Введение в теорию алгоритмов

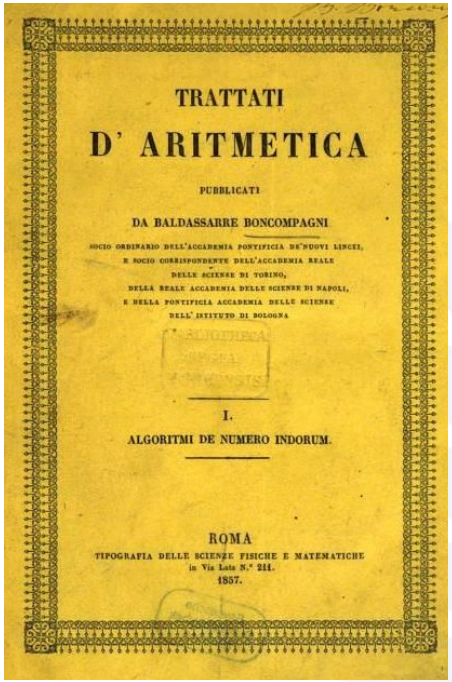
Понятие алгоритма

Направление «Искусственный интеллект и наука о данных», 23.Б16-мм, 23.Б18-мм

22.09.2023

Абрамов Максим Викторович, Есин Максим Сергеевич

История термина «алгоритм»


TRATTATI

D' ARITMETICA

PUBBLICATI

DA BALDASSARRE BONCOMPAGNI

SOCIO ORDINARIO DELL'ACCADEMIA PONTIFICIA DE'NUOVI LINCEI,
E SOCIO CORRISPONDENTE DELL'ACCADEMIA REALE
DELLE SCIENZE DI TORINO,
DELLA REALE ACCADEMIA DELLE SCIENZE DI NAPOLI,
E DELLA PONTIFICIA ACCADEMIA DELLE SCIENZE
DELL' ISTITUTO DI BOLOGNA

I.

ALGORITMI DE NUMERO INDORUM.

ROMA

TIPOGRAFIA DELLE SCIENZE FISICHE E MATEMATICHE
in Via Lata N.° 211.
1857.

- Происходит от имени учёного-математика из Средней Азии IX века – Аль-Хорезми
- Латинский перевод книги Аль-Хорезми назывался *Algoritmi de numero Indorum*, что дословно переводится как *Алгоритмы о счёте индийском*

Интуитивное понятие алгоритма

Санкт-Петербургский государственный университет

Алгоритм — это не просто набор конечного числа правил, задающих последовательность выполнения операций для решения задачи определенного типа. Помимо этого, он имеет пять важных *особенностей*.

1. **Конечность**. Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов
2. **Определенность**. Каждый шаг алгоритма должен быть точно определен. Действия, которые нужно выполнить, должны быть строго и недвусмысленно определены для каждого возможного случая
3. **Ввод**. Алгоритм имеет некоторое (возможно, равное нулю) число входных данных, т. е. величин, которые задаются до начала его работы или определяются динамически во время его работы. Эти входные данные берутся из определенного набора объектов
4. **Вывод**. У алгоритма есть одно или несколько выходных данных, т. е. величин, имеющих вполне определенную связь с входными данными
5. **Эффективность**. Алгоритм обычно считается эффективным, если все его операторы достаточно просты для того, чтобы их можно было точно выполнить в течение конечного промежутка времени с помощью карандаша и бумаги *Искусство программирования,* Дональд Кнут

В книге также дано формальное определение алгоритма с помощью теории множеств

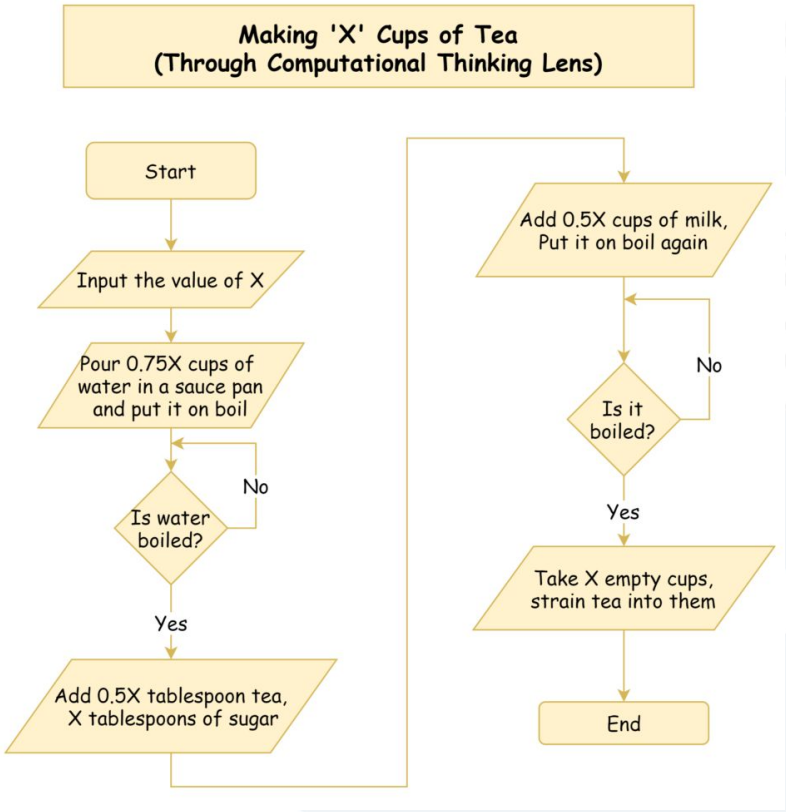
Интуитивное понятие алгоритма

Алгоритм — набор инструкций, четко описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий. Он обладает следующими свойствами:

1. **Дискретность**: алгоритм должен представлять решение задачи как последовательность некоторых простых шагов
2. **Детерминированность:** алгоритм должен в каждый конкретный момент времени однозначно определять следующий шаг исходя из состояния системы
3. **Понятность**: алгоритм должен включать только те команды, которые исполнитель может выполнить
4. **Конечность**: алгоритм при корректных входных данных должен заканчиваться и выдавать результат
5. **Универсальность**: алгоритм должен быть применим к разным наборам входных данных
6. **Результативность**: алгоритм должен завершаться с каким-то результатом

Андрей Николаевич Терехов

Представление алгоритма



- Любой **алгоритм**, который подходит под одно из интуитивных определений, возможно **представить в виде *функции***, которая переводит некоторое **входное слово** в заданном алфавите в **выходное слово** в том же алфавите по заданным правилам перехода и за конечное число шагов
- Дальнейшие формализации понятия *алгоритм* будут определять алгоритм как одну из таких функций
- Класс функций, которые подходят под интуитивное определение алгоритма, называют *классом вычислимых функций*

Формальные определения: машина Тьюринга

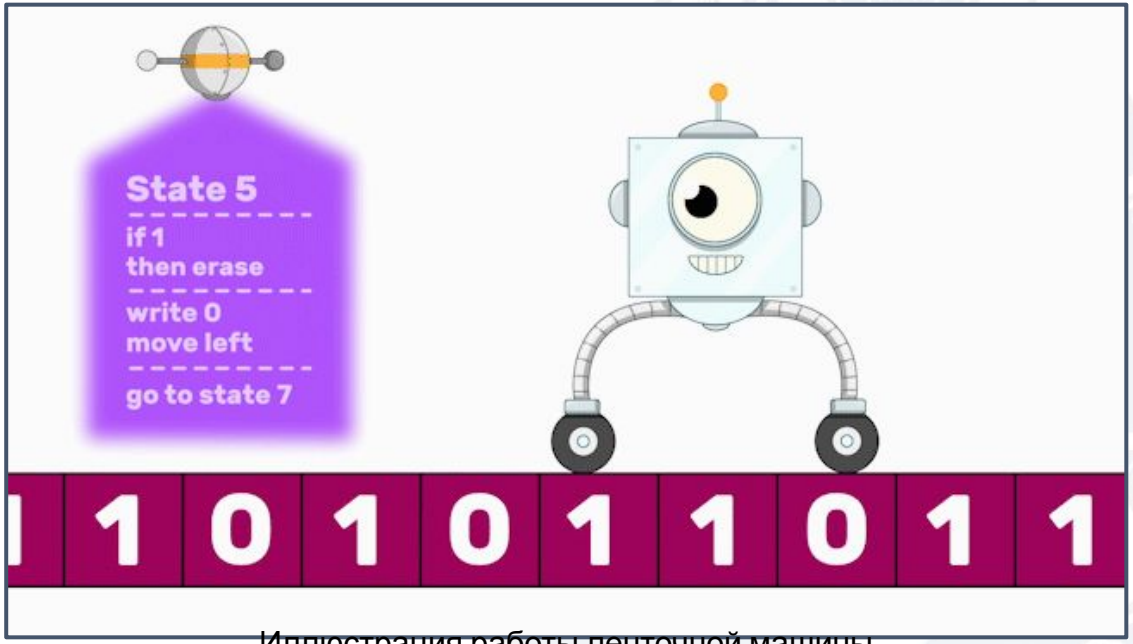
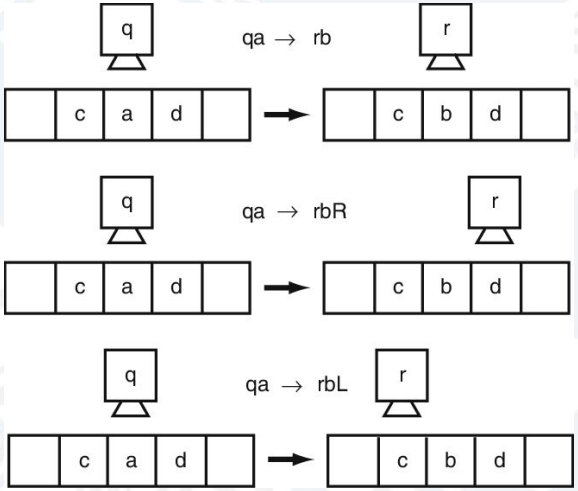


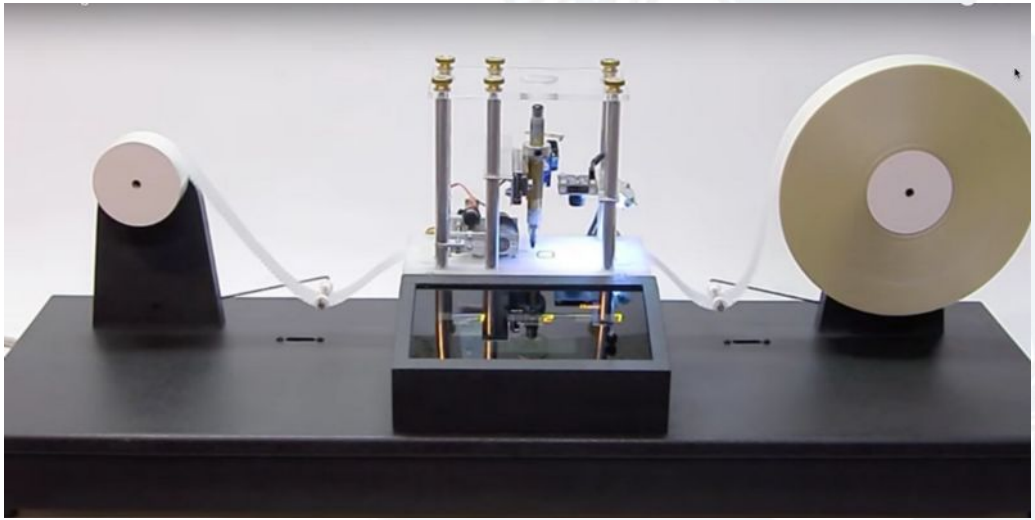
Иллюстрация работы ленточной машины Тьюринга

- Неограниченная в обе стороны лента, разделенная на ячейки
- Управляющее устройство, имеющее множество состояний

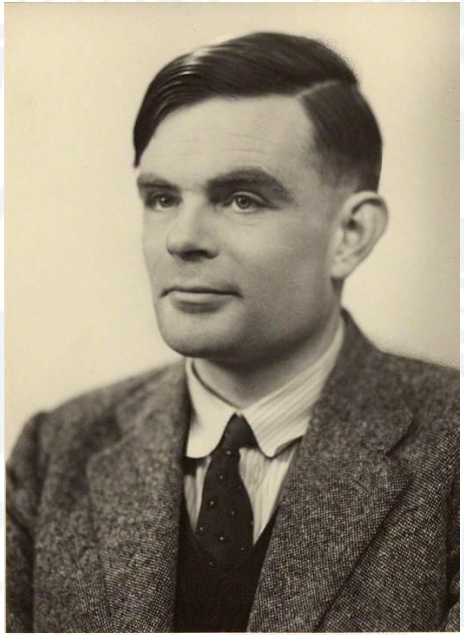


Примеры исполнения команд

Формальные определения: машина Тьюринга



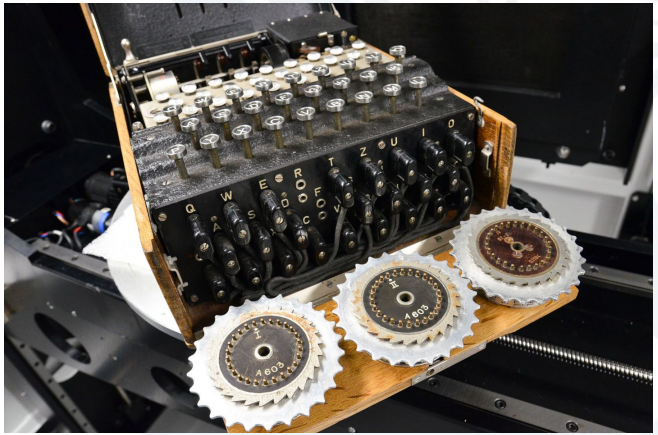
Ленточная машина Тьюринга



Алан Тьюринг (1912-1954) — английский математик, логик и криптограф

Формальные определения: машина Тьюринга

- Во времена Второй Мировой войны разработки Алана Тьюринга позволяли расшифровывать сообщения, зашифрованные с помощью немецкой машины *Энигма*



Формальные определения: машина Тьюринга

Основные положения

- **Всякий интуитивный алгоритм может быть реализован с помощью некоторой машины Тьюринга**
- Алгоритмом в данном случае будем считать **набор состояний и переходов**, входные и выходные данные располагаются на ленте
- Предложенная Тьюрингом в 1936 году абстрактная вычислительная машина до сих пор является предметом изучения теории вычислимости

Формальные определения: Марковские алгорифмы

0. 110
1. 0|10
2. 0|0|0
3. 00|||0
4. 00||0||
5. 00|0||||
6. 000||||||
7. 00||||||
8. 0||||||
9. ||||||

Rule

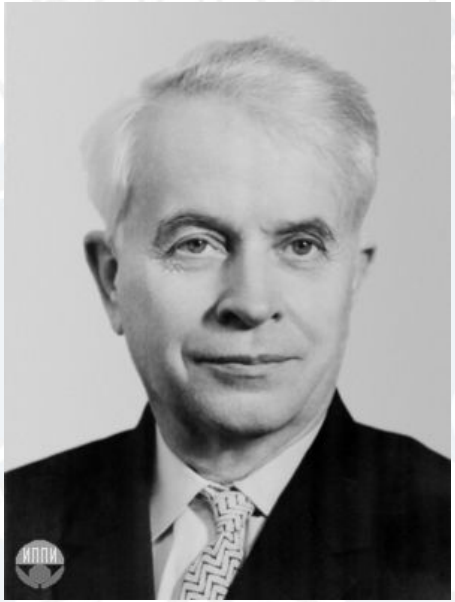
1. |0→0||
2. 1→0|
3. 0→””

Пример работы алгоритма Маркова

- Нормальный алгоритм (алгорифм) Маркова — один из способов формализации понятия *алгоритм*
- Механизм состоит из правил замены одной подстроки на другую
- Последовательное применение правил переводит входную строку в выходную (как и машина Тьюринга)

Формальные определения: Марковские алгорифмы

Андрей Андреевич Марков
(1856-1922) — русский математик, создатель теории цепей Маркова



Андрей Андреевич Марков
(1903-1979) — советский математик, создатель нормальных алгоритмов Маркова

Основные положения

- **Всякий интуитивный алгоритм может быть реализован с помощью некоторого нормального алгоритма Маркова**
- Алгоритмом в данном случае будем считать **набор переходов**, по которым из входной строки получаем выходную строку
- Нормальные алгоритмы впервые были предложены Марковым в конце 1940-х годов и являются эквивалентной заменой алгоритмов для машины Тьюринга

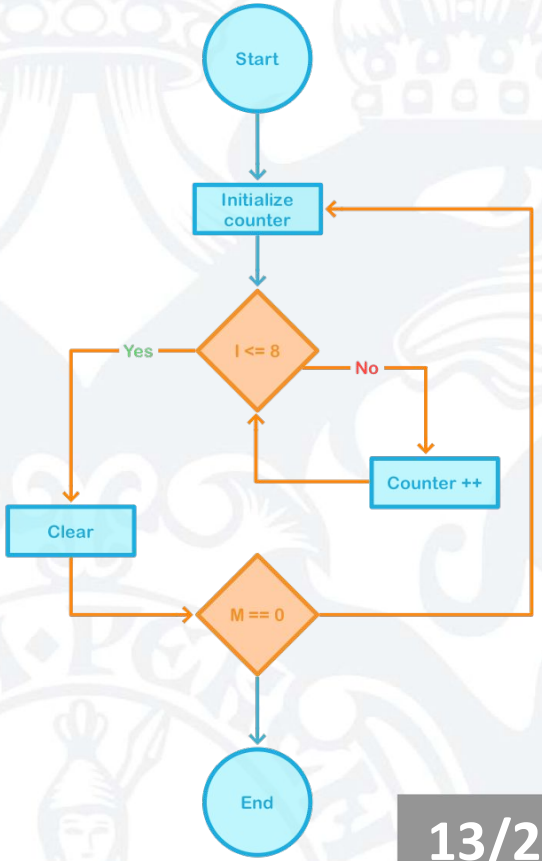
Тезис Тьюринга-Черча

- Алгоритмы для машины Тьюринга, нормальные алгоритмы Маркова и все алгоритмы, которые подходят под интуитивное определение на самом деле **определяют один и тот же класс вычислимых функций**
- Например, машина Тьюринга может моделировать работу нормальных алгоритмов, и наоборот
- Формально доказать этот факт невозможно, потому что понятие алгоритма вводилось неформальными средствами языка

Алгоритм как программа на ЯП

- Не все программы на одном из языков программирования подходят под интуитивное определение алгоритма

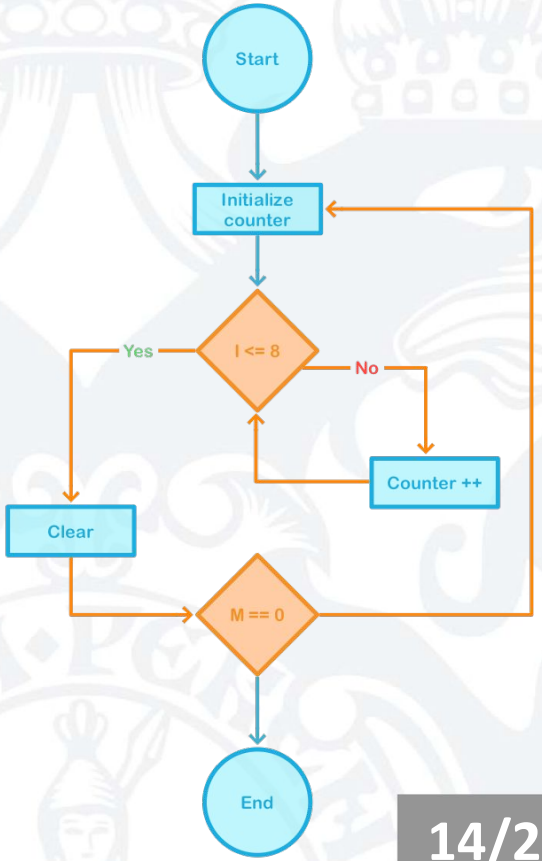
1. **Дискретность**:
2. **Детерминированность:**
3. **Понятность**:
4. **Конечность**:
5. **Универсальность**:
6. **Результативность:**



Алгоритм как программа на ЯП

- Не все программы на одном из языков программирования подходят под интуитивное определение алгоритма

1. **Дискретность**: алгоритм должен представлять решение задачи как последовательность некоторых простых шагов
2. **Детерминированность:** алгоритм должен в каждый конкретный момент времени однозначно определять следующий шаг исходя из состояния системы
3. **Понятность**: алгоритм должен включать только те команды, которые исполнитель может выполнить
4. **Конечность**: алгоритм при корректных входных данных должен заканчиваться и выдавать результат
5. **Универсальность**: алгоритм должен быть применим к разным наборам входных данных
6. **Результативность:** алгоритм должен завершаться с каким-то результатом



Введение в теорию алгоритмов

Структурная парадигма программирования

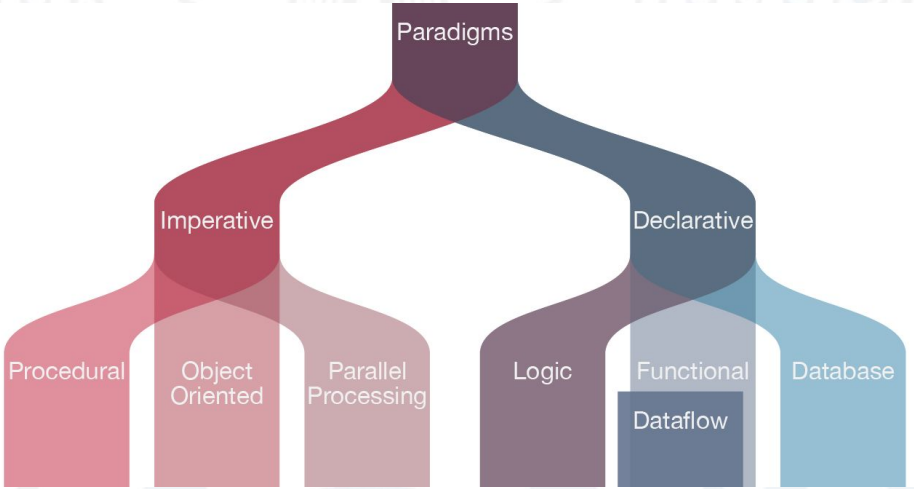
Направление «Искусственный интеллект и наука о данных», 23.Б16-мм, 23.Б18-мм

22.09.2023

Абрамов Максим Викторович, Есин Максим Сергеевич

Парадигма структурного программирования

- Любая парадигма программирования является набором концепций, правил и абстракций, определяющих один из стилей написания программ
- Парадигма структурного программирования в свое время являлась большим шагом в сторону «систематического, рационального подхода к конструированию программ» (*Бертран Мейер*)



Алгоритм как программа на ЯП

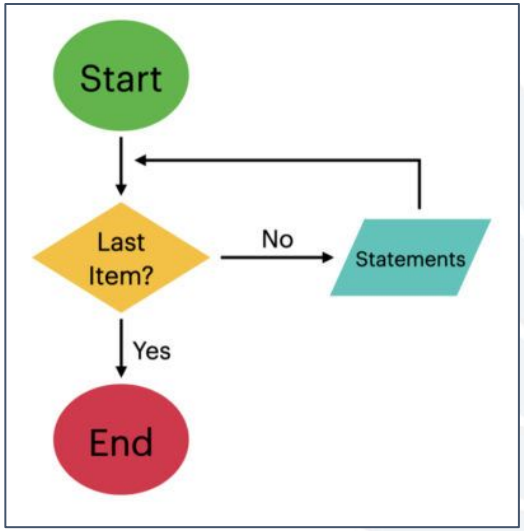
- **Любую программу-алгоритм можно описать с помощью трех основных управляющих конструкций** — главный принцип, на котором строится *структурная парадигма программирования*
- Основные управляющие конструкции языка:
  -
  -
  -
- *Теорема Бёма – Якопини*, доказанная двумя итальянскими математиками в 1966 году, является научным положением, которое использовал *Эдсгер Дейкстра* в 1968 году для описания способа представления алгоритмов с помощью лишь трех приведенных конструкций, отказавшись от оператора goto

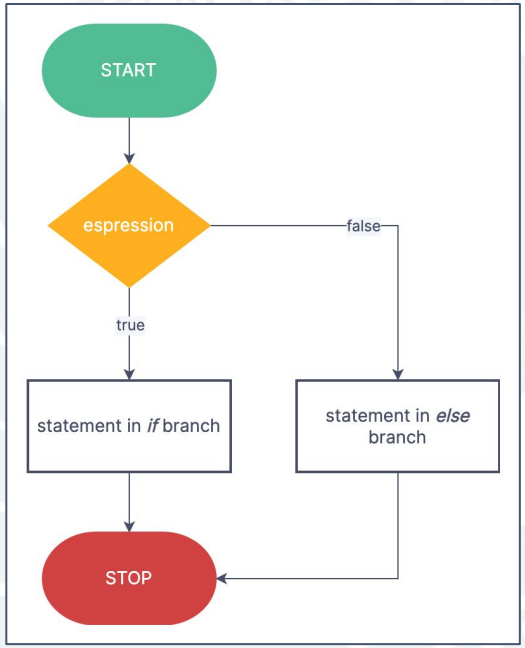
Алгоритм как программа на ЯП

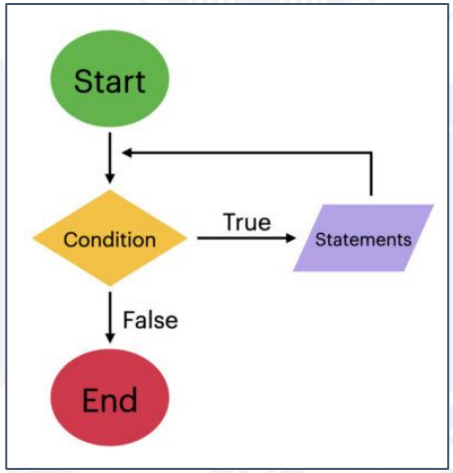
- **Любую программу-алгоритм можно описать с помощью трех основных управляющих конструкций** — главный принцип, на котором строится *структурная парадигма программирования*
- Основные управляющие конструкции языка:
  - Условный оператор
  - Последовательное выполнение
  - Циклическое выполнение
- *Теорема Бёма – Якопини*, доказанная двумя итальянскими математиками в 1966 году, является научным положением, которое использовал *Эдсгер Дейкстра* в 1968 году для описания способа представления алгоритмов с помощью лишь трех приведенных конструкций, отказавшись от оператора goto

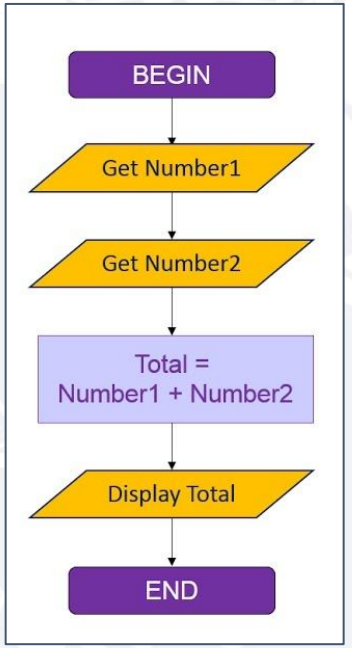
Основные управляющие конструкции

Какие конструкции изображены на схемах?









Немного об операторе goto


```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP: while ( a < 20 ) {
      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a = a + 1;
   }

   return 0;
}
```

Пример использования оператора goto

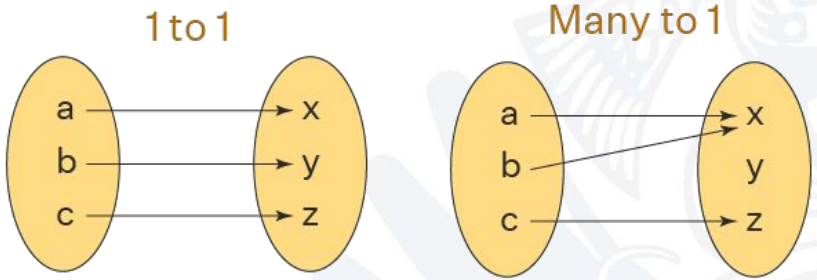
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Вывод программы

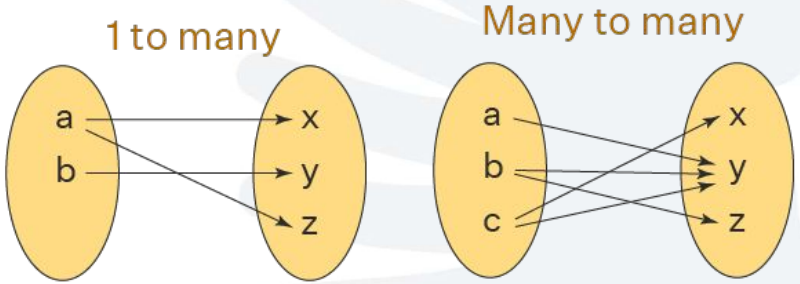
- Оператор *goto* является оператором *безусловного следования*
- Использование goto в программе снижает читабельность кода и гибкость к изменениям в нем
- Один из принципов структурного программирования постулирует отказ от оператора goto

Подпрограммы: функции и процедуры

Function



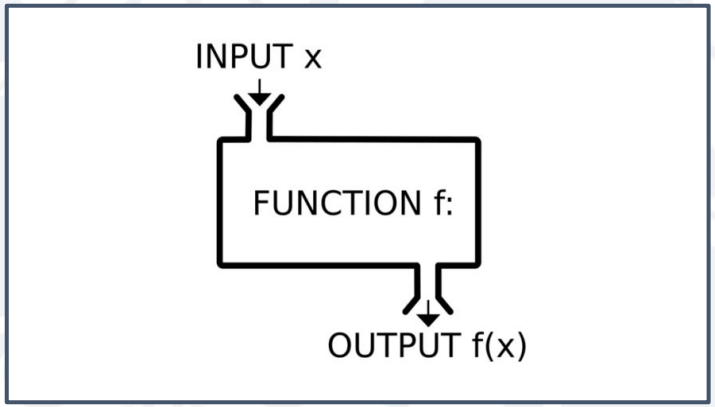
Non–Function



- Функцией в математике принято считать соответствие между двумя множествами, при котором каждому элементу одного множества соответствует **единственный** элемент другого множества
- Функции в программировании определяются аналогично: значение функции зависит от входных данных, и при этом для одних входных данных не может быть два набора выходных данных

Подпрограммы: функции и процедуры

- **Функция** – подпрограмма, которая кроме получения входных параметров, выполнения действий и передачи результатов работы через параметры **может возвращать результат**
- **Вызов функции** с точки зрения языка **является выражением** и может использоваться сколько угодно раз в других выражениях или в качестве правой части присваивания
- **Процедура** – любая подпрограмма, которая не является функцией



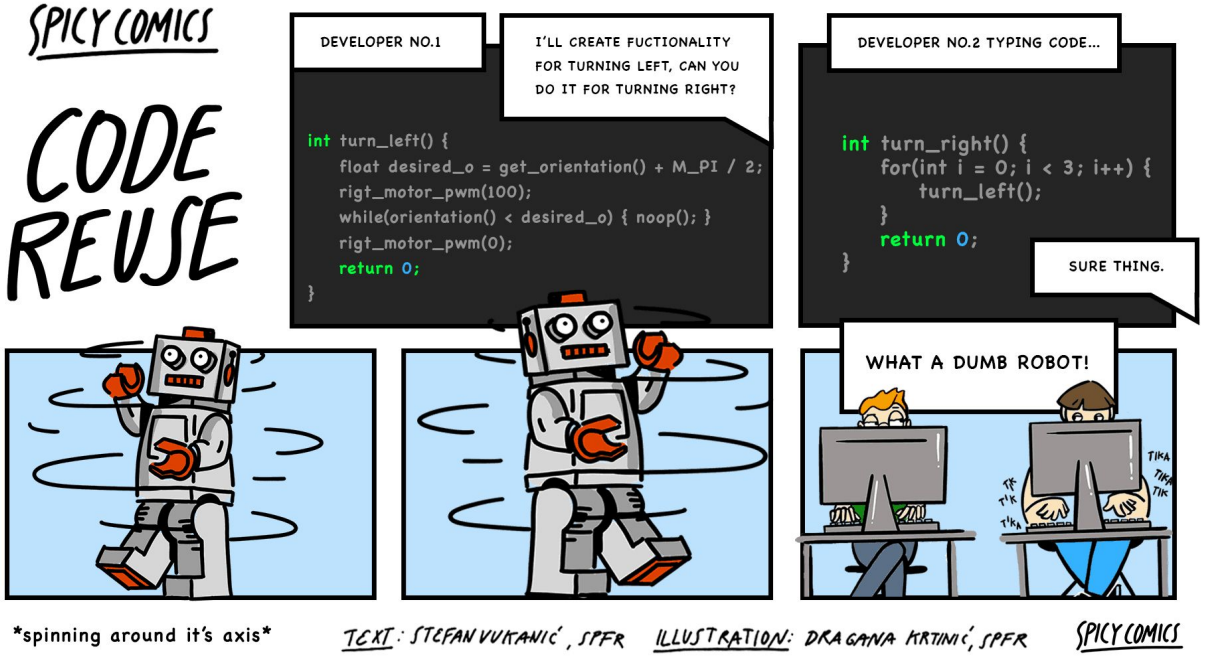
Вопрос: чем отличаются *аргументы* от *параметров*?

Подпрограммы: структурная декомпозиция

Пример декомпозиции большой задачи на несколько маленьких

- Разработка программы в структурном стиле ведётся пошагово, ***методом «сверху вниз»***
- Декомпозиция начинается с главной задачи: ее решение описывается как решение нескольких более простых задач
- Каждая из полученных задач аналогично разбивается на меньшие задачи

Подпрограммы: переиспользование



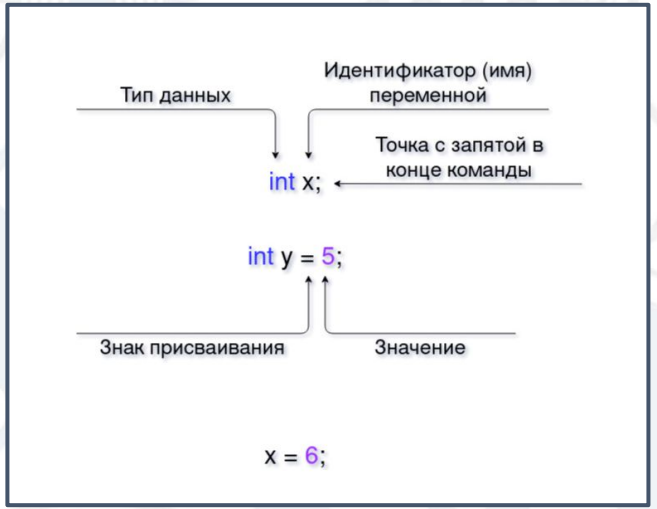
- Разбиение кода программы на подпрограммы позволяет переиспользовать функциональность в разных ситуациях

Основные принципы структурного программирования

- **Отказ от оператора goto** в пользу понятности и прямолинейности выполнения
- Любую программу можно описать с помощью **трех базовых конструкций**
- Базовые конструкции могут быть **вложены друг в друга**
- Сложные программы разбиваются на подпрограммы-функции-процедуры, при этом **декомпозиция выполняется *«сверху вниз»***
- Подпрограммы **подлежат переиспользованию**

Написание кода: переменные и типы данных

- ***Переменные*** используют в ЯП для хранения различных данных, иными словами, **переменная – область памяти**, имеющая имя, которое называют *идентификатором*. Хороший стиль программирования подразумевает осмысленные названия переменных
- ***Типы данных*** в ЯП указывают на то, как интерпретировать данные в памяти: целочисленный, строковый, булевый, символьный типы, числа с плавающей точкой, указатели, структуры, классы



Написание кода: синтаксис и семантика

Процесс компиляции исходного кода состоит из следующих этапов:

- **Лексический анализ**. Последовательность символов исходного файла преобразуется в последовательность лексем.
- **Синтаксический анализ**. Последовательность лексем преобразуется в дерево разбора.
- **Семантический анализ**. Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д.
- **Оптимизация**. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла.
- **Генерация кода**. Из промежуточного представления порождается машинный код, который понимает исполнитель

Темы для докладов

1. **Формальное определение алгоритма по Кнуту** в терминах теории множеств, раздел 1.1
2. **Алан Тьюринг и дешифрация Энигмы**, есть видео (первая часть, вторая часть)
3. **Класс частично-рекурсивных функций** (может быть сложно, по материалам пункта 3.2)
4. **Сила имен переменных** (доклад на основе XI главы книги "Совершенный код" Стива Макконнелла)
5. **Метрики качества кода**: метрики Холстеда и цикломатическая сложность, по материалам Теории разработки программного обеспечения

Тестовый тест

Санкт-Петербургский государственный университет

- В рамках семестра планируется провести примерно 6 тестов по мотивам лекций, первый тест будет в начале следующей лекции или на следующем семинаре
- Тест в формате ответов на вопросы будет оцениваться автоматически, то есть писать развернутые ответы скорее всего не придется (но это не точно)
- Тест будет по QR-коду на последнем слайде. Нужно проверить, что вы можете перейти по ссылке с телефона, заполнить и отправить форму