Лекция №6

Введение в понятие алгоритма
Классификация языков программирования

Алгоритмизация - важнейший этап в процессе решения задач на ПК. Но понятие алгоритма возникло задолго до появления ПК. Слово ***«алгоритм»*** произошло от имени среднеазиатского математика аль- Хорезми (IX в) и использовалось вначале только в математике.

НЕМНОГО ИСТОРИИ

Аль-Хорезми (786 —850 гг. н.э.)

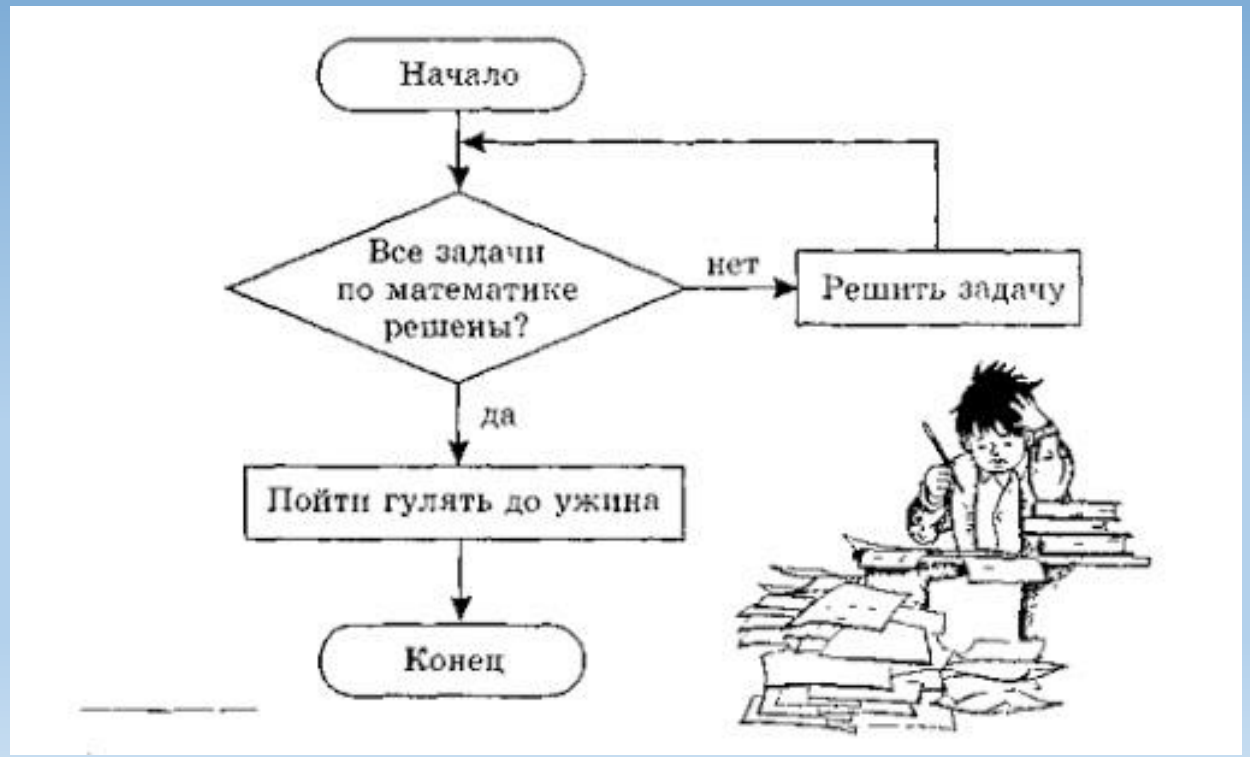
Абу Джафар Мухаммад ибн Муса аль-Хорезми, который родился приблизительно в 786 г. в г. Хива Хорезмской области Узбекистана.

Слово ***алгоритм*** - европеизированное произношение слов ***аль-Хорезми***.

Первоначально под словом алгоритм понимали способ выполнения арифметических операций над десятичными числами. В дальнейшем это понятие стали использовать для обозначения любой последовательности действий, приводящей к решению поставленной задачи.

Алгоритм – это понятное и точное предписание (указание) исполнителю совершить определенную последовательность действий над исходной информацией для решения поставленной задачи.

Сейчас понятие алгоритма используется на только в математике и программировании, но и в других областях.



Алгоритм обладает **пятью важнейшими свойствами**:

1) **Дискретность**. Это свойство означает, то что алгоритм должен быть разбит на отдельные достаточно простые действия, причем выполнение каждого шага начинается после завершения предыдущего.

2) **Определенность**. Однозначность выполнения каждого шага. Алгоритм не должен допускать произвольной трактовки исполнителем.

3) **Результативность (выполнимость)**. Алгоритм должен предоставлять возможность получения решения за конечное число шагов.

4) **Массовость**. Это пригодность для решения многих или даже всех задач данного типа при различных исходных данных.

5) **Инвариантность**. Алгоритм должен быть составлен таким образом, чтобы он, в идеальном случае, мог быть выполнен разными исполнителями: разными ПК, в разных средах, разными

Способы записи алгоритмов

Словесная запись ориентирована на исполнителя-человека. При таком способе команды записываются на естественном языке и нумеруются.

Например, рассмотрим алгоритм Евклида для поиска наибольшего общего делителя.

1) Задать два числа.

2) Если числа равны, то ответ равен одному из чисел, иначе перейти к п. 3.

3) Определить большее из двух чисел.

4) Заменить большее число на разность большего и меньшего чисел.

5) Перейти к п. 2.

Основное достоинство такого способа представления – понятность.

Недостатки: неоднозначность, избыточность, отсутствие наглядности связей.

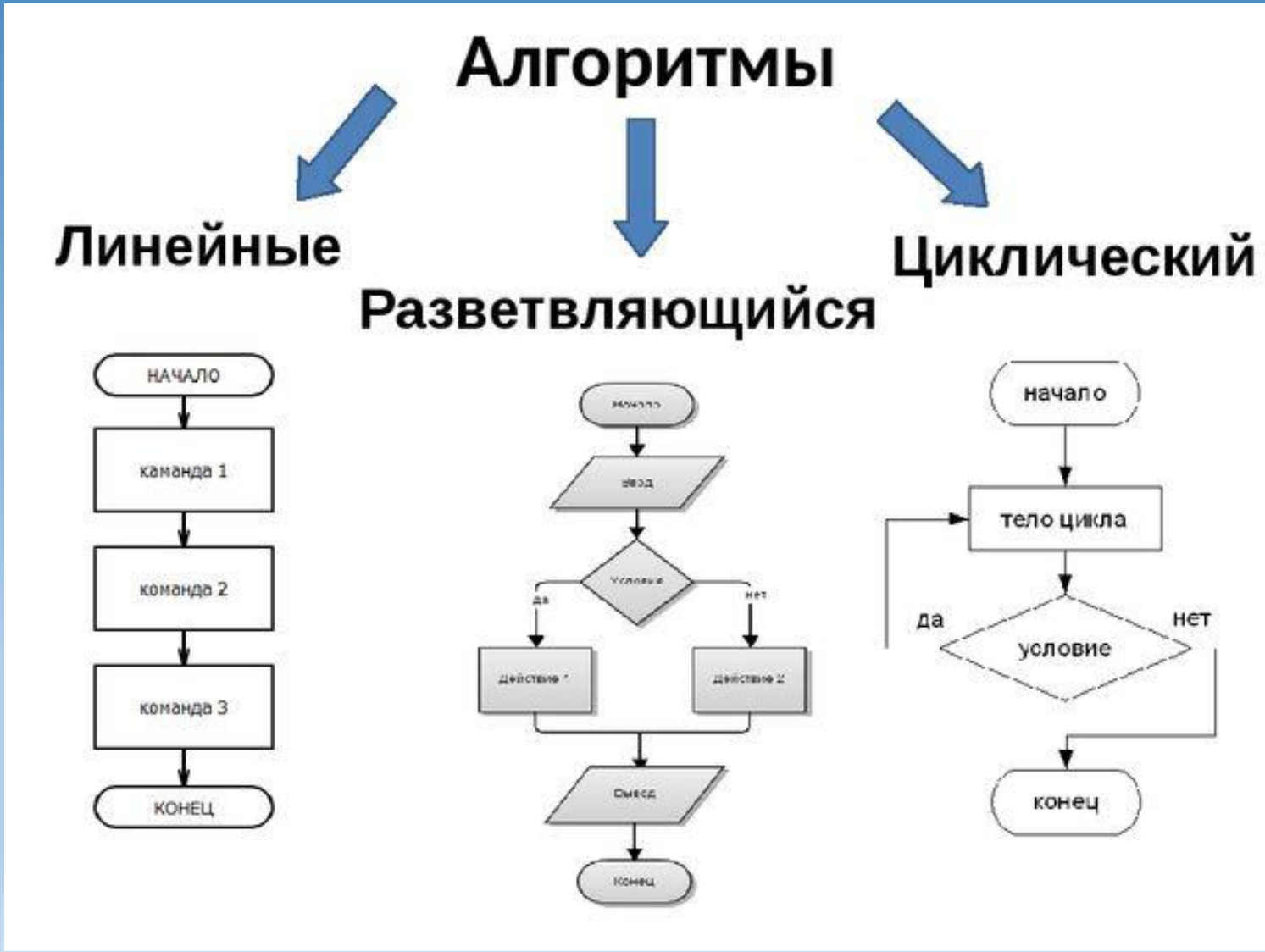
Схемы программ

Это графическое отображение алгоритма согласно утвержденным стандартам. При таком способе каждое действие записывается внутри блоков. Блоки соединяются линиями, которые указывают последовательность действий. Линии могут оканчиваться стрелками, но обычно стрелки не ставят, если линии отражают естественную последовательность действий. Естественная последовательность: сверху вниз и слева направо. В противном случае стрелки обязательны.

Достоинство представления алгоритмов в виде схем программ - наглядность.

Недостаток - большая трудоемкость выполнения.

Виды графических алгоритмов



Языки программирования

Во всех перечисленных способах представления алгоритмов допускается определенная свобода, они ориентированы на человека. На практике же основной исполнитель алгоритмов – это компьютер. Для компьютера запись алгоритма должна быть абсолютно точна, т.е. язык для записи алгоритмов должен быть формализован. Такой язык называется языком программирования, а запись на таком языке называется программой.

Классификация языков программирования

На заре компьютерной эры машинный код был единственным средством общения человека с компьютером. Огромным достижением создателей языков программирования было то, что они сумели заставить сам компьютер работать переводчиком с этих языков на машинный код.

Существующие языки программирования можно разделить на две группы:

- **процедурные**
- **непроцедурные.**


Языки программирования
Процедурные
Непроцедурные
Низкого уровня (машинно-зависимые)
Высокого уровня (машинно-независимые)
Объектно-ориентированные
Декларативные
Ассемблер
Фортран, Бейсик, Паскаль, Си (машинно-ориентированный)
Си++, Visual Basic, Delphi, Java
Логические
Функцио-нальные
Пролог
Лисп

Процедурные (алгоритмические) программы

Процедурные (или алгоритмические) программы представляют собой систему предписаний для решения конкретной задачи. Роль компьютера сводится к механическому выполнению этих предписаний.

Процедурные языки разделяют на языки ***низкого*** и ***высокого*** уровня.

Языки программирования низкого уровня

Языки низкого уровня ориентировались на определенный тип процессора и учитывали его особенности, поэтому для того, чтобы перенести программу, написанную на ассемблере, на другую аппаратную платформу её нужно было почти полностью переписать. Различия присутствовали также и в синтаксисе программ под разные компиляторы.

Языками низкого уровня пользуются преимущественно для написания небольших системных программ, драйверов устройств, модулей стыков с нестандартным оборудованием, программирования специализированных микропроцессоров, когда немаловажным является компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

Языки программирования высокого уровня

Языки программирования высокого уровня значительно ближе и понятнее человеку, нежели компьютеру. Особенности конкретных компьютерных архитектур в них не учитываются, поэтому создаваемые программы на уровне исходных текстов легко переносимы на другие платформы, для которых создан транслятор этого языка. Разрабатывать программы на языках высокого уровня с помощью понятных и мощных команд значительно проще, а ошибок при создании программ допускается гораздо меньше.

Основное достоинство алгоритмических языков высокого уровня - возможность описания программ решения задач в форме, максимально удобной для восприятия человеком. Но так как каждое семейство ПК имеет свой собственный, специфический внутренний (машинный) язык и может выполнять лишь те команды, которые записаны на этом языке, то для перевода исходных программ на машинный язык используются специальные программы-***трансляторы***.

Работа всех трансляторов строится по одному из двух принципов: ***интерпретация*** или ***компиляция***.

Интерпретация подразумевает пооператорную трансляцию и последующее выполнение оттранслированного оператора исходной программы. В связи с этим можно отметить два недостатка метода интерпретации:

- во-первых, интерпретирующая программа должна находиться в памяти ПК в течение всего процесса выполнения исходной программы, т. е. занимать определенный объем памяти;
- во-вторых, процесс трансляции одного и того же оператора повторяется столько раз, сколько раз должна исполняться эта команда в программе, что резко снижает производительность работы программы.

Несмотря на указанные недостатки, ***трансляторы-интерпретаторы*** получили достаточное распространение, так как они удобны при разработке и отладке исходных программ.

При ***компиляции*** процессы трансляции и выполнения разделены во времени: сначала исходная программа полностью переводится на машинный язык (после чего наличие транслятора в оперативной памяти становится ненужным), а затем оттранслированная программа может многократно исполняться. Следовательно, для одной и той же программы трансляция методом компиляции обеспечивает более высокую производительность вычислительной системы при сокращении требуемой оперативной памяти.

Большая сложность в разработке компилятора по сравнению с интерпретатором с того же самого языка объясняется тем, что компиляция программы включает два действия:

- **анализ**, т. е. определение правильности записи исходной программы в соответствии с правилами построения языковых конструкций входного языка,
- **синтез** - генерирование эквивалентной программы в машинных кодах.

Трансляция методом компиляции требует неоднократного «просмотра» транслируемой программы, т. е. *трансляторы-компиляторы* являются многопроходными: при первом проходе они проверяют корректность синтаксиса языковых конструкций отдельных операторов независимо друг от друга, при последующем проходе - корректность синтаксических взаимосвязей между операторами и т. д.


Трансляция
Компиляция
Интерпретация
Исходный текст программы
Компилятор, компоновщик
Программа на машинном языке
Выполнение программы
Оператор на исходном языке
Интерпре-татор
Команды на машинном языке
Выполнение команд
MyShared

Наряду с рассмотренными выше трансляторами-интерпретаторами и трансляторами-компиляторами на практике используются также трансляторы ***интерпретаторы-компиляторы***, которые объединяют в себе достоинства обоих принципов трансляции: на этапе разработки и отладки программ транслятор работает в режиме интерпретатора, а после завершения процесса отладки исходная программа повторно транслируется в ***объектный модуль*** (т. е. уже методом компиляции). Это позволяет значительно упростить и ускорить процесс составления и отладки программ, а за счет последующего получения объектного модуля обеспечить более эффективное исполнение программы.

Классическое процедурное программирование требует от программиста детального описания того, как решать задачу, т. е. формулировки алгоритма и его специальной записи. При этом ожидаемые свойства результата обычно не указываются. Основные понятия языков этих групп - оператор и данные. При процедурном подходе операторы объединяются в группы - процедуры. Структурное программирование в целом не выходит за рамки этого направления, оно лишь дополнительно фиксирует некоторые полезные приемы технологии программирования.

Непроцедурное программирование

Принципиально иное направление в программировании связано с методологиями (иногда говорят «парадигмами») ***непроцедурного программирования***. К ним можно отнести ***объектно-ориентированное*** и ***декларативное программирование***.

Объектно-ориентированный язык создает окружение в виде множества независимых объектов. Каждый объект ведет себя подобно отдельному компьютеру, их можно использовать для решения задач как «черные ящики», не вникая во внутренние механизмы их функционирования. Из языков объектного программирования, популярных среди профессионалов, следует назвать прежде всего Си-подобные языки программирования.

При использовании ***декларативного языка*** программист указывает исходные информационные структуры, взаимосвязи между ними и то, какими свойствами должен обладать результат. При этом процедуру его получения («алгоритм») программист не строит (по крайней мере, в идеале). В этих языках отсутствует понятие «оператор» («команда»). Декларативные языки можно подразделить на два семейства - ***логические*** (типичный представитель - Prolog) и ***функциональные*** (Lisp).

Функциональные языки являются языками искусственного интеллекта. Программа, написанная на функциональном языке, состоит из последовательности функций и выражений, которые необходимо вычислить. Основной структурой данных является связный список.

Логические языки, ориентированные на решение задач без описания алгоритмов, языки искусственного интеллекта.

Спасибо за внимание!