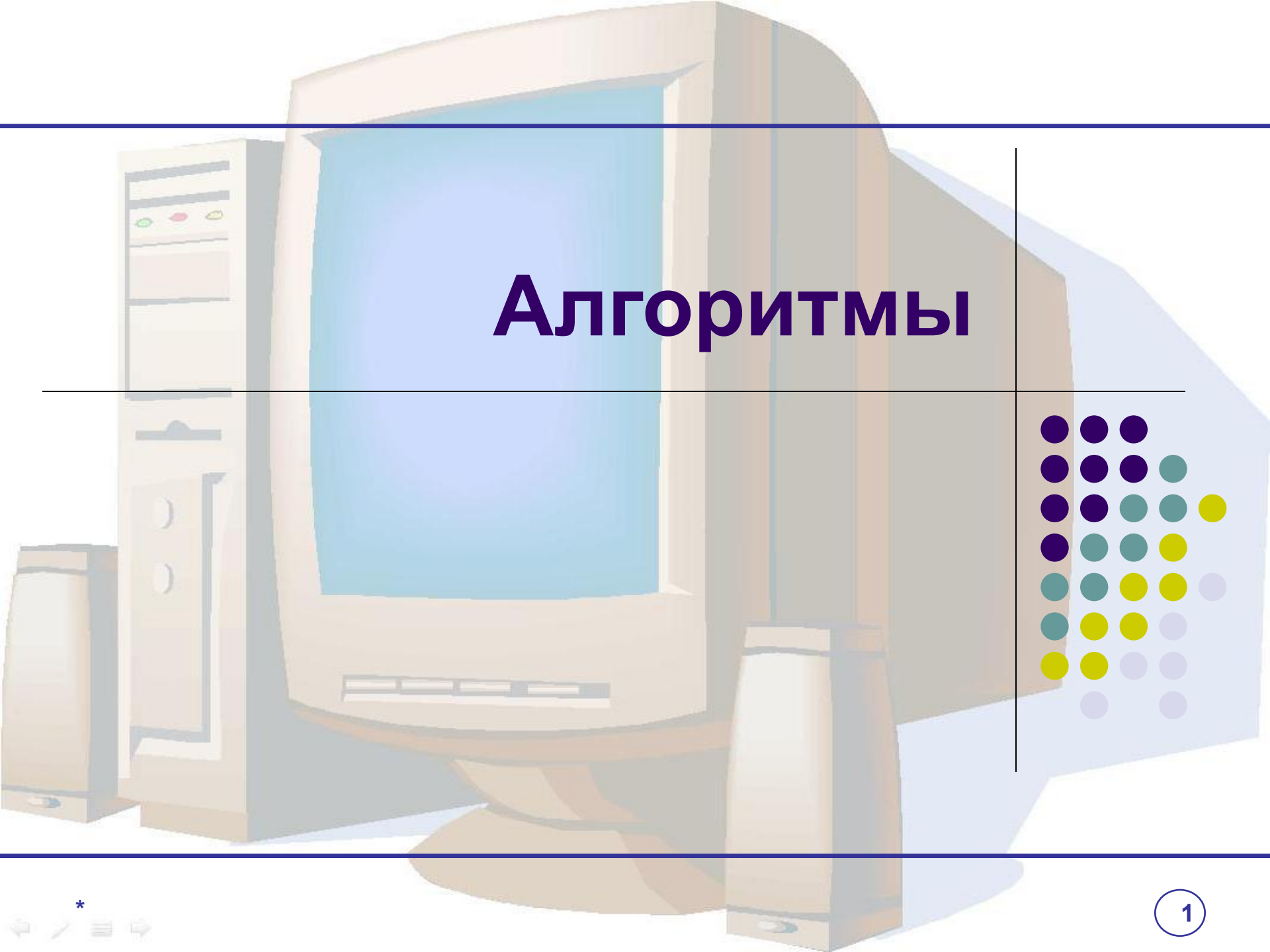
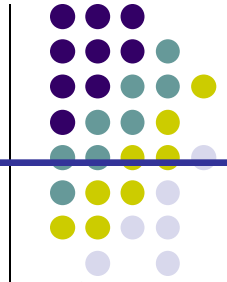
Алгоритмы



Этапы решения задачи



- Работа по решению любой задачи с использованием компьютера делится на следующие этапы:

1. **Постановка задачи**
2. **Формализация задачи**
3. **Построение алгоритма**
4. **Составление программы на языке программирования**
5. **Отладка и тестирование программы**
6. **Проведение расчетов и анализ полученных результатов**

- Часто эту последовательность называют технологической цепочкой решения задачи на ЭВМ.
- На этапе **постановки** задачи должно быть четко сформулировано, что дано и что требуется найти. Здесь очень важно определить полный набор исходных данных, необходимых для получения решения.

Этапы решения задачи

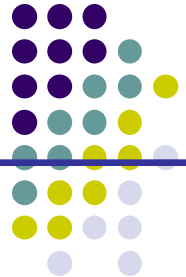
- Второй этап — **формализация** задачи. Здесь чаще всего задача переводится на язык математических формул, уравнений, отношений.
- Третий этап — **построение алгоритма**.
- Возникает впечатление, что опытные программисты часто «пропускают» данный этап, однако это не так, просто третий этап производится исключительно в их сознании.
- Однако в учебных целях бывает полезно использовать дополнительные средства алгоритмизации (блок-схемы, псевдокоды), а затем переводить полученный алгоритм на язык программирования.
- Четвертый этап – **написание программы** на выбранном алгоритмическом языке программирования.
- На пятом и шестом этапах производится **отладка и верификация** работы программы и, в итоге, **получение** требуемых результатов.

*

Понятие алгоритма

- Слово **«алгоритм»** происходит от Algorithmi — латинского написания имени Мухаммеда аль-Хорезми (787 — 850), математика средневекового Востока, разработавшего десятичную позиционную систему счисления и правилах арифметики многозначных чисел. Именно эти правила в то время называли алгоритмами.
- Сейчас понятие «алгоритм стало более общим»:
- **Алгоритм – это свод конечного числа правил, задающих последовательность выполнения операций при решении задачи.**
- Совокупность объектов с которыми работает алгоритм называется **данными**.

Особенности алгоритма



Алгоритм имеет 5 отличительных особенностей:

- *Конечность (финитивность)* - любой алгоритм должен приводить к цели за конечное число шагов (e.g. исключаются бесконечные циклы);
- *Определенность* – каждый шаг алгоритма должен быть точно и недвусмысленно определен;
- Алгоритм имеет некоторое количество *входных* данных;
- Алгоритм обязательно имеет от 1 до нескольких *выходных* данных;
- *Эффективность* – все входящие в алгоритм операции можно выполнить точно и за конечный отрезок времени;

Особенности алгоритма

Для решения практически важных задач требуются не просто алгоритмы, а хорошие алгоритмы =>

Критерии качества алгоритма:

- время выполнения – число вызовов каждого шага алгоритма;
- переносимость – адаптируемость алгоритма к различным вычислительным системам;
- простота и т.д.

Виды данных

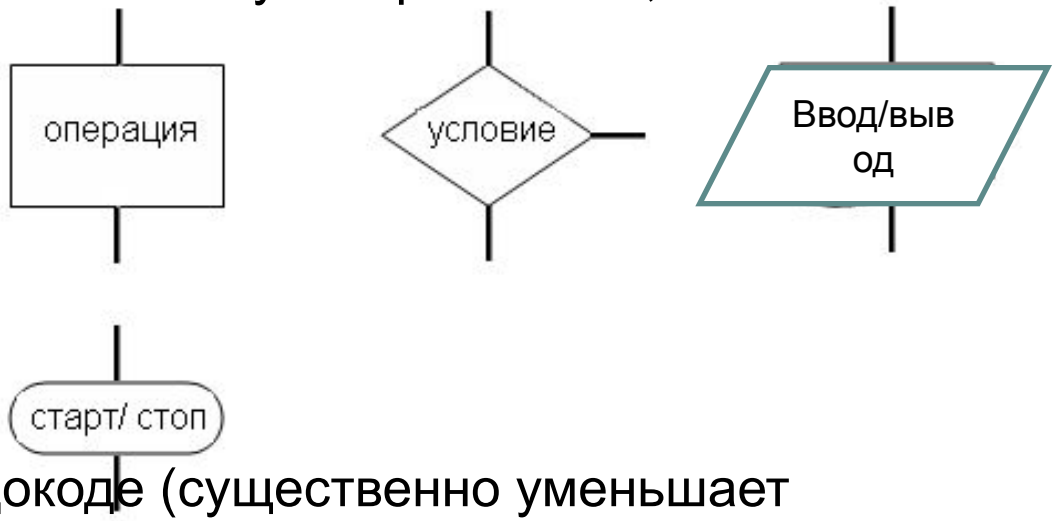
- По отношению к программе, данные делятся на **исходные**, **результаты** (окончательные данные) и **промежуточные**, которые получаются в процессе вычислений.
- Так, например, при решении квадратного уравнения $\mathbf{ax^2+bx+c=0}$ исходными данными являются коэффициенты a, b и c, результатами - корни x_1 и x_2, а к промежуточным данным можно отнести дискриминант $D=b^2-4ac$.

Исполнение алгоритма

- **ЭВМ – это устройство для исполнения алгоритмов.**
- Точнее говоря, исполнителем является комплекс ЭВМ + Система программирования (СП). Программист составляет программу на том языке, на который ориентирована СП.
- Иногда в литературе такой комплекс называют виртуальной ЭВМ. Например, компьютер с работающей системой программирования на Бэйсике называют Бэйсик-машиной; компьютер с работающей системой программирования на Паскале называют Паскаль-машиной и т. п.

Способы представления алгоритмов

- Словесный (вербальный) – является универсальным, но громоздок и неконкретен
- С помощью блок-схем:



- Запись алгоритма на псевдокоде (существенно уменьшает объем записи). К числу недостатков метода следует отнести отсутствие стандартных команд псевдокода (в отличие от метода блок-схем, где действуют нормы ГОСТа).
- Программирование алгоритмов при помощи алгоритмических языков (обладает достоинствами блок-схем и псевдокодов, по большей части лишен их недостатков)

*

Составные части алгоритма

- Независимо от того, на каком языке программирования будет написана программа, алгоритм решения любой задачи на ЭВМ может быть составлен из команд:

1. **присваивания;**
2. **ввода;**
3. **вывода;**
4. **обращения к вспомогательному алгоритму;**
5. **цикла;**
6. **ветвления.**

- Далее рассмотрим основные алгоритмические конструкции, возникающие при создании компьютерных программ.

*

Линейные алгоритмы

- **Линейный алгоритм** - алгоритм, в котором команды исполняются последовательно, одна за другой.
- Линейные алгоритмы состоят из операций ввода, вывода и присвоения.
- Пример. Правила деления обыкновенных дробей описаны так:

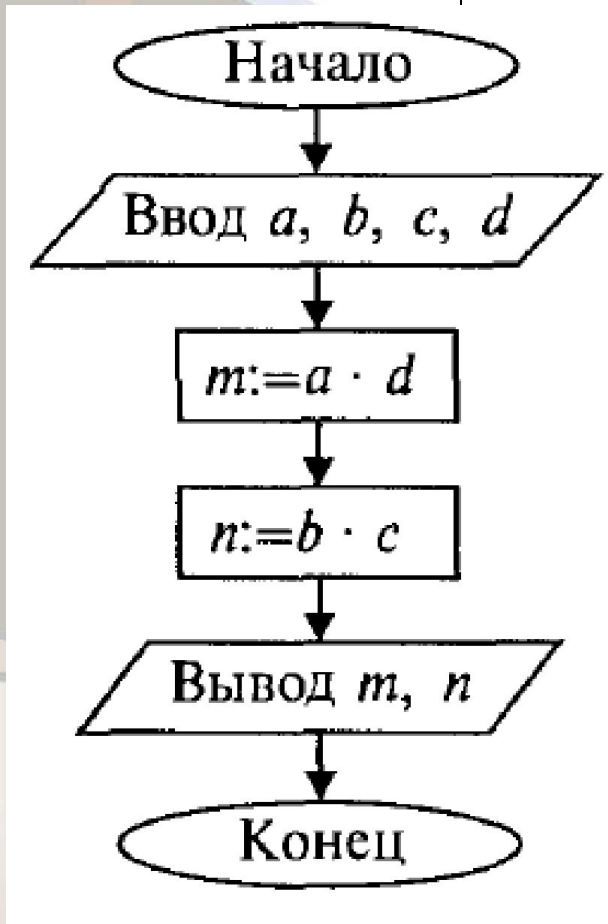
1. Числитель первой дроби умножить на знаменатель второй дроби.
2. Знаменатель первой дроби умножить на числитель второй дроби.
3. Записать дробь, числитель которой есть результат выполне ния пункта 1, а знаменатель — результат выполнения пункта 2.

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c} = \frac{m}{n}$$

*

Линейные алгоритмы

- Для описания алгоритмов чаще всего используют язык блок-схем.
- Эллипсами обозначают начало и конец программы.
- Параллелограммы – операции ввода и вывода данных.
- Прямоугольники – операции присваивания.
- Ромбы – ветвления алгоритма.
- На рисунке показана блок-схема алгоритма деления дробей.
- Вначале задаются входные параметры: a, b, c, и d.
- Последовательно вычисляются числитель и знаменатель искомой дроби.
- Выводятся числитель m и знаменатель n искомой дроби.



Разветвляющиеся алгоритмы

- **Разветвляющийся** алгоритм – алгоритм, в котором после проверки условия в разных ситуациях исполняются разные наборы команд.
- Пример, алгоритм решения квадратного уравнения

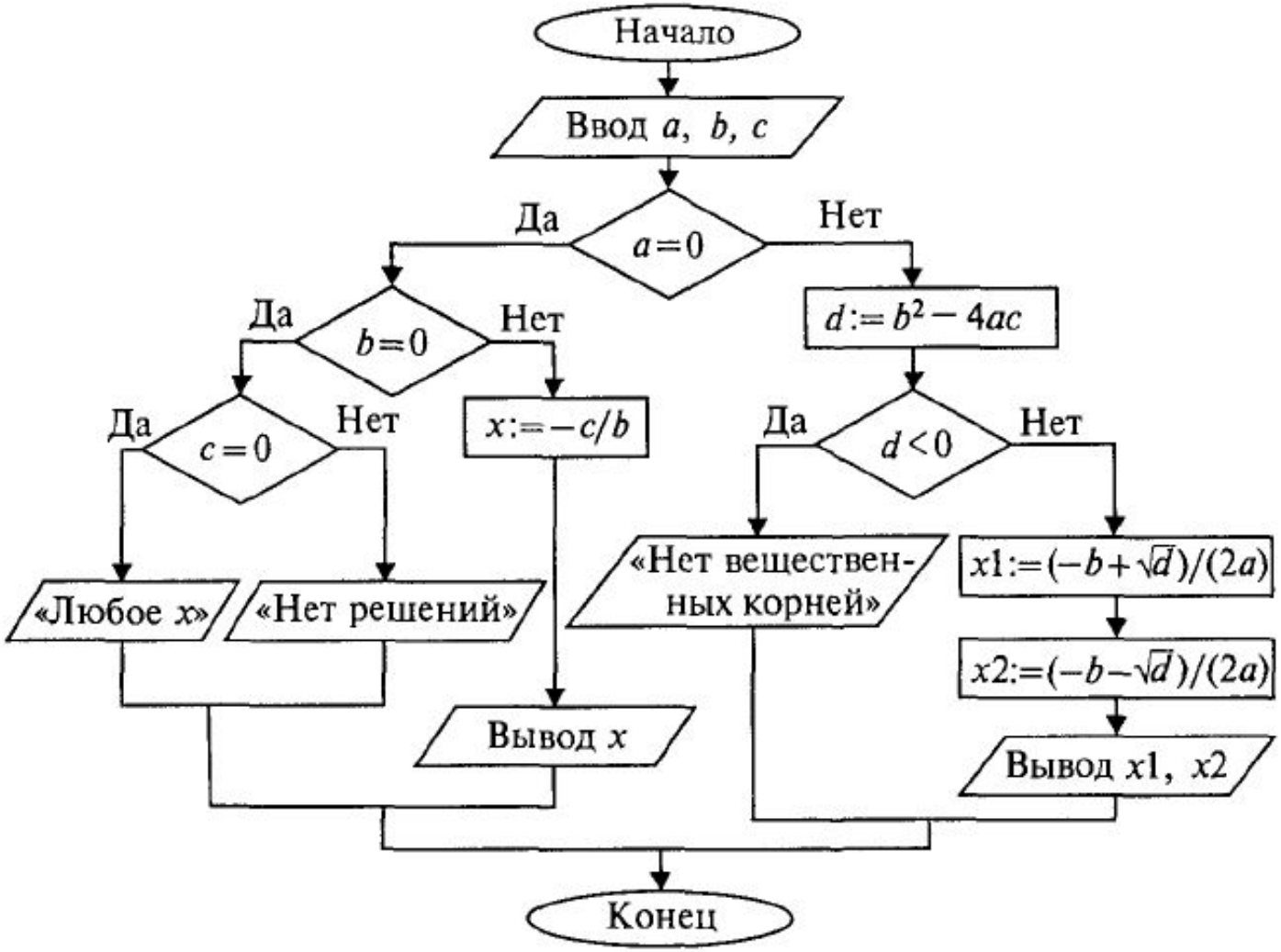
$$ax^2+bx+c=0$$

- Исходными данными являются коэффициенты a, b и c, решением - корни x_1 и x_2, вычисляемые по формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

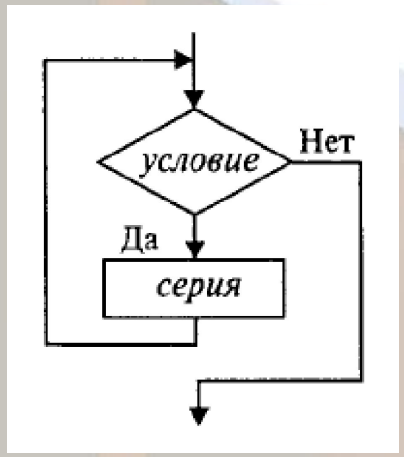
- Построим блок-схему данного алгоритма с учетом всех частных случаев:

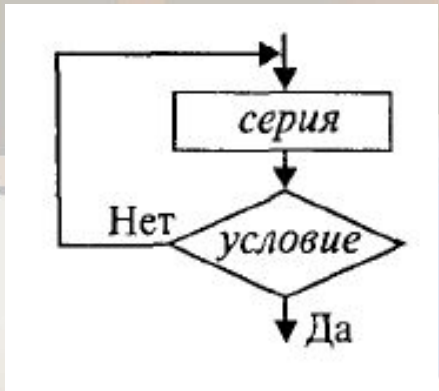
Разветвляющиеся алгоритмы



Циклические алгоритмы

- **Циклический алгоритм** - алгоритм, в котором содержится команда повторения, т.е. цикл.
- Для окончания циклического вычисления необходима проверка некоторого логического условия.
- Логическое условие может находится перед или после тела цикла. Соответственно такие циклы носят названия **цикла с предусловием** и **цикла с постусловием**



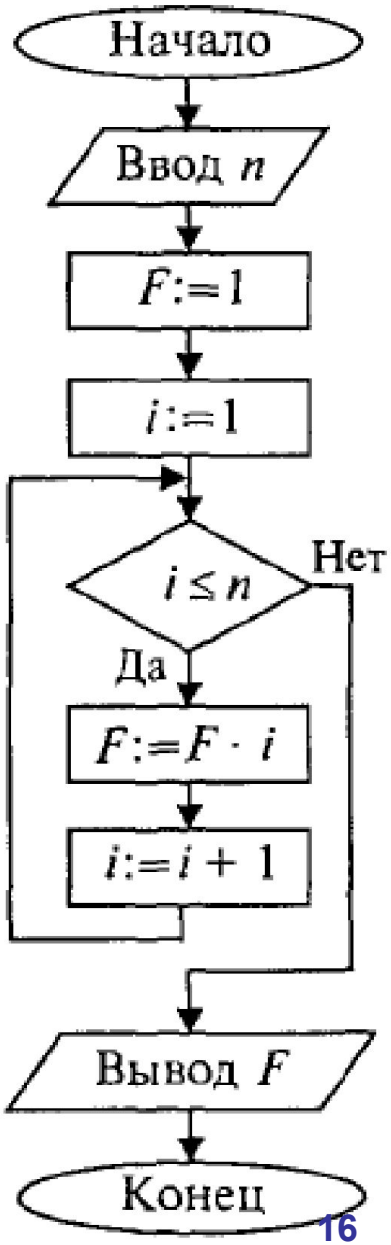


Циклические алгоритмы

- В качестве примера рассмотрим алгоритм нахождения факториала натурального числа:

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ 1 \cdot 2 \ldots n, & \text{если } n \geq 1. \end{cases}$$

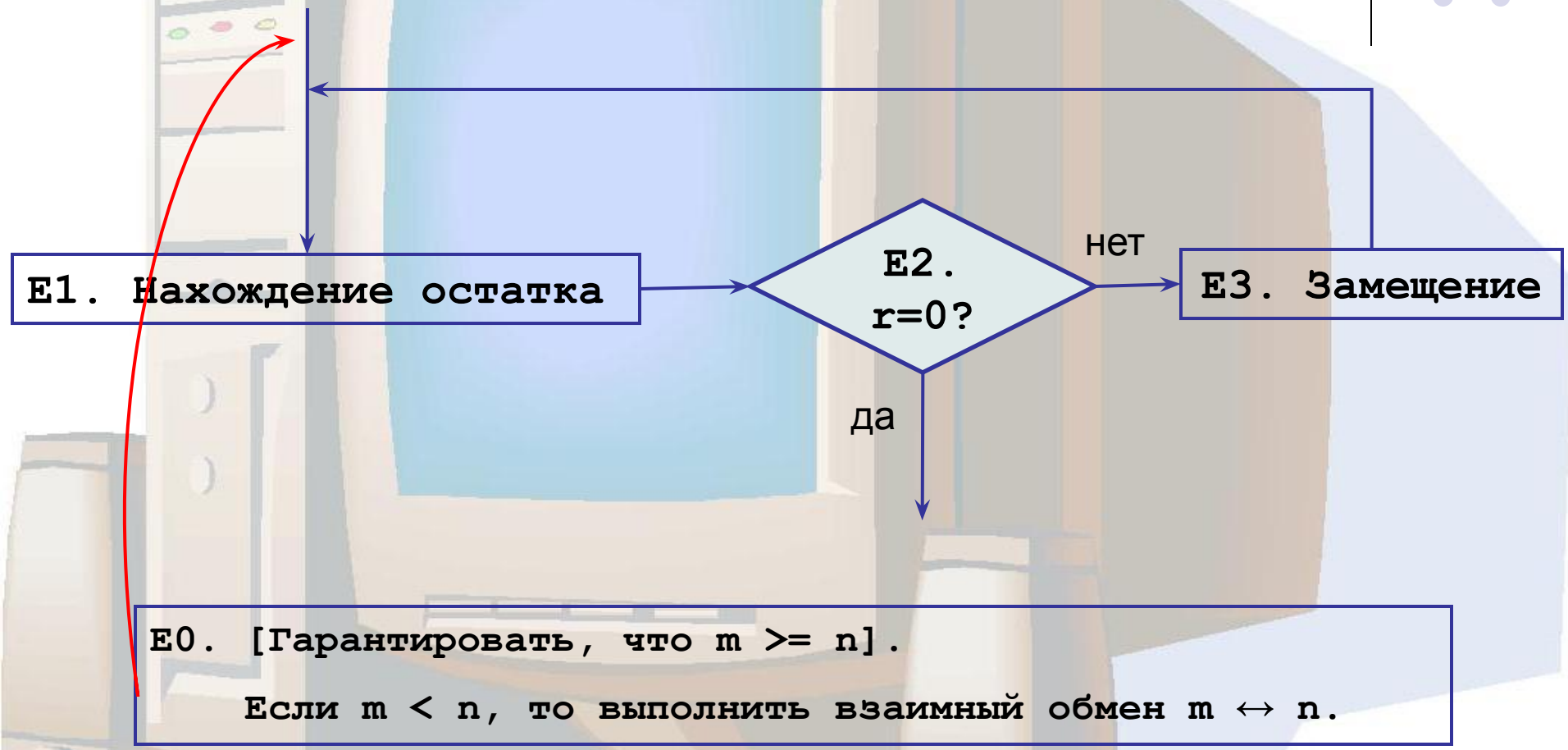
- Как правило в создаваемых программах присутствуют алгоритмические участки всех видов: линейные, ветвящиеся, циклические.
- Отдельные часто употребляемые алгоритмы можно выносить за пределы основной программы, такие алгоритмы называются **вспомогательными**.
- Согласно теоремам структурного программирования **любой алгоритм может быть представлен при помощи операций присваивания, ветвления и цикла**.



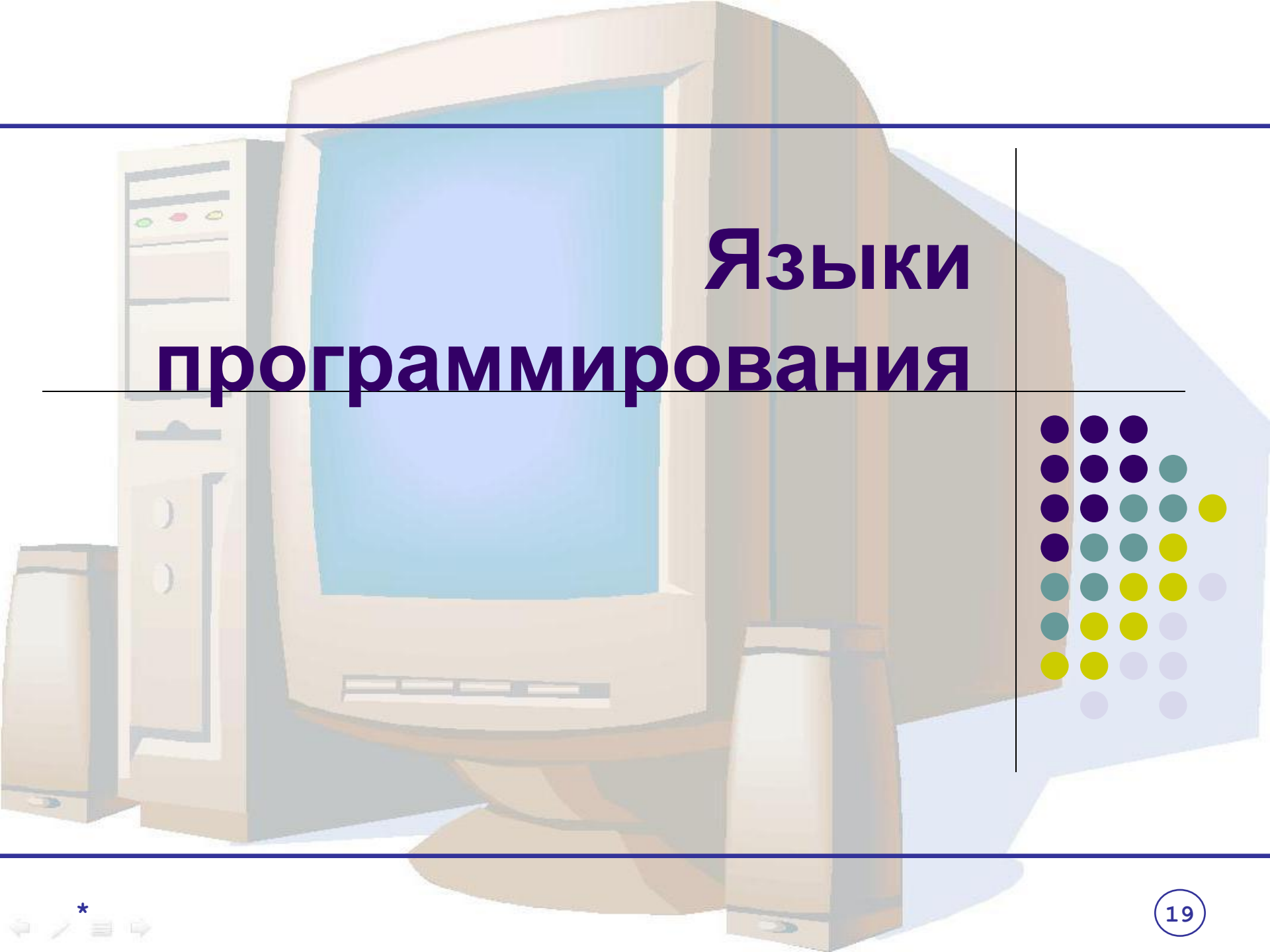
Алгоритм Евклида

- **Задача**: даны два целых положительных числа ***m*** и ***n***. Требуется найти их наибольший общий делитель, т.е. наибольшее целое положительное число, которое нацело делит оба числа ***m*** и ***n***.
- **E1.** [Нахождение остатка] Разделим m на n, и пусть остаток от деления будет равен r (где 0<=r<n).
- **E2.** [Сравнение с нулем] Если r=0, то выполнение алгоритма прекращается; n – искомое значение.
- **E3.** [Замещение] Присвоить m:=n, n:=r и вернуться к шагу **E1**.

Алгоритм Евклида



Языки программирования



Язык программирования

- *Языки программирования – искусственные языки, отличающиеся ограниченным числом «слов», значение которых понятно транслятору/ компилятору и строгими правилами записи команд (операторов).*
- Совокупность подобных требований образует *синтаксис* языка, а смысл каждой конкретной команды и других конструкций языка – его *семантику*.

Цели языка программирования:

- Язык программирования как инструмент проектирования.
- Язык программирования как средство человеческого восприятия.
- Язык программирования как средство управления компьютером.

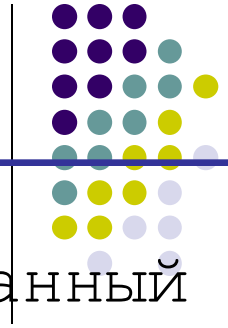
*

Алгоритмический язык

- **Алгоритмический язык программирования** – это способ записи алгоритмов решения различных задач на ЭВМ в понятной для системы программирования (виртуальной ЭВМ) форме.
- Алгоритм, записанный при помощи языка программирования будем называть **текстом компьютерной программы (исходным кодом).**
- Основная цель системы программирования – анализ и перевод текста программы с языка программирования на машинный язык.

*

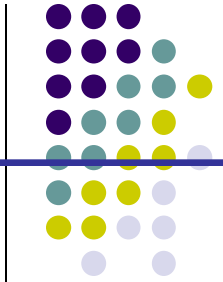
Алгоритмический язык



В зависимости от того, какой из целей данный язык программирования отвечает больше, выделяют т.н. языки ***низкого уровня*** и языки ***высокого уровня***:

- языки ***низкого уровня*** (императивные языки) – ориентированы на конкретный тип процессора, их операторы близки к машинному коду;
- языки ***высокого уровня*** (декларативные языки) – языки, описывающие ключевые абстракции предметной области.

Алгоритмический язык



Гради Буч:

- «Возникла тенденция перехода от языков, указывающих компилятору, что делать (императивные языки) к языкам, описывающим ключевые абстракции предметной области (декларативные языки)».

*

Машинно-зависимые языки

- Для программирования на самых первых компьютерах (ламповых) использовался **язык машинных команд (машинные коды).**
- Программирование в машинных кодах было процессом очень длительным и трудоемким. Программист должен знать числовые коды всех машинных команд, должен сам распределять память под команды программы и данные.
- Развитие компьютерной техники и усложнение программного обеспечения привело к необходимости создания первых языков программирования, в которых переменные величины стали изображаться символическими именами, а числовые коды операций заменились на словесные обозначения, которые легче запомнить.
- Такие языки получили название **языков низкого уровня (автокодов, ассемблеров)**, т.к. программирование на них сводилось, в конечном итоге, к более удобной записи машинных команд.
- Данные языки разрабатывались под конкретный тип (архитектуру) ЭВМ, поэтому носят название **машинно-зависимых**.

*

Машинно-зависимые языки

- Языки ассемблерного типа стали понятнее для человека, но менее понятны для компьютера.
- Чтобы компьютер мог исполнять программы на ассемблере, потребовался специальный переводчик - **транслятор**.
- **Транслятор - это системная программа, переводящая текст программы на искусственном языке в текст эквивалентной программы в машинных кодах.**
- Тем не менее ассемблеры все равно остаются неудобны для программной реализации любого, достаточно сложного алгоритма.
- В настоящее время ассемблерные фрагменты лишь изредка применяются для более эффективной реализации того или иного кусочка программы, но и эта область применения языков низкого уровня постепенно становится историей.
- Дальнейшее развитие вычислительной техники и усложнение программ привело к созданию **языков программирования высокого уровня.**

Языки высокого уровня

- Языки программирования высокого уровня (алгоритмические языки) являются **машинно-независимыми** языками. Одна и та же программа на таком языке может быть выполнена на ЭВМ разных типов, оснащенных соответствующим транслятором.
- Форма записи программ на языке высокого уровня по сравнению с ассемблерами еще ближе к традиционной математической форме, к естественному языку.
- При работе на языке высокого уровня программисту не надо задумываться об особенностях обработки информации на ЭВМ.
- Язык высокого уровня представляет собой особым образом кодифицированную запись алгоритма. Такой язык для компьютера еще менее понятен, чем ассемблер.
- За удаленность от машинных кодов приходится расплачиваться производительностью. Так ассемблеры на 10-15 %, а языки высокого уровня в 2-4 раза менее эффективны, чем машинные коды.

Виды трансляторов

- Реализовать тот или иной язык программирования на ЭВМ — значит создать транслятор с этого языка для данной ЭВМ.
- Существуют два принципиально различных метода трансляции. Они называются соответственно **интерпретация** и **компиляция**.
- Транслятор, работающий методом интерпретации, называется **интерпретатором**, по принципу компиляции - **компилятором**.
- **Интерпретатор** в течение всего времени работы программы находится во внутренней памяти. Интерпретатор в последовательности выполнения алгоритма «читает» очередной оператор программы, переводит его в команды и тут же выполняет эти команды. Затем переходит к переводу и выполнению следующего оператора. При этом результаты предыдущих переводов в памяти не сохраняются. При повторном выполнении одной и той же команды она снова будет транслироваться.

*

Виды трансляторов

- При **компиляции** в память ЭВМ загружается программа-компилятор. Она воспринимает текст программы как исходную информацию.
- После завершения компиляции получается программа на языке машинных команд. Затем в памяти остается только программа на языке машинных кодов, которая выполняется, и получаются требуемые результаты.
- При интерпретации, поскольку трансляция и выполнение совмещены, программа на ЭВМ проходит в один этап.
- При компиляции исполнение программы разбивается на два этапа: трансляцию и выполнение.
- Откомпилированная программа выполняется быстрее, чем интерпретируемая. Поэтому использование компиляторов удобнее для больших программ, требующих быстрого счета. Программы на **Паскале, Си, Фортране** всегда компилируются. **Бейсик,** а также современные математические прикладные пакеты: **Maple, Mathematica, MathCad** чаще всего реализованы через интерпретатор.

Поколения языков программирования

I поколение
/1954-1958 гг./ - языки ассемблера, простейшие языки высокого уровня

- принцип «одна строка – одна инструкция»;
- впервые появились простейшие операции (+,-, := и т.д.), математические формулы;
- большинство из языков первого поколения «мертвы», но некоторые используются в специфических программных системах и для специфических задач (например, язык **FORTRAN**);

- **П**: FORTRAN I, ALGOL – 58, Flowmatic, IPL V.

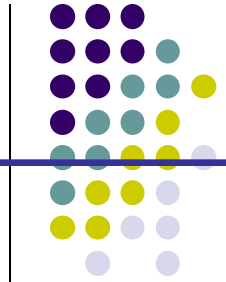
Поколения языков программирования

II поколение

/1959-1961 гг./ - разработаны под конкретные задачи

- **FORTRAN II** – появляются подпрограммы, раздельная компиляция;
- **Algol – 60** – типы данных, блочная структура, составной оператор;
- **COBOL** – язык работы с данными (операции с файлами и т.д.);
- **LISP** – возможность работы со списками, появление указателей.

Поколения языков программирования



III поколение

/1962-1970 гг./ – «смешение языков» – появление универсальных языков высокого уровня

- независимость от конкретного компьютера;
- возможность использования мощных синтаксических конструкций;
- понятная большинству пользователей структура;
- применимость к задачам любой сложности и любой предметной области;
- появление первой версии языка **Pascal** (на основе **ALGOL-68**), языка **C**;
- появление языка **Simula** – прародителя всех современных объектно-ориентированных языков (появляется понятие классов и объектов, абстрактных данных).

Поколения языков программирования

IV поколение

/1970-80-е гг./ – «потерянное поколение» – появляется огромное количество языков-потомков **Simula**, большинство из которых «мертвы»

- встраиваются мощные операторы, для реализации которых на языках младших поколений потребовались бы тысячи строк исходного кода;
- **Object Pascal, C++** – результат «смешения» соответствующих языков и **Simula**, в результате чего в эти языки вошло понятие объектного программирования.

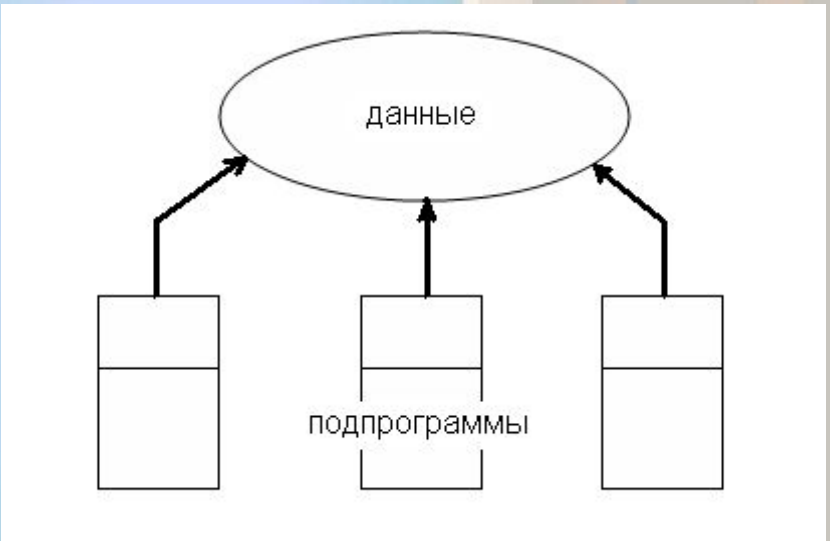
Поколения языков программирования

V поколение

/1990-е гг./ – системы автоматического создания прикладных программ

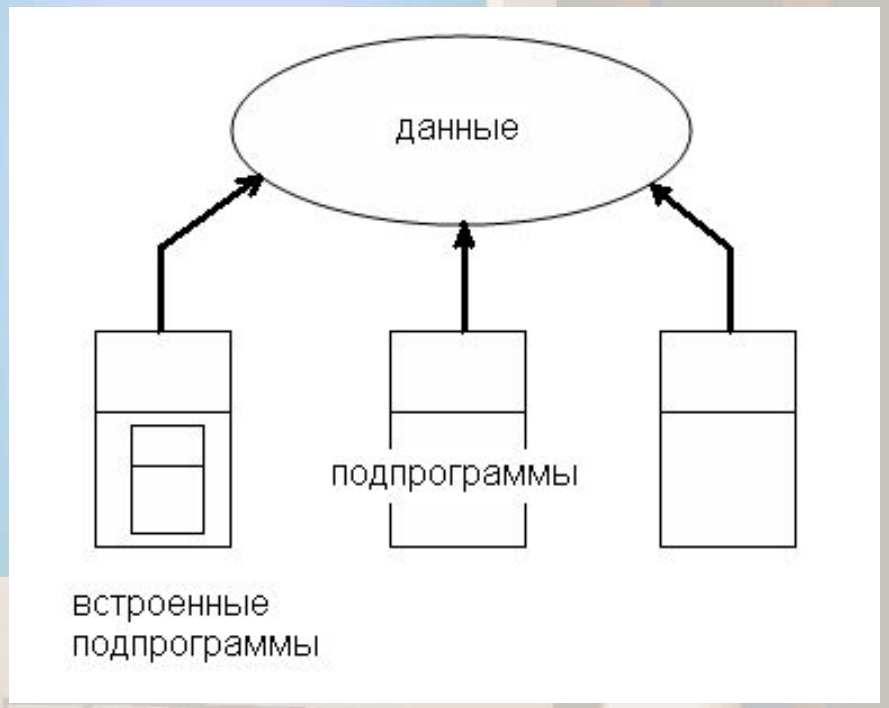
- автоматическое формирование результирующего текста на универсальных языках программирования;
- инструкции вводятся в максимально наглядном виде с помощью методов, наиболее удобных для человека, не знакомого с программированием.

I – начало II поколения



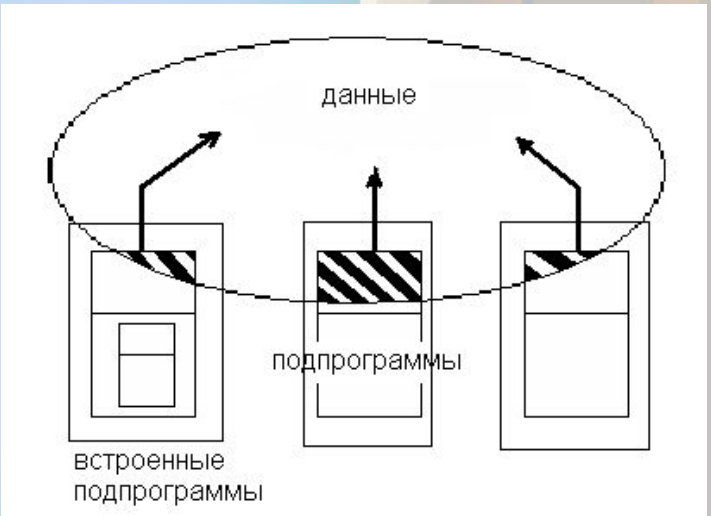
- наличие одной общей (глобальной) области данных, к которой обращаются подпрограммы;
- отсутствие контроля за типами данных (ошибка типов возникает лишь в момент выполнения).

II – начало III поколения



- вложенность подпрограмм;
- появляется механизм передачи аргументов.

III поколение

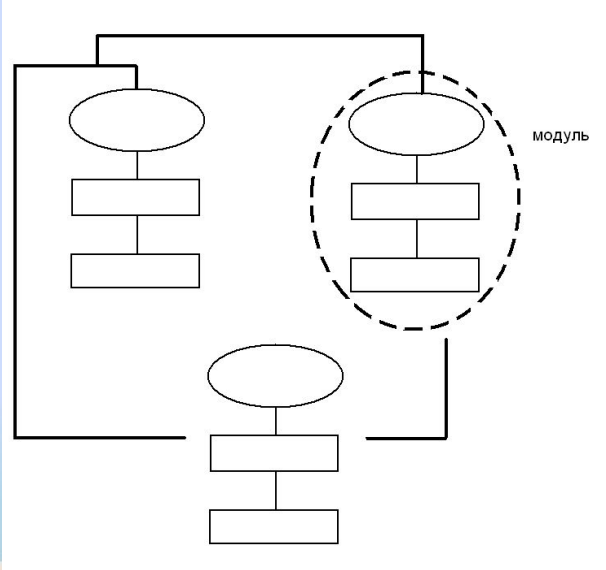


- появляются *модули* (некоторая группа подпрограмм);
- появление локальных данных модуля (недоступны другим модулям);
- обращение подпрограмм как к данным модуля, так и к области глобальных данных.

Топология языков программирования

/основные элементы программирования и их взаимодействие/

IV поколение /Объектно-ориентированные языки/



- появление модулей, состоящих из логически связанных классов и объектов, полей и методов;
- отсутствие глобальной области данных.

Структура алгоритмического языка

- Во всяком языке программирования определены способы организации данных и способы организации действий над данными.
- Кроме того, существует понятие «элементы языка», включающее в себя множество символов (алфавит), лексемы и другие изобразительные средства языка программирования.
- В изучении естественных языков и языков программирования есть сходные моменты. Во-первых, для того чтобы читать и писать на иностранном языке, нужно знать алфавит этого языка. Во-вторых, следует знать правописание слов и правила записи предложений, т.е. то, что называется синтаксисом языка. В-третьих, важно понимать смысл слов и фраз, чтобы адекватно реагировать на них: ведь из грамотно написанных слов можно составить абсолютно бессмысленную фразу. Смысловое содержание языковой конструкции называется семантикой.
- **Всякий язык программирования имеет три основные составляющие: алфавит, синтаксис и семантику.**

Структура алгоритмического языка

Фортран

- Первыми популярными алгоритмическими языками программирования были появившиеся в 1950-х гг. **Фортран, Кобол и Алгол.**
- **Фортран (Fortran)** — первый алгоритмический язык программирования, созданный ещё в 1954 году фирмой IBM. Его название является аббревиатурой от **FORmula TRANslator**, т.е. транслятор формул.
- Фортран широко использовался в основном для научных и инженерных вычислений. Он используется и по сей день в силу большого количества написанных на нём программ, изменять и, тем более, переписывать которые нет смысла.
- Большое количество языков программирования появилось в 1960 - 1970-х гг. За всю историю ЭВМ их было создано более тысячи. Но распространились и выдержали испытание временем немногие.

Бейсик

- В 1965 г. в Дартмутском университете был разработан язык **Бейсик (Basic)**.
- По замыслу авторов это простой язык, легко изучаемый, предназначенный для программирования несложных расчетных задач.
- Ближе всего Бейсик к Фортрану, поэтому он и многие похожие языки иногда называют «фортраноподобными».
- Наибольшее распространение Бейсик получил на микроЭВМ и персональных компьютерах.
- В 80-х годах этим языком заинтересовалась фирма Microsoft, выпустившая очень популярный транслятор **QBasic (Quick Basic),** а затем и революционную для своего времени визуальную среду **Visual Basic**, у которой до сих пор немало поклонников среди непрофессиональных программистов.

Паскаль

- В 1971 г. швейцарский математик Н.Вирт создал язык **Паскаль (Pascal),** как учебный язык структурного программирования.
- Наибольший успех в распространении этого языка обеспечили персональные компьютеры. Фирма **Borland International, Inc** разработала один из самых популярных трансляторов этого языка - **Турбо Паскаль (Turbo Pascal).**
- Турбо Паскаль и его более поздняя версия **Object Pascal**, ставшая основой визуальной среды программирования **Delphi** стали очень популярным языком для профессионального программирования прикладных задач, таких как научные расчеты и оболочки управления базами данных.
- Параллельно в последние годы активно развивается свободный кроссплатформенный транслятор **Free Pascal**.

Си и Си++

- Язык программирования **Си (C)** создавался как инструментальный язык для разработки операционных систем, трансляторов, баз данных и других системных и прикладных программ.
- Так же как и Паскаль, Си - это язык структурного программирования, но, в отличие от Паскаля, в нем заложены возможности непосредственного обращения к некоторым машинным командам, к определенным участкам памяти компьютера.
- Язык Си до сих пор является основополагающим языком unix-подобных операционных систем.
- На основе языка С был создан язык **Си++ (C++)** ставший основным языком профессионального системного программирования.
- Важным преимуществом С и С++ является их стандартизированность.

ЛИСП и Пролог

- ЭВМ будущего, пятого поколения называют машинами «искусственного интеллекта». Но прототипы языков для этих машин были созданы существенно раньше их физического появления. Это языки **ЛИСП и Пролог.**
- **ЛИСП** появился в 1965 г. Язык ЛИСП основан на понятии рекурсивно определенных функций. С его помощью на ЭВМ можно моделировать достаточно сложные процессы, в частности интеллектуальную деятельность людей. В настоящее время активно применяется в математических пакетах символьных вычислений.
- Язык **Пролог** разработан во Франции в 1972 г. также для решения проблемы «искусственного интеллекта». Пролог позволяет в формальном виде описывать различные утверждения, логику рассуждений и заставляет ЭВМ давать ответы на заданные вопросы.
- Интерес к этому языку то возрастает, то затихает, однако на его основе и сейчас создаются новые разработки.

*

Обзор языков программирования высокого уровня

- **COBOL** (Кобол). Это компилируемый язык для применения в экономической области и решения бизнес-задач, разработанный в начале 60-х годов. Он отличается большой «многословностью» — его операторы иногда выглядят как обычные английские фразы. В Коболе были реализованы очень мощные средства работы с большими объемами данных, хранящимися на различных внешних носителях. На этом языке создано очень много приложений, которые активно эксплуатируются и сегодня. Достаточно сказать, что наибольшую зарплату в США получают программисты на Коболе.

Обзор языков программирования высокого уровня

- **Algol** (Алгол). Компилируемый язык, созданный в 1960 году. Он был призван заменить Фортран, но из-за более сложной структуры не получил широкого распространения. В 1968 году была создана версия Алгол 68, по своим возможностям и сегодня опережающая многие языки программирования, однако из-за отсутствия достаточно эффективных компьютеров для нее не удалось своевременно создать хорошие компиляторы.

Обзор языков программирования высокого уровня

- **Java** (Джава, Ява). Этот язык был создан компанией Sun в начале 90-х годов на основе Си++. Он призван упростить разработку приложений на основе Си++ путем исключения из него всех низкоуровневых возможностей. Но главная особенность этого языка — компиляция не в машинный код, а в платформно-независимый байт-код (каждая команда занимает один байт). Этот байт-код может выполняться с помощью интерпретатора — виртуальной Java-машины JVM(Java Virtual Machine), версии которой созданы сегодня для любых платформ. Благодаря наличию множества Java-машин программы на Java можно переносить не только на уровне исходных текстов, но и на уровне двоичного байт-кода, поэтому по популярности язык Java сегодня занимает второе место в мире после Бейсика.
- Особое внимание в развитии этого языка уделяется двум направлениям: поддержке всевозможных мобильных устройств и микрокомпьютеров, встраиваемых в бытовую технику (технология Jini) и созданию платформно-независимых программных модулей, способных работать на серверах в глобальных и локальных сетях с различными операционными системами (технология Java Beans). Пока основной недостаток этого языка — невысокое быстродействие, так как язык Java интерпретируемый.

Обзор языков программирования высокого уровня

- **C#** (Си Шарп). В конце 90-х годов в компании Microsoft под руководством Андерса Хейльсберга был разработан язык C#. В нем воплотились лучшие идеи Си и Си++, а также достоинства Java. Правда, C#, как и другие технологии Microsoft, ориентирован на платформу Windows. Однако формально он не отличается от прочих универсальных языков, а корпорация даже планирует его стандартизацию. Язык C# предназначен для быстрой разработки .NET-приложений, и его реализация в системе Microsoft Visual Studio .NET содержит множество особенностей, привязывающих C# к внутренней архитектуре Windows и платформы .NET.

Языки программирования для Интернета

- С активным развитием глобальной сети было создано немало реализаций популярных языков программирования, адаптированных специально для Интернета Все они отличаются характерными особенностями: языки являются интерпретируемыми, интерпретаторы для них распространяются бесплатно, а сами программы — в исходных текстах. Такие языки называют скрипт-языками.

- **HTML**. Общеизвестный язык для оформления документов. Он очень прост и содержит элементарные команды форматирования текста, добавления рисунков, задания шрифтов и цветов, организации ссылок и таблиц. Все Wei-страницы написаны на языке HTML или используют его расширения.

Языки программирования для Интернета

- **VRML.** В 1994 году был создан язык VRML для организации виртуальных трехмерных интерфейсов в Интернете. Он позволяет описывать в текстовом виде различные трехмерные сцены, освещение и тени, текстуры (покрытия объектов), создавать свои миры, путешествовать по ним, «облетать» со всех сторон, вращать в любых направлениях, масштабировать, регулировать освещенность и т.д.

Языки программирования для Интернета

- **XML.** В августе 1996 года WWW-консорциум, ответственный за стандарты на Интернет-технологии, приступил к подготовке универсального языка разметки структуры документов, базировавшегося на достаточно давно созданной в IBM технологии SGML. Новый язык получил название XML. Сегодня он служит основой множества системных, сетевых и прикладных приложений, позволяя представлять в прозрачном для пользователей и программ текстовом виде различные аспекты внутренней структуры иерархически организованных документов. В недалеком будущем он может стать заменой HTML.

Прочие языки программирования

- **PL/I** (ПЛ/1). В середине 60-х годов компания IBM решила взять все лучшее из языков Фортран, Кобол и Алгол. В результате в 1964 году на свет появился новый компилируемый язык программирования, который получил название Programming Language One. В этом языке было реализовано множество уникальных решений, полезность которых удается оценить только спустя 33 года, в эпоху крупных программных систем. По своим возможностям ПЛ/1 значительно мощнее многих других языков (Си, Паскаля). Например, в ПЛ/1 присутствует уникальная возможность указания точности вычислений — ее нет даже у Си++ и Явы. Этот язык и сегодня продолжает поддерживаться компанией IBM.

Прочие языки программирования

- **Smalltalk** (Смолток). Работа над этим языком началась в 1970 году в исследовательской лаборатории корпорации XEROX, а закончились спустя 10 лет, воплотившись в окончательном варианте интерпретатора SMALLTALK-80. Данный язык оригинален тем, что его синтаксис очень компактен и базируется исключительно на понятии объекта. В этом языке отсутствуют операторы или данные. Все, что входит в Смолток, является объектами, а сами объекты общаются друг с другом исключительно с помощью сообщений (например, появление выражения 1+1 вызывает посылку объекту I сообщения «+», то есть «прибавить», с параметром 1, который считается не числом-константой, а тоже объектом). Больше никаких управляющих структур, за исключением «оператора» ветвления (на самом деле функции, принадлежащей стандартному объекту), в языке нет, хотя их можно очень просто смоделировать. Сегодня версия Visual Age for Smalltalk активно развивается компанией IBM.

Прочие языки программирования

- **Ada** (Ада). Назван по имени леди Огасты Ады Байрон, дочери английского поэта Байрона и его отдаленной родственницы Анабеллы Милбэнк. В 1980 году сотни экспертов Министерства обороны США отобрали из 17 вариантов именно этот язык, разработанный небольшой группой под руководством Жана Ишбиа. Он удовлетворил на то время все требования Пентагона, а к сегодняшнему дню в его развитие вложены десятки миллиардов долларов. Структура самого языка похожа на Паскаль. В нем имеются средства строгого разграничения доступа к различным уровням спецификаций, доведена до предела мощность управляющих конструкций.

Прочие языки программирования

- **Forth** (Форт). Результат попытки Чарльза Мура в 70-х годах создать язык, обладающий мощными средствами программирования, который мог быть эффективно реализован на компьютерах с небольшими объемами памяти, а компилятор мог бы выдавать очень быстрый и компактный код, то есть служил заменой ассемблеру. Однако сложности восприятия программного текста, записанного в непривычной форме, сильно затрудняли поиск ошибок, и с появлением Си язык Форт оказался забытым.