционного обучения МГТУ http://wwwcdl.bmstu.ru (иу)
- Методические указания к лабораторным работам по курсу «Типы и структуры данных» М.Ю. Барышникова, А.В. Силантьева

Языки программирования
Типы данных

Языки программирования, классификация

- В зависимости от того, в каких терминах необходимо описать задачу, все языки программирования делят на языки **низкого** и **высокого уровня**
- язык программирования близкий к естественному языку называется языком высокого уровня (ЯВУ), если язык программирования ближе к машинным командам, он относится к языкам низкого уровня

Уровни языков программирования

Классификация языков программирования

Языки низкого уровня:

- *Машинно-зависимые - ассемблеры*
- *Машинно-ориентированные – Си*

Языки высокого уровня:

- *Процедурные*
- *Непроцедурные*

ИСТОРИЯ РАЗВИТИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

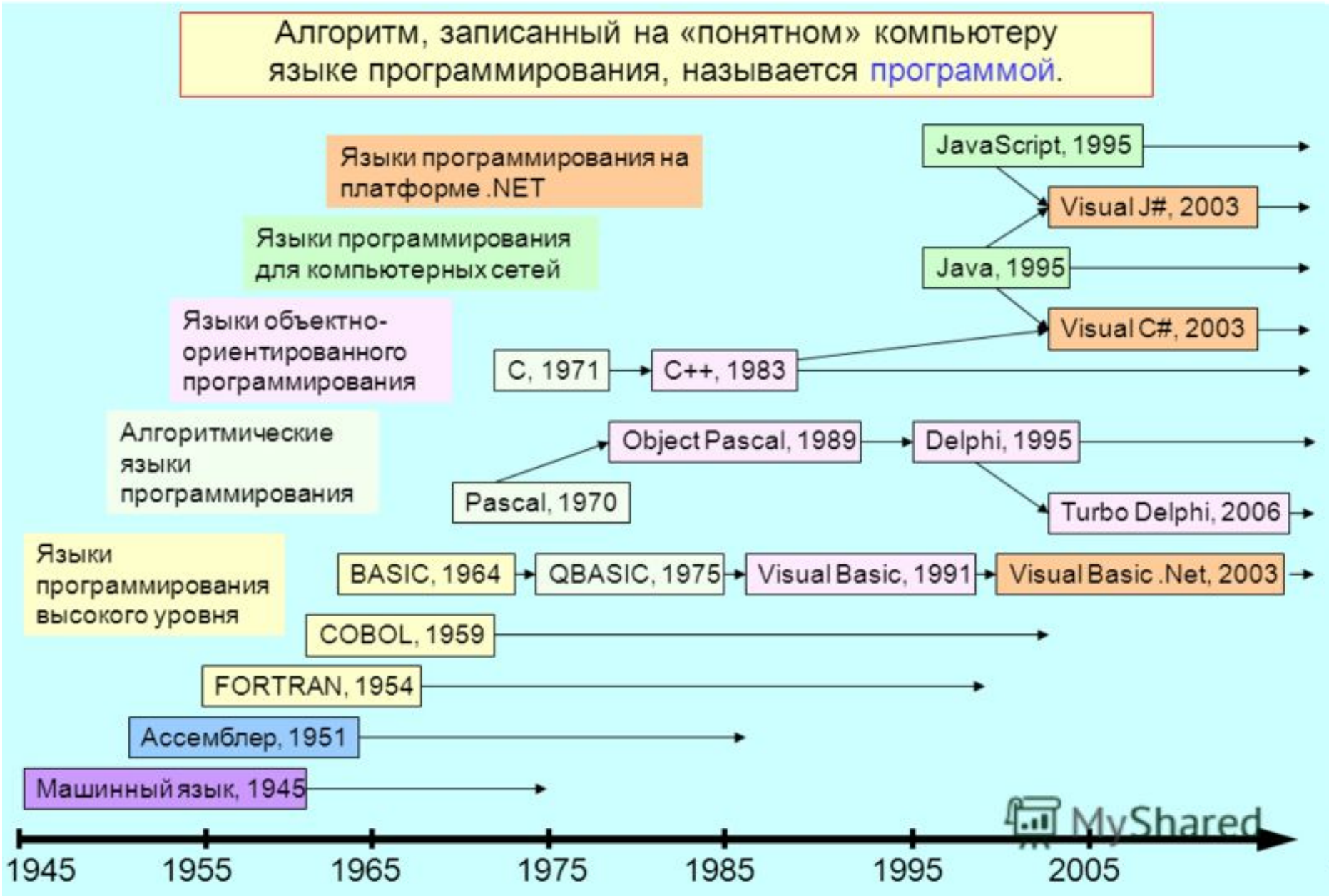
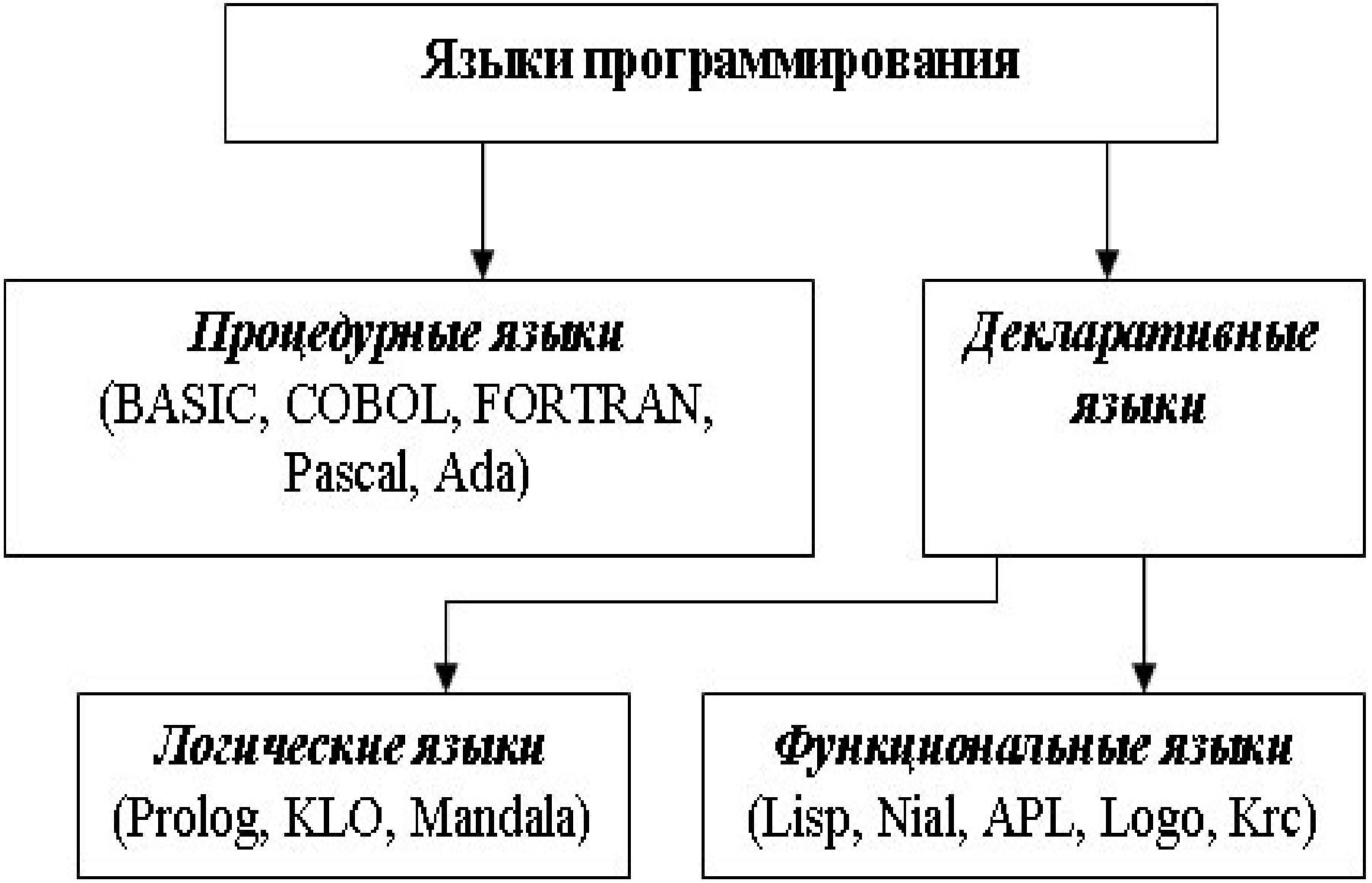


Рисунок 1 - Классификация парадигм программирования (по Д.Федюшину).


Языки программирования
Процедурные языки
(BASIC, COBOL, FORTRAN,
Pascal, Ada)
Декларативные
языки
Логические языки
(Prolog, KLO, Mandala)
Функциональные языки
(Lisp, Nial, APL, Logo, Krc)

Процедурное программирование

Отражает принципы классической архитектуры фон Неймана.

- Программа представляет собой последовательность команд, определяющая алгоритм решения задачи.
- Использование памяти для хранения данных
- Концепция памяти, как хранилища значений, содержимое которой может обновляться операторами программы, является фундаментальной.
- Основной оператор – присваивание, служащий для изменения содержимого областей памяти
- Программа преобразует содержимое памяти, изменяя его от исходного состояния к результирующему.

Необходимость использования типов данных

Концепция типа данных появилась в языках программирования высокого уровня как естественное отражение того факта, что:

- обрабатываемые программой данные могут иметь различные множества допустимых значений,
- храниться в памяти компьютера различным образом,
- занимать различные объёмы памяти
- обрабатываться с помощью различных команд процессора

Типы данных

- Тип (сорт) — относительно устойчивая и независимая совокупность элементов, которую можно выделить во всём рассматриваемом множестве (предметной области)
- Полиморфный тип — представление набора типов как единственного типа
- Математически тип может быть определён двумя способами:
  - множеством всех значений, принадлежащим типу
  - предикатной функцией, определяющей принадлежность объекта к данному типу

Практическое применение

- Наличие **типовых** описаний констант, переменных и функций и предписанные правила определения типов выражений вместе с поддержкой свойства инкапсуляции типов дают возможность компиляторам языков программирования и языков баз данных **производить** существенный **контроль** допустимости **языковых конструкций на этапе компиляции,** что позволяет сократить число проверок на стадии выполнения программ и облегчить их отладку
- Сильная типизация делает процесс программирования более сложным, но даёт в результате программы, содержащие заметно меньше труднообнаруживаемых ошибок

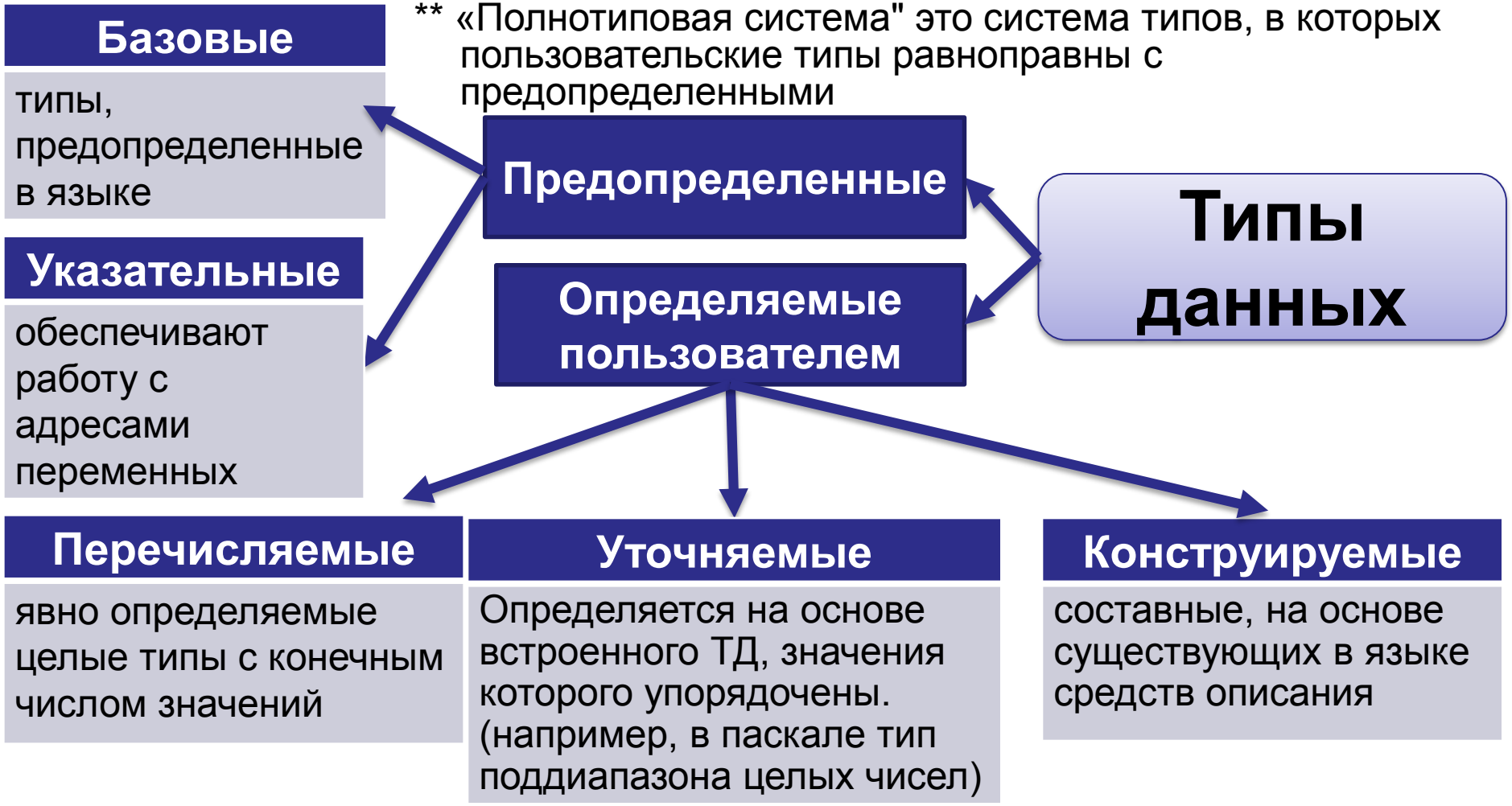
Преимущества использования типа данных:

- Надежность (защита от 3-х ошибок: некорректное присваивание, операция, передача параметров)
- Стандартизация –программисты могут быстро менять свои раб. Инструменты. А программа не требуют больших переделок

Тип данных опрёделяет:

- Объем оперативной памяти, выделяемый под размещение переменной данного типа
- Диапазон допустимых значений
- Диапазон допустимых операций над данными

Классификация типов данных по Вирту



Встроенные (предопределенные) типы данных

Тип CHARACTER (или **CHAR**) в зависимости от языка программирования — это:

- один из символов алфавита, зафиксированного в описании языка (для большинства языков англоязычного происхождения этот алфавит соответствует кодовому набору ASCII)
- произвольная комбинация нулей и единиц, размещаемых в одном байте

Операции:

1. Сравнения
2. Целочисленная арифметика

Логический тип

- Тип **BOOLEAN** в тех языках, где он явно поддерживается, содержит два значения — **TRUE** (истина) и **FALSE** (ложь)
- Несмотря на то, что для хранения значений этого типа теоретически достаточно одного бита, обычно в реализациях переменные этого типа занимают один байт памяти
- Для всех типов данных, для которых определены операции сравнения, определены также и правила, по которым эти операции сравнения вырабатывают булевские значения

Операции:

1. Конъюнкция – логическое «И» (&&, AND)
2. Дизъюнкция – логическое «ИЛИ» (||, OR)
3. Отрицание – логическое «НЕ» (!, NOT)
4. Другие логические операции, предусмотренные языком.

В языках линии **C** прямая поддержка булевского типа данных отсутствует, но имеется логическая интерпретация значений целых типов.

- Значением операции сравнения может быть "0" (FALSE) или "1" (TRUE)
- Значение целого типа "0" интерпретируется как FALSE, а значения, отличные от нуля, – как TRUE.
- В остальном, все работает как в случае наличия явной поддержки булевского типа

Целые типы

- В общем случае включает подмножество целых чисел,

определяемое числом разрядов **n,** используемых для внутреннего представления значений в диапазоне

- от 0 до 2^n-1
- Для представления знаковых целых (т.е. значений, симметричных относительно нуля) приходится тратить один бит на значение знака числа и при использовании **n** бит для внутреннего представления целого соответствующий тип содержит значения в диапазоне от -2^{n-1} до 2^{n-1}-1
- В подавляющем большинстве современных процессоров отрицательные целые числа обычно представляют в дополнительном коде .
- Операции: арифметические, логические.
- Нет ошибок переполнения

Вещественные типы

- **Числа с плавающей точкой.** Базовое название **REAL, FLOAT или SINGLE**
- Диапазон и точность уточняются в реализации и, как правило, существенно зависят от особенностей процессора
- В языках семейства **C** (32-разрядных)

  float (обычно с размером 16 бит),

  double float (размером в 32 бит)

  long double float (размером 64 бит)
- Операции: математические,

  сравнения.

Равенство – не рекомендуется.

Ошибки – переполнения порядка, накопление погрешности.

Уточняемые типы данных

- На базе встроенного **порядкового** типа данных можно описать новый, используя свойство упорядоченности встроенного типа (например для c1 <= c2)
- Определение нового уточненного типа может иметь вид:

  TYPE T = [c1..c2] (пример из языка Модула-2)
- Основная проблема уточняемых типов - потребность в динамическом контроле значений, формируемых при вычислении выражений и возвращаемых функциями

Перечисляемые типы данных

- В классическом варианте (например в **Паскале)**, типа состоит из перечисления имен значений – литеральных констант этого типа и должны отличаться от литерального изображения констант любого другого типа
- Т.к. значения типа перечисляются, то каждому значению можно однозначно сопоставить натуральное число от 1 до n, где n — число значений перечисляемого типа

Операции:

- получения значения по его номеру
- получения номера по значению.
- Сравнения
- получения следующего и предыдущего значения

- В языках линии **C** под "перечисляемый тип" понимается другое, при определении такого типа явно сопоставляется имени значения некоторое целое (не обязательно положительное) число; при отсутствии явного задания целого первому элементу перечисляемого типа неявно соответствует 0, а каждому следующему — целое значение, на единицу большее целого значения предыдущего элемента.

- При этом:

  - использование имени перечисляемого типа для объявления переменной эквивалентно использованию типа **integer**, и такая переменная может содержать любое целое значение;

  - имена значений перечисляемого типа на самом деле понимаются как имена целых констант, и к этим значениям применимы **все операции над целыми числами**, даже если они выводят за пределы множества целых значений элементов перечисляемого типа. Так что перечисляемый тип в смысле языка C — это не совсем тип в строгом смысле этого слова, а скорее удобное задание группы именованных констант целого типа

Указатели

- Понятие указателя в языках программирования является абстракцией понятия машинного адреса
- Для объявления указательных переменных служат т. наз. указательные, или ссылочные типы
- Для определения указательного типа, значениями кот. являются указатели на переменные встроенного или ранее определенного типа T0, в языке **Паскаль** используется конструкция:
  type T = ^T0
- В языке **C** отсутствуют отдельные возможности определения указательного типа, и, чтобы объявить переменную **var**, которая будет содержать указатели на переменные типа T0, используется конструкция **T0 *var**

- В языках линии Паскаль переменной указательного типа можно присваивать только значения, вырабатываемые встроенной процедурой **динамического** выделения памяти **new**, значения переменных того же самого указательного типа и специальное "пустое" ссылочное значение **nil**, которое входит в любой указательный тип
- Не допускаются преобразования типов указателей и какие-либо арифметические действия над их значениями. С переменной-указателем **var** можно выполнять только операцию **var^,** обеспечивающую доступ к значению переменной типа T0, на которую указывает значение переменной **var**

- В языках **C** и **C++** имеется полная свобода работы с указателями. С помощью операции "**&**" можно получить значение указателя для любой переменной, над указателями определены **арифметические** действия, возможно явное преобразование указательных типов и даже преобразование целых типов к указательным типам.

- В этих языках не фиксируется значение "пустых" (ни на что не ссылающихся) указательных переменных. Имеется лишь рекомендация использовать в качестве такого значения константу с символическим именем **NULL**, определяемую в библиотечном файле включения. По сути дела, понятие указателя в этих языках очень близко к понятию машинного адреса.

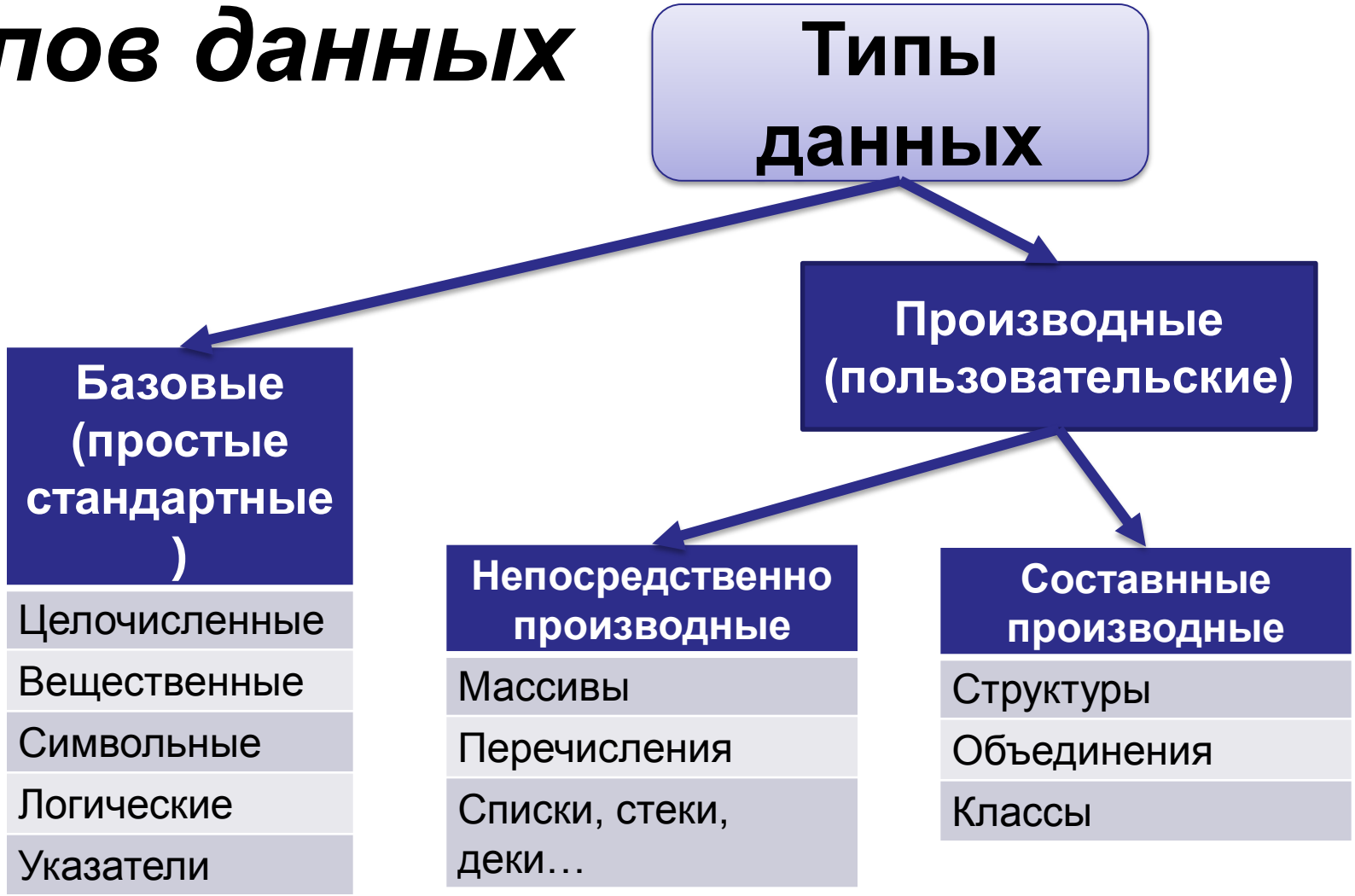
Основные понятия

- ***Программа*** – операторы или структуры языка программирования, взаимодействующие с набором информац-х объектов, над которыми производятся действия.
- ***Данные*** – это запомненная и используемая программой информация.
- Данные могут быть ***одиночным значением*** или ***набором значений***.
- Информация характеризуется ***типом данных***, кот. определяет:

1) диапазон допустимых операций над данными;
2) диапазон допустимых значений для данных;
3) количество оперативной памяти (ОП), выделяемое под размещение данного и выбор его представления в ЭВМ.

- На *машинном* уровне ***тип данного*** представляет собой ***выбор машинных команд***, в выполнении которых участвует это данное.

Классификация типов данных



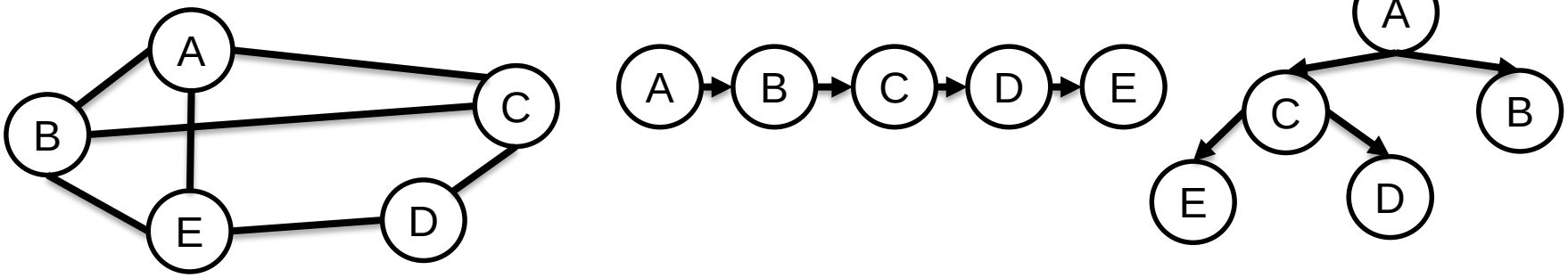
Данные

- **Простые**
- **Структура данных (СД):** совокупность элементов данных, между кот. существуют к.-л. отношения. Т. е. логическая и математическая модель организации данных называется структурой данных.
- Выбор СД зависит от следующих параметров:
- СД должна отражать фактические связи данных в реальном мире.
- СД д. б. достаточно проста, для простоты ее обработки.
- Т.о. ***СД – это организация и тип данных,***
- Она характеризует:
- 1. характер организации данных;
- 2. множество допустимых значений;
- 3. набор допустимых операций.

СД (S) – это ***множество элементов данных (D) и множество отношений между ними (R):***

- **S = (D,R)**

Пример. Пусть есть 5 элементов типа char: ‘A’, ’B’, ’C’, ’D’, ’E’. Отношения между ними можно представить:



Как видно из примера, способ построения структур данных определяется характером связей между элементами.

Все связи одного элемента данных с другими образуют элементы отношений, которые можно представить в виде конечного множества пар, каждая из которых отражает отношение этого элемента с другим и указатель этого элемента.

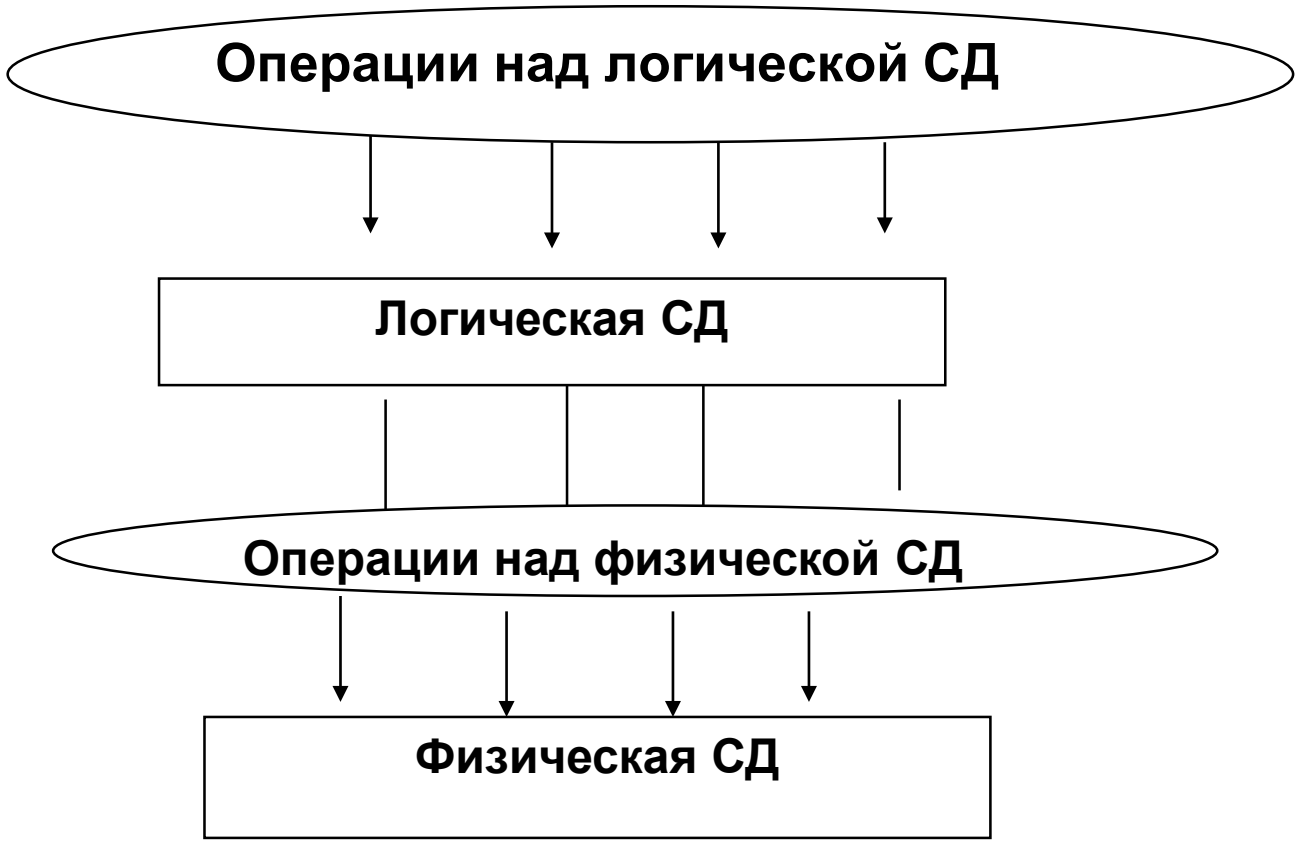
При этом связи элемента данных с другими элементами данных являются одновременно связями с соответствующими элементами структуры.


Структуры данных по организации взаимосвязей
Линейные:
Массив, список, стек, очередь, хэш….
Иерархические:
Дерево, иерархический список….
Сетевые:
Граф (ориентированный, взвешенный…)
Табличные:
N-мерные массивы, таблицы базы данных
Другие: лес
(список непересекающихся деревьев)

Обобщенная СД делится на логическую СД и физическую СД.

Логическая (абстрактная) СД	**Физическая (конкретная) СД**
Идеальная схема представления или модель. Напр., матрица как прямоугольный двумерный массив, каждый элемент кот. характеризуется двумя индексами: строки и столбца.	***Схема размещения и хранения данных в памяти***. Напр., матрица как последовательность ячеек памяти, каждая из кот. содержит один элемент массива, доступ к кот. осуществляется по одному адресу.

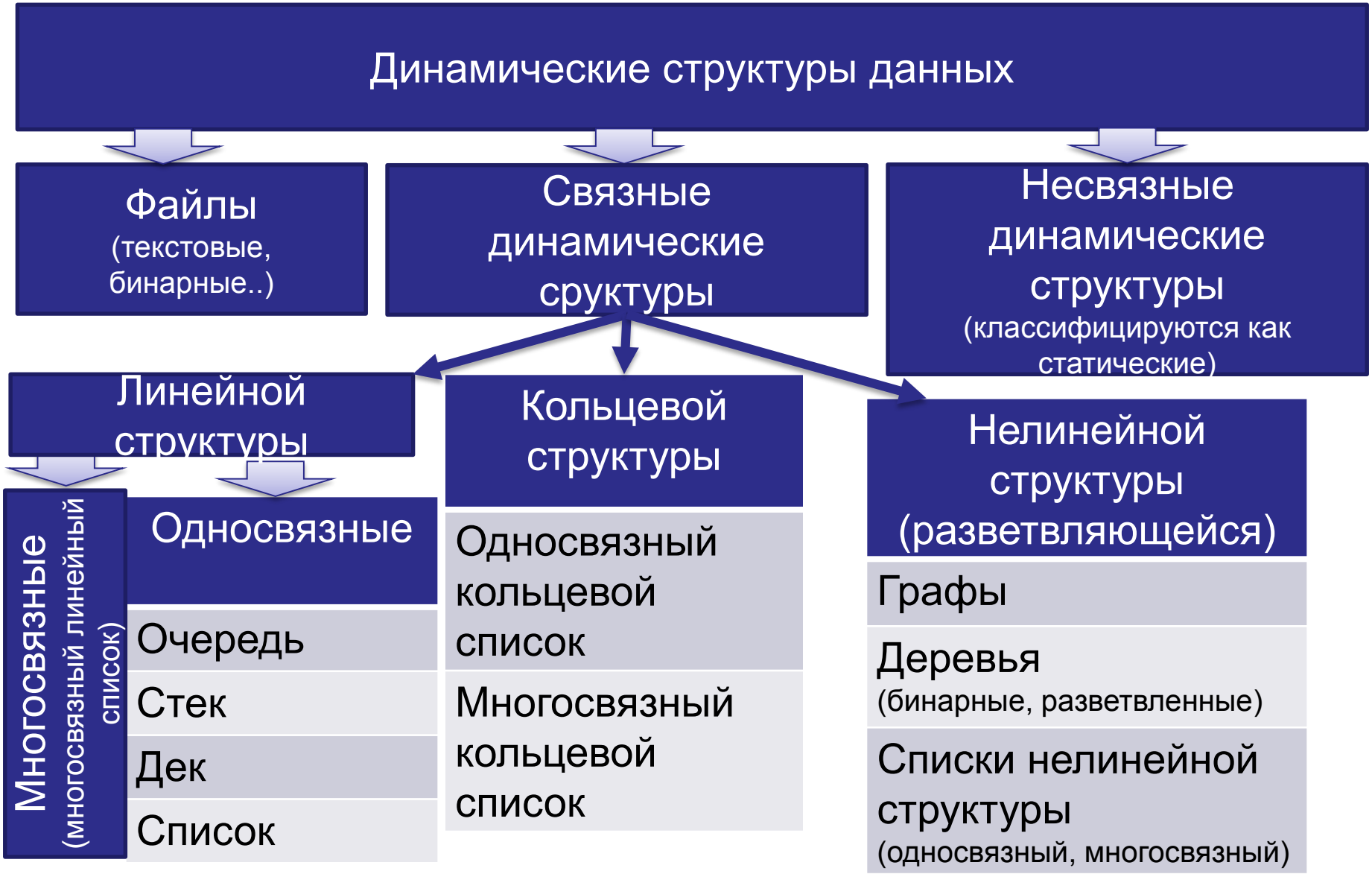
- Т. О., каждая ***СД определяется логическим и физическим представлением и совокупностью отношений на этих двух уровнях представления СД***


Операции над логической СД
Логическая СД
Операции над физической СД
Физическая СД

- Т.е., при решении любой задачи необходимо определить множество данных, описывающее конкретную ситуацию и разработать алгоритм решения задачи

- В конечном счете, **программа** представляет собой ***конкретную*** **формулировку *абстрактного* алгоритма**, основанную на ***конкретных*** **представлениях и структурах данных.**
  - Таким образом, по формулировке Н. Вирта,
    - **АЛГОРИТМ + СД = ПРОГРАММА**
      - **С наших позиций:**
    - **СД + АЛГОРИТМ = ПРОГРАММА**


Статические структуры данных
Простые
Составные
Простые
Символы
Логические
Указатели
Числа
(вещественные, целые, натуральные, комплексные)
Определяемые программистом
Интервалы
Перечисления
Однородные
Строки
Множества
Массивы
(по размерности: одномерные, многомерные, по типу данных: однородные, неодногодные)
Неоднородные
Структуры
Записи
(простые, вариантные)
Объединения
Объекты


Динамические структуры данных
Файлы
(текстовые, бинарные..)
Связные динамические сруктуры
Несвязные динамические структуры
(классифицируются как статические)
Линейной структуры
Кольцевой структуры
Нелинейной структуры (разветвляющейся)
Односвязные
Очередь
Стек
Дек
Список
Многосвязные
(многосвязный линейный список)
Односвязный кольцевой список
Многосвязный кольцевой список
Графы
Деревья
(бинарные, разветвленные)
Списки нелинейной структуры
(односвязный, многосвязный)

Простые типы данных

- **целые**,
- **вещественные**,
- **указатели**
- **символьные**,
- **логические**.

Лабораторная работа № 1

Задание:

- **Составить программу умножения или деления вещественных чисел. Программа должна осуществлять проверку ввода чисел и выдавать верный результат в указанном формате, либо выдавать сообщение о невозможности произвести счет.**
- При этом числа будут представляться в виде:
- **М А Н Т И С С А П О Р Я Д О К**

Знак мантиссы Поле мантиссы Знак порядка Поле порядка

- **--- 1 0 ---------- 30 --- 1 0 ----- 5**

- Мантисса может представляться с точкой и без нее
- **.000…25,** или **123….001 или 123.**
- Порядок при вводе может быть или не быть.
- При выдаче результат должен быть в виде: **0.мантисса**.
- Т.е. перед обработкой числа необходимо нормализовывать, при этом порядок может выйти за рамки своего представления, **но это не должно приводить к ошибке**, т. к.к при обработке действительных чисел в ПК реально выделятся два регистра для операций умножения и деления

- При стандартном представлении вещественных чисел максимальная характеристика **– 65535,** а мантисса – **18438436*10**, то есть, 20 знаков.
- При обработке больших чисел, (когда порядок имеет до 5 знаков: от –99999 до +99999, а мантисса – до 30 знаков) нельзя применять стандартные ТД. Необходимо моделировать это число, используя, например, массив для мантиссы, то есть, фактически, моделировать работу регистров: обрабатывать каждый разряд числа последовательно, начиная с младшего. Но и тут возникают свои сложности:

сложности обработки:

Могут быть единицы переноса: например,

- 00……..79
- *
- 00……..99
- 63**1**
- + 8 - перенос
- 71**1**
- 631
- + 8 - перенос
- 782**1**

Если длина мантиссы больше 30, возникает необходимость округления числа (если >= 5, то +1, если < 5, то 0), при этом может возникнуть циклический поразрядный перенос из младшего разряда в старший.

При работе с порядком также может возникнуть переполнение (если порядок больше +99999) и машинный «0» (если порядок меньше – 99999).

Статические СД

Массивы

Массивы

Понятия массива и типа массива сильно различаются в сильно и слабо типизированных языках.

Классическое понятие в сильно типизированных языках (например, в языке Паскаль).

Тип массива определяется на основе двух вспомогательных типов: типа элементов массива (базового типа) и типа индекса массива.

В языке Паскаль определение типа массива выглядит:

type T = array [I] of T0;

где T0 — базовый тип, а I — тип индекса.

T0 может быть любым встроенным или ранее определенным типом. Тип индекса I должен состоять из конечного числа перечисляемых значений, т.е. быть уточненным, перечисляемым, символьным или булевским типом.

В языках линии Паскаль допускается и неявное определение уточненного типа массива.

Например, допустимы следующие определения типа массива:

type T = array [1..6] of integer; или **type T = array ['a'..'e'] of real;.**

Для выборки элемента массива используется конструкция **x[i],** значением которой является значение i-того элемента массива.

Эта же конструкция может использоваться в левой части оператора присваивания, т.е. элементы массива могут изменяться индивидуально. Кроме того, при подобной строгой типизации массивов допустимы присваивания значений переменных типа массива.

Базовым типом типа массива м. б. любой встроенный или определенный тип, в том числе и тип массива –это будет многомерный массив или матрица:

type T = array [1..10] of array [1..5] of real;

можно написать:

type T = array [1..10],[1..5] of real;

а если x — переменная типа T, то для выборки скалярного элемента вместо **x[i][j]** можно написать **x[i,j].**

- В слабо типизированных языках используем язык **C**.

- В этом языке нет средств определения типов массива, хотя имеется возможность определения "массивных переменных". Число элементов в массивной переменной определяется либо явно, либо с помощью задания списка инициализирующих значений базового типа. Напр., массивную переменную с четырьмя элементами целого типа можно определить как **int x[4]** (неинициализированный вариант) или как **int x[] = {0, 2, 8, 22}** (инициализированная массивная переменная). Доступ к элементам массивной переменной производится с помощью конструкции выбора, по виду **аналогичной** соответствующей конструкции в сильно типизированных языках **x[i],** где i — выражение, принимающее целое значение (но, в отличие от языка **Паскаль** в языке **C** зафиксирована интерпретация операции выбора на основе более примитивных операций адресной арифметики). Однако, в реализациях языка **Си** в принципе невозможен контроль выхода значения индекса за пределы массива. Кр. того, по аналогичным причинам невозможно присваивание значений массивных переменных и не допускаются функции, вырабатывающие "массивные значения".

Отмеченные свойства механизма указателей существенно повлияли на особенности реализации в языках C и C++ работы с массивами. Имя массива здесь интерпретируется как имя константного указателя на **первый** элемент массива. Операция доступа к **i**-тому элементу массива **arr** хотя и обозначается как и в языках линии Паскаль **arr[i],** имеет низкоуровневую интерпретацию ***(arr+i).** Поэтому было логично допустить подобную запись для любой переменной **var** с указательным типом: **var[i]** интерпретируется как ***(var+i).** По этой причине понятие массива в C/C++ существенно отличается от соответствующего понятия в Паскале.

Размер массива существенен только при его определении и используется для выделения соответствующего объема памяти. При работе программы используется только имя массива как константный указатель соответствующего типа. Нет операций над "массивными переменными" целиком; в частности, невозможно присваивание. Фактически отсутствует поддержка массивов как параметров вызова функций — передаются именно значения указателей (в связи с этим, при описании формального параметра-массива его размер не указывается). Функции не могут вырабатывать "массивные" значения.

- ***Массив*** **– это регулярная однородная структура со случайным доступом, т. е., все компоненты структуры ОДНОГО типа, могут выбираться произвольно и являются одинаково доступными.**
- массив – это СД, обладающая след. свойствами:

1. Все элементы массива имеют один и тот же тип.
2. Размерность массива определяется при его описании и в дальнейшем не меняется.
3. К каждому элементу массива имеется прямой доступ.

Т. о., каждый элемент массива имеет **одно имя** и свой **индекс**, т. е., номер элемента в массиве.

По размерности массивы м. б. ***одномерными*** (вектор)***, двумерными*** (матрица) и ***многомерными.***

Вектор

- **конечное упорядоченное множество элементов одного типа, называемых *элементами вектора.*** Элементы вектора связаны между собой **отношением *непосредственного следования***, т. е., они м. б. пронумерованы последовательными целыми числами (или переменными любого порядкового типа) – **индексами**, называемыми ***селекторами*** **элементов вектора*.***

Описание векторов в программе должно содержать:

- Имя вектора;
- Минимальный и максимальный индексы элементов;
- Длину элемента (т. е., размер поля памяти, выделенной под элемент).

Важнейшей операцией над векторами является организация доступа к элементу вектора.

доступ к элементу вектора

- Различают ***логический*** и ***физический (структурный, системный) уровни доступа*** к элементам вектора.
- ***Логический уровне доступа:*** указывают **имя и индекс**

- ***Физический уровень доступа*** предполагает **вычисление адреса** элемента, исходя из его имени, индекса и длины.
- Память ЭВМ – это последовательность ячеек, имеющих адреса.
- Элементы вектора в памяти располагаются **последовательно**, по возрастанию индексов, поэтому к каждому элементу возможен **прямой доступ.** Если известен адрес начала вектора, можно вычислить адрес любого элемента.
- Физической структуре ставится в соответствие дескриптор (заголовок), который содержит общие сведения о физической структуре.

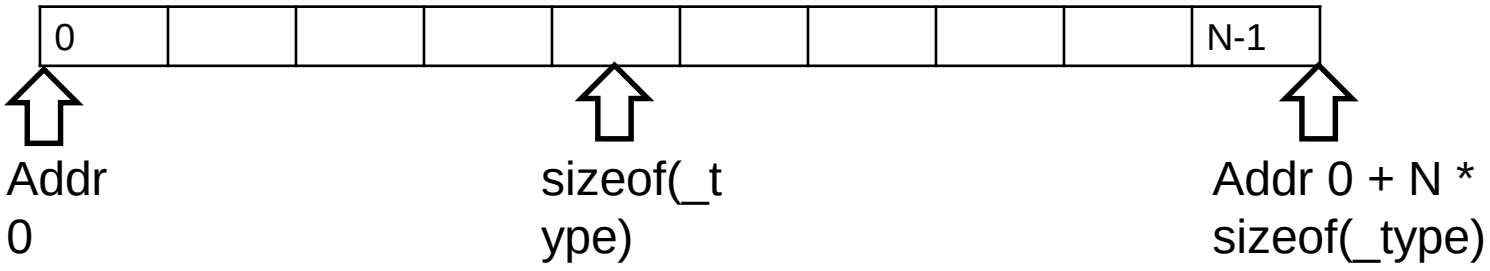
Вектор. Машинное представление. Адресация элементов структуры

Элементы вектора размещаются в памяти в подряд расположенных ячейках памяти.

Под элемент вектора выделяется количество байт памяти, определяемое базовым типом элемента этого вектора.

Необходимое число байтов памяти для хранения одного элемента вектора называется ***слотом***.

Размер памяти для хранения вектора определяется произведением длины слота на число элементов.



дескриптор (заголовок) вектора

Пример: Дан вектор **Var V: array [I1..K1] of T;**
Дескриптор вектора V содержит след. информацию:
Имя – **V**
Адрес 1-го элемента - **Addr (V[I1])**
Индекс первого элемента - **I1** (для Си = 0)
Индекс последнего элемента - **K1**
Тип элемента - **T**
Длина элемента - слот - **L**
При этом или тип, или слот относятся к избыточной информации.
Доступ к элементам вектора:

Элемент1	Элемент2	…	ЭлементK1
Addr (V[I1])	Addr (V[I1])+L		Addr (V[I1])+L*(K1-I1)

Т. о. доступ к J-му элементу:

Addr (V[J1]) = Addr (V[I1]) + (J1 – I1) * L = Addr (V[I1]) – I1 * L + J1 * L
= D + J1 * L, – это есть линейная функция адресации
где **D = Addr (V[I1]) – I1 * L**

Для СИ:

Обращение к i-тому элементу вектора выполняется по адресу вектора плюс смещение к данному элементу. Смещение i-ого элемента вектора определяется по формуле

Byte_Num = i * sizeof(type)

Адрес i элемента:

&Byte_Num = &V_name + Byte_Num

Программисты Си часто вместо выражения вида: **Имя[i]**

употребляют выражение вида: ***(Имя+i)**

Двумерные массивы (матрицы)

Двумерным массивом или ***матрицей*** называется **одномерный массив, компонентами которого являются вектора.**

Доступ к любому из элементов матрицы осуществляется **по двум** индексам: **номеру строки** и **номеру столбца.**

Двумерный массив хорошо иллюстрирует различие между ***логическим и физическим*** представлением данных.

Логически матрица представляется прямоугольным массивом, каждый элемент которого характеризуется двумя индексами: строки и столбца.

Физически матрицы можно хранить двумя способами: отображением **по строкам** и отображением **по столбцам** (способ расположения в памяти зависит от языка программирования). При различном отображении в памяти адрес одного и того же элемента матрицы будет различен.

Из определения матрицы следует, что матрица реализуется структурой ***вектора векторов:***

Mt[I1..K1, I2..K2] или Mt[I1..K1][I2..K2];

Для того чтобы развернуть этот вектор в ”линию”, нужна ***линейная функция адресации***

Дан двумерный массив: **Var Mt: array[1..n, 1..m] of T;**

При отображении **по строкам** получается след. функция адресации:

Addr (Mt[I,J]) = Addr (Mt[1,1]) + L* (m * (I-1) + (J-1)),

при отображении **по столбцам** функция адресации::

Addr (Mt[I,J]) = Addr (Mt[1,1]) + L* (n * (J-1) + (I-1))

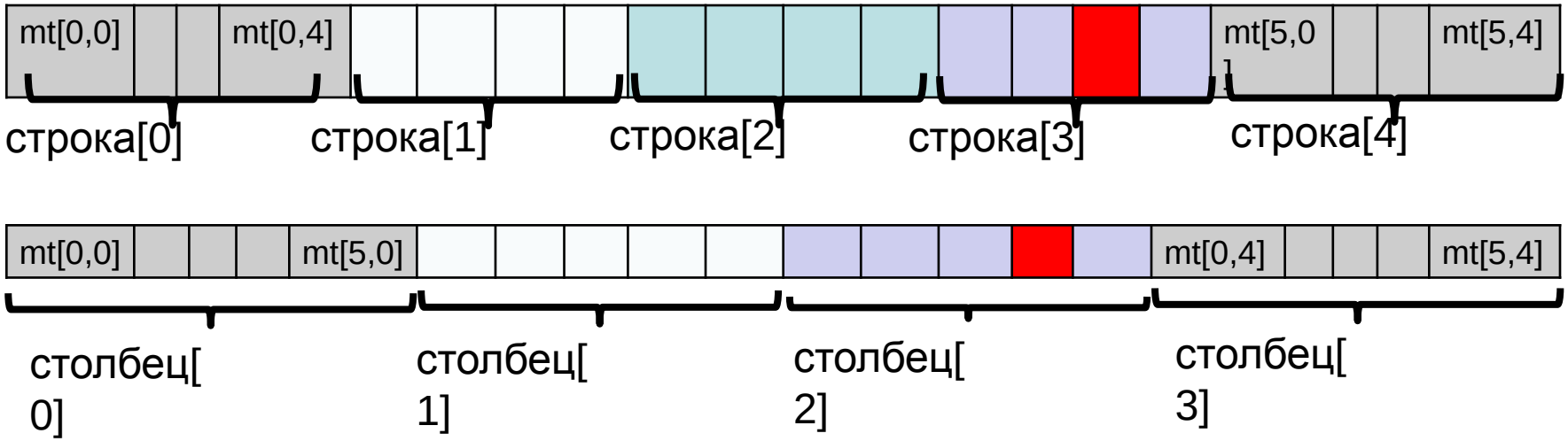
Для С: Дан двумерный массив: **_type mt [n][m];**

Функция адресации при отображении по **строкам**:

&mt[i][j] = &mt[0][0] + L * (m * i + j),

Функция адресации при отображении по **столбцам**:

&mt[i][j] = &mt[0][0] + L * (n * j + i).



для массива: **mt(5*4) при &mt[0][0] = 200** и **L = 4**
адрес элемента **mt[3,2]** (n = 5, m = 4, i = 3, j = 2) будет:
а) при расположении по строкам:
200 + 4 * (4 * 3 + 2) = **256**
б) при расположении по столбцам:
200 + 4 * (5 * 2 + 3) = **252**

Многомерный массив

Аналогично матрице многомерный массив:

M[I1..K1, I2..K2, ……, In..Kn];

реализуется как вектор массивов на 1 меньшей размерности.

Многомерный массив **логический** массив можно преобразовать в **физическую** одномерную структуру путем процесса линеаризации: линейной последовательностью элементов массива.

Надо только иметь в виду, что при представлении массива по **«строкам»** быстрее всего меняется последний индекс, а при представлении по **«столбцам»** – первый индекс.

- Общая формула для вычисления произвольного элемента
- n – мерного массива **M** (J**1**,J**2**,……,J**n**),
- если известен адрес его первого элемента: **M** [I**1**,I**2**,……,I**n**] и длина элемента **L**, будет выглядеть так:
  - Addr (M[J**1**,J**2**,……,J**n**]) =
- Addr (M[I**1**,I**2**,……,I**n**]) – L * $\sum_{m=1}^{n} I_m D_m$ + L * $\sum_{m=1}^{n} J_m Đ$
- постоянная часть для одного массива (п.ч.)
- = Addr (M[I**1**,I**2**,……,I**n**]) – L * $\sum_{m=1}^{n} I_m$ * Dm + L * $\sum_{m=1}^{n} J_n$ * Dm
- можно вычислить заранее

- Коэффициенты D**1**, … ,D**m** можно вычислить по рекуррентным формулам:
- а) **при отображении по строкам**:
- D**m** = ($K_{m+1} – I_{m+1} + 1$) * D_{m+1}, где m = n – 1,…,1 и D**n** = 1
- б) **при отображении по столбцам:**
- D**m** = ($K_{m-1} – I_{m-1} + 1$) * D_{m-1} , где m = 2,…,n и D_1 = 1,
- где I**m –** начальный индекс, а K**m** – конечный индекс m – ной размерности массива.

Пример:

Составить дескриптор матрицы из целых чисел Mt [2..3,-1..1] при отображении по строкам.

Размерность массива n = 2, длина элемента = 4;

I1 = 2, K1 = 3, I2 = -1, K2 = 1;

m = n – 1 = 2 – 1 = 1, D**n** = D**2** = 1;

D**1** = (K2 – I2 + 1) * D**2** = (1 - (-1) + 1) * 1 = 3;

L * Dm = L * (I1 * D**1** + I2 * D**2**) = 4 * (2 * 3 + (-1) * 1) = 20

Для ускорения вычислений адресных выражений эффективно еще на этапе трансляции вычислить и занести в дескриптор индексные множители L * Dm.

Для нашего примера индексные множители будут такими:

L * D**1** = 4 * 3 = 12; L * D**2** = 4 * 1 = 4

Таким образом, мы получили дескриптор с точностью до начального адреса массива:

Имя	**Адрес первого Элемента Addr (M [2,-1])**	**Постоянная часть: addr (M [2,-1]) - 20**	Раз-мер-ность	I1	K1	I2	K2	**L * D1**	**L * D2**	Длина Эле-мента	Тип Эле-мента
Mt	4	4-20= -16	2	2	3	-1	1	12	4	4	T

Значит, адрес, напр., элемента **Mt[3,0]** при начальном адресе = 4, J1 = 3 и J2 = 0 следующий:

4 – 20 + **3** * 12 + **0** * 4 = 20

Постоянная часть — Индексные множители

Т.к. количество индексов равно размерности массива, то при этом методе получения адресных выражений **количество умножений пропорционально размерности массива.**

Рассчитать дома:

- Пример:
- Дано: матрица A (6,4), длина L = 6,
- Addr(A[1,1]) = 100;
- Определить элемент матрицы A[3,4]
- а)при расположении по строкам;
- б) при расположении по столбцам.

- Составить дескриптор матрицы
- Осуществить проверку расчета адресов

Вектора Айлиффа

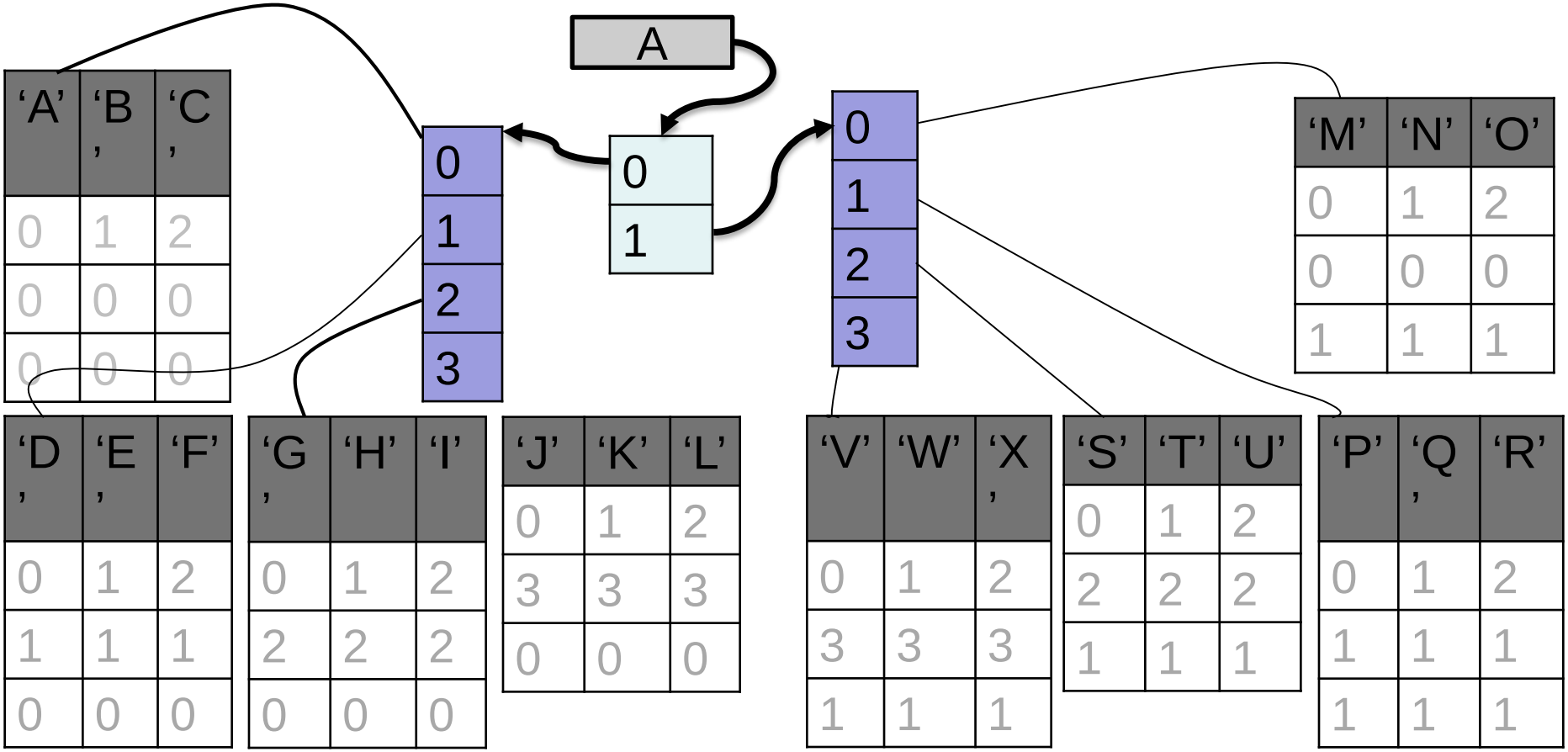
Для ускорения вычислений можно исключить умножение, используя так наз. **метод *векторов Айлиффа***.

Вектор Айлиффа – ***вспомогательный дескриптор.*** Каждый элемент в-ра Айлиффа содержит указатель на дескриптор следующего, более низкого уровня, а дескр-р самого низкого уровня указывает на группы элементов массива. Основной дескриптор массива (нулевой деск-р) хранит указатель вектора Айлиффа первого уровня. Каждый из указателей адресует обл-ть памяти, соответствующую **нулевому** значению индекса. После прибавления к указателю значения индекса получаем адрес соот-ющего компонента вектора Айлиффа или, в случае вектора нижнего уровня, – адрес требуемого элемента массива. Т. о., к элементу массива M[J**1**,J**2**,……,J**n**] можно обратиться, пройдя по цепи от основного дескр-ра через все компоненты вектора Айлиффа, после чего и будет получен адрес требуемого элемента:

Addr (M[J**1**,J**2**,……,J**n**]) = (…(…(D + J**1**) + J**2**) +…) + J**n** ,

где D – значение указателя вектора Айлиффа первого уровня, хранящееся в основном дескрипторе.

Адресация элементов с помощью векторов Айлиффа для трехмерного массива A[3][4][2]

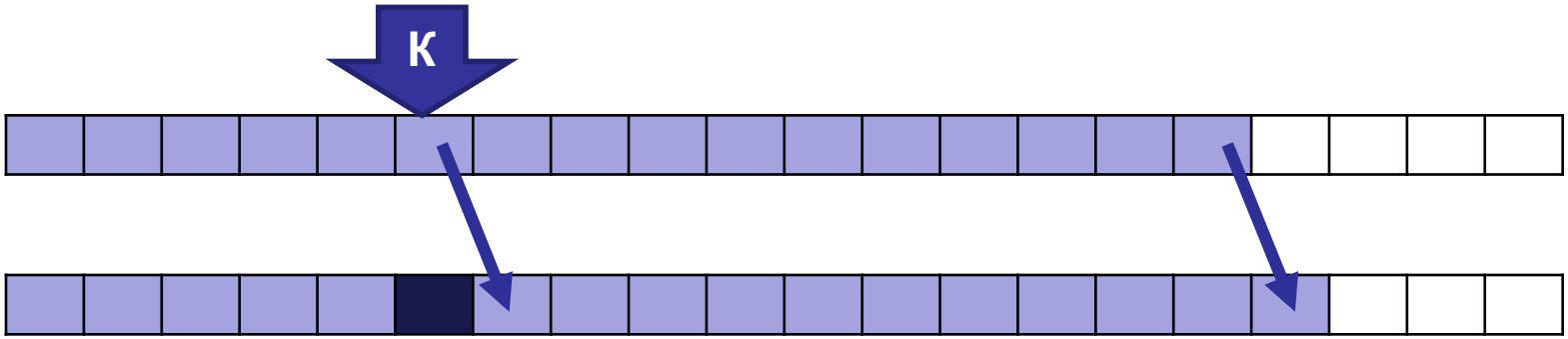


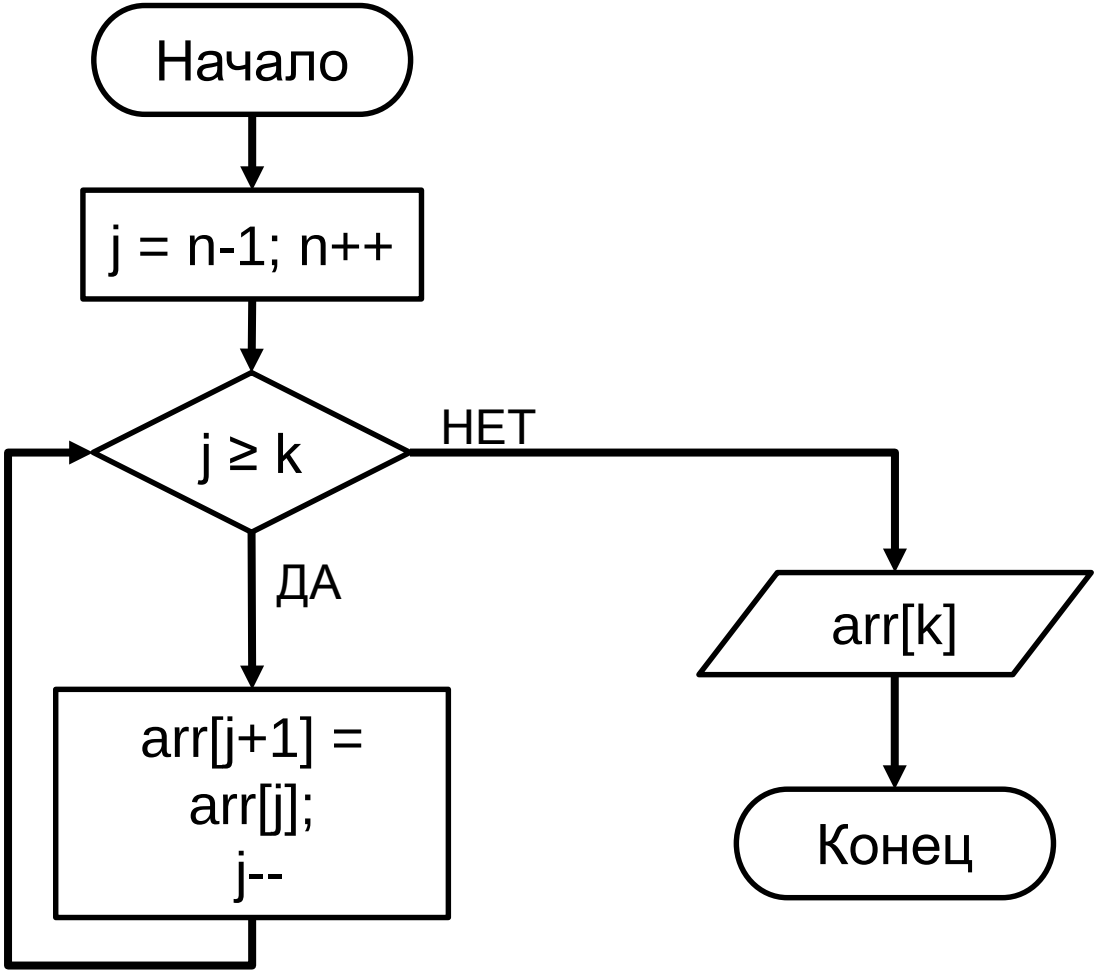
Операции над созданными структурами данных

- **Обход структуры:** доступ к каждому элементу структуры («посещение» элемента);
- **Поиск:** нахождение расположения элемента с данным значением (ключом);
- **Вставка:** включение нового элемента в структуру;
- **Удаление:** исключение элемента из структуры.

Алгоритм вставки элемента в массив на позицию k

1. Проверить, есть ли k-я позиция в описанных индексах массива и не полон ли массив (тогда вставлять нельзя!)
2. Увеличить количество элементов массива на 1
3. Переместить на одну позицию вправо все элементы массива, начиная с конца до позиции вставки.
4. Вставить в освободившуюся позицию элемент.




Начало
j = n-1; n++
j ≥ k
НЕТ
ДА
arr[k]
arr[j+1] =
arr[j];
j--
Конец

Алгоритм удаления элемента на позиции k из массива

1. Проверить, есть ли k-я позиция в описанных индексах массива и не последний ли он.
2. Начиная с k+1 элемента переставить каждый последующий элемент на место предыдущего и
3. уменьшить количество элементов массива на 1