**Определение алгоритма**

Слово «Алгоритм» происходит от algorithmi – латинского написания имени аль-Хорезми, под которым в средневековой Европе знали величайшего математика из Хорезма (город в современном Узбекистане) Мухаммеда бен Мусу, жившего в 783-850 гг. В своей книге «Об индийском счете» он сформулировал правила записи натуральных чисел с помощью арабских цифр и правила действий над ними.

***В дальнейшем алгоритмом стали называть точную и конечную последовательность действий, обеспечивающую получение требуемого результата на основе исходных данных****.*

Существуют природные (например, биологические) алгоритмы. К примеру, развитие живого существа из клетки по алгоритму, заданному в ДНК (Дезоксирибонуклеи́новая кислота). Такие алгоритмы разрабатывает природа (или бог, если хотите), а человек их изучает, т.к. подобные алгоритмы чрезвычайно сложны.

Но более простые алгоритмы для решения практических или теоретических задач может разработать и человек. Далее мы будем рассматривать именно такие алгоритмы.

Создание алгоритма, пусть даже самого простого, - процесс творческий. Он доступен исключительно живым существам, а долгое время считалось, что только человеку.

Другое дело – реализация уже имеющегося алгоритма. Ее можно поручить субъекту или объекту, который не обязан вникать в существо дела, а возможно, и не способен его понять. Такой субъект или объект принято называть **формальным исполнителем**.

Примером формального исполнителя может служить стиральная машина-автомат, которая неукоснительно исполняет предписанные ей действия, даже если вы забыли положить в нее порошок.

Человек тоже может выступать в роли формального исполнителя, но в первую очередь формальными исполнителями являются различные автоматические устройства, и компьютер в том числе.

Каждый алгоритм создается в расчете на вполне конкретного исполнителя. Те действия, которые может совершать исполнитель, называются его **допустимыми действиями**.

**Система команд** – состоит из совокупности допустимых действий алгоритма.

Алгоритм должен содержать только те действия, которые входят в систему команд данного исполнителя.

**Свойства алгоритмов**

Данное выше определение алгоритма нельзя считать строгим - не вполне ясно, что такое «точное последовательность действий, обеспечивающая получение требуемого результата». Поэтому обычно формулируют несколько общих свойств алгоритмов, позволяющих отличать алгоритмы от других инструкций.

Такими свойствами являются:

**Дискретность** - алгоритм должен быть представлен как последовательность простых шагов (команд). Каждый последующий шаг алгоритма, исполняется только после того, как закончилось исполнение предыдущего.

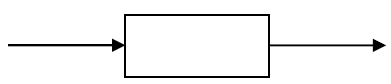
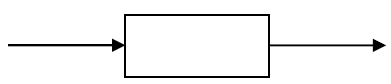
**Определенность** - каждое шаг алгоритма должен быть однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

**Результативность** (конечность) - алгоритм должен приводить к решению задачи за конечное число шагов.

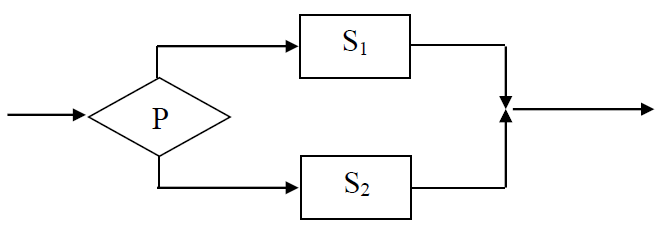
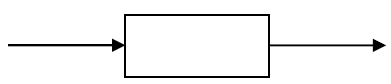
**Массовость** - алгоритм решения задачи разрабатывается в общем виде, то есть, он должен быть применим для некоторого класса задач, различающихся только исходными данными

**Виды алгоритмов и их реализация**

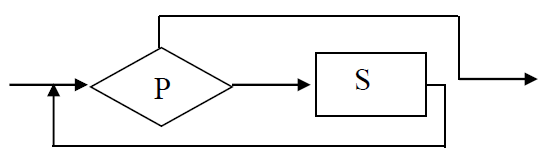
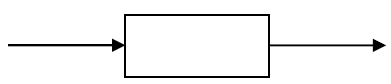
***Линейный*** алгоритм – набор команд (указаний), выполняемых последовательно во времени друг за другом.



***Разветвляющийся*** алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого компьютер обеспечивает переход на один из двух возможных шагов.



***Циклический*** алгоритм – алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов.



**Методы описания алгоритмов**

На практике наиболее распространены следующие формы представления алгоритмов:

* словесная (описание на естественном языке);
* графическая или блок-схема (изображения из графических символов);
* псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
* программная (тексты на языках программирования).

**Словесное описание алгоритма**

Данный способ получил значительно меньшее распространение из-за его многословности и отсутствия наглядности.

*Пример открытия входной двери (снаружи).*

Словесный способ не имеет широкого распространения по следующим причинам:

 страдают многословностью;

 допускают неоднозначность толкования отдельных предписаний.

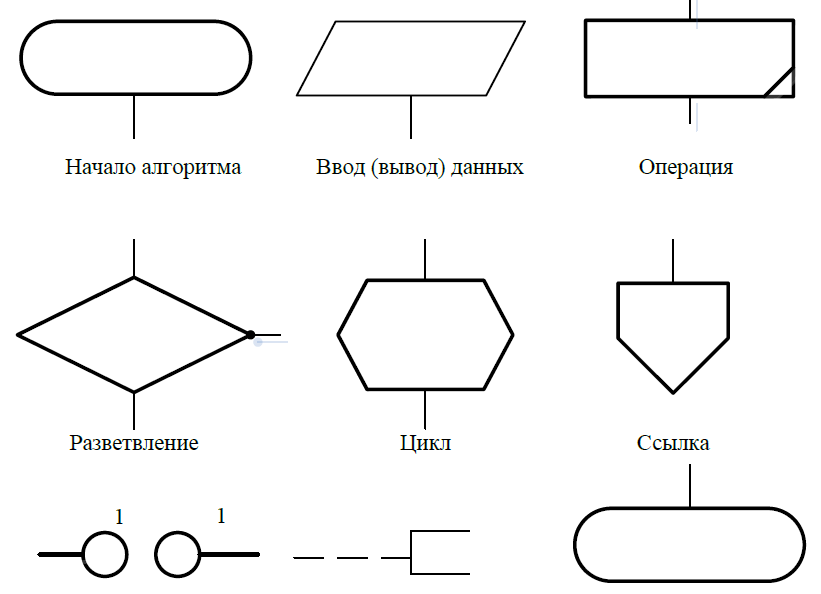
Необходимо отметить, что в будущем, вероятно, именно этот способ будет преобладать.

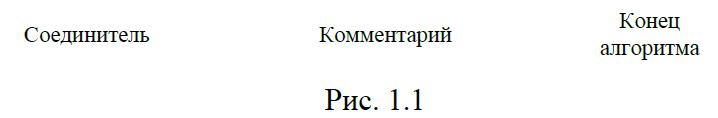
**Блок-схема алгоритма**

А этот способ оказался очень удобным средством изображения алгоритмов и получил широкое распространение в научной и учебной литературе.

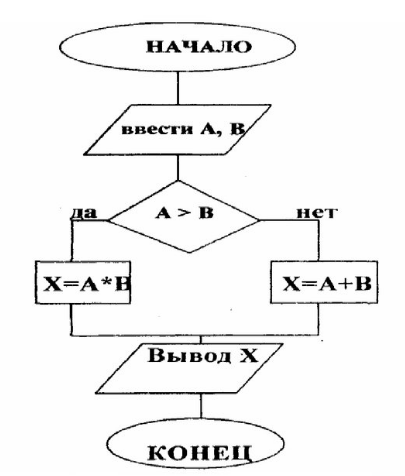
Структурная (блок-, граф-) схема алгоритма – графическое изображение алгоритма в виде схемы связанных между собой с помощью стрелок (линий перехода) блоков – графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

Графическое изображение алгоритма широко используется перед программированием задачи вследствие его наглядности, т.к. зрительное восприятие обычно облегчает процесс написания программы, ее корректировки при возможных ошибках, осмысливание процесса обработки информации (рис 1)





**Пример блок схемы**

****

**Псевдокод**

Псевдокод представляет собой систему обозначений, предназначенную для записи алгоритмов. Он занимает промежуточное место между естественным и формальным языками программирования.

С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде, так же, как и в формальных языках, есть служебные слова, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются. Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Примеры псевдокода:

**алг hello  
 вывод (**'Здравствуй, Мир!**')  
кон алг hello**

или

**алг hello**

**переменная** мир=’Здравствуй, Мир!’

**вывод (**мир**)  
кон алг hello**

или

**алг hello**

**переменная** мир:текст **нач**

мир=’Здравствуй, Мир!’ **вывод (**мир**)  
кон алг hello**

Знаменитая книга «Искусство программирования для ЭВМ» автора Д. Кнут, которая является классикой в этой области, использует псевдокод MIX. Хотя это существенно затрудняет понимание, но позволяет реализовывать фундаментальные алгоритмы без излишних подробностей, какими оснащены современные системы программирования.

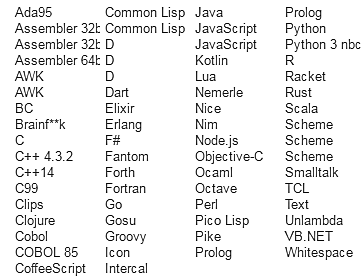
**Программное представление алгоритма**

При записи алгоритма в словесной форме, в виде блок-схемы или на псевдокоде допускается определенный произвол при изображении команд. Вместе с тем такая запись точна настолько, что позволяет человеку понять суть дела и исполнить алгоритм.

Однако на практике в качестве исполнителей алгоритмов используются специальные автоматы — компьютеры. Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на «понятном» ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного толкования их исполнителем.

Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке — программой для компьютера.

Примеры языков программирования:



Разработка алгоритма на языке программирования – процесс достаточно творческий и называется программированием. Один и тот же алгоритм разные программисты, с большой степенью вероятности напишут по-разному, хотя бы потому, что используют разные наименования переменных. Но есть некоторые общие принципы «хорошего» программирования.

**Программирование «сверху-вниз».**

Программирование «сверху вниз», или нисходящее программирование – это методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой.

Таким образом, первоначально рассматривается вся задача в целом. На каждом последующем этапе задача разбивается на более мелкие подзадачи, каждая подзадача, в конечном итоге на еще более мелкие подзадачи и так до тех пор, пока не будут получены такие подзадачи, которые легко кодируются на выбранном языке программирования. При этом на каждом шаге уточняются все новые и новые детали («пошаговая детализация»).

Для демонстрации технологии рассмотрим программирование «сверху-вниз» при разработке тестирующей программы.

1.

ZTesting

ZTesting

Ввод теста (vvtest)

Тестирование (test)

Тестирование (test)

Подготовка (podtest) ((PZTesting

Ввод ответа (inv) (podtest) ((PZTesting

Вывод вопроса (outv) (podtest) ((PZTesting

Анализ (analiz) ((PZTesting

2.

3.

4. . . . . . . . . . . . . . . . . . . . . . . . .

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность некоторые подпрограммы не реализовывать сразу, а временно откладывать, пока не будут закончены другие части.

**Введение в структуры данных.**

При записи алгоритмов на любом языке программирования важное значение имеют данные, которые подаются на вход программы, выходные данные, а также различные внутренние данные, которые используются для вычислений.

Код программы и данные при выполнении всегда размещены в памяти: оперативной или внешней.

Структура размещения данных в памяти разная.

Например, целое и вещественное числа в памяти компьютера хранятся по-разному.

Кроме чисел, в процессе программирования часто возникает необходимость использовать и другие структуры.

Наиболее часто используются следующие структуры данных:

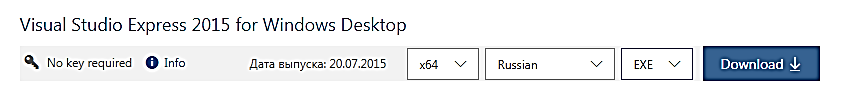
1. Число
2. Булево значение
3. Символ
4. Указатели
5. Строка
6. Массив
7. Запись
8. Класс
9. Множество
10. Таблица
11. Список
12. Стек
13. Очередь
14. Граф
15. Дерево
16. Файл
17. База данных

Все эти структуры представляются в памяти по-разному, и программист обязан отчетливо представлять и знать эти структуры, т.к. это имеет значение для правильной работы алгоритма и в конечном итоге программы.

Для получения навыков программирования будем использовать язык С++.

Это не самый простой язык для изучения, но он используется для решения многих задач. На С++ разработаны такие программы, как IOS, Windows, Unix, Linux, Android на основе Linux, 1C, компилятор Delphi, Photoshop и пр.

Диалектов С++ множество, но мы будем использовать бесплатную официальную версию **Visual Studio Express for Windows Desktop 2015**. Ее можно скачать по адресу: <https://visualstudio.microsoft.com/ru/vs/older-downloads/>



Для скачивания необходимо иметь учетную запись Microsoft.

Временно, для выполнения лабораторных работ можно использовать облачный сервис c++ online эти сервисы бесплатны и доступеы на компьютере, ноутбуке, планшете, смартфоне, при наличии Интернет. Но все лабораторные работы с помощью этого сервиса выполнить будет невозможно.

**Примеры алгоритмов.**

1. Переход дороги на перекрестке со светофором.
2. В куче камней найти самый легкий камень без весов.
3. В куче камней найти самый тяжелый камень без весов.

**Посложнее**

1. Камни лежат в один ряд, вес вперемешку. Отсортировать камни по возрастанию весов, без весов, сравнивая лишь два соседних.
2. Камни лежат в один ряд, вес вперемешку. Отсортировать камни по убыванию весов, без весов, сравнивая лишь два соседних.

**Примеры структур.**

1. Очередь в кассу. Есть первый, есть последний. Первый обслуживается и уходит, следующий становится первым. Новый встает в конец очереди (очередь)
2. Магазин с патронами. Имеет один вход. Вставляется очередной патрон. Следующий патрон вставляется и «топит» предыдущий. И так, пока магазин не заполнится (стек)
3. Отара овец (множество)
4. Целые числа (множество)
5. Буквы алфавита (множество)
6. Платежная зарплатная ведомость (таблица)
7. Каталог книг в библиотеке (список)
8. Города, связанные дорогами (граф)
9. Дерево с ветками и листьями (дерево)

**Программа** – это описание алгоритма решения какой-либо дискретной задачи на специальном языке, который называется языком программирования.

Языки программирования бывают разными, они могут иметь разные команды, предназначение, но у всех языков имеется стандартный набор команд, который позволяет реализовать алгоритм любой задачи. Какие же это команды (или конструкции, или операторы)?

* Начало программы.
* Присваивание.
* Ввод (read).
* Вывод (write).
* Условие (if).
* Цикл (for, while, repeat).
* Конец программы.

Кроме команд любой язык программирования позволяет работать с разными **типами** данных (числами, датами, символами, строками, логическими).

Любые переменные или массивы переменных, которые на самом деле являются участками памяти, могут быть отнесены к тем или иным типам. В зависимости от типа, данные представляются в разном виде и занимают разные объемы памяти.

На протяжении двух-трех десятков лет программисты решали различные задачи на подобных языках программирования: Алгол, паскаль, PL-1, фортран, C и пр. За это время накопилось огромное количество готовых программ и систем.

С развитием технологий, с появлением персональных компьютеров возможности для программистов выросли неизмеримо. Стало недостаточно вводить текст программы в каком-либо редакторе, а затем выполнять его. Появились некоторые стандарты на внешний вид программ и систем (в основном заданный Apple и Microsoft).

Программистам хотелось также использовать программы других программистов. Все эти новые требования привели к тому, что для программистов стали создавать системы программирования, в который язык программирования входит как ее часть.

Итак, под **системой программирования**, мы будем понимать систему, которая автоматизирует составление программы. Система программирования обычно включает:

* Редактор для ввода и редактирования программы.
* Встроенный язык программирования.
* Набор готовых объектов и форм, для вставки в свою программу.
* Для некоторых систем свою базу данных.
* Средства отладки.
* Средства запуска на выполнение.
* Встроенную справочную систему.

Систем программирования достаточно много. Перечислим некоторые из них:

* Delphi, встроенный язык Object Pascal.
* Visual Basic, встроенный язык Basic.
* Microsoft Access, встроенный язык Object Basic.
* Microsoft Visual Studio, встроенный язык C++, C#, Python.
* WordPress, Jumla, Drupal и пр. – встроенный язык PHP или Java
* 1C:Предприятие, встроенный язык 1C.
* IntelliJ IDEA – для Java, Kotlin
* и прочие