№ 1 Издание основано в 1995 г inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА

внутри номера CD и код доступа к электронной версии

издательский дом 1september.ru
Первое сентября
январь 2012
ИНФОРМАТИКА Подписка: «Роспечать» – 32291 (бумажная версия), 19179 (электронная); «Почта России» – 79066 (бумажная версия), 12684 (электронная)

НА ОБЛОЖКЕ

Алгоритм Евклида для нахождения НОД — один из древнейших алгоритмов, дошедший до нас в “документированном” виде. Алгоритм не был открыт Евклидом, упоминание о нем имеется еще у Аристотеля. В “Началах” алгоритм изложен (причем дважды) в геометрической форме — для нахождения “общей меры” двух отрезков.

Имеется удивительная связь между алгоритмом Евклида и числами Фибоначчи. Эта связь была установлена в XIX веке французским математиком Габриелем Ламе, который установил, что наихудшими в смысле количества шагов алгоритма исходными данными являются последовательные числа Фибоначчи.

В НОМЕРЕ

НА ДИСКЕ

ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ:

- Интерпретаторы машин Поста и Тьюринга
- Задачи для машины Тьюринга из номеров “Информатики” прошлых лет
- Учебные базы данных
- Коды примеров к статье о языке Dart
- Презентации к публикациям раздела “В мир информатики”
- Материалы для подготовки к ЕГЭ по информатике (автор — Д.М. Златопольский)

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ: по каталогу “Роспечати”: 32291 (бумажная версия), 19179 (электронная версия); “Почта России”: 79066 (бумажная версия), 12684 (электронная версия)

http://inf.1september.ru

Учебно-методический журнал для учителей информатики
Основан в 1995 г.
Выходит один раз в месяц

РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский (редактор вкладки “В мир информатики”)
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 6300 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ “ПЕРВОЕ СЕНТЯБРЯ”
Главный редактор: Артем Соловейчик (генеральный директор)
Коммерческая деятельность: Константин Шмарковский (финансовый директор)
Развитие, IT и координация проектов: Сергей Островский (исполнительный директор)
Реклама, конференции и техническое обеспечение Издательского дома: Павел Кузнецов
Производство: Станислав Савельев
Административно-хозяйственное обеспечение: Андрей Ушков
Главный художник: Иван Лукьянов
Педагогический университет: Валерия Арсланьян (ректор)

ГАЗЕТА ИЗДАТЕЛЬСКОГО ДОМА
Первое сентября – Е.Бирюкова

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА
Английский язык – А.Громушкина
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – О.Коротова
Дошкольное образование – М.Аромштам
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – М.Сартан
История – А.Савельев
Классное руководство и воспитание школьников – О.Леонтьева
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Управление школой – Е.Рачевский
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школьный психолог – И.Вачков

УЧРЕДИТЕЛЬ: ООО “ЧИСТЫЕ ПРУДЫ”

Зарегистрировано ПИ № ФС77-44341 от 22.03.2011 в Министерстве РФ по делам печати
Подписано в печать: по графику 12.12.2011, фактически 12.12.2011
Заказ №
Отпечатано в ОАО “Чеховский полиграфический комбинат” ул. Полиграфистов, д. 1, Московская область, г. Чехов, 142300

АДРЕС ИЗДАТЕЛЯ: ул. Киевская, д. 24, Москва, 121165
Тел./факс: (499) 249-31-38
Отдел рекламы: (499) 249-98-70
http://1september.ru

ИЗДАТЕЛЬСКАЯ ПОДПИСКА:
Телефон: (499) 249-47-58
E-mail: podpiska@1september.ru

Перед употреблением ознакомиться, или С боем московских курантов

С боем московских курантов этот номер был доставлен в Личные кабинеты подписчиков на нашем сайте. Если вы читаете эти строки в 0 часов 1 минуту 1 января, вы и так знаете, что это правда, если позже (почему? ☺) — спросите у коллег.

Разумеется, доставку номеров ровно в полночь обеспечивает автомат (информатикам об этом можно не говорить, но не для всех наших коллег — учителей других предметов — это очевидно). Но уже в 9 утра 1 января в Личных кабинетах загорелась "зеленая лампочка", свидетельствующая о начале работы живого сотрудника службы поддержки (какой он был "живой" после Нового года мы, пожалуй, умолчим). И 1 января, и в любой другой день, включая выходные и праздники, сотрудники службы поддержки работают в Личном кабинете с 9 утра до 23 часов по московскому времени.

Собственно, все это — для информации. Подписчики электронной версии "Информатики" все вышеизложенное знают и так. А внимание тех, кто оформил на почте подписку на бумажную версию журнала — глянцевую, тяжелую, пахнущую типографской краской (наш любимый формат!), хотим обратить на то, что в номер вложена карточка с кодом доступа к электронной версии на сайте. Зарегистрируйте Личный кабинет, активируйте код — и уже февральский номер вы получите с боем курантов (они же бьют каждую ночь, вот 1 февраля и ждите). Разумеется, январский номер вы тоже получите.

Сказанное касается и тех, кто подписался на версию "Информатики" на CD, — в упаковку диска также вложена карточка доступа.

Редакционная политика "Информатики" предписывает публиковать материалы, максимально "подготовленные к употреблению". Что это значит? Мы стараемся обеспечивать публикации электронной поддержкой, достаточной для того, чтобы применить материалы на практике, приложив не очень много усилий.

Если на диске к номеру (диск полностью соответствует комплекту файлов в Личном кабинете) вы не нашли чего-то, что необходимо для использования опубликованных материалов, пожалуйста, свяжитесь с нами. Возможно, мы сможем выполнить ваш запрос в индивидуальном порядке.

Как связаться? По редакционным вопросам проще всего напрямую писать главному редактору, адрес указан ниже.

Важная информация для тех, кто хочет напечатать в "Информатике" свой материал. Мы публикуем авторские статьи! Мы, разумеется, делаем это бесплатно! (Честно говоря, я в силу своей наивности долгое время не понимал, что в некоторых журналах бывает иначе.) Мы платим авторам гонорары.

Но! Позиция редакции по определению субъективна. Мы сами принимаем решение, что публиковать, что нет. Как бы жестко это ни звучало, принимая решение о публикации, мы всегда исходим не из интересов автора, а из интересов читателей. Поэтому, в частности, мы практически никогда не публикуем поступающие к нам во множестве материалы в жанре "отчет о проделанной работе". Отправляя материал с предложением о публикации, пожалуйста, просто честно ответьте себе на вопрос — если бы мы опубликовали похожий материал вашего коллеги, вам это было бы интересно? Также будьте, пожалуйста, готовы к тому, что редакторы "Информатики" — "добрые звери". Любя авторов и читателей, они готовы "разорвать" любой материал. Жертв не было ни разу, но обиды были.

Мы желаем вам удачного Нового года, здоровья и радости на работе и дома!

Сергей Островский,
главный редактор
(*so@1september.ru*)

О чем это?
Предлагаем вашему вниманию исчерпывающее и систематическое изложение одной из самых глубоких и интересных тем профильного курса информатики. Что из этого и в каком объеме рассказывать детям, каждый решит сам, но, уверены, все получат удовольствие от ясных и четких ответов на сложные вопросы.

И это все?!
Нет! На диске имеются авторские интерпретаторы машин Тьюринга и Поста, а также электронные материалы публикаций "Информатики" прошлых лет, включающие, в частности, цикл задач для машины Тьюринга.

Элементы теории алгоритмов

Уточнение понятия алгоритма

Зачем нужно определение алгоритма?

Как вы знаете, алгоритмом называют точный набор инструкций для исполнителя, который приводит к решению задачи за конечное время.

Особый интерес проявляли к алгоритмам математики. Один из древнейших известных алгоритмов — алгоритм Евклида для вычисления наибольшего общего делителя (НОД) двух натуральных чисел. Само слово "алгоритм" (от имени узбекского математика IX века *аль-Хорезми*, которого считают основателем алгебры) ввел в науку в XVII веке немецкий математик Г.В. Лейбниц.

Долгое время считалось, что для любой математической задачи можно найти метод (алгоритм) решения, просто для ряда задач такие алгоритмы еще не найдены. Эту идею высказал еще аль-Хорезми, такой же точки зрения придерживались и другие математики вплоть до начала XX века.

Однако, несмотря на все усилия, решить некоторые задачи не удавалось в течение столетий. Например, безуспешно закончились многочисленные попытки найти алгоритм доказательства правильности любой теоремы на основе заданной системы аксиом.

В 1931 году австрийский математик К.Гедель доказал теорему о неполноте, смысл которой состоит в том, что в любой достаточно сложной формальной системе, основанной на аксиомах (например, в арифметике, где введены натуральные числа и операции сложения и умножения), есть утверждение, которое невозможно ни доказать, ни опровергнуть в рамках этой системы. Поэтому было высказано предположение о том, что некоторые задачи *алгоритмически неразрешимы*, то есть для них в принципе не существует алгоритма решения, и поэтому искать его бессмысленно. Чтобы строго доказать или опровергнуть эту гипотезу, нужно было ввести математическое понятие алгоритма.

"Определение", которое мы привели в начале статьи, часто называют *интуитивным*, потому что оно содержит такие

Авторы благодарят д. ф.-м. н. М.А. Ройтберга за детальное обсуждение этого материала и критические замечания, которые помогли существенно улучшить его содержание.

К.Ю. Поляков,
Е.А. Еремин

“нематематические” понятия, как “точный набор”, “инструкция”, “исполнитель”, “решение задачи”. Эти термины невозможно записать строго, используя язык математики и логики, поэтому для математического доказательства такое определение не подходит.

Исследования в этой области, которые начали активно проводиться в 30-х годах XX века, привели к возникновению *теории алгоритмов*, которая занимается

- доказательством алгоритмической неразрешимости задач;
- анализом сложности алгоритмов;
- сравнительной оценкой качества алгоритмов.

Значительный вклад в развитие теории алгоритмов внесли математики А.Тьюринг (Великобритания), Э.Пост (США), А.Чёрч (Великобритания), С.Клини (США) и А.А. Марков (СССР).

Что такое алгоритм?

Первые известные алгоритмы — это правила выполнения арифметических действий с числами. В них четко определены объекты (числа в десятичной записи) и элементарные шаги (сложить, вычесть, перемножить два однозначных числа — вспомните таблицы сложения и умножения). Постепенно сложность задач, которые решались с помощью алгоритмов, увеличивалась, и понятие “шаг алгоритма” оказалось нечетким, размытым. Например, можно ли считать элементарным шагом разложение числа на простые множители или сложение многозначных чисел?

Со временем понятие алгоритма расширилось — сейчас мы говорим об алгоритмах для исполнителей, которые работают с текстами и другими объектами реального мира. Однако оказалось, что все эти объекты можно тем или иным способом закодировать в виде цепочек символов, так что любой алгоритм сводится к преобразованию одной символьной строки в другую. Таким способом можно представить и классические вычислительные алгоритмы — операции с цифрами. В алгоритме шахматной игры объекты — это фигуры на доске, но их расположение легко закодировать в символьной форме (вспомните запись шахматных партий).

Поэтому можно рассматривать только алгоритмы обработки символьных строк, а полученные результаты будут применимы к любым алгоритмам. Как вы знаете, текст, записанный с помощью любого алфавита, всегда можно перевести в двоичный код, поэтому, вообще говоря, достаточно рассматривать только алгоритмы, работающие с двоичными последовательностями.

Итак, про любой алгоритм можно сказать следующее:

- алгоритм получает на вход дискретный объект (например, слово);
- алгоритм обрабатывает входной объект по шагам (дискретно), строя на каждом шаге промежуточные дискретные объекты; этот процесс может закончиться или не закончиться;
- если выполнение алгоритма заканчивается, его результат — это объект, построенный на последнем шаге;
- если выполнение алгоритма не заканчивается (алгоритм зацикливается), то результат его работы при данном входе не определен.

Любой алгоритм рассчитан на определенного исполнителя: он должен использовать только понятные этому исполнителю команды. Задание для исполнителя — это текст на специальном (формальном) языке, который обычно называют *программой*. Поэтому можно определить алгоритм так:

> **Алгоритм** — это программа для некоторого исполнителя.

Напомним, что с точки зрения теории алгоритмов достаточно рассматривать только алгоритмы, работающие с цепочками символов, которые называют *словами*.

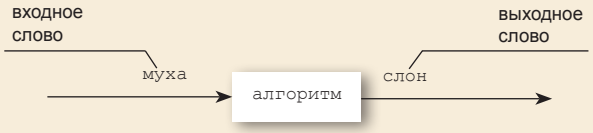


Рис. 1

Каждый алгоритм задает (вычисляет) функцию, которая преобразует входное слово в результат (выходное слово). Такая функция может быть не определена для некоторых входных слов, если алгоритм зацикливается.

Функция, заданная алгоритмом, может быть нигде не определена. Например, алгоритм

```
нц пока да
кц
```

выходит в бесконечный цикл при любом входном слове.

Алгоритмы называются *эквивалентными*, если они задают одну и ту же функцию. То есть при любом входном слове оба алгоритма должны приводить к одному и тому же результату или зацикливаться (оба алгоритма не дают никакого результата). Например, следующие алгоритмы для выбора минимального из значений переменных **a** и **b** эквивалентны:

```
если a < b то
  м:= a
иначе
  м:= b
все
```

```
м:= b
если a < b то
  м:= a
все
```

Универсальные исполнители

Как мы уже видели, понятие алгоритма оказывается “привязанным” к его исполнителю и некоторому языку программирования. Это не позволяет определить алгоритм как математический объект. Поэтому возникла идея попытаться построить универсального исполнителя.

Универсальный исполнитель — это исполнитель, для которого можно построить алгоритм, эквивалентный любому алгоритму для любого другого исполнителя.

Такого исполнителя можно было бы использовать для доказательства разрешимости или неразрешимости задач. Если удается построить алгоритм решения задачи для универсального исполнителя, то задача разрешима. Если доказано, что алгоритма не существует, то задача неразрешима. Система команд такого исполнителя должна быть как можно проще — так его будет легче использовать в доказательствах.

В середине XX века было предложено (разными учеными, независимо друг от друга) несколько исполнителей, претендующих на роль универсальных (они будут рассмотрены далее), причем в теории алгоритмов доказано, что все они эквивалентны друг другу, то есть алгоритм, который можно запрограммировать для одного универсального исполнителя, можно запрограммировать также и для остальных.

А.Тьюринг (1912–1956) (computerhistory.org)
Рис. 2

Как же связан универсальный исполнитель с проблемой строгого определения алгоритма?

Любой алгоритм может быть представлен как программа для универсального исполнителя.

Это основная идея теории алгоритмов. Строго доказать это утверждение невозможно, потому что здесь используется интуитивное понятие “алгоритм”.

Как мы увидим, каждый универсальный исполнитель описывается с помощью математических терминов, поэтому на его основе можно дать строгое определение алгоритма:

Алгоритм — это программа для универсального исполнителя.

Универсальный исполнитель — это некоторая *модель вычислений*, которая задает способ описания алгоритмов и их выполнения. Модель вычисления должна содержать

- “процессор”, задающий систему команд и способ их выполнения;
- “память”, определяющую способ хранения данных;
- язык программирования (способ записи программ);
- способ ввода данных (чтения входного слова);
- способ вывода слова-результата.

Все универсальные исполнители эквивалентны по определению. Это значит, что для любого алгоритма для одного универсального исполнителя можно построить эквивалентный алгоритм для другого универсального исполнителя. Поэтому последнее приведенное определение алгоритма фактически не зависит от конкретного исполнителя.

Машина Тьюринга

Первым предложил универсального исполнителя английский математик А.Тьюринг. Придуманное им воображаемое устройство состоит из трех частей:

- *бесконечной ленты*, разделенной на ячейки;
- *каретки* (читающей и записывающей головки);
- *программируемого автомата*.

Программируемый автомат управляет кареткой, посылая ей команды в соответствии с заложенной в него сменяемой программой. Лента выполняет роль внешней памяти компьютера, автомат — роль процессора, а каретка служит для ввода и вывода данных. Такое устройство называют машиной Тьюринга. Теоретически лента в машине Тьюринга бесконечна, однако в каждый момент работы машины времени используется лишь конечная ее часть.

Каретка в любой момент времени находится над одной ячейкой, автомат может читать и изменять содержимое этой ячейки, которая называется текущей (рабочей) ячейкой.

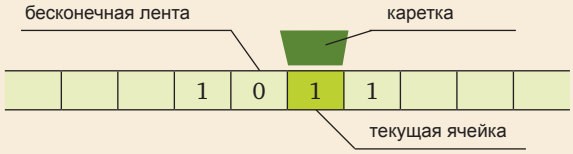


Рис. 3

В каждую ячейку ленты можно записать один любой символ, принадлежащий выбранному алфавиту. Любой алфавит обязательно содержит пробел (пустой символ, соответствующий “чистым” участкам ленты), который мы будем обозначать знаком “☐”. Алфавит обычно обозначается буквой A, а его элементы — строчными буквами a с индексами: $A = \{a_1, a_2, \ldots, a_N\}$. Например, алфавит машины Тьюринга, работающей с двоичными числами, задается в виде $A = \{0, 1, ☐\}$.

Непрерывную цепочку символов на ленте называют *словом*. На *рис.* 3 лента содержит слово “1011”, которое можно воспринимать как двоичное число.

Автоматом называют устройство, работающее без участия человека. Автомат в машине Тьюринга имеет несколько состояний и при определенных условиях переходит из одного состояния в другое. Состояние

автомата определяет ту промежуточную задачу, которую решает автомат в данный момент. Это напоминает состояния человека: ночью он спит (состояние 1), утром встает и умывается (состояние 2), завтракает (состояние 3), идет на работу (состояние 4) и т.д.

Множество всех состояний автомата обозначается буквой Q, а его элементы — строчными буквами q с индексами: $Q = \{q_1, q_2, \ldots, q_M\}$. Принято, что в начальный момент машина Тьюринга находится в состоянии q_1.

Особое состояние q_0 — это состояние останова. Если машина переходит в это состояние, выполнение программы сразу останавливается.

Автомат управляется программой. Во время каждого шага программы автомат выполняет последовательно три действия:

1) изменяет символ в рабочей ячейке на другой (или оставляет без изменений);

2) перемещает каретку влево или вправо (или оставляет на месте);

3) переходит в другое состояние (или остается в прежнем состоянии).

Поэтому при составлении программы для каждой пары (*символ, состояние*) нужно определить три параметра: символ a_i из выбранного алфавита A, направление перемещения каретки ("←" — влево, "→" — вправо, "точка" — нет перемещения) и новое состояние автомата q_k. Например, команда 1 "←" q_2 обозначает "заменить символ на 1, переместить каретку влево на одну ячейку и перейти в состояние q_2".

Пример 1. На ленте записано число в двоичной системе счисления. Каретка находится где-то над числом. Требуется увеличить число на единицу.

Для решения задач такого типа нужно:

- определить алфавит машины Тьюринга A;
- выделить простейшие подзадачи и определить набор возможных состояний Q; задать начальное состояние q_1 и конечное состояние q_0 (в котором машина останавливается);
- составить программу, то есть для каждой пары (a_i, q_k) определить команду, которую должен выполнить автомат.

Как мы уже выяснили, алфавит машины Тьюринга, работающей с двоичными числами, включает символы 0, 1 и пробел: $A = \{0, 1, \square\}$. Определим возможные состояния (разобьем задачу на элементарные подзадачи):

1) q_1 — автомат ищет правый конец слова (числа) на ленте;

2) q_2 — автомат увеличивает число на 1, проходя его слева направо, и останавливается, закончив работу.

Теперь займемся программой. На первом этапе, когда автомат ищет конец слова, его работа может быть описана так:

1) если в рабочей ячейке записана цифра 0, переместиться вправо;

2) если в рабочей ячейке записана цифра 1, переместиться вправо;

3) если в рабочей ячейке пробел, переместить каретку влево и перейти в состояние q_2.

Тогда действия автомата в состоянии q_1 можно представить в виде таблицы, где в заголовках строк записываются символы алфавита, а в заголовках столбцов — состояния:

	q_1
0	$0 \to q_1$
1	$1 \to q_1$
□	$\square \leftarrow q_2$

Рис. 4

Заметим, что во всех случаях символ под кареткой не меняется. Кроме того, состояние меняется только в последней ячейке. Поэтому для упрощения записи не будем указывать в таблице то, что остается без изменений. Так, на наш взгляд, более кратко и понятно:

	q_1
0	→
1	→
□	$\leftarrow q_2$

Рис. 5

Второй этап — увеличение двоичного числа на единицу. Это можно сделать следующим способом (вспомните тему "Системы счисления"):

1) если в рабочей ячейке записана цифра 0, записать в нее 1 и стоп;

2) если в рабочей ячейке записана цифра 1, выполнить перенос в старший разряд — записать в ячейку 0 и переместиться влево;

3) если в рабочей ячейке пробел, записать в нее 1 и стоп.

Для того чтобы остановить работу машины Тьюринга, нужно перевести ее в состояние останова q_0. Теперь можно добавить в таблицу столбец, соответствующий состоянию q_2:

	q_1	q_2
0	→	$1\ .\ q_0$
1	→	$0 \leftarrow$
□	$\leftarrow q_2$	$1\ .\ q_0$

Рис. 6

Построенная полная таблица — это и есть программа для машины Тьюринга. Обратите внимание, что мы разбили исходную задачу на подзадачи, для каждой из них составили программу, а потом их соединили. Две подзадачи связаны через ячейку $(\square, q_1)$, в которой состояние автомата изменяется на q_2. В данном простейшем случае в каждом из двух алгоритмов было использовано только одно состояние, но это не обязательно — можно таким же способом соединять и более сложные алгоритмы. Если алгоритмы А и Б можно запрограммиро-

вать на машине Тьюринга, то и любую их комбинацию тоже можно запрограммировать.

Тьюринг предположил, что

> Любой алгоритм (в интуитивном смысле этого слова) может быть представлен как программа для машины Тьюринга.

Это утверждение в теории алгоритмов известно как *тезис Чёрча — Тьюринга*.

Э.Л. Пост (1897–1954) (ru.wikipedia.org)
Рис. 7

Машина Тьюринга может быть строго задана с точки зрения математики. Алфавит A и набор возможных состояний Q могут быть записаны в виде множеств, а программа — в виде пятерок вида $(a_i, q_k, a_j, d_{ik}, q_m)$, задающих команду "если машина находится в состоянии q_k и в рабочей ячейке записан символ a_i, то записать в рабочую ячейку символ a_j, сместиться в направлении d_{ik} и перейти в состояние q_m". Например, приведенная выше программа увеличения двоичного числа на 1, записанная в виде таких пятерок, выглядит так:

$(0, q_1, 0, \rightarrow, q_1)$, $(1, q_1, 1, \rightarrow, q_1)$, $(\square, q_1, \square, \leftarrow, q_2)$,
$(0, q_2, 1, ., q_0)$, $(1, q_2, 0, \leftarrow, q_2)$, $(\square, q_2, 1, ., q_0)$

Эта машина — математический объект, и данное на ее основе определение алгоритма может использоваться для доказательств. Едва ли можно применить машину Тьюринга для решения практических задач, но эта простая модель алгоритма очень удобна для проведения теоретических исследований. В отличие от "интуитивного" определения алгоритма новое определение не содержит таких неопределенных понятий, как "инструкция", "исполнитель", "решение задачи". Таким образом, удается дать формальное определение слова "алгоритм" (по Тьюрингу):

> **Алгоритм** — это программа для машины Тьюринга.

Машина Поста

Практически одновременно с Тьюрингом (в том же 1936 году) и независимо от него американский математик Э.Л. Пост предложил еще более простого универсального исполнителя, который позднее получил название "машина Поста".

Лента в машине Поста (так же, как и в машине Тьюринга) бесконечна и разбита на ячейки. Каждая ячейка может содержать метку (быть отмечена) или не содержать ее (пустая ячейка).

Таким образом, Пост сократил алфавит всего до двух цифр. Это допустимо, потому что любые данные можно перекодировать в двоичный код, сопоставив каждой букве исходного алфавита уникальную последовательность нулей и единиц.

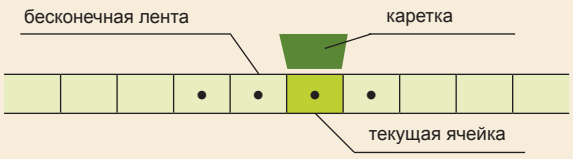


Рис. 8

Кроме того, алгоритм работы машины Поста задается не в виде таблицы, а как программа для универсального исполнителя. Система команд машины Поста содержит только 6 команд:

"←" — переместить каретку на одну ячейку влево;

"→" — переместить каретку на одну ячейку вправо;

"**0**" — стереть метку в рабочей ячейке (записать 0);

"**1**" — поставить метку в рабочей ячейке (записать 1);

"**? n_0,n_1**" — если в рабочей ячейке нет метки, перейти к строке n_0, иначе перейти к строке n_1;

"**стоп**" — остановить машину.

Попытка стереть метку там, где ее нет, или поставить метку повторно считается ошибкой, и машина аварийно останавливается.

Все строки в программе нумеруются по порядку, это необходимо для работы команды ветвления (**? n_0,n_1**). С помощью этой команды можно также строить циклы, как с предусловием, так и с постусловием. Например, следующая программа перемещает каретку влево до первой отмеченной ячейки:

```
1. ←
2. ? 1,3
3. стоп
```

После команд "←", "→", "**0**" и "**1**" можно указать номер строки, на которую нужно перейти сразу после выполнения этой команды. Например, команда

```
← 3
```

означает "переместить каретку влево и перейти на строку 3".

При работе с машиной Поста числа обычно записывают в *унарной* (единичной) системе счисления, в виде непрерывной цепочки меток нужной длины (вспомните счетные палочки в младшей школе). Например, на ленте, показанной на *рис.* 8, записано число 4.

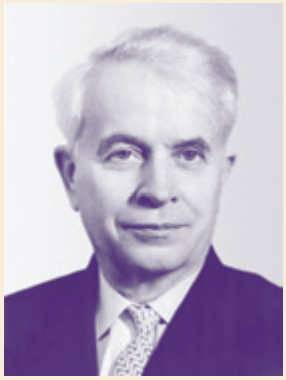
Пост предположил, что любой алгоритм может быть записан как программа для машины Поста. В теории алгоритмов доказано, что машины Поста и Тьюринга одинаковы по своим возможностям. Это значит, что круг задач, который они решают, тоже одинаков.

Нормальные алгорифмы Маркова

Советский математик А.А. Марков, который в середине XX века изучал разрешимость некоторых

задач алгебры, предложил новую модель вычислений, которую он назвал *нормальными алгорифмами*.

Нормальные алгорифмы Маркова (НАМ) — это строгая математическая форма записи алгоритмов обработки символьных строк, которую можно использовать для доказательства разрешимости или неразрешимости различных задач. Марков предположил, что любой алгоритм можно записать как НАМ. В отличие от машин Тьюринга и Поста НАМ — это "чистый" алгоритм, который не связан ни с каким "аппаратным обеспечением" (лентой, кареткой и т.п.).



А.А. Марков (1903–1979) (www.ras.ru)
Рис. 9

НАМ преобразует одно слово (цепочку символов некоторого алфавита) в другое и задается алфавитом и системой подстановок. Заметьте, что в жизни мы нередко применяем такие замены. Например, при умножении в столбик мы не вычисляем каждый раз произведение 7 × 8, а просто помним, что оно равно 56.

Пусть алфавит НАМ — это русские буквы, и задана система подстановок

а → н
ух → ло
м → с

Применим эту систему подстановок к начальному слову "**муха**". Подстановки нужно просматривать по порядку, начиная с первой. Первая подстановка означает "если в слове есть буквы **а**, заменить первую букву **а** на букву **н**". В слове **муха** есть буква "**а**", поэтому заменяем ее на "**н**". Получается "**мухн**".

Начинаем просмотр подстановок сначала. Букв "**а**" больше нет, поэтому переходим ко второй подстановке. Сочетание "**ух**" есть в слове "**мухн**", поэтому вторая подстановка срабатывает, и мы заменяем "**ух**" на "**ло**": получается "**млон**".

Теперь ни первая, ни вторая подстановки не применимы, а использование третьей дает в результате слово "**слон**". Больше ни одну подстановку сделать нельзя, и НАМ заканчивает работу. Таким образом, приведенная система подстановок преобразует слово "**муха**" в слово "**слон**".

При поиске образца рабочая цепочка символов просматривается сначала. Если в строке слово-образец встречается несколько раз, то за один шаг заменяется только первое из них. Так как на следующем шаге просмотр опять начинается с начала цепочки, после первой выполненной замены может "сработать" совсем другая подстановка.

В записи подстановок слово-образец может быть пустым, в этом случае слово-замена приписывается в начало рабочей строки:

→ 0

Такая подстановка всегда должна быть последней в списке, иначе программа зациклится: в начало слова будут постоянно дописываться все новые и новые нули.

Если после слова-замены стоит точка, после выполнения такой подстановки работа программы заканчивается. Например, если применить НАМ

а → о.

к слову "**карова**", то в результате получим "**корова**", потому что после первого же действия работа программы закончится, и последняя буква не будет заменена.

Пример 2. Построим НАМ для следующей задачи: удалить из строки, состоящей из букв *a* и *b*, первый символ. Например, строка *abba* должна быть преобразована в *bba*. Казалось бы, здесь нужно использовать систему подстановок

a → .
b → .

Однако такой НАМ будет неправильно работать для слов, начинающихся с буквы *b*, например, для слова *bba*, в котором будет удалена *последняя* буква, потому что первая подстановка выполнится раньше, чем вторая. Перестановка двух строк также не дает решения — теперь алгоритм неправильно работает для слов, начинающихся с буквы *a*. Чтобы решить эту задачу, в алфавит НАМ добавляют еще один специальный символ, например, символ "*". Этим символом помечают начало слова, используя подстановку

→ *

Полный алгоритм выглядит так:

***a → .**
***b → .**
→ *

Сначала срабатывает третья подстановка (ставим * в начало строки), затем, в зависимости от первой буквы исходного слова, работает первая или вторая подстановка, и алгоритм заканчивает работу.

Дополнительный символ похож на маркер в текстовом редакторе — он отмечает место в тексте, с которым потом будут выполняться какие-то действия.

Как показано в теории алгоритмов, любой алгоритм для машин Тьюринга и Поста можно записать как НАМ, и наоборот. Поэтому все три рассмотренных подхода к строгому определению понятия "алгоритм" эквивалентны (равносильны).

Контрольные вопросы

1. Зачем понадобилось уточнять понятие "алгоритм"?
2. Какие задачи рассматриваются в теории алгоритмов?
3. Почему можно ограничиться алгоритмами обработки символьных строк? Можно ли рассматривать только алгоритмы для преобразования двоичных кодов?
4. Как вы понимаете утверждение "алгоритм задает некоторую функцию"?

5. Как связаны понятия “алгоритм” и “исполнитель”?

6. Что такое программа?

7. В каком случае говорят, что два алгоритма эквивалентны?

8. Что такое универсальный исполнитель?

9. Сравните интуитивное и строгое понятия алгоритма.

10. Опишите устройство и систему программирования машины Тьюринга.

11. Что такое состояние машины Тьюринга?

12. Сопоставьте устройство машины Тьюринга с устройством компьютера. Какие устройства машины Тьюринга выполняют те же функции, что и аналогичные устройства компьютера?

13. В чем особенность состояний q_0 и q_1 машины Тьюринга?

14. По какому принципу можно построить программу для машины Тьюринга, которая последовательно выполняет операции А и Б?

15. Сформулируйте тезис Чёрча — Тьюринга.

16. Сравните машины Тьюринга и Поста.

17. Зачем нумеруются строки в программе для машины Поста?

18. Что такое нормальный алгорифм Маркова?

19. Зачем используют специальные символы в НАМ?

20. Что означает эквивалентность различных универсальных исполнителей?

Задачи

1. Что делают следующие программы для машины Тьюринга:

а)

	q_1
0	←
1	←
□	$\to q_0$

б)

	q_1
0	$\to q_0$
1	$\to q_0$
□	←

в)

	q_1	q_2
а	q_2	□ ←
б	q_2	□ ←
□	←	q_0

Рис. 10

В каких случаях эти программы зацикливаются?

2. Предложите программу для машины Тьюринга и начальное состояние ленты, при котором эта программа зацикливается.

3. Составьте программу для машины Тьюринга, которая уменьшает двоичное число на 1.

4. Составьте программы для машины Тьюринга, которые увеличивают и уменьшают на единицу число, записанное в десятичной системе счисления.

5. Составьте программу для машины Тьюринга, которая складывает два числа в двоичной системе, разделенные на ленте знаком “+”.

6. Составьте программы для машины Тьюринга, которые выполняют сложение и вычитание двух чисел в десятичной системе счисления.

7. *Найдите в литературе или в Интернете информацию о разновидностях машины Тьюринга.

8. Что делают следующие программы для машины Поста:

а)

1	1
2	→
3	→ 1

б)

1	←
2	? 3,4
3	1 1
4	стоп

в)

1	? 2,3
2	1 4
3	→ 1
4	стоп

Рис. 11

Как будет работать каждая из программ при различных начальных состояниях ленты?

9. Напишите программу для машины Поста, которая увеличивает (уменьшает) число в единичной системе счисления на единицу. Каретка расположена слева от числа.

10. Напишите программу для машины Поста, которая складывает два числа в единичной системе счисления. Каретка расположена над пробелом, разделяющим эти числа на ленте.

11. Напишите программу для машины Поста, которая складывает два числа в единичной системе счисления. Каретка расположена над пробелом, разделяющим эти числа на ленте.

12. Что делают следующие НАМ, если применить их к символьной цепочке, состоящей из нулей и единиц:

а)
```
0 → 00
1 → 11
```

б)
```
*0 → 0*
*1 → 1*
* → =.
→ *
```

в)
```
*0 → 00*
*1 → 11*
* → .
→ *
```

Как будет работать каждая из программ при различных начальных состояниях ленты?

13. Напишите НАМ, который “сортирует” цифры двоичного числа так, чтобы сначала стояли все нули, а потом — все единицы.

14. Дополните приведенный НАМ для удаления первого символа строки так, чтобы он не зацикливался на пустом слове.

15. Напишите НАМ, который умножает двоичное число на 2, добавляя 0 в конец записи числа.

Алгоритмически неразрешимые задачи

Вычислимые и невычислимые функции

Мы уже говорили, что любой алгоритм задает некоторую функцию, которая для каждого входного слова, к которому применим алгоритм, однозначно

задает результат — выходное слово. Такие функции называются вычислимыми.

Вычислимая функция — это функция, для вычисления которой существует алгоритм.

Любая вычислимая функция может задаваться разными алгоритмами (разными программами для выбранного универсального исполнителя). Например, следующие два нормальных алгорифма Маркова заменяют во входном двоичном слове все буквы “а” на нули и все буквы “б” на единицы:

```
а → 0      б → 1
б → 1      а → 0
```

Любая вычислимая функция может быть вычислена с помощью любого универсального исполнителя: машин Тьюринга и Поста, нормальных алгорифмов Маркова и др.

Рассмотрим, например, такую функцию, определенную для всех натуральных чисел:

$$f(n)=\begin{cases}1, & \text{если } n \text{ — четное}\\ 0, & \text{если } n \text{ — нечетное}\end{cases}$$

Попробуем составить программу для машины Тьюринга, которая вычисляет эту функцию. Будем считать, что число записано в единичной системе счисления (в виде цепочки единиц), и каретка в начальный момент стоит над самой левой единицей. Четное число мы должны заменить на одну единицу, а нечетное — на пустую ленту. Оказывается, такая программа действительно существует:

	q_1	q_2	q_3	q_4
1	$\to q_2$	$\to q_1$	$\leftarrow q_4$	$\square \leftarrow$
$\square$	$\leftarrow q_3$	$\leftarrow q_4$		q_0

Рис. 12

Как принято, в начальный момент машина находится в состоянии q_1. Затем она движется вправо вдоль числа, поочередно переходя из состояния q_1 (пройдено четное число единиц) в состояние q_2 (пройдено нечетное число единиц) и обратно. Таким образом, если встречен пробел и машина находится в состоянии q_2, то число нечетное и нужно просто стереть все единицы (состояние q_4). Если машина закончила просмотр в состоянии q_1, то число четное; при этом нужно оставить одну единицу (состояние q_3) и перейти в состояние q_4 (стереть все остальные единицы). Обратите внимание, что ячейка ($\square$, q_3) в таблице пустая — это невозможное состояние (покажите это самостоятельно).

Таким образом, рассмотренная функция вычислима, то есть ее можно вычислять с помощью машины Тьюринга, а значит, и с помощью любого универсального исполнителя. Например, нормальный алгорифм Маркова для алфавита $A = \{1\}$ выглядит так:

```
11 → ""
1 → .
→ 1.
```

В первой подстановке две соседние единицы удаляются (слово-замена здесь пустое, для ясности оно взято в кавычки, которыми можно ограничивать слова в НАМ). Это происходит до тех пор, пока не будут удалены все пары, поскольку эта подстановка стоит первой. Если остается одна единица, она удаляется с помощью второй подстановки, и работа программы заканчивается. Если все единицы удалены (число четное), то с помощью третьей подстановки мы ставим одну единицу и останавливаем автомат.

Существуют и *невычислимые функции*. Рассмотрим простой пример, предложенный В.А. Успенским в книге “Машина Поста”. Известно, что математическая постоянная π — иррациональное число, его десятичная запись бесконечна и не периодична. Введем функцию $h(n)$, которая для любого натурального числа n равна 1, если в десятичной записи числа π есть n стоящих подряд девяток, окруженных другими цифрами, и равна нулю, если такой цепочки девяток нет. Как вычислить значение этой функции при некотором заданном n? Конечно, можно вычислять друг за другом десятичные знаки числа π (такие алгоритмы математикам известны!) и проверять, не нашлась ли в полученной последовательности цифр цепочка из n девяток. С помощью такого “наивного” алгоритма можно найти такие значения n, при которых $h(n) = 1$: обнаружив требуемую цепочку, алгоритм закончит работу. Например, анализ первых 800 знаков показывает, что $h(n) = 1$ при $n = 0, 1, 2, 6$. Но если для какого-то n функция $h(n)$ равна нулю, то наивный алгоритм никогда не остановится. Более того, для этой функции вообще не существует алгоритма, который при любом n останавливается и выдает значение $h(n)$ в качестве результата. Поэтому такая функция невычислима.

Когда задача алгоритмически неразрешима?

Как вы знаете, невозможно создать вечный двигатель, потому что это противоречит универсальным физическим законам сохранения. Точно так же в математике и информатике существуют задачи, для которых общее решение не только неизвестно сейчас, но и вообще отсутствует. Поэтому искать его бесполезно.

Поскольку алгоритм работает только с дискретными объектами, любая алгоритмическая задача — это *функция*, заданная на множестве дискретных объектов (входных слов). Пусть, например, требуется по шахматной позиции определить, кто выигрывает при правильной игре — белые, черные или будет ничья. Выберем способ кодирования, при котором каждая позиция может быть закодирована словом (символьной строкой) v в подходящем алфавите. Тогда приведенной задаче может соответствовать функция $f(v)$, заданная на множестве таких слов:

$$f(v) = \begin{cases} \text{'Б'}, \text{ если } v \text{ — код позиции, в которой выигрывают белые;} \\ \text{'Ч'}, \text{ если } v \text{ — код позиции, в которой выигрывают черные;} \\ \text{'0'}, \text{ если } v \text{ — код позиции, в которой будет ничья;} \\ \text{'?'}, \text{ если } v \text{ — ошибочный код позиции.} \end{cases}$$

Если функция, соответствующая задаче, вычислима, то задача называется *алгоритмически разрешимой* — для ее вычисления можно построить алгоритм. Если определенная в задаче функция невычислима, то алгоритма для ее решения не существует.

Алгоритмически неразрешимая задача — это задача, соответствующая невычислимой функции.

В 1900 году на Международном математическом конгрессе в Париже известный математик Давид Гильберт сформулировал 23 нерешенные математические проблемы[1]. В знаменитой "десятой проблеме Гильберта" требуется найти метод, который позволяет определить, имеет ли заданное алгебраическое уравнение с целыми коэффициентами решение в целых числах. Например, уравнение $x^2 + y^3 + 2 = 0$ имеет два целочисленных решения, (5; –3) и (–5; –3). Сложность состояла в том, что требовалось найти единый метод (алгоритм), позволяющий решить задачу для *любого* такого уравнения со многими неизвестными.

В начале XX века была уверенность, что такой алгоритм есть, и поэтому его упорно искали. Однако в 1970 году советскому математику Ю.В. Матиясевичу удалось доказать, что общего алгоритма решения этой задачи не существует.

Ю.В. Матиясевич (р. 1947) (ru.wikipedia.org)
Рис. 13

Немецкий математик Г.В. Лейбниц в XVII веке безуспешно пытался найти метод проверки правильности любых математических утверждений. Как вы знаете, почти все математические теории основаны на использовании *аксиом* (положений, принимаемых без доказательства), из которых выводятся все остальные утверждения (теоремы). Задача заключалась в том, чтобы разработать алгоритм, позволяющий установить, можно ли вывести формулу Б из формулы А в рамках заданной системы аксиом (*проблема распознавания выводимости*). В 1936 году американский математик А.Чёрч доказал, что эта задача в общем виде алгоритмически неразрешима, поэтому нельзя сформулировать универсальный алгоритм, пригодный для доказательства любой теоремы[2].

А.Чёрч (1903–1995) (ru.wikipedia.org)
Рис. 14

Таким образом, уточненные определения алгоритма, основанные на понятии универсальных исполнителей, сыграли в науке очень важную роль — позволили получить *отрицательные* результаты, то есть доказать, что алгоритмов решения некоторых задач в общем виде в принципе не существует.

Для того чтобы доказать неразрешимость какой-то новой задачи, пытаются свести ее к уже известным алгоритмически неразрешимым задачам. Если это удается, значит, и новая задача алгоритмически неразрешима[3]. Существуют также задачи, про которые неизвестно, алгоритмически разрешимы они или нет — решение не найдено, но алгоритмическая неразрешимость не доказана.

Алгоритмически неразрешимые задачи встречаются не только в математике, но и в информатике, например, при разработке программ. Оказывается, невозможно написать программу для машины Тьюринга (алгоритм), которая по тексту любой программы P и ее входным данным X определяет, завершается ли программа P при входе X за конечное число шагов или зацикливается. Это так называемая *проблема останова*. Ее неразрешимость означает, в частности, что нельзя полностью автоматизировать тестирование любых программ, поручив это компьютеру. Однако для некоторых классов алгоритмов проблему останова решить можно. Например, линейная программа, не содержащая ветвлений и циклов, всегда завершится.

Было доказано, что алгоритмически неразрешима *проблема эквивалентности*: по двум заданным алгоритмам определить, будут ли они выдавать одинаковые результаты для любых допустимых исходных данных. Следовательно, невозможно полностью автоматизировать решение многих важных задач, связанных с разработкой программ, например:

- по заданному тексту программы определить, что она "делает";
- определить, "правильно" ли работает программа при любых допустимых исходных данных;
- найти ошибку в программе, работающей "неправильно".

Поэтому при отладке программы большую роль играет интуиция. Помогают (но не решают проблему полностью!) стандартные приемы, позволяющие найти ошибку:

- сравнение результатов работы программы с результатами ручного счета;
- эксперименты с программой при различных исходных данных для того, чтобы выявить закономерность появления ошибок;

[1] Сейчас большинство из них решено полностью или частично.

[2] Тем не менее отдельные классы теорем компьютер способен доказывать.

[3] Допустим, что (1) задача А неразрешима и (2) если мы можем построить алгоритм для решения задачи Б, то с его помощью можно построить алгоритм решения задачи А. Тогда задача Б тоже неразрешима.

• временное отключение (комментирование) частей программы, и др.

Поскольку многие этапы разработки программного обеспечения в принципе невозможно представить в виде алгоритмов, программирование всегда останется работой человека. Полностью поручить ее компьютеру не удастся, хотя решение некоторых задач все же можно автоматизировать.

Контрольные вопросы

1. Что такое вычислимая функция?
2. Приведите пример невычислимой функции.
3. Что такое алгоритмически неразрешимые задачи? Приведите известные вам примеры.
4. Что такое “проблема останова”? Каковы ее следствия?
5. Что такое “проблема эквивалентности”?
6. Как обычно доказывается алгоритмическая неразрешимость новых задач?

Задачи

1. Докажите, что следующая функция вычислима:

$$f(n) = \begin{cases} 1, & \text{если } n \text{ – четное} \\ 0, & \text{если } n \text{ – нечетное} \end{cases}$$

В качестве доказательства напишите программу для машины Поста.

2. Докажите, что следующая функция вычислима:

$$f(n) = \begin{cases} 1, & \text{если } n \text{ делится на 3} \\ 0, & \text{если } n \text{ не делится на 3} \end{cases}$$

В качестве доказательства напишите программы для машин Тьюринга и Поста, а также НАМ.

3. *Первой задачей, неразрешимость которой была доказана, была проблема *самоприменимости*: по заданному тексту программы P определить, останавливается ли программа P, если ей на вход подать текст этой же программы. Докажите, что проблема останова сводится к проблеме самоприменимости (именно так и была доказана неразрешимость проблемы останова).

Сложность вычислений

Что такое сложность вычислений?

Центральная задача теории алгоритмов — выяснить, существует ли алгоритм решения той или иной задачи. Если да, то возникает следующий вопрос: а можно ли им воспользоваться на практике, при современном уровне развития вычислительной техники? То есть способен ли компьютер за приемлемое время получить результат? Например, в игре в шахматы возможно лишь конечное количество позиций и, значит, только конечное количество различных партий. Значит, теоретически можно перебрать все возможные партии и выяснить, кто побеждает при правильной игре — белые или черные. Однако количество вариантов настолько велико, что современные компьютеры не могут выполнить такой перебор за приемлемое время.

Что мы хотим от алгоритма? Во-первых, чтобы он работал как можно быстрее. Во-вторых, чтобы объем необходимой памяти был как можно меньше. В-третьих, чтобы он был как можно более прост и понятен, что позволяет легче отлаживать программу. К сожалению, эти требования противоречивы, и в серьезных задачах редко удается найти алгоритм, который был бы лучше остальных по всем показателям.

Часто говорят о *временно́й сложности* алгоритма (быстродействии) и *пространственной сложности*, которая определяется объемом необходимой памяти. Поскольку память постоянно дешевеет, а быстродействие компьютеров растет медленно, мы будем рассматривать главным образом временную сложность — время выполнения программы, работающей по данному алгоритму.

В общем случае, говоря о сложности алгоритма, нужно уточнить, о каком исполнителе идет речь, какие элементарные операции мы используем. Как правило, это один из универсальных исполнителей (во многих случаях универсальным исполнителем можно считать компьютер). Временем работы алгоритма называется количество выполненных им элементарных операций T. Такой подход позволяет оценивать именно качество алгоритма, а не свойства исполнителя (например, быстродействие компьютера, на котором выполняется алгоритм).

Как правило, величина T будет существенно зависеть от объема исходных данных: поиск в списке из 10 элементов завершится гораздо быстрее, чем в списке из 10 000 элементов. Поэтому сложность алгоритма обычно связывают с размером входных данных n и определяют как функцию $T(n)$. Например, для алгоритмов обработки массивов в качестве размера n используют длину массива. Функция $T(n)$ называется *временно́й сложностью алгоритма*.

Примеры

Рассмотрим алгоритмы выполнения различных операций с массивом **A** длины n, который может быть объявлен в программе на алгоритмическом языке как

```
цел A[1:n]
```

Пример 1. Вычислить сумму первых трех элементов массива (при $n \geq 3$).

Решение этой задачи содержит всего один оператор:

```
S := A[1] + A[2] + A[3]
```

Этот алгоритм включает две операции сложения и одну операцию записи значения в память, поэтому его сложность $T(n) = 3$ не зависит от размера массива вообще.

Пример 2. Вычислить сумму всех элементов массива.

В этой задаче уже не обойтись без цикла:

```
S := A[1]
нц для i от 2 до n
  S := S + A[i]
кц
```

Здесь выполняется $n - 1$ операций сложения и n операций записи в память[4], поэтому его сложность $T(n) = 2n - 1$ возрастает линейно с увеличением длины массива[5].

Пример 3. Отсортировать все элементы массива по возрастанию методом выбора.

Напомним, что метод выбора предполагает поиск на каждом шаге минимального из оставшихся неупорядоченных значений (здесь `i`, `j`, `nMin` и `c` — целочисленные переменные):

```
нц для i от 1 до n-1
  nMin:=i;
  нц для j от i+1 до n
    если A[i] < A[nMin] то nMin:=i все
  кц
  если nMin <> i то
    c:=A[i]; A[i]:=A[nMin]; A[nMin]:= c
  все
кц
```

Подсчитаем отдельно количество сравнений и количество перестановок. Количество сравнений не зависит от данных и определяется числом шагов внутреннего цикла:

$$T_c(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n .$$

Число перестановок зависит от данных. Например, если массив уже отсортирован в нужном порядке, перестановок не будет вообще. В худшем случае на каждом шаге основного цикла происходит перестановка, всего их будет $T_p(n) = n - 1$.

Что такое асимптотическая сложность?

Допустим, что нужно выбрать между несколькими алгоритмами, которые имеют разную сложность. Какой из них лучше (работает быстрее)? Оказывается, для этого необходимо знать размер массива данных, которые нужно обрабатывать. Сравним, например, три алгоритма, сложность которых $T_1(n) = 10\,000 \cdot n$, $T_2(n) = 100 \cdot n^2$ и $T_3(n) = n^3$.

Построим эти зависимости на графике (см. *рис.* 15). При $n \le 100$ получаем $T_3(n) < T_2(n) < T_1(n)$, при $n = 100$ количество операций для всех трех алгоритмов совпадает, а при больших n имеем $T_3(n) > T_2(n) > T_1(n)$.

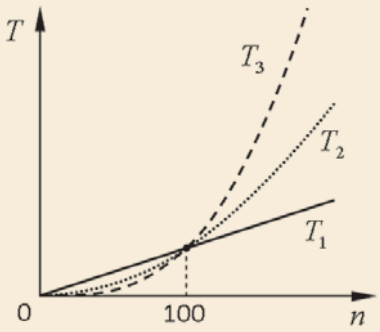


Рис. 15

Обычно в теоретической информатике при сравнении алгоритмов используется их *асимптотическая сложность*, то есть скорость роста количества операций при больших значениях n. При этом запись $O(n)$ (читается “О большое от n”) обозначает, что, начиная с некоторого значения $n = n_0$, количество операций ограничено функцией $c \cdot n$, где c — некоторая константа:

$$T(n) \le c \cdot n \text{ для } n \ge n_0.$$

Такие алгоритмы имеют *линейную сложность*, то есть при увеличении размера данных в 10 раз объем вычислений увеличивается тоже примерно в 10 раз.

Пусть, например, $T(n) = 2n - 1$, как в алгоритме поиска суммы элементов массива. Очевидно, что при этом $T(n) \le 2n$ для всех $n \ge 1$, поэтому алгоритм имеет линейную сложность.

Многие известные алгоритмы имеют *квадратичную сложность* $O(n^2)$. Это значит, что сложность алгоритма ограничена функцией $c \cdot n^2$:

$$T(n) \le c \cdot n^2 \text{ для } n \ge n_0.$$

При этом если размер данных увеличивается в 10 раз, то количество операций (и время выполнения) увеличивается примерно в 100 раз. Пример такого алгоритма — сортировка методом прямого выбора, для которой число сравнений

$$T_c(n) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \quad \text{для всех } n \ge 0.$$

> Алгоритм имеет **асимптотическую сложность** $O(f(n))$, если найдется такая постоянная c, что, начиная с некоторого $n = n_0$, выполняется условие $T(n) \le c \cdot f(n)$.

Это значит, что график функции $c \cdot f(n)$ идет выше, чем график функции $T(n)$, по крайней мере при $n \ge n_0$ (см. *рис.* 16).

Если количество операций не зависит от размера данных, то говорят, что сложность алгоритма $O(1)$, то есть количество операций меньше некоторой постоянной при любых n.

Существует также немало алгоритмов с кубичной сложностью, $O(n^3)$. При больших значениях n алгоритм с кубичной сложностью требует большего количества вычислений, чем алгоритм со сложностью $O(n^2)$, а тот, в свою очередь, работает дольше, чем алгоритм с линейной сложностью. За-

[4] Здесь и далее для упрощения выводов мы не учитываем команды, необходимые для организации цикла, потому что при больших n время их выполнения очень мало в сравнении со временем выполнения остальных операторов.

[5] Предполагается, что сложение любых двух чисел выполняется одинаковое время.

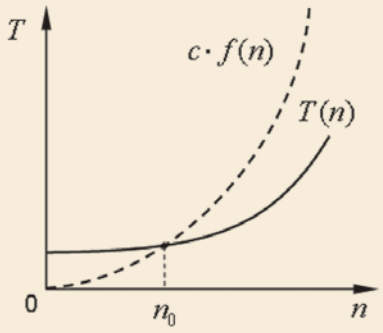


Рис. 16

метьте, что при малых значениях n все может быть наоборот, это зависит от постоянной c для каждого из алгоритмов.

Известны и алгоритмы, для которых количество операций растет быстрее, чем любой полином, например, как $O(2^n)$ или $O(n!)$. Они встречаются чаще всего в задачах оптимизации, которые решаются только методом полного перебора. Самая известная задача такого типа — это *задача коммивояжера* (бродячего торговца), который должен посетить по одному разу каждый из указанных городов и вернуться в начальную точку. Для него нужно выбрать оптимальный маршрут, при котором стоимость поездки (или общая длина пути) будет минимальной.

Еще один пример сложной задачи, которая решается только полным перебором всех вариантов — *задача выполнимости*. Дано логическое выражение, которое содержит только имена логических переменных, скобки, а также операции “И”, “ИЛИ” и “НЕ”. Требуется определить, существует ли набор значений логических переменных, при котором заданное выражение истинно.

Алгоритмы поиска

Сравним вычислительную сложность двух наиболее известных алгоритмов поиска.

Пример 4 (линейный поиск). Дан массив, в котором элементы расположены в произвольном порядке. Найти в нем заданное значение x или сообщить, что его нет.

Решение этой задачи сводится к последовательному просмотру всех элементов массива:

```
nX:=0
нц для i от 1 до n
  если A[i] = X то
    nX:=i
    выход
  все
кц
если nX > 0 то
  вывод "A[", nX, "]=", X
иначе
  вывод "Элемент не найден"
все
```

В этом алгоритме число сравнений (в худшем случае) равно $T(n) = n$, поэтому он имеет линейную сложность.

Пример 5 (двоичный поиск). Дан массив, в котором элементы упорядочены по возрастанию. Найти в нем заданное значение x или сообщить, что его нет.

В сравнении с предыдущей задачей элементы массива отсортированы, и это ускоряет решение, потому что можно применить метод двоичного поиска (дихотомии):

```
L:=1; R:=n+1
нц пока L + 1 < R
  c:=div(L+R, 2) |или c:=L + div(R-L,2)
  если X < A[c] то
    R:=c
  иначе
    L:=c
  все
кц
если A[L] = X то
  вывод "A[", L, "]=", X
иначе
  вывод "Элемент не найден"
все
```

Попробуем определить, сколько раз выполняется основной цикл при двоичном поиске. Сначала ширина интервала поиска — все n элементов массива. На каждом шаге этот интервал делится на 2, процесс завершается, когда левая и правая границы интервала совпадут. Предположим, что число элементов — это целая степень двойки, то есть $n = 2^m$. Тогда за m шагов ширина интервала сужается до 1, а на следующем шаге его границы совпадут и цикл закончится. Таким образом, количество шагов цикла равно $m + 1$. Из равенства $n = 2^m$ получаем $m = \log_2 n$, так что

$$T(n) = \log_2 n + 1.$$

Например, при $n = 2^{16}$ линейный поиск потребует в худшем случае $2^{16} = 65\,536$ сравнений, а двоичный — всего $16 + 1 = 17$ сравнений.

Таким образом, алгоритм двоичного поиска имеет асимптотическую сложность $O(\log n)$. Основание логарифма обычно не указывают, потому что выражения $\log_a n$ и $\log_b n$ отличаются на постоянный множитель (который можно включить в постоянную c):

$$\log_a n = \frac{1}{\log_b a} \cdot \log_b n.$$

Можно ли сказать, что алгоритм двоичного поиска лучше алгоритма линейного поиска? Нет! Ведь алгоритм линейного поиска применим к любым массивам данных, а алгоритм двоичного поиска — только к упорядоченным (отсортированным). А если мы сначала отсортируем массив, а потом применим к нему двоичный поиск, то общее время работы будет больше, чем при линейном поиске.

Ситуация меняется, если нам нужно многократно выполнять операцию поиска для одних и тех же данных (так, как правило, бывает при работе с базами данных). Тогда имеет смысл заранее отсортировать массив (применить "предварительную обработку данных"), а затем, используя двоичный поиск, экономить время при каждом новом поисковом запросе.

Алгоритмы сортировки

Ранее мы проанализировали один из простых методов сортировки массивов — метод прямого выбора, и выяснили, что его асимптотическая сложность $O(n^2)$. Повторим анализ для метода "пузырька", который также изучался в 10-м классе:

```
нц для i от 1 до n-1
  нц для j от n-1 до i шаг -1
    если A[j] > A[j+1] то
      c:=A[j]; A[j]:=A[j+1]; A[j+1]:= c;
    все
  кц
кц
```

На первом шаге основного цикла выполняется $n-1$ шагов внутреннего цикла, то есть $n-1$ сравнений. Далее количество сравнений уменьшается до 1, так что общее количество сравнений равно

$$T_c(n) = (n-1)+(n-2)+\ldots+2+1 = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n,$$

так же, как и у алгоритма прямого выбора. В то же время в худшем случае при каждом сравнении выполняется перестановка, что требует

$$T_p(n) = 3\cdot\frac{n(n-1)}{2} = \frac{3}{2}n^2 - \frac{3}{2}n$$

операций присваивания. Таким образом, этот алгоритм имеет асимптотическую сложность $O(n^2)$ как по числу сравнений, так и по числу присваиваний.

Существуют ли более эффективные сортировки, имеющие, например, линейную сложность? Да, для некоторых особых случаев существуют. Например, если известно, что все значения исходного массива находятся в интервале от 1 до некоторого значения MAX, можно использовать *сортировку подсчетом*. Для этого выделяется дополнительный массив счетчиков

```
цел C[1:MAX]
```

который предварительно обнуляется:

```
нц для i от 1 до MAX
  C[i]:=0
кц
```

Затем в цикле проходим весь массив с данными, и для каждого элемента `A[i]` увеличиваем счетчик `C[A[i]]`. Например, если `A[i]=20`, счетчик `C[20]` увеличивается на 1. После окончания цикла в каждом счетчике `C[i]` находится количество значений исходного массива, равных `i`.

```
нц для i от 1 до n
  C[A[i]]:=C[A[i]]+1
кц
```

Теперь остается расставить их в массиве **A** в нужном количестве. Например. Если `C[20]=5`, а значений, меньших 20, нет, в массив **A** записывается последовательно 5 значений, равных 20:

```
k:=1
нц для i от 1 до MAX
  нц для j от 1 до C[i]
    A[k]:=i
    k:=k+1
  кц
кц
```

Попробуем подсчитать количество операций для этого алгоритма. Заполнение массива `C` нулями требует **MAX** присваиваний. Цикл подсчета элементов содержит n сложений и присваиваний, то есть его сложность — линейная, $O(n)$. Наконец, последний вложенный цикл выполняет также n сложений и присваиваний (по числу элементов массива **A**), поэтому алгоритм в целом имеет *линейную сложность* по n.

Однако нужно учитывать, что принципиальное ускорение алгоритма в сравнении с предыдущими получено за счет того, что

- все значения — целые числа в ограниченном диапазоне;
- есть возможность использовать дополнительный массив размером **MAX**, который может значительно превышать размер исходного массива.

Здесь проявляется компромисс "скорость – память", который присутствует во многих задачах: ускорение алгоритма возможно за счет использования дополнительной памяти и наоборот, экономия памяти приводит к замедлению работы алгоритма.

Доказано, что в общем случае вычислительная сложность сортировки, основанной только на использовании операций "сравнить" и "переставить", не может быть меньше, чем $O(n\log n)$. Именно такую сложность имеют, например, сортировка слиянием (англ. *Merge sort*) и пирамидальная сортировка (англ. *Heap sort*), которые применяются при работе с большими наборами данных. Быстрая сортировка (англ. *Quick sort*), которая изучалась в 10-м классе, в среднем тоже имеет сложность $O(n\ \log n)$, однако в худшем случае (когда на каждом шаге массив делится на две части, одна из которых состоит из одного элемента) требуется $O(n^2)$ обменов.

Контрольные вопросы

1. Какие критерии используются для оценки качества алгоритмов?
2. Почему скорость работы алгоритма оценивается не временем выполнения, а количеством элементарных операций?
3. Как учитывается размер данных при оценке скорости алгоритма?

4. Что означают записи $O(1)$, $O(n)$, $O(n^2)$ и $O(2^n)$?

5. В каких случаях алгоритм, имеющий асимптотическую сложность $O(n^2)$, может работать быстрее, чем алгоритм с асимптотической сложностью $O(n)$?

Задачи

1. Оцените количество операций для алгоритмов:

а) поиска всех делителей числа;

б) нахождения минимального и максимального элементов массива;

в) определения количества положительных элементов массива;

г) проверки числа на простоту.

В каждом случае опишите набор используемых элементарных операций. Определите асимптотическую сложность этих алгоритмов.

2. *Предложите алгоритм, позволяющий найти и вывести на экран те символы, которые встречаются в строке более одного раза. Оцените его асимптотическую сложность.

3. *Алфавит языка племени “тумба-юмба” содержит k символов. Предложите алгоритм построения всех возможных слов этого языка длиной n символов и оцените его асимптотическую сложность.

Доказательство правильности программ

Как доказать правильность программы?

Как правило, программист разрабатывает программу на заказ, и от него требуется не только написать код, но и убедиться, что он работает правильно, то есть в соответствии с требованиями заказчика.

Очевидно, что если программа выдает неверный результат хотя бы для одного варианта входных данных, можно сразу сказать, что она некорректна, то есть содержит ошибки.

Сложнее доказать правильность программы — убедиться, что она выдает верные результаты при любых допустимых входных данных. Программисты-практики для решения этой задачи используют **тестирование**: проверяют работу программы с помощью набора тестовых данных, для которых известен правильный результат. Если полученный результат не совпадает с заданным, выполняется *отладка* программы, то есть поиск и исправление ошибок.

Однако, как писал замечательный нидерландский ученый, один из создателей современного программирования Э.Дейкстра, “отладка может показать лишь наличие ошибок и никогда их отсутствие”. В результате можно гарантировать верную работу программы только при тех данных, которые использовались в контрольных тестах. Кроме того, неясно, как определить, что все ошибки выявлены и нужно завершить отладку.

Э.В. Дейкстра (р. 1930) (http://bwnw.cwi-incubator.nl)
Рис. 17

Пример 1. Рассмотрим следующую программу для выбора максимального из трех значений, записанных в переменных **a**, **b** и **c**:

```
если a > b то M:=a иначе M:=b все
если b > c то M:=b иначе M:=c все
```

Проверяя ее на тестах `(a,b,c) = (1,2,3), (1,3,2), (2,1,3)` и `(2,3,1)` мы во всех этих случаях получаем в переменной **M** верный ответ 3. Однако это не означает, что программа правильная, так как существует контрпример `(3,2,1)`: для этого набора входных данных в переменной **M** в результате оказывается число 2.

Чтобы быть уверенным в том, что программа работает правильно при *любых* допустимых исходных данных, применяют методы **доказательного программирования**: для каждого блока программы составляются требования к входным и выходным данным, и строго доказывают, что программа всегда работает верно.

К сожалению, доказывать правильность программ не так просто, и в таких доказательствах тоже возможны ошибки. Однако при этом автор должен глубоко разобраться в алгоритме и его “подводных камнях”, и часто при этом обнаруживаются ошибки, которые могли бы проявиться уже после выпуска программы в свет.

На практике редко доказывают правильность всей программы в целом. В то же время очень полезно доказывать правильность отдельных блоков (циклов, процедур и функций) для уменьшения количества “необъяснимых” ошибок и сокращения времени отладки.

Покажем метод доказательства правильности программы на простом примере.

Пример 2. Доказать, что после выполнения следующей программы значения переменных **a** и **b** меняются местами:

```
b:=a + b     | 1
a:=b - a     | 2
b:=b - a     | 3
```

Предполагается, что сумма исходных чисел не приводит к переполнению разрядной сетки. Для удобства операторы программы пронумерованы.

Обозначим начальные значения переменных **a** и **b** через a_0 и b_0. После выполнения оператора 1 в переменной **b** будет записано значение $a_0 + b_0$. Оператор 2 записывает в переменную **a** значение $b - a = a_0 + b_0 - a_0 = b_0$. В результате выполнения оператора 3 получаем новое значение переменной **b**, равное $b - a = a_0 + b_0 - b_0 = a_0$. Таким образом, в результате выполнения программы переменные **a** и **b** будут равны b_0 и a_0 со-

ответственно, что и требовалось доказать. Поэтому приведенная программа — правильная.

Пример 3. Попробуем доказать или опровергнуть правильность уже встречавшейся ранее программы для выбора максимального из трех значений, записанных в переменных **a**, **b** и **c**:

```
если a > b то M:=a иначе M:=b все   | 1
если b > c то M:=b иначе M:=c все   | 2
```

Анализируя строку 2, выясняем, что в ней значение переменной **M** *всегда* будет изменено, то есть результат работы первой строки программы стирается, и

$$M = \begin{cases} b, & b > c \\ c, & c \geq b \end{cases}$$

Конечно, эта величина не совпадает с определением максимального значения из **a**, **b** и **c**. Таким образом, программа неправильная: она выдает неверное значение, если максимальное из трех чисел хранилось в переменной **a**. Контрпример мы уже приводили: `(3,2,1)`.

Алгоритм Евклида

Теперь докажем, что один из древнейших известных алгоритмов — *алгоритм Евклида* — действительно вычисляет наибольший общий делитель (НОД) двух натуральных чисел (мы рассматривали его в 10-м классе).

Алгоритм Евклида. Пусть заданы два натуральных числа m и n, причем $m > n$. Для вычисления НОД(m, n) следует многократно заменять большее число остатком от деления большего на меньшее до тех пор, пока меньшее число не станет равным нулю. Тогда оставшееся ненулевое число и есть НОД(m, n).

Программа, основанная на алгоритме Евклида, может выглядеть, например, так (здесь **a**, **b** и **r** — целочисленные переменные):

```
a:=m; b:=n      | 1  НОД(a,b) = НОД(m,n)
нц пока b <> 0  | 2
  r:=mod(a, b)  | 3
  a:=b; b:=r    | 4 НОД(a,b) = НОД(m,n)
кц              | 5
вывод a     | 6 НОД(a,b) = НОД(m,n), b = 0
```

Докажем, что в результате этого алгоритма в переменной **a** находится НОД(m, n).

В строке 1 исходные значения копируются из переменных m и n соответственно в переменные a и b. Очевидно, что при этом выполнено условие НОД(a, b) = НОД(m, n).

На каждом шаге цикла (в строках 3–4) вычисляется остаток r от деления a на b, и пара (a, b) заменяется на пару (b, r). Какими свойствами обладают полученные значения b и r?

Поскольку r — это остаток от деления a на b, до выполнения строк 3–4 было справедливо равенство $a = bp + r$, где p — некоторое целое число. Тогда, если a и b имеют общий делитель, то такой же делитель имеет и r. Следовательно, НОД(b, r) = НОД(a, b) = НОД(m, n). Это значит, что условие НОД(a, b) = НОД(m, n) по-прежнему выполняется после каждого шага цикла.

Поскольку остаток r с каждым шагом строго уменьшается, в конце концов он станет равным нулю и запишется в переменную **b** при выполнении строки 4. Цикл сразу же закончится, поскольку нарушается условие его выполнения. После завершения работы цикла условие НОД(a, b) = НОД(m, n) по-прежнему выполняется, но, кроме того, $b = 0$. Отсюда следует, что a = НОД(m, n).

Инвариант цикла

Таким образом, для алгоритма Евклида существует условие НОД(a, b) = НОД(m, n), которое остается справедливым на протяжении всего выполнения алгоритма: перед началом цикла, после каждого шага цикла и после окончания работы цикла. Такое условие называется *инвариантом цикла* (англ. *invariant* — неизменный).

> **Инвариант цикла** — это соотношение между значениями переменных, которое остается справедливым после завершения любого шага цикла.

Как мы увидели, выделение инварианта цикла в явном виде очень важно для доказательства правильности программы. Поэтому полезно задумываться об инварианте уже при разработке циклических алгоритмов. Таким образом, мы делаем первый шаг к доказательству их правильности и избегаем многих возможных ошибок на начальной стадии. Как писал академик А.П. Ершов, один из первых теоретиков программирования в СССР, “программиста бьют по рукам, если он посмеет написать оператор цикла, не найдя перед этим его инварианта”.

Рассмотрим несколько примеров.

Пример 1. Двое играют в следующую игру: перед ними лежат в ряд $N + 1$ камней, сначала N белых, и в конце цепочки — один черный. За один ход каждый может взять от 1 до 3 камней. Проигрывает тот, кто берет черный (“несчастливый”) камень.

Начнем анализ с простейших случаев. Если $N = 0$, то первый игрок проиграл, он может взять только черный камень. Если $N = 1, 2, 3$, то, наоборот, при правильной игре проигрывает второй игрок, потому что первый может забрать все камни, кроме черного. Вариант $N = 4$ снова приводит к проигрышу первого, потому что забрать все белые камни он не может, а после его хода второй оставит только черный камень. Также проигрышными будут позиции при $N = 8, 12, 16, \ldots$, то есть при любых значениях N, которые делятся на 4.

Таким образом, для своего выигрыша игрок должен каждым своим ходом восстанавливать *инвариант*: число оставшихся белых камней должно быть кратно 4. Если инвариант выполнен в начальной позиции, положение проигрышное и первый игрок

может надеяться только на ошибку соперника.

Пример 2. Пусть задан массив **A** длины n. Найдем инвариант цикла в программе суммирования элементов массива:

```
S := 0
нц для i от 1 до n
  S := S + A[i]
кц
```

Здесь на каждом шаге к переменной **S** добавляется элемент массива **A[i]**, так что при любом **i** после окончания очередного шага цикла в **S** накоплена сумма всех элементов массива с номерами от 1 до **i**. Это и есть инвариант цикла. Поэтому сразу можно сделать вывод о том, что после завершения цикла в переменной **S** будет записана сумма всех элементов массива.

А.П. Ершов (1931–1988) (inf.1september.ru)
Рис. 18

Аналогично можно показать, что в алгоритме поиска наименьшего значения в массиве

```
min := A[1]
нц для i от 2 до n
  если A[i] < min то
    min:= A[i]
  все
кц
```

инвариант формулируется так: после выполнения каждого шага цикла в переменной **min** записан минимальный из первых **i** элементов. Отсюда сразу следует, что после завершения цикла (при **i = n**) в этой переменной будет минимальный из всех элементов.

Пример 3. Для того же массива найдем инвариант цикла в программе сортировки элементов массива методом “пузырька”:

```
нц для i от 1 до n-1
  нц для j от n-1 до i шаг -1
    если A[j] > A[j+1] то
      c:=A[j]; A[j]:=A[j+1]; A[j+1]:= c;
    все
  кц
кц
```

До начала алгоритма элементы расположены произвольно. На каждом шаге внешнего цикла на свое место “всплывает” один элемент массива. Поэтому инвариант этого цикла можно сформулировать так: “после выполнения i-го шага цикла первые **i** элементов массива отсортированы и установлены на свои места”.

Теперь построим инвариант внутреннего цикла. В этом цикле очередной “легкий” элемент поднимается вверх к началу массива. Перед первым шагом внутреннего цикла элемент, который будет стоять на i-м месте в отсортированном массиве, может находиться в любой ячейке от **A[i]** до **A[n]**. После каждого шага его “зона нахождения” сужается на одну позицию, так что инвариант внутреннего цикла можно сформулировать так: “элемент, который будет стоять на i-м месте в отсортированном массиве, может находиться в любой ячейке от **A[i]** до **A[j]**”. Очевидно, что когда в конце этого цикла **j = i**, элемент **A[i]** встает на свое место.

В предыдущих примерах мы определяли инвариант готового цикла. Теперь покажем, как можно строить цикл с помощью заранее выбранного инварианта.

Пример 4. Рассмотрим алгоритм быстрого возведения в степень, основанный на использовании операций возведения в квадрат и умножения. Он использует две очевидные формулы:
(1) $a^k = a^{k-1} \cdot a$ при нечетной степени k
и (2) $a^k = (a^2)^{k/2}$ при нечетной степени k.

Покажем, как работает алгоритм на примере возведения числа a в степень 7:

$$a^7 = a^6 \cdot [a] = (a^2)^3 \cdot [a] = (a^2)^2 \cdot [a^2 \cdot a] =$$
$$= (a^4)^1 \cdot [a^2 \cdot a] = (a^4)^0 \cdot [a^4 \cdot a^2 \cdot a] = [a^4 \cdot a^2 \cdot a].$$

Здесь поочередно применяются первая и вторая формулы. Заметим, что на каждом этапе выражение a^n можно представить в виде $a^n = b^k \cdot p$, где через p обозначена часть, взятая выше в квадратные скобки. Если нам каким-то образом удастся уменьшить k до нуля, сохранив это равенство, то мы получим $a^n = p$, то есть задача будет решена, а результат будет находиться в переменной p.

Таким образом, равенство $a^n = b^k \cdot p$ можно использовать как инвариант цикла. Для того чтобы обеспечить выполнение этого равенства в начальный момент, можно принять, например, $b = a$, $k = n$ и $p = 1$. Далее в цикле применяются формулы (1) и (2) (в зависимости от четности k на данном шаге). Цикл заканчивается, когда $k = 0$. В результате получаем следующее решение:

```
b:=a; k:=n; p:=1
нц пока k <> 0
  если mod(k,2)=0 то
    k:=div(k,2)
    b:=b*b
  иначе
    k:=k-1
    p:=b*p
  все
кц
вывод p
```

Заметим, что инвариант цикла $a^n = b^k \cdot p$ выполняется до начала цикла, после каждого шага, а также после завершения цикла. Таким образом, мы написали код программы и одновременно доказали правильность этого блока.

Доказательное программирование

Для доказательства правильности программы необходимо иметь спецификацию — точное описание того, что должно быть сделано в результате работы программы.

Спецификация — точная и полная формулировка задачи, содержащая информацию, необходимую для построения алгоритма ее решения.

На практике спецификации программ обычно формулируют на естественном языке, в котором слова могут иметь несколько разных значений. Для строгого доказательства желательно, чтобы спецификация была задана в формальном виде, с помощью формул или соотношений между величинами.

По предложению английского ученого Ч.Хоара, спецификация записывается в форме {Q}S{R}, где Q — начальное условие (*предусловие*), S — программа и R — утверждения, описывающие конечный результат (*постусловие*). Запись {Q}S{R} означает следующее: “если выполнение программы S началось в состоянии, удовлетворяющем Q, то гарантируется, что оно завершится через конечное время в состоянии, удовлетворяющем R”.

Корректная программа — это программа, соответствующая спецификации.

Если для исходных данных не удовлетворяется предусловие Q, программа должна сообщать об этом пользователю и закончить работу. В этом случае программу называют *надежной*.

Надежная программа — это программа, которая корректна и, кроме того, не завершается аварийно при недопустимых входных данных.

Например, для алгоритма Евклида условия Q и R могут выглядеть так:

Q: $m > n > 0$, R: $a = \text{НОД}(m, n)$

— а для программы суммирования элементов массива `A[1:n]` так:

Q: $n > 0$,

$$\text{R: } s = \sum_{i=1}^{n} A[i] = A[1] + A[2] + \ldots + A[n]$$

Спецификации могут (и должны) быть составлены не только для программы в целом, но и для ее отдельных блоков (процедур, функций, циклов и т.д.). Полезно вносить утверждения Q и R прямо в текст программы. Построенная таким образом *аннотированная программа* — это еще один шаг к доказательному программированию.

Ч.Хоар разработал специальный аппарат, позволяющий доказывать правильность программы на основе спецификаций отдельных блоков. Приведем простейшие правила преобразования:

- если {Q}S{P} и P ⇒ R (из истинности P следует истинность R), то {Q}S{R};
- если {Q}S{P} и R ⇒ Q, то {R}S{P};
- если программа S — это последовательное выполнение блоков S_1 и S_2, для которых выполняются спецификации {Q}S_1{P} и {P}S_2{R}, то выполняется спецификация {Q}S{R}.

Доказательство правильности программ используют в двух ситуациях:

- доказывают правильность готовых программ (*верификация* программ);
- строят программы одновременно с доказательством их правильности (синтез программ).

Как правило, верификация — это очень трудоемкий и сложный процесс, и оказывается значительно проще использовать доказательства правильности во время разработки программы. При этом программы получаются проще, эффективнее и значительно надежнее.

Контрольные вопросы

1. Зачем нужно доказывать правильность программ?
2. Расскажите о двух подходах к проверке правильности программ.
3. Почему с помощью тестирования сложно доказать правильность программы? В каких случаях это все же можно сделать?
4. Что изменится в доказательстве алгоритма Евклида, если m и n — это произвольные натуральные числа (неравенство $m > n$ может не выполняться)?
5. Что такое инвариант цикла?
6. Зачем нужно определять инвариант цикла?
7. Что такое спецификация? Почему желательно формулировать ее в виде формальных утверждений, а не на естественном языке?
8. Что такое предусловие и постусловие?
9. Объясните запись {Q}S{R}.
10. Какая программа называется корректной?
11. В чем различие между надежной и корректной программами?
12. Как вы думаете, можно ли назвать корректной программу, которая зависает при неверных исходных данных? Обсудите этот вопрос в классе.
13. Что такое верификация программы?
14. Как вы думаете, что сложнее — доказывать правильность готовой программы или сразу писать программу, доказывая правильность отдельных блоков? Почему? Обсудите этот вопрос в классе.

Задачи

1. Докажите, что следующие операторы дают одинаковый результат при любых значениях **L** и **R** (рассмотрите четные и нечетные значения обеих переменных):

```
c:= div(L+R,2)
```

```
c:= L + div(R-L,2)
```

Какие достоинства и недостатки есть у каждого метода вычисления этой величины?

2. Докажите, что в результате выполнения следующего фрагмента программы в переменной **M** не всегда будет записано максимальное из трех чисел (**a**, **b** и **c**):

```
M:=a
если b > a то M:=b все
если c > b то M:=c все
```

Приведите контрпример, то есть такие значения **a**, **b** и **c**, при которых значение **M** будет отличаться от $\max(a, b, c)$. Как можно исправить эту программу, заменив в ней всего один символ?

3. Докажите или опровергните правильность программы для выбора максимального из трех значений, записанных в переменных **a**, **b** и **c**:

```
если a > b то M:= a
иначе если b > c то M:= b
иначе если c > a то M:= c
все; все; все
```

Если это программа некорректная, приведите контрпример. Может ли быть, что при каких-то входных данных значение переменной **M** будет неопределенным?

4. Докажите, что следующий фрагмент программы правильно сортирует значения в переменных **a**, **b** и **c** по возрастанию, то есть всегда получается **a ≤ b ≤ c**:

```
если a > b то поменять(a, b) все
если b > c то поменять(b, c) все
если a > b то поменять(a, b) все
```

Алгоритм **поменять** меняет местами значения переменных-параметров.

5. В игре “ним” двое игроков по очереди берут камни из двух кучек. За один ход можно взять любое ненулевое количество камней, но только из одной кучки. Тот, кому не осталось камней, проигрывает. Как определить, кто выиграет при правильной игре? Какой инвариант обеспечивает выигрыш?

6. Определите инварианты для следующих циклов. Что будет вычислено в переменной **b**?

а)
```
k := 0; b := 1;
while k < n do begin
  k := k + 1;
  b := b * a;
end;
```

б)
```
k := n; b := 1;
while k > 0 do begin
  k := k - 1;
  b := b * a;
end;
```

7. Определите предусловие и постусловие для алгоритмов

а) нахождения суммы всех делителей числа;

б) проверки числа на простоту;

в) определения количества слов в символьной строке;

г) двоичного поиска элемента в отсортированном массиве;

д) перестановки элементов массива в обратном порядке;

е) преобразования числа из символьной записи в значение целого типа.

8. Предложите другие начальные значения переменных b, k и p в алгоритме быстрого возведения в степень. Инвариант цикла должен сохраниться.

9. Оцените сложность алгоритма быстрого возведения в степень при $n = 2^m$.

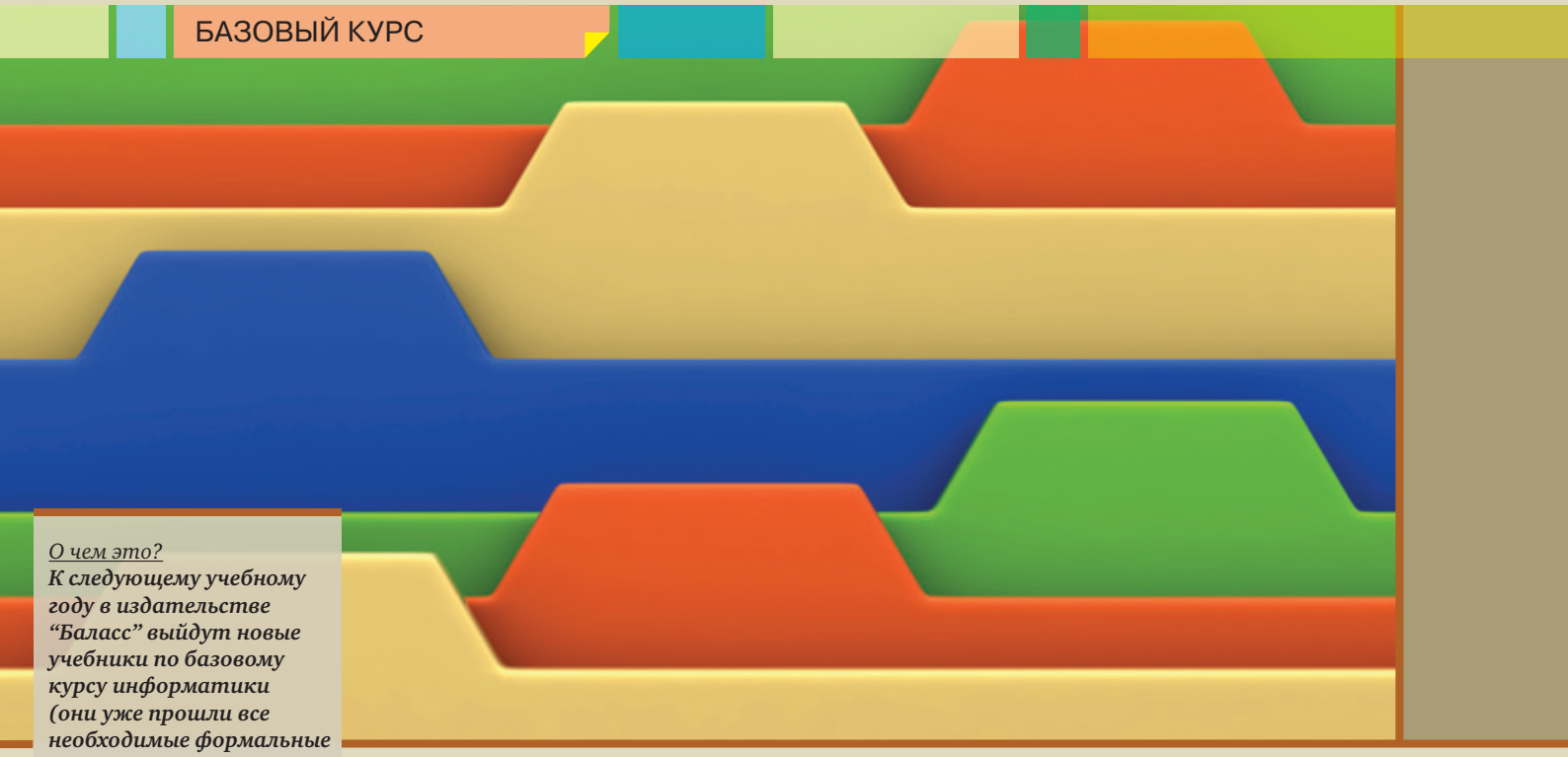
Самое важное

- Интуитивное понятие алгоритма, которое мы использовали ранее, непригодно для математического доказательства неразрешимости задач.
- Входные и выходные данные любого алгоритма можно закодировать в виде последовательностей символов некоторого алфавита (и даже двоичного алфавита).
- Про любой алгоритм можно сказать следующее:
  - алгоритм получает на вход дискретный объект (например, слово);
  - алгоритм обрабатывает входной объект по шагам (дискретно), строя на каждом шаге промежуточные дискретные объекты; этот процесс может закончиться или не закончиться;
  - если выполнение алгоритма заканчивается, его результат — это объект, построенный на последнем шаге;
  - если выполнение алгоритма не заканчивается (алгоритм зацикливается), то результат его работы при данном входе не определен.
- Алгоритм — это программа для некоторого исполнителя.
- Универсальный исполнитель — это исполнитель, для которого можно построить алгоритм, эквивалентный любому алгоритму для любого другого исполнителя. Все универсальные исполнители эквивалентны между собой.
- Каждый алгоритм задает (вычисляет) функцию, которая преобразует входное слово в результат (выходное слово). Алгоритмы называются эквивалентными, если они задают одну и ту же функцию.
- Вычислимая функция — это функция, для вычисления которой существует алгоритм. Любая вычислимая функция может задаваться разными алгоритмами.
- Алгоритмически неразрешимая задача — это задача, соответствующая невычислимой функции.
- Говорят, что алгоритм имеет асимптотическую сложность $O(f(n))$, если найдется такая постоянная c, что, начиная с некоторого $n = n_0$, выполняется условие $T(n) \le c \cdot f(n)$.
- Задачи оптимизации, которые решаются только полным перебором вариантов, могут иметь, например, асимптотическую сложность $O(2^n)$ или $O(n!)$. Эти функции при больших n возрастают быстрее, чем многочлен любой степени.
- Чтобы обеспечить надежность программы, используют методы доказательного программирования: разработка программы ведется одновременно с доказательством ее правильности.
- Инвариант цикла — это соотношение между величинами, которое остается справедливым после завершения любого шага цикла.

Список литературы

1. *Андреева Е.В., Босова Л.Л., Фалина И.Н.* Математические основы информатики. Учебное пособие. М.: Бином, 2005.

2. *Успенский В.А.* Машина Поста. М.: Наука, 1988.



О чем это?
К следующему учебному году в издательстве “Баласс” выйдут новые учебники по базовому курсу информатики (они уже прошли все необходимые формальные процедуры). О новых учебниках рассказывает руководитель авторского коллектива.

И это все?!
Нет . В следующей статье читайте фрагмент из нового учебника для 9-го класса – два первых урока по теме “Базы данных”.

Информатика во взрослых играх и жизненных задачах

А.В. Горячев, координатор направления “Информатика” в образовательной системе “Школа 2100”, канд. пед. наук

Много воды утекло с начала 90-х, когда вышло первое, еще черно-белое издание “Информатики в играх и задачах”. За это время наша информатика обрела цвет, дополнительные элементы УМК (плакаты и новые учебники: “Логика и алгоритмы” и “Мой инструмент — компьютер”), компьютерную поддержку в единой коллекции ЦОР, но самое главное наше обретение — это большое число учителей-единомышленников. И уже много лет наши учителя спрашивают нас, когда же мы появимся в основной школе.

Еще одно важное событие произошло с нами за эти годы — мы с нашими учебниками влились в ряды Образовательной системы “Школа 2100”. Вместе с авторами учебников по другим предметам мы обсуждали наши взгляды на цели образования, на технологии воплощения массового развивающего образования, вместе разрабатывали способы диагностирования общих учебных умений и их отражение в новых дневниках и других изданиях, вместе получали премию правительства России в области образования в 2008 году. И уже много лет наши коллеги — авторы учебников по другим предметам — спрашивают нас, когда же мы появимся в основной школе.

Когда мы уже начали уставать отводить глаза и как бы незаметно менять тему разговора, появился новый ФГОС — Федеральный государственный образовательный стандарт. И тогда для нас слились вместе и идеи “Школы 2100”, и положения нового стандарта, и наши собственные взгляды на обучение информатике. Мы разработали концепцию нового учебника информатики и наметили пути реализации этой концепции.

Идеи образовательной системы “Школа 2100”

Цель обучения в образовательной системе “Школа 2100” — вырастить функционально грамотную личность. У нас есть более полное определение, что это такое — функционально грамотная личность, и упрощенное, “для родителей”. Это личность, которая способна исполь-

зовать все постоянно приобретаемые в течение жизни знания, умения и навыки для решения максимально широкого диапазона жизненных задач в различных сферах человеческой деятельности, общения и социальных отношений. Это человек, ориентирующийся в мире и действующий в соответствии с общественными ценностями, ожиданиями и интересами. И попроще: это человек, способный решать самые разные, возникающие в жизни задачи, оставаясь при этом достойным человеком.

В “Школе 2100” для каждого учебного предмета формулируются линии развития — качества личности, развиваемые средствами этого учебного предмета. Как правило, это освоение каких-либо обобщенных видов действий. Например, для математики это умение создавать математические модели, выполнять точные вычисления при конструировании и т.д. На первое место по значимости выводятся умения выполнения действий, а знания рассматриваются как необходимая основа для выполнения этих действий.

Каждый предмет вносит свой вклад в выращивание функционально грамотной личности. Информатика растит Homo Informaticus — человека, успешно действующего в условиях информационного общества, умеющего применять средства ИКТ в качестве инструмента для достижения своих целей — от коммуникации до моделирования и автоматизации своей деятельности.

Положения нового стандарта

В новом образовательном стандарте акценты смещены с минимума содержания на требования к результату образования. Предметные программы рассматриваются только как составная часть основной образовательной программы школы. При этом практически все системы учебников из федерального комплекта предлагают свои примерные основные образовательные программы. Такую примерную образовательную программу предложила школам и образовательная система “Школа 2100”. Новые понятия из нового стандарта появляются и в проекте нового закона об образовании (они же должны друг другу соответствовать). В этом проекте закона вы уже не найдете такого термина из предыдущего стандарта, как “примерная программа по предмету”. На ее место встают примерные основные образовательные программы, включающие программы по предметам, причем таких примерных основных образовательных программ может быть несколько.

Результаты обучения, помимо традиционных предметных, дополнены личностными и метапредметными, среди которых мы видим получившие дальнейшее развитие три основных вида универсальных учебных действий из стандарта для начальной школы: познавательные, регулятивные и коммуникативные.

В самом стандарте информатика представлена в новой предметной области “Математика и информатика”. Но мы обращаем внимание и на отнесенные в стандарте к метапредметным результатам обучения формирование и развитие ИКТ-компетентности, включая владение информационно-коммуникационными технологиями, поиском, построением и передачей информации, презентацией выполненных работ, основами информационной безопасности, умением безопасного использования средств информационно-коммуникационных технологий и сети Интернет.

Наши взгляды на обучение информатике

Во-первых, мы видим, что на уроках по другим учебным дисциплинам нет такого противоречия между общеобразовательным и профильным (точнее, предпрофильным) содержанием. Предпрофильная физика, химия, биология, математика — это углубленная (а иногда и расширенная) базовая физика, химия, биология, математика. Во всех этих дисциплинах мы выступаем пользователями этого мира. Появление в школе предмета “Информатика” связано с изучением использования рукотворного мира — мира компьютерной техники, поэтому профильное обучение информатике — это не углубленное изучение использования, а обучение созданию и развитию этого рукотворного мира. Конечно, можно представить профильный курс с углубленным изучением информационных аспектов всего мира (рукотворного и нерукотворного), но содержание профиля всегда определяется профильными вузами. Поэтому на уроках по другим дисциплинам и базовое, и профильное обучение — это обучение жителей мира вокруг нас (на разных уровнях), а на уроках информатики базовый уровень — это обучение жителей компьютерного мира вокруг нас, а профильный уровень — это обучение его создателей. *Обучение уверенного водителя и обучение конструктора автомобилей — это совершенно разные вещи.* При этом совершенно нелишне конструктору уметь водить, а разработчику техники и программ уметь применять средства ИКТ в своей профессии и вне ее. Закрывать глаза на это противоречие, на наш взгляд, неправильно. Так же как неправильно, копируя модели других предметов, считать наш базовый курс облегченным вариантом профильного. Это особенность предмета “Информатика”, и эту специфику надо спокойно принимать и учитывать в своих рассуждениях и действиях.

Во-вторых, мы рассматриваем известный тезис родоначальника школьной информатики Андрея Петровича Ершова “Программирование — вторая грамотность” как призыв к подготовке широкого круга компьютерных пользователей, потому что в то время программирование было единственным способом использования компьютера.

В-третьих, после долгих расспросов учителей информатики мы точно поняли, какая именно информатика им нужна — разная. Причем не только разная для разных регионов и школ, но порою разная для 9-го “А” и 9-го “Б” в одной школе. Мы поняли, что учитель информатики должен иметь свободу формирования учебного плана, но при этом он же должен иметь возможность использовать разработанные для него учебно-методические материалы. Это противоречие разрешимо.

Спроектированный нашим авторским коллективом комплект учебников для 7–9-х классов основной школы обладает следующими характеристиками.

Модульная структура

Учебники состоят из отдельных взаимозаменяемых модулей. Для обеспечения их взаимозаменяемости введен единый квант — восемь уроков. Большинство модулей включают восемь минимально необходимых уроков и восемь дополнительных. Конечно, для полноценного изучения темы модуля лучше брать все шестнадцать уроков.

Компетентностная ориентация и разноуровневое обучение

Все модули нацелены на освоение каких-либо умений, причем умений диагностируемых. В каждых восьми уроках есть две диагностические работы: промежуточная на пятом уроке и итоговая на восьмом. Эти работы разделены работой над ошибками и трехуровневым закреплением освоенных умений. Результат промежуточной диагностической работы задает уровень, начиная с которого ученик начинает работу над исправлением ошибок и закреплением освоенных умений.

Гибкое сочетание общеобразовательного и предпрофильного содержания

Модули разнесены на две группы, условно называемые нами “общеобразовательные” и “предпрофильные и профориентационные”. По нашим замыслам модули из первой группы предназначены для освоения умений применять средства ИКТ в качестве инструмента при достижении своих целей. В первую очередь мы хотели бы ориентироваться на применение средств ИКТ в универсальных действиях. Новый стандарт подсказал нам структуру таких универсальных действий:

- познавательные (например, поиск информации, моделирование, применение интеллект-карт),
- коммуникативные (например, непосредственная коммуникация: общение в сети, публичные выступления, и опосредованная коммуникация: создание печатных, мультимедийных и электронных изданий),
- регулятивные (например, управление личными проектами, тайм-менеджмент).

Во вторую группу мы выделили модули, относящиеся к следующим темам:

- теоретические основы информатики (системы счисления, мат. логика),
- программирование (сквозная линия с 7-го по 9-й класс),
- основы профессионального мастерства (основы дизайна и печати изображений, основы издательской деятельности, веб-конструирование).

Жизненные задачи и проекты

Они есть в каждом модуле. Жизненные задачи — это проблемы, с которыми школьники могут столкнуться в жизни и для решения которых им понадобятся изучаемые знания и умения. Жизненные задачи оформлены так:

Название

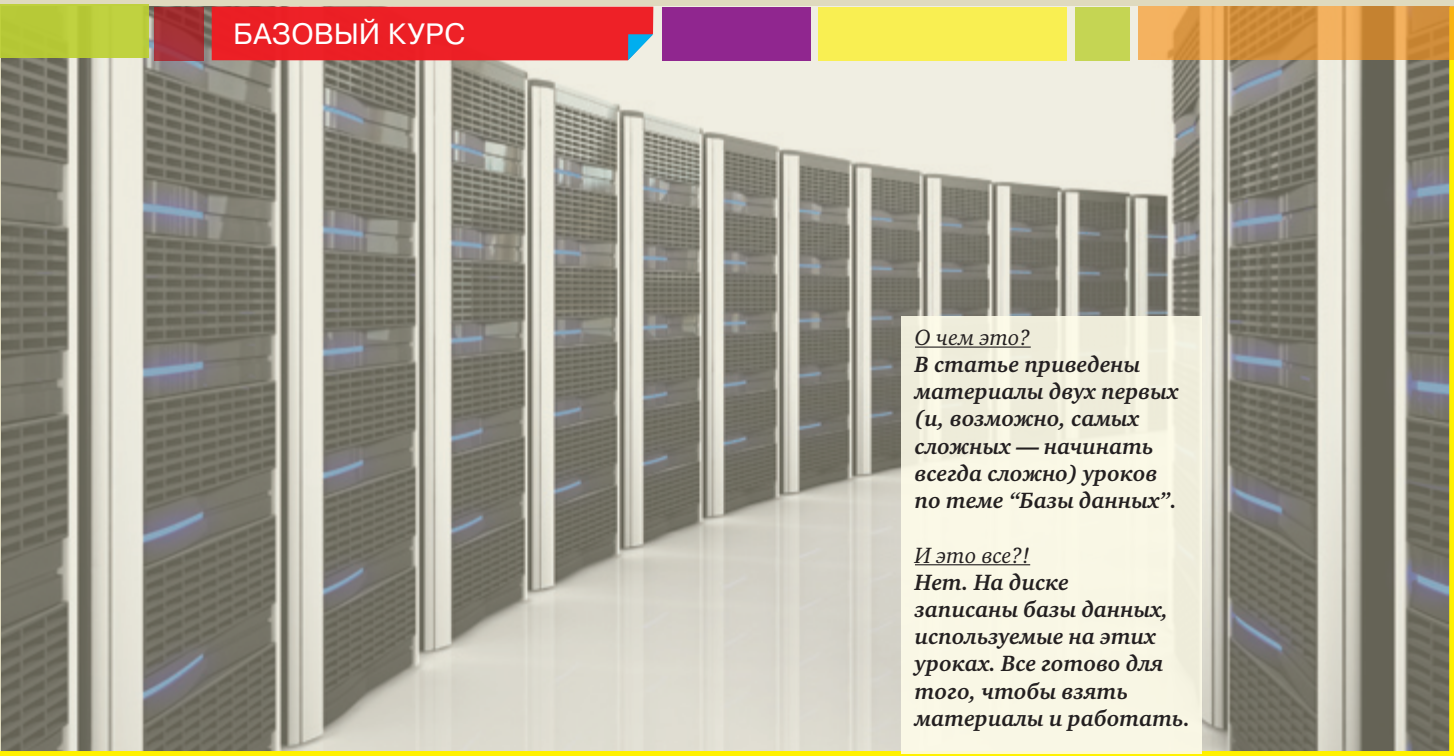
Ситуация. Условия, в которых возникла проблема.

Роль. Человек, в роли которого ученики должны себя представить, решая проблему.

Результат. То, что нужно получить в итоге.

Под проектами мы понимаем любое самостоятельное дело, которое предполагает оригинальный замысел (цель), выполнение работы за определенный отрезок времени и конкретный результат, представленный в итоге (предметы, сделанные своими руками, мероприятия, решение проблемы, результаты самостоятельных исследований и др.).

Мы полагаем, что учебники, разработанные по этим принципам, будут отвечать положениям нового ФГОС, идеям образовательной системы “Школа 2100” и дадут ответы на вопросы, которые ставит перед учителями современная информатика. Наш принцип: все, что вам надо, найдется в наших учебниках, а если не найдется — скажите нам, и мы добавим еще модуль.



О чем это?
В статье приведены материалы двух первых (и, возможно, самых сложных — начинать всегда сложно) уроков по теме “Базы данных”.

И это все?!
Нет. На диске записаны базы данных, используемые на этих уроках. Все готово для того, чтобы взять материалы и работать.

Тема “Базы данных”: два первых урока

С.Л. Островский,
Москва

Предлагаем вниманию коллег материалы двух первых уроков по теме “Базы данных” в 9-м классе. Отметим, что хотя эти материалы соответствуют параграфам нового учебника, они являются “самодостаточными”. Приведенные в них примеры (их даже можно назвать задачами и задачками — особенно, когда речь пойдет о ключах) можно использовать, по какому бы учебнику (или вообще без оного ☺) вы ни работали.

Урок 1. Базы данных и системы управления базами данных. Табличные базы данных

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Мы покупаем билеты на поезд или самолет. Кассир быстро подбирает нам удобные даты, места, предлагает различные варианты пересадок.

Мы оплачиваем мобильный телефон. Платежный терминал сразу после ввода номера обязательно проверит, имеется ли такой номер телефона в принципе, не ошиблись ли мы.

Мы ищем в web-интерфейсе своей электронной почты все письма от данного адресата. Результат появляется моментально.

Какой вопрос у вас возникает? *Как организовывают и хранят большие объемы цифровых данных так, чтобы их можно было быстро обрабатывать?*

РЕШЕНИЕ ПРОБЛЕМЫ

Организовывай и властвуй!

Всем нам из жизненного опыта известно, что способ организации любых данных существенно влияет на производительность их обработки. Если вы знаете, что тетрадки у вас находятся в левом верхнем ящике стола, дневник там же, учебники — на первой полке книжного шкафа, ручки и карандаши — в пенале, а пенал всегда лежит в первом отделении рюкзачка, вы утром быстро — за секунды — соберетесь в школу. А вот если минут пять искать нужную тетрадку, потом еще минут пять дневник, потом… ну, ладно, сегодня не буду брать учебник… пенал, где мой пенал?! — нет времени, возьму ручку у соседа…

А, кстати, куда я задевал летние фотографии?.. Эх, надо было сразу сделать папку “Летний поход-2011”… Так нет же, спешил, забросил все фотографии в папку “Мои документы”. И теперь там тысячи картинок с именами, похожими на IMG1234. Сиди часами — перебирай.

Организовывать данные очень важно! Согласны?

Как именно следует организовывать данные? А вот на этот вопрос мы не сможем дать однозначного ответа. В зависимости от вида самих данных, от задач, которые придется решать, могут быть удобны различные способы организации.

Например, структура папок на вашем компьютере образует иерархию. Иерархическая структура для хранения файлов изначально была придумана программистами. Им просто было так удобно с технической точки зрения. Это были времена, когда обычные пользователи и близко не подходили к компьютерам — в лучшем случае им разрешалось потоптаться у дверей с табличкой “Вычислительный центр”. В дальнейшем чисто техническое решение приспособили и для нужд пользователей. Например, у вас на компьютере имеются папки “Мои документы” и “Мои рисунки”. А в папке “Мои рисунки” хорошо было бы сделать папку “Летний поход-2011”. Неужели так и сделали? Молодцы!

Удобно ли хранить файлы в иерархической структуре? Большинству из нас удобно — мы привыкли к ней и приспособились решать возникающие проблемы в ее рамках. А проблемы вполне могут быть. Допустим, у вас уже имеются папки для фотографий “Лето-2011” и “Походы”. И куда положить фотографии летнего похода?

Не стоит ломать над этим голову! Как привыкли — так и делайте. Но важно понимать, что любая форма организации данных имеет некоторые ограничения, которые можно воспринимать и как неудобства. Кстати, все эти неудобства/ограничения обычно можно устранить. Но зачастую лишь за счет усложнения формы организации данных.

Вы ведь, конечно, понимаете, что наиболее существенным является именно сам способ организации данных, а не его физическая реализация? С этой точки зрения нарисованная на листе бумаги файловая структура ничем не отличается от реальной файловой структуры вашего компьютера. Только вопрос “куда записать файл?” превратится в вопрос “куда вписать имя файла?”.

Таблицы всюду

С представлением информации в табличной форме мы имеем дело постоянно. В начальной школе, только изучая таблицу умножения, мы уже прекрасно умели пользоваться таблицей с расписанием уроков, и таблицей-календарем, и таблицей с автобусным расписанием. Таблицы всюду! Прежде чем идти дальше, давайте-ка немного “разомнемся”.

• Представьте следующую информацию в табличной форме.

Задание 1. У Васи Иванова день рождения 1 сентября, живет Вася в Москве, позвонить ему можно по телефону 8(495)123-45-67. В Екатеринбурге живет Маша Петрова. Ей можно позвонить по телефону 8(343)76-543-21. А 19 октября мы можем поздравить Машу с днем рождения. По телефону 8(846)109-87-65 можно позвонить Денису Сидорову. Судя по коду города в телефонном номере, он живет в Самаре. Когда у Дениса день рождения, мы, к сожалению, не знаем.

Задание 2. 1 сентября в 12:00 в кинотеатре шел фильм “Олимпийский мишка”. В зале было всего 10 человек. В тот же день после обеда в 16:45 показывали “Смешных ребят”. Народу было уже побольше — 55 человек. Вечернего сеанса 1 сентября не было, зато он был на следующий день в 19:00. Зал был полон — на фильм “Таблица” пришло 205 человек.

Справились? Конечно, справились! Ничего сложного! Но все же давайте посмотрим, что у вас получилось. Сравните ваши таблицы и то, что получилось у авторов учебника.

Фамилия	Имя	День рождения	Город	Телефон
Иванов	Вася	1 сентября	Москва	8(495)123-45-67
Петрова	Маша	19 октября	Екатеринбург	8(343)76-543-21
Сидоров	Денис		Самара	8(846)109-87-65

Дата	Время	Название	Количество зрителей
1 сентября	12:00	“Олимпийский мишка”	10
1 сентября	16:45	“Смешные ребята”	55
2 сентября	19:00	“Таблица”	205

Укажем, на что важно обратить внимание. Во-первых, мы снабдили каждый столбец таблицы заголовком, в котором записали, какую именно информацию мы будем записывать в этот столбец. Далее мы “выуживали” из текста — в данном случае это было не сложно — объекты, о которых идет речь. В первом примере речь шла о людях, во втором — о сеансах в кинотеатре. Информацию о каждом объекте мы записали в одной строке таблицы. Причем в каждой ячейке мы записывали информацию, соответствующую заголовку столбца.

Отметим сразу, что нередко одни и те же данные можно организовать посредством различных таблиц, каждая из которых будет выглядеть вполне разумно. Например, в первой таблице, возможно, имеет смысл разместить коды городов в отдельном столбце.

Фамилия	Имя	День рождения	Город	Код города	Телефон
Иванов	Вася	1 сентября	Москва	495	123-45-67
Петрова	Маша	19 октября	Екатеринбург	343	76-543-21
Сидоров	Денис		Самара	846	109-87-65

Мы с вами начинаем заниматься изучением баз данных. В этой области информатики есть своя терминология, с которой мы познакомимся, но в минимально необходимом объеме. Сейчас как раз такой случай. Давайте запомним, что строки таблицы принято называть **записями**, а столбцы — **полями**. Отметим, что последний термин часто используют для обозначения различных объектов. Это не ведет к непониманию, но все же необходимо прояснить этот вопрос.

Мы сказали, что полем называют столбец таблицы. Фактически речь идет и о названии столбца, и обо всем столбце целиком, и о каждой конкретной ячейке — тогда говорят о поле конкретной записи. Если имеется риск неточного понимания, то предпочитают употреблять точные термины: “название поля” — о заголовке, “поле записи” — о ячейке. Если из контекста понятно, о чем идет речь, можно просто употреблять слово “поле”.

Мы начали с обсуждения вопроса о том, что способ организации данных крайне важен для их эффективной обработки. Поскольку сейчас мы говорим о таблицах, вы наверняка уже сделали вывод (правильный!), что мы будем рассматривать именно табличную организацию данных. Конкретнее — базы данных, в которых данные организованы в таблицы. В настоящее время именно табличные базы данных наиболее распространены. О самом термине “база данных” мы поговорим далее в этом параграфе.

Но это же не всегда может быть удобно, скажете вы. Согласимся — не всегда. Но опять же вспомним то, о чем мы уже говорили, — всякая форма организации данных имеет свои преимущества (удобства) и недостатки (неудобства). Таблицы — достаточно удобная и универсальная форма организации данных. С помощью таблиц можно представить данные даже очень сложной структуры. Правда, иногда придется хорошенько подумать — как именно. Но это ведь и интересно!

Не все то таблица…

Всякую ли таблицу, нарисованную от руки на листе бумаги или даже расчерченную в текстовом редакторе, можно использовать для хранения информации в базе данных? Увы — нет. Имеется немало требований к таблицам баз данных. Большую часть этих требований мы рассмотрим на следующих уроках (эти материалы не вошли в журнальную статью), а самые основные — прямо сейчас.

Итак. **Во-первых, все записи должны состоять из одного и того же набора полей.** Порядок следования полей тоже, разумеется, должен быть единым. (Чувствуете, как мы сразу перешли на использование терминологии — записи, поля. Привыкайте!)

Может ли так случиться, что у какой-то записи нет того или иного поля? Ответ: нет, не может! Другое дело, что мы можем не знать значение в этом поле, и его придется оставить пустым. Но просто “выбросить” поле нельзя. В текстовом редакторе, например, можно также “объединить ячейки” в таблице, но такие таблицы не годятся в качестве таблиц баз данных.

Во всех записях в одном и том же поле должны находиться данные одного и того же вида. Вы ведь уже изучали основы программирования? Тогда скажем точнее: **во всех записях в одном и том же поле должны находиться данные одного и того же типа**. Конкретный набор допустимых типов данных обычно определяется используемой системой — где-то больше, где-то меньше. Основные типы — числа и строки символов имеются в любой системе. Практически в любой системе имеются также специальные типы для хранения значений даты и времени.

Поясним это требование примером того, как нельзя. Допустим, мы храним в поле “Оценка”… оценку, что же еще в нем хранить? Если мы храним оценки в виде чисел, то всюду и надо использовать именно числа.

Фамилия	Имя	Оценка
Иванов	Вася	4
Петрова	Маша	5
Сидоров	Денис	Пять

Если бы мы (хотя это жутко неудобно!) решили хранить оценки в виде слов (строк), то ошибочными были бы, напротив, данные в первых двух строках.

К последнему требованию примыкает еще одно — **в каждом поле может быть записано не более одного данного**. В частности, две следующие таблицы с этой точки зрения являются неправильными.

Фамилия	Имя	Телефон
Иванов	Вася	123-45-67 (д.), 765-43-21 (м.)

Дата	Время	Название
1 сентября	12:00, 14:00	“Олимпийский мишка”

• “Почините” несколько таблиц. Способ ремонта в каждом случае выбирайте сами — может быть, придется заводить новые поля.

Фамилия	Имя	День рождения	Город	Телефон
Иванов	Вася	1 сентября	Москва	8(495)123-45-67 (д.), 8(926)007-07-07 (м.)
Петрова	Маша	19 октября	Екатеринбург	8(343)76-543-21
Сидоров	Денис	Не знаю	Самара	8(846)109-87-65

Дата	Время	Название	Количество зрителей
1 сентября	12:00, 16:00	“Олимпийский мишка”	Утром — совсем мало, человек 10, во второй половине дня побольше — 30
1 сентября	16:45	“Смешные ребята”	55
2 и 3 сентября	19:00	“Таблица”	205 и 200 человек соответственно

Проверим на всякий случай, что получилось у вас и авторов? Это интересно, поскольку решения не являются однозначными!

При починке первой таблицы понятно, что “Не знаю” надо просто убрать. А вот с телефонами не все так просто. Можно предложить два варианта. Первый — очевидный.

Фамилия	Имя	День рождения	Город	Телефон домашний	Телефон мобильный
Иванов	Вася	1 сентября	Москва	8(495)123-45-67	8(926)007-07-07
Петрова	Маша	19 октября	Екатеринбург	8(343)76-543-21	
Сидоров	Денис		Самара	8(846)109-87-65	

Второй вариант починки первой таблицы менее очевидный. Но многие специалисты сделали бы именно так.

Фамилия	Имя	День рождения	Город	Телефон домашний	Тип телефона
Иванов	Вася	1 сентября	Москва	8(495)123-45-67	Домашний
Иванов	Вася	1 сентября	Москва	8(926)007-07-07	Мобильный
Петрова	Маша	19 октября	Екатеринбург	8(343)76-543-21	Домашний
Сидоров	Денис		Самара	8(846)109-87-65	Домашний

Результат ремонта второй таблицы выглядит однозначно.

Дата	Время	Название	Количество зрителей
1 сентября	12:00	“Олимпийский мишка”	10
1 сентября	16:00	“Олимпийский мишка”	30
1 сентября	16:45	“Смешные ребята”	55
2 сентября	19:00	“Таблица”	205
3 сентября	19:00	“Таблица”	200

БД и СУБД

Вы наверняка уже поняли — хотя бы прочитав название этого параграфа, что за аббревиатурами “БД” и “СУБД” скрываются термины **база данных** и **система управления базами данных**. А вообще — различаются ли эти понятия? Ведь их так часто путают! Ответ: различаются.

Давайте договоримся, что мы будем понимать под указанными понятиями. Базой данных мы будем называть организованную совокупность цифровых данных. В этом параграфе мы обсуждали организацию данных в виде таблиц. Сами данные также, разумеется, включаются в базу данных, но наиболее важным является именно сам способ их организации. Нередко используют термин **структура**, под которым понимают именно способ организации данных в чистом виде. Когда говорят о структуре таблицы, то говорят, например, о названиях и типах полей, а сами записи при этом не интересуют — их может быть много, мало или не быть вообще.

Для удобства и компактной записи структуру таблицы нередко представляют в виде перечня полей. В частности, структуру последней рассмотренной выше починенной таблицы можно записать так:

Дата
Время
Название
Количество зрителей

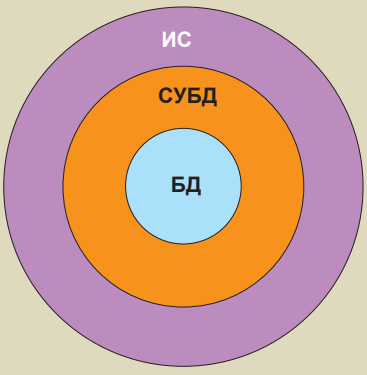
С системой управления базами данных мы пока не работали, с ней мы познакомимся далее. СУБД — компьютерная программа, предназначенная для того, чтобы базу данных можно было хранить и обрабатывать в компьютерном виде.

Одну и ту же базу данных можно представить в компьютерном виде при помощи различных СУБД. Можно даже использовать обычный текстовый документ с таблицами. Только в таком случае данные будет не слишком удобно обрабатывать. А именно удобство и скорость обработки данных — важные функции СУБД.

Информационные системы (ИС)

Термин “информационные системы” широко распространен. Так часто называют разнообразные комплексы программных, аппаратных и иных (например, организационных) средств, обеспечивающих обработку данных. Как соотносятся между собой базы данных (БД), системы управления базами данных (СУБД) и информационные системы (ИС)?

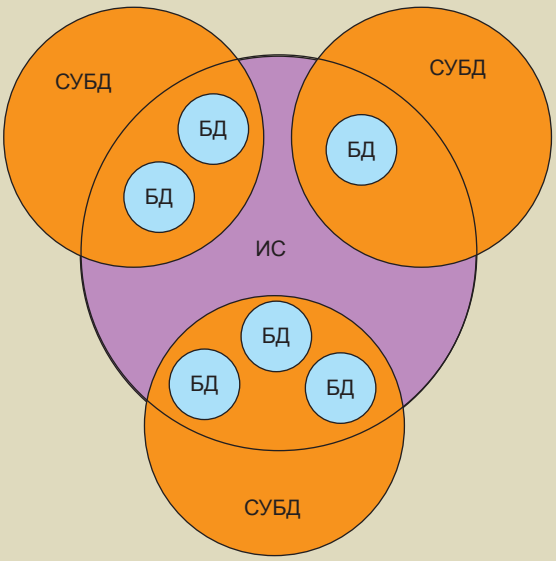
В достаточно упрощенном виде схему их взаимодействия можно представлять следующим образом:



Для компьютерной реализации базы данных требуется система управления базами данных. И именно с этой системой взаимодействует информационная система, которой требуется использовать ресурсы базы данных.

Мы назвали схему достаточно упрощенной по причине того, что одна СУБД может управлять множеством баз данных и одна информационная система тоже может использовать ресурсы нескольких баз данных. Например, информационная система по продаже авиабилетов может использовать множество баз данных авиакомпаний, каждая из которых может обслуживаться собственной СУБД. Но на принципиальном уровне эта схема отражает суть взаимодействия.

Приведем более точную схему, отражающую отношения между понятиями ИС, СУБД и БД. В этой схеме учтены указанные выше обстоятельства — ИИС может использовать ресурсы различных БД, каждая из которых может обслуживаться собственной СУБД.



ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

Организация данных существенно влияет на производительность их обработки. И эффект этот проявляется тем сильнее, чем больше объемы данных. Мы будем заниматься знакомством с технологиями хранения и обработки больших объемов данных при помощи табличных баз данных. Строки таблиц называют записями, столбцы — полями. Не всякая таблица, которую можно нарисовать на бумаге, может быть таблицей базы данных. В частности, необходимо, чтобы все записи состояли из одного и того же набора полей, чтобы в одном и том же поле в различных записях находились данные одного и того же типа, а в каждом поле было записано только одно данное. Системы управления базами данных представляют собой компьютерные программы для работы с базами данных.

ПРИМЕНЕНИЕ ЗНАНИЙ

• Представьте следующую информацию в табличной форме. Позаботьтесь о том, чтобы таблица была правильной в рассмотренном выше смысле. Постарайтесь предложить несколько вариантов табличного представления данных.

1 сентября Вася сходил на фильм “Новый учебный год”. Причем фильм так понравился Васе, что, посмотрев сеанс в 16:00, он сразу же пошел на фильм еще раз. Следующий сеанс был в 18:00. На следующий день Миша по рекомендации Васи тоже захотел посмотреть “Новый учебный год”. Но билетов на сеанс 16:00 уже не оказалось. Миша сразу купил себе билеты на сеанс в 18:00, а чтобы не терять времени даром, в 16:00 посмотрел фильм “Таблица возвращается”. Оба фильма Мише понравились. 4 сентября Вася по рекомендации Миши посмотрел фильм “Таблица возвращается”. Билеты были только на сеанс 20:00. Может быть, оттого, что было уже поздновато, но фильм Васе совсем не понравился. 7 сентября товарищи пригласили одноклассницу Машу и втроем сходили на фильм “Абак”. Маше очень понравился фильм, а Васе с Мишей — нет.

Урок 2. Ключевое понятие — ключ

ПОСТАНОВКА ПРОБЛЕМЫ УРОКА

Допустим, у нас в классе учатся не просто однофамильцы, а полные тезки. Как учителя различают их в журнале?

Так ведь в журнале, помимо фамилии и имени, имеется номер ученика! Вот и договариваются, кто из них какой номер имеет. Так и различают.

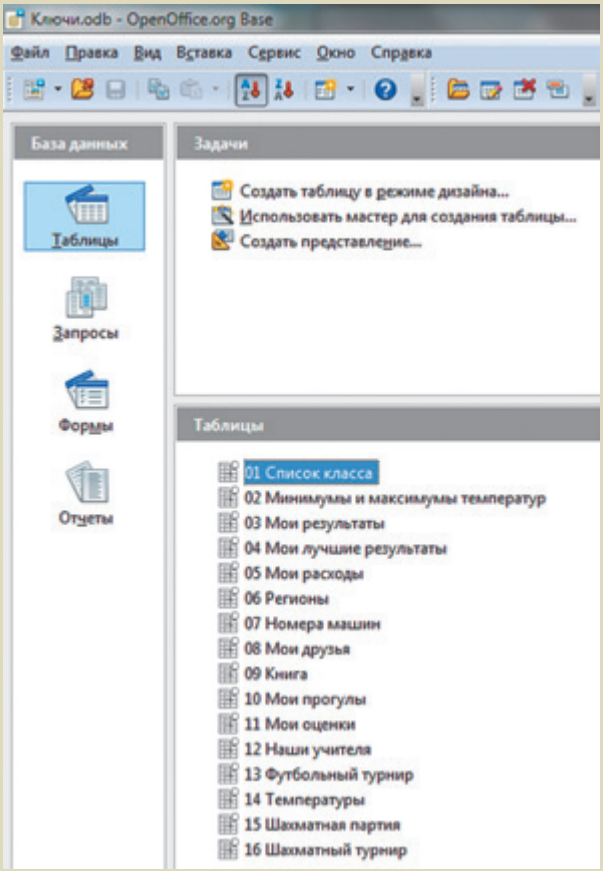
РЕШЕНИЕ ПРОБЛЕМЫ

СУБД OpenOffice Base

Прошлый урок мы закончили на том, что пообещали от “слов” — тетрадки — перейти к “делу” — компьютеру и познакомиться с системой управления базами данных. Начиная с этого урока, мы будем решать задачи и выполнять упражнения в СУБД OpenOffice Base. Знакомиться с системой мы станем постепенно. Сегодня — просто “поздороваемся” и поработаем с уже заготовленной заранее базой данных, для того чтобы разобраться с важнейшим, ключевым понятием — ключом. Поэтому и база данных, с которой мы будем работать, называется “Ключи”.

Нет одинаковым строкам в таблицах!

На прошлом уроке мы говорили о том, что не всякая таблица, которую можно нарисовать на бумаге или набрать в текстовом редакторе, может быть таблицей базы данных. Это был разговор о том, “как нельзя”. На бумаге — можно, а в базе данных — нельзя. К сожалению, рассмотренными ра-



нее “нельзя” дело не ограничивается. В частности, есть еще одно важное ограничение: **в таблицах баз данных не может быть полностью одинаковых строк**. Нам ничего не мешает напечатать такую таблицу в учебнике:

Фамилия	Имя	Дата рождения
Иванов	Петя	1.09.1999
Иванов	Петя	1.09.1999

Но в таблицу базы данных такие две одинаковые строки записать нельзя!

Почему? Посудите сами — как компьютер, система управления базами данных будет различать наших Петь?

Простой ключ

Итак, любые две строки в таблицах баз данных обязательно отличаются, по крайней мере значением какого-то одного поля. Часто бывает так, что можно точно указать одно конкретное поле таблицы, значения в котором гарантированно различаются для всех записей. Такое поле называют **простым ключом**. При этом не имеет значения, что именно сейчас содержится в таблице. Важно именно, что в принципе может в ней содержаться. Например, то, что у нас в данный момент в классе нет однофамильцев, не значит, что их никогда не было и никогда не появится в будущем.

• Поработаем с базой данных “Ключи”. В ней имеется несколько различных таблиц, между ними нет никаких связей — каждая таблица сама по себе. Выделив имя таблицы, можно сразу в окне просмотра увидеть ее структуру. Вам требуется просмотреть таблицы и для каждой решить: есть ли в ней одно-единственное поле, значения в котором гарантированно будут различными, тот самый простой ключ.

Пожалуйста, вдумывайтесь каждый раз — “о чем эта таблица”, какие именно значения предполагается в ней хранить. Обращайте внимание на названия таблиц.

Для удобства работы таблицы пронумерованы. По ходу просмотра таблиц, пожалуйста, оставляйте записи в тетрадке: таблица такая-то, простой ключ такой-то; таблица такая-то, простого ключа нет и т.п. Не спешите, нет необходимости успеть просмотреть все таблицы.

Справились? Давайте сравним ваши результаты и то, что получилось у авторов. Это интересно!

Таблица “01 Список класса”

Номер ученика	**Фамилия**	**Имя**	**Дата рождения**

Это задание нам кажется простым. Мы ведь уже не раз подступались к нему на уроке. **Номер ученика** и является тем самым простым ключом — именно в этом поле значения точно не могут повторяться. Обратите внимание: обнаружив простой ключ, мы его подчеркнули. Мы и далее будем так делать, но точный смысл подчеркивания поясним позже в этом параграфе.

Таблица
“02 Минимумы и максимумы температур”

Дата	**Минимальная температура**	**Максимальная температура**

В данном случае простым ключом является поле **Дата**. Нет возражений?

Таблица “03 Мои результаты”

Дистанция	**Время**

Авторы склонны думать, что ни одно из полей этой таблицы нельзя считать простым ключом. Действительно, одну и ту же дистанцию можно пробежать несколько раз, да и время одно и то же вполне можно показать. Например, бегали вы раньше 100 м за некоторое время, а потом хорошенько потренировались и стали за то же время бегать 110 м. Почему нет?

Таблица “04 Мои лучшие результаты”

Дистанция	**Время**

А вот тут ситуация иная! Ведь лучший результат точно только один для данной дистанции! Поэтому поле **Дистанция** — простой ключ.

Таблица “05 Мои расходы”

Дата	**Назначение**	**Сумма**

Авторы учебника считают, что в этой таблице поля, которое можно было назвать простым ключом, нет. Почему? Можем ли мы считать таковым поле **Дата**?

Нет — вполне можно несколько раз потратиться в течение дня. Поле **Назначение**? Тоже нет — ничего не мешает нам купить, например, две ручки. Поле **Сумма**? И здесь ответ отрицательный — даже тетрадка и мороженое могут стоить одинаково.

Таблица “06 Регионы”

Код региона	**Название**

Коды регионов России знают все — эти числа имеются, например, на автомобильных номерах. То, что сам по себе **Код региона** является простым ключом, достаточно очевидно. А вот что можно сказать о поле **Название**? Вполне возможно, что кто-то отметил именно это поле. Правильно ли это? Оказывается, это зависит от того, как именно мы смотрим на таблицу “Регионы”. Если мы подумали, что речь идет только о регионах России, то и поле **Код региона**, и поле **Название** могут быть ключами! Если речь идет о перечислении регионов нашей страны, то и в одном, и в другом поле содержатся уникальные и не повторяющиеся значения.

А вот если кто-то подумал, что речь идет о кодах регионов на автомобильных номерах, ситуация иная. Тогда **Код региона** по-прежнему простой ключ, а вот **Название** — нет! Все ведь знают, что когда автомобильные номера в крупных регионах стали заканчиваться, этим регионам выдали дополнительные коды для автомобильных номеров. Например, в Москве это и 77, и 97, и 99, и 177, и 197, и 199.

Это очень важно! **Является то или иное поле простым ключом или нет, зависит от содержательной сущности таблицы! Сами по себе значения мало что значат — важно, что мы знаем о том, какими они могут быть.**

Таблица “07 Номера машин”

Три цифры	**Три буквы**	**Код региона**	**Марка**

Ни одно из полей этой таблицы нельзя назвать простым ключом. Согласны?

Таблица “08 Мои друзья”

Фамилия	**Имя**	**Телефон**	**E-mail**	**Дата рождения**

Это очень интересное задание! Результаты его выполнения в классе вполне могут оказаться различными. И каждый из этих результатов может оказаться правильным! Вот как!

Давайте разберемся. С полями **Фамилия**, **Имя**, **Дата рождения** все понятно. Их нельзя назвать простыми ключами. Понятно почему? А вот с полями **Телефон** и **E-mail** не все так очевидно. В наше время мобильные телефоны есть практически у каждого. И когда мы звоним на мобильный, мы звоним именно конкретному человеку. То есть можно считать, что если мы имеем в виду номер мобильного телефона, то он для каждого свой, и два разных человека не могут иметь один и тот же номер. Значит, поле **Телефон** можно будет считать простым ключом.

Те же рассуждения справедливы и для поля **E-mail**. Электронные адреса бывают разными — бывают адреса людей, а бывают адреса организаций. Если мы считаем, что каждый человек имеет свой личный уникальный электронный адрес, то и поле **E-mail** можно считать простым ключом.

Таблица “09 Книга”

Номер страницы	**Количество символов**	**Количество слов**	**Количество знаков препинания**

Здесь поле **Номер страницы** является простым ключом, и обсуждать, пожалуй, нечего. Нет возражений?

Таблица “10 Мои прогулы”

Дата	**Номер урока**

Во-первых, прогуливать нехорошо! Можно даже сказать, что прогуливать нельзя! Но… со всеми бывает. Можно ли назвать простым ключом какое-то одно поле в этой таблице? Увы, нет. И в один день можно несколько уроков прогулять, и один и тот же урок можно прогулять в различные дни. Но это нехорошо!

Таблица “11 Мои оценки”

Дата	**Номер урока**	**Оценка**

Ни одно из полей в этой таблице нельзя назвать простым ключом. Согласны?

Таблица “12 Наши учителя”

Фамилия	**Имя**	**Отчество**	**Предмет**

И в этой таблице ни одно из полей нельзя назвать простым ключом. В каждом из них вполне могут повторяться значения.

Таблица “13 Футбольный турнир”

Дата	**Команда 1**	**Команда 2**	**Счет**

Скорее всего простого ключа в этой таблице нет. По крайней мере авторы ни одно из полей таковым не считают. Хотя… надо бы поподробнее знать об особенностях данного турнира. Вдруг, допустим, в течение одного дня может состояться только один матч. Тогда поле **Дата** будет простым ключом.

Таблица “14 Температуры”

Дата	**Время**	**Температура**

Может быть, кто-то подумал о собственных температурах? Авторы-то имели в виду, например, данные метеостанции — измерения температуры воздуха. В любом случае ни одно из полей тут нельзя считать простым ключом.

Таблица “15 Шахматная партия”

Номер хода	**Ход**

Ну, наконец-то! А то в предыдущих примерах слишком долго мы не могли найти простой ключ. Здесь же все очевидно.

Таблица “16 Шахматный турнир”

Номер партии	**Номер хода**	**Ход**

А вот опять простого ключа нет! Понятно, что и номера партий, и номера ходов, и сами ходы могут повторяться.

При выполнении упражнения и в ходе обсуждения его результатов мы искали в таблицах одно-единственное поле, которое можно было назвать простым ключом. Почему простым? Что, бывают сложные? Нет, сложных не бывает. Помните из математики: бывают простые числа, а бывают… составные. Вот составными ключами мы далее и займемся.

Составной ключ

До сих пор мы искали одно-единственное поле, значения в котором гарантированно не могут повторяться. И не всегда находили. Но можно не ограничиваться лишь одним полем, а искать комбинацию полей, сочетание которых гарантированно будет уникальным. Правда, здесь необходимы уточнения. Если, например, мы возьмем простой ключ и добавим к нему любые поля, то получим ту самую уникальную комбинацию. Понятно почему? Это чисто формальное и довольно бессмысленное действие. Интересно искать такие комбинации полей, из которых нельзя убрать ни одного поля без потери уникальности. Если такую комбинацию полей удается найти, ее называют **составным ключом**.

• Давайте еще раз просмотрим таблицы базы данных “Ключи”. Те таблицы, в которых мы нашли простые ключи, нас уже интересовать не будут. А вот для остальных таблиц мы попробуем найти составной ключ.

Сверим результаты?

Таблица “03 Мои результаты”

Дистанция	Время

В этой таблице два поля, и ранее мы не нашли в ней простого ключа. Есть ли здесь составной? Авторы учебника, исходя из своих представлений о том, что именно может храниться в этой таблице, думают, что нет.

Таблица “05 Мои расходы”

Дата	Назначение	Сумма

Если мы хотим найти составной ключ, надо посмотреть на пары полей и на все три поля вместе. Можно ли какую-то пару считать уникальной? Авторы учебника думают, что нет. А все три поля вместе? Тоже нет. В один и тот же жаркий день вполне можно дважды купить одно и то же любимое мороженое. И, надеемся, стоить оно будет при этом одинаково.

Таблица “07 Номера машин”

Три цифры	Три буквы	Код региона	Марка

В данном случае ответ на вопрос, сформулированный в задании, нам известен из жизни. Автомобильные номера, представляющие собой комбинацию из трех цифр, трех букв и кода региона, являются уникальными. И ни одно из указанных полей нельзя “выкинуть”. Значит, мы нашли составной ключ!

Таблица “08 Мои друзья”

Фамилия	Имя	Телефон	E-mail	Дата рождения

Ситуацию с этой таблицей мы подробно обсудили при разборе результатов выполнения второго задания. Вполне вероятно, что при каких-то допущениях — знаниях о том, какие именно данные хранятся в таблице, какое-то одно из полей в ней можно считать простым ключом. Но если мы так не считаем, то и комбинация полей нам не поможет.

Таблица “10 Мои прогулы”

Дата	Номер урока

Не знаем, как дважды войти в одну и ту же реку, — это вопрос не нашего предмета, но дважды прогулять один и тот же урок в один и тот же день точно нельзя. Поэтому комбинация двух полей является составным ключом.

Таблица “11 Мои оценки”

Дата	Номер урока	Оценка

Очень интересное задание с точки зрения поиска составного ключа! Есть он или нет, зависит от… инструкций Министерства образования. Вот как! Почему? Дело в том, что важным здесь является то, можно ли выставлять в одну клеточку журнала более одной оценки. Если нет — комбинация полей **Дата+Номер урока** является составным ключом. Если разрешается — не является! Честно говоря, четких инструкций на эту тему нет. Из жизни мы знаем, что на одном уроке вполне можно получить несколько оценок, а на некоторых предметах это вообще обычное дело. (На каких?) Если исходить из этого, комбинация **Дата+Номер урока** не является составным ключом.

Таблица “12 Наши учителя”

Фамилия	Имя	Отчество	Предмет

Ну, нет здесь составного ключа, нет.

Таблица “13 Футбольный турнир”

Дата	Команда 1	Команда 2	Счет

Эта таблица — довольно любопытный случай. Можно ли считать комбинацию полей **Дата + Команда 1 + Команда 2** составным ключом, зависит от наших знаний о данном турнире. Могут ли две команды играть в день более одного матча? Здравый смысл подсказывает, что нет. А если могут, то это не составной ключ.

Таблица “14 Температуры”

Дата	Время	Температура

Классический пример составного ключа.

Таблица “16 Шахматный турнир”

Номер партии	Номер хода	Ход

И еще один столь же классический пример.

Простые ключи, составные ключи, просто ключи

Мы дважды прошлись по набору предложенных таблиц. В первый раз мы искали только простые ключи. Во второй раз — составные ключи. Вообще говоря, большой разницы между ними нет, и часто используют один термин **ключ**, под которым понимают и простые, и составные ключи. Кстати, как вы думаете, может ли одна таблица иметь и простые, и составные ключи?

Ответ — может. Например, таблица

"Граждане страны"

ИНН	Фамилия	Имя	Серия паспорта	Номер паспорта

Имеет простой ключ **ИНН** — значение ИНН уникально для каждого гражданина страны. Но в ней также имеется составной ключ **Серия паспорта+Номер паспорта** — их комбинация также уникальна. И простой ключ, и составной ключ — ключи данной таблицы.

В общем случае — и мы это видели на примерах — в таблице может быть несколько простых ключей и несколько составных ключей. То есть в таблице может быть несколько ключей.

Ключей — много, золотой (первичный) — один

Среди всех возможных ключей таблицы обязательно выбирают один-единственный, который называют **первичным**. Выбор первичного ключа — дело проектировщика. Именно про первичный ключ таблицы проектировщик сообщает СУБД — о первичном ключе знает не только человек, но и компьютерная программа. Кстати, обычно СУБД "жестко требует", чтобы в таблице был указан первичный ключ, и не дает создать таблицу без первичного ключа.

Подчеркиванием, которое мы использовали выше, принято обозначать именно первичный ключ. А поля, входящие в состав первичного ключа, часто называют **ключевыми полями**. Словосочетание "ключевое поле" широко используется, но обращаться с ним следует осторожно. Например, если мы говорим, что данное поле является ключевым, то этим утверждается, что поле входит в состав первичного ключа, но ничего не говорится о том, является данный ключ простым или составным — есть ли еще ключевые поля в таблице.

Добавляем ключ. При необходимости

Во всех ли рассмотренных выше таблицах мы обнаружили ключи? Нет. А как быть с таблицами, в которых мы пока не нашли ни простого, ни составного ключа? Нет ли здесь противоречия? Ведь в таблице базы не должно быть повторяющихся строк! Значит, уж комбинация всех полей-то должна быть составным ключом. Что-то мы запутались!

В этом месте любой специалист в области проектирования баз данных с удовольствием пришел бы нам на помощь. И сказал бы он примерно следующее. Проблема ваша (наша) в том, что вы искали в таблицах так называемые "естественные ключи" — те, наличие которых обусловлено смыслом хранимых данных. А таких ключей может и не быть.

Посмотрим на "проблемную" таблицу.

Таблица "03 Мои результаты"

Дистанция	Время

Когда мы смотрим на таблицу, в которой не получается выделить ключ, вопрос, на который нам нужно ответить: а могут ли строки в этой таблице повторяться с содержательной точки зрения? В базу данных нам точно не удастся занести повторяющиеся строки, это мы уяснили. Но если бы мы использовали ту же таблицу практических целей для записей в тетрадке. Может ли возникнуть необходимость занести в нее одинаковые строчки?

Применительно к этой таблице ответ: скорее — да. Действительно, вполне может быть так, что мы фиксируем свои результаты и несколько раз показываем один и тот же результат. Что бы мы сделали в тетрадке? Скорее всего мы бы нумеровали записи. Даже если бы мы это не делали явно, сам порядок записей — уже нумерация. Но большинство бы ставило номера записей:

1. 100 м 15 с
2. 110 м 18 с
3. 100 м 14,8 с

Если с содержательной точки зрения в таблице могут повторяться строки, то необходимо завести специальное поле, которое было бы ключом и обеспечивало уникальность записей. Ровно так, как мы сделали бы в тетрадке. То есть в рассматриваемую таблицу надо добавить, например, поле **Номер записи.**

Таблица "Мои результаты"

Номер записи	Дистанция	Время

Рассмотрим оставшиеся таблицы.

Таблица "05 Мои расходы"

Дата	Назначение	Сумма

И снова надо добавить такое же поле.

Таблица "Мои расходы"

Номер записи	Дата	Назначение	Сумма

Таблица "08 Мои друзья"

Фамилия	Имя	Телефон	E-mail	Дата рождения

И снова то же решение! Смотрите, как часто оно используется!

Таблица "Мои друзья"

Номер записи	Фамилия	Имя	Телефон	E-mail	Дата рождения

Таблица "11 Мои оценки"

Дата	Номер урока	Оценка

Если мы исходим из того, что комбинация полей Дата + Номер урока не является составным ключом, надо добавлять ключевое поле — вариантов нет.

Таблица “Мои оценки”

Номер записи	Дата	Номер урока	Оценка

Таблица “12 Наши учителя”

Фамилия	Имя	Отчество	Предмет

И здесь тоже надо добавлять поле простого ключа.

Таблица “Наши учителя”

Номер записи	Фамилия	Имя	Отчество	Предмет

Таблица “13 Футбольный турнир”

Дата	Команда 1	Команда 2	Счет

И здесь. Хотя, конечно, два раза в день играть… Но если считать, что такое может случиться, надо предусмотреть и такой случай.

Таблица “Футбольный турнир”

Номер записи	Дата	Команда 1	Команда 2	Счет

Обратите внимание, как часто нам пришлось добавлять ключи!

Отметим в заключение, что нередко ключ добавляют и в таблицы, в которых ключи есть. Это чисто технический вопрос, но мы кратко его обозначим. Если ключ составной и длинный, то для повышения быстродействия обработки данных может быть целесообразно добавить простой ключ.

ОБОБЩЕНИЕ НОВЫХ ЗНАНИЙ

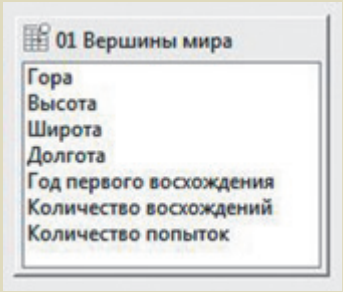
В таблицах баз данных не может быть повторяющихся строк — любые две строки обязательно должны различаться значением хотя бы одного поля. Поле или совокупность полей, которые для любых двух строк таблицы гарантированно различаются, называются ключом данной таблицы. Если ключ состоит из одного поля, его называют простым, если из нескольких полей — составным. В составном ключе не должно быть полей, которые можно было бы исключить без потери свойства уникальности. Среди всех ключей выделяют один, о котором сообщают СУБД. Этот ключ называют первичным. Поля, входящие в состав первичного ключа, называют ключевыми. Если среди полей в таблице не удается обнаружить естественного ключа, ключ приходится добавлять…

Дополнительные задачи на выявление естественных ключей

Для практики на уроках дополнительно предлагается база данных “Ключи1.odb”. Как и в базе данных “Ключи”, которая рассматривалась выше, в базе “Ключи1” имеются не связанные между собой таблицы. Дети должны в процессе обсуждения, исходя из знаний, здравого смысла, разумных допущений, возможно используя дополнительные источники информации, установить наличие или отсутствие естественных ключей (простых из составных) в таблицах.

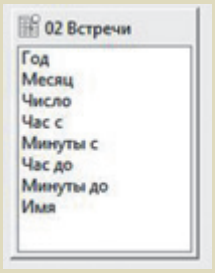
Отметим, что ряд заданий достаточно сложны и даже не имеют однозначного ответа — в зависимости от сделанных допущений, можно сделать различные выводы.

Вершины мира



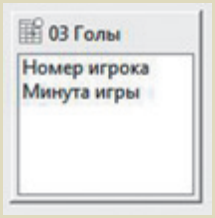
С точки зрения географа, парой широта и долгота можно однозначно определить гору, то есть их можно было бы сделать составным ключом, если бы не было особенностей представления вещественных чисел. Особенность заключается в том, что вещественные числа на различных компьютерах представляются по-разному, с разной степенью точности. Поэтому, с точки зрения проектировщика баз данных, в этой таблице ключей нет.

Встречи



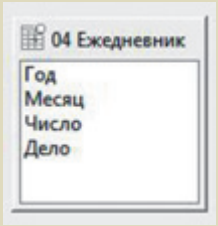
Ключей нет, так как в один и тот же день, в одно и то же время может быть назначена встреча с двумя людьми с одинаковыми именами.

Голы



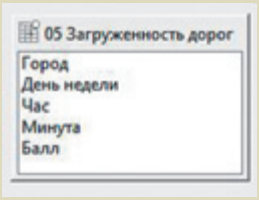
Ключей нет, так как один игрок может в течение минуты забить несколько голов. Если в таблицу добавить поле секунды, то можно было бы получить составной ключ из всех полей таблицы.

Ежедневник



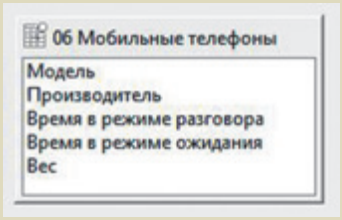
Ключей нет, так как в один год, один месяц, одного числа может делаться сразу несколько дел.

Загруженность дорог



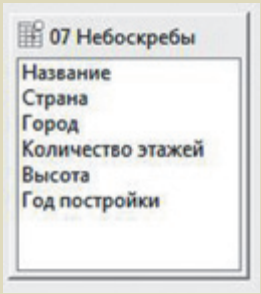
Это задача по мотивам сервиса “Яндекс-пробки”. Если под баллом понимать среднюю загруженность дорог в данный момент времени данного дня недели, то в этой таблице имеется составной ключ.

Мобильные телефоны



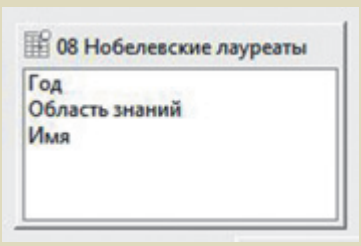
Модель+производитель может быть составным ключом.

Небоскребы



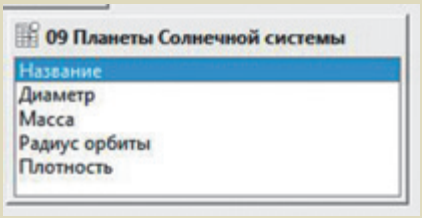
Название+страна+город может быть составным ключом.

Нобелевские лауреаты



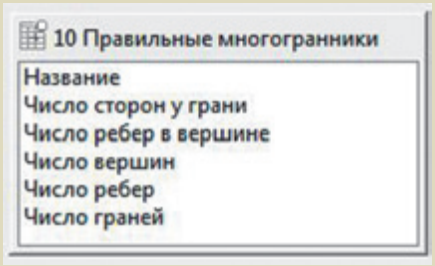
Если подходить к вопросу формально, то возможна ситуация получения премии в один год, в одной области знаний двух ученых с одинаковыми именами. С этой точки зрения в таблице нет ключа.

Планеты Солнечной системы



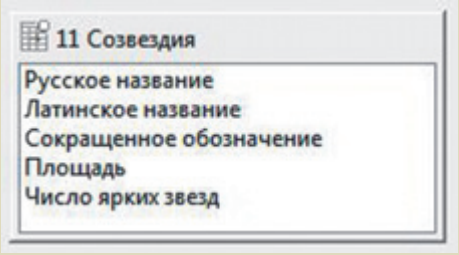
Название каждой планеты уникально, поэтому его можно выбрать в качестве простого ключа.

Правильные многогранники



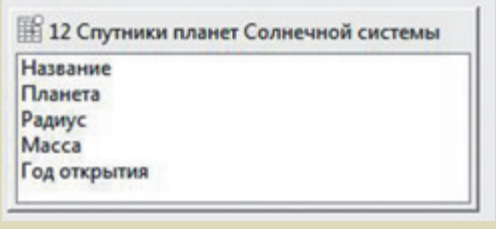
Это очень нетривиальная математическая задача. Понятно, что название правильного многогранника уникально и может быть простым ключом. Но простыми ключами также могут быть число граней и число вершин. Подтверждение этому можно увидеть в любом справочнике по математике.

Созвездия



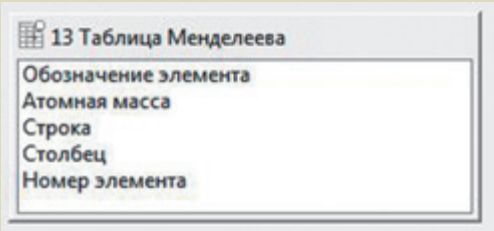
Поиск ключей в этой таблице может потребовать знакомства с вопросом. В результате этого знакомства выяснится, что и русское, и латинское название, и сокращенное обозначение могут быть простыми ключами.

Спутники планет Солнечной системы



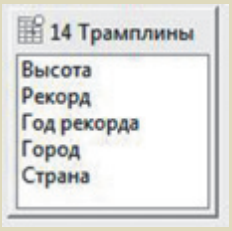
Названия двух спутников для разных планет могут совпадать. Названия планет всегда уникальны, поэтому можно в качестве составного ключа взять название+планета.

Таблица Менделеева



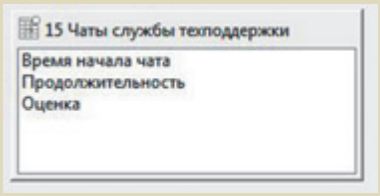
С содержательной точки зрения в качестве простого ключа можно выбрать любое из полей: обозначение элемента, атомная масса, номер элемента. Это следует из самого способа построения таблицы. Из практических соображений дробные атомные массы делать ключами неудобно. Строка+столбец также однозначно определяют элемент, поэтому их можно выбрать в качестве составного ключа.

Трамплины



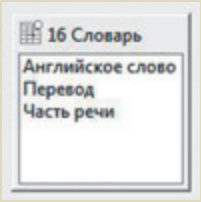
С формальной точки зрения возможно совпадение двух записей по всем пяти полям, поэтому даже составной ключ найти невозможно. На практике таких совпадений пока не случалось.

Чаты службы поддержки



Это весьма специфическая задача. Некоторые сайты имеют службу поддержки в виде on-line-чата. Обычно по завершении чата пользователь может оценить работу специалиста. Ключа в этой таблице нет — специалист службы поддержки может общаться сразу с несколькими клиентами в окне чата, поэтому возможно совпадение сразу трех полей. Следовательно, ни простых, ни составных ключей у таблицы нет.

Словарь



В качестве составного ключа можно выбрать английское слово и его перевод. Если вам удастся привести контрпример, авторы будут вам благодарны (эта задачка часто бывает головоломной — многим кажется, что контрпример есть, но пока его никто не привел).

Новая интерактивная приставка

MimioTeach™

превращает вашу классную доску в интерактивную.

Всё, что Вам нужно от интерактивной доски,
но проще, удобнее и быстрее.
И за меньшие деньги.

Компактное и недорогое решение, позволяющее использовать в качестве интерактивной доски обычные маркерные и меловые доски.

MimioTeach™ была создана учителями и для учителей, поэтому обучиться работать с этой приставкой очень просто. Легкость установки и полнофункциональное программное обеспечение позволяют начать активно использовать **MimioTeach™** на уроке практически сразу же после покупки.

Традиционные интерактивные доски громоздки и дороги, каждую доску можно использовать только в одном классе. Приставка **MimioTeach™** компактна, ее легко переносить из класса в класс, а по окончании урока можно забрать с собой и хранить в учительской.

MimioTeach™ обходится значительно дешевле обычной интерактивной доски. При этом технологически приставка полностью соответствует возможностям интерактивных досок, а часто и превосходит их.

Обратите внимание: **MimioTeach™** является частью комплексного интерактивного решения **MimioClassrom™**, включающего в себя также документ-камеру, систему тестирования и другое интерактивное оборудование.

Мы предлагаем нашим пользователям широкие возможности бесплатного обучения: семинары, курсы повышения квалификации и дистанционные обучающие курсы всегда к услугам учителей, использующих **Mimio®**.

Узнайте больше, посетите наш сайт или позвоните:

http://www.mimioclass.ru

8 (800) 5555-33-0

Звонок по России бесплатный

ООО «Рене» — генеральный дистрибьютор Mimio

DYMO | Mimio®

О чем это?
О! В двух словах описать сложно ☺. Вкратце: Google придумал новый язык программирования, а наш автор Иван Сукин из замечательного города программистов Переславля-Залесского попробовал в нем разобраться. У Ивана-то получилось, но предупреждаем сразу — в целом статья сложная. Однако ее можно читать, просто пропуская все сугубо технические места — все равно интересно.

И это все?
Нет. На диске имеются примеры программы для начального знакомства с языком Dart.

Dart, или Куда целится Google

О “зачинщиках” и их мотивах

Dart — это новый язык программирования, предложенный компанией Google для упрощения веб-разработки и преодоления фундаментальных недостатков существующих решений. В силу значительной известности и неординарности этой корпорации было бы просто неприлично оставить описание этого языка без небольшой предыстории.

Немного истории

Американская компания Google, известная рядовому пользователю Интернета своей поисковой системой, почтовым сервером, “ультрабыстрым” браузером и надоедливой контекстной рекламой (а также, в последнее время, новой социальной сетью), популярна среди программистов еще по нескольким причинам. Google — корпорация, старающаяся быть на переднем крае инноваций, сама создающая их, следующая принципу, озвученному в свое время Аланом Кэем[1]: “*Лучший способ предсказать будущее — изобрести его самому*”. Google не боится экспериментировать и отбраковывать результаты неудачных экспериментов, среди сотрудников компании подобное качество ценится, пожалуй, превыше всего. Эта статья об одном из подобных экспериментов.

В последние годы еще одним хобби Google стало “коллекционирование” разработчиков и дизайнеров языков программирования и операционных систем, тех людей, которых в англоязычной литературе называют емким словом “masterminds”. Среди них такие известные люди, как Роб Пайк, стоявший у истоков Unix и Plan 9, Кен Томпсон, один из создателей такого известного языка, как Си, и Гвидо ван Россум, автор языка Python, на котором основана значительная часть современных веб-сервисов Google. Засветился также Джеймс Гослинг, создавший в начале 90-х годов невероятно популярный сегодня язык Java. В дальнейшем я вернусь к этому хобби Google, потому что с ним тесно связаны имена “виновников торжества”.

Поскольку речь пойдет о новом языке программирования, а эта практика в Google имеет прецеденты, было бы нелишне сказать пару слов о предыдущем подобном опыте и попытаться экстраполировать его на то, что мы имеем сейчас. Вышеупомянутому Робу Пайку, являющемуся

[1] По иронии судьбы Алан имеет косвенное, но значительное отношение к предмету статьи.

И.А. Сукин

одним из апологетов старой школы UNIX-систем, во время его работы в Google было предложено разработать язык программирования, который смог бы заменить Си в привычной для последнего нише — системном программировании, с учетом всех современных принципов разработки вычислительных систем, главный из которых — мультипроцессорность и многопоточность, параллелизация. Пайк уже пытался осуществить подобное, являясь членом команды по проектированию языков Alef в Plan 9 и Limbo в операционной системе Inferno (причиной низкой популярности обоих языков отчасти является низкая популярность их программных платформ), поэтому задача была для него сравнительно простой. В команде с Робертом был также Кен Томпсон, один из авторов Си, как никто другой знающий недостатки своего детища. Новый язык, названный Go, увидел свет в 2009 году и имел действительно серьезные возможности обставить Си. Однако этого не произошло. Одним из главных виновников этого является *консервативность* практической информатики, вытекающая из больших объемов уже существующего кода, который никому не хочется переписывать (а потом еще и отлаживать) заново и на новом языке. Поскольку какой-либо другой серьезной цели, кроме замены Си в качестве языка для системного программирования, у Go не было, он завис в воздухе, и в настоящее время его положение не совсем ясно.

Возможно, причиной низкой популярности Go является и то, что область проектирования операционных систем и среднеуровневых языков пока не слишком близка Google (а в 2007 году это было для них вообще в новинку), и “первый блин вышел комом”. Как бы то ни было, компания сильно не расстроилась и продолжила экспериментировать. Здесь мы уже подходим к объекту нашей статьи — языку Dart.

Обновленный веб

В 2011 году Google решила сконцентрировать свои экспериментаторские силы на привычном для себя поприще — веб-разработке. Как известно людям, знакомым с этой областью не понаслышке, ключевой технологией разработки клиентских (то есть работающих на компьютерах конечных пользователей) веб-приложений является Javascript (JS), объектно-ориентированный язык программирования, в наше время ставший уже укоренившимся стандартом в создании сайтов, порталов и даже пользовательских веб-интерфейсов для операционных систем. Именно на него и его недостатки, которые трудно устранить с помощью простой эволюции языка: динамическую типизацию, прототипную объектную модель, низкую производительность, как следствие двух предыдущих минусов, сложность поддержки и разработки больших проектов, — обратили свое внимание инженеры из Google. Ими были предложены две стратегии дальнейших действий в этом направлении:

- *Harmony* — стратегия с маленькими рисками, но и маленькой отдачей — продолжать работу над улучшением Javascript. Это просто и безопасно, однако даже в лучшем случае это займет годы и все равно в конце концов упрется в фундаментальные недостатки JS. Собственно говоря, это стратегия не столько Google, сколько консорциума Javascript.
- *Dash (то, что мы сейчас знаем как Dart)* — стратегия с большими рисками и большой отдачей — разработать новый язык программирования, который был бы пропитан динамическим духом Javascript, но имел бы значительно улучшенную производительность и обеспечивал бы легкость управления крупными проектами. В будущем — закрепить Dash (Dart) в качестве открытого стандарта веб-программирования и обеспечить поддержку нового языка со стороны браузеров, а также реализовать его транслятор в Javascript для браузеров, таковой поддержки не имеющих. Это крайне сложно и рискованно, поскольку потребуется убедить в правомерности и необходимости такого решения множество программистов.

Попытка применения только одной из этих стратегий заранее обречена на провал. Если следовать пути “обновления Javascript”, то веб может зависнуть в устаревшем состоянии и оказаться неспособным конкурировать с более инновационными, но зачастую менее открытыми технологиями. При изолированном применении второго пути в безнадежно устаревшем состоянии может оказаться сам Javascript, а новый язык при этом будет еще слишком “сырым”, что приведет к “смутным временам” веб-программирования.

Единственным верным решением в данном случае является развитие двух этих стратегий одновременно: новый язык будет создаваться, однако в качестве страховки в то время, когда он будет еще слишком слаб (или если он все же не оправдает возложенных на него ожиданий), будет эволюционировать старый добрый Javascript.

В заключение этого обзора предпосылок для появления нового языка веб-программирования я рассмотрю также некоторые другие родственные технологии от Google:

- *JSPrime* — попытка объединить Javascript и Google Web Toolkit в единую систему, пригодную для разработки больших высоконагруженных веб-приложений (технологий, являющихся приоритетными для Google). В настоящее время все разработчики JSPrime входят в команду по созданию Dart, в который была перенесена часть уже сделанных наработок;
- *JS++* — множество улучшений и нововведений для Javascript, которые в настоящее время являются частью первой стратегии, Harmony;
- *Joy* — система шаблонных и Model-View-Controller фреймворков, которые будут построены поверх Dart.

Что же это за зверь?

Настало время вплотную подойти к описанию предмета нашего разговора — второй стратегии развития веба, языка программирования Dart.

Как я уже говорил ранее, Google имеет страсть к "коллекционированию" лучших умов computer science, используя их наработки в собственных экспериментах. Dart в этом плане не стал исключением: его проектированием занималась большая команда специалистов, среди которых двое — Гилад Брача и Ларс Бэк — являются поистине мастерами в вопросах создания динамических языков программирования. Оба они связаны с сообществом Smalltalk — языка, положившего основы объектно ориентированного программирования в его современном виде. Создателем этого языка и ключевой фигурой в этом сообществе является уже упомянутый мной Алан Кэй. По своей динамической природе Smalltalk является сравнительно медленным языком, что всегда считалось одним из его ключевых минусов. Однако Гилад и Ларс в свое время уже провели эксперимент, подобный описываемому мной, — эксперимент по устранению фундаментальных недостатков Smalltalk. Так родились два языка: Strongtalk и Self, — которые при дальнейшем развитии предопределили появление высокопроизводительных объектно ориентированных технологий. Брача и Бэк сыграли в этом далеко не последнюю роль. Уже в середине девяностых годов Бэк принимал участие в разработке очень быстрой Java-машины — Hotspot, которая и по сей день является основной виртуальной машиной Java. Можно сделать вывод, что Dart уже изначально был в хороших руках.

Каковы же основные принципы, лежащие в основе Dart? Сами разработчики выделяют три ключевых перспективы:

• *Производительность* — Dart проектировался с оглядкой на скорость выполнения программ, столь низкую для всех современных реализаций Javascript. Виртуальные Dart-машины могут быть очень быстрыми за счет таких особенностей языка, как, например, необязательная *статическая* типизация.

• *Удобство использования* — Dart является языком, сравнимым по простоте с Javascript, поскольку усложнение языка могло бы оттолкнуть от него начинающих и даже бывалых веб-разработчиков от области, традиционно считающейся простой. В этом также помогает *необязательная* статическая типизация и динамическая природа языка в целом.

• *Легкая поддержка дополнительных инструментов* — таких, как "умные" среды разработки и многочисленные фреймворки. Плохая поддержка со стороны IDE — один из важнейших минусов Javascript. Dart же был спроектирован таким образом, чтобы комплексные среды разработки могли осуществлять такие действия, как простой рефакторинг кода и поиск мест, откуда вызываются методы. В упрощении этого важную роль играет замена прототипной объектной системы на классовую.

Также выделяется пять основных целей разработки Dart:

• *Dart должен быть структурированным и в то же время гибким языком программирования для веба.* Часто сильная структурированность и ортогональность языка мешают его гибкому использованию, однако можно найти золотую середину где-то на пути между статическими и динамическими языками. Как раз в этой нише находится Dart.

• *Dart должен быть особенно легким и естественным языком для людей, которые уже программировали до этого на чем-либо.* К сожалению, естественнее всего он выглядит для C-, Java- и Javascript-программистов. Люди же, предпочитающие языки семейства Pascal или вообще функциональное программирование, сочтут это утверждение Google голословным.

• *Спроектировать Dart так, чтобы код на нем выполнялся быстро и время запуска приложения было минимальным.* По поводу этого пункта пока мало что можно сказать, поскольку единственным способом запуска Dart-кода является кросс-компиляция в Javascript.

• *Создать возможность запуска приложений на Dart на самых разных, в том числе мобильных устройствах.* Для поддержки устаревших или необычных устройств можно использовать кросс-компилятор в Javascript (правда, при этом Google не учитывает, что JS поддерживается тоже далеко не всеми устройствами. Возможно, с развитием Dart появятся его кросс-компиляторы в другие языки).

• *Разработать средства для запуска Dart во всех современных веб-браузерах.* К сожалению, в этом направлении пока наблюдается самый незначительный прогресс — разработчики браузеров, кроме Chrome, не спешат поддерживать начинание конкурента. Разработчики браузера Internet Explorer от Microsoft даже напрямую высказались о своем неудовлетворении языком Dart и идеями Google. Вероятно, на этом пути будет еще немало препятствий.

Безопасность в Dart не является краеугольным камнем разработки, однако ей уделено достаточно внимания в тех местах, где это не входит в противоречие с тремя вышеупомянутыми принципами.

Dart создавался как чрезвычайно гибкий и переносимый язык, который мог бы работать не только бок о бок или вместо Javascript, но и на стороне сервера вместе с приложениями на Java, Python, Ruby, PHP. Планируется, что будет четыре основных варианта исполнения кода, написанного на Dart:

• *Клиентская Dart-машина, встроенная в веб-браузер.* Специалисты Google надеются, что в ближайшем будущем разработчики наиболее популяр-

ных браузеров реализуют поддержку Dart в своих продуктах.

• *Серверная виртуальная машина.* Dart проектировался как язык, который может выполняться и на стороне сервера, при этом серверная Dart-машина может быть масштабируемой. Судя по всему, Google хочет использовать Dart в качестве основного серверного языка в своей сети.

• *Кросс-компилятор.* В настоящее время, то есть на ранних этапах развития Dart (а версия Dart пока что — 0.0.5), это основной способ запустить Dart-программу. Программа транслируется в аналогичную программу на Javascript, которая может исполняться любым интерпретатором JS. После появления в браузерах самостоятельных Dart-машин Google планирует использовать кросс-компилятор для поддержки устаревших платформ и систем. Также одной из целей стратегии Harmony является развитие этой Javascript-поддержки для Dart. Разумеется, что преимущества вроде улучшенной производительности при таком способе исполнения не будут проявляться.

• *Образ системы.* Этот способ выполнения Dart-кода не является самостоятельным, это просто еще один вариант подачи кода в виртуальную машину. Вместе с Dart распространяются инструменты, позволяющие сделать образ приложения, выполняющегося на Dart-машине, с оптимизированным кодом (хотя бы потому, что он имеет двоичный, а не текстовый вид). В дальнейшем этот образ может быть запущен на любой виртуальной машине Dart. Идею образов язык позаимствовал у таких систем, как Smalltalk, Self и Forth.

Оглядываясь на мой рассказ о Go и его слабой возможности работать с уже имеющимся Си-кодом, справедливо задать подобный вопрос и о Dart: с помощью каких механизмов и насколько удобно реализована поддержка существующих наработок на Javascript в нем? К сожалению, ответа на этот вопрос не могут дать даже сами разработчики языка.

Подробно описав и оценив основные принципы и идеи, которыми руководствовались разработчики, проектировавшие Dart, стоит перейти к более приземленным материям и описать возможности самого языка.

Основные отличительные характеристики Dart таковы:

• *Классовая объектно ориентированная модель с одиночным наследованием и интерфейсами.* Существуют два типа истинно (таких, в которых все является объектом) объектно ориентированных моделей: классовые и прототипные. В первых центральным понятием является *класс*, к которому применимы действия *инстанцирования* (создания объекта класса) и *наследования.* При этом, если наследование одиночное, то между классами сохраняется строгая древовидная иерархия. Иногда для гибкости наследование может быть множественным, однако в таком случае иерархия классов становится графом общего вида, и работать с ней и делать предположения для автоматического вывода типов куда сложнее. В настоящее время в качестве замены множественному наследованию предлагаются *интерфейсы*. В прототипных объектных системах понятия класса нет и *объекты* существуют сами по себе, и к ним применима операция *клонирования.* После клонирования клон может быть произвольно изменен. Прототипные объектные системы обеспечивают большую гибкость, чем классовые ценой серьезного снижения производительности. Объектная модель Javascript — прототипная, и это одна из причин его медленной скорости и появления Dart.

• *Необязательная статическая типизация,* так часто упоминавшаяся мной ранее. Традиционные динамические языки вроде Javascript используют динамическую типизацию, то есть типы всех выражений и переменных в них известны лишь во время выполнения программы, что сильно осложняет оптимизацию кода, однако статическая типизация снижает гибкость программирования. В Dart использован уникальный подход, который пока не применялся ни в одном из широко распространенных языков (кроме, пожалуй, Cython — языка написания расширений для Python): вы можете указать тип переменной, а можете не указывать. Если вы указали тип, то будут применены соответствующие оптимизации и, если необходимо, выведены предупреждения о несовместимости типов. Примечательно, что “тип” динамически типизированных переменных входит в иерархию типов Dart и называется Dynamic.

• *Овеществленное обобщенное программирование.* Под обобщенным программированием понимают возможность создания сложных типов данных (списков, массивов, таблиц, объектов) с параметрически задаваемым элементарным типом. Читатели, знакомые с C++, имели дело с обобщенным программированием, используя шаблоны. Что же дает право обобщенному программированию в Dart называться *овеществленным*? В языках вроде C++ отождествление контейнера с типом происходит на этапе компиляции / другой обработки кода. В Dart же элементарный тип сосуществует вместе с контейнером во время выполнения, и операция отождествления также производится во время выполнения. Это дает еще одну возможность выбора между динамической и статической типизацией.

• *Действительно лексические области видимости.* Многие динамические языки используют динамические области видимости переменных: это значит, что переменная связывается с тем окружением, которое в данный момент является текущим. Dart же использует лексические области видимости: переменная в них связывается с тем окружением, в котором она была определена в коде программы. Это позволяет повысить производительность и избежать сложнодиагностируемых ошибок.

• *Однопоточность.* Весь код на Dart исполняется в одном потоке, это позволяет избежать сложноотлаживаемых ошибок вроде гонок за ресурсы. Для обеспечения конкурентности используется акторная модель, которую я опишу ниже.

• *Анонимные, или лямбда-функции.* Эта возможность, широко распространившаяся в императивных языках и пришедшая из функциональных, позволяет определять функции на ходу, не задавая для них никакого имени. Это часто бывает полезно, когда используются функции, чьими аргументами также являются функции (например, разнообразные функции поиска и сортировки).

• *Шаблонная подстановка на уровне строк.* Востребованная и широко используемая в веб-программировании, позволяющая подставлять значения переменных и даже целых выражений в строковые литералы при создании последних. При этом стоит помнить, что Dart использует лексические области видимости переменных.

Как вы могли заметить из описания мной некоторых особенностей языка, Dart настолько сложен, насколько сложна ваша программа и насколько сложный язык вам нужен. Вы можете писать на Dart простые динамические программы в Javascript-стиле или же можете писать сложные и высокопроизводительные системы, используя всю мощь статической типизации и объектной системы Dart. Данная особенность является уникальной, я не смогу привести пример еще одного распространенного языка программирования, сложность которого мог бы определять сам программист.

Вы, вероятно, уже ждете от меня практических примеров программирования на Dart и пребываете в недоумении, почему я несколько тяну с ними. К сожалению, Dart пока лишь ненамного ушел от состояния proof-of-concept и является активно меняющимся и нестабильным языком, многие инструменты для него пока не созданы. Однако возможность опробовать его у нас есть уже сейчас. Для этого можно использовать несколько способов: для простейших манипуляций с языком можно использовать веб-интерфейс *Dartboard* по адресу *http://try.dartlang.org*, правда, примеры с действительным веб-программированием (например, примеры, использующие canvas) не будут там работать. Специально созданная среда для программирования на Dart — *DartEditor* в данный момент находится в чрезвычайно "сыром" состоянии: периодически возникают серьезные недочеты, мешающие нормальному программированию в ней. В целом DartEditor является довольно амбициозным проектом, в нем можно опробовать все примеры Dart-кода, однако я хочу сказать, что читатель должен его использовать только на свой страх и риск. Третий вариант удобного написания программ на Dart — использование *Dart-плагина для Eclipse*. К сожалению, этот плагин находится в еще более "сыром" состоянии, чем упомянутый мной ранее DartEditor. Самым сложным вариантом является ручное использование кросс-компилятора Dart и создание веб-страниц с получившимися скриптами на Javascript или запуск этих скриптов в отдельных интерпретаторах Javascript. Пока что этот вариант является наиболее надежным для тех, кто хочет попробовать все существующие на данный момент возможности Dart, правда, стоит заметить, что для этого читатель должен быть хоть немного искушен в веб-программировании. Увы, кросс-компилятор пока тоже находится в состоянии альфа-тестирования, имеет некоторые недочеты, а ни одного бинарного релиза его пока не было выпущено, поэтому я могу порекомендовать его только "экстремалам"-знатокам, умеющим и желающим собирать кросс-компилятор из исходных кодов.

Ввиду описанных мной проблем, я не могу заставить читателя использовать сложные способы запуска кода и на протяжении оставшейся части статьи буду использовать Dartboard, также нескромно предполагая, что у человека, интересующегося веб-программированием, есть свободный доступ в Интернет. Для запуска сложных примеров читателю придется либо окунуться во все проблемы "сырого" программного обеспечения, либо дождаться выхода стабильных версий необходимых инструментов.

Все "вкусности" крупным планом

Итак, мы переходим к непосредственной работе с Dart и решили использовать для этой цели Dartboard (для этого необходимо перейти в браузере на адрес *http://try.dartlang.org*). Интерфейс Dartboard предельно прост — изначально это поле для ввода кода с подсветкой ключевых слов и тремя-четырьмя кнопками в панели инструментов над ним.

Первая кнопка позволяет запустить введенный код, при этом внизу открывается окно вывода, в которое перенаправляется текстовый вывод программы. Если при интерпретации кода обнаружены ошибки или спорные ситуации, строки, содержащие их, будут подсвечены специальным маркером (желтым или красным). К сожалению, сообщения об ошибках пока либо не очень информативны, либо отсутствуют вовсе. Также при интерпретации введенного вами кода в панели инструментов появляется ссылка, с помощью которой вы можете делиться вашими примерами с коллегами.

Вторая кнопка (с флажком-чекбоксом) позволяет включать проверку соответствия типов переменных во время исполнения. Это полезно, если вы хотите написать корректную и стабильную программу, однако может привести к значительному замедлению выполнения кода.

Третья кнопка, появляющаяся только тогда, когда вы вносите в Dartboard какие-то изменения, по-

зволяет вернуться к первоначальному состоянию. Учтите, что при этом может потеряться весь код, введенный вами.

Четвертая кнопка просто позволяет развернуть окошко Dartboard до границ веб-страницы в браузере.

Простые шаги

Начнем наше рассмотрение с простейшего кода, который вы можете видеть в окне Dartboard:

```
// Welcome to try.dartlang.org
//
// Here you can try out the Dart Language
// from the comfort of your own
// *modern* web browser. When you run the
// code … it is submitted to AppEngine,
// translated into the equivalent
// JavaScript, and run right in your
// browser window.
//
// Ok, your turn!
main() {
  print('Give try.dartlang.org a try.');
}
```

Здесь вы можете видеть уже привычный в наше время С-подобный синтаксис: //-комментарии (также поддерживаются многострочные /* … */ комментарии), фигурные скобки для отделения блоков друг от друга и точку с запятой для разделения выражений языка. Функция main имеет тот же смысл, что и аналогичная функция в языке Си — в ней происходят вычисления основного потока программы. Подобно Си Dart — язык, чувствительный к регистру, имена “Test” и “test” в нем обозначают разные сущности.

Простые операторы и конструкции вроде операторов ветвления if, switch и циклов позаимствованы из С-подобных языков, поэтому я не буду уделять им особое внимание.

В Dart имеются все стандартные встроенные типы — double, int, num — обобщение числовых типов, bool, String. Важное уточнение: не стоит забывать инициализировать переменные — поскольку встроенные типы являются объектами, начальные значения переменных равны null, а не, к примеру, 0. В отличие от многих других языков здесь встроенные типы реализованы в виде интерфейсов, что позволяет без проблем смешивать их свойства при создании собственных типов, наследуемых от базовых. Отмечу также, что именно поддержка в Dart виртуальных конструкторов на уровне интерфейсов позволила так гибко реализовать систему встроенных типов.

Переменные в Dart определяются также просто и очевидно, особенно для тех, кто знаком с Javascript: либо с помощью слова “var”, либо с помощью указания типа (приведенные ниже определения вы можете вбивать в функцию main):

```
some_value = 4; // неправильно!
var some_value = 4; // правильно
```

```
nt a = 4;
double b = 4; // предупреждение
int c = 4.1;  // также предупреждение,
              // тип динамически
              // приводится к double
```

```
iDynamic a = 4.1; // нет предупреждения
a = "test";       // нет предупреждения
Object b = 4;     // нет предупреждения
b = "test1";      // нет предупреждения
insdfs a = 5; // предупреждение, тип a
              // динамически приводится
              // к Dynamic
```

Как вы видите, типизация в Dart — слабая, и типы переменных по необходимости приводятся, вызывая предупреждения при видимой несовместимости. Как и во всех других объектно ориентированных языках, приведение типа наследника к типу-суперклассу (или к типу Dynamic) не вызывает предупреждений. Типы изначально статически типизированных переменных также приводятся к статическим типам. Уникальной и в то же время спорной возможностью является указание несуществующего или неправильного имени типа. В этом случае Dart-система выдает предупреждение и автоматически приводит тип определяемой переменной к Dynamic.

Одной из самых простых и в то же время мощнейших возможностей Dart является подстановка шаблонов в строки. Похожие средства существуют во многих языках программирования, однако те языки, в которых такие инструменты поставляются, что называется, “из коробки”, можно пересчитать по пальцам (например, Tcl), большинство предпочитает реализовывать шаблонную подстановку в виде отдельной библиотеки. Польза такого подхода была очевидна раньше, в эпоху малых вычислительных ресурсов, когда нагружать интерпретатор такими механизмами было непозволительной роскошью. В настоящее же время, как мне кажется, существуют все предпосылки для включения шаблонной подстановки в базовую систему языка, и я полностью поддерживаю Dart в этом. Рассмотрим несколько примеров (в примерах будет формироваться строка с некоторым именем, например, s; чтобы проверить результат, вы можете вывести ее с помощью выражения print(s)):

```
String s = "Test string ${2+3}";
```

```
int a = 4;
String s = "Test string ${a+3};
```

```
int a = 4;
double b = 5.1; // без предупреждения
String s = "Test string ${a+b};
```

```
int a = 4;
double b = 5.1;
// будет подставлено только значение a
String s = "Test string $a+b";
```

```
int a = 4;
double b = 5.1;
String s = "Test string " + a + b;
// значения a и b будут преобразованы
// в строки и все три имеющиеся строки
// будут склеены в одну
```

Как видно, в Dart существуют три различных способа включения переменных и выражений в строки: от самого мощного с фигурными скобками до простейшей конкатенации строк с необходимыми типовыми преобразованиями переменных.

В мире объектов

Этот подраздел будет посвящен объектно ориентированному программированию на Dart. Если вы имеете опыт программирования на Javascript, но не имеете опыта работы с классово ориентированными языками, некоторые понятия будут для вас в новинку. Также вам придется не совсем просто, если вы вообще не имеете понятия об объектно ориентированном программировании.

Объектно ориентированное программирование тесно связано с имитационным моделированием и ставит задачу не формального описания мира, а описания его структурных и функциональных параметров. В связи с этим возникла модель мира как набора элементарных “объектов”, обладающих сложной иерархией взаимоотношений. Однако в свете двойственности характеристик этой модели — “структуры” и “функций”, она распалась на две подмодели: “структурный функционализм” классово ориентированного программирования и “функциональный структурализм” прототипно ориентированного программирования. С прототипно ориентированным программированием знакомы все программисты на Javascript, однако не все из них знают, что появилось оно значительно позднее классового варианта из-за серьезных проблем эффективной реализации. Я не буду углубляться в особенности этого подхода, поскольку предметом обсуждения являются не языки семейства ECMAScript. Google созданием Dart, как я уже говорил ранее, решил убрать прототипно ориентированное программирование из веба, заменив его комплексной классовой системой с поддержкой интерфейсов, обеспечив нужную гибкость без обременительных затрат на дополнительные вычисления.

Классы в Dart создаются и описываются довольно просто (определять классы нужно вне функций и функции `main` в том числе):

```
// вне функции main
class TestMe {
    String name = "Test";
    test() {
        print("Here I am: ${name}");
    }
}
// в функции main
TestMe a = new TestMe();
a.test();
```

Из этого примера можно видеть, что синтаксически класс в Dart — это просто блок кода, в котором определяются поля-переменные и методы-функции. Определения полей и методов могут идти в любом порядке. Для читателей, знакомых только с языками с ручным управлением памяти, такими, как C++ и Pascal, стоит сказать, что в Dart используется автоматическая сборка мусора, и созданные объекты не нужно удалять вручную.

Если вы любитель языков Object Pascal и C++, вы можете спросить: где же определения областей видимости вроде public или private? Как и во многих других динамических по своей природе языках, в Dart подобного понятия не существует, это наследие Smalltalk, и такой тон особенно ярко выражен в таких языках, как Python или Ruby. Все поля в классе по умолчанию являются public, кроме тех, что начинаются с символа подчеркивания, которые не определены как private, но считаются таковыми по соглашению. К примеру:

```
// считаем, что класс TestMe уже определен
// и объект a инициализирован
// ошибки нет, name теперь равно
// "Look at me!"
a.name = "Look at me!";
class TestMe {
    String _name = "Test";
    test() {
        print("Here I am: ${_name}");
    }
}
// в main
TestMe a = new TestMe();
// также нет ошибки, однако это плохой тон
a._name = "Look at me!";
```

Как я уже сказал в комментарии к вышеприведенному коду, изменение поля, чье имя начинается с подчеркивания, — дурной тон. Более того, в настоящих ОО-языках дурным тоном считается прямое изменение любого поля объекта. Для этого вводятся так называемые “методы-аксессоры”, которые служат для управляемого доступа к полю. Они могут быть геттерами — методами, читающими значение поля, и сеттерами — методами, записывающими в поле соответствующее значение. Подобная практика очень удобна, например, в тех случаях, когда необходимо производить проверку вводимых данных из текстового поля, и является дальнейшим развитием простой инкапсуляции.

```
class TestMe {
   // private по соглашению
   String _name = "Test";
   // геттер
   String get name() => _name;
   // здесь также используется
  // лямбда-выражение, обозначаемое как =>
    void set name(String value) {
    // void здесь можно опустить
        if (value == null) value = "";
        _name = value;
    }
```

```
    test() {
        print("Here I am: ${name}");
       // можно использовать и _name
    }
}
```

В примере выше сеттер используется для того, чтобы значение поля _name было правильной строкой, даже если ее пытаются сделать объектом-пустышкой. Сущность name, полученная с помощью методов-аксессоров, называется *свойством*. В силу динамической природы языка все это является лишь соглашениями хорошего тона, например, никто не мешает явно установить поле _name, однако это приведет лишь к запутанному коду, который сложно поддерживать.

Наследование в Dart очень просто реализуется с помощью ключевого слова “extends” (при этом, как я уже говорил, все поля и методы наследуются как public):

```
// считаем, что класс TestMe уже
// определен в предыдущем примере
class AnotherTester extends TestMe {
    set name(String value) {
    // переопределяем сеттер
        if (value == null) value = "";
        _name = value + ", ha-ha!";
    }
    test() { // переопределяем test
     super.test(); //вызываем родительский
      // метод test
        print("Sure it's not me,
               it's: ${name}");
    }
}
```

Так куда же все-таки целится Google?

Суммируя все вышесказанное, можно сделать вывод, что Dart является языком, занимающим промежуточное место между Java и Javascript. Это место является промежуточным не только относительно динамичности/статичности языка, но и относительно областей применения — клиентского и серверного веб-программирования. Многие возможности, анонсированные в Dart, уже входят в существующие языки, но ни один из этих языков не реализует эти возможности вместе и с такой же гибкостью, как предмет нашего обсуждения. Когда я только столкнулся с Dart, он сильно напомнил мне Python, с которым я знаком уже довольно продолжительное время. Я вяло посматривал в сторону творения Google, говоря себе: “Ну что там может быть такого интересного, чего я еще не видел в том же Python или в конце концов Ruby?”. К счастью, в один прекрасный момент я все же нашел в себе силы и попытался вникнуть в примеры на Dart и спецификацию языка. Моему удивлению практически не было предела: я увидел там почти все то, что мне хотелось бы видеть в Python, — необязательную статическую типизацию, интерфейсы, акторы. С тех пор я посвятил изучению Dart больше времени, чем раньше. Мои знакомые программисты (не только на Python) испытали похожие чувства от знакомства с новичком. Поэтому, резюмируя, можно сказать, что Google нацелился не просто на веб-программирование, а на скриптовые языки в целом, и Dart является преодолением недостатков не только и не столько Javascript, сколько всех распространенных динамических языков. Насколько успешными будут их начинания — войдет ли Dart в свою нишу победоносно или же станет еще одним внутренним языком Google, подобно Go, — покажет время.

Что же дальше?

В рамках небольшой статьи нельзя многого сказать о языке программирования, особенно таком неординарном, как Dart. Все также усугубляется тем, что язык вышел совсем недавно, имеет пока версию 0.0.5, находится в стадии альфа-тестирования и активно развивается. Многих инструментов и заявленных возможностей пока нет. При написании этой статьи я старался не сухо описать сам язык, как это сделано в спецификации, а выделить те моменты, благодаря которым он уже до выхода приобрел статус чего-то “революционного”, основные аспекты, ради которых он создавался, — такие, как смешанная типизация и невероятная динамическая гибкость в разумном сочетании со статической строгостью, то новое, чего пока нет ни в одном из распространенных языков. При этом за бортом остались такие более сугубо технические моменты, как акторная модель изолятов, встраивание Dart в веб-страницы, обобщенное программирование, классы-фабрики, работа в современных IDE, и многое другое. Часть из этих возможностей я постарался показать в примерах к статье, имеющихся на диске, за остальными вам придется обращаться к спецификации, писать самим методом проб и ошибок или ждать, пока появится большое сообщество Dart-программистов. Серьезной литературы по языку пока нет, поэтому основными источниками информации являются официальный сайт, форумы и списки рассылки. Несмотря ни на что, Dart выглядит очень многообещающим проектом, и кто знает, каково будет его положение и как он будет выглядеть ко времени своего первого официального релиза.

Использованные источники информации

1. Официальный сайт языка Dart [Электронный ресурс], URL: *http://www.dartlang.org*.

2. *Bracha G.* On the Interaction of Method Lookup and Scope with Inheritance and Nesting, 3rd ECOOP Workshop on Dynamic Languages and Applications, 2007.

3. Future of Javascript [Электронный ресурс], e-mail из внутренней рассылки Google, URL: *http://markmail.org/message/uro3jtoitlmq6x7t*.

4. The Dart Programming Language Specification 0.0.5 [Электронный ресурс], The Dart Team, URL: *http://www.dartlang.org/docs/spec/dartLangSpec.html*.

Повышение квалификации

Педагогический университет «Первое сентября»

описание курсов и подача заявок: edu.1september.ru

Педагогический университет «Первое сентября» создан в 2003 г. На основании договоров о сотрудничестве взаимодействует с факультетом педагогического образования МГУ им. М.В. Ломоносова и Московским институтом открытого образования. Университет специализируется на проведении дистанционных (заочных) и очных краткосрочных курсов повышения квалификации для работников образовательных учреждений. Курсы ведут признанные специалисты в области образования: разработчики стандартов, программ, авторы учебников, опытные методисты.

Лицензия Департамента образования г. Москвы 77 № 000349, рег. № 027477 от 15.09.2010

Дистанционные курсы

Для всех педагогов вне зависимости от места проживания

Нормативный срок освоения курса: 72 часа.
По окончании: УДОСТОВЕРЕНИЕ О ПОВЫШЕНИИ КВАЛИФИКАЦИИ УСТАНОВЛЕННОГО ОБРАЗЦА.
Объем предложения: более 120 курсов актуальной тематики по 21 образовательно-педагогическому направлению.
Учебный процесс организован в два потока.
Второй поток 2011/12 учебного года:
Прием заявок: до 15 января 2012 г.
Выдача удостоверений: сентябрь 2012 г.
Первый поток 2012/13 учебного года:
Прием заявок: с 1 апреля до 30 сентября 2012 г.
Выдача удостоверений: май 2013 г.
Стоимость обучения с первого потока 2012/13 учебного года: 2190 руб. (без видеоподдержки); 2390 руб. (с видеоподдержкой).

Модульные курсы

Для всех педагогов вне зависимости от места проживания

«Навыки личной эффективности» – цикл дистанционных модульных курсов.
Нормативный срок освоения курса: 6 часов.
По окончании: СЕРТИФИКАТ.
Объем предложения: цикл из 7 дистанционных модульных курсов по психологии, менеджменту и экономике.
Учебный процесс: индивидуальный.
График изучения модуля определяется слушателем.
Минимальный срок обучения по одному модулю – 1 месяц.
Итоговая работа: тест в режиме on-line.
Прием заявок: круглогодично.
Стоимость обучения с 1 марта 2012 года: 300 руб. за один модуль.

Очные курсы

Для жителей Москвы и Московской области

Нормативный срок освоения курса: 72 часа.
По окончании: УДОСТОВЕРЕНИЕ О ПОВЫШЕНИИ КВАЛИФИКАЦИИ ГОСУДАРСТВЕННОГО ОБРАЗЦА.
Объем предложения: более 30 курсов актуальной тематики по 10 образовательно-педагогическим направлениям.
Учебный процесс организован в три потока.
Второй поток 2011/12 учебного года:
Прием заявок: до 31 января 2012 г.
Расписание занятий: раз в неделю с февраля по апрель .
Выдача удостоверений: май 2012 г.
Интенсив 2011/12 учебного года:
Прием заявок: до 31 мая 2012 г.
Расписание занятий: ежедневно в первой половине июня 2012 г.
Выдача удостоверений: сентябрь 2012 г.
Первый поток 2012/13 учебного года: Прием заявок: с 1 апреля до 30 сентября 2012 г. Расписание занятий: раз в неделю с октября по декабрь.
Выдача удостоверений: февраль 2013 г.
Стоимость обучения с первого потока 2012/13 учебного года: 5900 руб.

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ г. МОСКВЫ
ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»

ММ МОСКОВСКИЙ ПЕДАГОГИЧЕСКИЙ МАРАФОН УЧЕБНЫХ ПРЕДМЕТОВ 2012

20 МАРТА – 13 АПРЕЛЯ

РАСПИСАНИЕ ДНЕЙ ПЕДАГОГИЧЕСКОГО МАРАФОНА

20 марта	**Открытие** День классного руководителя
21 марта	День школьного психолога
22 марта	День здоровья детей, коррекционной педагогики, логопеда, инклюзивного образования и лечебной физической культуры
	День учителя технологии (ЦО № 293)
23 марта	День учителя начальной школы (день первый)
24 марта	День учителя начальной школы (день второй)
25 марта	День дошкольного образования
27 марта	День учителя географии
28 марта	День учителя химии
	День учителя ОБЖ (Кадетская школа № 1784)
29 марта	День учителя биологии
30 марта	День учителя информатики
31 марта	День учителя физики
1 апреля	День учителя математики
3 апреля	День учителя истории
4 апреля	День учителя МХК, музыки и ИЗО
5 апреля	День школьного и детского библиотекаря
6 апреля	День учителя литературы
7 апреля	День учителя русского языка
8 апреля	День учителя английского языка
10 апреля	День учителя французского языка
11 апреля	День учителя немецкого языка
12 апреля	День учителя физической культуры
13 апреля	День школьной администрации **Закрытие**

marathon.1september.ru

! Обязательная предварительная регистрация на все дни Марафона откроется 20 февраля 2012 года на сайте marathon.1september.ru.

! Каждый участник Марафона, посетивший три мероприятия одного дня, получает официальный именной сертификат (6 часов)

В дни Марафона ведущие издательства страны представляют книги для учителей

Начало работы – 9.00. Завершение работы – 15.00

УЧАСТИЕ БЕСПЛАТНОЕ. ВХОД ПО БИЛЕТАМ

РЕГИСТРИРУЙТЕСЬ, РАСПЕЧАТЫВАЙТЕ СВОЙ БИЛЕТ И ПРИХОДИТЕ!

Место проведения Марафона: лицей № 1535, ул. Усачева, дом 50 (в 3 минутах ходьбы от станции метро «Спортивная»)
Место проведения Дня учителя технологии: ЦО № 293, ул. Ярославская, д. 27 (ст. метро «ВДНХ»)
Место проведения Дня учителя ОБЖ: Кадетская школа № 1784, ул. Скаковая, д. 20 (ст. метро «Динамо», «Белорусская»)

По всем вопросам обращайтесь по телефону **8-499-249-3138** или по электронной почте: marathon@1september.ru

MICROSOFT POWERPOINT УГЛУБЛЕННО

“Спусковой крючок” анимации

А.И. Азевич, Москва

*Встречалось ли вам когда-нибудь звучное слово “триггер”? На первый взгляд кажется, это что-то из области физики. Близко, да не совсем. А может, это остров, пока еще неизвестный даже заядлым путешественникам? Нет, такого острова на карте не найти. Впрочем, зачем гадать, заглянем лучше в энциклопедию. Она наверняка знает о триггере все. Вот что о нем написано в Большой Советской Энциклопедии: “**Триггер** (англ. **trigger**) — спусковое устройство (спусковая схема), которое может сколь угодно долго находиться в одном из двух (реже многих) состояний устойчивого равновесия и скачкообразно переключаться из одного состояния в другое[1]”. По свидетельству умной книги триггер — это что-то вроде спускового крючка…*

В программе Microsoft PowerPoint тоже встречается триггер. Но подготовка компьютерных презентаций к переключателям и спусковым крючкам не имеет никакого отношения. Что же тогда тут делает триггер? Попробуем разобраться.

В школе часто приходится готовить презентации. Мир растений, ареалы обитания животных, история вычислительной техники, великие географические открытия — великое множество увлекательных визуальных докладов! Они демонстрируются на уроках, конференциях, форумах. Одни презентации выглядят живыми, содержательными и интерактивными. Их хочется смотреть. Щелкаешь мышкой по слайду: цветы распускаются; зайчики прыгают; пути мореплавателей светятся на карте замысловатой кривой, огибающей материки. Другие школьные презентации — непрерывное чередование скучного текста и статических картинок. Их создатели наверняка не имеют понятия о триггере. Жаль, ведь с помощью “компьютерного щелчка” любое выступление будет смотреться куда интересней!

Что же такое триггер? Триггер — это интерактивное средство анимации, позволяющее задать действие *выделенному элементу*. Его можно применить к любому объекту на слайде (это может быть фрагмент текста, изображение или звук). Триггер, как и кнопка, срабатывает по щелчку левой кнопки мыши, при этом в момент наведения указателя мыши на объект он (указатель) принимает вид руки с отогнутым указательным пальцем.

Чтобы не осталось никаких неясностей относительно триггера, покажем все на примере.

Предположим, что вам поручено подготовить игру для младших школьников. Небольшое, но ответственное поручение классного руководителя, и к нему стоит отнестись серьезно и ответственно. В игре дети должны выбрать из предложенного набора ягоды (см. *рис.* 1). Овощи и грибы должны исчезнуть с экрана компьютера, как только к ним “прикоснется” мышь.

Правила игры очень просты: при нажатии на ягоду изменяется положение или размеры картинки и одновременно слышится одобрительный звук; как только касаемся любого другого объекта, он исчезает и раздается грустный возглас.

Рис. 1

[1] Именно поэтому, очевидно, триггером называют электронное устройство с двумя или более устойчивыми состояниями. — **Прим. ред.**

Создадим слайд презентации. Для этого предварительно подготовим необходимый материал: картинки ягод, овощей, грибов, а также звуковые файлы (одобрительный — для верных ответов, грустный — для ошибочных). Соответствующие графические и звуковые файлы представлены на диске к данному номеру журнала.

После создания слайда приступим к созданию первого триггера. А вообще их у нас будет столько, сколько изображений на слайде.

Как сделать так, чтобы при нажатии на клубнику она завертелась, при нажатии на бруснику, чернику и арбуз[2] — они увеличились в размерах, а при щелчке на овощах и грибах соответствующие картинки моментально пропадали? Кроме того, хотелось бы, чтобы одновременно с анимацией воспроизводились нужные звуки. Этим и займутся триггеры!

Начнем с клубники. Выделяем картинку с этой ягодой и задаем для нее эффект анимации. Порядок действий следующий: **Настройка анимации — Добавить эффект — Выделение — Вращение** (*рис.* 2).

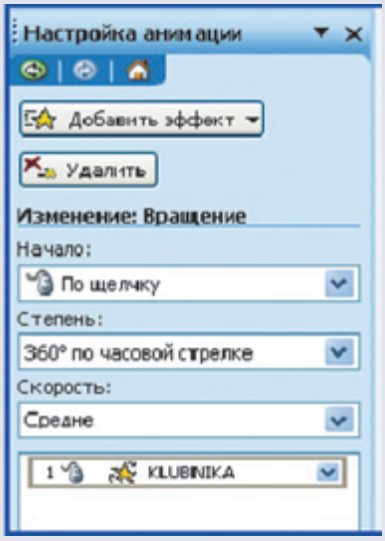


Рис. 2

Можно установить параметры `Начало`, `Степень` и `Скорость`, а также при необходимости посмотреть, как будет протекать эффект (кнопка **Просмотр**). Обратим внимание, что имя эффекта совпадет с именем графического файла с изображением клубники (в нашем случае — KLUBNIKA), с типом автофигуры и т.п.

При нажатии на рисунок клубники компьютер еще должен издать “одобрительный” звук. Такие звуки есть в самой программе PowerPoint, их можно скачать из Интернета, предварительно набрав в любом из поисковых сервисов “коллекция звуков”. Вставим в презентацию два звуковых файла: один — одобрительно-веселый (его имя VERNO), другой — пронзительно-грустный (NEVERNO). Для вставки звуков необходимо выполнить команды **Вставка — Звук из файла**, после выбрать нужный файл. При этом стоит поместить значок звука за границы слайда, чтобы его не было видно при показе презентации.

А дальше приступим к созданию триггера. Для этого щелкнем правой кнопкой мыши на панели анимации по названию файла — картинки клубники и в открывшемся контекстном меню выберем команду **Время** (*рис.* 3), а в диалоговом окне **Вращение** (*рис.* 4) — вкладку `Время`.

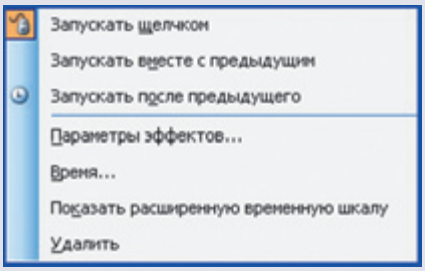


Рис. 3

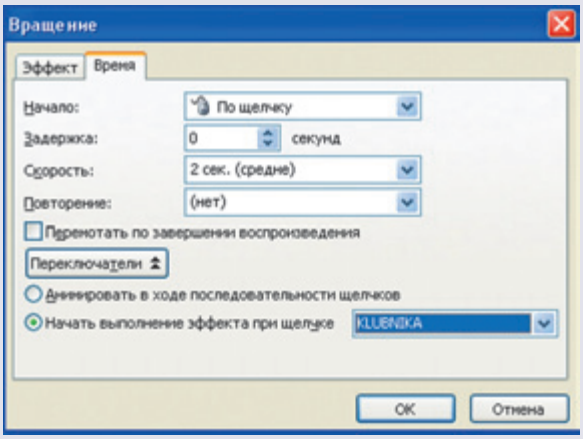


Рис. 4

Затем нажмем на кнопку **Переключатели** и активируем радиокнопку **Начать выполнение эффекта при щелчке**. Выбираем в списке пункт KLUBNIKA и нажимаем **ОК**.

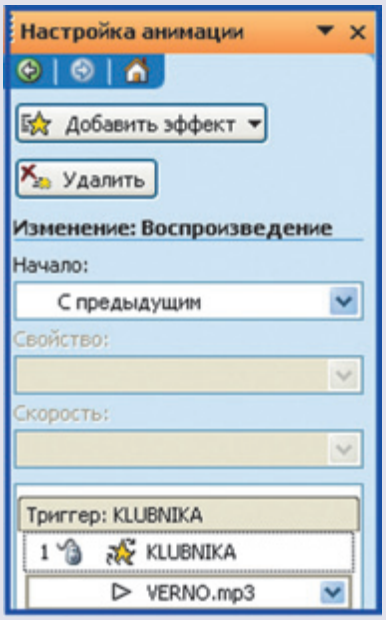


Рис. 5

[2] Обратим внимание, что арбуз — это ягода. — **Прим. ред.**

Выполнив действия, заметим, что на панели **Настройка анимации** появился триггер с именем KLUBNIKA (*рис.* 5), а на слайде около изображения клубники — значок руки (это информация о том, что с изображением связан триггер и по щелчку на нем начнутся эффекты анимации) — см. *рис.* 6.

Рис. 6

После этого добавим в триггер звук. Для этого выделяем звуковой файл и на панели **Настройка анимации** нажимаем последовательно **Добавить эффект — Действия со звуком — Воспроизведение**. Перетаскиваем надпись `Verno` (звуковой файл) вдоль панели так, чтобы она оказалась в нашем триггере (*рис.* 5). Правой кнопкой мыши выделяем звуковой файл и в контекстном меню выбираем пункт `Запускать вместе с предыдущим`. Это означает, что при демонстрации презентации, как только мы щелкнем по клубнике, компьютер издаст также одобрительный звук. Первый триггер готов!

Аналогично подготовим триггеры для других ягод и овощей. При этом у всех изображений на слайде появится значок руки (*рис.* 7). А на панели **Настройка анимации** будет целых семь триггеров (по числу рисунков). Конечно, триггеров может быть и больше, это зависит от характера заданий, которые они должны выполнить.

Рис. 7

Теперь запустим презентацию и посмотрим, как все работает. Убеждаемся, что объекты двигаются, пропадают, звучат то радостные, то грустные звуки. Ягоды остаются, овощи и грибы исчезают.

Используя триггеры, несложно придумать самые разнообразные игры. Незаменимы они в случае, если надо привлечь внимание слушателей к презентации. Пригодятся они для подготовки учебных тестов, проверочных заданий, викторин. Да что говорить — с помощью триггеров можно изобрести еще не такое! Тем, кто пока не умеет пользоваться этой интерактивной функцией программы PowerPoint, можно посоветовать внимательно и скрупулезно ее изучить. Как знать, когда настанет момент нажать “спусковой крючок” в содержательной, интересной и живой презентации? “Выстрел” должен быть точным!

От редакции. Создайте всю презентацию с игрой, описанной в статье. Триггеры, “привязанные” к другим (кроме клубники) изображениям, разработайте самостоятельно. Придумайте и создайте также собственные презентации, в которых используются триггеры. Результаты присылайте в редакцию. Авторы лучших работ будут награждены дипломами.

ЗАДАЧНИК

Ответы, решения, разъяснения к заданиям, опубликованным в газете “В мир информатики” № 167 (“Информатика” № 13/2011)

1. Задача “Четыре сообщения”

Напомним, что необходимо было из четырех полученных сообщений:

1) 110100000100110011;
2) 111010000010010011;
3) 110100001001100111;
4) 110110000100110010

определить единственное сообщение, которое прошло без ошибки и которое может быть корректно декодировано при следующей таблице кодировки:

В	К	А	Р	Д
000	11	01	001	10

Решение

Начиная с первого символа, будем разбивать каждое сообщение на части, соответствующие допустимым кодам:

1) первое сообщение:
11 01 000 001 001 10 011
— получается недопустимый код (выделен красным);

2) второе сообщение:
11 10 10 000 01 001 001 1
— последний код — недопустимый;

3) третье сообщение:
11 01 000 01 001 10 01 11
— все коды — допустимые (соответствуют слову КАВАРДАК);

4) четвертое сообщение:

11 01 10 000 10 01 10 01 0

— последний код — недопустимый.

Ответ — третье сообщение.

Правильные ответы прислали:

— Андрющенко Александр и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Батракова Полина, Бетхер Алексей, Дукач Светлана, Клименко Надежда, Максимова Ксения и Фалалеева Надежда, г. Лесосибирск Красноярского края, поселок Стрелка, школа № 8 им. Константина Филиппова, учитель **Лопатин М.А.**;

— Глазкова Екатерина и Семяшкин Иван, Республика Коми, г. Сыктывкар, МОУ "Лицей народной дипломатии", учитель **Гранаткина О.М.**;

— Демьянова Елена, Костюнин Александр и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Долгополов Сергей, средняя школа поселка Озеры Красноярского края, учитель **Филипченко И.С.**;

— Зайнакаев Вадим, Коробейникова Полина, Кузнецова Екатерина, Ожгихина Екатерина и Смагин Сергей, Республика Башкортостан, Краснокамский р-н, село Николо-Берёзовка, школа № 1, учитель **Ситдикова А.Г.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Неофитова Елена, средняя школа села Янтиково, Чувашская Республика, учитель **Неофитова Н.Н.**;

— Махмутов Роберт, Сухоруков Антанас и Фазлыев Фанис, Республика Башкортостан, г. Стерлитамак, школа № 24, учитель **Орлова Е.В.**;

— Холодилова Дарья и Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**;

— Шейкин Александр, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**

Отметим ответ Александра Шейкина, приведшего подробное обоснование ответа. Заметим также, что в ряде ответов недопустимые варианты исключались с точки зрения осмысленности сообщения.

2. Задача "Три брата"

Напомним, что необходимо было назвать имена старшего, среднего и младшего из трех братьев, если Ваня был не старше Коли, а Саша не старше Вани.

Решение

Расположим имена ребят в порядке возрастания возраста (при просмотре слева направо).

Из первого факта следует, что этот порядок такой:

Ваня Коля

а из второго:

Саша Ваня Коля

Ответ. Старший — Коля, средний — Ваня, младший — Саша.

Правильные ответы представили:

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Валуев Федор, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Дукач Светлана и Клименко Надежда, г. Лесосибирск Красноярского края, поселок Стрелка, школа № 8 им. Константина Филиппова, учитель **Лопатин М.А.**;

— Зайнакаев Вадим, Коробейникова Полина, Кузнецова Екатерина, Ожгихина Екатерина и Смагин Сергей, Республика Башкортостан, Краснокамский р-н, село Николо-Берёзовка, школа № 1, учитель **Ситдикова А.Г.**;

— Куценко Евгения и Парамонова Анастасия, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Сухоруков Антанас и Фазлыев Фанис, Республика Башкортостан, г. Стерлитамак, школа № 24, учитель **Орлова Е.В.**;

— Чернова Ксения, поселок Надвоицы, школа № 1, учитель **Каликина Л.М.**

3. Чайнворд "Соточка"

Ответы. 1–11. Пиктограмма. 11–13. Ада. 13–21. Ассемблер. 21–26. Россия. 26–31. Ячейка. 31–35. Абзац. 35–38. Цикл. 38–44. Лавлейс. 44–49. Спрайт. 49–54. Трафик. 54–63. Клавиатура. 63–73. Архитектура. 73–79. Адаптер. 79–83. Растр. 83–88. Разряд. 88–93. Датчик. 93–97. Кобол. 97–100. Лисп (или Лого).

Правильные ответы прислали:

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Демьянова Елена, Костюнин Александр, Ломакина Елена и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Дукач Светлана и Клименко Надежда, г. Лесосибирск Красноярского края, поселок Стрелка, школа № 8 им. Константина Филиппова, учитель **Лопатин М.А.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Махмутов Роберт, Республика Башкортостан, г. Стерлитамак, школа № 24, учитель **Орлова Е.В.**;

— Неофитова Елена, средняя школа села Янтиково, Чувашская Республика, учитель **Неофитова Н.Н.**;

— Шадрина Юлия, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Воеводина Р.В.**;

— Чернова Ксения, поселок Надвоицы, школа № 1, учитель **Каликина Л.М.**;

— Шейкин Александр, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**;

— Щукин Андрей, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

4. Задача "Как проехать из A в B?"

Ответ. Условие: "Минимальная стоимость проезда от станции **A** до станции **B** не больше 6" выполняется для таблицы 3. Маршрут движения **A** — **C** — **E** — **B** (стоимость равна 6).

Правильные ответы представили:

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Демьянова Елена, Костюнин Александр и Хомякова Анна, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Дукач Светлана и Клименко Надежда, г. Лесосибирск Красноярского края, поселок Стрелка, школа № 8 им. Константина Филиппова, учитель **Лопатин М.А.**;

— Зайнакаев Вадим, Коробейникова Полина, Кузнецова Екатерина, Ожгихина Екатерина и Смагин Сергей, Республика Башкортостан, Краснокамский р-н, село Николо-Берёзовка, школа № 1, учитель **Ситдикова А.Г.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Сухоруков Антанас и Фазлыев Фанис, Республика Башкортостан, г. Стерлитамак, школа № 24, учитель **Орлова Е.В.**;

— Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**

5. Числовой ребус с "ЭММОЙ"

Напомним, что необходимо было решить числовой ребус

МА × **МА** = **ЭММА**

Решение

Есть только три цифры, которые, будучи умноженными на саму себя, дают число, оканчивающееся на эту же цифру: 0, 1, 5 и 6.

Значение **A** = 0 не подходит явно (квадрат числа, оканчивающегося на 0, оканчивается на 00).

Для анализа варианта **A** = 1 запишем ребус "в столбик" (символом "*" может быть любая цифра, кроме 0):

		×	**M**	1
			M	1
			M	1
*	*		**M**	
Э	**M**	**M**	1	

Но **M** + **M** = **M** только при **M** = 0, что недопустимо.

Проведем анализ варианта **A** = 5:

		×	**M**	5
			M	5
	*		**X**	5
*	*		**У**	
Э	**M**	**M**	5	

Можно сказать, что цифра **X** равна 2 при четной **M** и **X** = 7 при нечетной **M**. А цифра **У** — соответственно, 0 и 5. Это означает, что в любом случае цифра **M** в результате равна 2. Но 25^2 — трехзначное число, т.е. вариант **A** = 5 не подходит. Итак, **A** = 6.

Проверка остальных цифр **M** (**M** ≥ 3) показывает, что решением ребуса является: 76 × 76 = 5776.

Правильные ответы прислали:

— Андрющенко Александр, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Батракова Полина, Дукач Светлана, Клименко Надежда и Максимова Ксения, г. Лесосибирск Красноярского края, поселок Стрелка, школа № 8 им. Константина Филиппова, учитель **Лопатин М.А.**;

— Глазкова Екатерина и Семяшкин Иван, Республика Коми, г. Сыктывкар, МОУ "Лицей народной дипломатии", учитель **Гранаткина О.М.**;

— Зайнакаев Вадим, Коробейникова Полина, Кузнецова Екатерина, Ожгихина Екатерина и Смагин Сергей, Республика Башкортостан, Краснокамский р-н, село Николо-Берёзовка, школа № 1, учитель **Ситдикова А.Г.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Любимов Антон, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Махмутов Роберт, Республика Башкортостан, г. Стерлитамак, школа № 24, учитель **Орлова Е.В.**;

— Холодилова Дарья и Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**;

— Шейкин Александр, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**

Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**, для решения задачи разработал программу на языке Паскаль.

6. Задача "Пять бусинок"

Напомним, что необходимо было установить, какая из четырех цепочек бусинок, помеченных буквами *M*, *N*, *O*, *P*, *S*:

1) SMP; 3) SNO;
2) MSO; 4) OSN,

— создана с соблюдением следующих правил:

А) в середине цепочки стоит одна из бусин *M*, *O*, *S*;

Б) на третьем месте должна быть любая гласная, если первая буква согласная, и любая согласная, если первая гласная;

В) на первом месте стоит одна из бусин *O*, *P*, *S*, не стоящая в цепочке в середине,

а также определить, сколько вариантов цепочек можно составить согласно указанным требованиям.

Решение первой задачи

Проверим соответствие цепочек каждому требованию.

Требованию А не соответствует цепочка 3.

Требованию В не соответствует цепочка 2.

Так как в цепочке 1 первая буква согласная, то на третьем месте должна стоять любая гласная, а это требование не соблюдается.

Итак, все три требования соблюдаются в цепочке 4.

Решение второй задачи

С соблюдением требования А возможны 3 варианта:

	M	
	O	
	S	

С учетом также требования В возможны 7 вариантов:

O	*M*	
P		
S		
P	*O*	
S		
O	*S*	
P		

С соблюдением требования Б варианты с первой согласной (их 5) буквой дают один вариант каждый, с первой гласной буквой (их два) — дают четыре варианта каждый. Итак, общее число допустимых вариантов равно:

$$5 \times 1 + 2 \times 4 = 13.$$

Правильные ответы представили:

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Глазкова Екатерина и Семяшкин Иван, Республика Коми, г. Сыктывкар, МОУ “Лицей народной дипломатии”, учитель **Гранаткина О.М.**;

— Долгополов Сергей, средняя школа поселка Озеры Красноярского края, учитель **Филипченко И.С.**;

— Зайнакаев Вадим, Коробейникова Полина, Кузнецова Екатерина, Ожгихина Екатерина и Смагин Сергей, Республика Башкортостан, Краснокамский р-н, село Николо-Берёзовка, школа № 1, учитель **Ситдикова А.Г.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Любимов Антон, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Харламов Виталий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**

Задача “Любительница мороженого”

Напомним, что необходимо было определить, какое мороженое и с какими фруктами можно приготовить Ирине по ее вкусу, если из перечня:

1) пломбир с орехами;
2) пломбир с бананами;
3) пломбир с черникой;
4) шоколадное с черникой;
5) шоколадное с клубникой

в четырех вариантах ей не нравились или тип мороженого, или наполнитель, а в одном варианте — ни мороженое, ни наполнитель.

Решение

Допустим, что Ирина любит мороженое типа пломбир. Тогда она не любит наполнитель в виде орехов (это следует из пункта 1 перечня), бананов (пункт 2) и черники (пункт 3). Кроме того, раз она любит пломбир, то ей не нравится шоколадное мороженое (пункты 4 и 5 перечня). При этом против клубники она ничего не имеет (пункт 5 перечня является четвертым из тех, в которых девушке не нравились или тип мороженого, или наполнитель, а пункт 4 — единственный, в котором ей не нравится ни мороженое, ни наполнитель). Итак, при сделанном допущении она не против съесть только пломбир с клубникой.

Можно также составить таблицу:

Вариант в перечне	Тип	Обоснование	Наполнитель	Обоснование
1) пломбир с орехами;	Нравится	Из допущения	Не нравится	Из условия
2) пломбир с бананами;	Нравится	Из допущения	Не нравится	Из условия
3) пломбир с черникой;	Нравится	Из допущения	Не нравится	Из условия
4) шоколадное с черникой;	Не нравится	Из допущения	Не нравится	См. выше
5) шоколадное с клубникой	Не нравится	Из допущения	Нравится	Из условия

Допустим, что девушка любит шоколадное мороженое. Тогда она не любит наполнитель в виде черники (это следует из пункта 4 перечня) и клубники (пункт 5). Составим таблицу:

Вариант в перечне	Тип	Обоснование	Наполнитель	Обоснование
1) пломбир с орехами;	Не нравится	Из допущения		
2) пломбир с бананами;	Не нравится	Из допущения		
3) пломбир с черникой;	Не нравится	Из допущения	Не нравится	См. ниже
4) шоколадное с черникой;	Нравится	Из допущения	Не нравится	Из условия
5) шоколадное с клубникой	Нравится	Из допущения	Не нравится	Из условия

Из нее следует, что орехи и бананы в мороженом она ест (пункты 1 и 2). Поскольку в условии речь шла о наполнителе — фрукте, то получается, что шоколадное мороженое с бананом ей можно приготовить.

Ответ. Задача имеет два варианта решения: Ирине можно приготовить пломбир с клубникой или шоколадное мороженое с бананом.

Ответы прислали:

— Андрющенко Александр, Куценко Евгения, Парамонова Анастасия и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Долгополов Сергей, средняя школа поселка Озеры Красноярского края, учитель **Филипченко И.С.**;

— Зайнакаев Вадим, Коробейникова Полина, Кузнецова Екатерина, Ожгихина Екатерина и Смагин Сергей, Республика Башкортостан, Краснокамский р-н, село Николо-Берёзовка, школа № 1, учитель **Ситдикова А.Г.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Любимов Антон, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Харламов Виталий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**

Решение задачи “День рождения” и объяснение фокуса “Отгадывание задуманного двузначного числа” представил Базылев Юрий, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**

Компьютерные программы, моделирующие игру “крестики-нолики”, разработали:

— Базылев Юрий, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Журавлев Алексей, Лукина Полина и Пехов Андрей, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Зуйков Денис, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Кузнецов Денис и Сосновцев Илья, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**

Все перечисленные читатели будут награждены дипломами. Отдельно отметим ответ Андрея Пехова, который разработал не только два варианта программы, предложенные в статье (выбор хода компьютером “наугад” и “осмысленный” выбор хода им), но и “промежуточный” вариант.

Списки читателей, приславших ответы “обычной” почтой, будут опубликованы в следующем выпуске.

Встреча друзей

Шесть друзей — Моисеев, Потапов, Ефимов, Дмитриев, Алексеев и Осипов — закончили один университет. И вот однажды они снова встретились в санатории на берегу Черного моря и весь вечер, сидя за круглым столом, рассказывали о своей работе, о планах на будущее. Один из них стал видным литератором, другой — биологом, третий — инженером, четвертый — капитаном, пятый — юристом, шестой — физиком.

За столом они расположились так: юрист сел против Ефимова, литератор — против Осипова, который расположился между капитаном и юристом, биолог — против Дмитриева, рядом с литератором и слева от Алексеева. Инженер оказался между капитаном и литератором. Моисеев поместился рядом с биологом, против физика.

Попробуйте определить специальность каждого из друзей.

Шахматный турнир

В шахматном турнире играли пять шахматистов. Турнир проводился в один круг. Известно, что игрок А набрал 3,5 очка, Б — 1,5, В — 2,5 и Д — тоже 2,5 очка, причем последний не имел ни одного поражения. Результаты игрока Г неизвестны.

Требуется заполнить турнирную таблицу, установив, как сыграл каждый шахматист со своими соперниками, и определить результат игрока Г.

Покупка

Четверо товарищей купили вместе одну вещь. Первый внес половину суммы, внесенной остальными, второй — треть суммы, внесенной остальными, третий — четверть суммы, внесенной остальными,

а четвертый внес 130 рублей. Сколько стоит вещь и сколько внес каждый?

Числа на доске

На доске написано число 1. Каждую секунду к числу на доске прибавляется сумма его цифр. Может ли через некоторое время на доске появиться число 123456? Ответ получите путем рассуждений и/или разработав программу на языке программирования, который вы изучаете.

Соревнования по гимнастике

В соревнованиях по гимнастике участвовали Алла, Валя, Сима, Даша. Были высказаны предположения:

1) Сима будет первой, Валя — второй;
2) Сима будет второй, Даша — третьей;
3) Алла будет второй, Даша — четвертой.

Оказалось, что в каждом предположении только одно высказывание верно. Какое место заняла каждая из девушек?

СЕМИНАР

Как в компьютере проводится вычитание?

В предыдущей публикации [1] было показано, что реле можно соединить проводами в более сложные устройства — логические вентили и создать из них двоичный сумматор — устройство для сложения двух двоичных чисел. Возникает вопрос: “А как же вычитание?” Задавая его, не опасайтесь прослыть занудой ☺. На самом деле вы просто осмотрительны. Сложение и вычитание определенным образом дополняют друг друга, но механика двух этих операций различна. Сложение выполняется последовательно от крайнего правого столбца цифр к крайнему левому. Перенос из каждого столбца прибавляется к следующему столбцу. При вычитании мы не *переносим,* а *заимствуем,* и это приводит к внутренне отличной последовательности действий, усложненной своего рода движением вперед и назад.

Рассмотрим типичную задачу на вычитание, усложненную заимствованием:

$$\begin{array}{cccc} & 2 & 5 & 3 \\ - & 1 & 7 & 6 \\ \hline & ? & ? & ? \end{array}$$

Начнем с крайнего правого столбца. Очевидно, что 6 больше 3, поэтому приходится занять 1 у 5 и вычесть 6 из 13. Получается 7. Поскольку мы заняли у пятерки единицу, она превратилась в 4. Эта цифра меньше 7, и мы занимаем 1 у 2 и вычитаем 7 из 14. Получаем 7. Вспоминаем, что заняли 1 у 2, так что вместо 2 имеем 1 и вычитаем 1 из 1. Получаем 0. Окончательный ответ 77.

$$\begin{array}{cccc} & 2 & 5 & 3 \\ - & 1 & 7 & 6 \\ \hline & & 7 & 7 \end{array}$$

Как нам заставить набор вентилей разобраться в столь запутанной логике?

На самом деле не стоит даже пробовать. Вместо этого мы прибегнем к небольшой хитрости, которая позволит выполнять вычитание *без* заимствования. Подробное исследование этого способа вычитания полезно с теоретической точки зрения, так как он напрямую связан с методикой хранения двоичных отрицательных чисел в компьютерах.

Для начала вспомним, что числа, участвующие в вычитании, называются *уменьшаемым* и *вычитаемым.* Вычитаемое вычитается из уменьшаемого, а результат называется *разностью*.

$$\begin{array}{cc} & \text{Уменьшаемое} \\ - & \text{Вычитаемое} \\ \hline & \text{Разность} \end{array}$$

Чтобы обойтись в вычитании без заимствования, во-первых, вычтем вычитаемое не из уменьшаемого, а из 999:

$$\begin{array}{cccc} & 9 & 9 & 9 \\ - & 1 & 7 & 6 \\ \hline & 8 & 2 & 3 \end{array}$$

Здесь мы используем 999, поскольку вычитаемое является трехзначным числом. Если бы оно было четырехзначным, мы вычли бы его из 9999. Результат вычитания числа из строки девяток называется *дополнением до девяти*. Дополнение до девяти числа 176 есть 823. Верно и обратное: дополнение до девяти числа 823 есть 176. И, что приятно, — каким бы ни было вычитаемое, вычисление его дополнения до девяти *никогда не требует заимствования.*

Вычислив дополнение числа до девяти, прибавьте к нему уменьшаемое: +1 сейчас.

$$\begin{array}{cccc} & 2 & 5 & 3 \\ + & 8 & 2 & 3 \\ \hline 1 & 0 & 7 & 6 \end{array}$$

Наконец, сложите результат с 1 и вычтите 1000:

$$\begin{array}{ccccc} & 1 & 0 & 7 & 6 \\ & & & + & 1 \\ - & 1 & 0 & 0 & 0 \\ \hline & & & 7 & 7 \end{array}$$

Готово! Мы получили тот же результат, что и раньше, без единого заимствования.

Как это получилось? Исходная задача на вычитание выглядит так:

$$253 - 176$$

К этому выражению можно прибавить и вычесть любое число — результат от этого не изменится. Прибавим и вычтем 1000.

$$253 - 176 + 1000 - 1000$$

Это выражение эквивалентно выражению:

$$253 - 176 + 999 + 1 - 1000$$

Числа можно перегруппировать:

$$253 + (999 - 176) + 1 - 1000$$

Это как раз и есть тот расчет, что мы провели, используя дополнение вычитаемого до девяти. Мы заменили одно вычитание двумя вычитаниями и двумя сложениями, попутно освободившись от всех заимствований.

А что делать, если вычитаемое больше уменьшаемого? Ведь задача на вычитание может выглядеть и так:

$$\begin{array}{rrrr} & 1 & 7 & 6 \\ - & 2 & 5 & 3 \\ \hline & ? & ? & ? \end{array}$$

Первое, что приходит на ум: “Хм. Я вижу, что вычитаемое больше уменьшаемого. Я меняю числа местами и произвожу вычитание, помня при этом, что результат будет отрицательным”. Ответ записывается следующим образом:

$$\begin{array}{rrrr} & 1 & 7 & 6 \\ - & 2 & 5 & 3 \\ \hline & - & 7 & 7 \end{array}$$

Чтобы выполнить такой расчет без заимствования, придется поступить несколько иначе, чем в предыдущем примере. Начинаем опять с вычисления дополнения вычитаемого (253) до девяти:

$$\begin{array}{rrrr} & 9 & 9 & 9 \\ - & 2 & 5 & 3 \\ \hline & 7 & 4 & 6 \end{array}$$

Теперь складываем дополнение до девяти с уменьшаемым:

$$\begin{array}{rrrr} & 1 & 7 & 6 \\ + & 7 & 4 & 6 \\ \hline & 9 & 2 & 2 \end{array}$$

В предыдущей задаче на этом этапе нужно было прибавить 1 и вычесть 1000, чтобы получить окончательный ответ. В данном случае эта стратегия не сработает. Пришлось бы вычесть 1000 из 923, а это в реальности оборачивается вычитанием 923 из 1000 и требует заимствований.

На самом деле, вычисляя дополнение вычитаемого до девяти, мы фактически прибавили к нему 999, поэтому теперь будем вычитать 999:

$$\begin{array}{rrrr} & 9 & 2 & 2 \\ - & 9 & 9 & 9 \\ \hline & ? & ? & ? \end{array}$$

Глядя на этот пример, мы понимаем, что ответ будет отрицательным. Поэтому числа нужно поменять местами и вычесть 922 из 999. Это действие снова не требует заимствования. Ответ, как мы и ожидали, равен 77:

$$\begin{array}{rrrr} & 9 & 9 & 9 \\ - & 9 & 9 & 2 \\ \hline & & 7 & 7 \end{array}$$

При вычитании двоичных чисел используется точно такой же метод. Более того, он проще, чем в десятичном случае. Посмотрим, как он работает.

Вот как выглядит наша задача на вычитание:

$$\begin{array}{rrrr} & 2 & 5 & 3 \\ - & 1 & 7 & 6 \\ \hline & & 7 & 7 \end{array}$$

Записав эти числа в двоичном представлении, получаем задачу такого вида:

$$\begin{array}{rr} & \texttt{11111101} \\ - & \texttt{10110000} \\ \hline & \texttt{????????} \end{array}$$

Шаг 1. Вычтем вычитаемое из 11111111 (т.е. из 255):

$$\begin{array}{rr} & \texttt{11111111} \\ - & \texttt{10110000} \\ \hline & \texttt{01001111} \end{array}$$

Когда мы работали с десятичными числами, вычитаемое вычиталось из строки девяток, а результат назывался дополнением до девяти. При расчетах с двоичными числами вычитаемое вычитается из строки единиц, и результат называется *дополнением до единицы* (по аналогии с дополнением до девяти). Но заметьте: чтобы получить дополнение до единицы, вычитать на самом деле не нужно! И вот почему: каждый 0 в исходном числе становится 1 в дополнении, а каждая 1 превращается в 0. Именно поэтому дополнение до единицы называют также *обратным,* или *инвертированным*, *кодом* [2].

Шаг 2. Складываем обратный код вычитаемого с уменьшаемым:

$$\begin{array}{rr} & \texttt{11111101} \\ + & \texttt{01001111} \\ \hline & \texttt{101001100} \end{array}$$

Шаг 3. Прибавляем к результату 1:

$$\begin{array}{rr} & \texttt{101001100} \\ + & \texttt{1} \\ \hline & \texttt{101001101} \end{array}$$

Шаг 4. Вычитаем 100000000 (т.е. 256):

$$\begin{array}{rr} & \texttt{101001101} \\ - & \texttt{100000000} \\ \hline & \texttt{1001101} \end{array}$$

Этот результат соответствует десятичному числу 77.

Теперь повторим последовательность действий, выполненных для тех же чисел, но с заменой вычитаемого на уменьшаемое. В десятичном виде эта задача записывалась так:

$$\begin{array}{rrrr} & 1 & 7 & 6 \\ - & 2 & 5 & 3 \\ \hline & ? & ? & ? \end{array}$$

а в двоичном она выглядит так:

```
  10110000
– 11111101
----------
  ????????
```

Шаг 1. Вычитаем вычитаемое из 11111111, чтобы получить его обратный код.

```
  11111111
– 11111101
----------
  00000010
```

Шаг 2. Прибавляем обратный код к уменьшаемому.

```
  10110000
+ 00000010
----------
  10110010
```

Теперь из результата нужно вычесть 11111111. Когда вычитаемое было меньше уменьшаемого, эта задача решалась добавлением 1 и вычитанием 100000000. Теперь сделать это без заимствования не удастся, поэтому мы вычитаем полученный результат из 11111111:

```
  11111111
– 10110010
----------
  01001101
```

Как и ранее, это действие эквивалентно инвертированию. В ответе получилось 77, но мы помним, что в действительности это –77.

Теперь у нас есть все необходимые познания для модернизации сумматора из предыдущей публикации, чтобы он выполнял не только сложение, но и вычитание. Чтобы не слишком усложнять задачу, наша новая суммирующая и вычитающая машина будет выполнять вычитание, только когда вычитаемое меньше уменьшаемого.

Вспомним, что основой суммирующей машины был 8-битовый сумматор, собранный из логических вентилей:

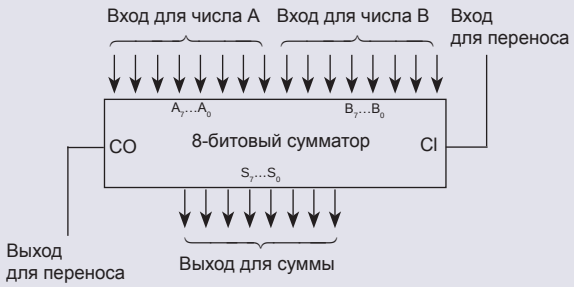


Рис. 1

Как вы помните, входы с A_7 по A_0 и с B_7 по B_0 соединялись с переключателями, задававшими два 8-битовых слагаемых. Вход для переноса соединялся с землей. Выходы с S_7 по S_0 соединялись с восемью лампочками, отображавшими результат сложения. Поскольку результатом сложения могла быть 9-битовая величина, выход для переноса соединялся с девятой лампочкой.

Пульт управления сумматором выглядел примерно так:

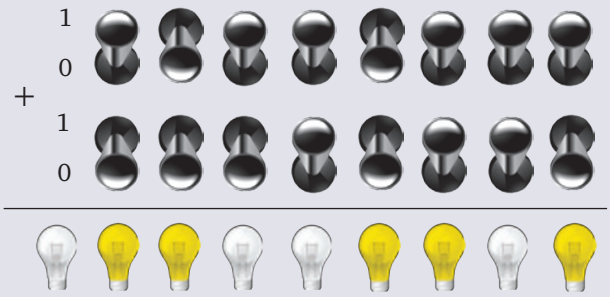


Рис. 2

На *рис.* 2 переключатели установлены в положения, соответствующие сложению чисел 183 (10110111) и 22 (00010110). Результат сложения, о чем свидетельствуют лампочки, равен 205 (11001101).

Вид пульта управления для сложения или вычитания двух 8-битовых чисел слегка иной. На нем появился дополнительный переключатель, с помощью которого можно задать действие — вычитание или сложение (см. *рис.* 3).

Рис. 3

Если этот переключатель разомкнут, производится сложение, если замкнут — вычитание. Кроме того, теперь для отображения результата используются только правые 8 лампочек. Девятая лампочка подписана “Переполнение/Исчезновение”. Она сигнализирует, что вычисленное число не может быть представлено только 8 лампочками. Это происходит, если сумма больше 255 (переполнение разрядов) или если разность оказалась отрицательной (исчезновение разрядов).

Основным новшеством в нашем сумматоре станет схема, которая вычисляет дополнение 8-разрядного числа до единицы. Как вы помните, это эквивалентно инвертированию битов, так что для вычисления дополнения 8-разрядного числа до единицы можно использовать просто 8 инверторов:

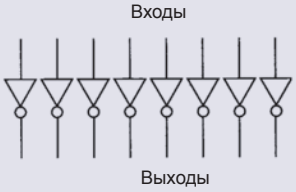


Рис. 4

Но эта схема *всегда* инвертирует биты, поступающие на ее входы, в нашей машине для сложения и

вычитания нам нужно устройство, которое инвертировало бы биты, только если производится вычитание. Более подходящая схема выглядит так:

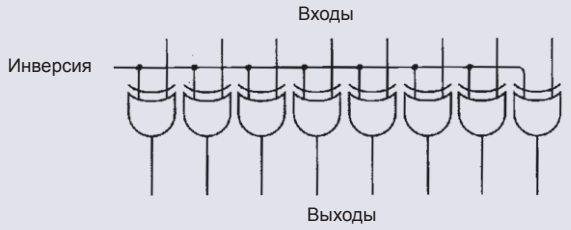


Рис. 5

Сигнал `Инверсия` подается на входы всех девяти вентилей “Искл-ИЛИ”. Напомним, что вентиль “Искл-ИЛИ” работает так:

Искл-ИЛИ	**0**	**1**
0	0	1
1	1	0

Если сигнал `Инверсия` равен 0, восемь выходов вентилей “Искл-ИЛИ” повторяют сигнал на восьми их входах. Например, если на вход подается число 01100001, на выходе также будет 01100001. Если же сигнал `Инверсия` равен 1, восемь входных сигналов будут инвертироваться (если на входе 01100001, на выходе имеем 10011110).

Давайте поместим восемь вентилей “Искл-ИЛИ” в общий прямоугольник под названием “Дополнение до единицы”:

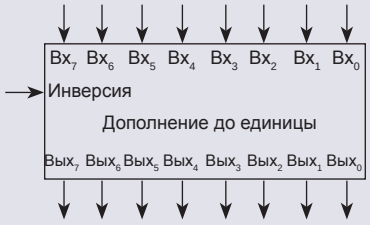


Рис. 6

Теперь вентиль “Дополнение до единицы”, 8-битовый сумматор и выходной вентиль “Искл-ИЛИ” можно собрать в единую схему:

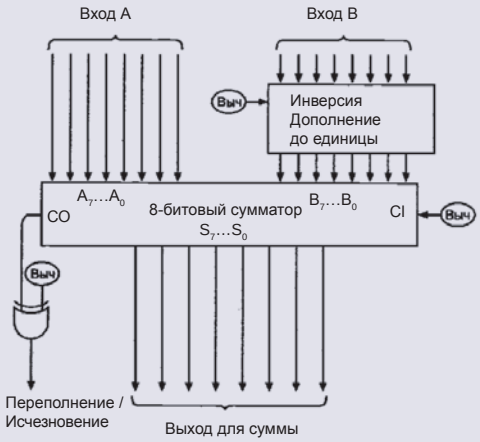


Рис. 7

Обратите внимание на сигнал `Выч`, идущий с переключателя “Сложение/Вычитание”. Этот сигнал равен 0, если нужно выполнить сложение, и 1, если нужно выполнить вычитание. При вычитании входы B (второй ряд переключателей) перед сумматором инвертируются схемой “Дополнение до единицы”, а к результату сумматора прибавляется 1, для чего его вход `CI` установлен в 1. При сложении схема “Дополнение до единицы” не делает ничего; сигнал на входе `CI` равен 0.

Сигнал `Выч` и выход `CO` (выход для переноса) сумматора также поступают в вентиль “Искл-ИЛИ”, который управляет включением лампочки “Переполнение/Исчезновение”. При равенстве 0 сигнала `Выч` (выполняется сложение) лампочка горит, если сигнал `CO` сумматора равен 1, т.е. результат сложения оказался больше 255.

Пусть вас не смущает, что выход `CO` сумматора равен 1, если выполняется вычитание и вычитаемое (переключатели B) меньше уменьшаемого (переключатели A). Так проявляется число 100000000, которое в этом случае должно вычитаться на заключительном шаге. Лампочка “Переполнение/Исчезновение” горит, только если выход `CO` сумматора равен 0. Это означает, что вычитаемое больше уменьшаемого и результат отрицателен. Отображение отрицательных чисел в нашем устройстве не предусмотрено.

Мы в этой статье уже неоднократно говорили об отрицательных числах, но все еще не показали, на что они похожи в двоичном представлении. Можно, конечно, обозначать отрицательные двоичные числа знаком “минус”, как и в десятичной системе, скажем, записать число –77 как –1001101. Но одна из целей введения двоичных чисел в том, чтобы представлять в виде нулей и единиц *все что угодно*, в том числе и знак “минус” перед отрицательным числом.

Признаком знака может служить и дополнительный бит, который, например, равнялся бы 1 у отрицательных чисел и 0 у положительных. Однако есть другой способ представления отрицательных чисел, который позволяет без особых хлопот складывать отрицательные и положительные числа. Недостаток его лишь в том, что вы должны заранее решить, сколько цифр понадобится для представления чисел, с которыми вам предстоит иметь дело.

Давайте немного подумаем. Преимущество стандартного способа записи положительных и отрицательных чисел в том, что он не накладывает никаких ограничений на их величины. Как отрицательные, так и положительные числа расходятся от 0 до бесконечности.

... –1000000 –999999 ...

... –3 –2 –1 0 1 2 3 ...

... 999999 1000000 ..,

Но допустим, что бесконечный числовой ряд нам не нужен, так как мы заранее знаем, что все числа, с которыми мы столкнемся, заключены в определенных пределах.

Рассмотрим в качестве примера условный банковский счет, где нам иногда приходится на практике сталкиваться с отрицательными числами. Предположим, что мы никогда не держали на счете больше 499 рублей и что банк при пользовании счетом разрешает кратковременный перерасход также на 500 рублей (“в долг”). Это значит, что баланс нашего счета всегда заключен между 499 рублями и –500 рублями. Допустим также, что мы никогда не вносим на счет больше 500 рублей, никогда не списываем со счета больше, чем 500 рублей, и, наконец, имеем дело только с рублями, не обращая внимания на копейки.

Этот набор условий означает, что в работе со счетом нам никогда не приходится иметь дело с числами, выходящими за пределы от –500 до 499. Всего в этом диапазоне 1000 целых чисел. Это значит, что для представления всех нужных чисел мы можем использовать три десятичные цифры и не прибегать к знаку “минус”. Хитрость в том, что нам не нужны положительные числа из диапазона от 500 до 999, так как по нашим условиям максимальное положительное число, в котором мы нуждаемся, — 499. Значит, с помощью трехзначных чисел из диапазона от 500 до 999 можно представлять отрицательные числа. Вот как это сделать:

Вместо –500 используем 500
Вместо –499 используем 501
Вместо –498 используем 502
…
Вместо –2 используем 998
Вместо –1 используем 999
Вместо 0 используем 000
Вместо 1 используем 001
Вместо 2 используем 002
…
Вместо 497 используем 497
Вместо 498 используем 498
Вместо 499 используем 499

Иначе говоря, все трехзначные числа, начинающиеся с 5, 6, 7, 8 и 9, представляют отрицательные значения. Вместо записи чисел с минусами:
–500 –499 –498 … –4 –3 –2 –1 0 1 2 3 4 … 497 498 499
— мы записываем их так:
500 501 502 … 996 997 998 999 000 001 002 003 004 … 497 498 499

Заметьте: числа идут по кругу. Наименьшее отрицательное число (500) выглядит как продолжение наибольшего положительного (499). Число 999 (представляющее –1) на единицу меньше 0. Если прибавить к 999 единицу, мы получим 1000, но поскольку мы имеем дело только с тремя цифрами, в действительности получаем 000.

Такой способ обозначения отрицательных чисел называется *дополнением до десяти*. Чтобы преобразовать трехзначное отрицательное число в дополнение до десяти, вычитаем его из 999 и добавляем 1. Иными словами, дополнение до десяти — это дополнение до девяти плюс 1. Например, чтобы переделать –255 в дополнение до десяти, вычтем его из 999, получив 744, и прибавим 1, что дает 745.

Вам, вероятно, приходилось слышать, что “вычитание — это просто сложение с отрицательным числом”. На это вы скорее всего отвечали: “Да, но при этом оно остается вычитанием”. Дополнение до десяти позволяет избавиться от вычитания, *полностью заменив его сложением*.

Пусть на вашем счете 143 рубля. Вы потратили 78 рублей. Это означает, что к 143 нужно прибавить –78. Дополнение –78 до десяти равно 999 – 078 + 1, или 922. Таким образом, ваш баланс составляет 143 + 922, что равно 65 рублей (без учета переполнения). Если вы после этого потратите 150 рублей, то должны будете прибавить к балансу 1 число –150, дополнение которого до десяти равно 850. Сумма предыдущего баланса 065 и 850 равна 915. В обычном представлении число 915 эквивалентно –85 рублям — вашему новому балансу.

Аналогичная система в двоичном счислении называется *дополнением до двух*. Предположим, что мы работаем только с 8-битовыми числами, заключенными в пределах от 00000000 до 11111111, или от 0 до 255 в десятичной системе. Чтобы отображать с их помощью также и отрицательные числа, придется отдать в отрицательную область все 8-битовые числа, начинающиеся с 1, как показано в таблице:

Двоичное значение	Десятичное значение
10000000	–128
10000001	–127
10000010	–126
…	
11111101	–3
11111110	–2
11111111	–1
00000000	0
00000001	1
00000002	2
…	
01111110	126
01111111	127

Диапазон чисел, которые вы теперь можете представить, ограничен пределами от –128 до +127. Старший значащий бит (крайний слева) называется *знаковым разрядом*. Знаковый разряд равен 1 для отрицательных чисел и 0 для положительных.

Чтобы вычислить дополнение до двух, нужно посчитать дополнение до единицы и прибавить 1 или, что эквивалентно, инвертировать все цифры и прибавить 1. Например, десятичное число 125 в двоичной системе выглядит как 01111101. Чтобы представить –125 в виде дополнения до двух, инвертируем цифры в числе 01111101, получая в результате 10000010, а затем прибавляем единицу, что дает 10000011. Проверьте результат по табли-

це. Чтобы проделать обратную операцию, нужно сделать то же самое — инвертировать все биты и прибавить 1.

Эта система позволяет выражать положительные и отрицательные числа без знака “минус”, а также складывать положительные и отрицательные числа, используя только правила сложения. Давайте в качестве примера сложим двоичные эквиваленты –127 и 124. Используя предыдущую таблицу как шпаргалку, запишем:

+	10000001
	01111100
	11111101

Этот результат эквивалентен десятичному числу –3 (то есть –127 +124 = –3).

При этом необходимо следить за переполнением или исчезновением разрядов. Такое случится, если результат сложения окажется больше 127 или меньше –128. Допустим, мы хотим сложить число 125 с ним самим.

+	01111101
	01111101
	11111010

Старший бит равен 1, т.е. результат интерпретируется как отрицательное число и становится равным –6.

То же самое происходит, если с самим собой вы сложите число –125.

+	10000011
	10000011
	100000110

Мы условились в начале, что ограничимся 8-битовыми числами, поэтому крайнюю левую цифру результата приходится игнорировать. Правые же 8 битов эквивалентны +6.

В заключение напомним, что описанная идея расчетов используется для получения так называемого “дополнительного k-разрядного кода отрицательного числа”. Алгоритм его получения [2]:

1) модуль отрицательного числа представить прямым кодом в k двоичных разрядах;

2) получить его обратный код;

3) к полученному обратному коду, рассматриваемому как k-разрядное неотрицательное двоичное число, прибавить единицу.

Задания для самостоятельной работы

1. Подумайте над вопросом: “Можно ли только по виду двоичного числа определить его десятичный эквивалент?” Например, кто-то скажет вам: “У меня есть 8-битовое двоичное число, равное 110110. Каков его десятичный эквивалент?” Что ответите вы?

2. Определите, каким десятичным числам соответствуют:

1) 8-битовые числа без знака;

2) 8-битовые числа со знаком.

3. Подумайте, как по знаковым разрядам двух слагаемых (положительных и/или отрицательных чисел) и знаковому разряду результата определить правильность результата сложения.

4. Ответьте на вопрос: “Для чего используется дополнительный код отрицательного числа?”

Ответы присылайте в редакцию (можно отвечать не на все вопросы).

Литература

1. Создаем сами двоичный сумматор. / “В мир информатики” № 171 (“Информатика” № 17/2011).

2. *Андреева Е.В.*, *Босова Л.Л.*, *Фалина И.Н.* Математические основы информатики. М.: Бином. Лаборатория знаний, 2005.

3. *Петцольд Ч.* Код. М.: Издательско-торговый дом “Русская редакция”, 2001.

ТВОРЧЕСТВО НАШИХ ЧИТАТЕЛЕЙ

Ребусы, посвященные Году космонавтики. Часть 4

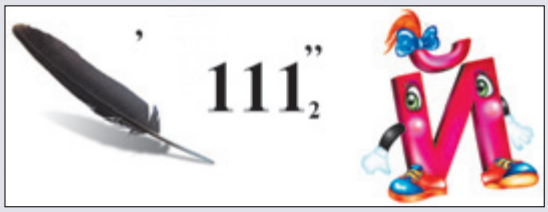
Мы завершаем публикацию ребусов, посвященных Году космонавтики (таким был 2011 год), которые разработали Анатолий и Ульяна Тимофеевы, ученики Именевской основной школы, Красноармейский р-н Чувашской Республики (учитель **Тимофеева И.А.**).

Ребус № 1

Ребус № 2

Ребус № 3

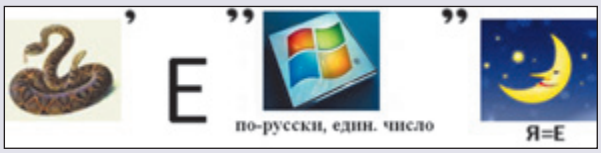
Ребус № 4



Ребус № 5

Ребус № 6

Ребус № 7



Ребус № 8

Ребус № 9

Ответы присылайте в редакцию (можно решать не все ребусы). Подумайте также, с какой темой, относящейся к космонавтике, связаны ребусы.

Спасибо авторам и всем приславшим ответы!

“ЛОМАЕМ” ГОЛОВУ

Числовой ребус с неизвестным числом

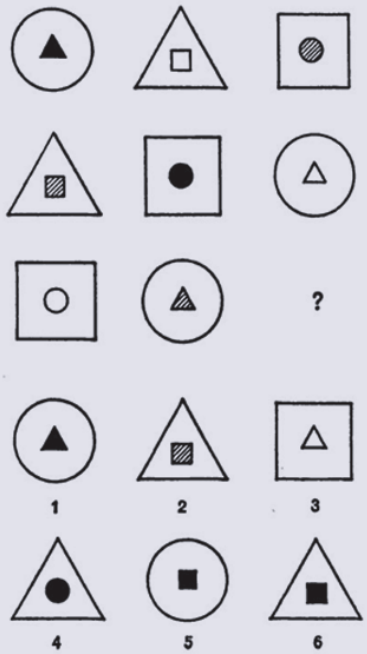
Рассмотрим числовой ребус:

	A	B	C	D
+	D	C	B	A
	X	X	X	X
1	2	3	0	0

В нем **A**, **B**, **C**, **D** — четыре последовательные цифры, идущие в порядке возрастания. Четыре буквы **X** — это те же самые четыре цифры в неизвестном порядке. Какое число обозначено четырьмя буквами **X**? Каково все решение ребуса? Можно ли решить ребус, если система счисления, в которой записаны числа, неизвестна?

Выбрать фигуру

Из шести пронумерованных фигур выберите ту, которая должна стоять на месте вопросительного знака:



Как перейти реку?[3]

Река шириной 4 метра делает поворот под прямым углом (см. рисунок). Как переправиться через нее на другой берег, имея лишь две доски длиной 3 метра 90 сантиметров?

Кто лишний?

В приведенной ниже группе все названия животных, за исключением одного, выбраны по какому-то определенному признаку: слон, лошадь, ехидна, лось, тигр, лев.

Установите (используя информацию в Интернете или другие источники) объединяющий признак и укажите, что ему не соответствует.

Как разделить молоко?

Как за наименьшее число переливаний с помощью пустых трехлитрового и семилитрового бидонов разлить пополам 10-литровый бидон с молоком?

Алгоритм решения задачи, пожалуйста, оформите в виде:

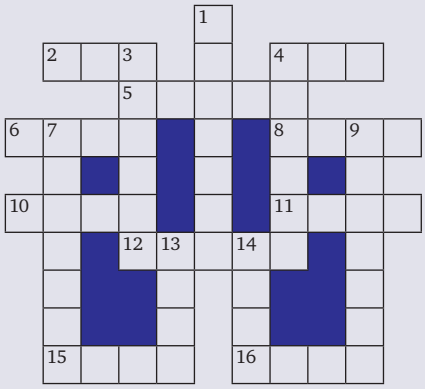
		Бидон 10 л	**Бидон 7 л**	**Бидон 3 л**
	Исходное состояние	10	0	0
1	Из 10 в 7	3	7	0
2	…			

Активные ученики ☺

Однажды в одной из стран инспектор учебного управления, проверявший одну из школ, задал ученикам класса 10 вопросов. Во всех случаях в ответ поднимали руку *все* ученики. И хотя школьный учитель каждый раз выбирал разных учеников, ответ всегда был правильным. Как это получалось? (Конечно, так, чтобы *все* ученики знали ответы на *все* вопросы, бывает очень редко.)

Кроссворд

Решите, пожалуйста, кроссворд.



По горизонтали

2. В программировании — одна из характеристик переменной величины.

4. Денежная единица, в которой получают зарплату болгарские программисты.

5. Величина, с помощью которой осуществляется счет.

6. Компонент данных типа *запись*.

8. Конечное число точек на плоскости, соединенных отрезками кривых линий.

10. Структура данных, в которых применен принцип “последним пришел — первым вышел”.

11. Семейство совместимых друг с другом компьютеров.

12. Буква греческого алфавита.

15. Название фигуры в настольной игре, для которой имеются компьютерные варианты.

16. Часть рабочей книги программы Microsoft Excel.

По вертикали

1. Язык программирования.

3. Элемент электронной таблицы.

4. Наука о законах и формах мышления.

7. Результат целочисленного деления.

9. Пользователь телефонной линии.

13. Константа логического типа.

14. Поименованная совокупность данных на носителе.

ДЛЯ ЭРУДИТОВ

Выберите правильный вариант

Мы предлагаем читателям прочитать начало занимательного факта и выбрать один из двух вариантов вероятного окончания. Но не наугад, а найдя соответствующую информацию в Интернете.

1. Хорошо натренированный бегун, рванув со старта, в первые десять метров способен опередить…

А. … арабского скакуна;

Б. …гоночную машину.

2. На каждый доллар, что американские автолюбители тратят на бензин, только на налоги приходится…

А. …70 центов;

Б. …27 центов.

3. Голливудская актриса Николь Кидман, играя волевых и бесстрашных дам, на самом деле панически боится…

А. …электричества;

Б. …бабочек.

[3] Задание предназначено для учащихся начальной школы и учеников 5–7-х классов.

4. Даже самая маленькая капля алкоголя, помещенная на скорпиона, практически сводит его с ума. Он начинает…

А. …жалить себя;

Б. …кувыркаться.

5. Среди золотых рыбок встречаются и “реальные долгожители”. Самая старая из них по имени Фреда прожила…

А. …3 года;

Б. …41 год.

6. Несмотря на отсутствие Интернета в 30–40-х годах прошлого века, США и Великобритания были буквально заполнены “спамом”. Но не надоедливой интернет-рекламой, а…

А. …перченой колбасой;

Б. …конфетами.

7. В странах Индокитая есть предупреждающие знаки на дверях отелей с перечеркнутым фруктом. Он очень популярен в тех местах, но имеет отвратительный запах и называется:

А. …дуриан;

Б. …маракуйя.

Ответы (можно не все) присылайте в редакцию.

ВНИМАНИЕ! КОНКУРС

Конкурс № 91 “Удалить и заменить букву”

Тур 2

По заданным словам, удалив в них одну букву и заменив другую, необходимо получить термин (и фамилию ученого), связанный с информатикой и ИКТ.

Кунак.	Кабала.
Шланг.	Байкер.
Агат.	Динар.
Банту.	Тостер.
Венера.	Работа.
Сайра.	Треть.
Череда.	Артек.
Вода.	Плитка.
Ртуть.	CALGON.
Дуплет.	Банан.
Флокс.	Беда.
Дамка.	Удача.
Тиски.	Сторожка.
Поступь.	Записка.
Контроль.	Дева.

Приведите также комментарии к найденным словам.

Ответы отправьте в редакцию до 10 февраля по адресу: 121165, Москва, ул. Киевская, д. 24, “Первое сентября”, “Информатика” или по электронной почте: *vmi@1september.ru*. Пожалуйста, четко укажите в ответе свои фамилию и имя, населенный пункт, номер и адрес школы, фамилию, имя и отчество учителя информатики.

В ответах сохраните нумерацию слов в задании (для ненайденных слов поставьте прочерк).

Итоги конкурса будут подводиться с учетом двух туров в целом.

Итоги конкурса № 86

Напомним, что необходимо было решить следующую задачу из старинной русской рукописи конца XVI — начала XVII века:

Смотри сице: $\overline{\text{г}}$ пуд воску, цена ему $\overline{\text{аі}}$ рублев $\overline{\text{б}}$ алтына с полуденгой. Что да(с)т $\overline{\text{сі}}$ пудов?

Иными словами, нужно было определить, сколько надо уплатить за указанное количество пудов, учитывая, что все числа в задаче записаны в так называемой “славянской алфавитной системе”, или “буквенной цифири”, — способе записи чисел в Древней Руси с помощью букв применявшегося тогда алфавита.

Ответы представили:

— Андрющенко Александр и Свистунов Николай, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Базылев Юрий и Галушкова Карина, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Богданова Л.М.**;

— Дукач Светлана и Клименко Надежда, г. Лесосибирск Красноярского края, поселок Стрелка, школа № 8 им. Константина Филиппова, учитель **Лопатин М.А.**;

— Кондратов Богдан, г. Ярославль, школа № 33, учитель **Цикина Е.Н.**;

— Семенова Виктория, г. Саратов, школа № 52, учитель **Пуйшо Н.В.**;

— Слипенчук Александр, Москва, гимназия № 1530, учитель **Козырева О.В.**;

— Тощев Андрей, г. Ярославль, школа № 33, учитель **Ярцева О.В.**

Решение

Прежде всего все денежные единицы в стоимости 16 пудов воска переведем в самые мелкие, в данном случае — в полушки (полуденги):

— 15 рублей = 6000 полушек;

— 2 алтына = 24 полушки;

и еще 1 полушка — итого 6025 полушек. Тогда стоимость 9 пудов составит:

$\frac{6025 * 16}{9}$ = 10 711 и 1/9 полушки, или 26 рублей 25 алтын 5 денег одна и 1/9 полушки.

Абсолютным победителем конкурса признаны Александр Слипенчук из московской гимназии № 1530 и Андрей Тощев из школы № 33 г. Ярославля, получившие указанный ответ. Остальные участники конкурса также будут награждены дипломами.

ОБРАЗОВАТЕЛЬНОЕ ПРОСТРАНСТВО
1september.ru
5 000 000 посетителей в месяц

- ВСЕРОССИЙСКАЯ ОБЩЕПЕДАГОГИЧЕСКАЯ ГАЗЕТА
- ЕЖЕМЕСЯЧНЫЕ ПРЕДМЕТНО-МЕТОДИЧЕСКИЕ ЖУРНАЛЫ С ЭЛЕКТРОННЫМИ ВЛОЖЕНИЯМИ

ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ «ПЕРВОЕ СЕНТЯБРЯ»

- Повышение квалификации учителей: дистанционное и очное обучение
- В 2011/12 учебном году 170 курсов

ФЕСТИВАЛЬ ПЕДАГОГИЧЕСКИХ ИДЕЙ «ОТКРЫТЫЙ УРОК»

- Крупнейшая в России заочная конференция учителей
- Размещение работ в Интернете и на DVD с публикацией сборников тезисов
- Оформление сертификатов, подтверждающих авторство педагогических идей
- В 2011/12 учебном году 25 000 участников

ФЕСТИВАЛЬ ИССЛЕДОВАТЕЛЬСКИХ И ТВОРЧЕСКИХ РАБОТ УЧАЩИХСЯ

- Заочная конференция учеников
- В 2011/12 учебном году 12 000 участников

ЕЖЕГОДНЫЙ МОСКОВСКИЙ ПЕДАГОГИЧЕСКИЙ МАРАФОН УЧЕБНЫХ ПРЕДМЕТОВ

- Самая большая учительская конференция России
- 22 тематических дня, 250 мероприятий, 300 лекторов и ведущих
- 15 000 участников

ФЕСТИВАЛЬ «УЧИТЕЛЬСКАЯ КНИГА»

- Самая большая ярмарка учебно-методической литературы в стране
- 4 дня, более 70 ведущих издательств, более 1500 наименований каждый год

ЕЖЕДНЕВНАЯ ПЕДАГОГИЧЕСКАЯ НОВОСТНАЯ ЛЕНТА

news.1september.ru

ул. Киевская, д. 24, Москва, 121165
E-mail: *info@1september.ru*
Интернет: *www.1september.ru*
Интернет-магазин: *shop.1september.ru*

секретариат: *(499) 249-31-38*
отдел подписки: *(499) 249-47-58*
факс: *(499) 249-31-38*

2012