Алгоритмы и их свойства. основные конструкции алгоритмов

Зачем нужны алгоритмы

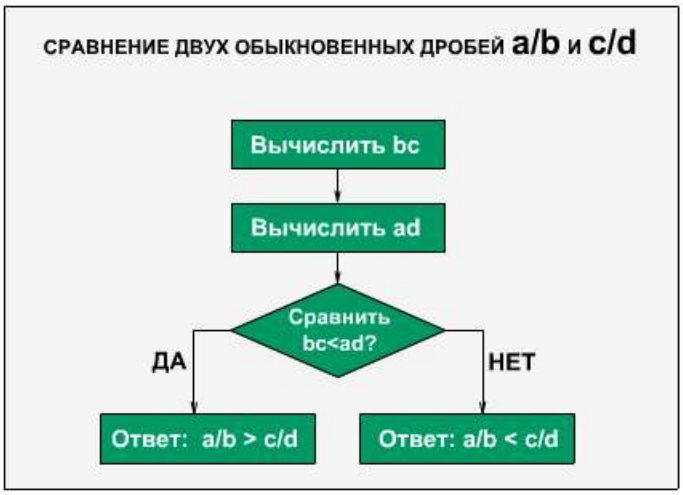
Слово «алгоритм» в нашей жизни встречается очень часто. Это слово употребляют люди самых разных профессий, оно звучит по радио и по телевидению. То и дело слышно: «алгоритм здоровья», «алгоритм успеха», много-много других «алгоритмов» объявляется и расхваливается на все лады.

А вот что это такое – алгоритм – объясняют редко. На то есть свои причины и в них мы будем разбираться ниже. Пока же отметим, что на житейском языке слово «алгоритм», видимо, может означать предписание, выполняя которое можно достигнуть некоторой цели. Например, здоровья. Или успеха.

Однако очевидно, что с «алгоритмами успеха», «алгоритмами здоровья» что-то не в порядке. Все хотят быть здоровыми, успешными и богатыми. Если бы был единый рецепт, как этого добиться, причем такой, чтобы этот рецепт каждый мог выполнить – все бы стали богатыми, успешными, здоровыми и счастливыми. А алгоритм должен быть исполнимым.

Как же проверить, является ли предлагаемое предписание действительно алгоритмом? В чем причина неуспеха при его использовании – в негодности такого алгоритма-предписания или в том, что его неправильно используют? Чтобы ответить на такой вопрос ответственно, необходимо прежде всего ввести достаточно строгие определения. А для этого – выявить необходимые свойства любого алгоритма и требования к нему. Этим мы и займемся.

Так что же такое алгоритм в самых общих чертах? Алгоритм - это набор инструкций, правильно выполняя которые можно гарантированно получить некий необходимый результат. Это, конечно, не строгое определение алгоритма, но пока рассмотрим примеры.



Слайд 1.1. Сравнение двух обыкновенных дробей

Как видно, алгоритм состоит из набора инструкций. Каждая отдельная инструкция подразумевается понятной тому, кто исполняет алгоритм. Инструкции выполняются последовательно, одна за другой.

Какие другие алгоритмы нам заведомо известны? Очень много алгоритмов уже проходились в школьной математике и физике – пусть они и не назывались напрямую алгоритмами. Например, порядок вычисления по любой формуле является алгоритмом. Посмотрим это на примере какой-нибудь совсем простой формулы.

Слайд 1.1.б. Алгоритм вычисления площади треугольника

Конечно, бывают формулы и гораздо сложнее, но их использование, то есть вычисления по формулам, производятся похожим образом. Ведь формулы строятся из хорошо известных действий (функций, операций), как в рассмотренном нами примере – умножений, сложений, извлечений корней и т.д. Эти действия – инструкции такого алгоритма. Разумеется, чтобы использовать эти алгоритмы, надо знать, как выполнять операции - как умножать, складывать, извлекать корни.

А ведь каждая из таких операций сама имеет алгоритм для вычисления, который построен из еще более простых операций. Существует даже много разных вариантов алгоритмов для каждой из таких операций, причем, конечно, каждый из таких алгоритмов дает правильный результат.

Приведем еще пример одного простого вычислительного алгоритма из школьной математики: это поиск ответа на вопрос – какая из двух данных конечных десятичных дробей больше?

Сравнение двух конечных десятичных дробей:

1) Даны две десятичные дроби:

23.6223 и 23.62241

2) Чтобы сравнить их, возьмем целые части от этих дробей:

В нашем случае - 23 и 23

3) Сравним их.

Если одна часть больше другой, то выдаем ответ: "соответствующая дробь больше другой".
Если части равны, следуем дальше.

4) Сравним первую десятичную цифру у каждого из чисел, в нашем случае:

6 и 6

Если у одного из чисел нет цифры, выдать ответ: "такое число меньше".

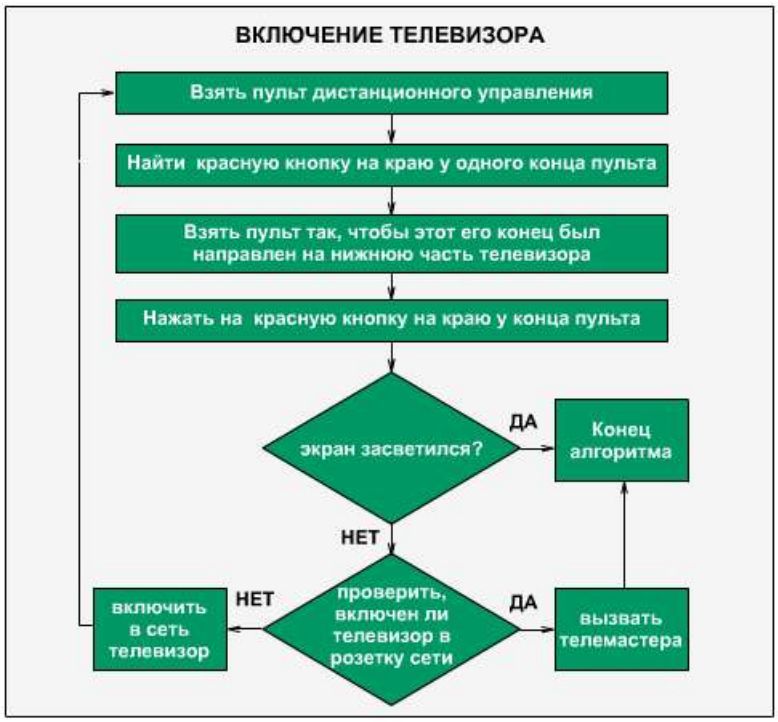
Если у обоих чисел нет цифр – выдать ответ: "числа равны"

Если цифры есть у обеих дробей и одна цифра больше другой - выдать ответ - "соответствующая дробь больше".

Если цифры есть у обеих дробей и они равны – расссмотреть следующую пару цифр после запятой.

В нашем случае цифры равны, по этому переходим к инструкции 4) и аналогично рассмотрим следующую пару цифр.

Очень похожи на алгоритмы инструкции к различным приборам. Например, к телевизору. Чтобы включить телевизор и смотреть программу, надо выполнить всего несколько действий, которые указаны в инструкции. Заметим, что при этом совсем не надо знать, как устроен телевизор.



Слайд "Включение телевизора"

Это, конечно, не самый лучший алгоритм – он не всегда оправдает ожидания пользователя.

Алгоритмами, по сути, является использование многочисленных компьютерных программ. Например, для того, чтобы играть в компьютерную игру, надо выполнить некоторые инструкции (при этом не надо знать, как эта игра «устроена изнутри»). Заметим, что иногда есть и алгоритмы, как выигрывать в такую игру.

Как можно заметить, выше все время повторялись слова «не надо знать». Используя алгоритмы, не надо знать, почему именно таким образом устроен алгоритм, почему с помощью такого алгоритма будет получен удовлетворительный результат. С помощью алгоритмов можно (гарантированно) получать некоторые результаты, исполняя некоторые достаточно простые инструкции. В смысл исполняемых инструкций вникать не имеет смысла. В этом - важнейшая из причин, по которой алгоритмы настолько широко распространены в нашей жизни: большая экономия усилий, прежде всего – в обучении. Действительно, чтобы что-то выучить, понять – надо затратить большие усилия и много времени.

Имеются еще и устройства, которые могут исполнять алгоритмы, но которые обучаться или понимать что-либо не могут вообще. Таких устройств сейчас множество (электронные «процессоры» прежде всего), а вот по-настоящему «знающих» или «обучающихся» устройств еще не придумано.

С точки зрения человека, алгоритм еще можно сравнить с замечательно сделанной шпаргалкой. В такой шпаргалке должно быть все написано абсолютно ясно и понятно, надо только выполнять предписания и все получится. А все предписания выбраны заранее так, что в них нет возможности ошибиться. Ну и, конечно, обыкновенными шпаргалками пользоваться запрещают (надо самому знать!), а вот алгоритмами пользуются все и при этом все очень довольны - алгоритмы очень, очень упрощают жизнь.

Совсем другая задача – составление алгоритмов, т.е. самих способов решения задач. Даже при кажущейся внешней простоте это бывает достаточно сложно. Попробуйте записать алгоритм для умножения любых двух натуральных чисел. Вы этот алгоритм прекрасно «знаете» и постоянно им пользуетесь. А вот правильно записать этот алгоритм (чтобы, если найдется человек, который знает только умножение чисел, меньших десяти, и сложение, то он мог его исполнять правильно) – это сделать непросто. Попробуйте сами – напишите его так, чтобы во всех случаях этот алгоритм работал правильно.

Ввод и вывод данных, повторные исполнения инструкций

Алгоритмы, как правило, представляют интерес тогда, когда с их помощью можно решить много проблем, которые похожи друг на друга, но отличаются в некоторых заданных величинах.

Приведем пример: алгоритм «умножить 213 на 336». Этот алгоритм существует, но сам по себе он не интересен: такая задача встречается, но довольно редко. Если уж кому-то требуется такой алгоритм, то он может быть записан в очень короткой форме: «71568». То есть просто выдан ответ: так устроены таблицы результатов. Поэтому все алгоритмы, которые применимы только в единственном случае, как правило, интереса не представляют, хотя алгоритмы, использующие таблицы, конечно, существуют.

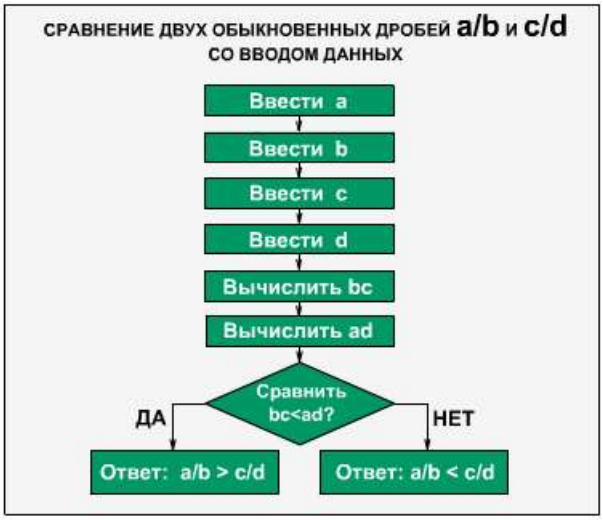
Другое дело – алгоритм умножения двух трехзначных чисел. Вот такая задача – умножить два некоторых трехзначных числа – встречается уже гораздо чаще. А алгоритм для нее делает не очень нужным отдельный алгоритм для умножения 213 на 336: это всего лишь одна из задач, которые можно решить таким алгоритмом.

И еще лучше – алгоритм умножения двух произвольных чисел – не обязательно трехзначных. Он может требоваться очень часто.

Таким образом, очень часто востребованными оказываются алгоритмы, у которых конкретно не указано, с какими числами они работают. Но алгоритм, когда он применяется, работает с конкретными числами. Поэтому в алгоритме обязательно должно быть указано, какие именно числа должны быть указаны для обработки алгоритму. Это делается с помощью инструкций ввода данных.

В приведенных уже алгоритмах ввод данных обычно выглядит так: «возьмите первое число», «возьмите второе число», «возьмите следующее число» и т.д. Откуда эти числа берутся? Их конкретные значения задаются, перед тем, как начинать исполнение алгоритма.

Конечно, исполнителю должно быть известно, откуда эти данные «брать», т.е. в каком они виде и где записаны. Посмотрим еще раз алгоритм сравнения двух дробей.

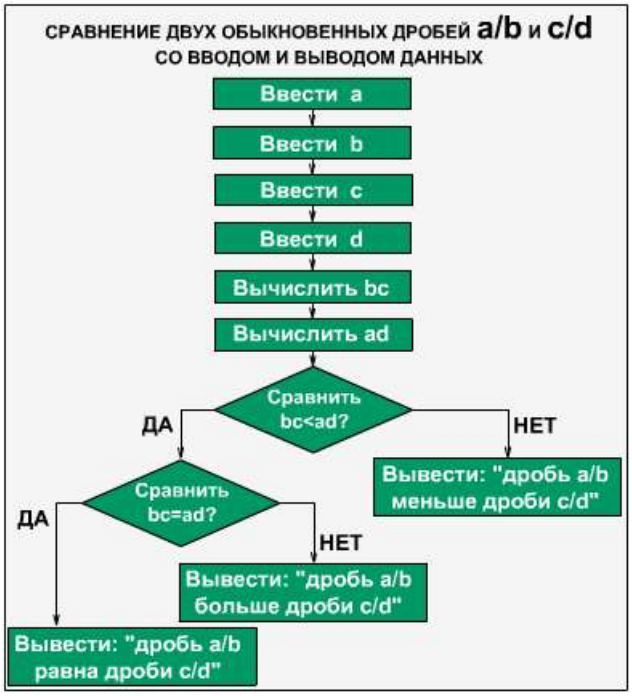


Слайд «Сравнение двух обыкновенных дробей»

В этом алгоритме есть ввод 4 чисел – числителя первой дроби, знаменателя второй, числителя второй и знаменателя первой.

Для того, чтобы использовать такой алгоритм, очевидно, исполнитель должен знать, что такое знаменатель и числитель. Эти знания можно исключить, если просто иметь четыре числа, написанные в определенном порядке и вместо «взять» использовать инструкции «прочитать»: «прочитать число». При этом, чтобы каждый раз не указывать, какое число прочитать, можно договориться так: каждый раз читается следующее число за последним уже прочитанным. Это упрощает работу исполнителя алгоритма.

Посмотрим, как может быть изменен алгоритм сравнения двух дробей при устроенном таким образом вводе данных



Слайд «Сравнение двух обыкновенных дробей с вводом и выводом данных»

Результаты, полученные исполнителем алгоритма, нужно иметь возможность использовать. Для этого исполнитель должен иметь возможность (инструкции) для вывода полученных результатов. Добавим в алгоритм такую инструкцию.

Еще одна важная особенность алгоритмов – это порядок выполнения команд.

Само собой разумеется, нужно заранее установить порядок выполнения инструкций. Самый простой – когда инструкции выполняются в том порядке, в каком они и написаны. Обычный порядок исполнения инструкций именно так и выбирается – исполнять инструкции одну за другой, как они записаны.

Если ограничиться только таким порядком исполнения, то, как легко увидеть, в ходе выполнения алгоритма будет выполнено ровно столько действий, сколько инструкций записано в алгоритме. Например, таков алгоритм сравнения простых дробей.

Это очень сильно ограничивает в возможностях алгоритмов. Например, одним и тем же алгоритмом нельзя будет сравнивать числа произвольной величины: ведь необходимо сравнить каждую цифру, а это – отдельное действие.

Посмотрим пример со сравнением десятичных дробей:

1) Даны две десятичные дроби:

23.6223 и 23.62241

2) Чтобы сравнить их, возьмем целые части от этих дробей:

В нашем случае - **23 и 23**

3) Сравним их.

Если одна часть больше другой, то выдаем ответ: "соответствующая дробь больше другой".
Если части равны, следуем дальше.

4) Сравним первую десятичную цифру у каждого из чисел, в нашем случае:

6 и 6

Если у одного из чисел нет цифры, выдать ответ: "такое число меньше".

Если у обоих чисел нет цифр – выдать ответ: "числа равны"

Если цифры есть у обеих дробей и одна цифра больше другой - выдать ответ - "соответствующая дробь больше".

Если цифры есть у обеих дробей и они равны – расссмотреть следующую пару цифр после запятой.

В нашем случае цифры равны, по этому переходим к инструкции 4) и аналогично рассмотрим следующую пару цифр.

Мы видим из примера, что в нем использована команда «начать заново («повторить») с такого то места («такой-то команды»).

Такую инструкцию и необходимо включить в состав алгоритмов, например, в таком виде: «перейти на инструкцию с такой-то отметкой». В нашем случае отметкой было число.

Слайд «Алгоритм, у которого исполнение сколь угодно долгое (бесконечное)»

В дальнейшем мы увидим, что таких инструкций по повторению частей алгоритмов может быть много, в т.ч. и более удобных для применения.

В виде «платы» за введение возможности неоднократного выполнения инструкций мы сталкиваемся с одной не очень приятной особенностью алгоритмов. Именно – что легко написать даже очень простой с виду алгоритм, который исполнитель никогда не сможет закончить.

Может возникнуть вопрос – как долго может исполняться исполнение алгоритма?

Конечно, лучше всего, если можно предсказать заранее, за какое число выполнений инструкций алгоритм выдаст ответ. Но такой подход снова приносит очень большие ограничения. Более того, если писать алгоритмы даже из очень небольшого набора простейших инструкций, то невозможно предсказать, закончится когда-нибудь исполнение такого алгоритма или нет. Это доказывается в специальной области математики, теории алгоритмов.

А сейчас посмотрим на алгоритм сравнения двух десятичных дробей. Он годится и для бесконечных десятичных дробей, только никогда не даст ответа, равны они или нет. Если окажется, что равны (и действительно бесконечны), то и алгоритм будет работать бесконечно.

Можно ли такого избежать? Оказывается, нет, но мы этого доказывать не будем.

Инструкции алгоритма и исполнители

Разберемся теперь с тем, что можно и нужно потребовать от исполнителя алгоритмов.

Во-первых, необходимо, чтобы он «понимал» инструкции в алгоритме. Тут мы, правда, нужно уточнить, о какой «самой малой степени» понимания может идти речь. Ведь, как уже говорилось, основное преимущество алгоритмов в том, что исполнитель не обязан знать, «почему».

Поэтому простейший вариант понимания инструкции исполнителем – это правильное исполнение такой инструкции. Требовать каких-либо объяснений по поводу исполнения инструкций от исполнителя излишне, если он никогда не делает ошибок. Ведь это и есть основное свойство алгоритмов – получение результата и исключение при этом ошибок.

Кроме того, требование только безошибочного исполнения (а не объяснений) позволяет включить в число исполнителей и тех, кто думать не умеет вовсе – например, всевозможные механизмы и устройства. Конечно, прежде всего такими устройствами оказываются компьютеры, которые и предназначены исключительно для исполнения алгоритмов. Именно это делает алгоритмы настолько важной составляющей в сегодняшней жизни – большинство алгоритмов исполняется компьютерами, а сами компьютеры достаточно недороги, чтобы использовать их повсеместно, даже в самых простых устройствах.

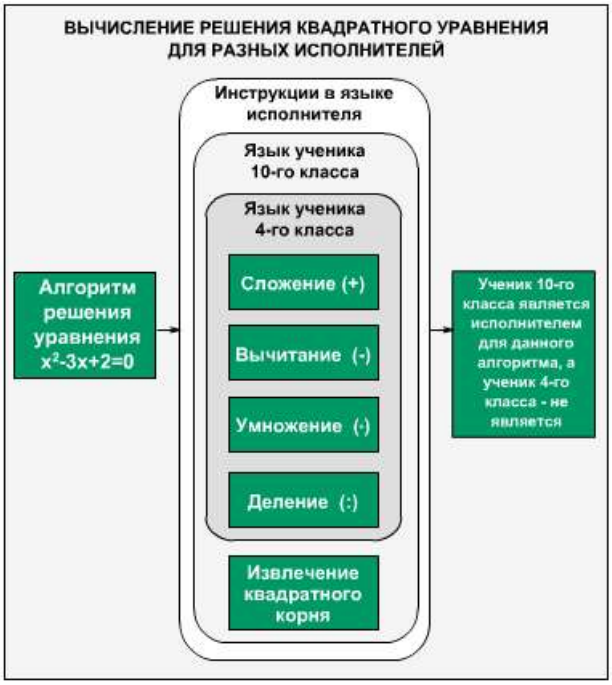
Таким образом, основное требование к исполнителю – безошибочное исполнение каждой инструкции алгоритма делает человека не лучшим исполнителем алгоритмов. У человека есть человеческие возможности ошибаться: он может устать, о чем-то постороннем подумать и т.д. Поэтому и оказывается, что наилучший исполнитель – тот, который хотя и вообще не думает, но в исполнении инструкций не ошибается - компьютер. К тому же компьютер, например, исполняет математические инструкции несравненно быстрее человека.

Но если исполнитель вообще не думает, а только исполняет – он не способен «учиться», а значит, у него есть только раз заложенный набор инструкций, которые он способен исполнять. Этот набор можно назвать языком исполнителя. Исполнитель способен выполнять только тот алгоритм, который не содержит инструкций, не входящих в язык исполнителя. Поэтому при написании алгоритмов всегда необходимо знать, какой язык является «разрешенным» - ведь иначе не понятно, для какого исполнителя предназначен этот алгоритм.

Ну а если исполнитель по каким-то причинам начал исполнять алгоритм, содержащий инструкции, не входящие в язык исполнителя? Это, конечно, можно обнаружить только дойдя до соответствующей инструкции. Как должен действовать исполнитель? Есть два очевидных варианта:

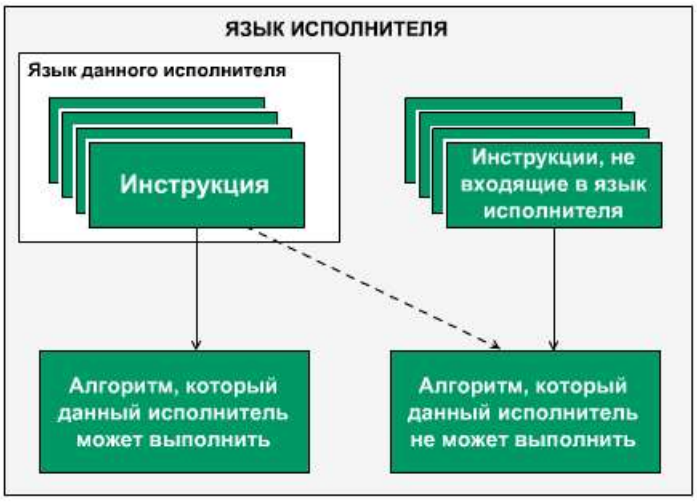
попытаться произвести какое-то действие, например, «похожее»;
ничего не делать, а еще лучше – выдать сообщение «ошибка: такой инструкции нет».

Второй вариант гораздо лучше первого по практическим соображениям. Действительно, действия «наугад» приведут почти наверняка к ошибке в результате. А если этого не заметят? А во втором случае будет известно, что в алгоритме ошибка – либо он «не так» написан, либо в нем другая ошибка (например, описка составителя алгоритма).



Слайд "Действия исполнителя, встретившего инструкцию не из языка данного исполнителя"

Так же само исполнитель должен поступать, когда инструкция из алгоритма есть в его языке, но она не исполнима. Например, инструкция «разделить a на b», а b равно 0. Здесь также желательно, чтобы исполнитель выдал результат – сообщение об ошибке.



Слайд "Ситуация двух исполнителей с разными языками"

Для любого написанного алгоритма можно собрать инструкции, которые необходимо «знать» исполнителю, чтобы выполнить этот алгоритм. Получается язык, но уже для данного алгоритма. Чтобы исполнитель смог этот алгоритм выполнять, нужно, чтобы каждая инструкция его языка входила в язык исполнителя.

Как пример рассмотрим некоторые инструкции, которые применяются в алгоритмах из школьной математики. Одна группа совершенно очевидна – это обычные вычислительные действия над числами. Среди них:

1) Сложить два числа.
2) Вычесть одно число из другого.
3) Умножить два числа.
4) Разделить одно число на другое

Слайд "Вычисление решения квадратного уравнения для разных исполнителей"

Можно добавить сюда и более сложные действия из числа тех, которые проходят в старших классах – например, возведение в степень, вычисление квадратного корня и другие. Но можно и не добавлять – ведь такие действия (операции) можно выразить через эти же сложения, умножения, вычитания и деления.

Вторая группа инструкций – не «непосредственные» операции с числами, а те, которые указывают порядок действий в алгоритме. Среди них

5) проверить условие (например, больше одного число другого или нет) – и, в соответствии с результатом проверки, перейти к одной из двух предусмотренных для этого инструкций (или из трех, если равенство чисел рассматривается как отдельный результат сравнения).

6) Инструкция «повторить (с такого-то места)». Разумеется, для каких–то новых чисел.

Еще одна группа - подстановки чисел вместо переменных (в формулах, например), запись и хранение данных при исполнении алгоритма (примерно как «запомним это число здесь»), а также ввод вывод данных (результатов).

Слайд "Некоторые инструкции для алгоритмов в матекматике"

Одно из требований к алгоритму – чтобы он при правильном использовании давал гарантированный автором алгоритма результат. Какие это может накладывать требования на алгоритмы?

Сначала выясним, что означает «гарантированный». Это означает – всегда правильный, т.е. и один и тот же. Чтобы каждый из «незнающих» сути алгоритма исполнителей, несмотря на свое незнание, всегда получал один и тот же результат, который туда «вложил» автор алгоритма.

Отсюда происходит требование к однозначности и строгости в алгоритмах.

Слайд "Требование однозначнгости результата"

Однозначной по результатам исполнения должна быть и каждая инструкция, и весь алгоритм в целом.

Но ведь можно придумать инструкции, которые, будучи выполненными вполне правильно, могут давать разные результаты. Например, такая инструкция:

«бросить монетку; если выпадет решка – прибавить единицу, если орел – вычесть единицу».

Эта инструкция, конечно, вполне выполнима, но не удовлетворяет требованию постоянства результата.

Мы поступаем так: все неоднозначные по результатам инструкции в алгоритмах допускать не будем. Причина такого ограничения на допустимые инструкции: иначе проверять «качество» составления алгоритма будет очень сложно. А качество алгоритма – это прежде всего отсутствие ошибок.

Не будет понятным и то, как можно проверить - правильно ли исполнитель выполняет ту или иную инструкцию, если у нее не однозначно определенный результат. Например, пусть какой-то исполнитель подкидывает монетку и при этом у него значительно чаще выпадает

«орел». Может такое случиться? Может. А может исполнитель просто «неправильно» выполнять подбрасывание (например, жульничать)? Тоже может.

Поэтому, чтобы избежать таких проблем, мы и отделяем алгоритмы с однозначно исполняемыми инструкциями от наборов инструкций, допускающих неоднозначные результаты. Последние тоже могут существовать, но их надо называть как-нибудь по-другому, у них иные свойства.

Каждый алгоритм должен решать какую-то проблему (или набор проблем). Что означает, что алгоритм решает такую проблему правильно?

Во-первых, алгоритм должен давать правильный результат на тех данных, на которые он рассчитан.

Слайд "Проверка правильности алгоритмов"

Во-вторых, алгоритм решает только те задачи, на которые он рассчитан. Например, если алгоритм должен умножать два натуральных числа, то, если его применить к двум дробным числам, этот алгоритм может выдать результат, который «ошибочен», т.е. к умножению дробей отношения не имеет. Каждый алгоритм обязан работать правильно только на таких данных, на которые он рассчитан.

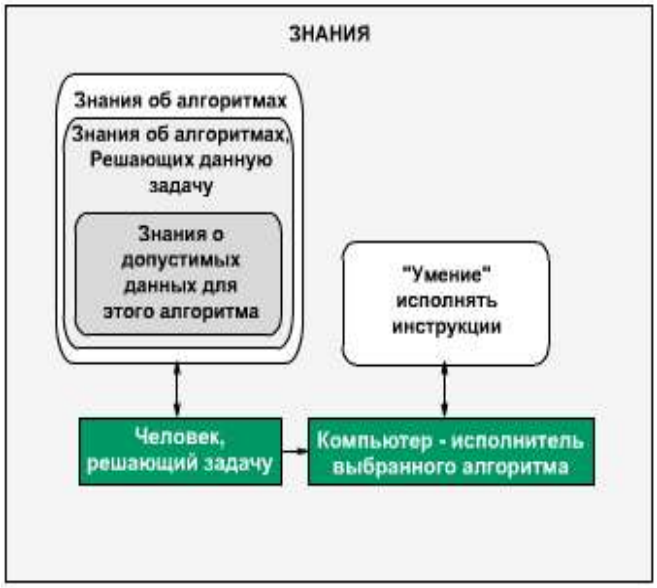
Слайд "Данные и правильная работа алгоритма"

Как мы уже говорили, преимущества алгоритмов в том, что они не требуют от исполнения многих знаний – например, про задачу, которую решает алгоритм. Но это не означает, что тот, кто решает эту задачу, не должен о ней знать. Например, если Вы хотите решить некоторую задачу с помощью компьютера (программы, алгоритма для этого компьютера), то компьютер, можно сказать, об этой задаче ничего не знает. Но Вы уже должны знать, можно ли именно эту задачу решить с помощью данного алгоритма (программы для компьютера).

Поэтому алгоритмы не заменяют вообще «обдумывания» и знаний. Но с помощью алгоритмов помощью оказывается измененной ценность знаний и то, кому и насколько эти знания нужны.

Огромная масса знаний - как достигнуть тот или иной результат с помощью действий стандартного типа, оказывается закрепленной в форме алгоритмов. Эти алгоритмы оказываются доступны не тому, кто «знает» сами эти алгоритмы, а тому, кто может их исполнить. Но одного исполнения недостаточно: нужно еще уметь сопоставлять задачи с алгоритмами, уметь выбрать подходящий алгоритм.

Поэтому по-настоящему алгоритмы - для того человека, разбирающегося в задачах, которые решаются этими алгоритмами. Поэтому по мере алгоритмизации решения различных проблем, возрастают требования к пониманию того, где и какие задачи решаются и в чем различаются друг от друга.



Слайд "Знания"

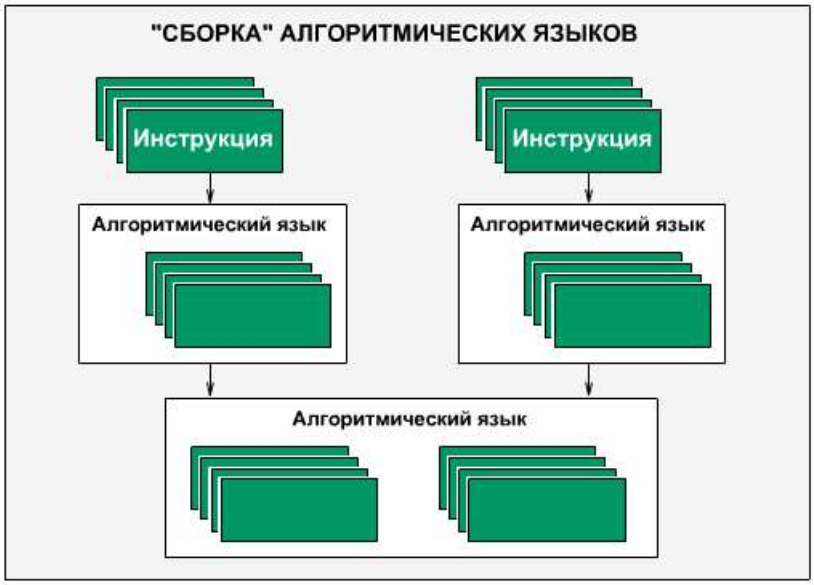
Исполнители алгоритмов и языки написания программ

Теперь, когда мы разобрались с тем, что мы хотели бы ожидать или можем ожидать от алгоритмов, попробуем собрать все это вместе, в одну сводку свойств алгоритмов.

Слайд: "Требования к алгоритмам"

Теперь соберем вместе свойства алгоритмов, которые мы выявили: Алгоритм – это набор инструкций с указанием порядка их исполнения. Алгоритмы выполняются исполнителями, которые только выполняют инструкции алгоритма в указанном в этом алгоритме порядке. Каждая инструкция представляет собой отдельное действие исполнителя, которое тот выполняет всегда одинаково («правильно»). Каждый исполнитель имеет свой набор инструкций, который он способен выполнить. Если в алгоритме есть инструкции, не входящие в такой набор, то исполнитель не сможет выполнить алгоритм. Таким образом, каждый алгоритм рассчитан на определенного исполнителя. Каждый (представляющий интерес) алгоритм обрабатывает некоторые данные, для которых в алгоритме имеются инструкции ввода. Результаты работы алгоритма выводятся также специальными инструкциями. Время работы исполнителя по алгоритму заранее не задается (не ограничивается). С помощью специальных способов одна и та же инструкция в алгоритме может выполняться много раз.

Инструкции алгоритма вместе взятые образуют язык алгоритма.



Слайд "Сборка алгоритмических языков"

В заключение обсудим, как конструируются реальные алгоритмические языки. Как мы видели, язык можно составить и для одного алгоритма. Но это очень редко имеет смысл (может быть, только для какого-нибудь исключительно ценного алгоритма).

Другое дело – язык алгоритмов для какой-то области задач, чтобы можно было использовать одного исполнителя. Такие языки есть, их начинает использоваться достаточно много. Однако основной причиной разработки таких языков является удобство человека – составителя алгоритмов, а не наличие одного исполнителя.

Оказалось, что проблему с исполнителем (особенно исполнителем-компьютером) можно решить, используя универсальные алгоритмические языки. Универсальный язык – тот, в который можно перевести алгоритм с любого другого языка. Переводчик при этом – снова алгоритм, конечно, свой собственный для каждой пары языков (с которого переводят и на который переводят). Такой алгоритм – транслятор – должен быть написан на языке исполнителя с универсальным языком. Тогда этот исполнитель сначала сам «переводит» данный ему на «постороннем» языке алгоритм на свой язык (используя алгоритм-транслятор примерно как словарь) и затем уже выполняет полученный «перевод». Таким образом, можно вообще обойтись только одним исполнителем – исполнителем с какого-то универсального языка, а всех остальных исполнителей заменить алгоритмами-трансляторами. На практике универсальных языков и исполнителей-процессоров (опять исполнитель - по-английски) не так мало, по причине разработки со временем все более совершенных образцов.

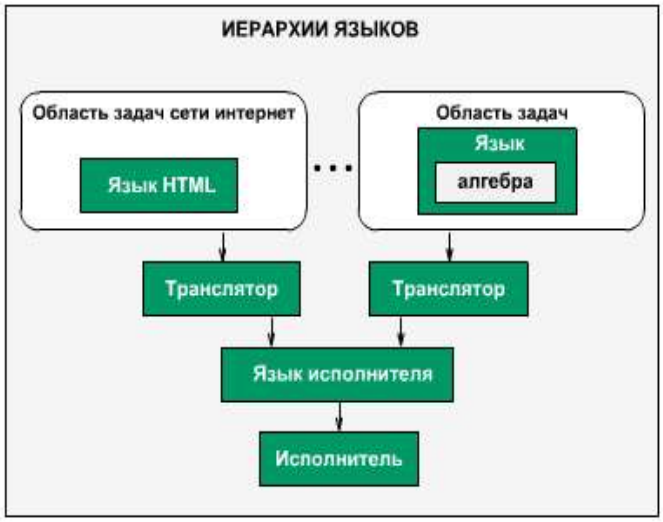
Как же устроен универсальный язык? Казалось бы, сделать простой по составу команд язык, достаточный для представления всех алгоритмов, нельзя. Оказывается, можно, причем удивительно простой язык. Например, для вычислительных математических задач достаточно, чтобы в него входили дроби, четыре обычные арифметические операции над ними, инструкции по разбору условий («если А то Б иначе В») и указатели перехода на инструкцию («теперь инструкция номер n»).

Более того, такой простой язык может быть приспособлен для самого широкого круга задач (нематематические объекты получают числа-обозначения и обрабатываются как числа). Впервые такие языки были построены в первой трети двадцатого века. Использование чисел вместо объектов можно посмотреть на следующем смешном примере.

Однако «совсем маленькие», пусть и универсальные, языки не очень удобны для практического применения. Например, пусть имеется множество постоянно решаемых задач, в которых много вычислений корней (или, скажем, логарифмов). Тогда в указанном «малом» языке каждый раз придется выписывать, как вычислять логарифмы с помощью сложений, умножений и т.д. Гораздо удобнее было бы, чтобы операция извлечения корня (или вычисления логарифма) была в самом языке.

Но если включать все, что может повстречаться на практике, язык может чрезмерно разрастись в объеме и будет труден для изучения теперь уже составителям алгоритмов. Поэтому всегда находят применение «компромиссные» языки – и универсальные, и достаточно богатые средствами для применения, но все же ограниченные в своей громоздкости.

Еще один компромисс делается в том, что под очень важные области применения алгоритмов создаются отдельные языки. Примером такого языка является HTML, разработанный специально для использования в сети Интернет. Его вы будете изучать позднее.



Слайд "Иерархия алгоритмических языков"