

Технология программирования

Блок 1. Основы систем программирования

Тема 1. Основы алгоритмизации и программирования. Системы программирования. Виды языков программирования

Цели изучения темы:

- ознакомить студентов с основными понятиями и механизмами процесса алгоритмизации;
- сформировать базу знаний о языках программирования, их видах, особенностях и сферах применения.

Задачи изучения темы:

- сформировать знания о принципах построения наиболее часто встречаемых алгоритмов.

В результате изучения данной темы Вы будете знать:

- основные типы алгоритмов и их использование для решения вычислительных, инженерных задач;
- методы проектирования и производства программного продукта, и средства описания алгоритмов;
- основные методологии программирования.

уметь:

- использовать рабочую среду программирования для выбора эффективного и современного метода решения задач;
- классифицировать языки программирования по видам и механизму работы.

владеть:

- основами алгоритмизации и программирования.

Учебные вопросы темы:

1. Понятие алгоритма
2. Системы программирования

Основные термины и понятия, которые Вам предстоит изучить:

алгоритмизация, алгоритм, дискретность, детерминированность, псевдокод, машинный язык, графический алгоритм.

Вопрос 1. Понятие алгоритма

Практически все в нашем мире подчиняется каким-то законам и правилам. Современная наука не стоит на месте, благодаря чему человечеству известна масса формул и алгоритмов, следуя которым, можно рассчитать и воссоздать множество действий и строений, созданных природой, и воплотить в жизнь идеи, придуманные человеком

Большинство действий, которые мы выполняем в течение своей жизни, требуют соблюдения ряда правил. От того, насколько верное представление имеет человек о том что, как и в какой последовательности он должен сделать, зависит качество и результат выполнения поставленных перед ним задач

Алгоритмизация – это процесс создания алгоритма решения задачи.

Алгоритм – это точное и понятное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи. То бишь определяющее вычислительный процесс, ведущий от варьируемых исходных данных к искомому результату.

В алгоритме обязательно должны быть предусмотрены все ситуации, которые могут возникнуть в процессе решения комплекса задач. Само слово алгоритм происходит от имени арабского математика аль Хорезми. Его полное имя было Абу Абдуллах (или Абу Джафар) Мухаммад ибн Муса аль Хорезми. Он использовал индийскую позиционную систему счисления с нулем и в своих трудах сформулировал правила четырех арифметических действий над многозначными числами. Эти действия и стали впоследствии называть алгоритмами

Создание алгоритма требует творческого подхода, поэтому новый список последовательных действий может создать только живое существо. А вот для исполнения уже существующих инструкций фантазию иметь не обязательно, с этим справится даже бездушная техника.

Отличным примером точного исполнения заданной инструкции является пустая микроволновая печь, которая продолжает работать, несмотря на отсутствие пищи внутри нее.

Субъект или объект, которому не обязательно вникать в суть алгоритма, называется формальным исполнителем. Человек тоже может стать формальным исполнителем, однако в случае нерентабельности того или иного действия мыслящий исполнитель может все сделать по-своему. Поэтому основными исполнителями являются компьютеры, микроволновые печи, телефоны и другая техника. Понятие алгоритма в информатике имеет самое важное значение. Каждый алгоритм составляется с расчетом на конкретного

субъекта, с учетом допустимых действий. Те объекты, к которым субъект может применить инструкции, составляют среду исполнителя. Составляя алгоритм для какого-нибудь автоматического устройства, мы должны принимать во внимание следующее:

- **Первое** – это кто исполнитель, тот, кто будет работать по нашему алгоритму. Какие действия он может выполнять, какие команды и в каком виде мы можем ему отдавать. Исполнителем, в принципе, может быть кто или что угодно: человек, компьютер, станок с ЧПУ, автоматический луноход и т. д.
- **Второе** – это как скомбинировать команды, чтобы исполнитель сделал то, что от него требуется. Это самый трудный этап в составлении алгоритма, использовать набор команд, понятных исполнителю, какие действия и в каком порядке выполнять.
- **Третье** – это форма записи алгоритма. Человек понимает смысл сказанной фразы по множеству неуловимых оттенков и интонаций. Одну и ту же мысль мы можем выразить различными словами. Компьютер же понимает алгоритм буквально («Хозяин, тебе дрова не нужны?»), для него каждая конкретная команда всегда имеет один и тот же смысл. Поэтому алгоритмы для компьютеров записываются по строгим, заранее определенным правилам, чтобы их нельзя было истолковать по-разному.

Исполнителем алгоритма предстает некоторая абстрактная или реальная (техническая, биологическая или биотехническая) система, которая способна выполнить действия, предписываемые алгоритмом.

Для характеристики исполнителя используют несколько понятий:

- среда;
- система команд;
- элементарные действия;
- отказы.

Среда (или обстановка) представляет собой «место обитания» исполнителя.

Любой из исполнителей может выполнять команды только из некоторого строго заданного списка, который является *системой команд* исполнителя. Для каждой команды задаются условия применимости (в каких состояниях среды может быть выполнена команда) и приводятся результаты выполнения команды.

После вызова команды исполнитель производит соответствующее *элементарное действие*.

Может возникнуть и *отказ* исполнителя в случае, если команда вызывается при недопустимом для нее состоянии среды. Чаще всего исполнитель ничего не знает о цели алгоритма. Он выполняет все предложенные ему действия, не задавая вопросов «почему» и «зачем».

В информатике универсальным исполнителем алгоритмов является компьютер.

К **основным свойствам алгоритмов** относятся:

- 1) **понятность** для исполнителя – исполнитель алгоритма должен знать, как его выполнять;
- 2) **дискретность** (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное исполнение простых (или ранее определенных) шагов (этапов); т.е. обязательна возможность разбиения алгоритма на отдельные элементарные действия, которые можно реализовать на ЭВМ и результат их выполнения определен и понятен
- 3) **детерминированность** (определенность) – каждое правило алгоритма должно быть четким, однозначным и не оставлять места для ее неоднозначного толкования и неопределенного исполнения. Это свойство обеспечивает выполнение алгоритма механически, не требуя никаких дополнительных указаний или сведений о решаемой задаче;
- 4) **результативность** (или конечность) – алгоритм должен приводить к решению задачи за конечное число шагов; т.е. для любого допустимого набора исходных данных он должен через определенное число шагов завершить работу.
- 5) **массовость** – алгоритм решения задачи производится в общем виде, т.е. его можно будет применять для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из определенной области, которая называется областью применимости алгоритма. Хотя есть мнение, что последний пункт относится не к алгоритмам, а ко всем математическим методам в целом.

Часто в школах, чтобы дать детям более понятное описание алгоритмов, учителя приводят в пример приготовление пищи по кулинарной книге, изготовление лекарства по рецепту или процесс мыловарения на основе мастер-класса. Однако, учитывая второе свойство алгоритма, в котором говорится о том, что каждый пункт алгоритма должен быть настолько понятным, чтобы его мог выполнить абсолютно любой человек и даже машина, можно прийти к выводу что любой процесс, требующий проявления хоть какой-то фантазии, алгоритмом назвать нельзя. А готовка и рукоделие требуют определенных навыков и хорошо развитого воображения.

Алгоритм должен давать строгую и четкую последовательность действий, поэтому для него существенным является способ его задания. От исполнителя требуется лишь четкое выполнение каждого действия. Мы не должны объяснять ему, для каких целей предназначается алгоритм.

Возможности любого исполнителя всегда ограничены. Поэтому, прежде чем составлять алгоритм решения задачи, нужно узнать, какие действия исполнитель может выполнить. Эти действия называются допустимыми действиями исполнителя. Для решения одних и тех же задач исполнители с более «бедным» набором допустимых действий требуют более сложных подробных алгоритмов.

Алгоритм можно записывать разными способами.

Способы записи алгоритма:

1. С помощью рисунка, (например, процесс подключения монитора),
2. На естественном языке - построчно, каждая команда - с новой строки (последовательность проявления фотопленки, последовательность склеивания поверхностей на тюбике с клеем и т. д.
3. Использование псевдокода – полуформализованные описания алгоритмов на некотором условном алгоритмическом языке, которые включают в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др. Псевдокод занимает промежуточное место между естественным и формальным языками. Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций (например, школьный АЯ).
4. Графическое представление – блоксхема.
5. Программное – тексты на языках программирования

Словесный способ записи алгоритмов является описанием последовательных этапов обработки данных. Алгоритм может быть задан в произвольном изложении на естественном языке. Например, алгоритм нахождения наибольшего общего делителя двух натуральных чисел можно представить как следующую последовательность действий:

1. задание двух чисел;
2. если числа равны, то выбор любого из них в качестве ответа и остановка, в противном случае – продолжение выполнения алгоритма;
3. определение большего из чисел;

4. замена большего из чисел разностью большего и меньшего из чисел;
5. повтор алгоритма с шага 2.

Приведенный алгоритм используется для любых натуральных чисел и должен приводить к решению поставленной задачи.

Словесный способ не имеет широкого распространения, так как обладает некоторыми недостатками:

- данные описания строго не формализуемы;
- отличаются многословностью записей;
- допускают неоднозначность толкования отдельных предписаний.

Графический способ представления алгоритмов оказывается более компактным и наглядным по сравнению со словесным. При данном виде представления алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению некоторого числа действий.

Для графического представления алгоритм использует изображение в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Это графическое представление называется *схемой алгоритма*, или *блок-схемой*.

В блок-схеме каждый из типов действий (ввод исходных данных, вычисление значений выражений, проверка условий, управление повторением действий, окончание обработки и т. п.) соответствует геометрической фигуре, представленной в виде блочного символа. Блочные символы соединены линиями переходов, которые определяют очередность выполнения действий.

Псевдокод является системой обозначений и правил, которая предназначена для единообразной записи алгоритмов. Он занимает промежуточное место между естественным и формальным языками. С одной стороны, псевдокод похож на обычный естественный язык, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, благодаря чему запись алгоритма приближается к общепринятой математической записи.

В псевдокоде не применяются строгие синтаксические правила для записи команд, которые присущи формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя. Однако в псевдокоде чаще всего имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном

языке. Например, в псевдокоде, также как и в формальных языках, существуют служебные слова, смысл которых определен раз и навсегда. Их выделяют в печатном тексте жирным шрифтом, а в рукописном тексте подчеркивают. Единый или формальный подход к определению псевдокода не существует, поэтому используются различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Программная форма представления алгоритмов иногда характеризуется некоторыми структурами, состоящими из отдельных базовых (основных) элементов. При данном подходе к алгоритмам изучение основных принципов их конструирования следует начинать с этих базовых элементов. Их описание осуществляется с использованием языка схем алгоритмов и алгоритмического языка.


Графический алгоритм — понятие, подразумевающие под собой разложение действий, которые нужно выполнить для решения определенной задачи, по определенным геометрическим фигурам. Графические схемы изображаются не как попало. Для того чтобы их мог понять любой человек применяются чаще всего блок-схемы.

Также блок-схемы изображаются в соответствии с ГОСТ-19701-90 и ГОСТ-19.003-80. Графические фигуры, применяемые в алгоритме, делятся на:

- *Основные.* Основные изображения применяются для обозначения операций, нужных для обработки данных при решении задачи.
- *Вспомогательные.* Вспомогательные изображения нужны для обозначения отдельных, не самых важных, элементов решения задачи

В графическом алгоритме геометрические фигуры, используемые для обозначения данных, называются блоками. Все блоки идут в последовательности "сверху вниз" и "слева направо" — это правильное направление потока. При правильной последовательности линии, соединяющие между собой блоки, не показывают направление. В остальных случаях направление линий обозначается с помощью стрелок. У правильной схемы алгоритма не должно быть больше одного выхода из обрабатывающих блоков и менее двух выходов из блоков, отвечающих за логические операции и проверку выполнения условий

| Название символа | Обозначение и пример заполнения | Пояснение |
|------------------|---------------------------------|-----------------------------|
| Процесс | $x = (a - b) / \sin(1)$ | Вычислительное действие или |

| | | |
|--------------------------|-------------------------------------------------------------------------------------|------------------------------------------------------|
| | | последовательность действий |
| Решение |  | Проверка условий |
| Модификация |  | Начало цикла |
| Предопределенный процесс |  | Вычисление по подпрограмме, стандартной подпрограмме |
| Вывод-ввод |  | Вывод-ввод в общем виде |
| Пуск остановка |  | Начало, конец алгоритма, вход-выход в подпрограмму |
| Документ |  | Вывод результатов на печать |

Вопрос 2. Системы программирования

Прогресс компьютерных технологий определил процесс появления новых разнообразных знаковых систем для записи алгоритмов – **языков программирования**. Смысл появления такого языка – оснащенный набор вычислительных формул дополнительной информации, превращает данный набор в алгоритм.

Языки программирования – это системы символов и правил их сочетания, предназначенные для взаимодействия человека со сложными машинами. Существуют сотни таких организованных систем, выполняющих различные функции. Чтобы сориентироваться в этом многообразии, специалисты создают общие классификации языков программирования, основанные на том или ином характерном признаке.

Языки программирования могут быть сгруппированы по десяткам различных признаков. Они являются принципиально важными или имеют прикладное значение. Наблюдается сильная зависимость классификации языков программирования от истории развития. С годами технологии усложнялись и кардинально изменялись, на первый план выходило удобство программиста, появлялись эффективные алгоритмы, сложные команды и новые уровни абстракции.

Основные классификации языков программирования по видам и механизму работы базируются на следующих параметрах:

- Особенности лексики и грамматики в совокупности с уровнем абстракции и степенью удобства для человека.
- Базовая концепция и методология составления алгоритмов.
- Способ представления данных.
- Организация процесса взаимодействия с машиной, механизм исполнения программ.
- Область жизни, в которой применяется язык.
- Историческая эпоха, в которую языковая система была сформирована.

Составить четкую классификацию языков и систем программирования нельзя, но возможно разделить и систематизировать их по принципиально важным признакам

Удобство для человека

Инструкции, написанные на машинном языке, понятны компьютеру, но крайне неудобны для человека. В них трудно разобраться, их практически невозможно быстро изменить или составить с их помощью сложный алгоритм. Для повышения эффективности программисты поднялись на новые уровни абстракции и научили машину принимать более человекопонятные инструкции и самостоятельно переводить их в машинный код. Рассмотрим классификацию и особенности языков программирования разного уровня:

- Машинный код. Это тоже язык программирования, на котором при должной подготовке можно написать инструкцию.
- Низкий уровень. По-настоящему низкоуровневыми являются языки ассемблера, которые используют нативные машинные команды, закодированные с помощью мнемонических кодов.
- Средний уровень. Системы программирования этой группы могут считаться и низко- и высокоуровневыми, в зависимости от конкретных представлений о степени абстракции. Сюда можно отнести C и C++.
- Высокий уровень. Эти языки позволяют создавать сложные алгоритмы, но требуют дополнительной обработки перед выполнением, поэтому сгенерированный ими код менее эффективен и выполняется медленнее.
- Сверхвысокий уровень. Эта немногочисленная группа характеризуется появлением сверхмощных команд и операторов. Сюда можно отнести Алгол-68

Особенностью языков низкого уровня является их машинозависимость. Они тесно привязаны к особенностям организации конкретного типа ЭВМ, но в целом похожи друг на друга.

Машинный язык — единственный язык, понятный ЭВМ. Он реализуется аппаратно: каждую команду выполняет некоторое электронное устройство. Программа на машинном языке представляет собой последовательность команд и данных, заданных в цифровом виде.

Данные на машинном языке представлены числами и символами. Операции являются элементарными и из них строится вся программа. Ввод программы в цифровом виде производился непосредственно в оперативную память.

Естественно, что процесс программирования был очень трудоемким, разобраться в программе даже автору было довольно сложно, а эффект от применения ЭВМ был довольно низким.

Этот этап в развитии языков программирования показал, что программирование является сложной проблемой, трудно поддающейся автоматизации, но именно программное обеспечение определяет в конечном счете эффективность применения ЭВМ. Поэтому на всех последующих этапах усилия направлялись на совершенствование интерфейса между программистом и ЭВМ — языка программирования.

Стремление программистов оперировать не цифрами, а символами, привело к созданию мнемонического языка программирования, который называют **ассемблером**, мнемокодом, автокодом. Этот язык имеет определенный синтаксис записи программ, в котором, в частности, цифровой код операции заменен мнемоническим кодом.

Языки низкого уровня обеспечивают:

- высокую скорость выполнения и максимальную компактность создаваемых программ;
- прямое взаимодействие с аппаратными ресурсами;
- полный контроль над памятью

Основные недостатки низкоуровневых языков:

- для каждого типа ЭВМ необходимо использовать определенную систему команд, зависящую от особенностей функционирования машины;
- сложность и низкая скорость процесса программирования;
- высокая вероятность появления ошибок, которые сложно отследить;
- отсутствие мобильности программ, невозможность запустить их на ЭВМ другого типа.

Высокоуровневые системы программирования не привязаны к определенной машинной системе команд и могут выполняться на любом компьютере. Благодаря высокому уровню абстракции они могут себе позволить использовать различные концепции и методологии в составлении инструкций. Поэтому классификация языков программирования высокого уровня очень обширна и сложна.

Принципиальными отличиями языков высокого уровня от языков низкого уровня являются:

- использование переменных;
- возможность записи сложных выражений;
- расширяемость типов данных за счет конструирования новых типов из базовых;
- расширяемость набора операций за счет подключения библиотек подпрограмм;
- слабая зависимость от типа ЭВМ.

Для *машинно-независимых языков* не требуется полного знания специфики компьютеров. С их помощью можно записывать программу в виде, допускающем ее реализацию на ЭВМ с различными типами машинных операций, привязка к которым возлагается на соответствующий транслятор.

Причина бурного развития и применения высокоуровневых языков программирования заключается в быстром росте производительности ЭВМ и хронической нехватке программистских кадров.

Промежуточное место между машинно-независимыми и машинно-зависимыми языками отводится *языку Си*. Он создавался при попытке объединения достоинств, присущих языкам обоих классов. Данный язык обладает рядом особенностей:

- максимально использует возможности конкретной вычислительной архитектуры; из-за этого программы на языке Си компактны и работают эффективно;
- позволяет наилучшим образом использовать огромные выразительные средства современных языков высокого уровня.

Обработка программы машиной

Чтобы выполнить сложную инструкцию, компьютер прежде всего должен понизить ее абстрактность и перевести на понятный для себя язык. Способ, которым это делается, называется моделью исполнения. Выделяют две основных модели и одну гибридную:

- Компиляция - единовременный перевод всей программы в машинный код.
- Интерпретация - последовательное выполнение каждого выражения.

- Транскомпиляция - перевод на язык более низкого уровня, например С или ассемблер, и его последующая компиляция.

Для перевода необходима специальная программа-транслятор - компилятор или интерпретатор, без которой работа с языком невозможна

Интерпретация представляет собой пооператорную трансляцию и последующее выполнение оттранслированного оператора исходной программы. Его присутствие необходимо от начала и до самого конца работы программы

Существует два основных недостатка метода интерпретации:

- постоянное нахождение транслятора в памяти ЭВМ; интерпретирующая программа должна располагаться в памяти ЭВМ на протяжении всего процесса выполнения исходной программы. Другими словами, она должна занимать некоторый установленный объем памяти
- повторная обработка повторяющихся команд, процесс трансляции одного и того же оператора повторяется такое число раз, которое должна исполнять эта команда в программе. Это приводит к резкому снижению производительности работы программы.

Несмотря на это, интерпретируемые языки очень удобны для циклической разработки и отладки, так как позволяют быстро вносить изменения в программу. *Трансляторы-интерпретаторы* являются достаточно распространенными, так как они поддерживают диалоговый режим.

Компилятор же трудится только один раз, сразу преобразуя всю инструкцию в понятный для компьютера вид - машинный код или некий промежуточный байт-код, а затем покидает память ЭВМ. Процессы трансляции и выполнения при *компиляции* разделяются во времени: сначала исходная программа в полном объеме переводится на машинный язык, после чего оттранслированная программа может многократно исполняться. Для трансляции методом компиляции необходим неоднократный «просмотр» транслируемой программы, т. е. *трансляторы-компиляторы* являются многопроходными. Трансляция методом компиляции носит название *объектного модуля*, который представляет собой эквивалентную программу в машинных кодах. Необходимо, чтобы перед исполнением объектный модуль обрабатывался специальной программой ОС и преобразовывался в *загрузочный модуль*.

Здесь выполнение отделено от процесса трансляции, что является более эффективной.

Основные недостатки компиляционной модели: ***большая сложность***. Прежде чем перевести программу на понятный машине язык,

транслятор много раз проходит по исходной инструкции, анализируя и проверяя ее.

Применяют также трансляторы *интерпретаторы-компиляторы*, объединяющие в себе достоинства обоих принципов трансляции.

Четкого разграничения систем не существует, так как традиционно интерпретируемые языки могут быть скомпилированы и наоборот. Классификация языков программирования высокого уровня по модели исполнения:

- Интерпретируемые - Python, Haskell, PHP, JavaScript.
- Компилируемые сразу в машинный код: C, C++, Fortran, ASM.
- Компилируемые в байт-код: Python, Java.
- Транскомпилируемые: Haskell, Fortran, C, C++.

Принципиальный способ взаимодействия

Высокоуровневые языки используются в машинно-независимых системах программирования. Такие системы программирования в сравнении с машинно-ориентированными системами предстают более простыми в использовании. Высокоуровневые языки могут быть разделены по основной парадигме программирования. Существуют десятки методологий составления программ, некоторые из которых очень похожи друг на друга, поэтому создать четкую систему различий невозможно.

Языки программирования высокого уровня подразделяют на процедурно-ориентированные, проблемно-ориентированные и объектно-ориентированные.

- *Процурно-ориентированные языки* применяются для записи процедур или алгоритмов обработки информации на каждом определенном круге задач.. Требуют явного последовательного описания алгоритма решения задачи. Операторы при этом объединяются в процедурные группы, отделенные от самих данных. Примеры процедурных языков – Фортран, Pascal, Basic, C;
- *Проблемно-ориентированные языки* используются для решения целых классов новых задач, возникших в связи с постоянным расширением области применения вычислительной техники. Максимально формализовано описывают саму задачу и требуемый результат. Решение при этом должно логически следовать из этого описания; Примеры проблемно ориентированных языков: Лисп, Пролог, VHDL.
- *объектно-ориентированные*. Имеют в основе концепцию объекта, объединяющего в себе данные и методы их обработки. Большинство из этих языков являются версиями процедурных и проблемных языков, но программирование с помощью языков

этой группы является более наглядным и простым. К наиболее часто употребляемым языкам относятся:

- а) Visual Basic (~ Basic);
- б) Delphi (~ Pascal);
- в) Visual Fortran (~ Fortran);
- г) C++ (~ C);
- д) Prolog++ (~ Prolog).

Языки последней группы описывают все сущности в виде независимых объектов, скрывающих в себе сложные механизмы. Основными концепциями ООП (объектно-ориентированного программирования) являются:

- инкапсуляция - скрытие функционала внутри объекта;
- наследование одними объектами методов других;
- полиморфизм - изменение сути с сохранением внешнего интерфейса.

Можно составить также классификацию языков объектно-ориентированного программирования по способу реализации основных концепций этого подхода - наследования, инкапсуляции и полиморфизма. Помимо классических механизмов существуют другие, например прототипный, используемый в JavaScript.

Методология ООП считается самой прогрессивной, эффективной и в некотором смысле модной. Однако в ряде случаев для решения конкретной задачи более эффективными могут быть другие подходы, например функциональный

Поколения языков программирования

Классификация языков программирования по истории их появления считается условной, так как не учитывает особенности конкретных систем. Однако она позволяет проследить, как менялись со временем концепции и усложнялись задачи, стоящие перед программистами.

В попытках связать классификацию и эволюцию языков программирования выделились несколько крупных групп, названных поколениями:

- Первое поколение - машинные языки низкого уровня, привязанные к реализации конкретной ЭВМ. "Программы" на этих языках выглядели как ряды переключателей, приведенных в нужное положение, или перфокарты (перфоленты). Таким образом, все команды представляли собой последовательность нулей и единиц - бинарный код. Пример: язык ARM-процессора.
- Во втором поколении языки стали немного понятнее для человека, но отвязать их от конкретного аппарата так и не удалось. Это время языков ассемблера с его мнемоническими

кодами и однозначной сборкой в машиночитаемую форму. Пример: Макроассемблер.

- Языки третьего поколения сняли с программиста заботу о непринципиальных деталях составления инструкций, таких как перевод программы в машинный код. Теперь компьютер научился делать это самостоятельно. Синтаксис и лексика приблизились к человеческим, стали понятнее. В этом поколении зародились практически все современные языки высокого уровня с широкой областью применения, независимо от их парадигмы: PHP, Fortran.
- В четвертом поколении уровень абстракции возрос еще больше, резко сужая область использования. К этой группе относятся такие специфичные языки, как FoxPro, Simulink, SQL. Появились языки визуального программирования: CAD-пакеты, системы RAD.
- Наконец, языки пятого поколения должны были сами писать программы, получая лишь описание от программиста. Эта задумка так и не была реализована полностью, так как для составления эффективного алгоритма иногда недостаточно прямой машинной логики, а требуется еще человеческая интуиция и смекалка. Примерами языков пятого поколения являются MathCAD, Prolog и Mercury.

По сути, поколения языков в точности соответствуют этапам программирования, рассмотренным в начале статьи, перечисленным в обратном порядке. Изначально программист все операции брал на себя, а машина лишь выполняла заданную последовательность действий. Теперь же ЭВМ способна выдавать результат по формализованному описанию задачи.

Облегчение труда программиста сопровождается увеличением нагрузки на машину, программа выполняется медленнее и требует больших ресурсов.

Несмотря на то что технический прогресс движется семимильными шагами, языки первых поколений вовсе не исчезли. Они применяются в областях, требующих максимальной простоты и эффективности.

Область применения

Каждый язык программирования хорош в своей области, для которой он создавался. Так, для программирования микроконтроллеров используются ассемблеры, а с Java там делать нечего. Низкоуровневое программирование драйверов эффективно с C, который позволяет строго контролировать ресурсы памяти. Для веб-программирования стоит выбрать скриптовые языки PHP и JavaScript, интерпретатор которого встроен в каждый современный браузер. Важные банковские программы написаны на Java, обеспечивающем контроль ошибок. Аэрокосмические - тоже на Java или на Паскале, который даже уборку мусора отдает на контроль программисту.

Все языки хороши, нужно лишь подобрать соответствующий задаче

Важность классификации

Сложно разделить сотни существующих систем взаимодействия человека и ЭВМ на несколько четких групп. И все же обзор классификаций языков программирования кратко прослеживает историю их эволюции и глубже раскрывает заложенные в них идеи.

Каждый язык одновременно относится к нескольким из перечисленных групп - он может быть строго типизированным, компилируемым и объектно-ориентированным одновременно. Поэтому нельзя рассматривать многообразие систем программирования через призму какой-то одной классификации

Изобретение языка программирования высшего уровня позволило нам общаться с машиной, понимать её (если конечно Вам знаком используемый язык), как понимает американец немного знакомый с русским языком древнюю азбуку Кириллицы. Проще говоря, мы в нашем развитии науки программирования пока что с ЭВМ на ВЫ. Поверьте мне, это не сарказм вы только посмотрите как развилась наука программирования с того времени, как появились языки программирования, а ведь язык программирования высшего уровня, судя по всему ещё младенец. Но если мы обратим внимание на темпы роста и развития новейших технологий в области программирования, то можно предположить, что в ближайшем будущем, человеческие познания в этой сфере, помогут произвести на свет языки, умеющие принимать, обрабатывать и передавать информации в виде мысли, слова, звука или жеста.

Вопросы для самопроверки:

1. Что такое алгоритм?
2. Какие условия нужно учитывать при создании алгоритма?
3. Что характеризует исполнителя алгоритма?
4. Перечислите основные свойства алгоритма
5. Перечислите способы записи алгоритма
6. Что такое язык программирования?
7. По каким основным признакам классифицируют языки программирования?
8. Какое принципиальное отличие языков низкого уровня от языков высокого уровня?
9. Какие есть модели обработки программы компьютером?
10. Какие существуют методологии программирования?