Лекция 1

АЛГОРИТМЫ И СПОСОБЫ ИХ ОПИСАНИЯ

Понятие алгоритма

Алгоритм — это точное предписание, которое определяет процесс, ведущий от исходных данных к требуемому конечному результату.

Пример: правила сложения, умножения, решения алгебраических уравнений, умножения матриц и т.п.

К сведению: Слово алгоритм происходит от *algoritmi*, являющегося латинской транслитерацией арабского имени хорезмийского математика IX века **аль-Хорезми**. Благодаря латинскому переводу трактата **аль-Хорезми** европейцы в XII веке познакомились с позиционной системой счисления, и в средневековой Европе алгоритмом называлась десятичная позиционная система счисления и правила счета в ней.

Понятие алгоритма

Применительно к ЭВМ алгоритм определяет вычислительный процесс, начинающийся с обработки некоторой совокупности возможных исходных данных и направленный на получение определенных этими исходными данными результатов. Термин ***вычислительный процесс*** распространяется и на обработку других видов информации, например, символьной, графической или звуковой.

Основные свойства алгоритмов

1. ***Результативность*** означает возможность получения результата после выполнения конечного количества операций.

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

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

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

Задание алгоритма

Для задания алгоритма необходимо описать следующие его элементы:

- набор объектов, составляющих совокупность возможных исходных данных, промежуточных и конечных результатов;
- правило начала;
- правило непосредственной переработки информации (описание последовательности действий);
- правило окончания;
- правило извлечения результатов.

Способы описания алгоритмов

- Словесно - формульный;
- структурный или блок - схемный;
- с помощью графов - схем;
- с помощью сетей Петри.

Словесно – формульный алгоритм

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

Пример: необходимо найти значение следующего выражения: y = 2a – (x+6).

Словесно-формульным способом алгоритм решения этой задачи может быть записан в следующем виде:

1. Ввести значения *a* и *x.*
2. Сложить x и 6.
3. Умножить *a* на 2.
4. Вычесть из *2a* сумму (x+6).
5. Вывести *y* как результат вычисления выражения.

Блок - схемы

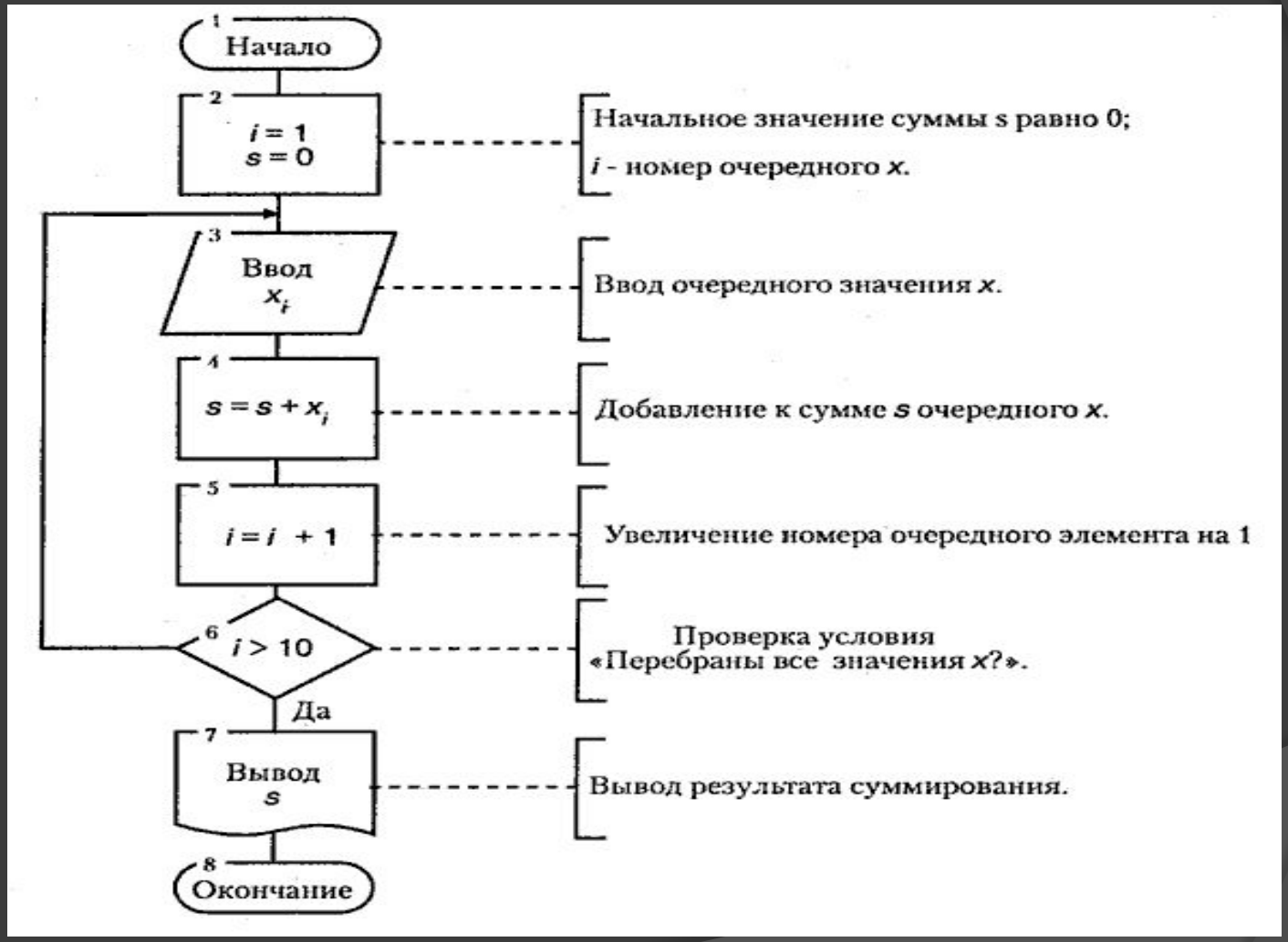
При ***блок - схемном*** описании алгоритм изображается геометрическими фигурами (блоками), связанными по управлению линиями (направлениями потока) со стрелками. В блоках записывается последовательность действий.

Преимущества:

1. наглядность: каждая операция вычислительного процесса изображается отдельной геометрической фигурой.
2. графическое изображение алгоритма наглядно показывает разветвления путей решения задачи в зависимости от различных условий, повторение отдельных этапов вычислительного процесса и Другие детали.

К сведению: Оформление программ должно соответствовать определенным требованиям. В настоящее время действует единая система программной документации (ЕСПД), которая устанавливает правила разработки, оформления программ и программной документации. В ЕСПД определены и правила оформления блок-схем алгоритмов (ГОСТ 10.002-80 ЕСПД, ГОСТ 10.003-80 ЕСПД).

Пример блок - схемы



Алгоритм нахождения суммы 10-ти чисел

Блоки на блок - схемах

Операции обработки данных и носители информации изображаются на схеме соответствующими ***блоками***.

Большая часть блоков по построению условно вписана в прямоугольник со сторонами *a и b.* Минимальное значение ***a* = 10 мм**, увеличение *a* производится на число, кратное **5 мм**. Размер **b=1,5a**. Для от дельных блоков допускается соотношение между *a и b,* равное 1:2. В пределах одной схемы рекомендуется изображать блоки одинаковых размеров. Все блоки нумеруются.

Виды блоков

Наименование	Обозначение	Функции
Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных.
Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод).
Решение		Выбор направления выполнения алгоритма в зависимости от некоторых переменных условий.
Предопределенный процесс		Использование ранее созданных и отдельно написанных программ (подпрограмм).
Документ		Вывод данных на бумажный носитель.

Виды блоков

Наименование	Обозначение	Функции
Магнитный диск		Ввод-вывод данных, носителем которых служит магнитный диск.
Пуск-останов		Начало, конец, прерывание процесса обработки данных.
Соединитель		Указание связи между прерванными линиями, соединяющими блоки.
Межстраничный соединитель		Указание связи между прерванными линиями, соединяющими блоки, расположенные на разных листах.
Комментарий		Связь между элементом схемы и пояснением.

Правила создания блок - схем

1. Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводится параллельно линиям рамки.
2. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз.
3. В блок может входить несколько линий, то есть блок может являться преемником любого числа блоков.
4. Из блока (кроме логического) может выходить только одна линия.
5. Логический блок может иметь в качестве продолжения один из двух блоков, и из него выходят две линии.
6. Если на схеме имеет место слияние линий, то место пересечения выделяется точкой. В случае, когда одна линия подходит к другой и слияние их явно выражено, точку можно не ставить.
7. ***Схему алгоритма*** следует выполнять как единое целое, однако в случае необходимости допускается обрывать линии, соединяющие блоки.

Структурные схемы алгоритмов

- Последовательность двух или более операций;
- выбор направления;
- повторение.

Любой вычислительный процесс может быть представлен как комбинация этих элементарных алгоритмических структур.

Виды алгоритмов

- линейные;
- ветвящиеся;
- циклические.

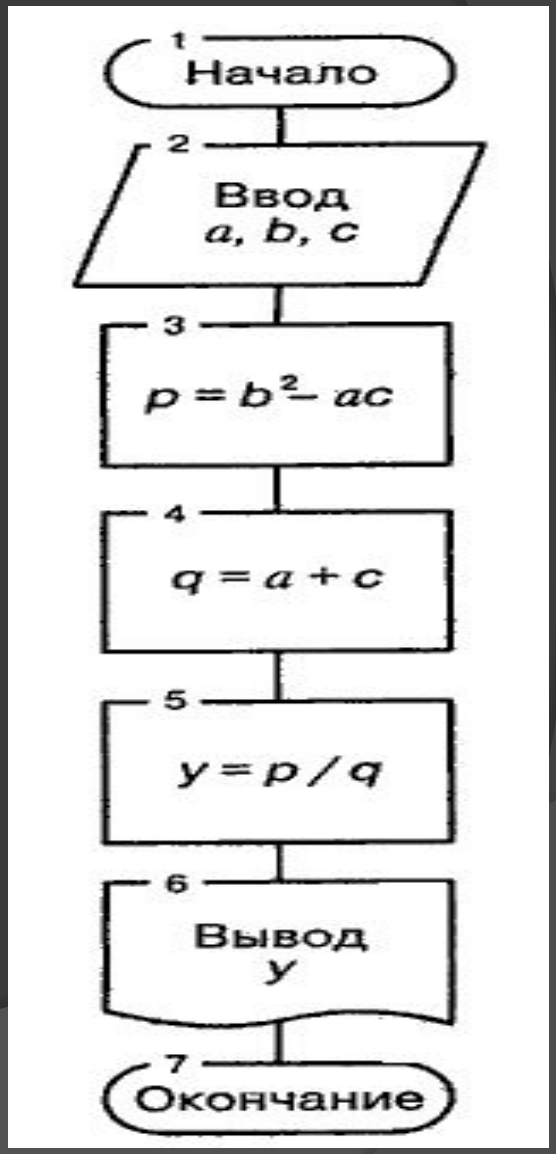
Линейные алгоритмы

В <u>линейном</u> алгоритме операции выполняются последовательно, в порядке их записи. Каждая операция является самостоятельной, независимой от каких-либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности.

Линейные алгоритмы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и над ними выполняются соответствующие условию задачи действия.

Пример линейного алгоритма

Составить блок – схему алгоритма вычисления арифметического выражения $y=(b^2-ac):(a+c)$



Алгоритм с ветвлением

Алгоритм называется ***ветвящимся,*** если для его реализации предусмотрено несколько направлений (ветвей). Каждое отдельное направление алгоритма обработки данных является отдельной ветвью вычислений.

Ветвление в программе — это выбор одной из нескольких последовательностей команд при выполнении программы. Выбор направления зависит от заранее определенного признака, который может относиться к исходным данным, к промежуточным или конечным результатам. Признак характеризует свойство данных и имеет два или более значений.

Ветвящийся процесс, включающий в себя две ветви, называется ***простым***, более двух ветвей — ***сложным***.
Сложный ветвящийся процесс можно представить с помощью простых ветвящихся процессов.

Алгоритм с ветвлением

Направление ветвления выбирается логической проверкой, в результате которой возможны два ответа:

1. «да» — условие выполнено
2. «нет» — условие не выполнено.

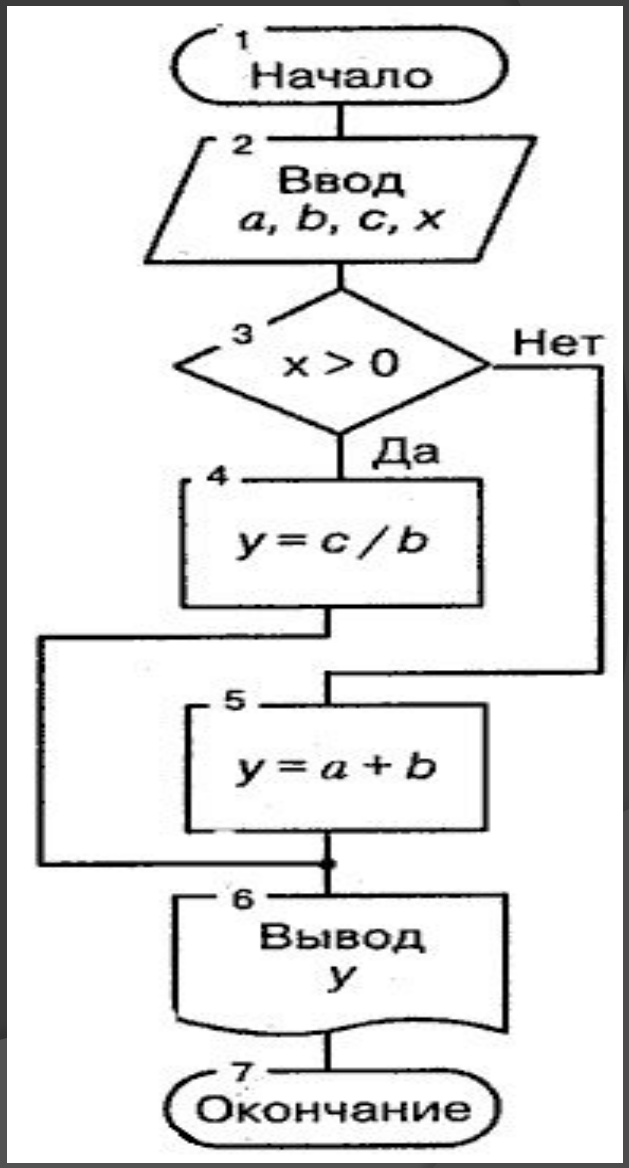
Следует иметь в виду, что, хотя на схеме алгоритма должны быть показаны все возможные направления вычислений в зависимости от выполнения определенного условия (или условий), при однократном прохождении программы процесс реализуется только по одной ветви, а остальные исключаются.

Важно! Любая ветвь, по которой осуществляются вычисления, должна приводить к завершению вычислительного процесса.

Пример алгоритма с ветвлением

Составить блок-схему алгоритма с ветвлением для вычисления следующего выражения:

$$Y = (a+b), \text{ если } X < 0;$$
$$c/b, \text{ если } X > 0.$$



Циклические алгоритмы

Циклическими называются алгоритмы, содержащие циклы.

Цикл — это многократно повторяемый участок алгоритма.

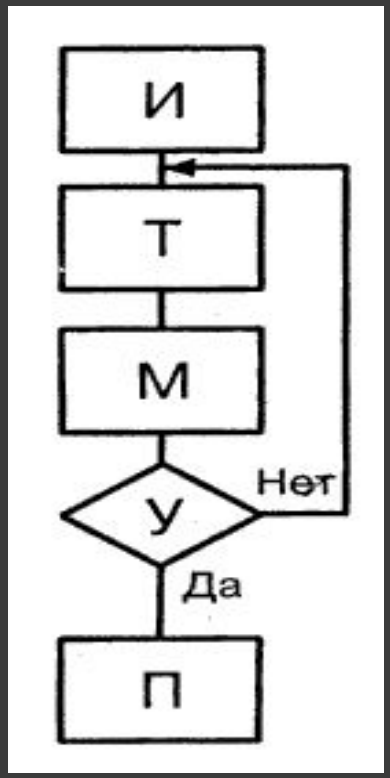
Этапы организации цикла

- подготовка (инициализация) цикла (**И**);
- выполнение вычислений цикла (тело цикла) (**Т**);
- модификация параметров (**М**);
- проверка условия окончания цикла (**У**).

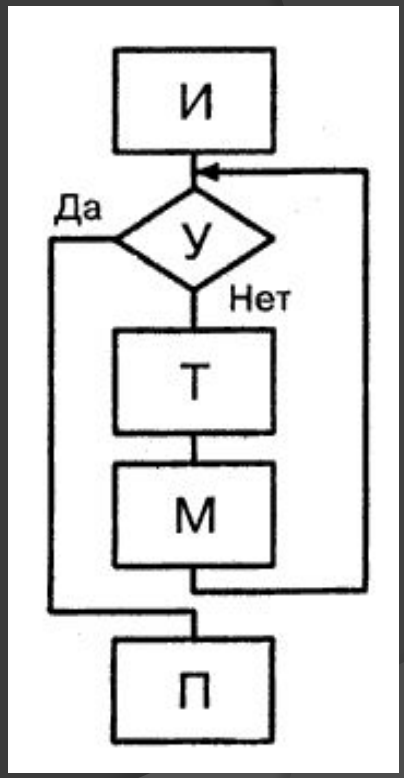
Порядок выполнения этих этапов, например, **Т** и **М**, может изменяться.

Типы циклов

В зависимости от расположения проверки условия окончания цикла различают циклы с нижним и верхним окончаниями.
Для цикла с нижним окончанием (рис. а) тело цикла выполняется как минимум один раз, так как сначала производятся вычисления, а затем проверяется условие выхода из цикла.
В случае цикла с верхним окончанием (рис. б) тело цикла может не выполниться ни разу в случае, если сразу соблюдается условие выхода.



а



б

Примеры циклических алгоритмов

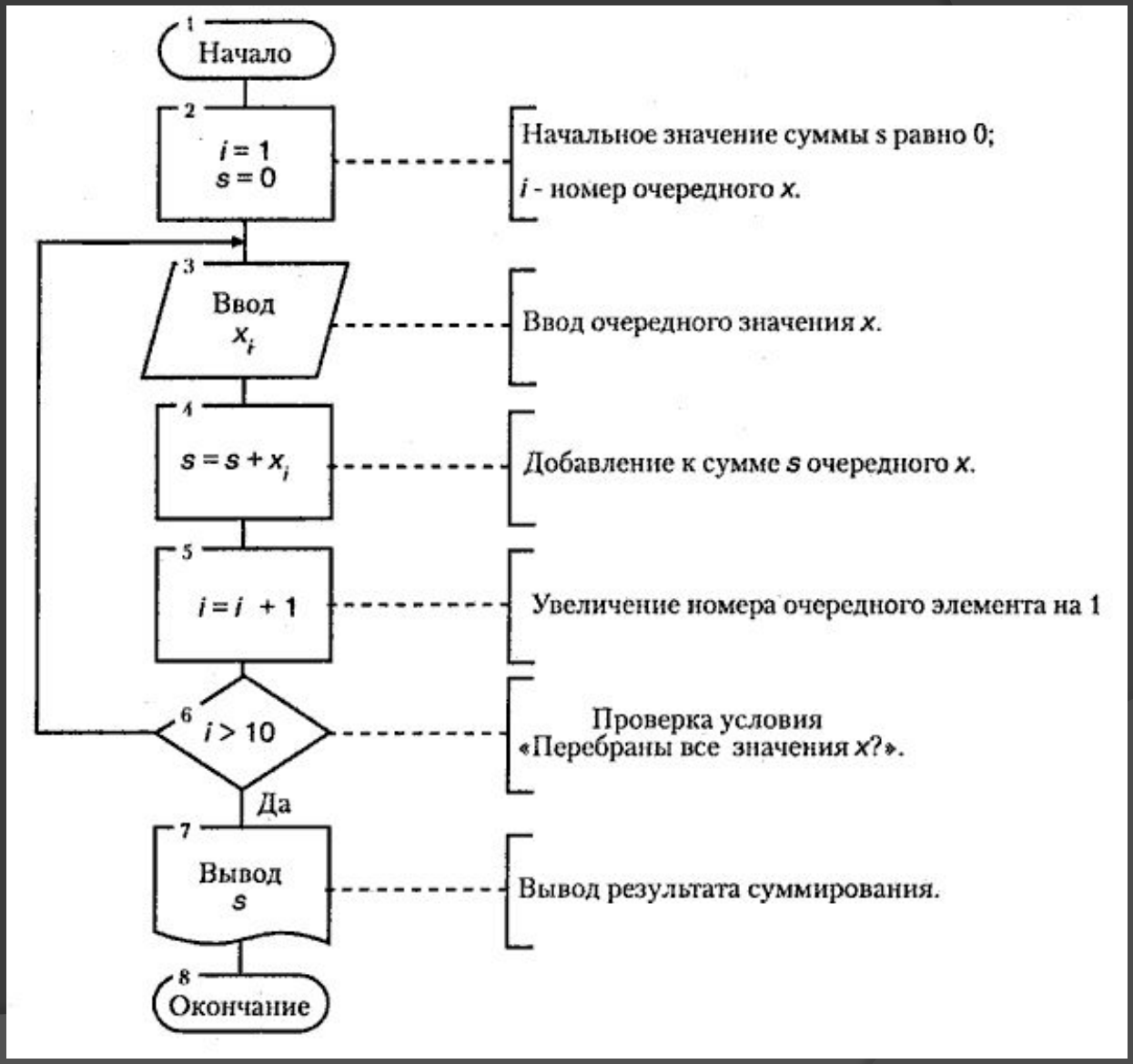
Виды циклов

- Цикл называется ***детерминированным,*** если число повторений тела цикла заранее известно или определено.

- Цикл называется ***итерационным,*** если число повторений тела цикла заранее неизвестно, а зависит от значений параметров (некоторых переменных), участвующих в вычислениях.

Пример циклического алгоритма

Алгоритм нахождения суммы 10-ти чисел



Этапы подготовки и решения задач на ЭВМ

На ЭВМ могут решаться задачи различного характера, например:

научно-инженерные; разработки системного программного обеспечения; обучения; управления производственными процессами и т. д.

В процессе подготовки и решения на ЭВМ *научно -инженерных задач* можно выделить следующие ***этапы:***

1. постановка задачи;
2. математическое описание задачи;
3. выбор и обоснование метода решения;
4. алгоритмизация вычислительного процесса;
5. составление программы;
6. отладка программы;
7. решение задачи на ЭВМ и анализ результатов.

В задачах другого класса некоторые этапы могут отсутствовать, например, в задачах разработки системного программного обеспечения отсутствует математическое описание.

Постановка задачи

На данном этапе формулируется цель решения задачи и подробно описывается ее содержание. Анализируются характер и сущность всех величин, используемых в задаче, и определяются условия, при которых она решается.

Корректность постановки задачи является важным моментом, так как от нее в значительной степени зависят другие этапы.

Математическое описание задачи

Настоящий этап характеризуется математической формализацией задачи, при которой существующие соотношения между величинами, определяющими результат, выражаются посредством математических формул.

Так формируется математическая модель явления с определенной точностью, допущениями и ограничениями. При этом в зависимости от специфики решаемой задачи могут быть использованы различные разделы математики и других дисциплин.

Математическая модель должна удовлетворять по крайней мере двум требованиям: реалистичности и реализуемости. Под ***реалистичностью*** понимается правильное отражение моделью наиболее существенных *черт* исследуемого явления. ***Реализуемость*** достигается *разумной* абстракцией, отвлечением от второстепенных деталей, чтобы свести задачу к проблеме с известным решением. Условием реализуемости является возможность практического выполнения необходимых вычислений за отведенное время при доступных затратах требуемых ресурсов.

Выбор и обоснование метода решения

Модель решения задачи с учетом ее особенностей должна быть доведена до решения при помощи конкретных методов решения. Само по себе математическое описание задачи в большинстве случаев трудно перевести на язык машины. Выбор и использование метода решения задачи позволяет привести решение задачи к конкретным машинным операциям. При обосновании выбора метода необходимо учитывать различные факторы и условия, в том числе точность вычислений, время решения задачи на ЭВМ, требуемый объем памяти и другие.

Одну и ту же задачу можно решить различными методами, при этом в рамках каждого метода можно составить различные алгоритмы.

Алгоритмизация

На данном этапе составляется алгоритм решения задачи согласно действиям, задаваемым выбранным методом решения. Процесс обработки данных разбивается на отдельные относительно самостоятельные блоки, и устанавливается последовательность выполнения блоков. Разрабатывается блок-схема алгоритма.

Составление программы

При составлении программы алгоритм решения задачи переводится на конкретный язык программирования.

Для программирования обычно используются языки высокого уровня, поэтому составленная программа требует перевода ее на машинный язык ЭВМ. После такого перевода выполняется уже соответствующая машинная программа.

Отладка программы

Отладка заключается в поиске и устранении синтаксических и логических ошибок в программе.

В ходе *синтаксического контроля* программы транслятором выявляются конструкции и сочетания символов, недопустимые с точки зрения правил их построения или написания, принятых в данном языке. Сообщения об ошибках ЭВМ выдает программисту, при этом вид и форма выдачи подобных сообщений зависят от вида языка и версии используемого транслятора.
После устранения синтаксических ошибок проверяется *логика работы программы* в процессе ее выполнения с конкретными исходными данными. Для этого используются специальные методы, например, в программе выбираются контрольные точки, для которых вручную рассчитываются промежуточные результаты. Эти результаты сверяются со значениями, получаемыми ЭВМ в данных точках при выполнении отлаживаемой программы.
Для поиска ошибок могут быть использованы *отладчики*, выполняющие специальные действия на этапе отладки, например, удаление, замена или вставка отдельных операторов или целых фрагментов программы, вывод или изменение значений заданных переменных.

Решение задачи на ЭВМ и анализ результатов

После отладки программы ее можно использовать для *решения прикладной задачи*. При этом обычно выполняется многократное решение задачи на ЭВМ для различных наборов исходных данных. Получаемые результаты интерпретируются и анализируются специалистом или пользователем, поставившим задачу.

Разработанная программа длительного использования устанавливается на ЭВМ, как правило, в виде готовой к выполнению машинной программы. К программе прилагается документация, включая инструкцию для пользователя.

Чаще всего при установке программы на диск для ее последующего использования помимо файлов с исполняемым кодом устанавливаются различные вспомогательные программы (утилиты, справочники, настройщики и т. д.), а также необходимые для работы программ разного рода файлы с текстовой, графической, звуковой и другой информацией.

Компиляция и интерпретация программ

ЭВМ непосредственно выполняет программы на ***машинном языке*** программирования данной ЭВМ.

Программа представляет собой набор отдельных команд компьютера. Эти команды являются достаточно «простыми», например, сложение, умножение, сравнение или пересылка отдельных данных.

Каждая команда содержит в себе сведения о том, какая операция должна быть выполнена (код операции), с какими операндами (адреса данных или непосредственно сами данные) выполняются вычисления и куда (адрес) должен быть помещен результат.

Компиляция и интерпретация программ

Машинные языки были первыми языками программирования.

Программирование на них затруднительно ввиду того, что:

1. эти языки различны для каждого типа ЭВМ,
2. являются трудоемкими для большинства пользователей по причине необходимости знания особенностей конкретной ЭВМ и большого количества реализуемых ею операций (команд).

Данные языки обычно используются для разработки системных программ, при этом чаще всего применяются специальные символические языки — Ассемблеры, близкие к соответствующим машинным языкам.

Компиляция и интерпретация программ

Человеку свойственно формулировать и решать задачи в выражениях более общего характера, чем команды ЭВМ. Поэтому с развитием программирования появились языки, ориентированные на более ***высокий уровень*** абстракции при описании решаемой на ЭВМ задачи.

Эти языки получили название ***языков высокого уровня***. Их теоретическую основу составляют алгоритмические языки, например, Паскаль, Си, Бейсик, Фортран, PL/1.

Виды процессоров

Для перевода программы, написанной на языке высокого уровня, в соответствующую машинную программу используются ***языковые процессоры***.

Различают два вида языковых процессоров:

1. интерпретаторы
2. трансляторы.

Интерпретатор

Интерпретатор — это программа, которая получает исходную программу и по мере распознавания конструкций входного языка реализует действия, описываемые этими конструкциями.

Транслятор

Транслятор — это программа, которая принимает исходную программу и порождает на своем выходе программу, записываемую на объектном языке программирования (объектную программу).

В частном случае объектным может служит машинный язык, и в этом случае полученную на выходе транслятора программу можно сразу же выполнить на ЭВМ. В общем случае объектный язык необязательно должен быть машинным или близким к нему (автокодом). В качестве объектного языка может служить и некоторый промежуточный язык.

Транслятор с языка высокого уровня называют ***компилятором***.

Стили программирования

Одним из важнейших признаков классификации языков программирования является принадлежность их к одному из стилей, основными из которых являются следующие:

1. процедурный,
2. функциональный,
3. логический и
4. объектно-ориентированный.

Процедурное программирование

Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 40-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор ***присваивания***, служащий для изменения содержимого областей памяти. Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.

Выполнение программы сводится к ***последовательному выполнению операторов*** с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

Процедурное программирование

Процедурные языки характеризуются следующими особенностями:

- необходимостью явного управления памятью, в частности, описанием переменных;
- малой пригодностью для символьных вычислений;
- отсутствием строгой математической основы;
- высокой эффективностью реализации на традиционных ЭВМ.

Одним из важнейших классификационных признаков процедурного языка является его уровень. Уровень языка определяется семантической (смысловой) емкостью его конструкций и степенью его ориентации на программиста.

Язык программирования частично ликвидирует разрыв между методами решения различного рода задач человеком и вычислительной машиной.

Чем более язык ориентирован на человека, тем выше его уровень.

Языки программирования

Basic (Бэйсик) (Beginners All-purpose Symbolic Instruction Code) — многоцелевой язык символических инструкций для начинающих) представляет собой простой язык программирования, разработанный в 1964 году для использования новичками. Он был разработан как простейший язык для непосредственного общения человека с ЭВМ. Поэтому первоначально работа велась в интерактивном режиме с использованием интерпретаторов. В настоящее время для этого языка имеются также и компиляторы.

Согласно концепциям, заложенным в Basic, этот язык в смысле строгости и стройности является антиподом языка Pascal. В частности, в нем широко распространены различные правила умолчания, что считается плохим тоном в большинстве языков программирования подобного типа.

Basic широко распространен на ЭВМ различных типов и очень популярен в среде программистов, особенно начинающих. Существует множество версий этого языка, мало совместимых между собой. Basic активно поглощает многие концепции и новинки из других языков. Поэтому он достаточно динамичен, и нельзя однозначно определить его уровень.

Языки программирования

Pascal (Паскаль) является одним из наиболее популярных среди прикладных программистов процедурным языком программирования, особенно для ПЭВМ.

Разработанный в 1970 году швейцарским специалистом в области вычислительной техники профессором Н. Виртом, язык назван в честь французского математика и по замыслу автора предназначался для обучения программированию. Однако язык получился настолько удачным, что стал одним из основных инструментов прикладных и системных программистов при решении задач вычислительного и информационно-логического характера. В 1979 году был подготовлен проект описания языка — Британский стандарт языка программирования Pascal BS6192, который стал также и международным стандартом ISO 7185.

Языки программирования

В Pascal реализован ряд концепций, рассматриваемых как основа «дисциплинированного» программирования и заимствованных впоследствии разработчиками многих языков. Одним из существенных признаков Pascal является последовательная и достаточно полная реализация концепции структурного программирования. Причем это осуществляется не только путем упорядочивания связей между фрагментами программы по управлению, но и за счет структуризации данных. В языке реализована концепция определения новых типов данных на основе уже имеющихся. Этот язык, в отличие от языка С, является строго типизированным.

Pascal характеризуется:

1. высоким уровнем;
2. широкими возможностями;
3. стройностью, простотой и краткостью;
4. строгостью, способствующей написанию эффективных и надежных программ;
5. высокой эффективностью реализации на ЭВМ.

Pascal реализован на ЭВМ различных типов, но наиболее распространен и развит для ПЭВМ. В настоящее время широко используются такие версии этого языка для ПЭВМ, как Borland Pascal и Turbo Pascal.

Функциональное программирование

Сущность функционального (аппликативного) программирования определена А. П. Ершовым как «... способ составления программ, в которых единственным действием является вызов функции, единственным способом расчленения программы на части является введение имени для функции, а единственным правилом композиции — оператор суперпозиции функции. Никаких ячеек памяти, ни операторов присваивания, ни циклов, ни, тем более, блок-схем, ни передачи управления».

Роль основной конструкции в функциональных языках играет выражение: К выражениям относятся скалярные константы, структурированные объекты, функции, тела функций и вызовы функций. Функция трактуется как однозначное отображение из X в *X,* где *X* — множество выражений.

Программа представляет собой совокупность описаний функций и выражения, которые необходимо вычислить.

Функциональное программирование

Функциональное программирование не использует концепцию памяти как хранилища значений переменных. Операторы присваивания отсутствуют, вследствие чего переменные обозначают не области памяти, а объекты программы, что полностью соответствует понятию переменной в математике. В принципе, можно составлять программы и вообще без переменных. Кроме того, нет существенных различий между константами и функциями, то есть между программами и данными. В результате этого функция может быть значением вызова другой функции и может быть элементом структурированного объекта. Число аргументов при вызове функции не обязательно должно совпадать с числом параметров, указанных при ее описании. Перечисленные свойства характеризуют аппликативные языки как языки программирования очень высокого уровня.

Первым таким языком был ***LISP(Лисп)*** (LISt Processing — обработка списков), созданный в 1959 году. Цель его создания состояла в организации удобства обработки символьной информации. Существенная черта этого языка — унификация программных структур и структур данных: все выражения записываются в виде списков.

Логическое программирование

Логическое, или реляционное программирование открыло появление языка ***PROLOG (Пролог)*** (PROgramming in LOGic — программирование в терминах логики). Этот язык был создан французским ученым А. Кольмероэ в 1973 году.

Языки логического программирования используются в системах искусственного интеллекта.

Программа представляет собой совокупность определений отношений между объектами (в терминах условий или ограничений) и цели (запроса). Процесс выполнения программы трактуется как процесс общезначимости логической формулы, построен- ной из программы по правилам, установленным семантикой используемого языка. Результат вычисления является побочным продуктом этого процесса. В реляционном программировании нужно только специфицировать факты, на которых алгоритм основывается, а не определять последовательность шагов, которые требуется выполнить.

Формула Р. Ковальского: «алгоритм = логика + управление»

Логическое программирование

Языки логического программирования характеризуются:

- высоким уровнем;
- строгой ориентацией на символьные вычисления;
- возможностью инверсных вычислений, то есть переменные в процедурах не делятся на входные и выходные;
- возможной логической неполнотой, поскольку зачастую невозможно выразить в программе определенные логические соотношения, а также невозможно получить из программы все выводы правильные.

Логические программы имеют небольшое быстродействие, так как вычисления осуществляются методом проб и ошибок, поиском с возвратами к предыдущим шагам.

Объектно-ориентированное программирование

Прототипом объектно-ориентированного программирования послужил ряд средств, входящих в состав языка SIMULA-67. Но в самостоятельный стиль оно оформилось с появлением языка SMALLTALK, разработанного А. Кеем в 1972 году и первоначально предназначенного для реализации функций машинной графики.

В основе объектно-ориентированного стиля программирования лежит понятие объекта, а суть его выражается формулой:

«объект - данные + процедуры».

Каждый объект интегрирует в себе некоторую структуру данных и доступные только ему процедуры обработки этих данных, называемые ***методами***. Объединение данных и процедур в одном объекте называется ***инкапсуляцией*** и присуще объектно-ориентированному программированию.

Объектно-ориентированное программирование

Для описания объектов служат классы. ***Класс*** определяет свойства и методы объекта, принадлежащего этому классу. Соответственно, любой объект можно определить как ***экземпляр*** класса.

Программирование рассматриваемого стиля заключается в выборе имеющихся или создании новых объектов и организации взаимодействия между ними. При создании новых объектов свойства объектов могут добавляться или ***наследоваться*** от объектов-предков. В процессе работы с объектами допускается ***полиморфизм*** — возможность использования методов с одинаковыми именами для обработки данных разных типов.

К наиболее современным объектно-ориентированным языкам программирования относятся C++ и Java.

Объектно-ориентированное программирование

В силу своей конструктивности идеи объектно-ориентированного программирования используются во многих универсальных процедурных языках. Так, например, в состав интегрированной системы программирования на языке Borland PASCAL в. 5.5 входит специальная библиотека объектно-ориентированного программирования Turbo Vision.

В последнее время многие программы, в особенности объектно-ориентированные, реализуются как ***системы визуального программирования.***

Отличительной особенностью таких систем является мощная среда разработки программ из готовых «строительных блоков», позволяющая создать интерфейсную часть программного продукта в диалоговом режиме, практически без кодирования программных операций.

К числу объектно-ориентированных систем визуального программирования относятся: Visual Basic, Delphi, C++Builder и Visual C++.