- Мой электронный адрес для переписки со студентами:

kash@telegraf.by

- Адрес «облака» с методическими материалами:

https://drive.google.com/folderview?id=0Bymq5SA5EB3NflhYX3c4S2tIQ25kYTJ4Q3Y4X19iRWVxUjE1TGJnNVNMMHdSbDFuUzh6SUE&usp=sharing

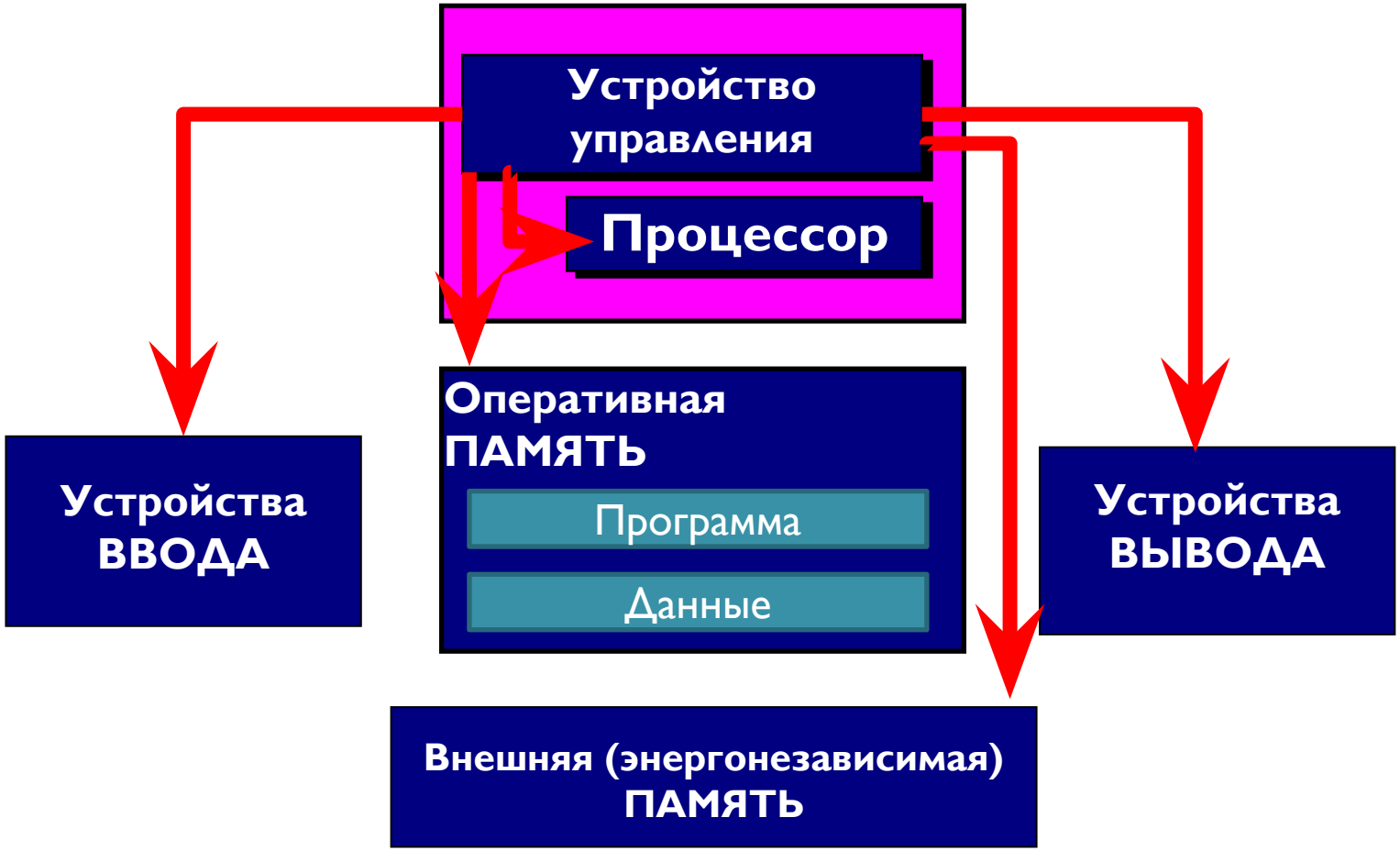
Тема I

Компьютер и его программное обеспечение

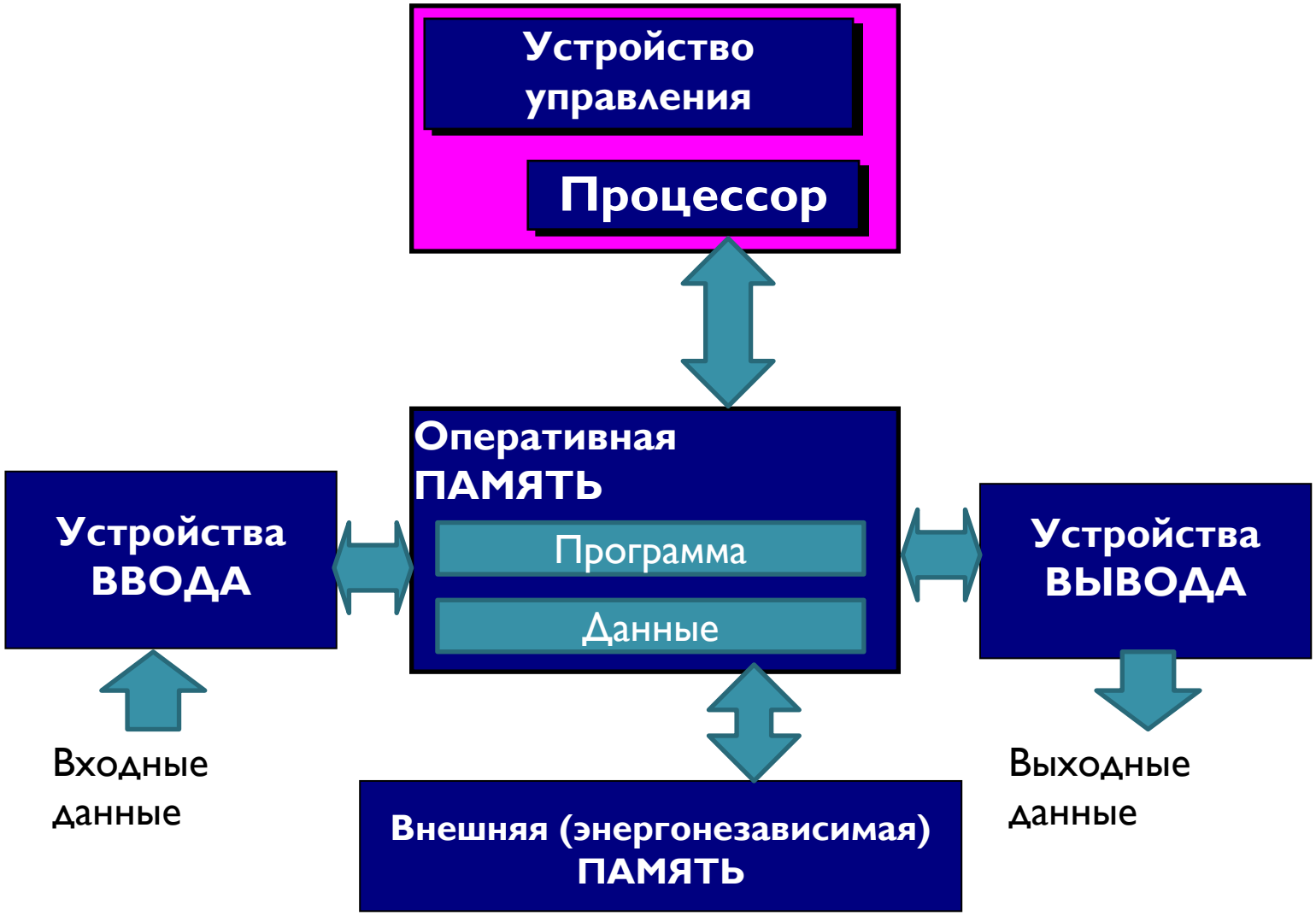
Определение компьютера

КОМПЬЮТЕР (англ. computer, от лат. computo — считаю) - машина для приема, переработки, хранения и выдачи информации в электронном виде, которая может воспринимать и выполнять сложные последовательности вычислительных операций по заданной последовательности инструкций — программе

Принципиальная схема компьютера (потоки управления)



Принципиальная схема компьютера (потоки данных)



Представление информации в компьютере

- Информация в компьютере хранится в виде последовательности символов двоичного алфавита – 0 или 1, каждый из которых представляется одним из двух устойчивых состояний некоторого физического объекта.
- Совокупность таких физических объектов составляет память компьютера.

Уровни памяти

1. **Память центрального процессора (ЦП)**
   - Регистры ЦП
   - Внутренняя кэш-память ЦП
2. **Оперативная память**
   - Внешняя кэш-память
   - Память с произвольным доступом (оперативное запоминающее устройство – ОЗУ)
   - Постоянное запоминающее устройство ROM (Read-Only Memory)
3. **Внешняя память**
   - Устройства оперативного хранения информации
   - Устройства резервного хранения и переноса информации

Адресация оперативной памяти

Оперативная память состоит из ячеек одинакового размера. Каждая ячейка имеет свой уникальный номер (адрес). Адресация ячеек начинается с 0 и является непрерывной.

Наиболее часто встречающийся размер ячейки – 8 бит (двоичных цифр). Такая ячейка называется байтом.

Байт является основной единицей измерения памяти (как оперативной, так и внешней). Производные единицы измерения:

- килобайт (1024 байта)
- мегабайт (1024 килобайта)
- гигабайт (1024 мегабайта)
- терабайт (1024 гигабайта)
- …

Размещение данных во внешней памяти

Данные хранятся в виде файлов - именованных областей внешней памяти, содержащих некоторую однородную (с точки зрения операционной системы) информацию.

По методу доступа к информации устройства внешней памяти разделяются на:

- **Устройства с прямым (произвольным) доступом -** возможность обращения к блокам информации по их номерам (адресам) в произвольном порядке

- **Устройства с последовательным доступом –** обращение к блокам информации также может осуществляется по номерам (адресам), но доступ к каждому блоку возможен только путем холостого чтения (сканирования) всех предшествующих блоков

Архитектура фон Неймана

Джон фон Нейман

Родился 3 ноября 1903 г. в Венгрии в богатой еврейской семье.

Первая научная работа – 1921 г.

Диплом инженера-химика и одновременная защита диссертации – 1925 г.

Статья «К теории стратегических игр» – доказательство теоремы о минимаксе – 1928 г.

Книга «Математические основы квантовой механики» – 1932 г.

Разработка методов оптимального бомбометания – 1939 – 1943 гг.

Математическая модель атомной бомбы

Метод Монте-Карло (совместно с С.Уламом)

Современная архитектура компьютера – 1945 год

Умер в 1957 г.

Принципы фон Неймана

1. **Принцип использования двоичной системы счисления для представления данных и команд.**
2. **Принцип программного управления.**

Программа состоит из набора команд, которые выполняются процессором друг за другом в определенной последовательности.

3. **Принцип однородности памяти.**

Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления — чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.

Принципы фон Неймана (продолжение)

4. **Принцип адресуемости памяти**.

Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.

5. **Принцип последовательного программного управления**

Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.

6. **Принцип условного перехода**.

Команды из программы не всегда выполняются одна за другой. Возможно присутствие в программе команд условного перехода, которые меняют последовательное выполнение команд в зависимости от значений данных

Предмет программирования

- **ПРОГРАММА - описание действий, которые должен выполнить компьютер, автоматически переводимое на язык машинных команд этого компьютера.**
- **Процесс разработки программ для решения определенных задач называют ПРОГРАММИРОВАНИЕМ.**
- **Группы программ, работающих как единое целое, составляют программное обеспечение (ПО) компьютера.**

Исполняемая программа

- Выполняемая программа хранится в памяти компьютера в виде машинных команд, закодированных в виде последовательности нулей и единиц. Каждая машинная команда занимает целое количество байт (не менее одного байта).
- Как получить программу в виде машинных команд?
  - Непосредственно записать эти команды
  - Написать программу на Ассемблере
  - Написать программу на языке высокого уровня

Непосредственная запись машинных команд

- сохранение программы в виде последовательности кодов команд в файле на диске;
- загрузка программы в оперативную память;
- запуск программы (указание устройству управления адреса первой исполняемой команды)

Недостатки такого подхода:

- Необходимо помнить коды всех машинных команд (несколько сотен);
- Программа велика, т.к. машинная команда выполняет небольшой объём работы;
- Непереносимость программы на компьютеры с другой архитектурой

Написание программы на Ассемблере

- Для каждой машинной команды разрабатывается команда Ассемблера, мнемоника которой более понятна человеку. Команда Ассемблера может соответствовать нескольким машинным командам.
- Программа на Ассемблере должна быть предварительно преобразована в машинный код с помощью специальной программы – транслятора.
- В оперативную память загружается результат работы транслятора – исполняемый код.

Написание программы на языке высокого уровня

- Программа представляет собой набор операторов (инструкций), структура которых напоминает естественные языки. Язык, как правило, не привязан к конкретной архитектуре (набору машинных команд). Текст программы гораздо более короткий по сравнению с программой на Ассемблере.
- Программа на языке высокого уровня должна быть предварительно преобразована в машинный код с помощью специальной программы – транслятора. Транслятор ориентирован на конкретную архитектуру компьютера.
- В оперативную память загружается результат работы транслятора – исполняемый код.

Интерпретация и компиляция

Трансляторы программ на языках высокого уровня или Ассемблере делятся на интерпретаторы и компиляторы

- Интерпретатор обрабатывает каждую инструкцию программы независимо от других. После обработки инструкция сразу же выполняется. Обработка следующей инструкции начинается после выполнения предыдущей. Исполняемый код, как правило, не создаётся.
- Компилятор обрабатывает программу целиком, и исполняемый код содержит результат обработки всей программы.

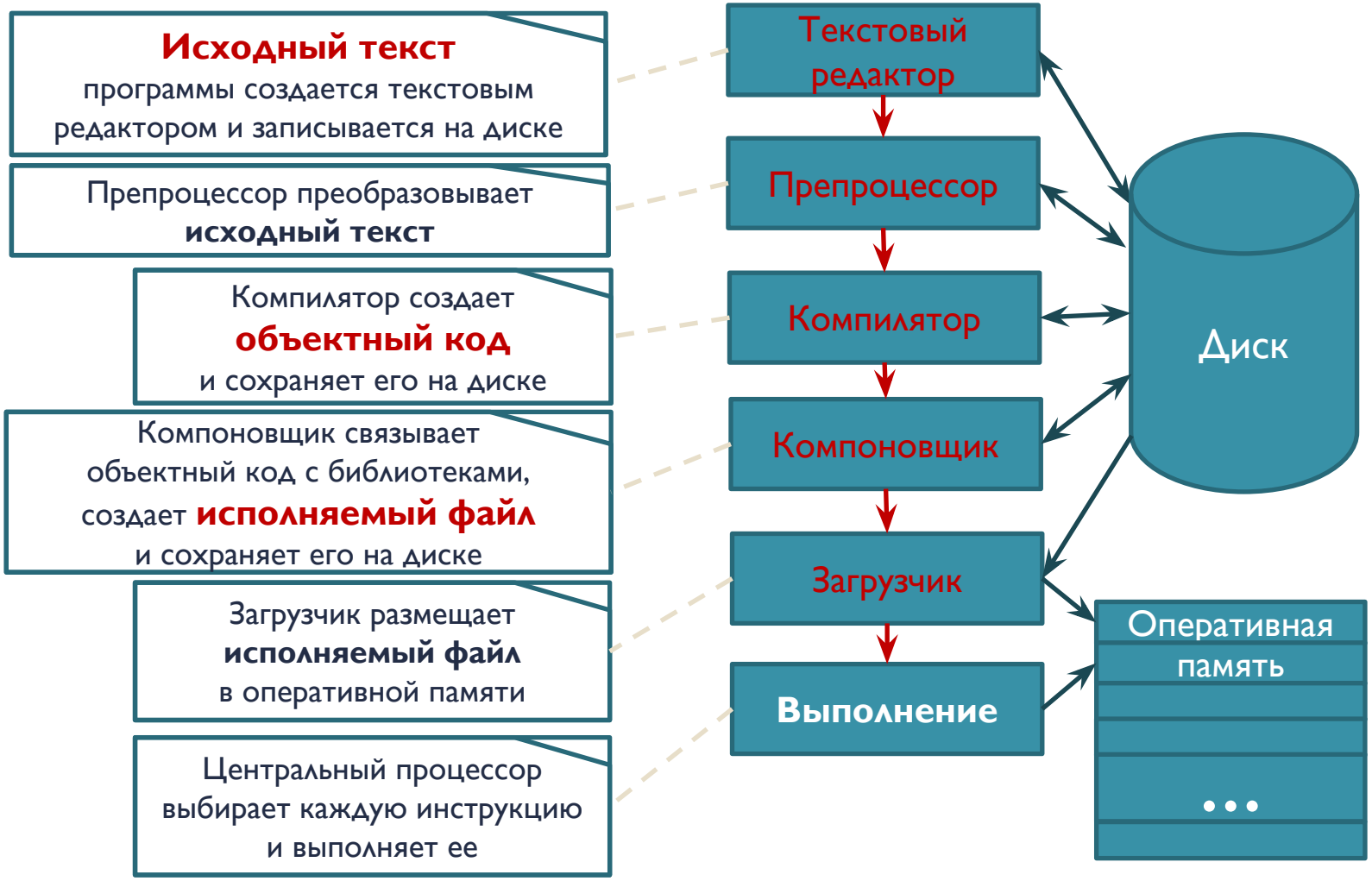
Преимущества и недостатки каждого вида трансляторов

- Интерпретаторы более просты в работе и требуют меньше ресурсов;
- Интерпретация выполняется быстрее, чем компиляция;
- При необходимости повторного выполнения программы должна выполняться её повторная трансляция;
- При обнаружении ошибки в программе оказывается, что часть работы уже проделана, а откат зачастую невозможен.

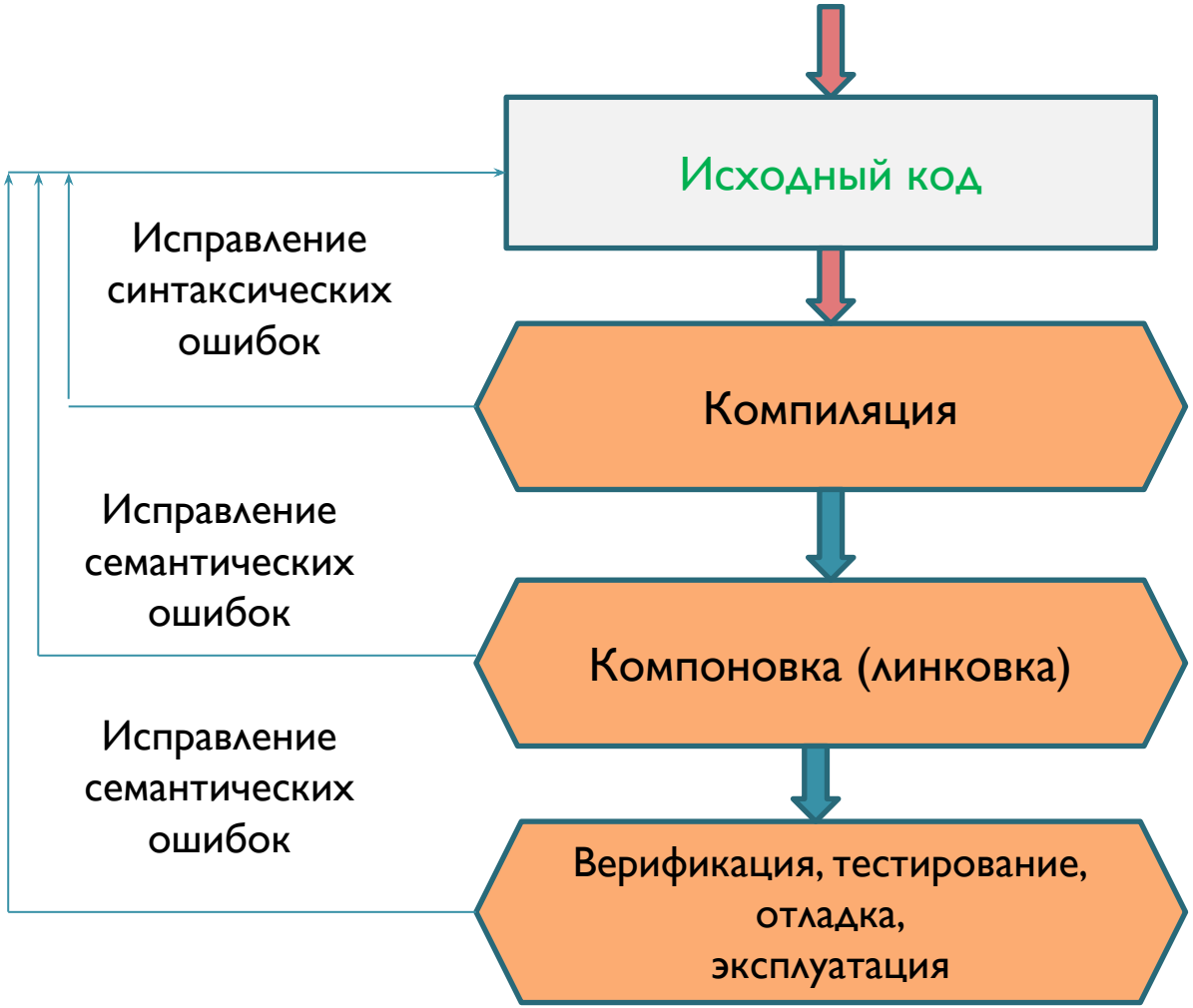
Классификация программных кодов

- Исходный текст (исходный код) – программа на языке высокого уровня или Ассемблере. Хранится, как правило, в виде текстового файла.
- Объектный код – результат компиляции исходного текста одного программного модуля. Объектный код представляет собой последовательность машинных команд и ссылок на другие объектные модули. Объектный код не может быть непосредственно выполнен.
- Исполняемый код получается из объектных кодов в результате разрешения ссылок (компоновки, линковки). Частный случай разрешения ссылок – подключение стандартных библиотек.

Схема создания исполняемого кода



Исправление ошибок в процессе реализации программы



Жизненный цикл программного обеспечения

Процесс создания и использования программного обеспечения, представленный в виде последовательности этапов и выполняемых на этих этапах процессов называется жизненным циклом программного обеспечения.

Основные этапы жизненного цикла ПО

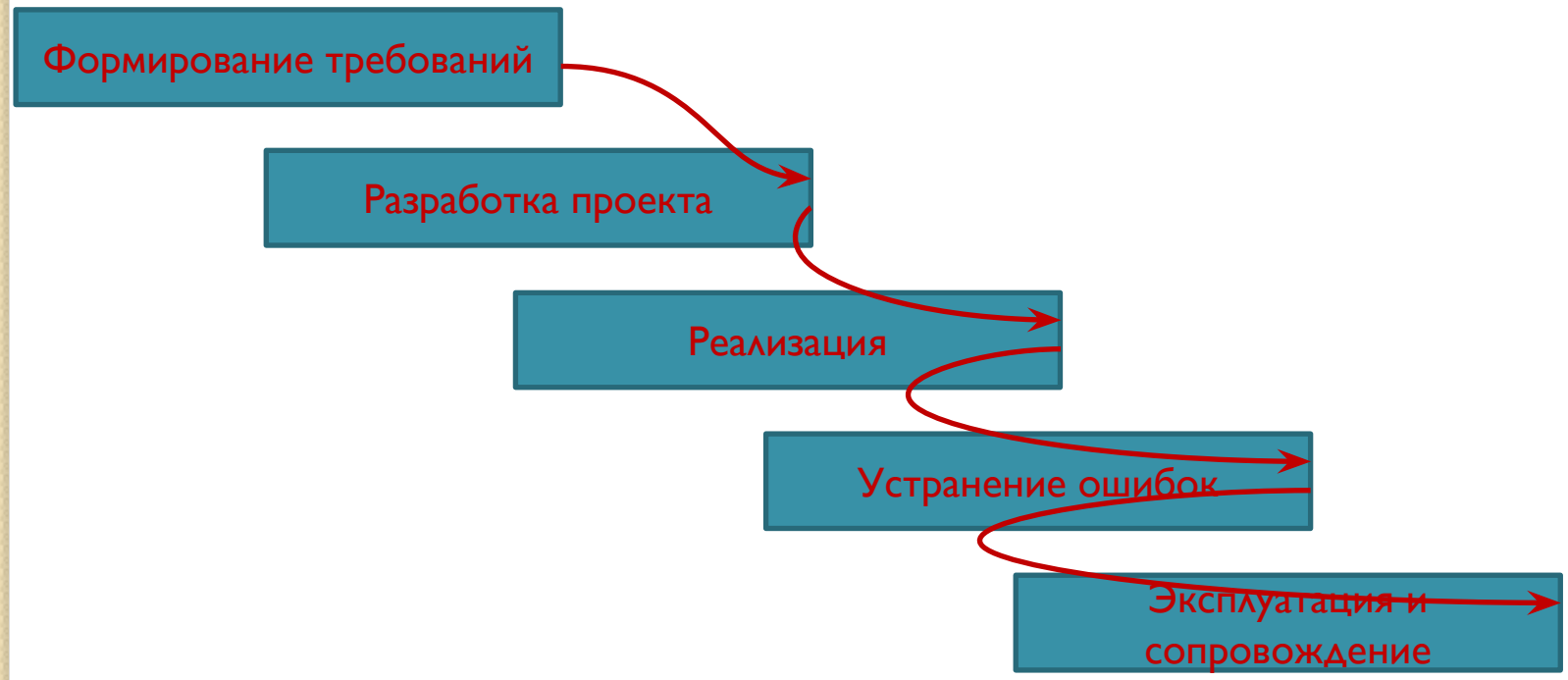
- **Формирование требований** – процесс сбора требований к системе, их систематизации, документирования, анализа, выявления противоречий и неполноты, разрешения конфликтов.
- **Разработка проекта -** деятельность по созданию проекта, то есть воспроизводимой модели программного обеспечения.
- **Реализация -** этап жизненного цикла программного обеспечения, объединяющий последовательные фазы создания программы в виде исходного кода, объектного кода и исполнимого кода. Результатом реализации является программа, которая может быть исполнена на компьютере.

Основные этапы жизненного цикла ПО (продолжение)

- **Устранение ошибок -** процесс устранения причин того, что программное обеспечение не работает, либо результат его работы не соответствует выработанным требованиям.
- **Эксплуатация** – деятельность по использованию программного обеспечения для решения практических задач.
- **Сопровождение** - модификация программного обеспечения с целью устранения ошибок, реализации потребностей заказчика в улучшениии тех или иных характеристик, а также его адаптации к использованию в модифицированном окружении.

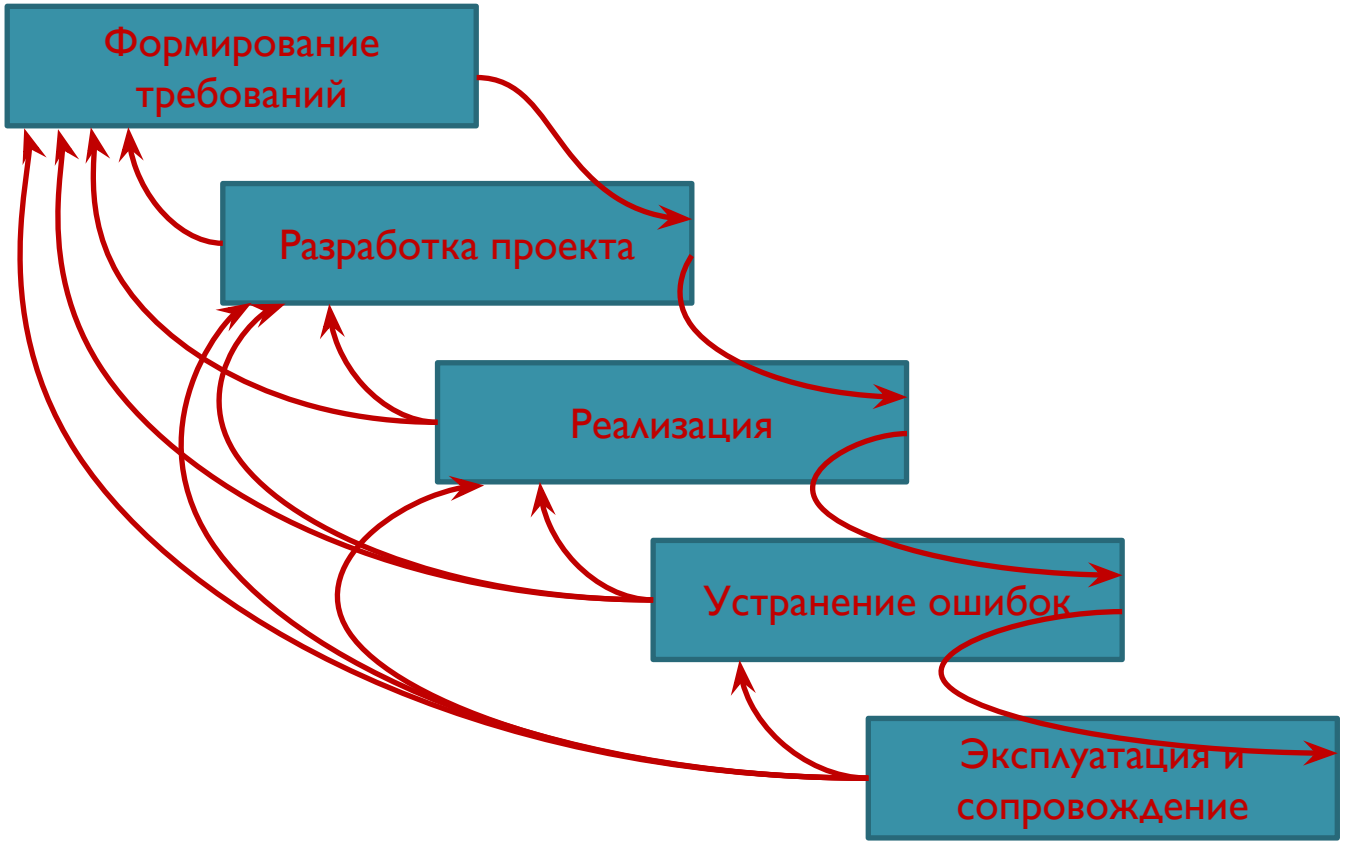
Каскадная модель жизненного цикла

- **Каскадная модель жизненного цикла** ("модель водопада") предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Эта модель была распространена в 70-80х годах XX века.



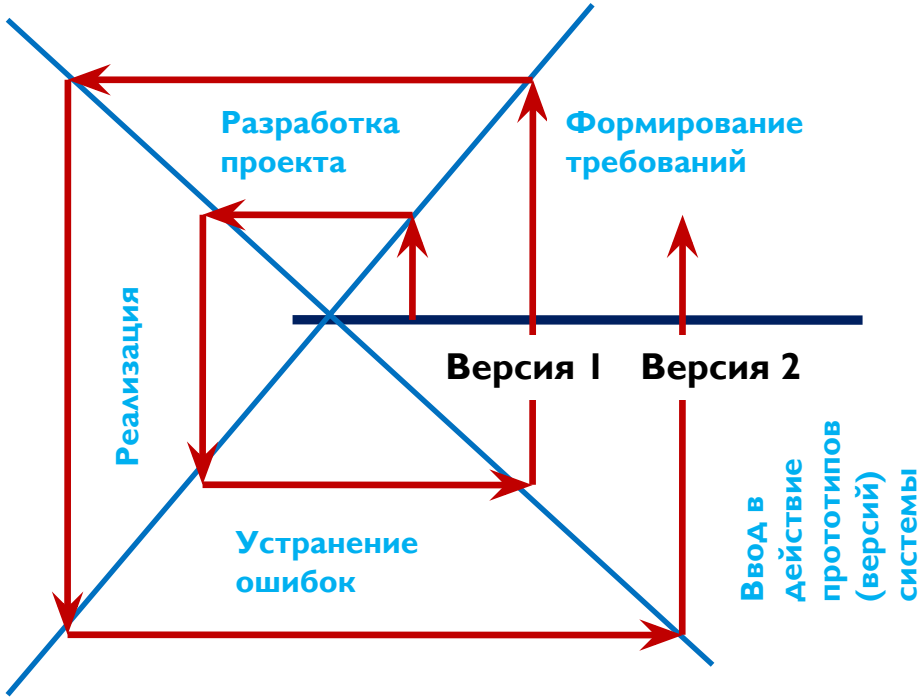
Каскадная модель жизненного цикла (продолжение)

На практике этапы каскадной модели реализуются итерационно, с циклами обратной связи между этапами.



Спиральная модель жизненного цикла ПО

Эта модель предусматривает спиралеобразное совершенствование системы путем последовательного создания прототипов (новых версий) этой системы. На каждом витке спирали при создании очередной версии продукта, уточняются требования проекта и планируются работы этого витка.



Определение алгоритма

Алгоритм – строгая и четкая конечная система правил, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к достижению поставленной цели.

Свойства алгоритма

- **понятность** (доступность) - все действия, описанные в алгоритме должны быть понятны исполнителю, то есть должны принадлежать системе действий данного исполнителя;
- **определенность** (детерминированность) – каждое действие должно быть четко и однозначно определено". Точное предписание", то есть, предписание, задающее алгоритм, должно выполняться однозначно и последовательно для получения конкретного и однозначного результата;
- **конечность** – выполнение алгоритма должно завершиться за конечное число шагов;

Свойства алгоритма (продолжение)

- **результативность** (сходимость) – достижение после конечного числа шагов искомого результата;
- **дискретность** (дискретная структура) – исполнение алгоритма должно состоять из отдельных шагов; Алгоритм представляет собой упорядоченное конечное множество шагов для получения результата. А всякое множество обладает свойством дискретности, то есть в любом алгоритме для каждого шага (кроме последнего), можно указать следующий за ним шаг;

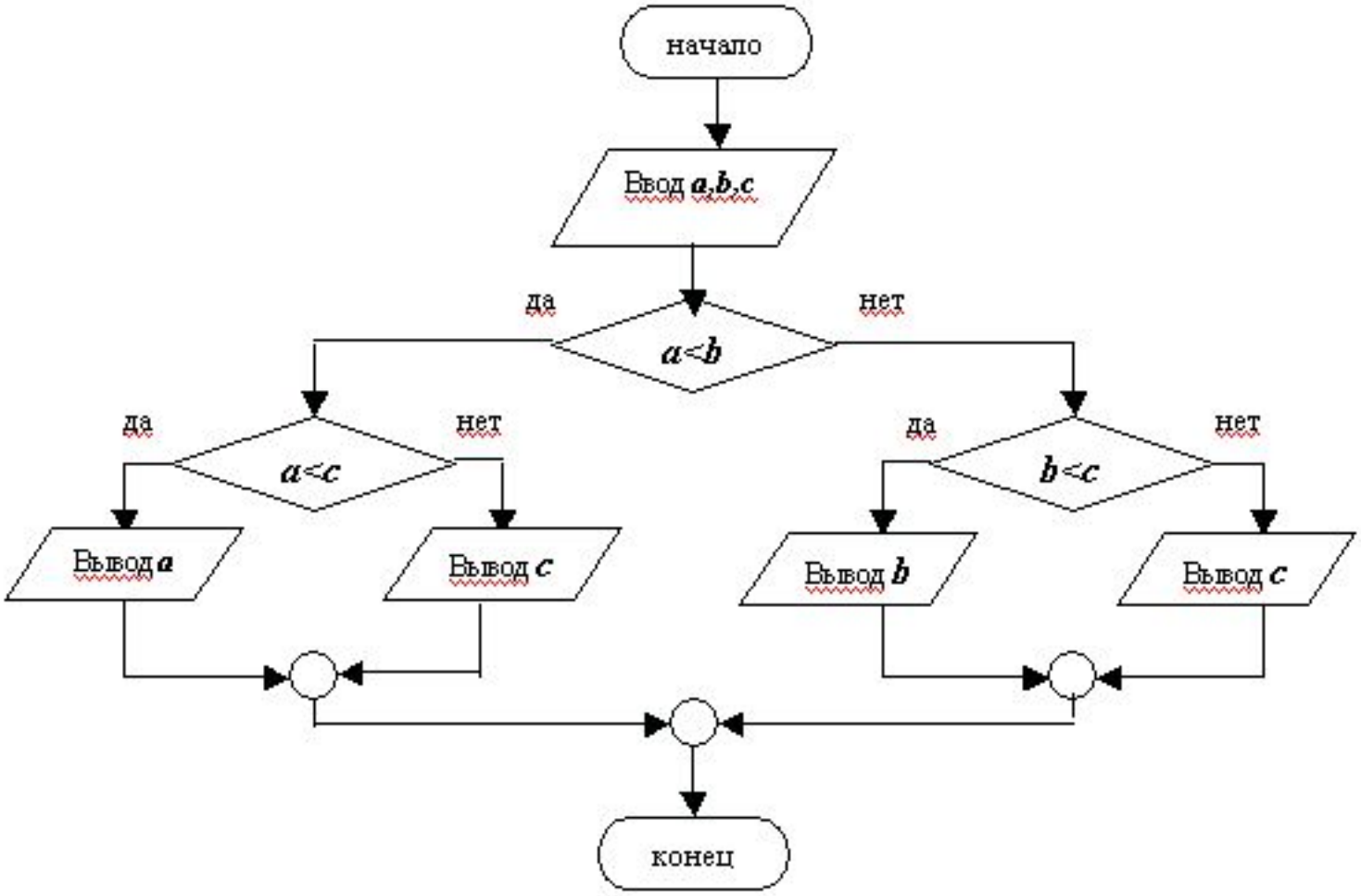
Свойства алгоритма (окончание)

- **конструктивность объектов -** исходные объекты, промежуточные и конечные результаты - это конструктивные объекты, которые могут быть построены целиком или допускают кодирование в каких-то алфавитах.

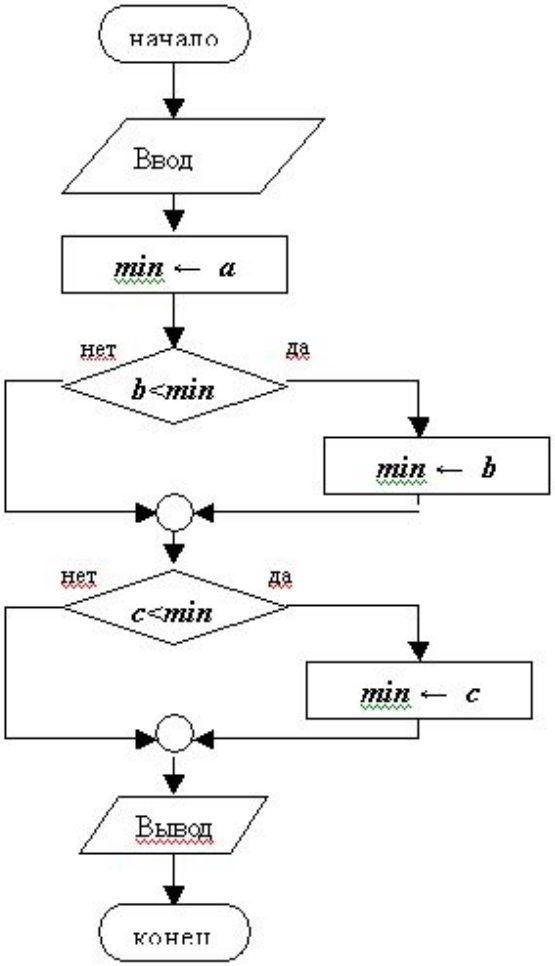
Способы записи алгоритма

- словесное описание на естественном языке;
- математическая запись;
- графическая запись в виде блок-схем, структурограмм и графов;
- запись на искусственном алгоритмическом языке (псевдокоде);
- запись на одном из языков программирования.

Нахождение минимума из 3 чисел (вариант 1)



Нахождение минимума из 3 чисел (вариант 2)



Парадигмы программирования

Парадигма программирования — это совокупность идей и понятий, определяющая стиль написания программ.

Основные парадигмы программирования:

- императивное, декларативное и функциональное программирование;
- структурное программирование:
- объектно-ориентированное программирование
- процедурное и событийно-ориентированное программирование

Структурное программирование

Структурное программирование – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.

1. Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

- **последовательное исполнение** — однократное выполнение операций в том порядке, в котором они записаны в тексте программы;
- **ветвление** — однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;
- **цикл** — многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

Структурное программирование (продолжение)

2. Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде т. н. подпрограмм (процедур или функций). В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция **вызова подпрограммы**. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.
3. Разработка программы ведётся пошагово, методом «сверху вниз».

Достоинства структурного программирования

- Структурное программирование позволяет значительно сократить число вариантов построения программы по одной и той же спецификации, что значительно снижает сложность программы и, что ещё важнее, облегчает понимание её другими разработчиками.
- В структурированных программах логически связанные операторы находятся визуально ближе, а слабо связанные — дальше, что позволяет обходиться без блок-схем и других графических форм изображения алгоритмов (по сути, сама программа является собственной блок-схемой).
- Сильно упрощается процесс тестирования и отладки структурированных программ.

Способы описания языков программирования

Наиболее часто используются следующие способы описания языков программирования (метаязыки):

- **нормальная форма Бэкуса-Наура (БНФ);**
- **синтаксические диаграммы Вирта;**
- **формальные грамматики.**

Форма Бэкуса-Наура

При записи в форме Бэкуса-Наура используются несколько типов объектов:

1. основные символы (или терминальные символы, в частности, ключевые слова) – будут записаны чёрным цветом;
2. металингвистические переменные (или нетерминальные символы), которые соответствуют каким-то понятиям описываемого языка. Металингвистические переменные изображаются русскими словами, записанными синим цветом, и заключаются в угловые скобки <>;

Форма Бэкуса-Наура (продолжение)

3. металингвистические связки, которые также будут записываться синим цветом:
   - ::= определение метапеременной (возможно, рекурсивное)
   - [] необязательная конструкция
   - []… конструкция повторяется 0 и более раз
   - {}… конструкция повторяется 1 и более раз
   - | разделитель для вариантов

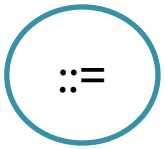
Пример использования БНФ

- <целое> ::= <целое без знака> | + <целое без знака> | - <целое без знака>
- <целое без знака> ::= <цифра> | <целое без знака> <цифра>
- <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <комментарий C++> ::= /* <текст> */ | // <текст до конца строки>

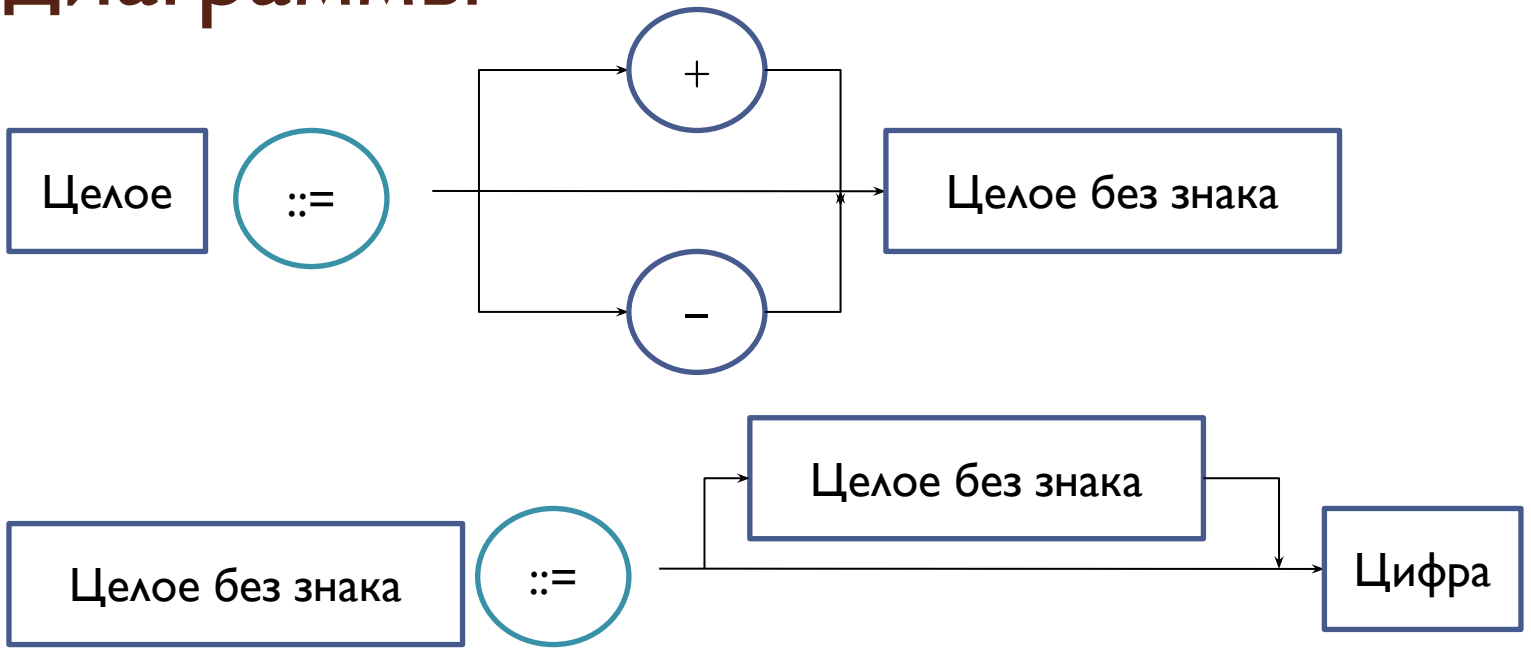
Угловые скобки в описаниях метапеременных часто убираются

Синтаксические диаграммы Вирта

Синтаксическая диаграмма – это графическое правило определения конструкции языка с помощью специальных обозначений (элементов).



Пример синтаксической диаграммы



Лексемы

Лексемы языка программирования аналогичны словам естественного языка.

- имена (идентификаторы);
- ключевые слова;
- знаки операций;
- разделители;
- литералы (константы).

Из лексем составляются выражения и операторы.

- Выражение задает правило вычисления некоторого значения.
- Оператор задает законченное описание некоторого действия.