

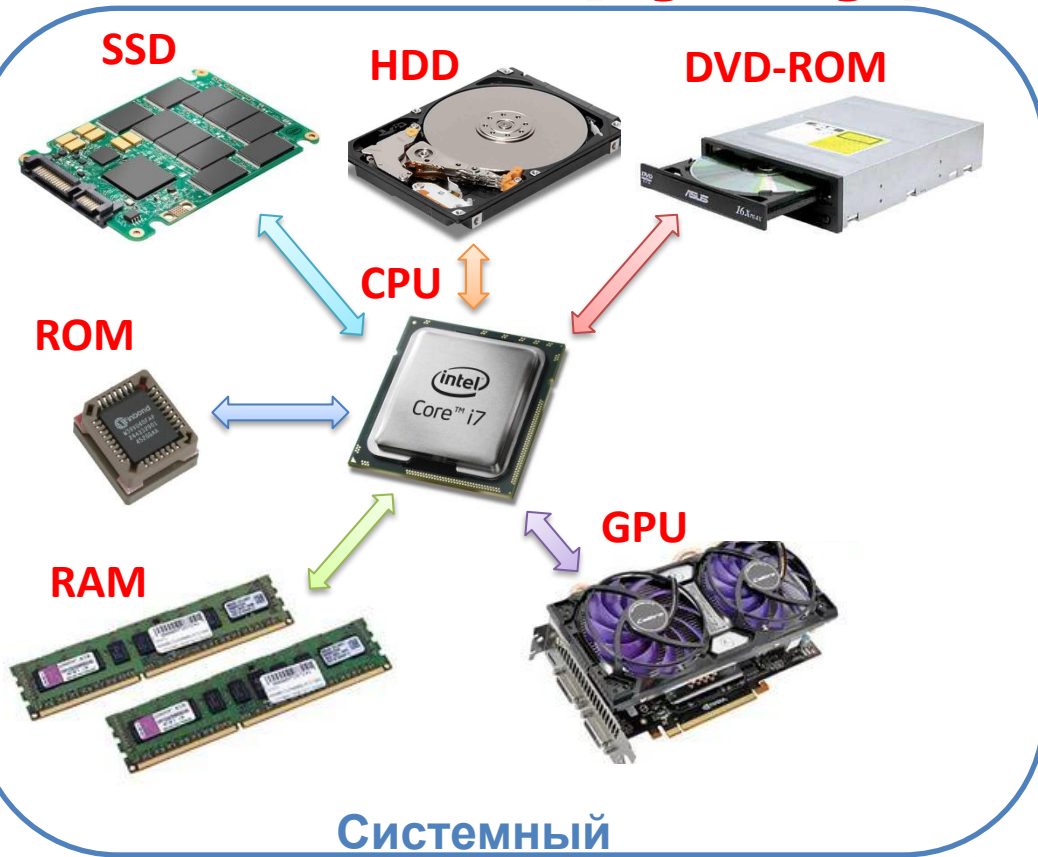
Основы алгоритмизации и программирования

Лекция 1 Вводные понятия

Составитель: ассистент кафедры общей
физики

Побияха Александр Сергеевич

Структура ПЭВМ



CPU – (*Central Processing Unit*)

исполняет машинные инструкции (код программ) и управляет работой

всего компьютера

RAM – (*Random Access Memory*) позволяет получить доступ к любой ячейке по её адресу

на чтение или запись. Здесь

располагаются

ROM – (*Read-Only Memory*) выполняемые программы. энергонезависимая

память, используется для хранения массива

неизменяемых данных, которые

GPU – (*Graphics Processing Unit*) необходимы для запуска компьютера. отдельное устройство ПЭВМ

выполняющее

получение изображения по модели

с помощью компьютерной программы

HDD – (*Hard (Magnetic) Disk Drive*) запоминающее устройство, основанное на принципе магнитной записи

SSD – (*Solid-State Drive*) немеханическое запоминающее устройство на основе микросхем памяти

DVD-ROM – (*Digital Versatile Disc*) устройство компьютера для записи и/или считывания данных записанных на съёмный носитель информации в форме DVD диска.

Размещение программ в ПЭВМ

Данные и программы в ПЭВМ размещаются в оперативной памяти. С точки зрения разработчика программного обеспечения оперативная память представляет собой **упорядоченную последовательность ячеек** — байт, предназначенных для размещения данных, которыми оперирует программа во время своего выполнения.

Упорядоченность означает, что каждый элемент последовательности (каждая ячейка памяти) имеет свой порядковый номер. Этот порядковый номер называют **адресом** ячейки памяти — адресом байта. Непрерывный диапазон ячеек, доступный для адресации в конкретной операционной системе, называют **адресным пространством**.

Оперативная память (Адресное пространство)

Ячейка с номером
4294967295

старшие адреса памяти

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	...
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

младшие адреса памяти

Размещение программ в ПЭВМ

Минимальная адресованная ячейка состоит из **8 двоичных позиций** (в каждую позицию может быть записан либо **“0”**, либо **“1”**).

Объем информации, который помещается в одну позицию называется **битом**. Объем информации, состоящий из **8 бит** называется **байтом**.

1 байт = $2^3 = 8$ бит

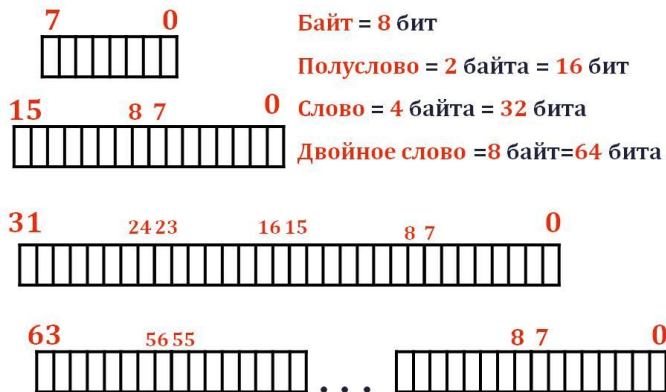
1 Кбайт (килобайт) = 2^{10} байт = 1024 байт

1 Мбайт (мегабайт) = 2^{10} Кбайт = 1024 Кбайт = 2^{20} байт

1 Гбайт (гигабайт) = 2^{10} Мбайт = 1024 Мбайт = 2^{30} байт

1 Тбайт (терабайт) = 2^{10} Гбайт = 1024 Гбайт = 2^{40} байт

1 Пбайт (петабайт) = 2^{10} Тбайт = 1024 Тбайт = 2^{50} байт



Общее количество доступных для адресации ячеек памяти определяется разрядностью операционной системы, более точно — разрядностью типа данных, используемого в конкретной операционной системе для хранения номера ячейки.

Современные операционные системы в своём большинстве представлены **32- и 64-разрядными версиями**. В **32-разрядных операционных системах** для работы с номерами ячеек памяти используются **32-разрядные (4-байтные)** типы данных. Адресовать в таких системах можно до **4294967296 ячеек** — максимальное число, которое можно представить 32 разрядами, что в точности соответствует **4 Гб** адресного пространства. В **64-разрядных операционных системах** для работы с номерами ячеек памяти используются **64-разрядные (8-байтные)** типы данных. Количество ячеек, которое можно адресовать в таких системах — **16 экзбайт**.

Размещение программ в ПЭВМ

При размещении данных производится их запись с помощью нулей и единиц – **кодирование**, при котором каждый символ заменяется последовательностью из 8 двоичных разрядов в соответствии со стандартной кодовой таблицей **ASCII**.

Например, символы: **D (код 68) – 01000100**, **G (код 71) – 01000111**, **7 (код 55) – 00110111**

При кодировании, числа (коды) преобразуются в двоичное представление следующим образом: **$2 = 1 \cdot 2^1 + 0 \cdot 2^0$** ; **$5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$** ;

$256 = 1 \cdot 2^8$

32 пробел	48 0	64 @	80 P	96 `	112 p
33 !	49 1	65 A	81 Q	97 a	113 q
34 "	50 2	66 B	82 R	98 b	114 r
35 #	51 3	67 C	83 S	99 c	115 s
36 \$	52 4	68 D	84 T	100 d	116 t
37 %	53 5	69 E	85 U	101 e	117 u
38 &	54 6	70 F	86 V	102 f	118 v
39 '	55 7	71 G	87 W	103 g	119 w
40 (	56 8	72 H	88 X	104 h	120 x
41)	57 9	73 I	89 Y	105 i	121 y
42 *	58 :	74 J	90 Z	106 j	122 z
43 +	59 ;	75 K	91 [	107 k	123 {
44 ,	60 <	76 L	92 \	108 l	124
45 -	61 =	77 M	93]	109 m	125 }
46 .	62 >	78 N	94 ^	110 n	126 ~
47 /	63 ?	79 O	95 _	111 o	127

Важно

С увеличением числа, количество разрядов резко увеличивается, поэтому для размещения большего числа выделяется несколько подряд расположенных байт. В этом случае адресом ячейки является адрес первого байта, один бит которого выделяется под знак числа.

Размещение программ в ПЭВМ

Программа – последовательность команд (инструкций), которые помещаются в памяти и выполняются процессором в указанном порядке.

Команда размещается в комбинированной ячейке следующим образом: в первом байте – код операции (**КОп**), которую необходимо выполнить над содержимым ячеек; в одной, двух или трех ячейках (операндах команды) по 2, 4 или 8 байт – адреса ячеек (**А1, А2, А3**), над которыми нужно выполнить указанную операцию. Номер первого байта называется адресом команды. Последовательность данных команд называется программой в машинных кодах.

Программа записывается на языке высокого уровня, наиболее удобном для реализации алгоритма решения определённого класса задач.

Машинный

код

0005	B4 09
0007	BA0000r
00A	CD 21

Ассембл

ер

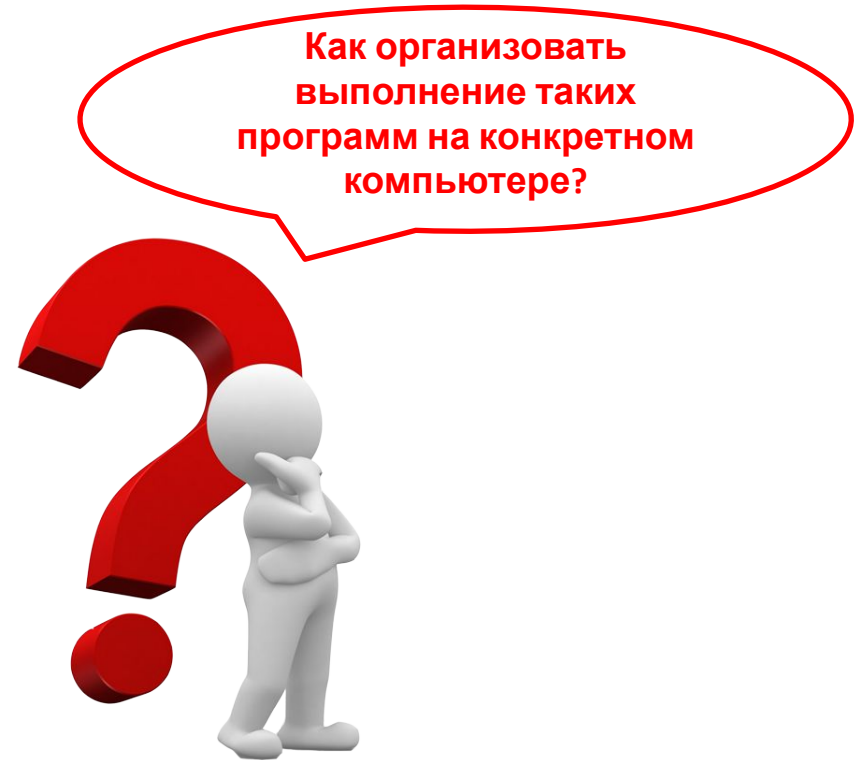
7	mov AH, 09h
8	mov DX, offset msg
9	int 21 h

5432	AC	LDSB	
5433	A5	MOUSW	
5434	E2E0	LOOP	5416
5436	AE	SCASB	
5437	A220AF	MOV	[AF20], AL
543A	AE	SCASB	
543B	A7	CMPSW	
543C	A2AEAB	MOV	[ABAE], AL
543F	EF	OUT	DX, AX
5440	A5	MOUSW	
5441	E220	LOOP	5463
5443	A2EBA2	MOV	[A2EB], AL
5446	A5	MOUSW	
5447	E1E2	LOOPZ	542B
5449	A820	TEST	AL, 20
544B	E2A5	LOOP	53F2
544D	AA	STOSB	
544E	E3E9	JCZ	5439
5450	A8A5	TEST	AL, A5

Трансляция и интерпретация программ

Подавляющее большинство программ пишется на языках программирования **высокого уровня**, существенно отличающихся от машинного языка реального компьютера. Теоретически можно создать аппаратный компьютер, использующий некий язык высокого уровня в качестве машинного, но это было бы очень сложно и дорого.

Более практичное решение - аппаратно реализовать язык **очень низкого уровня**, что обеспечит выполнение наиболее распространенных элементарных операций и снабдить компьютер дополнительным программным обеспечением для взаимодействия с программами, написанными на языках высокого уровня.



Трансляция и интерпретация программ

Трансляция (компиляция) - это метод перевода программ, написанных на языках высокого уровня, в эквивалентные программы на машинном языке используемого компьютера. После этого интерпретатор, встроенный в аппаратную часть микропроцессора, непосредственно выполняет оттранслированную в машинный код программу. Преимущество этого метода - очень быстрое выполнение программы после завершения процесса трансляции.

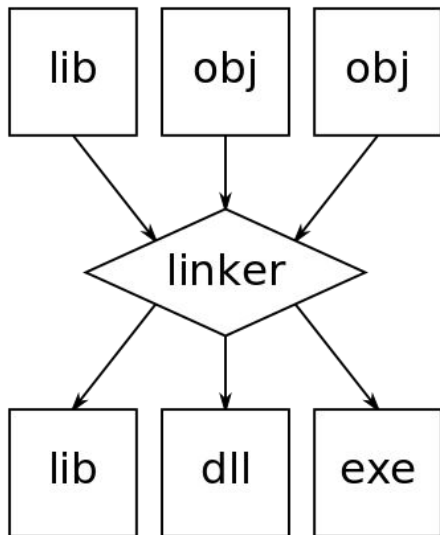
Ассемблер - это транслятор, у которого исходным языком является символическое представление машинного кода (ассемблер), а объектным языком является некая разновидность машинного языка какого-либо реального компьютера.

Транслятор - это языковой процессор, который воспринимает программы на некотором исходном языке в качестве входных данных, а на выходе выдает эквивалентные по своей функциональности программы, но уже на другом, так называемом объектном языке (который также может быть произвольного уровня).

Компилятор — транслятор, для которого исходным является язык высокого уровня, а его объектный язык близок к машинному языку реального компьютера. Это либо язык ассемблера, либо какой-нибудь вариант машинного языка. Например, программы на языке С компилируются, как правило, в программы на языке ассемблера, которые затем транслируются ассемблером в машинный язык.

Трансляция и интерпретация программ

Компоновщик (редактор связей , linker) - это транслятор, у которого исходный язык состоит из программ на машинном языке в перемещаемой форме и таблиц данных, указывающих те точки, в которых перемещаемый код должен быть модифицирован, чтобы стать выполняемым. Объектный язык состоит из готовых к выполнению машинных команд. Задачей компоновщика является создание единой выполняемой программы.



Препроцессор (макροпроцессор) - это транслятор, исходный язык которого является расширенной формой какого-либо языка высокого уровня (например, Java или C++), а объектный язык — стандартной версией этого языка. Объектная программа, созданная препроцессором, готова к трансляции и выполнению обычными процессорами исходного стандартного языка.

Главным недостатком трансляции является **потеря информации о программе**. Если в программе есть ошибка, то часто трудно определить, какой из операторов программы на исходном языке выполнялся и какие объекты данных использовались в нем. Кроме того, поскольку оператор на языке высокого уровня содержит гораздо больше информации, чем команда машинного языка, то исполняемая форма программы занимает в памяти гораздо больше места.



Трансляция и интерпретация программ

Интерпретация (программная имитация) - это метод, когда при помощи программы (**интерпретатора**), выполняемой на аппаратном компьютере, создается виртуальный компьютер с машинным языком высокого уровня. Интерпретатор декодирует и выполняет каждый оператор программы на языке высокого уровня в соответствующей последовательности и производит вывод результирующих данных, определяемый этой программой.

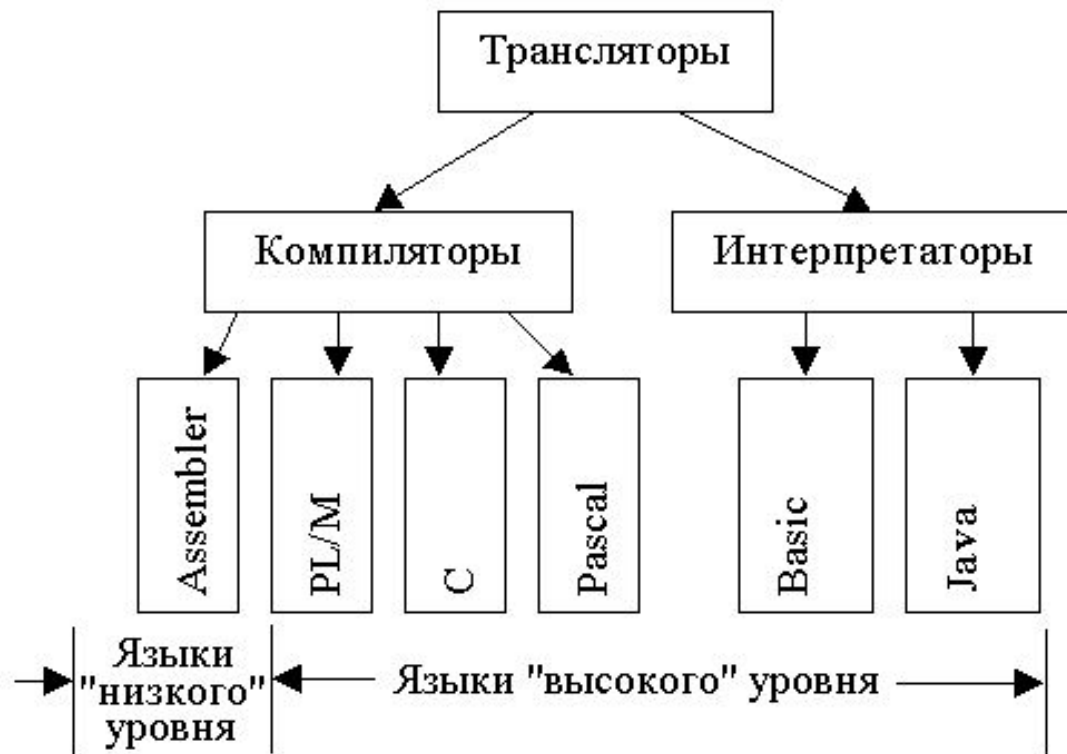


Недостаток : необходимость многократно декодировать один и тот же оператор, если он, например, встречается в цикле или подпрограмме, что существенно снижает скорость выполнения таких программ (в 10... 100 раз).

Достоинство: легкость реализации многих операций отладки на уровне исходной программы, поскольку все сообщения об ошибках, возникающих в процессе выполнения, могут ссылаться на исходные модули программы.

Трансляция и интерпретация программ

Чаще всего для реализации языка высокого уровня на компьютере используется **комбинированный подход**. Сначала программа транслируется из своей исходной формы в форму, более удобную для выполнения. Обычно это делается путем создания нескольких независимых частей программы, называемых модулями. На этапе загрузки эти независимые части объединяются с набором программ поддержки выполнения, реализующих программно-моделируемые (интерпретируемые) операции. Это приводит к созданию выполняемой формы программы, операторы которой декодируются и выполняются посредством их интерпретации.



Среды и реализации языков программирования

Среда программирования - это совокупность инструментов, используемых при разработке программного обеспечения. Этот набор обычно состоит из **файловой системы, текстового редактора, редактора связей и компилятора**. Дополнительно он может включать большое количество инструментальных комплексов с единообразным интерфейсом пользователя.

Последнюю стадию развития сред разработки ПО представляют **Microsoft Visual C++, Visual BASIC, Delphi и Java Development Kit**, которые предлагают легкий способ создания графических интерфейсов для программ пользователя.

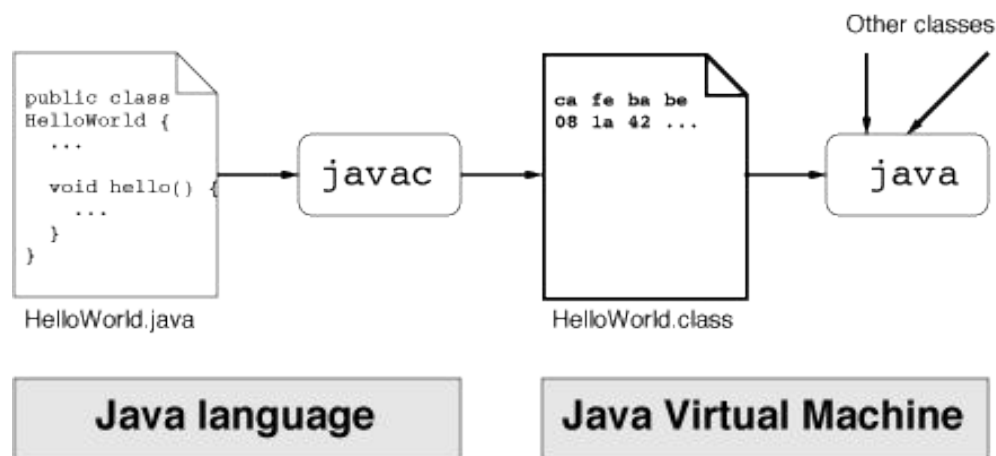
Ключевой вопрос реализации языка программирования заключается в том, какое представление имеет программа во время ее выполнения на реальном компьютере, является ли этот язык машинным языком данного компьютера или нет? В зависимости от ответа на этот вопрос языки (вернее, их реализации) делятся на **компилируемые** и **интерпретируемые**.

Компилируемые языки. Компилируемыми принято считать такие языки как **C, C++, FORTRAN, Pascal и Ada**. Это означает, что программы, написанные на этих языках, транслируются в машинный код данного компьютера перед началом выполнения.

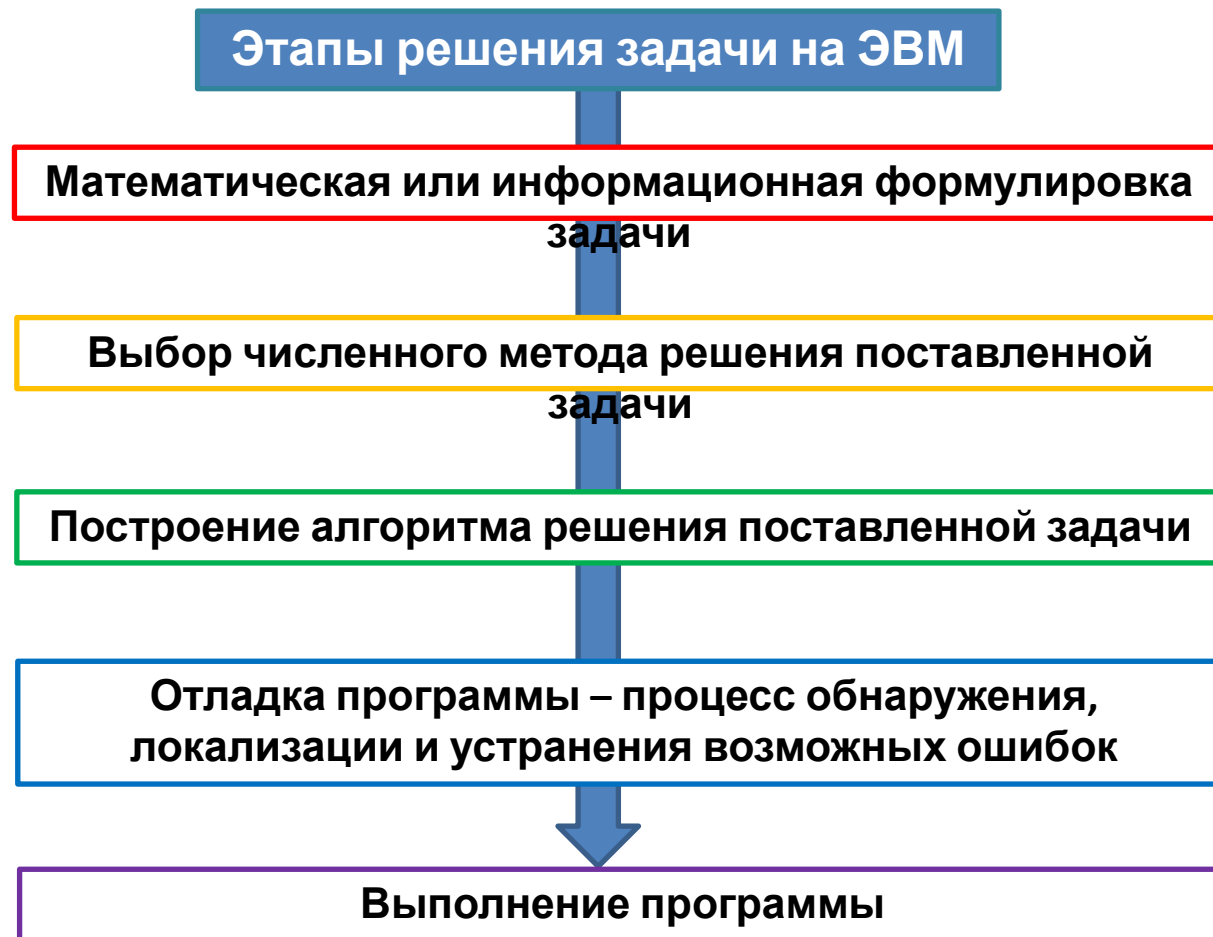
Интерпретируемые языки. Реализуются с использованием программного интерпретатора. К таковым относятся языки **LISP, ML, Perl, Postscript, Prolog и Smalltalk**. При такой реализации транслятор выдает не машинный код используемого компьютера, а некую промежуточную форму программы. Эта форма легче для выполнения, чем исходная программа, но все же она отличается от машинного кода.

Среды и реализации языков программирования

Развитие **Всемирной паутины WWW** и появление языка **Java** внесли изменения в описанную схему. Язык **Java** похож скорее на **Pascal** и **C++**, чем на **LISP**, но в большинстве случаев реализуется как **интерпретируемый** язык. Компилятор **Java** вырабатывает промежуточный набор байт-кодов для **виртуальной машины Java**. Передача байт-кодов на локальный компьютер (даже если он медленнее, чем **web-сервер**) выгоднее в отношении временных затрат, чем передача результатов выполнения программы на **web-сервере**. Однако **web-сервер** не в состоянии предугадать машинную архитектуру хост-компьютера. Поэтому браузер создает **виртуальную машину Java**, которая и выполняет стандартный набор **байт-кодов Java**.

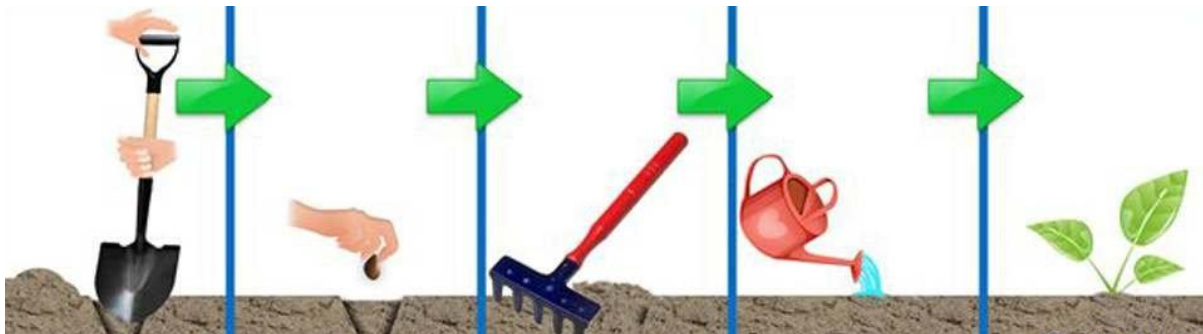


Понятие алгоритмов и способы их описания

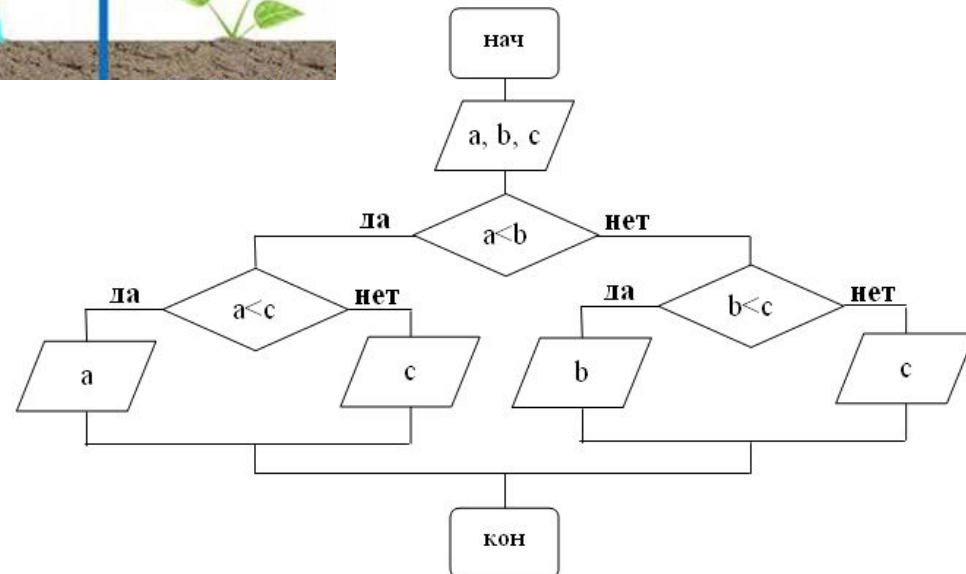


Понятие алгоритмов и способы их описания

Алгоритм — набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий



Числовой алгоритм — детально описанный способ преобразования числовых входных данных в выходные при помощи математических операций



Понятие алгоритмов и способы их описания

Основные свойства

Алгоритм

Результативность означает возможность получения результата после выполнения конечного количества операций.

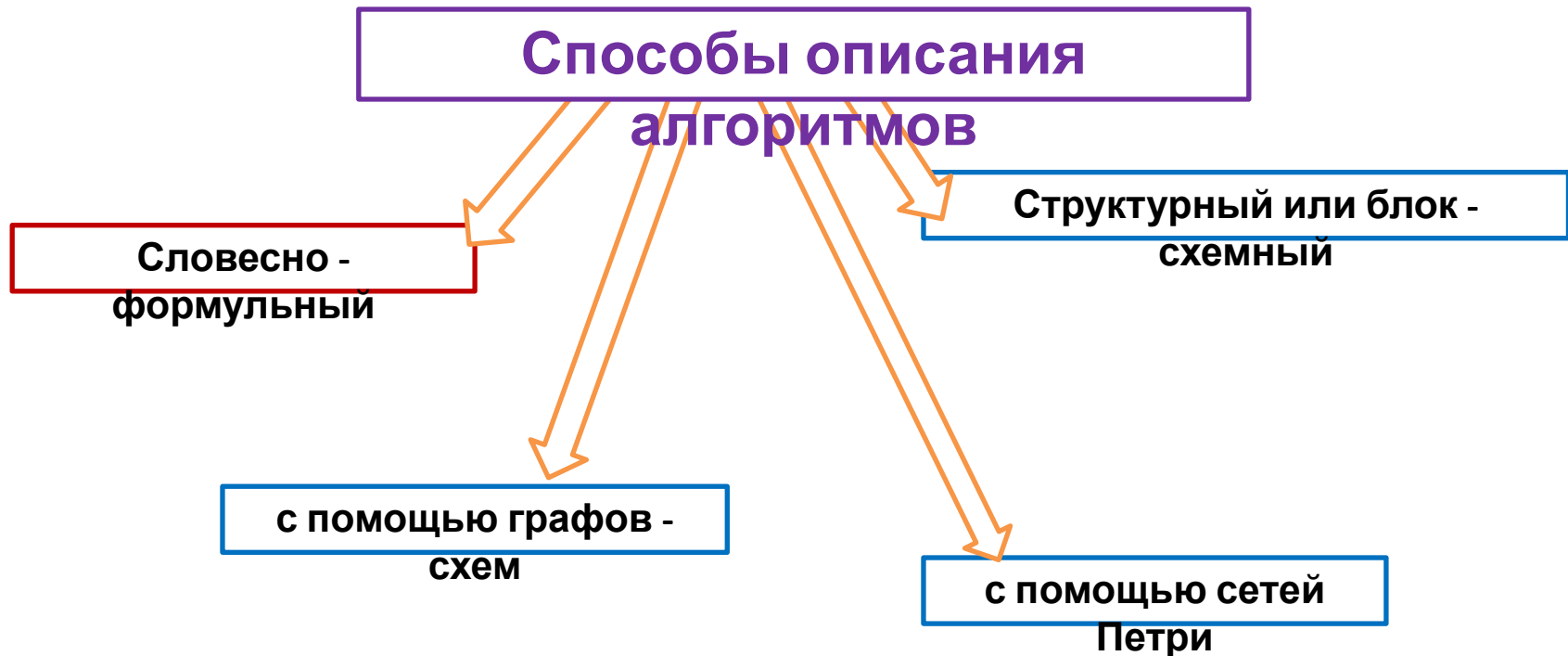
Определенность состоит в совпадении получаемых результатов независимо от пользователя и применяемых технических средств.

Массовость заключается в возможности применения алгоритма к целому классу однотипных задач, различающихся конкретными значениями исходных данных.

Дискретность — возможность разделения процесса вычислений, предписанных алгоритмом, на отдельные этапы, возможность выделения участков программы с определенной структурой.



Понятие алгоритмов и способы их описания



Понятие алгоритмов и способы их описания

При **словесно-формульном** способе алгоритм записывается **в виде текста с формулами** по пунктам, определяющим последовательность действий

Пример

Даны два целых положительных числа, найти их **наибольший общий делитель (НОД)**. Решение этой задачи может быть получено последовательным делением вначале большего числа на меньшее, затем меньшего числа на полученный остаток, первого остатка на второй остаток и т.д. до тех пор, пока в остатке не получится нуль. Последний по счету делитель и будет искомым результатом.

1. задать два числа M и N ;
2. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3. определить большее из чисел;
4. заменить большее из чисел разностью большего и меньшего из чисел;
5. повторить алгоритм с шага 2;
2. Ввести (M, N) ;
3. Если $M \leq N$, то перейти к п.4, иначе перейти к п. 7;
4. Если $M > N$, то перейти к п. 5, иначе перейти к п. 6;
5. $M := M - N$; перейти к п. 3;
6. $N := N - M$; перейти к п. 3;
7. $\text{НОД} := M$;
8. Печатать (НОД).

Понятие алгоритмов и способы их описания

При **блок - схемном** описании алгоритм изображается геометрическими фигурами (блоками), связанными по управлению линиями (направлениями потока) со стрелками. В блоках записывается последовательность действий

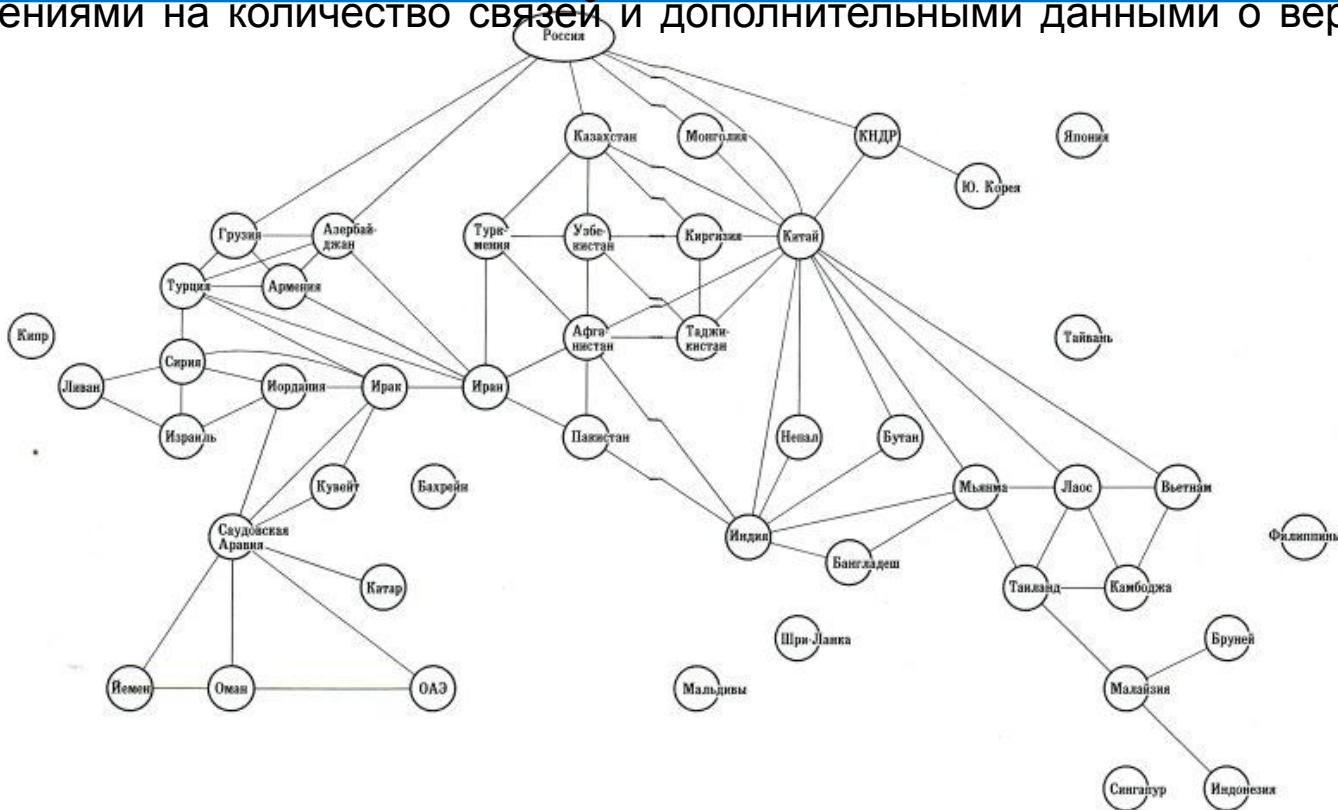


Схема - это графическое представление алгоритма, дополненное элементами словесной записи. Каждый пункт алгоритма отображается на схеме некоторой геометрической фигурой-блоком (блочным символом), причем различным по типу выполняемых действий блокам соответствуют различные геометрические фигуры, изображаемые по ГОСТу.

Понятие алгоритмов и способы их описания

Граф (graph) — основной объект изучения математической теории графов, совокупность непустого множества вершин и наборов пар вершин (связей между вершинами).

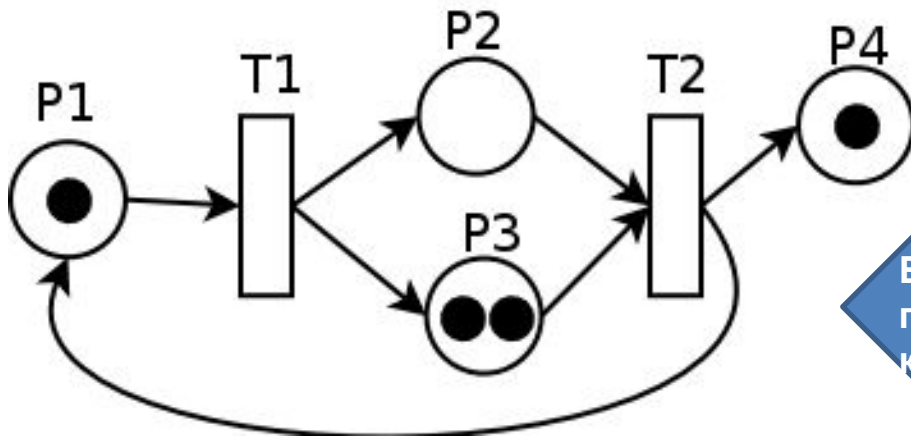
Объекты представляются как вершины, или узлы графа, а связи — как дуги, или рёбра. Для разных областей применения виды графов могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах.



Понятие алгоритмов и способы их описания

Сеть Петри представляет собой двудольный ориентированный **мультиграф**, состоящий из вершин двух типов — позиций и переходов, соединённых между собой дугами. Вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

Сеть Петри есть **мультиграф**, так как он допускает существование кратных дуг от одной вершины графа к другой. Так как дуги являются направленными, то это ориентированный мультиграф. Вершины графа можно разделить на два множества (позиции и переходы) таким образом, что каждая дуга будет направлена от элемента одного множества (позиций или переходов) к элементу другого множества (переходов или позиций); следовательно, такой граф является двудольным ориентированным мультиграфом.



Белыми кружками обозначены позиции, полосками — переходы, чёрными кружками — метки.

Понятие алгоритмов и способы их описания

ПРАВИЛА ИЗОБРАЖЕНИЯ БЛОК-СХЕМ АЛГОРИТМА

В блок-схеме можно использовать строго определённые типы блоков.

Для удобства блоки могут помечаться метками (буквами или цифрами)

Внутри блока **ввода/вывода** пишется **ВВОД** или **ВЫВОД** и перечисляются имена данных, подлежащих вводу/выводу

Внутри блока **действия** для присваивания переменных значений используется знак присваивания

Стрелки на линиях связи можно не ставить при направлении **сверху вниз** и **слева направо**; противоположные направления обязательно указывают стрелкой на линии.

Понятие алгоритмов и способы их описания

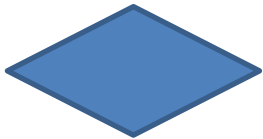
ОСНОВНЫЕ ТИПЫ БЛОКОВ



- начало или конец описания алгоритмов



- ввод исходных данных или вывод результатов



- блок проверки условий, от которых зависит выбор направления алгоритма



- блок арифметических или других действий

