№13 2011 Издание основано в 1995 г inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА

Простые и совершенные, ведущие в бесконечность

с. 4

Объектно-ориентированное програмирование

с. 22

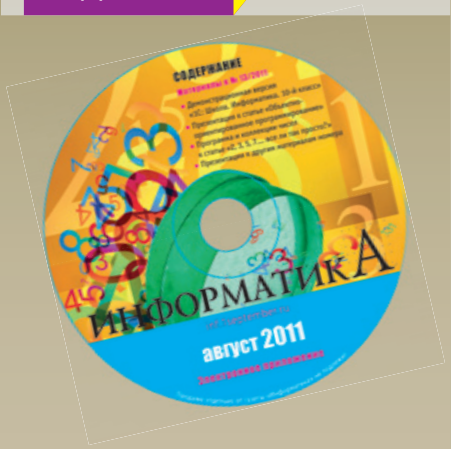
издательский дом Первое сентября
1september.ru ИНФОРМАТИКА Индексы подписки "Роспечати" – 32291 (инд.); 32591 (организации); 19179 (электронная версия)
август 2011

О НОМЕРЕ

Знаете известную поговорку: лечить и учить умеют все? То же можно сказать и о некоторых темах курса информатики. Все знают про простые и всякие другие замечательные числа, все понимают про объектно-ориентированное программирование и шифр Цезаря. Только на поверку часто оказывается, что все глубже, чем кажется на первый взгляд и чем написано в учебнике. Большая часть статей этого номера посвящена достаточно сложным темам. При этом с точки зрения глубины изложения материала тут заведомо с избытком, а уже каждому из нас решать, в каком объеме и как использовать его в работе с детьми.

В НОМЕРЕ

НА ДИСКЕ



ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ:

- Демонстрационная версия “1С:Школа. Информатика, 10 класс”
- Презентация к статье “Объектно-ориентированное программирование”
- Программа и коллекции чисел к статье “Простые и совершенные, ведущие в бесконечность”
- Презентации к другим материалам номера

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ: по каталогу “Газеты. Журналы” агентства “Роспечать” – 32291 (инд); – 32591 (орг.); 19179 (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
Газета распространяется по подписке
Цена свободная
Тираж 3000 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 и продвижение: Марк Сартан
Мультимедиа, конференции и техническое обеспечение: 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 – Я.Сартан
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 15.06.2011,
фактически 15.06.2011
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

“Ха! Хаскель!” и “Теория относительности”

Материалы номера получились сложнее, чем мне бы хотелось. Они интересные (очень!), разнообразные (очень!), но предназначены преимущественно для работы со старшими школьниками. С композиционной точки зрения это минус, и я это хорошо понимаю. Но все равно мне номер очень нравится — ведь материалы отличные, рабочие, их “хватит надолго” — они наверняка много раз пригодятся вам в работе.

Материалы номера получились сложнее, чем мне бы хотелось. Они интересные (очень!), разнообразные (очень!), но предназначены преимущественно для работы со старшими школьниками. С композиционной точки зрения это минус, и я это хорошо понимаю. Но все равно мне номер очень нравится — ведь материалы отличные, рабочие, их “хватит надолго” — они наверняка много раз пригодятся вам в работе.

Вместе с тем и выводы сделаны. Содержательно следующий номер будет более “разноцветным”. И именно о следующем номере мне и хочется рассказать. Рассказывать буду не по порядку следования статей и, конечно, не по порядку их значимости (мне этот порядок вообще не известен, мне все статьи нравятся).

Итак, “Ха! Хаскель!”. Кто знает, что такое Хаскель? Отвечу за себя. С одной стороны, знаю, с другой — нет. Замечательный ответ ☺. Я много занимался функциональными языками и люблю этот мир программирования. Но даже я очень круглыми глазами всегда смотрел на программы на Хаскеле.

```
hanoi' 0 _ _ _ = []
hanoi' n a b c = hanoi' (n-1)
       a c b ++ [(a, b)] ++
       hanoi' (n-1) c b a
hanoi n = hanoi' n 1 2 3
```

Ну, невероятно, чтобы *это* могло иметь смысл и работать! Оказалось же, что о Хаскеле можно написать коротко, просто и доходчиво. Сделал это наш коллега из замечательного города программистов Переславля. Необходимое уточнение: “Ха! Хаскель” — это название статьи, Хаскель — название языка (Haskell), слово “Ха!” употребляется в данном контексте матерыми программистами, которые еще и не то видели.

Еще одна замечательная статья — “Устройства ввода и вывода. Теория относительности”. Статья очень интересная и даже несколько провокационная — наверняка не все будут согласны с позицией автора. Ключевой вопрос — а что же такое устройства ввода-вывода?

“Так что же есть сканер? Для компьютера — это устройство ввода.

А для человека? А для человека сканер — устройство вывода. Мы ведь отдаем ему листок, “выводя” из папки, в которой листок хранили”.

И как это объяснить детям, а перед этим — самому себе? По странному стечению обстоятельств автор тоже из Переславля. Так сложилось ☺.

“В мир информатики” в следующем номере тоже замечательный. Приводить названия статей — одно удовольствие: “От кошек — через переключатели — к компьютерам”, “Штирлиц и IP-адрес”, “Индюки и жеребята”, “Из Ёлкино в Палкино”.

Кстати, дорогие коллеги, подписчики бумажной версии журнала. Пожалуйста, обязательно обратите внимание на информацию на последней странице обложки и на то волшебное число, которое там напечатано. Воспользовавшись этим числом, вы сможете получить сентябрьский номер (электронную версию) уже 1 сентября. Подробности описывать не буду, там все написано.

До встречи!

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

Простые и совершенные, ведущие в бесконечность

В.Е. Гончаренко,
г. Иваново

В статье рассматриваются различные алгоритмические задачи, связанные с “особенными” натуральными числами — простыми числами, совершенными числами и числами Мерсенна.

Натуральные числа являются подмножеством целых чисел. Натуральные числа (естественные числа) — числа, возникающие естественным образом при счете, например, при перечислении предметов: первый, второй, третий и т.д. Натуральные числа большие единицы подразделяются на простые и составные. Единица, будучи началом всех целых чисел, не является ни простым, ни составным числом (Л.Эйлер). Простое число имеет ровно два натуральных делителя — единицу и собственное значение, соответственно составные числа имеют более двух натуральных делителей. В соответствии с основной теоремой арифметики любое составное число можно представить в виде произведения простых чисел. Следовательно, для определения делимости натурального числа N достаточно использовать только простые числа.

Из истории чисел

История развития человеческой цивилизации неразрывно связана с историей изобретения чисел и развитием математики в целом. Значимость чисел красноречиво отражает одно из многочисленных мудрых высказываний древнегреческого мыслителя Пифагора Самосского (≈570 – ≈500 до н. э.): “Все вещи суть числа”. Слова, сказанные задолго до нашей эры, становятся только актуальней в XXI веке нашей эры. Оцифрованный звук и изображение, виртуальная реальность и многочисленные иные примеры современных IT-технологий — все суть числа, причем с использованием всего-навсего двух цифр (кодов) — нуля и единицы.

На изобретение и понимание этих двух удивительно плодотворных чисел — нуля и единицы — ушли века. Естественным образом сначала формировалось представление о числе 1 и других натуральных числах.

Сначала счет неразрывно связывался с перечисляемыми предметами — одно солнце, три дерева, два охотника. Со временем развитие абстрактного мышления позволило выделить в отдельные понятия числа, и единица стала прародительницей ряда натуральных чисел. Каждое следующее натуральное число получается из предыдущего увеличением его на 1.

Современное математическое определение числа 1 сводится к следующему: единица — это единственное положительное число, которое равно своему обратному. Напомним, что обратное число — это число, на которое надо умножить, чтобы получить единицу, а пара чисел, произведение которых равно единице, называются взаимно обратными, например, 5 и 1/5 или 3,75 и 1/3,75. За определение единицы также принимают утверждение, что при умножении единицы на любое другое число в результате получается это же число.

Понятие нуля как числа потребовало большего времени развития абстрактного мышления и потребности в сложных математических вычислениях. Вавилонские математики использовали особый клинописный значок для шестидесятеричного нуля примерно начиная с 300 г. до н. э., а их учителя — шумеры, вероятно, сделали это еще раньше. Но их изобретение не получило широкого распространения, потому что используемый ими математический аппарат базировался не на десятичной, а на шестидесятеричной системе счисления. Зато из их математики мы взяли принцип учета времени — 60 минут по 60 секунд составляют 1 час. Только в VI веке нашей эры в Индии также изобрели число “ноль”, после чего разработали позиционную систему счисления. Она была перенята арабами, которые называли цифры “индийскими знаками”. В период до X века их отображение немного изменилось, перейдя к привычным для нас цифрам 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Европа же получила эти цифры уже от арабов, и мы, пользуясь десятичной системой счисления, называем эти цифры арабскими. Но не все сразу приживалось. В Европе долгое время ноль считался условным символом и не признавался числом даже в XVII веке. В арифметических трудах отрицательное число истолковывалось как долг, а ноль — как ситуация полного разорения. Полному уравнению нуля в правах на понятия числа способствовали труды Леонарда Эйлера (1707–1783). Приятно сознавать, что такой великий математик долгие годы жил и плодотворно работал в России. Лаплас назвал Эйлера общим учителем всех математиков второй половины XVIII века. К этому надо добавить, что Эйлер явился идейным предшественником многих математиков XIX и XX вв.

Современное определение нуля сводится к тому, что ноль — целое число, разделяющее на числовой оси положительные и отрицательные числа. Основные свойства нуля: любое число при сложении с нулем не меняется; умножение любого числа на ноль дает ноль; ноль не имеет знака; ноль является четным числом и делится на все натуральные числа; деление на ноль невозможно, следовательно, для нуля не существует обратного числа (в пространстве комплексных чисел).

Всего двух чисел (цифр) — 1 и 0 с их удивительно глубинными свойствами достаточно для отображения в двоичной системе счисления любых других чисел.

Изучение чисел — это предмет древней и интересной науки “Теория чисел”, в развитие которой внесли вклад многие знаменитые ученые. Прочные основы этой теории были заложены еще в III веке до н. э. легендарным греческим математиком Евклидом и изложены в его книгах “Начала” как итог 300-летнего развития античной математики. С именем другого великого греческого математика — Эратосфена (276–194 гг. до н. э.) связано решение задачи определения всех простых чисел в ряду натуральных чисел от 2 до N. И по сей день идеи и результаты трудов великих греческих математиков востребованны. Основная теорема арифметики, определение НОД по алгоритму Евклида, решето Эратосфена и многое другое постоянно упоминается в учебных программах по математике, алгоритмизации и программированию.

Евклидом было выполнено доказательство бесконечности ряда простых чисел, а основная теорема арифметики делает простые числа незаменимыми в определении делимости чисел. Согласно теореме, каждое натуральное число $N > 1$ можно представить в виде $N = p_1 \cdot \ldots \cdot p_k$, где $p_1, \ldots, p_k$ — простые числа, причем такое представление единственно с точностью до порядка следования сомножителей. Такое представление числа N называется его каноническим разложением на простые сомножители, или факторизацией числа, например, $588\,000 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 \cdot 5 \cdot 5 \cdot 5 \cdot 7 \cdot 7 = 2^5 \cdot 3 \cdot 5^3 \cdot 7^2$.

Неоценимый вклад в нахождение простых чисел внес Эратосфен, предложив простой и очень эффективный алгоритм их определения от 2 до N, получивший название “решето Эратосфена”. Математиков буквально пленила идея собрать самую большую коллекцию простых чисел, которые в виде таблиц простых чисел издавались тысячестраничными томами. Титаническую работу проделал в XIX в. профессор чешского университета в Праге Якуб Филипп Кулик (1793–1863), составив самую большую на то время таблицу, которая содержала простые числа до 100 330 201. Читателям будет предложена программа, которая позволит создать за короткое время свою собственную коллекцию простых чисел, превышающую этот результат XIX в.

Составные числа

Среди натуральных чисел подавляющая их часть приходится на составные числа. Для каждого натурального числа априори известны два его целочисленных делителя — это единица и собственное значение. Отсюда сразу можно сформулировать следующую задачу: сколько и какие еще у числа имеются делители? На этот вопрос нет прямого ответа, необходимо рассмотреть весь ряд потенциальных делителей, начиная с двойки и заканчивая целой половиной числа. Например, для числа 12 ряд потенциальных делителей 2, 3, 4, 5, 6. Почему не имеет смысл рассматривать делители 7, 8, 9, 10, 11? Самое большое значение делителя — это собственное значение числа, для нашего примера это 12, а частное от деления равно 1. Следующее наименьшее значение частного от деления может быть 2, которое получается при делении на целую половину от числа, для нашего примера это 6. Поскольку между единицей и двойкой нет других целых чисел, то ни одно из значений более половины числа не может быть его целочисленным делителем. Для числа 12 можно практически не задумываясь ответить, что у него есть еще четыре делителя 2, 3, 4 и 6, кроме 1 и 12. Но если мы назовем произвольное, достаточно большое число, например, $N = 1\,673\,824$, то лучше поручить решение этой задачи верному и неутомимому помощнику — компьютеру. А для себя оставить создание четких и однозначных указаний ему в виде программы. Ниже представлен пример листинга программы на языке Pascal, будем считать, что синтаксис этого распространенного языка программирования достаточно легко читается.

Греческий философ и математик Никомах Герасский

Определение всех делителей составного числа N:

```
for i := 2 to (N div 2) do
    if  (N mod i = 0) then
    write (i, '  ');
```

Если следовать одному из девизов программирования — программа не должна делать ничего лишнего, — то в ограничении ряда потенциальных делителей можно предусмотреть наличие парных делителей числа. Например, число 12 делится на 3 с частным от деления 4, но и 4 является делителем числа 12. Реализуем эту идею в задаче определения совершенных чисел.

Совершенные числа

В теории чисел есть понятие совершенного числа, для которого сумма его целочисленных делителей, за исключением собственного значения, равна его значению, например, число 6 является совершенным, так как $6 = 1 + 2 + 3$, и является наименьшим совершенным числом. Поскольку ряд натуральных чисел бесконечен, то можно смело предположить о существовании следующих, больших по значению совершенных чисел. Знаменитый греческий философ и математик Никомах Герасский, живший в I в., отмечал, что совершенные числа красивы, а красивые вещи редки и немногочисленны. Он не знал, сколько имеется совершенных чисел. Не знаем этого и мы. До настоящего времени нет ответов на два важных вопроса:

1) Существует ли наибольшее четное совершенное число?

2) Существует ли нечетное совершенное число?

До сегодняшнего дня не обнаружено ни одного нечетного совершенного числа, хотя и не доказано, что такого числа не существует.

Первые четыре совершенных числа приведены в “Арифметике” Никомаха Герасского.

Числа $P_2 = 6$ и $P_3 = 28$ были известны еще пифагорийцам. Числа $P_5 = 496$ и $P_7 = 8128$ нашел Евклид. Пятое совершенное число 33 550 336 обнаружил немецкий математик Регионтан (XV век). В XVI веке немецкий ученый Шейбель нашел еще два совершенных числа: 8 589 869 056 и 137 438 691 328. Они соответствуют $p = 17$ и $p = 19$, где p — число в степени двойки формулы Евклида (см. с. 9). В начале XX века были найдены еще три совершенных числа (для $p = 89$, 107 и 127). В дальнейшем поиск затормозился вплоть до середины XX века, когда с появлением компьютеров стали возможными вычисления, превосходившие человеческие возможности.

Немецкий математик Регионтан

Сейчас известно 47 четных совершенных чисел, поиском новых таких чисел занимается проект распределенных вычислений GIMPS.

Определение совершенных чисел не из легких задач, но в наше время с неутомимым помощником — компьютером — все нипочем. Следующая программа решает поставленную задачу “в лоб”, проверяя

все числа исследуемого диапазона на равенство их значения сумме их делителей.

```
program  Program1;
var
  I,N,Summa: Longint ;
  Delitel: Longint;
begin
  for I := 3 to 34000000 do
  begin
    Summa := 1;
    for Delitel := 2 to Round (Sqrt(I)) do
    begin
      N := (I div Delitel);
      if  N * Delitel = I  then
        Summa := Summa + Delitel +
                 (I div Delitel);
    end;
    if Round(Sqrt(I)) * Round(Sqrt(I)) = I  then
      Summa := Summa - Round (Sqrt(I));
      if I = Summa then writeln(I);
  end ;
end.
```

Значение каждого очередного совершенного числа резко возрастает, поэтому в диапазоне значений стандартных типов данных языков программирования их небольшое количество. Значения первых десяти совершенных чисел следующие: 6, 28, 496, 8128, 33550336, 8589869056, 137438691328, 2305843008139952128, 26584559915698317446546926615953842176, 191561942608236107294793378084303638130997321548169216. Все остальные совершенные числа имеют очень много цифр в своей записи.

Оценив значение пятого совершенного числа, становится понятным значение верхней границы цикла `for` в представленной программе. Если число *I* является квадратом натурального числа `Delitel`, то его одинаковое парное значение вычитается из суммы делителей. Остается только пояснить, что функция `Sqrt(I)` извлекает корень квадратный из числа *I*, возвращая вещественное значение, а функция `Round()` преобразует его в целый тип, округляя по правилам математики. Достаточно быстро после запуска программы на выполнение на мониторе появятся первые четыре совершенных числа, а вот появление пятого числа придется подождать. Для его определения необходимо "перелопатить" десятки миллионов чисел в цикле `for`. Значение шестого совершенного числа уже выходит за пределы типа `Longint` языка Pascal. Для вычисления бóльших значений совершенных чисел необходимо использовать специальные методы их представления и обработки в вычислительной системе.

Крупным достижением теории совершенных чисел была теорема Евклида о том, что число $2^{n-1}(2^n - 1)$ — четное и совершенное, если число $2^n - 1$ — простое. Алгоритм построения четных совершенных чисел описан в IX книге "Начал" Евклида. Лишь две тысячи лет спустя Эйлер доказал, что формула Евклида содержит все четные совершенные числа. Поскольку неизвестно ни одного нечетного совершенного числа, то обычно, говоря о совершенных числах, имеют в виду четное совершенное число. Если в выражении использовать в качестве степени простое число *p*, то числа $M_p = 2^p - 1$ называют числами Мерсенна. О числах Мерсенна более подробно будет рассказано в разделе о простых числах.

С математической точки зрения четные совершенные числа по-своему уникальны. Все они треугольные. Треугольное число — это число кружков, которые могут быть расставлены в форме равностороннего треугольника. Сумма величин, обратных всем делителям числа, включая само число, всегда равна двум. Остаток от деления совершенного числа, кроме 6, на 9 равен 1. В двоичной системе совершенное число *Pp* начинается *p* единицами, потом следуют $p - 1$ нулей. Например:

$P_2 = 110, P_3 = 11100, P_5 = 111110000,$
$P_7 = 1111111000000$ и т.д.

Последняя цифра четного совершенного числа или 6, или 8, причем, если 8, то ей предшествует 2.

Наибольший общий делитель

До сих пор исследовались свойства отдельных чисел. Рассмотрим теперь весьма популярную задачу для двух натуральных чисел по определению их наибольшего общего делителя (НОД). Алгоритм Евклида нахождения НОД основан на следующих свойствах этой величины. Пусть *x* и *y* одновременно не равные нулю целые неотрицательные числа и пусть $x >= y$, тогда если $y = 0$, то НОД$(x, y) = x$, а если *y* не равен 0, то для чисел *x*, *y* и *r*, где *r* — остаток от деления *x* на *y*, выполняется равенство НОД(x, y) = НОД(y, r).

```
program Program2;
var
  x, y: integer;
begin
  writeln('Введите два числа');
  readln(x,y); {вводим два целых числа}
  repeat
    if x > y then x := x mod y
             else y := y mod x;
  until (x = 0) or (y = 0);
  {до тех пор, пока одно из чисел не станет равно нулю}
  writeln('НОД=', (x + y));
{вывод НОД - без условного оператора, так как одно из чисел обязательно равно нулю}
  readln;
end.
```

Простые числа

Вот уже более 2000 лет неизменный интерес вызывают вопросы, связанные с простыми числами. В наши дни только в поисковой системе Яндекс за

один месяц обрабатывается более 1 млн. 300 тысяч запросов по определению простых чисел. Впору говорить о магии простых чисел.

Помимо обычного человеческого любопытства к чему-либо необычному, простые числа играют важную роль в фундаментальном разделе математики — теории чисел.

В настоящее время большие простые числа используются в криптографии при шифровке данных. Простые числа и их распределение играют важную роль в алгебре, физике, теории информации, используются в тестировании предельных возможностей вычислительных машин.

Ученые состязаются в своеобразном соревновании по определению самого большого простого числа. Полученные результаты расцениваются как достижения в теории чисел, заносятся в Книгу рекордов Гиннесса, а авторы получают солидные денежные премии.

Очередное самое большое простое число определил канадец Майкл Камерон. В записи этого числа используется 4 миллиона 53 тысячи 946 знаков. На одно только написание этого простого числа уйдет три недели.

При определении самых больших простых чисел обычно рассматривают числа, имеющие форму $2^p - 1$, уже упоминавшиеся числа Мерсенна. Математики Калифорнийского университета Лос-Анджелеса определили 45-е по счету число Мерсенна, являющееся простым числом, — в его записи уже 12 миллионов 979 тысяч 189 знаков. Над его определением трудились 75 компьютеров, объединенных в сеть. Участники проекта получили денежную премию в размере 100 тысяч долларов. Возможно, что уже найдено очередное, еще большее число, ведь путь в бесконечность никогда не заканчивается.

Среди простых чисел есть и самые забавные — это простое число 73 939 133. Если в его записи последовательно удалять цифры справа, то оставшаяся запись, вплоть до 7, будет простым числом. Впрочем, это уже особенности записи простого числа, а не его величины. К особенностям записи простых чисел относится и понятие сверхпростого числа. Сверхпростые числа — это простые числа, которые при чтении их записи в обратном порядке также являются простыми, например, 13 и 31. Есть и другое определение сверхпростого числа.

Области исследований непосредственно простых чисел разнообразны, основными являются:

• определение принадлежности натурального числа X к простым числам;

• определение простых чисел в диапазоне натуральных чисел от 2 до N.

Помимо этого, существуют задачи определения простых чисел-близнецов, которые различаются на 2, например, (11, 13), (17, 19), (827, 829). Менее известной аналогичной задачей является определение простых чисел-триплетов. Триплеты — это тройка различных простых чисел, разность между наибольшим и наименьшим из которых минимальна, например, (5, 7, 11), (881, 883, 887).

Ученых в области теории чисел буквально завораживает задача, связанная с тайной распределения простых чисел. Они то собираются плотными стайками в ряду натуральных чисел, то располагаются в гордом одиночестве на большом расстоянии от своих собратьев. Премия в 1 млн. долларов ждет автора решения этой задачи.

Отношение количества простых и составных чисел в диапазоне до N можно определить по формуле $(N/\ln N)/(N - N/\ln N) \approx 1/\ln N$, а количество простых чисел в диапазоне N можно определить по формуле $N/\ln N$. В ряду натуральных чисел до 10 млн., согласно вычислителям, будет 620 421 простое число, что составляет 6,2% от всех чисел диапазона.

Нет простой операции проверки неделимости числа. Для решения этой задачи лучше идти от противного. У составного числа, которое можно представить в виде произведения простых чисел, должен быть как минимум один делитель, значение которого не более корня квадратного из N. Приведем доказательство этого утверждения. Самыми большими будут сомножители, если у составного числа их будет только два, меньше уже быть не может

$$N = p_1 \cdot p_2.$$

Хотя бы один из простых сомножителей должен принимать значение $p_i <= \sqrt{N}$, иначе p_i^2 будет больше N. Проверим это на простых примерах. Для $N = 9$ его каноническое разложение на простые сомножители, или факторизация числа, будет $N = 3 \cdot 3$, получим $p_1 = \sqrt{9}$ и $p_2 = \sqrt{9}$, т.е. если каноническое разложение происходит на два одинаковых простых числа, то они в точности равны $\sqrt{N}$.

Для $N = 21$ его каноническое разложение будет $N = 3 \cdot 7$ и $p_1 = 3$ меньше, чем $\sqrt{21}$ ($\sqrt{21} \approx 4{,}58$). При каноническом разложении на два различных сомножителя один из них будет меньше, чем $\sqrt{N}$.

Таким образом, если мы будем испытывать число на делимость, начиная с делителя простого числа 2, то ряд потенциальных делителей простых чисел можно заканчивать на значении $q <= \sqrt{N}$. Если в этом диапазоне нет ни одного делителя числа N, то оно является простым числом. Этим, собственно, отличается задача определения всех делителей натурального числа от проверки его на простоту.

Но, оказывается, при определении простых чисел в диапазоне от 2 до N можно вообще обойтись без операций проверки делимости. Если нам известна начальная последовательность простых чисел, например, 2, 3, 5, 7, 11, 13, то из ряда натуральных чисел — претендентов на звание простых чисел — сразу можно вычеркнуть все числа, кратные представленному ряду простых чисел. Причем проверку числа на кратность можно даже и не производить, достаточно для делителя 2 обращаться к каждому второму числу на числовой оси и их вычеркивать, для 3 — к каждому третьему и т.д. Если мы

выполним эти действия от двух до верхней границы ряда натуральных чисел N, то невычеркнутыми останутся только простые числа в этом диапазоне значений.

Для наглядности рассмотрим последовательные этапы для ряда чисел от 2 до $N = 20$. После вычеркивания каждого второго числа после первого простого числа 2 получим:

2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~,11, ~~12~~,
13, ~~14~~, 15, ~~16~~, 17, ~~18~~, 19, ~~20~~.

Следующим после простого числа 2 невычеркнутым окажется простое число 3, после него, вычеркивая каждое третье число, получим:

2, 3,~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~,
13, ~~14~~, ~~15~~, ~~16~~, 17, ~~18~~, 19, ~~20~~.

На этом можно остановиться и отметить, что следующее невычеркнутое простое число будет 5, значение которого уже больше, чем $\sqrt{N} = \sqrt{20} \cong 4$, 47 или округленно 4.

Мы, собственно, рассмотрели алгоритм, который предложил Эратосфен Киренский (276 год до н. э. — 194 год до н. э.) — греческий математик, астроном, географ и поэт. Эратосфен написал на папирусе, натянутом на рамке, все числа от 1 до 1000 и прокалывал составные числа. Папирус стал как решето, которое "просеивает" составные числа, а простые оставляет. Поэтому такой метод и получил название "решето Эратосфена".

Программная реализация решета Эратосфена

В наши дни легче воспользоваться ПК, чем найти папирус или таблички, покрытые воском, поэтому для воспроизводства алгоритма решета Эратосфена ряд натуральных чисел можно представить в виде линейного массива, индексы которого будут в диапазоне до значения N. Именно индексы элементов массива используются в качестве значений натуральных чисел. А какие значения хранят сами элементы массива? Они хранят признак — "вычеркнули" или "не вычеркнули", чем могут быть значения, соответственно, `false` или `true`.

Из источника [2] приведем фрагмент программы на языке Pascal для определения простых чисел в интервале от 2 до n.

```
q := round(sqrt(n));
for i := 2 to q do
 if a[i] then
  begin
   j := i * i;
   while j <= n do
    begin
     a[j] := false;
     j := j + i;
    end;
  end;
```

В приведенном фрагменте рассматривается линейный массив `a[i]` логических значений типа `boolean`, индексы которого — в интервале от 2 до n. Изначально всем элементам массива должны быть присвоены значения `true`. Значение `q` определяется как наибольшее значение в ряду делителей `i`, где `i` одновременно является и индексом массива `a[i]`. В переменную `j` записывается очередной номер элемента, который будет кратен `i`, т.е. отстоит от очередного значения `a[j]` на `i`. Подумайте, почему начальное значение `j` равно `i * i`?

Как следует из приведенного фрагмента программы, непосредственных операций деления одного числа на другое не используется, и в итоге в диапазоне от 2 до n номера элементов массива `a[i]` со значением `true` будут простыми числами. Эти значения можно вывести на экран или записать в любую другую структуру натуральных чисел, например, в линейный массив или в файл, используя цикл

```
for i := 2  to n do
  if a[i] then write (i, '  ');
```

В настоящее время имеется достаточно много дополнений и усовершенствований базового алгоритма решета Эратосфена, с ними можно познакомиться по ссылкам в [2].

Общим для этих алгоритмов остается то, что определяются простые числа на интервале от 2 до N. Следовательно, если мы задаем новую границу ряда натуральных чисел N, алгоритм будет заново выполнять определение простых чисел, в том числе и ранее определенных.

Если с течением времени накапливать коллекцию простых чисел, то естественное решение — ее надо хранить на устройствах постоянной памяти, т.е. в файле. Это решение очень важное во многих отношениях, в том числе и в необходимости модификации известных алгоритмов и программ. При хранении файла на жестком диске современных ПК его емкость можно считать неограниченной для нашей задачи. Реальным ограничением будет ограничение на величину целого числа. Если для начала ограничиться стандартными возможностями, то для типа `longint` удовлетворимся формированием коллекции в пределах от 2 до 2 147 483 647 (в других системах программирования и в зависимости от разрядности процессора это ограничение будет гораздо выше).

Ниже представлен фрагмент программы, в которой для определения простых чисел в интервале от 2 до значения N используется типизированный файл `f` записей логического типа.

Если в файл `f` записать $N + 1$ значений `true`, то после реализации представленного фрагмента программы номера записей в файле, значение которых `true`, будут простыми числами.

```
bo := true;
b := false;
q := round(sqrt(N));
for i := 0 to N do
 begin {для всех чисел пока
        записываем значение "истина"}
  write(f,bo);
 end;
```

```
seek(f,0);
for i := 2 to q do
 begin
  seek(f,i);
  read(f,bo);
  if ( bo ) then
   begin
    j := i * i;
    while j <= N do
      begin
       seek(f,j);
       write(f,b);
       j := j + i;
      end;{while}
   end;{then}
 end;{for i}
seek(f,0);
repeat
  read(f,bo);
  if bo then write(FilePos(f) - 1,' ');
until Eof(f);
```

Приведенный фрагмент мало чем отличается от ранее приведенного фрагмента с использованием массива `a[i]`. Поскольку вместо линейного массива используется файл `f`, то для обращения к его записям используется процедура позиционирования внутреннего указателя `seek(f, i)`, которая перемещает его к началу записи с номером `i` (нумерация записей в файле начинается с нуля). Функция `FilePos(f)` возвращает текущее значение позиции внутреннего указателя, при этом необходимо иметь в виду, что при операции чтения из файла или записи в него внутренний указатель автоматически смещается на одну позицию. Функция `Eof(f)` возвращает истину, если указатель будет находиться перед меткой конца файла (*end of file*). Пожалуй, это и все пояснения, которые помогут прочитать и понять работу приведенного фрагмента.

В конце фрагмента программы цикл `repeat` выводит на экран все простые числа из диапазона от 2 до *N* в качестве номеров записей в файле, значения которых `true` (остались "невычеркнутые").

После реализации программы на своем ПК может возникнуть вопрос: "Какое конкретное значение верхней границы натуральных чисел *N* можно задать?" Использование в программе файлов обеспечивает надежную сохранность полученных результатов, но существенно замедляет ее работу. Как решить эту проблему при создании большой коллекции простых чисел, рассмотрим в следующем разделе.

Создаем коллекцию простых чисел на своем ПК

Ранее отмечалось, что в алгоритме решета Эратосфена при изменении верхней границы диапазона *N* заново определяются все простые числа от начала диапазона.

Сформулируем задание: необходимо разработать программу, которая по желанию пользователя будет пополнять уже имеющуюся коллекцию простых чисел, основываясь на алгоритме решета Эратосфена, но не повторять определение ранее определенных простых чисел. Евклидом доказано, что ряд простых чисел бесконечен, поэтому любая коллекция будет ограничена верхней границей доступного диапазона натуральных чисел.

Введем обозначения: *N* — верхняя граница ранее обработанного диапазона натуральных чисел; *newN* — новая, больше *N* граница, до которой необходимо дополнительно обработать диапазон от $N + 1$ до *newN*. Таким образом, необходимо дополнительно обработать $(newN - N)$ чисел.

Принимаем следующее условие: дополнительному количеству чисел будет соответствовать дополнительный файл с необходимым количеством записей значений `true`. Их нумерация в файле начинается с нуля.

На *рис.* 1 представлена схема записей в итоговом файле `fbool`, где уже обработаны записи, и для записей со значением `true` их номера являются простыми числами. В дополнительном файле `fre` записи с номерами от 0 до `newN` изначально все со значением `true`.

Необходимо решить два вопроса с целью объединения двух файлов в одном алгоритме.

Решим первый вопрос: какому значению в дополнительном диапазоне натуральных чисел соответствует номер записи в дополнительном файле? Если исключить использование записи с номером 0, то номер записи 1 будет соответствовать значению числа $N + 1$ и так далее, номер последней записи будет соответствовать числу *newN*.

Решим второй вопрос: какие номера записей в дополнительном файле необходимо "вычеркивать"? Кратные номерам записей в результирующем файле, являющимся простыми числами, начиная с номера 2, затем 3 и т.д. Для каждого простого числа можно определить начальный номер *j* в дополнительном файле для значения *i* простого числа по следующей формуле:

$$j = (N \operatorname{div} i) \cdot i + i - N. \qquad (1)$$

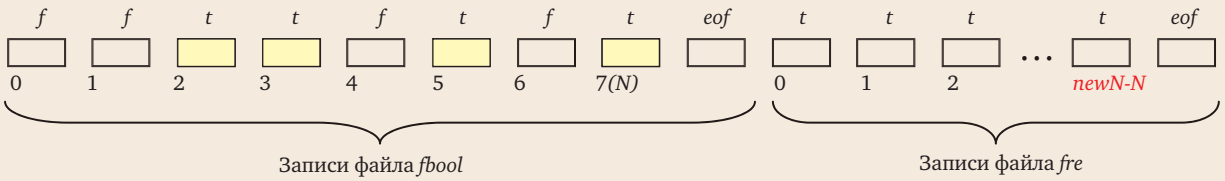


Рис. 1. Схема записей в итоговом и дополнительном файлах

Для приведенной на *рис.* 1 схемы имеем $N = 7$. С какого номера j необходимо в дополнительном файле "вычеркивать" записи, кратные первому простому числу $i = 2$:

$$j = (7\ div\ 2) \cdot 2 + 2 - 7 = 3 \cdot 2 + 2 - 7 = 1.$$

Следовательно, "вычеркивать" необходимо каждую вторую запись, начиная с первого номера. Записи с номером $j = 1$ соответствует число $N + 1 = 8$, т.е. это значение, кратное 2.

Для $i = 3$ получим $j = (7\ div\ 3) \cdot 3 + 3 - 7 = 2 \cdot 3 + 3 - 7 = 2$. Следовательно, начиная с номера 2 и далее с шагом 3 необходимо "вычеркивать". Запись с номером $j = 2$ соответствует числу $N + 2 = 9$, т.е. это число, кратное 3.

Для любых значений N и i формула (1) дает начальный номер записи, с которой в дополнительном файле начинается "вычеркивание". Когда все записи в дополнительном файле будут обработаны, их необходимо дописать в результирующий файл, после чего номер последней записи будет $N = newN$.

Рассмотренные вопросы и ответы на них решают задачу последовательного накопления коллекции простых чисел в результирующем файле в пределах ограничений на тип целых чисел используемого языка программирования и разрядности процессора. Для языка Pascal этим ограничением будет наибольшее значение типа `longint`, равное 2 147 483 647.

Программная реализация задачи зависит от поставленных в ней конечных целей. Как и любое другое коллекционирование, будем считать коллекционирование простых чисел делом неспешным. Коллекционер сам должен принимать решения, на сколько и как пополнить свою коллекцию, а коллекция должна быть доступна для ее обозрения. Исходя из сформулированной концепции, принимаем для ее реализации следующие параметры будущей программы: программа формирует три файла, из них

— **первый файл** является основным результирующим файлом, содержащим обработанный диапазон записей `true/false`, и соответствует файлу `fbool` на *рис.* 1. Количество записей в нем равно количеству всех натуральных чисел обработанного диапазона. Использование этого файла без сложных алгоритмов может решать следующие задачи:

а) проверка на простоту введенного числа, значения которого в пределах обработанного диапазона;

б) вывод на экран всех простых чисел от заданных нижней и верхней границ диапазона;

— **второй файл** также является результирующим (на *рис.* 1 не представлен), но он содержит только последовательность простых чисел (тип `longint`). Записей в файле будет намного меньше, чем в первом файле (среди всех натуральных чисел доля простых чисел менее 6%), но на их запись потребуется не 1 байт (тип `boolean`), а 4 байта (тип `longint`). Наличие такого файла позволяет легко решать задачи:

а) предоставление информации о количестве простых чисел в коллекции;

б) представление коллекции простых чисел в текстовом формате;

— **третий файл** с заданным количеством первоначальных записей `true` временно создается для обработки дополнительного интервала натуральных чисел в соответствии со схемой на *рис.* 1, его имя на схеме `fre`. После завершения его обработки полученные результаты дописываются в результирующие файлы и он удаляется, тем самым будет выполняться поэтапное пополнение коллекции простых чисел.

Выполнение различных задач пользователь программы может выбрать из предлагаемого меню программы, которое предлагает, в том числе, автоматический режим пополнения коллекции определенными порциями, пока не будет достигнута указанная верхняя граница диапазона. Пользователь может спокойно спать, а коллекция всю ночь будет пополняться миллион за миллионом. В конечном итоге многое будет зависеть от мощности используемой вычислительной машины.

Алгоритм формирования коллекции простых чисел

В алгоритме и, соответственно, в программе можно предусмотреть шесть пунктов меню, пять из которых могут циклически выбираться по желанию пользователя:

1. Дополнение коллекции простых чисел до верхней границы `MaxInt`.
2. Информация о количестве простых чисел в коллекции.
3. Проверка числа из обработанного диапазона на принадлежность к простым числам.
4. Список простых чисел из диапазона, заданного пользователем.
5. Режим автоматического пополнения коллекции.
6. Завершение работы программы.

На *рис.* 2 представлена сокращенная схема алгоритма формирования коллекции простых чисел на основе алгоритма решета Эратосфена.

Вначале (1) проверяется наличие в текущей директории результирующих файлов `fbool` и `fpr`. Файл `fbool` содержит записи логического типа (`true/false`). Номер записи со значением *true* является простым числом. Файл `fpr` содержит записи простых чисел в их последовательности от 2 до верхней границы обработанного диапазона. Если файлы еще не создавались, то они создаются пустыми (2), и в них записывается начальная последовательность простых чисел от 2 до 11 (3). Верхней границе диапазона присваивается новое значение `MaxInt = 100` (4) и вызывается подпрограмма `Eratos (MaxInt)` (5).

В подпрограмму `Eratos(X)` передается верхнее значение диапазона натуральных чисел `X`

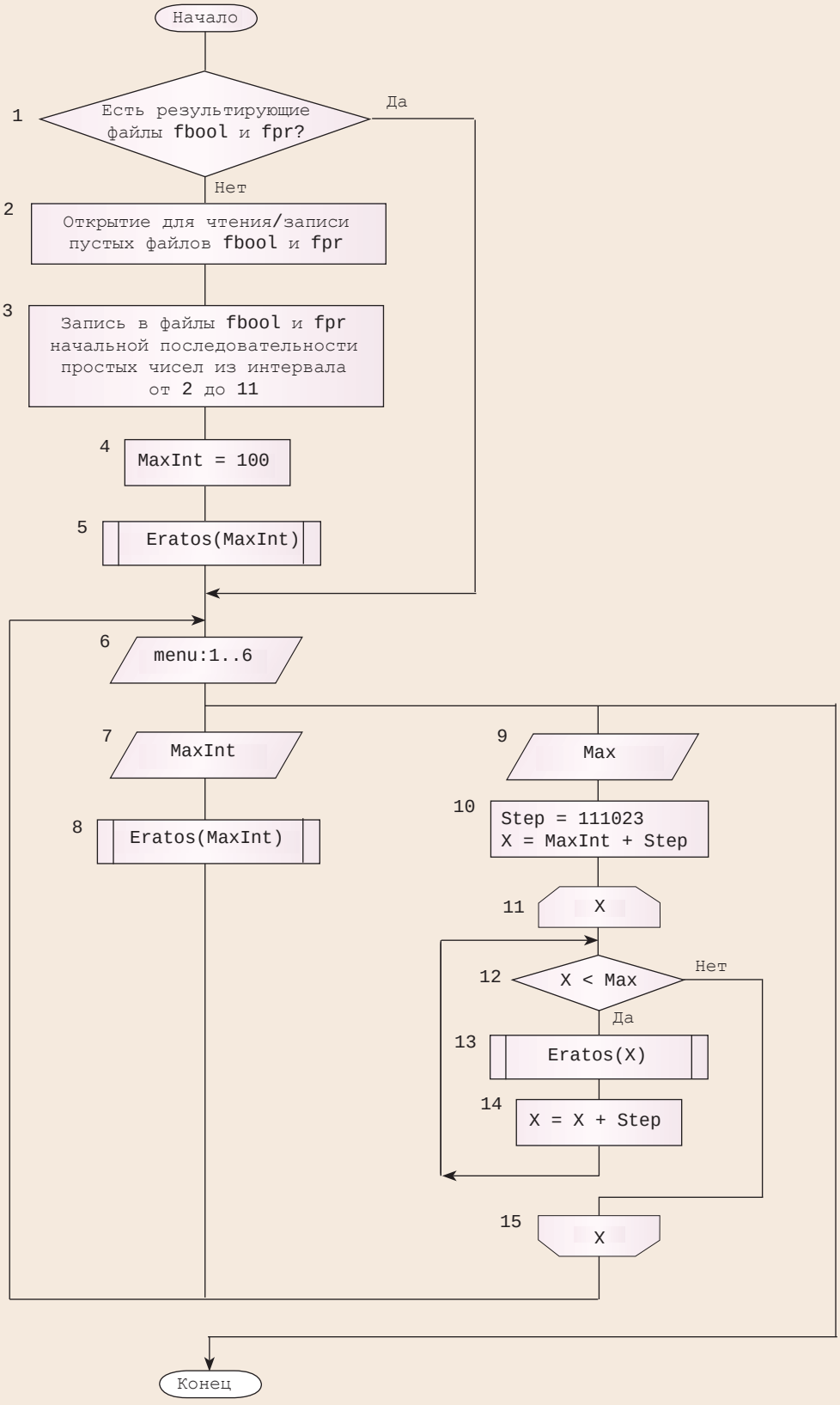


Рис. 2. Схема алгоритма формирования коллекции простых чисел

(большее, чем у ранее обработанного диапазона). Подпрограмма создает вспомогательный файл `fre`, в который записываются значения `true` столько раз, сколько натуральных чисел необходимо обработать до новой верхней границы диапазона. Затем определяются номера элементов по формуле (1) и далее в соответствии с алгоритмом решета Эратосфена в файл перезаписывают значения `false` тем записям, номера которых являются составными числами. После завершения обработки записей вспомогательного файла `fre` все его записи дописываются в результирующий

файл fbool, а в результирующий файл fpr дописываются простые числа.

Далее пользователю предлагается выбор одного из шести пунктов меню (6). Пункты меню 2...4 выполняют простые операции информационного характера и на схеме алгоритма не отражены.

При выборе первого пункта меню пользователю предоставляется возможность ввода нового значения верхней границы диапазона натуральных чисел MaxInt (7), после чего выполняется подпрограмма Eratos(MaxInt) (8) и возврат к пункту (6) алгоритма.

При выборе пятого пункта меню пользователь вводит верхнюю границу диапазона натуральных чисел Max (9), и далее уже без участия пользователя обработка дополнительного диапазона производится в цикле по X (11) с шагом Step = 111 023 (10).

Если X еще меньше, чем Max, то выполняется очередная итерация цикла. Вызывается подпрограмма Eratos(X) (13), затем значение X увеличивается на шаг Step (14). Если X превысит верхнюю границу диапазона Max, то цикл заканчивается (15) и происходит возврат к пункту (6) алгоритма.

Если пользователь выберет шестой пункт меню, то работа программы завершается.

Для удобства чтения представленной программы желательно познакомиться со спецификацией, представленной в таблице.

На с. 16 представлен листинг программы, реализующей перечисленные выше задачи. Для удобства комментирования строки программы пронумерованы.

Тело основной программы начинается со строки 80. В строках 82–85 файловые переменные ассоциируются с полными именами физических файлов. Если файл fbool еще не создавался (строки 89–108), то создаются пустые файлы fbool и fpr и открываются для чтения/записи. И в них записываются стартовые значения из диапазона от 0 до 11, соответственно в файле fbool значения true/false, а в файле fpr простые числа 2, 3, 5, 7, 11. Затем вызывается процедура Eratos, которая дополнительно обработает на основе алгоритма решета Эратосфена ряд натуральных чисел до верхней границы, равной 100. Эта граница обеспечена простыми числами, уже имеющимися в результирующем файле fpr, наибольшее значение которого 11 превышает значение $q = \sqrt{100} = 10$. В дальнейшем пользователю будет выводиться на экран предел верхней границы, для обработки которой уже достаточно простых чисел в результирующем файле.

Таблица "Спецификация"

№ п/п	Обозначение	Назначение	Тип	Байт
1	PathPr	Строковая константа, содержащая полное имя файла простых чисел	string	256
2	PathBo	Строковая константа, содержащая полное имя результирующего файла с записями true/false	string	256
3	PathRe	Строковая константа, содержащая полное имя вспомогательного файла записей true/false (для пополнения коллекции)	string	256
4	fpr	Файловая переменная для PathPr	file of longint	–
5	fbool	Файловая переменная для PathBo	file of boolean	–
6	fre	Файловая переменная для PathRe	file of boolean	–
7	bt, bf	Переменные	boolean	1
8	MaxInt	Граница обработанного диапазона	longint	4
9	newMax	Новая граница верхнего диапазона	longint	4
10	step	Шаг пополнения коллекции в автоматическом режиме	longint	4
11	q	Значение наибольшего делителя	longint	4
12	i, j	Переменные (номера записей в файлах и др.)	longint	4
13	top	Верхняя граница обрабатываемого диапазона (в процедуре Eratos)	longint	4
14	pr	Простое число (в процедуре Eratos)	longint	4
15	Eratos	Процедура. Обрабатывает дополнительный диапазон натуральных чисел на простоту	–	–
16	Range	Процедура. Выводит на экран простые числа из заданного диапазона	–	–
17	bootom	Нижняя граница диапазона (в процедуре Range)	longint	4

Если файл fbool существует, в строках 109–121 проверяется наличие файла fpr. Алгоритм предусматривает одновременное наличие соответствующих друг другу файлов fbool и fpr. Если по какой-то причине не оказывается файла fpr при наличии файла fbool, то файл fbool удаляется и программа будет готова начать свою работу с самого начала.

Если с наличием результирующих файлов все в порядке, то со строки 122, где установлена метка для безусловного перехода goto, пользователю будет циклически на экран выводиться меню программы с пунктами от 1 до 6 (строки 128–133), где пункт 6 предусматривает выход из программы.

Вывод меню сопровождается информацией о верхней границе уже обработанного диапазона натуральных чисел. Выполнение выбранного пункта меню реализуется оператором Case (строки 135–191). После выполнения пункта меню осуществляется возврат программы на метку в строке 122 (Metka:).

Основную работу в программе выполняет процедура Eratos, в которую передается новое значение верхнего диапазона натуральных чисел, копируемого в переменную top (строки 15–57). Процедура реализует алгоритм на базе решета Эратосфена, но для обработки дополнительного диапазона используются записи файла fre, а значения ранее определенных простых чисел последовательно считываются из файла fpr до значения $q <= \sqrt{top}$.

После завершения обработки записей типа boolean в файле fre выполняется пополнение файлов fbool и fpr (строки 43–56). В файле fpr, в том числе в нулевую запись, помещается новое значение верхней границы обработанного диапазона.

При использовании программы рекомендуется сначала в интерактивном режиме накопить коллекцию до верхней границы, равной корню квадратному из наибольшего значения типа переменной pr. Для типа longint языка Pascal это будет значение 46 341, что достигается в программе очень быстро. Затем можно использовать автоматический режим пополнения коллекции порциями в соответствии со значением переменной *step*, которая в исходном тексте принимает значение step = 111 023. Это значение без последовательности одних

```
1.  Program f3pr_bool_doo2;
2.  Uses Crt;
3.  Const PathPr = 'c:\TP\user\Simplnum.ppp';
4.    PathBo = 'c:\TP\user\avtobool.bbb';
5.    PathRe = 'c:\TP\user\rebool.bbb';
6.  Label Metka;
7.  Var fpr:file of longint;
8.    fbool:file of boolean;
9.    fre:file of boolean;
10.   bt,bf:boolean;
11.   MaxInt,Max,newMax,X:longint;
12.   i,j,q,step,pr:longint;
13.   menu:byte;
14.   code:integer;
15. Procedure Eratos(top:longint);
16. Var MaxInt:longint;
17.     bt,bf:boolean;
18.     i,j,q,start:longint;
19. begin
20.   writeln('Ждите, процессор пыхтит!');
21.   rewrite(fre);
22.   MaxInt := FileSize(fbool) - 1;
23.   Start := MaxInt + 1;
24.   Bt := true;
25.   Bf := false;
26.   seek(fbool,FileSize(fbool));
27.   seek(fpr,1);
28.   for i := start to top do
29.    write(fre,bt);
30.   q := round(sqrt(top));
31.   read(fpr,pr);
32.   while pr <= q do
33.    begin
34.    j := (maxInt div pr) * pr + pr - MaxInt;
35.    while j <= (top - MaxInt) do
36.      begin
37.      seek(fre,j);
38.      write(fre,bf);
39.      j := j + pr;
40.      end;{while j}
41.    read(fpr,pr);
42.    end;{while pr}
43.   seek(fre,1);
44.   seek(fpr,FileSize(fpr));
45.   seek(fbool,FileSize(fbool));
46.   repeat
47.    read(fre,bt);
48.    write(fbool,bt);
49.    if bt then
50.    begin
51.     pr := FilePos(fre) - 1 + MaxInt;
52.     write(fpr,pr);
53.    end;
54.   until Eof(fre);
55.   seek(fpr,0);
56.   write(fpr,top);
57.   end;{Eratos}
58.
59. Procedure Range(bootom:longint; top:longint);
60. Var k,n:longint;
61. bo:boolean;
62. begin
63.   n := 0;
64.   seek(fbool,bootom);
65.   for k := bootom to top do
66.   begin
67.     read(fbool,bo);
68.     if bo then
69.     begin
70.       write(k,' ');
71.       inc(n);
72.       if n mod 200 = 0 then begin
73.         writeln;
74.         write('Для продолжения - Enter');
75.         readln;
76.       end;
```

```
77.    end;
78.  end;
79.end;{Range}
80.begin {main}
81.   Clrscr;
82.   assign(fre,PathRe);
83.   rewrite(fre);
84.   assign(fpr,PathPr);
85.   assign(fbool,PathBo);
86.   {$I-}
87.   reset(fbool);
88.   {$I+}
89.   Code := IoResult;
90.   if code <> 0 then
91.     begin
92.       rewrite(fbool);
93.       rewrite(fpr);
94.       MaxInt := 11;
95.       write(fpr,MaxInt);
96.       bt := true;
97.       bf := false;
98.       for i := 0 to 11 do
99.         case i of
100.             0,1,4,6,8,9,10:write(fbool,bf);
101.             2,3,5,7,11:begin
102.                          write(fbool,bt);
103.                          write(fpr,i);
104.                        end;
105.           end;
106.         MaxInt := 100;
107.         Eratos(MaxInt);
108.       end;{if code <> 0}
109.     {$I-}
110.     reset(fpr);
111.     {$I+}
112.     Code := IoResult;
113.     if code <> 0 then
114.       begin
115.         close(fbool);
116.         erase(fbool);
117.         close(fre);
118.         erase(fre);
119.         {файлы должны быть только в паре, fpr и fbool}
120.         Exit;
121.       end;{if code}
122.       Metka:{writeln; }
123.       MaxInt := FileSize(fbool) - 1;
124.       Max := MaxInt;
125.       if Max < 46300 then Max := trunc(sqr(Max));
126.       writeln('Обработаны числа до ',MaxInt);
127.       writeln('Введите номер пункта меню:');
128.       writeln('1 - дополнить коллекцию простых чисел;');
129.       writeln('2 - количество простых чисел в коллекции;');
130.       writeln('3 - проверить введенное число в пределах коллекции;');
131.       writeln('4 - простые числа в заданном диапазоне;');
132.       writeln('5 - автоматическое пополнение коллекции;');
133.       writeln('6 - выход.');
134.       readln(menu);
135.       Case menu of
136.       1:begin
137.           writeln('Введите новую границу верхнего диапазона ');
138.           if MaxInt < 46300 then
139.           write('(но не более ', Max,') ');
140.           readln(newMax);
141.           Eratos(newMax);
142.           goto Metka;
143.       end; {1:}
144.       2:begin
145.          Max := FileSize(fpr) - 1;
146.          writeln('В коллекции ',Max,' простых чисел');
147.          goto Metka;
148.        end;{2:}
149.       3:begin
150.          write('Введите число ');
151.          readln(X);
```

```
152.           if X > MaxInt then goto Metka;
153.           seek(fbool,X);
154.           read(fbool,bt);
155.           if bt then writeln('ПРОСТОЕ')
156.              else writeln('НЕ ПРОСТОЕ');
157.           goto Metka;
158.         end;{3:}
159.        4:begin
160.           write('Введите нижнюю границу диапазона ');
161.           readln(i);
162.           write('Введите верхнюю границу диапазона ');
163.           readln(j);
164.           Range(i,j);
165.           writeln;
166.           goto Metka;
167.         end; {4:}
168.        5:begin
169.           MaxInt := FileSize(fbool) - 1;
170.        write('Введите границу диапазона больше ',MaxInt,': ');
171.           readln(Max);
172.           step := 111023;
173.           X := MaxInt + step;
174.        writeln('Ждите, автоматическое пополнение коллекции:');
175.           while X <= Max do
176.              begin
177.                 writeln(X);
178.                 Eratos(X);
179.                 X := X + step;
180.              end;
181.           goto Metka;
182.         end;{5:}
183.        6:begin
184.           close(fbool);
185.           close(fpr);
186.           close(fre);
187.           erase(fre);
188.           Exit;
189.         end;
190.        else begin goto Metka; end;
191.        end; {case}
192.   end.
```

нулей делает более легким считывание с экрана чисел с большой разностью, а величина пополнения коллекции примерно на сто тысяч обеспечивает надежность работы программы и информирование пользователя о поступательном продвижении до заданного им значения верхнего диапазона пополнений коллекции.

Качество своей коллекции можно проверить с помощью специального сайта *http://ru.numheteremp ire.com*, предоставляющего посетителям услугу по установлению принадлежности введенного числа к простым числам и его факторизацию.

Читателям предоставляются широкие права по внесению дополнений и изменений в представленный исходный код, в котором представлен только необходимый минимум для выполнения перечисленных задач.

Желательно, чтобы читатели создали свой исходный код на языке программирования, которым они владеют, ведь реализация на языке Pascal преследует в основном задачу реализации алгоритма, понятную более широкому кругу читателей. При реализации на другом языке программирования, возможно, наиболее трудным будет использование соответствующих функций при работе с файлами данных. Перечислим их:

`assign(f, Name)` — процедура ассоциирует файловую переменную `f` с полным именем физического файла `Name`, где `Name` — значение строкового типа;

`reset(f)` — процедура открывает существующий файл для чтения/записи;

`rewrite(f)` — процедура создает пустой файл и открывает его для чтения/записи;

`close(f)` — процедура закрывает файл;

`erase(f)` — процедура удаляет физический файл, с которым была ассоциирована переменная `f`;

`seek(f, i)` — процедура устанавливает внутренний указатель файла перед записью с номером `i`;

`IoResult(f)` — функция возвращает код успеха открытия файла для чтения/записи; если попытка открытия файла прошла успешно, то функция возвращает значение кода 0;

`FileSize(f)` — функция возвращает значения количества записей в файле (будет на 1 больше, чем номер последней записи);

`Eof(f)` — функция возвращает значение логического типа, например `true`, если достигнут конец файла *end of file*, и `false` в противном случае;

`FilePos(f)` — функция возвращает номер записи, на которую указывает внутренний указатель.

Для ознакомления с более изощренными программными реализациями задачи определения простых чисел можно обратиться к источникам [1, 3] и далее по многочисленным ссылкам на интернет-ресурсы.

Определение больших простых чисел

Повествование о простых числах можно считать незавершенным, пока не будут раскрыты вопросы определения очень больших простых чисел. На сегодняшний день самое большое, известное человечеству, простое число содержит в своей записи 12 978 189 десятичных цифр. В связи с такой огромной величиной числа возникает ряд вопросов:

- как формируют такие числа;
- как их хранят в памяти ПК;
- как над ними выполняются арифметические операции;
- как проверяется их принадлежность к простым числам.

Как формируют большие числа

Последовательность натуральных чисел бесконечна, и так же бесконечна россыпь в них простых чисел. Неэффективно проверять каждое следующее число на простоту. Существуют различные способы формирования значения натуральных чисел, вероятность принадлежности которых к простым числам намного выше, чем у обычной последовательности чисел на числовой оси. Наибольшую известность и использование в определении самых больших простых чисел получили числа Мерсенна, которые вычисляются по формуле

$$M_p = 2^p - 1, \qquad (2)$$

где p — простое число.

У самого большого известного простого числа Мерсенна $p = 43\,112\,609$.

Названы эти числа в честь французского монаха, физика, математика Марена Мерсенна, родившегося в 1588 г. В “Физико-математических размышлениях” (1644) Мерсенн утверждал, что числа $2^p - 1$ являются простыми для $p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$. Этот факт произвел сильное впечатление на его современников. Ведь оперировать такими огромными числами в те времена было делом немыслимым. Через сто лет Эйлер доказал простоту числа M_{31}. И хотя, как выяснилось позже, в этом списке два числа M_{67} и M_{257} составные, можно лишь поражаться глубине проникновения Мерсенна в структуру таких чисел.

Келья монаха Мерсенна являлась центром общения и обмена информацией между выдающимися учеными всей Европы. По замечанию Блеза Паскаля (в свою очередь, язык программирования Pascal назван в его честь), монах ордена миноритов имел уникальный талант ставить новые научные проблемы, но не разрешать их. Возможно, они так и не были бы решены, если бы он их не ставил. Именно этот талант и обусловил его миссию посредника в кругу самых знаменитых ученых того времени в Европе. Примечательно, что в одном из писем Р.Декарту Мерсенном было сформулировано более 30 вопросов, требующих разрешения. Основными результатами обширных научных исследований Мерсенна стало определение скорости распространения звука в атмосфере, результаты исследований по музыкальной акустике, была предложена схема зеркального телескопа. Умер Мерсенн 1 сентября 1648 г. после тяжелой болезни и неудачной операции, но даже свою кончину он использовал во благо науки. По его распоряжению были проведены исследования, выявившие ошибку операции. Это последние сведения, данные Мерсенном науке. На основе кружка Мерсенна (так называли регулярные встречи ученых в келье) была создана Парижская академия наук в 1666 г., в Англии — Лондонское королевское общество в 1662 г.

Марен Мерсенн

Как видно из формулы (2), усилия по формированию коллекции простых чисел не являются напрасными и могут быть востребованны в формировании чисел Мерсенна в поиске очередного самого большого простого числа. Их в коллекции с лихвой хватит на многие годы вперед, только в диапазоне натуральных чисел от 43 млн. до 53 млн. их около 560 тысяч. Число Мерсенна, которое по степени двойки отстоит от рекордсмена на 10 млн. (а именно $M_p = 2^{53\,112\,617} - 1$), содержит в своей записи 15 988 491 десятичных цифр, для его распечатки на бумаге формата А4 потребуется 4750 страниц. В его записи первые и последние 13 цифр 7331835188054…9765671043071, возможно, это число тоже является простым, но это надо еще доказать.

Как хранить большие числа в памяти ПК

Использование больших чисел в памяти ПК в информатике рассматривается в специальном направлении, получившем название “длинная арифметика”. В настоящее время в науке и повседневной практике для отображения чисел используется позиционная десятичная система счисления, алфавит которой: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Всего в алфавите десять цифр, поэтому система счисления и называется *десятичная*. Величина 10 называется основанием системы счисления, а самая большая цифра меньше основания на 1. Заметим,

что “цифра” и “число” — это разные понятия, но в быту между ними чаще всего не видят разницы и отдают предпочтение слову “цифра”. Даже из новостей мы можем слышать сообщение: “Окончательная цифра бюджета страны на следующий год еще не установлена”, хотя речь идет явно о числе.

Любое число можно записать с помощью цифр, а для записи целого числа меньше десяти потребуется одна цифра. В дальнейшем будем рассматривать только целые числа. Одной и той же цифрой можно, например, записать число 555, в котором 5 сотен, 5 десятков и 5 единиц. От положения цифры в записи числа зависит величина, которую она отображает. Место в записи числа называется разрядом, и их счет начинается с нуля, т.е. в нулевом разряде записываются единицы числа. Величины в смежных разрядах, записанные одинаковыми цифрами, отличаются в 10 раз.

Из рассмотренного порядка записи чисел в десятичной системе счисления вытекает простой и наглядный способ организации хранения больших чисел в памяти ПК. Достаточно запись числа представить в виде линейного массива с нумерацией его элементов от нуля и выше по необходимости. Нулевой элемент будет хранить количество единиц, первый — десятки, второй — сотни и так далее. Таким образом, от привычной записи числа его структура в линейном массиве будет “перевертышем”, т.е. цифры будут следовать в обратном порядке. Для современных ПК и систем программирования размещение в оперативной памяти нескольких линейных массивов длиной 100 млн. элементов и более не составляет проблемы, что существенно больше для размещения известных самых больших простых чисел Мерсенна. То, что десятичные числа автоматически преобразуются в двоичный код, в рамках данной статьи можно опустить. В записи длинных чисел уже нет возможности проговаривать величины в различных разрядах, таких названий просто не существует. Неизменным остается соотношение в смежных разрядах, что будет использовано в арифметических операциях над ними.

Арифметические операции над длинными числами

Рассмотрим самый простой пример арифметической операции сложения двух чисел столбиком:

$$\begin{array}{rr} & 1234567890 \\ + & 987654321 \\ \hline & 2222222211 \end{array}$$

Они записываются друг под другом с условием выравнивания их с нулевых разрядов. Затем, начиная с разрядов единиц, определяется сумма цифр двух чисел и в итог записывается остаток от целочисленного деления на 10, а целое количество десятков переносится в сумму следующего старшего разряда. Эта схема сложения без изменений повторяется многократно, пока не завершится запись цифры в самый старший разряд итога.

Ниже приводится демонстрационный пример программы на языке Pascal, реализующий арифметическую операцию сложения двух чисел, представленных в виде линейных массивов различной длины.

```
Program sum_a_b;
Uses Crt;
Const N = 1002;
Type Tmas = array[0..N] of byte;
Var a,b,sum:Tmas;
    rega,regb,regsum:integer;
    s,p:byte;
    i:word;
begin
     clrscr;
     randomize;
     for i := 0 to N do
         begin
              a[i] := 0;
              b[i] := 0;
              sum[i] := 0;
         end;
     write('Введите разрядность
            первого слагаемого ');
     readln(rega);
     dec(rega);
     write('Введите разрядность
            второго слагаемого ');
     readln(regb);
     dec(regb);
     for i := 0 to rega do
         a[i] := random(10);
     if (a[rega] = 0) then
         a[rega] := random(9) + 1;
     for i := 0 to regb do
         b[i] := random(10);
     if (b[regb] = 0) then
         b[regb] := random(9) + 1;
     regsum := -1;
     i := 0;
     p := 0;
     while ((i <= rega) or (i <= regb)) do
           begin
                s := a[i] + b[i] + p;
                sum[i] := s mod 10;
                p := s div 10;
                inc(regsum);
                inc(i);
           end;
     if (p <> 0) then
        begin
             inc(regsum);
             sum[regsum] := p;
        end;
```

```
      writeln('a + b = sum');
      for i := rega downto 0 do
          write(a[i]);
      writeln;
      for i := regb downto 0 do
          write(b[i]);
      writeln;
      for i := regsum downto 0 do
          write(sum[i]);
      writeln;
      writeln('Конец программы');
      readln;
end.
```

Данная программа приведена лишь в качестве демонстрации принципиальной возможности хранения чисел в памяти ПК в привычной последовательности записи десятичных цифр и выполнения арифметических операций над ними. Объем статьи не позволяет привести примеры всех арифметических операций, в том числе целочисленного деления.

Будем считать, что проблемы программной реализации всех необходимых арифметических операций над длинными числами решены. Теперь необходимо получить запись числа Мерсенна, в котором показатель степени двойки может принимать значения порядка 40 млн. Самый простой алгоритм вычисления заданной степени двойки — это последовательное умножение начального значения, равного единице, соответствующее степени количество раз. Эта идея реализована в нижеприведенной программе.

```
Program step2;
Uses Crt;
Const N = 1000;
Type Tmas = array[0..N] of byte;
Var m:Tmas;
   regm:word;
   st,i,k:word;
   p,s:byte;
begin
   clrscr;
   for i := 0 to N do
     m[i] := 0;
   m[0] := 1;
  write('Введите значение степени двойки st=');
   readln(st);
   regm := 0;
   for k := 1 to st do
     begin
          p := 0;
          for i := 0 to regm do
            begin
              s := m[i] * 2 + p;
              m[i] := s mod 10;
              p := s div 10;
            end;
          if (p <> 0) then
            begin
              inc(regm);
              m[regm] := p;
            end;
      end;
    write('2^',st,'=');
    for i := regm downto 0 do
      write(m[i]);
    writeln;
    writeln('Конец программы');
    readln;
end.
```

В итоге, если ввести значение st = 10, программа выдаст хорошо известное в информатике значение $2^{10} = 1024$.

Для расширения диапазона вычисляемых значений необходимо в программах использовать не статические, а динамические линейные массивы, но при этом могут потребоваться большие затраты процессорного времени. Для ускорения вычислений можно воспользоваться свойством сложения степеней

$$2^a + 2^b = 2^{a+b}.$$

Например, если необходимо вычислить $2^{44\,000\,000}$, а ранее уже были вычислены значения $2^{43\,000\,000}$ и $2^{1\,000\,000}$, то достаточно перемножить эти два числа для получения искомого результата. Можно подвести итог, что нет принципиальных ограничений в вычислениях в понятиях длинной арифметики больших значений чисел Мерсенна. Проблема программной реализации заключается в разработке быстрых алгоритмов необходимых вычислений. Например, известен алгоритм быстрого умножения, предложенный в 1962 г. А.А. Карацубой. Понятно, что в конечном итоге имеет смысл использовать более современный и мощный язык программирования, например *C++*, с помощью которого автором были выполнены все рассмотренные вычисления.

Как проверить принадлежность числа Мерсенна к простым числам

Получение очередного значения числа Мерсенна — это всего лишь первый и незначительный по затратам процессорного времени результат в поиске очередного самого большого простого числа. Если читатель сформировал коллекцию простых чисел, то можно убедиться, какое большое количество их расположено хотя бы в ближайшем диапазоне исследований от 43 млн. до 53 млн. И каждое значение M_p необходимо проверить на принадлежность к простым числам. Алгоритм решета Эратосфена здесь уже бессилен. Как отмечалось в одном из учебников по математике, “решето Эратосфена годится для поиска простых чисел Мерсенна не более, чем консервная банка для плавания через

Атлантический океан". Однако, отдавая должное заслугам великого грека, можно сказать, что решето Эратосфена стало парусом на фрегате французского математика Франсуа Люка, занимавшегося теорией чисел. Для проверки больших чисел существуют другие алгоритмы, которые подразделяются на вероятностные и детерминированные. В теории чисел признаются лишь те числа простыми, простота которых доказана детерминированными алгоритмами. Числам Мерсенна "повезло", для доказательства их принадлежности к простым числам есть относительно нетрудоемкий детерминированный алгоритм — тест Люка – Лемера.

Французский математик Франсуа Эдуард Анатоль Люка родился 4 апреля 1842 г. Важнейшие работы Люка относятся к теории чисел и неопределенному анализу. В 1878 г. Люка предложил критерий для определения того, простым или составным является число Мерсенна, и доказал простоту числа

$M_{127} = 2^{127} - 1 = 170\,141\,183\,460\,469\,231\,731\,687\,303\,715\,884\,105\,727$.

В течение 75 лет это число оставалось наибольшим простым числом Мерсенна, известным науке, но и по сей день оно остается самым большим, вычисленным "с карандашом в руках".

Не менее известен математик Люка своими изданиями занимательной математики. До сих пор задача, известная под названием "Ханойские башни", представляет интерес для математиков и программистов, проводятся даже международные семинары, посвященные исключительно Ханойским башням.

В 1930 г. американский математик Д.Х. Лемер усовершенствовал критерий Люка, который впоследствии стал называться "тест Люка – Лемера".

Расчеты по тесту Люка – Лемера заключаются в последовательных вычислениях критерия, начальное значение которого $S_0 = 4$. Каждое следующее значение критерия вычисляется по значению предыдущего значения и числа Мерсенна по рекуррентной формуле $S_m = (S^2_{m-1} - 2)\ mod\ Mp$. Если значение последнего критерия равно нулю, т.е. $S_{p-2} = 0$, то число Мерсенна является простым числом. Например, для числа $M_7 = 2^7 - 1 = 127$.

$S_0 = 4$
$S_1 = (4 * 4 - 2)\ mod\ 127 = 14$
$S_2 = (14 * 14 - 2)\ mod\ 127 = 67$
$S_3 = (67 * 67 - 2)\ mod\ 127 = 42$
$S_4 = (42 * 42 - 2)\ mod\ 127 = 111$
$S_5 = (111 * 111 - 2)\ mod\ 127 = 0$.

Легко проверить, что для следующего простого числа $p = 11$ тест Люка – Лемера не подтверждает простоту числа Мерсенна M_{11}.

Для больших значений степеней двойки p необходимо воспользоваться длинной арифметикой. Самым трудоемким по затратам процессорного времени является деление по модулю M_p. Стандартными операциями *div* и *mod* уже не воспользуешься. Эта проблема с большими затратами времени и ресурсов, но все-таки решается за счет распределенных вычислений в сети. Существует широкомасштабный открытый интернет-проект по поиску простых чисел Мерсенна Great Internet Mersenne Prime Search (GIMPS), целью которого является поиск больших простых чисел Мерсенна путем распределенных вычислений на компьютерах любителей со всего земного шара. GIMPS выиграла денежный приз в 100 000 долл. США за нахождение простого числа из более чем 10 миллионов десятичных цифр $M_{43112609}$ и намеревается выиграть аналогичные призы в 150 000 и 250 000 долл. США, обещанные Electronic Frontier Foundation за нахождение простых чисел соответственно из более чем 100 и 1000 млн. десятичных цифр.

Для проверки на простоту числа из более чем 100 млн. десятичных цифр, при текущем развитии вычислительной и алгоритмической техники, потребуется более трех лет. Скорее всего необходимо будет дождаться новых революционных прорывов в вычислительной технике или в математике до получения очередной премии.

На диске к этому номеру имеются:

1. Исходные коды всех рассмотренных программ.
2. Самое большое простое число Мерсенна $P_{43\,112\,609} = 2^{43\,112\,609} - 1$ в формате txt.
3. Число Мерсенна $M_{53\,112\,617} = 2^{53\,112\,617} - 1$ в формате txt.
4. Коллекция простых чисел из диапазона типа Longint языка Pascal в формате txt.

Литература

1. *Душкин Р.В.* Теория чисел и язык Haskell // Потенциал, 2006, № 5.
2. *http://ru.wikipedia.org/wiki/Решето_Эратосфена*
3. *http://www.lumar.ru/?page_id=20* Простые числа. Решето Эратосфена.

годовая подшивка газеты «ИНФОРМАТИКА» на компакт-диске

ПОЛНАЯ ПОДБОРКА МАТЕРИАЛОВ ЗА 2010 ГОД

А ТАКЖЕ ТЕМАТИЧЕСКИЕ СБОРНИКИ И ПОДШИВКИ ДРУГИХ ГАЗЕТ ИД «ПЕРВОЕ СЕНТЯБРЯ»

Удобная система навигации и поиска: материалы можно выбрать по тематике, рубрике или по номеру газеты.

Для пользователей любого уровня: включи и работай — не требуются инсталляция и место на винчестере.

Компакт-диск пригоден для работы на компьютерах даже устаревшей конфигурации (Windows-95 и выше).

Стоимость диска включает доставку. Рассылка производится только на территории РФ.

КУПОН ЗАПОЛНЯЕТСЯ ПЕЧАТНЫМИ БУКВАМИ!

ФАМИЛИЯ

ИМЯ

ОТЧЕСТВО

ИНДЕКС АДРЕС

	2003 г.	2004 г.	2005 г.	2006 г.	2007 г.	2008 г.	2009 г.	2010 г.
Цена за один диск с доставкой	299 руб.	299 руб.	299 руб.	299 руб.	399 руб.	399 руб.	499 руб.	699 руб.
Английский язык	x	x	x	x	___ шт.	___ шт.	___ шт.	___ шт.
Библиотека в школе	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Биология	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
География	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Дошкольное образование	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Здоровье детей	x	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Информатика	x	x	x	x	x	x	x	___ шт.
Искусство	x	x	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
История	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Классное руководство и воспитание школьников	x	x	x	x	x	___ шт.	___ шт.	___ шт.
Литература	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Математика	x	x	x	x	x	x	___ шт.	___ шт.
Начальная школа	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Немецкий язык	x	x	x	x	___ шт.	___ шт.	___ шт.	___ шт.
Русский язык	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Спорт в школе	x	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Управление школой	x	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Химия	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Физика	x	x	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.
Французский язык	x	x	x	x	___ шт.	___ шт.	___ шт.	___ шт.
Школьный психолог	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.	___ шт.

ЭТИ ДИСКИ МОЖНО ПРИОБРЕСТИ:

- заполнив купон и отправив его в конверте с пометкой «Книга — почтой» по адресу: **ИД «Первое сентября», ул. Киевская, д. 24, г. Москва, 121165**
- заказав по телефону: **(499) 249-47-58**
- заказав по электронной почте: **podpiska@1september.ru**
- заказав на сайте: **www.1september.ru**

ТЕМАТИЧЕСКИЕ СБОРНИКИ

Цена за один диск с доставкой – 399 руб.

Газета «Начальная школа»
«50 лет системе Л.В. Занкова» ___ шт.

«1001 ёлка на Новый год» ___ шт.

Газета «Школьный психолог»
«Тренинг в теории и на практике» ___ шт.

Газета «Школьный психолог»
«Тест со всех сторон» ___ шт.

Газета «Литература»
«Консультации по темам экзаменационных сочинений» ___ шт.

Цены действительны до 31 августа 2011 года

Объектно-ориентированное программирование

Избранные разделы нового учебника

К.Ю. Поляков,
А.П. Шестаков,
Е.А. Еремин

Предлагаем вниманию читателей избранные параграфы из нового учебника для профильного курса информатики, посвященные знакомству с технологиями объектно-ориентированного программирования — одной из самых сложных не только в понимании, но прежде всего в объяснении тем. Но сложность — сложностью, а представить без этой темы современный профильный курс невозможно, с этим, пожалуй, сегодня соглашаются все.

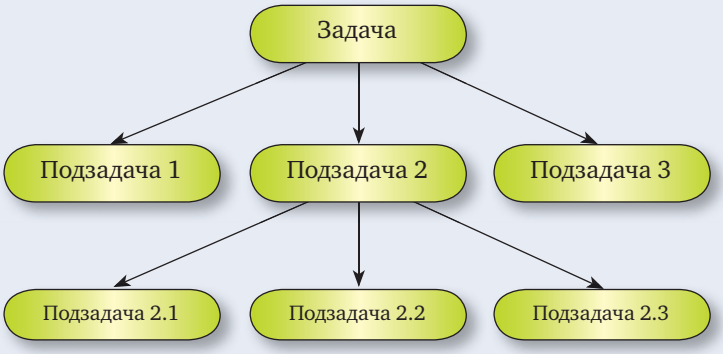
Что такое ООП?

Как вы знаете, работа первых компьютеров сводилась к вычислениям по заданным формулам различной сложности. Число переменных и массивов в программе было невелико, так что программист мог легко удерживать в памяти все взаимосвязи между ними и детали алгоритма.

С каждым годом производительность компьютеров росла, и человек “поручал” им все более и более трудоемкие задачи. Компьютеры следующих поколений стали использоваться для создания сложных информационных систем (например, банковских) и моделирования процессов, происходящих в реальном мире. Новые задачи требовали более сложных алгоритмов, объем программ вырос до сотен тысяч и даже миллионов строк, число переменных и массивов измерялось в тысячах.

Программисты столкнулись с проблемой сложности, которая превысила возможности человеческого разума. Один человек уже не способен написать надежно работающую серьезную программу, так как не может “охватить взором” все ее детали. Поэтому в разработке большинства современных программ принимает участие множество специалистов. При этом возникает новая проблема — нужно разделить работу между ними так, чтобы каждый мог работать независимо от других, а потом готовую программу можно было бы собрать вместе из готовых блоков, как из кубиков.

Как отмечал известный нидерландский программист Эдсгер Дийкстра, человечество еще в древности придумало способ управления сложными системами: “разделяй и властвуй”. Это означает, что исходную систему нужно разбить на подсистемы (выполнить *декомпозицию*) так, чтобы работу каждой из них можно было рассматривать и совершенствовать независимо от других.



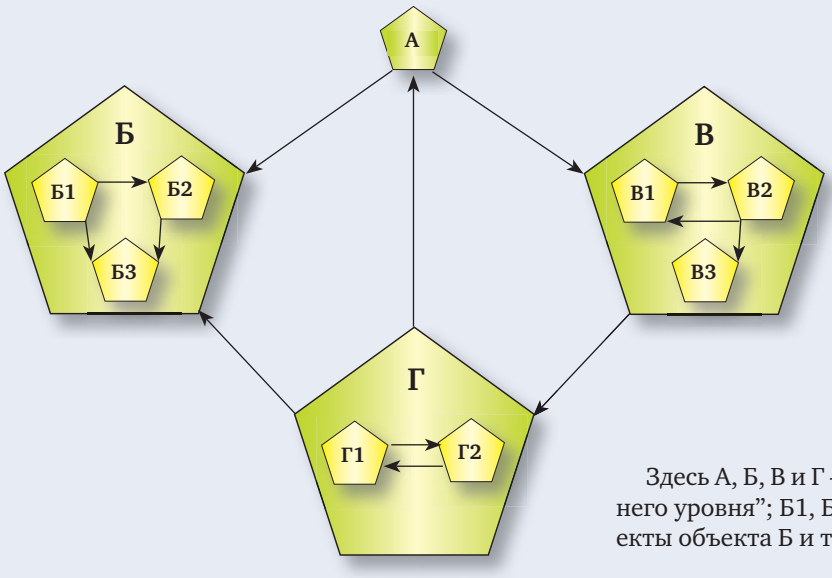
Для этого в классическом (процедурном) программировании используют метод проектирования “сверху вниз”: сложная задача разбивается на части (подзадачи и соответствующие им *алгоритмы*), которые затем снова разбиваются на более мелкие подзадачи и т.д. Однако при этом задачу “реального мира” приходится переформулировать, представляя все данные в виде переменных, массивов, списков и других структур данных. При моделировании больших систем объем этих данных увеличивается, они становятся плохо управляемыми, и это приводит к большому числу ошибок. Так как любой алгоритм может обратиться к любым глобальным (общедоступным) данным, повышается риск случайного недопустимого изменения каких-то значений.

В конце 60-х годов XX века появилась новая идея — применить в разработке программ тот подход, который использует человек в повседневной жизни. Люди воспринимают мир как множество *объектов* — предметов, животных, людей, — это отмечал еще в XVII веке французский математик и философ Рене Декарт. Все объекты имеют внутреннее устройство и состояние, свойства (внешние характеристики) и поведение. Чтобы справиться со сложностью окружающего мира, люди часто игнорируют многие свойства объектов, ограничиваясь лишь теми, которые необходимы для решения их практических задач. Такой прием называется *абстракцией*.

> **Абстракция** — это выделение существенных характеристик объекта, отличающих его от других объектов.

Для разных задач существенные свойства могут быть совершенно различные. Например, услышав слово “кошка”, многие подумают о пушистом усатом животном, которое мурлыкает, когда его гладят. В то же время ветеринарный врач представляет скелет, ткани и внутренние органы кошки, которую ему нужно лечить. В каждом из этих случаев применение абстракции дает разные модели одного и того же объекта, поскольку различны цели моделирования.

Как применить принцип абстракции в программировании? Поскольку формулировка задач, решаемых на компьютерах, все более приближается к формулировкам реальных жизненных задач, возникла такая идея: представить программу в виде множества объектов (моделей), каждый из которых обладает своими свойствами и поведением, но его внутреннее устройство скрыто от других объектов. Тогда решение задачи сводится к моделированию взаимодействия этих объектов. Построенная таким образом модель задачи называется *объектной*. Здесь тоже идет проектирова-



Здесь А, Б, В и Г — объекты “верхнего уровня”; Б1, Б2 и Б3 — подобъекты объекта Б и т.д.

ние “сверху вниз”, только не по алгоритмам (как в процедурном программировании), а *по объектам*. Если нарисовать схему такой декомпозиции, она представляет собой граф, так как каждый объект может обмениваться данными со всеми другими.

Для решения задачи “на верхнем уровне” достаточно определить, *что* делает тот или иной объект, не заботясь о том, *как* именно он это делает. Таким образом, для преодоления сложности мы используем *абстракцию*, то есть сознательно отбрасываем второстепенные детали.

Если построена объектная модель задачи (выделены объекты и определены правила обмена данными между ними), можно поручить разработку каждого из объектов отдельному программисту (или группе), который должен написать соответствующую часть программы, то есть определить, *как именно* объект выполняет свои функции. При этом конкретному разработчику не обязательно держать в голове полную информацию обо всех объектах, нужно лишь строго соблюдать соглашения о способе обмена данными (*интерфейсе*) “своего” объекта с другими.

Программирование, основанное на моделировании задачи реального мира как множества взаимодействующих объектов, принято называть объектно-ориентированным программированием (ООП). Более строгое определение мы дадим немного позже.

Контрольные вопросы

1. Почему со временем неизбежно изменяются методы программирования?
2. Что такое декомпозиция, зачем она применяется?
3. Что такое процедурное программирование? Какой вид декомпозиции в нем используется?
4. Какие проблемы в программировании привели к появлению ООП?
5. Что такое абстракция? Зачем она используется в обычной жизни?
6. Объясните, как связана абстракция с моделированием.
7. Какой вид декомпозиции используется в ООП?
8. Какие преимущества дает объектный подход в программировании?
9. Что такое интерфейс? Приведите примеры объектов, у которых одинаковый интерфейс и разное устройство.

Объекты и классы

Как мы увидели в предыдущем параграфе, для того чтобы построить объектную модель, нужно

- выделить взаимодействующие объекты, с помощью которых можно достаточно полно описать поведение моделируемой системы;
- определить их *свойства*, существенные в данной задаче;
- описать *поведение* (возможные действия) объектов, то есть команды, которые объекты могут выполнить.

Этап разработки модели, на котором решаются перечисленные выше задачи, называется *объектно-ориентированным анализом* (ООА). Он выполняется до того, как программисты напишут самую первую строчку кода, и во многом определяет качество и надежность будущей программы.

Рассмотрим объектно-ориентированный анализ на примере простой задачи. Пусть нам необходимо изучить движение автомобилей на шоссе, например, для того, чтобы определить, достаточна ли его пропускная способность. Как построить объектную модель этой задачи? Прежде всего нужно разобраться, что такое объект.

> **Объектом** можно назвать то, что имеет четкие границы и обладает *состоянием* и *поведением*.

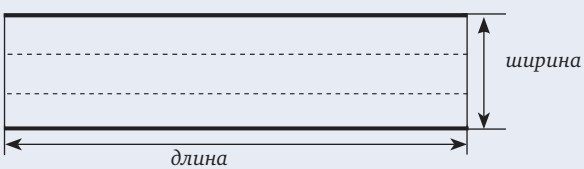
Состояние объекта определяет его возможное поведение. Например, лежачий человек не может прыгнуть, а незаряженное ружье не выстрелит.

В нашей задаче объекты — это дорога и двигающиеся по ней машины. Машин может быть несколько, причем все они с точки зрения нашей задачи имеют общие свойства. Поэтому нет смысла описывать отдельно каждую машину: достаточно один раз определить их общие черты, а потом просто сказать, что все машины ими обладают. В ООП для этой цели вводится специальный термин — *класс*.

> **Класс** — это множество объектов, имеющих общую структуру и общее поведение.

Например, в рассматриваемой задаче можно ввести два класса — *Дорога* и *Машина*. По условию дорога одна, а машин может быть много.

Будем рассматривать прямой отрезок дороги, в этом случае объект “дорога” имеет два свойства, важных для нашей задачи: длину и число полос движения. Эти свойства определяют *состояние* дороги. “*Поведение*” дороги может заключаться в том, что число полос меняется, например, из-за ремонта покрытия, но в нашей простейшей модели объект “дорога” не будет изменяться.



Дорога
длина ширина

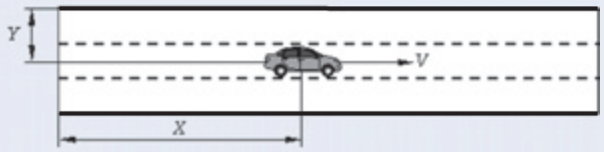
Схематично класс *Дорога* можно изобразить в виде прямоугольника с тремя секциями: в верхней записывают название *класса*, во второй части — свойства, а в третьей — возможные действия, которые называют *методами*. В нашей модели дороги два свойства и ни одного метода.

Теперь рассмотрим объекты класса *Машина*. Их важнейшие свойства — координаты и скорость движения. Для упрощения будем считать, что

- все машины одинаковы;
- все машины движутся по дороге слева направо с постоянной скоростью (скорости разных машин могут быть различны);
- по каждой полосе движения едет только одна машина, так что можно не учитывать обгон и переход на другую полосу;
- если машина выходит за правую границу дороги, вместо нее слева на той же полосе появляется новая машина.

Не все эти допущения выглядят естественно, но такая простая модель позволит понять основные принципы метода.

За координаты машины можно принять расстояние X от левого края рассматриваемого участка шоссе и номер полосы Y (натуральное число). Скорость автомобиля V в нашей модели — неотрицательная величина.

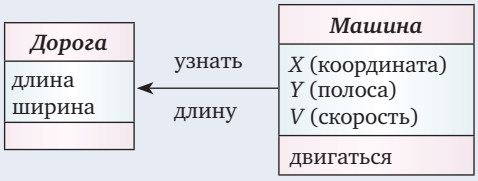


Машина
X (координата) *Y* (полоса) *V* (скорость)
двигаться

Теперь рассмотрим *поведение* машины. В данной модели она может выполнять всего одну команду — ехать в заданном направлении (назовем ее “двигаться”). Говорят, что объекты класса *Машина* имеют метод “двигаться”.

Метод — это процедура или функция, принадлежащая классу объектов.

Пока мы построили только модели отдельных объектов (точнее, классов). Чтобы моделировать всю систему, нужно разобраться, как эти объекты взаимодействуют. Объект-машина должен уметь “определить”, что закончился рассматриваемый участок дороги. Для этого машина должна обращаться к объекту “дорога”, запрашивая длину дороги (см. стрелку на схеме).



Такая схема определяет

- свойства объектов;
- операции, которые они могут выполнять;
- связи (обмен данными) между объектами.

В то же время мы пока ничего не говорили о том, *как* устроены объекты и *как* именно они будут выполнять эти операции. Согласно принципам ООП, ни один объект не должен зависеть от внутреннего устройства и алгоритмов работы других объектов. Поэтому, построив такую схему, можно поручить разработку двух классов объектов двум программистам, каждый из которых может решать свою задачу независимо от других. Важно только, чтобы все они четко соблюдали *интерфейс* — правила, описывающие взаимодействие “своих” объектов с остальными.

Контрольные вопросы

1. Какие этапы входят в объектно-ориентированный анализ?
2. Что такое объект?
3. Что такое класс? Чем отличаются понятия “класс” и “объект”?
4. Что такое метод?
5. Как изображаются классы на диаграмме?
6. Почему при объектно-ориентированном анализе не уточняют, как именно объекты будут устроены и как они будут решать свои задачи?

Задачи

1. Добавьте в рассмотренную модель светофоры (на дороге их может быть много). Подумайте, какие свойства и методы должны быть у объектов класса *Светофор*. Как могут быть связаны классы *Дорога*, *Светофор* и *Машина* (сравните разные варианты)?
2. Придумайте свою задачу и выполните ее объектно-ориентированный анализ. Примеры: моделирование работы магазина, банка, библиотеки и т.п.

Создание объектов в программе

Объект Дорога

Объектно-ориентированная программа начинается с описания *классов* объектов. В нашей программе самый простой класс — это *Дорога*. Объекты этого класса имеют два свойства: длину (англ. *length*), которая может быть вещественным числом, и ширину (англ. *width*) — количество полос, целое число. Для хранения значений свойств используются переменные, принадлежащие объекту, которые называются *полями*.

Поле — это переменная, принадлежащая объекту.

Значения полей описывают *состояние* объекта (а методы — его *поведение*).

Описание класса *Дорога* в программе на объектной версии Паскаля выглядит так:

```
type TRoad = class
       Length: single;
       Width: integer;
     end;
```

Эти строчки вводят новый тип данных — класс TRoad[1], то есть сообщают компилятору, что в программе, возможно, будут использоваться объекты этого типа. При этом в памяти не создается ни одного объекта. Это описание — как чертеж, по которому в нужный момент можно построить сколько угодно таких объектов.

Если мы хотим работать с объектом класса TRoad, в программе нужно объявить соответствующую переменную:

```
var road: TRoad;
```

Однако и это еще не объект, а ссылка (указатель), то есть переменная, в которой можно сохранить адрес любого объекта класса TRoad. Чтобы создать сам объект в памяти, нужно вызвать специальный метод Create, который называется *конструктором*. Адрес нового объекта записываем в переменную road:

```
road := TRoad.Create;
```

Созданный объект относится к классу TRoad, поэтому его называют *экземпляром* класса TRoad.

При описании класса мы ничего не говорили о методе Create. Он добавляется ко всем классам по умолчанию, при его вызове все переменные объекта заполняются нулями[2].

> **Конструктор** — это метод класса, который вызывается для создания объекта этого класса.

Свойства дороги можно изменить с помощью точечной нотации, с которой вы познакомились, работая с записями (структурами):

```
road.Length := 60;
road.Width := 3;
```

Полная программа, которая создает объект “дорога” (и больше ничего не делает), выглядит так (*Free Pascal*):

```
{$mode objfpc}
type TRoad = class
       Length: single;
       Width: integer;
     end;
var road: TRoad;
begin
  road := TRoad.Create;
  road.Length := 60;
  road.Width := 3;
end.
```

Начальные значения полей можно задавать прямо при создании объекта. Для этого нужно добавить в описание класса новый *конструктор*. Конструктору будет передаваться два параметра — начальные значения длины и ширины дороги:

```
type TRoad = class
       Length: single;
       Width: integer;
       constructor Create(length0: single;
                          width0: integer);
     end;
```

Реализация (программа) конструктора может выглядеть так:

```
constructor TRoad.Create(length0: single;
                         width0: integer);
begin
  if length0 > 0 then
       Length := length0
  else Length := 100;
  if width0 > 0 then
       Width := width0
  else Width := 1;
end;
```

Здесь проверяется правильность переданных параметров, чтобы по ошибке длина и ширина дороги не оказались нулевыми или отрицательными. Теперь создавать объект будет проще:

```
road := TRoad.Create(60, 3);
```

Длина этой дороги — 60 единиц, она содержит 3 полосы.

Таким образом, класс выполняет роль “фабрики”, которая “выпускает” (создает) объекты “по чертежу” (описанию класса) при вызове конструктора.

Объект Машина

Теперь можно описать класс *Машина* (в программе назовем его TCar). Объекты класса TCar имеют три свойства и один метод — процедуру move. Координата X и скорость V — это вещественные значения, а номер полосы Y — целое.

```
type TCar = class
       X, V: single;
       Y: integer;
       road: TRoad;
       procedure move;
       constructor Create(road: TRoad;
                          y0: integer;
                          v0: single);
     end;
```

Так как объекты-машины должны обращаться к объекту “дорога”, в область данных включено дополнительное поле road. Конечно, это не значит, что в состав машины входит дорога. Напомним, что это только ссылка, и сразу после создания объекта-машины нужно записать в нее адрес заранее созданного объекта “дорога”. Эту привязку удобно сделать прямо в конструкторе, при создании объекта. Заодно мы определяем полосу движения и скорость, а начальная координата X автоматически устанавливается в ноль:

[1] Буква T в начале названия класса — это сокращение от слова *type*.

[2] Так сделано во всех объектных реализациях Паскаля.

```
constructor TCar.Create(road0: TRoad;
                        y0: integer;
                        v0: single);
begin
  road := road0; Y := y0; V := v0;
end;
```

Теперь займемся *реализацией* (программированием) метода `move` (“двигаться”). В этом методе нужно вычислить новую координату `X` машины и, если она находится за пределами дороги, установить ее в ноль (машина появляется слева на той же полосе). Изменение координаты при равномерном движении описывается формулой

$$X = X_0 + V \cdot \Delta t,$$

где X_0 и X — начальная и конечная координаты, V — скорость, а Δt — время движения. Вспомним, что любое моделирование физических процессов на компьютере происходит в дискретном времени, с некоторым интервалом дискретизации. Для простоты можно измерять время в этих интервалах, а за скорость V принять расстояние, проходимое машиной за один интервал. Тогда метод `move`, описывающий изменение положения машины за один интервал ($\Delta t = 1$), может выглядеть так:

```
procedure TCar.move;
begin
  X := X + V;
  if X > road.Length then X := 0;
end;
```

Основная программа

В основной программе объявим массив объектов-машин:

```
const N = 3;
var cars: array [1..N] of TCar;
```

Как вы помните, это еще не объекты, а ссылки — переменные, в которые можно записать адреса объектов класса `TCar`. Теперь нужно создать сами объекты:

```
var i: integer;
...
for i := 1 to N do
  cars[i] := TCar.Create(road, i, 2. * i);
```

При вызове конструктора задаются три параметра: адрес объекта “дорога” (его нужно создать до выполнения этого цикла), номер полосы и скорость. В приведенном варианте машина на полосе с номером `i` идет со скоростью `2i` единиц за один интервал моделирования.

Сам цикл моделирования получается очень простой: на каждом шаге вызывается метод `move` для каждой машины:

```
repeat
  for i := 1 to N do cars[i].move;
until keypressed;
```

Полностью основная программа выглядит так:

```
const N = 3;
var road: TRoad;
    cars: array [1..N] of TCar;
      i: integer;
begin
  road := TRoad.Create(60, N);
  for i := 1 to N do
    cars[i] := TCar.Create(road, i, 2. * i);
  repeat
    for i := 1 to N do cars[i].move;
  until keypressed;
end.
```

Можно ли было написать такую же программу, не используя объекты? Конечно, да. И она получилась бы короче, чем наш объектный вариант (с учетом описания классов). В чем же преимущества ООП? Мы уже отмечали, что ООП — это средство разработки больших программ, моделирующих работу сложных систем. В этом случае очень важно, что при использовании объектного подхода

- основная программа, описывающая решение задачи в целом, получается простой и понятной; все команды напоминают действия в реальном мире (“машина № 2, вперед!”);
- разработку отдельных классов объектов можно поручить разным программистам, при этом каждый может работать независимо от других;
- если объекты *Дорога* и *Машина* понадобятся в других разработках, можно будет легко использовать уже готовые классы.

Контрольные вопросы

1. Что такое поле?
2. Как объявляется класс объектов в программе?
3. Как объявляется переменная для работы с объектом некоторого класса? Что в ней хранится?
4. Как в памяти создается экземпляр класса (объект)?
5. Что такое конструктор?
6. Что такое точечная нотация? Как она используется при работе с объектами?
7. Как можно задать начальные значения для полей объекта?
8. Почему в методе `TCar.move` не объявлены переменные `X` и `V`?
9. Сравните преимущества и недостатки решения рассмотренной задачи “классическим” способом и с помощью ООП. Сделайте выводы.

Задачи

1. Добавьте в программу операторы, позволяющие изобразить на экране перемещение машин (в текстовом или графическом режиме). Подумайте, какие методы можно добавить для этого в класс `TCar`.
2. *Добавьте в модель светофор, который переключается автоматически по программе (например, 5 с горит красный свет, затем 1 с — желтый, потом 5 с — зеленый и т.д.). Измените классы так, чтобы машина запрашивала у объекта *Дорога* ме-

стоположение ближайшего светофора, а затем обращалась к светофору, для того чтобы узнать, какой сигнал горит. Машины должны останавливаться у светофора с запрещающим сигналом.

Скрытие внутреннего устройства

Во время построения объектной модели задачи мы выделили отдельные объекты, которые для обмена данными друг с другом используют *интерфейс* — внешние свойства и методы. При этом все внутренние данные и детали внутреннего устройства объекта должны быть скрыты от "внешнего мира". Такой подход позволяет

- обезопасить внутренние данные (поля) объекта от изменений (возможно, разрушительных) со стороны других объектов;
- проверять данные, поступающие от других объектов на корректность, тем самым повышая надежность программы;
- переделывать внутреннюю структуру и код объекта любым способом, не меняя его внешние характеристики (интерфейс); при этом никакой переделки других объектов не требуется.

> Скрытие внутреннего устройства объектов называют **инкапсуляцией** ("помещение в капсулу").

Разберем простой пример. Во многих системах программирования есть класс, описывающий свойства "пера", которое используется при рисовании линий в графическом режиме. Назовем этот класс `TPen`, в простейшем варианте он будет содержать только одно поле `color`, которое определяет цвет. Будем хранить код цвета в виде символьной строки, в которой записан шестнадцатеричный код составляющих модели RGB. Например, `'FF00FF'` — это фиолетовый цвет, потому что красная (R) и синяя (B) составляющие равны $FF_{16} = 255$, а зеленой составляющей нет вообще. Класс можно объявить так:

```
type TPen = class
       color: string;
     end;
```

По умолчанию все члены класса (поля и методы) открытые, общедоступные (англ. *public*). Те элементы, которые нужно скрыть, в описании класса помещают в "частный" раздел (англ. *private*), например, так:

```
type TPen = class
     private
       FColor: string;
     end;
```

В этом примере поле `FColor` закрытое. Имена всех закрытых полей далее будем начинать с буквы `F` (от англ. *field*, поле). К закрытым полям нельзя обратиться извне (это могут делать только методы самого объекта), поэтому теперь невозможно не только изменить внутренние данные объекта, но и просто узнать их значения. Чтобы решить эту проблему, нужно добавить к классу еще два метода: один из них будет возвращать текущее значение поля `FColor`, а второй — присваивать полю новое значение. Эти методы доступа назовем `getColor` (англ. *получить Color*) и `setColor` (англ. *установить Color*):

```
type TPen = class
     private
       FColor: string;
     public
       function getColor: string;
       procedure
            setColor(newColor: string);
      end;
```

Обратите внимание, что оба метода находятся в секции `public` (общедоступные).

Что же улучшилось в сравнении с первым вариантом (когда поле было открытым)? Согласно принципам ООП, внутренние поля объекта должны быть доступны *только* с помощью методов. В этом случае внутреннее представление данных может как угодно отличаться от того, как другие объекты "видят" эти данные. Например, может быть, что для других объектов удобнее использовать кодирование цвета, принятое в языке HTML, где код начинается с символа `'#'`. В этом случае метод `getColor` можно написать так:

```
function TPen.getColor: string;
begin
  Result := '#' + FColor;
end;
```

С другой стороны, в методе `setColor` мы можем обрабатывать ошибки, не разрешая присваивать полю недопустимые значения. Например, установим, что код цвета, передаваемый нашему объекту, должен начинаться с символа `'#'` и состоять из семи символов. Если эти условия не выполняются, будем записывать в поле `FColor` код черного цвета `'000000'`. Если ошибки нет, в поле записывается код переданного цвета без знака `'#'`:

```
procedure
    TPen.setColor(newColor: string);
begin
  if (Length(newColor) <> 7) or
     (newColor[1] <> '#') then
       FColor := '000000'
       { если ошибка, то черный цвет }
  else FColor := Copy(newColor, 2, 6);
end;
```

Теперь, если `pen` — это объект класса `TPen`, то для установки и чтения его цвета нужно использовать показанные выше методы:

```
pen.setColor ( '#FFFF00' );
{ изменение цвета }
writeln( 'цвет пера: ', pen.getColor );
{ получение цвета }
```

Итак, мы скрыли внутренние данные, но одновременно обращение к свойствам стало выглядеть довольно неуклюже: вместо `pen.color:='#FFFF00'`

теперь нужно писать pen.setColor('#FFFF00'). Чтобы упростить запись, во многие объектно-ориентированные языки программирования ввели понятие свойства (англ. *property*), которое внешне выглядит как переменная объекта, но на самом деле при записи и чтении свойства вызываются методы объекта.

> **Свойство** — это способ доступа к внутреннему состоянию объекта, имитирующий обращение к его внутренней переменной.

Свойство color в нашем случае можно определить так:

```
type TPen = class
     private
       FColor: string;
       function getColor: string;
       procedure
          setColor(newColor: string);
     public
       property color: string read
       getColor write setColor;
     end;
```

Здесь методы getColor и setColor перенесены в раздел private, то есть закрыты от других объектов. Однако есть общедоступное свойство color строкового типа:

```
property color: string read
getColor write setColor;
```

При чтении этого свойства (англ. *read*) вызывается метод getColor, а при записи нового значения (англ. *write*) — метод setColor. В программе можно использовать это свойство так:

```
pen.color := '#FFFF00';
{ изменение цвета }
writeln( 'цвет пера: ', pen.color );
{ получение цвета }
```

Таким образом, с помощью свойства color другие объекты могут изменять и читать цвет объектов класса TPen. Для обмена данными с "внешним миром" важно лишь то, что свойство color — символьного типа, и оно содержит 7-символьный код цвета, начинающийся со знака '#'. При этом внутреннее устройство объектов TPen может быть любым, и его можно менять как угодно. Покажем это на примере.

Хранение цвета в виде символьной строки неэкономно и неудобно, так как большинство стандартных функций используют числовые коды цвета. Поэтому лучше хранить код цвета как целое число и поле FColor сделать целого типа:

```
FColor: integer;
```

При этом необходимо поменять методы getColor и setColor, которые непосредственно работают с этим полем:

```
function TPen.getColor: string;
begin
  Result := '#' + IntToHex(FColor,6);
end;
procedure
TPen.setColor(newColor: string);
begin
  if (Length(newColor) <> 7) or
     (newColor[1] <> '#') then
    FColor := 0
   { если ошибка, то черный цвет }
  else begin
    newColor[1] := '$';
   { шестнадцатеричное число}
    FColor := StrToInt(newColor);
  end;
end;
```

Для перевода числового кода в символьную запись используется функция IntToHex, входящая в библиотеку Lazarus. Ее второй параметр — количество цифр, которое будет в шестнадцатеричном числе. Обратный перевод выполняет функция StrToInt. Для того чтобы указать, что число записано в шестнадцатеричной системе, перед ним записывают символ '$' (фактически первый символ переданной строки '#' заменяется на '$').

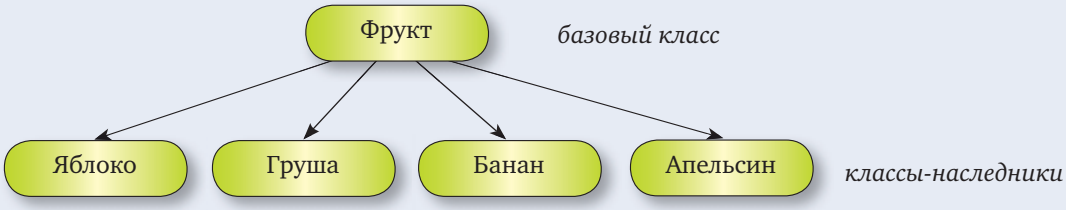
В этом примере мы принципиально изменили внутреннее устройство объекта — заменили строковое поле на целочисленное. Однако другие объекты даже не "догадаются" о такой замене, потому что сохранился интерфейс — свойство color по-прежнему имеет строковый тип. Таким образом, инкапсуляция позволяет как угодно изменять внутреннее устройство объектов, не затрагивая интерфейс. При этом все остальные объекты изменять не требуется.

Иногда не нужно разрешать другим объектам менять свойство, то есть требуется сделать свойство "только для чтения" (англ. *read-only*). Пусть, например, мы строим программную модель автомобиля. Как правило, другие объекты не могут непосредственно менять его скорость, однако могут получить информацию о ней — "прочитать" значение скорости. При описании такого свойства слово write и название метода записи не указывают вообще:

```
type TCar = class
     private
       Fv: single;
       ...
     public
       property v: single read Fv;
       ...
     end;
```

Таким образом, доступ к внутренним данным объекта возможен, как правило, только с помощью методов. Применение свойств (*property*) очень удобно, потому что позволяет использовать ту же форму записи, что и при работе с общедоступной переменной объекта.

При использовании скрытия данных длина программы чаще всего увеличивается, однако мы по-



лучаем и важные преимущества. Код, связанный с объектом, разделен на две части: общедоступную часть (секция `public`) и закрытую (`private`). Их можно сравнить с надводной и подводной частями айсберга.

Объект взаимодействует с другими объектами только с помощью своих общедоступных свойств и методов (интерфейс). Поэтому при сохранении интерфейса можно как угодно менять внутреннюю структуру данных и код методов, и это никак не будет влиять на другие объекты. Подчеркнем, что все это становится действительно важно, когда разрабатывается большая программа и необходимо обеспечить ее надежность.

Контрольные вопросы

1. Что такое “интерфейс объекта”?
2. Что такое инкапсуляция? Каковы ее цели?
3. Чем отличаются секции `public` и `private` в описании классов? Как определить, в какую из них поместить свойство или метод?
4. Почему рекомендуют делать доступ к полям объекта только с помощью методов?
5. Что такое свойство? Зачем во многие языки программирования введено это понятие?
6. Можно ли с помощью свойства обращаться напрямую к полю объекта, не используя метод?
7. Почему методы доступа, которые использует свойство, делают закрытыми?
8. Зачем нужны свойства “только для чтения”? Приведите примеры.
9. Подумайте, в каких ситуациях может быть нужно свойство “только для записи” (которое нельзя прочитать). Как ввести такое свойство в описание класса?

Задачи

1. Измените построенную ранее программу моделирования движения так, чтобы все поля у объектов были закрытыми. Используйте свойства для доступа к данным.

Иерархия классов

Классификации

Как в науке, так и в быту важную роль играет *классификация* — разделение изучаемых объектов на группы (классы), объединенные общими признаками. Прежде всего это нужно для того, чтобы не запутаться в большом количестве данных и не описывать каждый объект заново.

Например, есть много видов фруктов[3] (яблоки, груши, бананы, апельсины и т.д.), но все они обладают некоторыми общими свойствами. Если перевести этот пример на язык ООП, класс *Яблоко* — это подкласс (производный класс, класс-наследник, потомок) класса *Фрукт*, а класс *Фрукт* — это базовый класс (суперкласс, класс-предок) для класса *Яблоко* (а также для классов *Груша*, *Банан, Апельсин* и других).

Классический пример научной классификации — классификация животных или растений. Как вы знаете, она представляет собой *иерархию* (многоуровневую структуру). Например, горный клевер относится к роду *Клевер* семейства *Бобовые* класса *Двудольные* и т.д. Говоря на языке ООП, класс *Горный клевер* — это наследник класса *Клевер*, а тот, в свою очередь, наследник класса *Бобовые*, который также является наследником класса *Двудольные* и т.д.

> Класс Б является наследником класса А, если можно сказать, что Б — это разновидность А.

Например, можно сказать, что яблоко — это фрукт, а горный клевер — одно из растений семейства *Двудольные*. В то же время мы не можем сказать, что “машина — это разновидность двигателя”, поэтому класс *Машина* не является наследником класса *Двигатель*. Двигатель — это составная часть машины, поэтому объект класса *Машина* содержит в себе объект класса *Двигатель*. Отношения между двигателем и машиной — это отношение “часть — целое”.

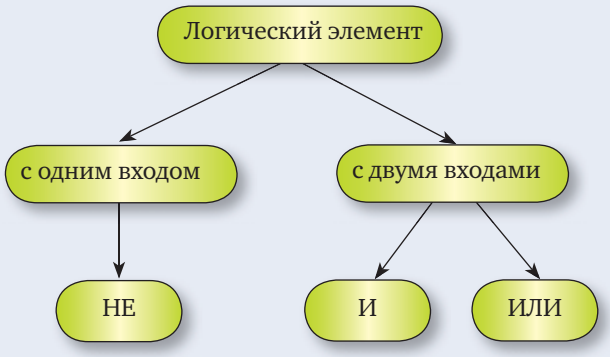
Иерархия логических элементов

Рассмотрим такую задачу: составить программу для моделирования управляющих схем, построенных на логических элементах (см. главу 3 в учебнике для 10-го класса). Нам нужно “собрать” заданную схему и построить ее таблицу истинности.

[3] Фруктами называют сочные съедобные плоды деревьев и кустарников.

Как вы уже знаете, перед тем как программировать, нужно выполнить объектно-ориентированный анализ. Все объекты, из которых состоит схема, — это логические элементы, однако они могут быть разными (“НЕ”, “И”, “ИЛИ” и другие). Попробуем выделить общие свойства и методы всех логических элементов.

Ограничимся только элементами, у которых один или два входа. Тогда иерархия классов может выглядеть так:



Среди всех элементов с двумя входами мы показали только элементы “И” и “ИЛИ”, остальные вы можете добавить самостоятельно.

Итак, для того чтобы не описывать несколько раз одно и то же, классы в программе должны быть построены в виде иерархии. Теперь можно дать классическое определение объектно-ориентированного программирования:

Объектно-ориентированное программирование — это такой подход к программированию, при котором программа представляет собой множество взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Базовый класс

Построим первый вариант описания класса *Логический элемент* (`TLogElement`). Обозначим его входы как `In1` и `In2`, а выход назовем `Res` (от англ. *result* — результат). Любой логический элемент должен уметь вычислять значение выхода по известным входам, для этого введем метод `calc`:

ЛогЭлемент
In1 (вход 1) In2 (вход 2) Res (результат)
calc

```
type
  TLogElement = class
    In1, In2: boolean;
    Res: boolean;
    procedure calc;
  end;
```

В таком варианте все данные открытые (общедоступные). Чтобы защитить внутреннее устройство объекта, скроем внутренние поля (добавив в их название первую букву F) и введем свойства:

```
type
  TLogElement = class
  private
    FIn1, FIn2: boolean;
    FRes: boolean;
    procedure setIn1(newIn1: boolean);
    procedure setIn2(newIn2: boolean);
    procedure calc;
  public
    property In1: boolean read FIn1
       write setIn1;
    property In2: boolean read FIn2
       write setIn2;
    property Res: boolean read FRes;
  end;
```

Обратите внимание, что свойство `Res` — это свойство только для чтения, и другие объекты не могут его менять. Кроме того, мы поместили процедуру `calc` в скрытый раздел (`private`), потому что пересчет результата должен выполняться автоматически при изменении любого входного сигнала (другие объекты не должны об этом беспокоиться).

Несложно написать процедуру `setIn1` (и аналогичную ей процедуру `setIn2`), в ней новое входное значение присваивается полю и сразу пересчитывается результат:

```
procedure TLogElement.setIn1(newIn1:
          boolean);
begin
  FIn1 := newIn1;
  calc;
end;
```

Если внимательно проанализировать построенное описание класса, можно выявить несколько проблем. Во-первых, элемент “НЕ” имеет только один вход, поэтому не хотелось бы для него открывать доступ к свойству `In2` (это не нужно и может привести к ошибкам).

Во-вторых, процедуру `calc` *невозможно написать*, пока мы не знаем, какой именно логический элемент моделируется. С другой стороны, мы знаем, что такую процедуру имеет любой логический элемент, то есть она должна принадлежать именно классу `TLogElement`. Здесь можно написать процедуру-“заглушку” (которая ничего не делает):

```
procedure TLogElement.calc;
begin
end;
```

Но нужно как-то дать возможность классам-наследникам изменить тот метод так, чтобы он выполнял нужную операцию. Такой метод называется *виртуальным*.

Более точное определение этого понятия мы дадим несколько позже.

Получается, что классы-наследники могут по-разному реализовать один и тот же метод. Такая возможность называется *полиморфизм*.

Полиморфизм (от греч. *πολυ* — много и *μορφη* — форма) — это возможность классов-наследников по-разному реализовать метод, описанный для класса-предка.

Мы уже говорили о том, что виртуальный метод `calc` не нужно делать общедоступным (`public`). В то же время его нельзя делать закрытым (`private`), потому что в этом случае он не будет доступен классам-наследникам. В таких случаях в описании класса используется третий блок (кроме `private` и `public`), который называется `protected` (защищенный). Данные и методы в этом блоке доступны для классов-наследников, но недоступны для других классов. В этот же блок `protected` мы переместим и объявление свойства `In2` — оно будет скрыто для элемента "НЕ", а элементы с двумя входами его "откроют" (чуть позже).

```
type
  TLogElement = class
  private
    FIn1, FIn2: boolean;
    FRes: boolean;
    procedure setIn1(newIn1: boolean);
    procedure setIn2(newIn2: boolean);
  protected
    property In2: boolean read FIn2
     write setIn2;
    procedure calc; virtual; abstract;
  public
    property In1: boolean read FIn1
    write setIn1;
    property Res: boolean read FRes;
  end;
```

Обратите внимание на объявление метода `calc`: после него стоят слова `virtual` (виртуальный) и `abstract` (англ. *абстрактный*). Описатель `virtual` говорит о том, что метод `calc` — виртуальный и классы-наследники могут его переопределять. Как уже отмечалось, мы должны объявить этот метод (ввести его в описание класса), поскольку он должен быть у любого логического элемента. С другой стороны, *невозможно* написать процедуру `calc`, пока неизвестен тип логического элемента. Такой метод называется *абстрактным* и обозначается описателем `abstract`. Для абстрактного метода не нужно ставить "заглушку".

Абстрактный метод — это метод класса, который объявляется, но не реализуется в классе.

Чтобы не писать процедуру-заглушку, нужно объявить метод не только виртуальным, но и абстрактным, добавив в конец описания слово `abstract`:

```
procedure calc; virtual; abstract;
```

Более того, *не существует* логического элемента "вообще", как не существует "просто фрукта", не относящегося к какому-то виду. Такой класс в ООП называется абстрактным. Его отличительная черта — хотя бы один абстрактный (нереализованный) метод.

Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод.

Итак, полученный класс `TLogElement` — это абстрактный класс (компилятор определит это автоматически). Его можно использовать только для разработки классов-наследников, создать в программе объект этого класса нельзя.

Чтобы класс-наследник не был абстрактным, он должен переопределить все абстрактные методы предка, в данном случае — метод `calc`. Как это сделать, вы увидите в следующем пункте.

Классы-наследники

Теперь займемся классами-наследниками от `TLogElement`. Поскольку у нас будет единственный элемент с одним входом ("НЕ"), сделаем его наследником прямо от `TLogElement` (не будем вводить специальный класс "элемент с одним входом").

```
type
  TNot = class(TLogElement)
    procedure calc; override;
  end;
```

После слова `class` в скобках указано название базового класса. Все объекты класса `TNot` обладают всеми свойствами и методами класса `TLogElement`.

Новый класс *переопределяет* метод `calc`, на это указывает слово `override` (англ. "перекрыть"). Заметим, что у базового класса `TLogElement` этого метода нет вообще — он абстрактный, поэтому в данном случае мы фактически программируем метод, объявленный в базовом классе. Для элемента "НЕ" он выглядит очень просто:

```
procedure TNot.calc;
begin
  FRes := not FIn1;
end;
```

Класс `TNot` уже не абстрактный, потому абстрактный метод предка переопределен и теперь известно, что делать при вызове метода `calc`. Поэтому можно создавать объект этого класса и использовать его:

```
var n: TNot;
...
n := TNot.Create;
n.In1 := False;
writeln(n.Res);
```

Остальные элементы имеют два входа и будут наследниками класса

```
TLog2In = class(TLogElement)
public
  property In2;
end;
```

Единственное, что делает этот класс, — переводит свойство `In2` в раздел `public`, то есть дела-

ет его общедоступным. Отметим, что видимость можно только повышать, то есть нельзя, например, в наследнике сделать общедоступное свойство класса-предка закрытым или защищенным.

Класс TLog2In — это тоже абстрактный класс, потому что он не переопределил метод calc. Это сделают его наследники TAnd (элемент "И") и TOr (элемент "ИЛИ"), которые определяют конкретные логические элементы:

```
type
  TAnd = class(TLog2In)
    procedure calc; override;
  end;
  TOr = class(TLog2In)
    procedure calc; override;
  end;
```

Реализация переопределенного метода calc для элемента "И" выглядит так:

```
procedure TAnd.calc;
begin
  FRes := FIn1 and FIn2;
end;
```

Для элемента "ИЛИ" этот метод определяется аналогично.

Обратим внимание на метод setIn1, введенный в базовом классе:

```
procedure TLogElement.setIn1(newIn1:
boolean);
begin
  FIn1 := newIn1;
  calc;
end;
```

В нем вызывается метод calc, который пересчитывает значение на выходе логического элемента при изменении входа. Какой же метод будет вызван, если в базовом классе TLogElement он только объявлен, но не реализован?

Проблема в том, что для вызова любой процедуры нужно знать ее адрес в памяти. Для обычных методов транслятор сразу записывает в машинный код нужный адрес, потому что он заранее известен. Это так называемое *статическое* связывание (на этапе трансляции), при выполнении программы этот адрес не меняется. В нашем случае адрес метода неизвестен: в классе TLogElement его нет вообще, а у каждого класса-наследника адрес метода calc — свой собственный. Чтобы выйти из положения, используется *динамическое связывание*, то есть адрес вызываемой процедуры определяется при *выполнении* программы, когда уже определен тип объекта, с которым мы работаем. Такой метод нужно объявлять *виртуальным*, что мы и сделали ранее. Это означает не только то, что его могут переопределять наследники, но и то, что будет использоваться динамическое связывание. Теперь можно дать полное определение виртуального метода.

> **Виртуальный метод** — это метод базового класса, который могут переопределить классы-наследники так, что конкретный адрес вызываемого метода определяется только при выполнении программы.

Теперь мы готовы к тому, чтобы создавать и использовать построенные логические элементы. Например, таблицу истинности для последовательного соединения элементов "И" и "НЕ" можно построить так:

```
var elNot: TNot;
    elAnd: TAnd;
    A, B: boolean;
begin
  elNot := TNot.Create;
  elAnd := TAnd.Create;
  writeln('| A | B | not(A&B) ');
  writeln('------------------');
  for A := False to True do begin
    elAnd.In1 := A;
    for B := False to True do begin
      elAnd.In2 := B;
      elNot.In1 := elAnd.res;
      writeln('| ', integer(A), ' | ',
                   integer(B),
              ' | ', integer(elNot.res));
    end;
  end;
  readln;
end.
```

Сначала создаются два объекта — логические элементы "НЕ" (класс TNot) и "ИЛИ" (класс TAnd). Далее в двойном цикле перебираются все возможные значения логических переменных A и B, они подаются на входы элемента "И", а его выход — на вход элемента "НЕ". Чтобы при выводе логических значений вместо False и True выводились более компактные обозначения 0 и 1, значения входов и выхода преобразуются к целому типу (integer).

Модульность

При разработке больших программ нужно разделить работу между программистами так, чтобы каждый делал свой независимый блок (*модуль*). Для того чтобы разбить программу на модули, нужно выделить внутренне связные, но слабо связанные между собой модули.

В нашей программе с логическими элементами в отдельный модуль можно вынести все, что относится к логическим элементам.

Модуль в языке Паскаль в отличие от основной программы начинается со слова unit, после которого ставится название модуля.

```
unit log_elem;
interface
  …
implementation
  …
end.
```

В модуле два основных раздела: interface (интерфейс, общедоступная часть) и implementation (реализация, недоступная другим модулям). В разделе interface обычно размещают объяв-

ления классов, функций и процедур, а в разделе implementation — программный код. Модуль, содержащий классы логических элементов, на объектной версии языка Паскаль можно записать так:

```
unit log_elem;
{$mode objfpc}
interface
type
  TLogElement = class
  private
    FIn1, FIn2: boolean;
    FRes: boolean;
    procedure setIn1(newIn1: boolean);
    procedure setIn2(newIn2: boolean);
  protected
    procedure calc; virtual; abstract;
    property In2: boolean read FIn2
                  write setIn2;
  public
    property In1: boolean read FIn1
                write setIn1;
    property Res: boolean read FRes;
  end;
  TNot = class(TLogElement)
    procedure calc; override;
  end;
  TLog2In = class(TLogElement)
    property In2;
  end;
  TAnd = class(TLog2In)
    procedure calc; override;
  end;
  TOr = class(TLog2In)
    procedure calc; override;
  end;
implementation
  procedure TLogElement.setIn1(newIn1:
            boolean);
  begin
    FIn1 := newIn1; calc;
  end;
  procedure TLogElement.setIn2(newIn2:
            boolean);
  begin
    FIn2 := newIn2; calc;
  end;
  procedure TNot.calc;
  begin
    FRes := not FIn1;
  end;
  procedure TAnd.calc;
  begin
    FRes := FIn1 and FIn2;
  end;
  procedure TOr.calc;
  begin
    FRes := FIn1 or FIn2;
  end;
end.
```

Чтобы использовать такой модуль, нужно подключить его в основной программе с помощью ключевого слова uses, после которого через запятую перечисляются все используемые модули:

```
program logic;
{$mode objfpc}
uses log_elem;
var elNot: TNot;
    elAnd: TAnd;
    ...
begin
  elNot := TNot.Create;
  elAnd := TAnd.Create;
  ...
end.
```

Разделение программы на модули облегчает понимание и совершенствование программы, потому что каждый модуль можно разрабатывать, изучать и оптимизировать независимо от других.

Сообщения между объектами

Когда логические элементы объединяются в сложную схему, желательно, чтобы передача сигналов между ними при изменении входных данных происходила автоматически. Для этого можно немного расширить базовый класс TLogElement, чтобы элементы могли передавать друг другу сообщения об изменении своего выхода.

Будем считать, что выход любого логического элемента может быть подключен к любому (но только одному!) входу другого логического элемента. Добавим к описанию класса два поля и один метод:

```
type
  TLogElement = class
  private
    FNextEl: TLogElement;
    FNextIn: integer;
    ...
  public
    procedure Link(nextElement:
    TLogElement; nextIn: integer);
    ...
  end;
```

Поле FNextEl хранит ссылку на следующий логический элемент, а поле FNextIn — номер входа этого следующего элемента, к которому подключен выход данного элемента. С помощью общедоступного метода Link можно связать данный элемент со следующим:

```
procedure TLogElement.Link(nextElement:
          TLogElement; nextIn: integer);
begin
  FNextEl := nextElement;
  FNextIn := nextIn;
end;
```

Нужно немного изменить методы setIn1 и setIn2: при изменении входа они должны не только пересчитывать выход данного элемента, но и отправлять сигнал на вход следующего

```
procedure TLogElement.setIn1(newIn1:
          boolean);
begin
  FIn1 := newIn1; calc;
  if FNextEl <> nil then
    case FNextIn of
      1: FNextEl.In1 := res;
      2: FNextEl.In2 := res;
    end;
end;
```

Условие `FNextEl <> nil` означает "если следующий элемент задан". Если он не был установлен, значение поля `FNextEl` будет равно `nil` и никаких дополнительных действий не выполняется.

С учетом этих изменений вывод таблицы истинности функции "И-НЕ" можно записать так (операторы вывода заменены многоточиями):

```
elNot := TNot.Create;
elAnd := TAnd.Create;
elAnd.Link(elNot, 1);
...
for A := False to True do begin
  elAnd.In1 := A;
  for B := False to True do begin
    elAnd.In2 := B;
    ...
  end;
end;
```

Обратите внимание, что в самом начале мы установили связь элементов "И" и "НЕ" с помощью метода `Link` (связали выход элемента "И" с первым входом элемента "НЕ"). Далее в теле цикла обращения к элементу "НЕ" нет, потому что элемент "И" автоматически сообщит ему об изменении своего выхода.

Контрольные вопросы

1. Что такое классификация? Зачем она нужна? Приведите примеры.
2. В каком случае можно сказать, что "класс Б — наследник класса А", а когда "объект класса А содержит объект класса Б"? Приведите примеры.
3. Что такое иерархия классов?
4. Объясните приведенную иерархию логических элементов. Обсудите ее достоинства и недостатки.
5. Дайте полное определение ООП и объясните его.
6. Что такое базовый класс и класс-наследник? Какие синонимы используются для этих терминов?
7. На примере класса `TLogElement` покажите, как выполнена инкапсуляция.
8. Что такое виртуальный метод?
9. Что такое полиморфизм?
10. Что такое абстрактный класс? Почему нельзя создать объект этого класса?
11. Как транслятор определяет, что тот или иной класс — абстрактный?
12. Что нужно сделать, чтобы класс-наследник абстрактного класса не был абстрактным?
13. Зачем нужен описатель `protected`? Чем он отличается от `private` и `public`?
14. Что означает описатель `override`?
15. Какие преимущества дает применение модулей в программе?
16. Из каких частей состоит каждый модуль? Что включают в каждую из них?
17. Можно ли все содержимое модуля включить в секцию `interface`? Чем это плохо?
18. Можно ли все содержимое модуля включить в секцию `implementation`? Чем это плохо?
19. Объясните, как объекты могут передавать сообщения друг другу.

Задачи

1. Добавьте в иерархию классов элементы "исключающее ИЛИ", "И-НЕ" и "ИЛИ-НЕ".
2. "Соберите" в программе триггер из двух логических элементов "ИЛИ-НЕ", постройте его таблицу истинности (обратите внимание на вариант, когда оба входа нулевые).

Выводы

Сложность и размеры современных программ таковы, что в их разработке принимает участие множество программистов. Объектно-ориентированное программирование — это метод, позволяющий разбить задачу на части, каждая из которых в максимальной степени независима от других.

Программа в ООП — это набор объектов, которые обмениваются сообщениями.

Перед программированием выполняется объектно-ориентированный анализ задачи. На этом этапе выделяются взаимодействующие объекты, определяются их существенные свойства и поведение.

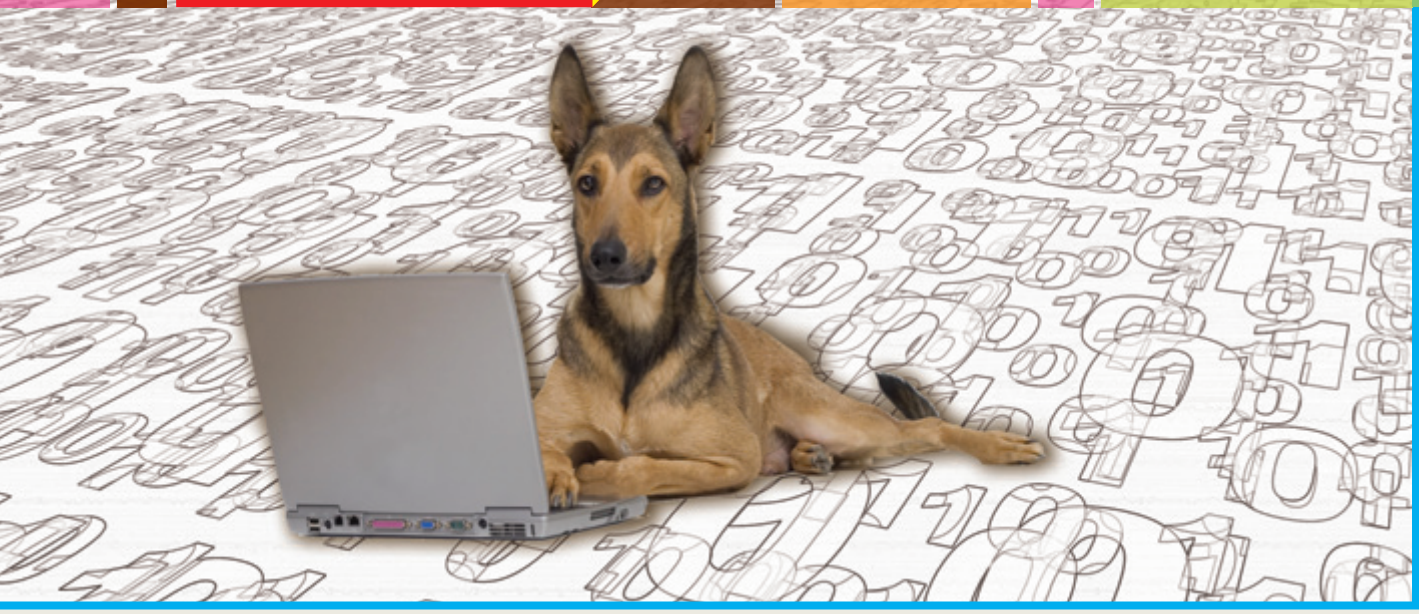
Любой объект — экземпляр какого-то класса. Классом называют группу объектов, обладающих общими свойствами.

Объекты не могут "узнать" устройство других объектов (принцип инкапсуляции). При описании класса закрытые поля и методы помещаются в секцию `private`, а общедоступные — в секцию `public`.

Обмен данными между объектами выполняется с помощью общедоступных свойств и методов, которые составляют интерфейс объектов. Изменение внутреннего устройства объектов (реализации) не влияет на взаимодействие с другими объектами, если не меняется интерфейс.

Как правило, классы образуют иерархию (многоуровневую структуру). Классы-потомки обладают всеми свойствами и методами классов-предков, к которым добавляются их собственные свойства и методы.

ООП позволяет обеспечивать высокую скорость и надежность разработки больших и сложных программ. В простых задачах применение ООП, как правило, увеличивает длину программы и замедляет ее работу.



Как криптография защищает информацию от активного злоумышленника

А.Ю. Зубов,
Москва

Изучение основ криптографии практически обязательно входит в профильный курс информатики. В учебники (включая разрабатываемые) для профильного курса соответствующий материал также включен. В предлагаемой вниманию читателей статье содержится строгое и исчерпывающее описание основных фактов и методов криптографии, которые могут быть рассмотрены в рамках школьного курса, включая проектную и исследовательскую работу.

Аутентификация информации

В литературе по защите информации противник, наблюдающий передаваемые в канале связи цифровые сообщения и пытающийся получить из них конфиденциальную информацию, но не изменяющий самих сообщений, называется *пассивным противником*. *Активным* называют противника, который имеет техническую возможность изменять передаваемые (или хранимые) данные, а также имитировать передачу сфабрикованных им самим сообщений (или сообщений, перехваченных ранее) от имени одного легального пользователя другому. Следует отметить, что “активным злоумышленником” может быть компьютерный вирус, “троянский конь” или “логическая бомба”[1]. Они могут нарушать целостность информации путем ее изменения. Подобные активные угрозы особенно опасны в банковской сфере, где аутентификация информации (т.е. ее защита от активных атак) более важна, чем конфиденциальность. Представьте себе, что будет изменено сообщение о переводе крупной суммы денег с одного счета на другой. Например, будет изменена сумма или номер счета. Успешная активная атака приводит к утрате доверия пользователей к системе. Еще одно приложение — защита аутентичности изображений и подвижных образов, что весьма важно при организации видеоконференций. Требуется, чтобы не было возможно несанкционированное добавление или изъятие видео- или аудиофрагментов. В повседневной жизни все мы, как пользователи сети связи или компьютерной системы, должны быть уверены в аутентичности приходящих в наш адрес сообщений или хранимых файлов. При загрузке на жесткий диск программного обеспечения нам бы также хотелось убедиться в его аутентичности (т.е. подлинности).

Как же можно защитить цифровые данные от активных атак типа имитации и подмены? Для этой цели используются *криптографиче-*

[1] Программа, которая запускается при определенных временных или информационных условиях для осуществления вредоносных действий (как правило, несанкционированный доступ к информации, изменение или уничтожение данных). Часто их действие привязано к определенной дате, например, к пятнице 13-го или ко дню смеха.

ские контрольные суммы[2], которые вычисляются как значения некоторой функции от защищаемых данных и секретного ключа. Владелец секретного ключа может проверить аутентичность данных, вычисляя КС и сравнивая ее с исходной КС. Для вычисления КС используются хеш-функции и системы шифрования. Давайте разберемся в том, как это делается.

Системы аутентификации на основе хеш-функций

Для удобства изложения будем представлять защищаемую информацию в виде конечных строк, состоящих из нулей и единиц. Обозначим через $\{0, 1\}^*$ множество всех таких строк, а его подмножество, состоящее из строк длины n (n — натуральное число), — через $\{0, 1\}^n$.

Хеш-функция — это отображение $F : \{0,1\}^* \to \{0,1\}^n$. Аргументом хеш-функции является произвольная двоичная строка M, а значением $F(M)$ — двоичная строка длины n. Значение $F(M)$ называют *хеш-образом* строки M.

На практике в качестве n используются числа 64, 128, 160 и некоторые другие. Хеш-образ вычисляется с помощью итеративной процедуры, в которой принимает участие так называемая *сжимающая функция* типа $g : \{0,1\}^{m+n} \to \{0,1\}^n$. Один из распространенных методов состоит в следующем. M представляется в виде последовательности подряд идущих строк M_i длины m: $M = M_1 || M_2 || \ldots || M_r$, выбирается начальная строка iv длины m и последовательно вычисляются строки длины n

$$C_1 = g(M_1 || iv),$$
$$C_2 = g(M_2 || C_1), \ldots, C_r = g(M_r || C_{r-1}). \qquad (1)$$

Так вот, $F(M)$ полагается равным C_r. Как правило, строка iv выбирается состоящей из одних нулей. В случае если длина M не кратна m, используется процедура дополнения последней неполной строки. Обычно дополнением служит строка 10…0, начинающаяся единицей, после которой следует нужное число нулей. Здесь и далее $a || b$ обозначает строку, полученную приписыванием справа от строки a строки b.

Имеются государственные и международные стандарты хеш-функций. Так, стандарт хеш-функции России имеет название “ГОСТ Р 34.11-94”. Его описание можно найти в Интернете.

Пример 1

Пусть $m = n$ и $g(x || y) = x \oplus y$, где $\oplus$ — это операция покоординатного сложения двоичных строк, в которой биты складываются по таблице

+	0	1
0	0	1
1	1	0

Например, если $x = 0110$, $y = 1111$, то $x \oplus y = 1001$.

Положим $m = n = 2$ и $M = 0110001011$. Вычислим $F(M)$. Имеем:

$M_1 = 01$, $M_2 = 10$, $M_3 = 00$, $M_4 = 10$, $M_5 = 11$, $iv = 00$.

Теперь находим

$C_1 = g(01||00) = 01$, $C_2 = g(10||01) = 11$,

$C_3 = g(00||11) = 11$, $C_4 = g(10||11) = 01$,

$C_5 = g(11||01) = 10$.

Таким образом, $F(M) = 10$.

К хеш-функциям предъявляется ряд требований, отсутствие которых может привести к успеху активной атаки. Требуется, чтобы следующие три задачи были труднорешаемыми[3]:

1. По любой данной строке $y \in \{0,1\}^n$ найти строку $x \in \{0,1\}^*$, такую, что $F(x) = y$. Эта задача называется задачей *нахождения прообраза*.

2. По любой данной строке $x \in \{0,1\}^*$, найти строку $x' \in \{0,1\}^*$, такую, что $F(x) = F(x')$. Эта задача называется задачей *нахождения второго прообраза*.

3. Найти аргументы x, x', для которых $F(x) = F(x')$. Эта задача называется задачей *нахождения коллизии*[4].

Хеш-функции, удовлетворяющие требованиям 1–3, называются *криптографическими*. Именно такие хеш-функции используются в системах аутентификации.

Пример 2

Выясним, выполняются ли требования 1–3 для хеш-функции из примера 1.

Для любой строки $y \in \{0,1\}^n$ строка $x = y || 0^n$, в которой 0^n — строка из n нулей, удовлетворяет равенству $F(x) = y$. Следовательно, первое требование не выполняется. Если x_1, x_2 — произвольные строки из $\{0,1\}^n$, то, очевидно, $F(x_1 || x_2) = F(\bar{x}_1 || \bar{x}_2)$, где $\bar{x}$ — строка, полученная из строки x путем замены всех единиц нулями и всех нулей единицами. И второе требование не выполняется. Ясно, что и третье требование (как более слабое) также не выполняется.

Один из возможных вариантов КС файла M, хранимого в памяти компьютера, — это хеш-образ $C = F(M)$. Если C хранить отдельно от M, то для проверки целостности файла (который может подвергаться изменениям) следует повторно вычислить хеш-образ файла и сравнить его с C. Если F — это криптографическая хеш-функция, то совпадение хеш-образов гарантирует целостность файла. Такая КС называется *кодом обнаружения ошибок* (в англоязычной литературе используется сокращение MDC[5]). Однако MDC не обеспечивает защиты от угрозы имитации, поскольку сама функция F, с помощью которой вычисляется MDC, несекретна и

[2] Будем для краткости называть их далее КС.

[3] Задачи, вычислительная сложность которых не допускает их решение с использованием вычислительной техники за реальное время. Для этого необходимо, чтобы n было достаточно большим числом.

[4] Коллизия — это сама пара (x, x'), для которой $F(x) = F(x')$.

[5] От *modification detection code*.

кто угодно может составить любое сообщение и вычислить его MDC. Более того, в некоторых случаях это можно сделать даже тогда, когда функция F известна лишь частично.

Пример 3

Пусть текст M записан в алфавите из n букв. Пронумеруем буквы алфавита по порядку числами от 0 до $n-1$ и заменим буквы их порядковыми номерами. Обозначим через $\mathbb{Z}_n$ множество чисел $\{0,1,\ldots,n-1\}$. На $\mathbb{Z}_n$ введем операции сложения и умножения. Для $a,b\in\mathbb{Z}_n$ через $(a+b)\bmod n$ и $(a\cdot b)\bmod n$ будем обозначать остаток от деления на n числа $a+b$ (соответственно, $a\cdot b$).

Будем представлять сообщения M в виде $M=m_1||m_2$, где $m_1,m_2\in\mathbb{Z}_n$, и определим MDC формулой $F(M)=(a\cdot m_1+b\cdot m_2)\bmod n$, где a,b — фиксированные числа из $\mathbb{Z}_n$.

Пусть противник наблюдает пару передаваемых сообщений $M||F(M)$, $M'||F(M')$, снабженных MDC, где $M=(m_1,m_2)$, $M'=(m_1',m_2')$. Тогда, даже не зная чисел a,b, он сможет построить поддельное сообщение $M''||F(M'')$ с корректным MDC и имитировать его передачу от имени легального пользователя. В самом деле, он выберет $M''=(m_1'',m_2'')$ так, чтобы была разрешима система уравнений

$$\begin{cases} m_1''=(u\cdot m_1+v\cdot m_1')\bmod n,\\ m_2''=(u\cdot m_2+v\cdot m_2')\bmod n,\end{cases}$$

относительно $u,v\in\mathbb{Z}_n$, найдет u,v и вычислит $F(M'')$ следующим образом:

$$\begin{aligned}F(M'')&=F((u\cdot m_1+v\cdot m_1')\bmod n,(u\cdot m_2+v\cdot m_2')\bmod n)=\\&=[a\cdot(u\cdot m_1+v\cdot m_1')+b\cdot(u\cdot m_2+v\cdot m_2')]\bmod n=\\&=[u\cdot(a\cdot m_1+b\cdot m_2)+v\cdot(a\cdot m_1'+b\cdot m_2')]\bmod n=\\&=[u\cdot F(M)+v\cdot F(M')]\bmod n.\end{aligned}$$

Для защиты от угрозы имитации отправитель и получатель сообщения должны (как это указывалось выше) использовать некий общий “секрет”[6], неизвестный противнику. Если ключ использовать при вычислении КС, то противник не сможет создать корректное поддельное сообщение, и имитация не удастся. Однако пример 3 показывает, что далеко не каждая хеш-функция может использоваться для выработки такой КС. В самом деле, в этом примере параметры a и b могут быть секретными, но, как показано, и в этом случае противник легко добьется успеха.

Пусть ключ представляет собой строку из $\{0,1\}^s$. В системах аутентификации ключ добавляется к аргументу хеш-функции F. Первые известные способы использовали в качестве КС сообщения M значение типа $F(M||k)$, $F(k||M)$ или $F(k||M||k)$. Однако в таких способах были обнаружены недостатки.

[6] Будем называть его далее *ключом*.

Пример 4

Пусть КС сообщения M вычисляется по формуле $C=F(k||M)$, причем один и тот же ключ может быть использован для вычисления КС нескольких сообщений. Предположим, что активный противник знает об этом[7] и наблюдает снабженное КС передаваемое сообщение $M||C$, в котором длина l строки M удовлетворяет условию, что $l+s$ кратно m. Тогда противник, не зная ключа k, может снабдить поддельное сообщение $M||M'$, где M' — произвольная строка длины m, корректной КС. В самом деле, согласно определению, КС сообщения $M||M'$ равна значению $C'=F(k||M||M')$. По условию $k||M$ представляется в виде последовательности строк $M_1||\ldots||M_r$ длины m, при этом $C=g(M_r,C_{r-1})$, где g — сжимающая функция. Но тогда $C'=g(M'||C)$. Зная g, M' и C, противник может вычислить C'. В результате успех имитации ему гарантирован.

Сегодня используется следующий способ вычисления КС сообщения M с помощью добавляемого к аргументу хеш-функции секретного ключа k: $F(k||F(k||M))$. Таким образом, чтобы вычислить КС, следует сначала вычислить $M'=F(k||M)$, а затем $F(k||M')$. Такая система аутентификации называется HMAC[8].

Замечание 1

Аббревиатура HMAC — сокращение от *Hash-MAC*. В свою очередь, MAC — это сокращение от *Message Authentication Code — код аутентичности сообщения*. Часто термин “MAC” используется сегодня для обозначения контрольной суммы, вычисленной с помощью хеш-функции, зависящей от секретного ключа.

Теперь поговорим о способе вычисления КС, в котором используется шифрование.

Системы аутентификации на основе шифрования

Введем сначала понятие шифрсистемы.

Шифрсистема — это совокупность двух семейств отображений $E_{k_e}:\mathrm{P}\to\mathrm{C}$, $D_{k_d}:\mathrm{C}\to\mathrm{P}$ для каждой пары $(k_e,k_d)\in\mathrm{K}$, таких, что для любых $x\in\mathrm{P}$ выполняется равенство

$$D_{k_d}(E_{k_e}(x))=x. \qquad (2)$$

Множества K, P, C называются, соответственно, множеством *ключей*, *открытых текстов* и *шифрованных текстов*. Каждый ключ $k\in\mathrm{K}$ представляет собой пару (k_e,k_d), в которой k_e — *ключ зашифрования* и k_d — *ключ расшифрования*. Отображение E_{k_e} называется *правилом зашифрования на ключе* k_e,

[7] Такое допущение общепризнано в криптографии и носит название правила Керкгоффса.

[8] Ее описание можно найти в Интернете.

а отображение D_{k_d} — *правилом расшифрования на ключе* k_d.

Для того чтобы передать конфиденциальное сообщение $x \in P$, отправитель и получатель должны выбрать некоторый ключ $k = (k_e, k_d)$. При этом ключ k_e должен быть у отправителя, а ключ k_d — у получателя. Отправитель вычисляет $y = E_{k_e}(x)$ и посылает y получателю. Получатель y вычисляет (пользуясь свойством (2)) $x = D_{k_d}(y)$.

В случае если $k_d = k_e$, или если один из ключей пары (k_e, k_d) несложно получить из другого, то мы имеем дело с *шифрсистемой с секретным ключом*. В этом случае оба ключа должны быть недоступны противнику.

Если же практически невозможно найти ключ k_d по ключу k_e [9], то мы имеем дело с *шифрсистемой с открытым ключом*. Ключ k_e объявляется *открытым ключом*, он может быть общедоступен; ключ k_d должен быть недоступным никому, кроме получателя; он называется *секретным* (или *закрытым*) *ключом*.

Каждый пользователь шифрсистемы с открытым ключом (обозначим его H) имеет алгоритм вычисления $E_{k_e}(\cdot)$ для произвольного значения аргумента (обозначим этот алгоритм E_H) и алгоритм вычисления $D_{k_d}(\cdot)$ для произвольного значения аргумента (обозначим этот алгоритм D_H). Алгоритм E_H публикуется в общедоступном справочнике пользователей, а алгоритм D_H хранится в секрете. Если пользователь A желает передать конфиденциальное сообщение x пользователю H, он берет из справочника E_H, вычисляет $y = E_H(x)$ и посылает y. Получив y, H вычисляет $x = D_H(y)$.

Если для шифрования сообщений используется шифрсистема с секретным ключом, то говорят о *симметричном шифровании*, а если шифрсистема с открытым ключом, то говорят об *асимметричном шифровании*.

Пример 5

Шифрсистема Цезаря

Пусть открытые тексты записаны в алфавите, состоящем из n букв. Так же, как в примере 3, заменим буквы их порядковыми номерами. Будем использовать $\mathbb{Z}_n$ в качестве множества ключей шифрсистемы Цезаря. Результатом зашифрования строки $x = a_1 \ldots a_l$, $a_i \in \mathbb{Z}_n$, $l \geq 1$, на ключе $k \in \mathbb{Z}_n$ является $y = b_1 \ldots b_l$, где

$$b_1 = (a_1 + k) \bmod n, \ \ldots, \ b_l = (a_l + k) \bmod n.$$

Для расшифрования $y = b_1 \ldots b_l$ нужно вычислить

$$a_1 = (b_1 - k) \bmod n, \ \ldots, \ a_l = (b_l - k) \bmod n,$$

где $(a - k) \bmod n$ обозначает остаток от деления числа $a - k$ на n, в случае, если $a - k > 0$, и числа $a + n - k$ на n, если $a - k < 0$.

[9] Это означает, что с использованием всех известных средств и методов для этого потребуется недопустимо большое время.

Зашифруем, например, на ключе $k = 13$ фразу “шифр цезаря”, записанную в алфавите

а	б	в	г	д	е	ж	з	и	к	л	м	н
00	01	02	03	04	05	06	07	08	09	10	11	12

о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь
13	14	15	16	17	18	19	20	21	22	23	24	25

ы	э	ю	я								
26	27	28	29								

Открытый текст записывается в виде числовой последовательности

23081915210507001529,

а шифрованный текст — в виде последовательности

06210228041820132812.

Переходя вновь к буквам, получаем шифртекст: “жцбюдучоюн”.

Шифрсистема Цезаря относится к числу шифрсистем с секретным ключом. Приведем пример шифрсистемы с открытым ключом.

Пример 6

Шифрсистема RSA

Каждый пользователь системы выбирает достаточно большие [10] простые числа p и q и вычисляет произведение $n = pq$. Затем выбираются бóльшие единицы числа e и d из множества $\mathbb{Z}_n$, такие, что $ed - 1 = 0 \bmod \varphi(n)$. Здесь $\varphi(n)$ — значение *функции Эйлера*, равное количеству чисел из множества $1, 2, \ldots, n-1$, взаимно простых с n. Например, $\varphi(8) = 4$, $\varphi(20) = 8$. Для выбора чисел e и d может быть использован алгоритм Евклида. Пара чисел (n, e) объявляется открытым ключом, а число d является секретным ключом данного пользователя.

Открытые и шифрованные тексты шифрсистемы RSA — это числа из множества $1, 2, \ldots, n-1$, записанные $t \leq [n]$ десятичными знаками, где $[n]$ — число знаков в десятичной записи n. Если открытый текст слишком длинный, то он представляется в виде последовательности отрезков $M_1 || \ldots || M_r$, в которой для записи каждого отрезка M_i требуется ровно t десятичных знаков. Каждый отрезок M_i шифруется с помощью открытого ключа e, и результаты зашифрования — C_i (также записанные t десятичными знаками) составляют шифртекст: $C_1 || \ldots || C_r$.

Для того чтобы представить открытый текст в таком виде, достаточно, как и в предыдущем примере, заменить буквы алфавита, в котором записан открытый текст, их порядковыми номерами в этом алфавите, используя десятичную запись.

Пусть пользователь A желает послать сообщение M пользователю H. Тогда A берет из справочника

[10] Сегодня это должны быть числа, для записи которых требуется более 150 десятичных знаков.

пользователей открытый ключ (n_H, e_H), вычисляет шифртекст

$$C = E_H(M) = M^{e_H} \bmod n_H \qquad (3)$$

и посылает его к H. Получив C, пользователь H вычисляет открытый текст по формуле

$$M = D_H(C) = C^{d_H} \bmod n_H. \qquad (4)$$

Заметим, что вычисления по формулам (3) и (4) легко производятся (для не слишком больших чисел) с помощью калькулятора системы Windows, в котором имеется режим Mod, позволяющий производить вычисления по модулю.

Зашифруем, например, с помощью "маленького RSA" фразу "шифр Цезаря", пользуясь алфавитом из примера 4.

Пусть $p = 47$, $q = 53$, $n = pq = 2491$. Пусть $e = 5$, тогда, пользуясь алгоритмом Евклида, находим $d = 957$. Открытым ключом служит пара чисел (2491, 5), а секретным ключом — число 957. Возьмем числовую запись открытого текста из примера 4 и представим ее (при $t = 4$) в виде

2308||1915||2105||0700||1529.

Далее с помощью калькулятора ПК вычислим значения

$$2308^5 \bmod 2491 = 1527,$$

$$1915^5 \bmod 2491 = 1490,$$

$$2105^5 \bmod 2491 = 486,$$

$$700^5 \bmod 2491 = 1998,$$

$$1529^5 \bmod 2491 = 1046.$$

Теперь получаем шифртекст:

15271490048619981046

Произведем его расшифрование. Для этого представим шифртекст в виде

1527||1490||0486||1998||1046

и с помощью калькулятора вычислим значения

$$1527^{957} \bmod 2491 = 2308,$$

$$1490^{957} \bmod 2491 = 1915,$$

$$486^{957} \bmod 2491 = 2105,$$

$$1998^{957} \bmod 2491 = 700,$$

$$1046^{957} \bmod 2491 = 1529.$$

Далее восстанавливаем открытый текст

23081915210507001529

и его буквенный вариант "шифр Цезаря".

Шифрсистемы делятся на *поточные* и *блочные*. Первые последовательно шифруют отдельные знаки открытого текста (как, например, шифрсистема Цезаря), а вторые — блоки знаков открытого текста (например, как RSA). Сегодня стандарты шифрования (как государственные, так и международные) основаны на блочных шифрсистемах с секретным ключом. При этом размеры блоков составляют 64 или 128 бит. Например, стандарт шифрования России — ГОСТ 28147-89 имеет размер блока 64 бита и ключ размера 256 бит, а стандарт шифрования США — AES имеет размер блока 128 бит и ключ размера 128, 192 или 256 бит (в зависимости от модификации).

Аутентификация на основе асимметричного шифрования

Шифрсистемы с открытым ключом используются для построения *цифровых подписей*, которые служат эквивалентом собственноручных подписей автора для цифровых документов. Идея очень проста. Пусть E_A и D_A — открытый и секретный алгоритмы шифрования пользователя A. Тогда $D_A(M)$ — результат зашифрования документа M с помощью секретного алгоритма — можно рассматривать как "подписанный" документ. В самом деле, любой владелец справочника пользователей может взять E_A и вычислить $E_A(D_A(M))$. Результатом вычисления должно быть M, если E_A и D_A осуществляют взаимно обратные преобразования[11]. Такое вычисление может служить доказательством того, что документ M был составлен именно пользователем A, поскольку только он обладает секретным алгоритмом D_A и никто, кроме него, не мог составить $D_A(M)$.

Выясним, можно ли $D_A(M)$ использовать в качестве КС сообщения M. Если речь идет о проверке целостности M, то такая КС вполне пригодна. При замене M на $M' \neq M$ проверка целостности сообщения сводится к проверке равенства $D_A(M) = D_A(M')$. Вероятность того, что это равенство выполняется, ничтожно мала. Однако для защиты от имитации предложенная КС не годится. В самом деле, пусть противнику известна пара $(M, D_A(M))$. Тогда, применяя доступное ему преобразование E_A, противник может составить сообщение $M' = E_A(M)$ и снабдить его КС $D_A(M')$, равной M. Поддельное сообщение будет принято как аутентичное.

В качестве КС сообщения M используется не $D_A(M)$, а $D_A(F(M))$, где F — криптографическая хеш-функция. Такую КС можно рассматривать и как цифровую подпись сообщения. Такой метод вычисления КС не обладает указанным выше недостатком. Кроме того, он обладает существенным преимуществом, состоящим в том, что хеш-образ $F(M)$ имеет небольшую длину, и его шифрование требует меньше времени, чем шифрование всего сообщения M, которое может быть достаточно длинным. При этом нужно иметь в виду, что скорость вычисления значения хеш-функции значительно выше скорости асимметричного шифрования.

Использование для аутентификации цифровой подписи целесообразно в сочетании с удостоверением авторства документа, то есть в тех случаях, когда требуется доказательство принадлежности документа его автору. Это требование характерно для среды *не доверяющих друг другу* пользователей. В такой среде возможны угрозы типа *отказа от авторства* или *приписывания авторства*. Цифровая подпись дает возможность разрешения подобных конфликтов в суде, в то время как средства симме-

[11] Именно такие шифрсистемы используются для построения систем цифровой подписи.

тричного шифрования (с секретным ключом) такой возможности не дают.

Отметим, что цифровая подпись не всегда является достаточно эффективным инструментом в среде доверяющих друг другу пользователей. Например, для приложений с короткими сообщениями, используемыми в смарт-картах, выработка цифровой подписи и ее проверка требуют значительного времени. Кроме того, для хранения заголовка цифровой подписи, а также ключей и системных параметров требуется значительный объем памяти. Поэтому в тех приложениях, где не требуется удостоверение авторства и желательна высокая скорость обработки информации, целесообразно использовать средства симметричного шифрования, о чем пойдет речь далее.

Пример 7

Получим цифровую подпись сообщения “шифр Цезаря” типа $D_A(F(M))$, пользуясь RSA из примера 6 и хеш-функцией F из примера 1.

Запишем сообщение в цифровой форме, пользуясь таблицей из примера 5:

$23||08||19||15||21||05||07||00||15||29$.

Запишем каждую пару десятичных цифр в двоичной форме:

$(23)_{10} = (10111)_2$, $(08)_{10} = (01000)_2$, $(19)_{10} = (10011)_2$, $(15)_{10} = (01111)_2$, $(21)_{10} = (10101)_2$, $(05)_{10} = (00101)_2$, $(07)_{10} = (00111)_2$, $(00)_{10} = (00000)_2$, $(15)_{10} = (01111)_2$, $(29)_{10} = (11101)_2$.

В результате получим двоичную строку

$$M = 10111010001001101111101000101001110000000111111101. \quad (5)$$

Вычислим значение $F(M)$, пользуясь начальной строкой $iv = 10$. Легко проверить, что $F(M) = (2)_{10}$.

Вычислим $D_H(F(M))$ (D_H из примера 5):

$$D_H(2) = 2^{957} \bmod 2491 = 1532.$$

Итак, цифровой подписью сообщения служит число 1532.

Аутентификация на основе симметричного шифрования

Многие системы аутентификации информации основаны на блочных шифрсистемах. В современных системах симметричного шифрования ключи зашифрования и расшифрования совпадают, поэтому отправитель и получатель сообщений имеют общий секретный ключ k. В силу того, что при этом в качестве алфавитов открытого и шифрованного текстов используется множество $\{0,1\}^n$, блочный шифр представляется совокупностью взаимно однозначных преобразований

$$E_k : \{0,1\}^n \to \{0,1\}^n,\ k \in K = \{0,1\}^s.$$

Для вычисления контрольной суммы открытый текст M представляется в виде последовательности блоков $M = M_1||M_2||\ldots||M_r$, где $M_i \in \{0,1\}^n$. Если при этом последний блок M_r окажется неполным, то он дополняется строкой 10…0 с нужным числом нулей до полного блока. Шифрование M может осуществляться в различных *режимах*. Простейший режим[12] вырабатывает шифртекст $C = C_1||C_2||\ldots||C_r$, в котором $C_i = E_k(M_i)$ для каждого $i = 1, 2, \ldots, r$, а k — секретный ключ. Имеются и другие режимы шифрования. Для выработки КС в системах аутентификации используется *режим сцепления блоков*[13]. Алгоритм зашифрования в этом режиме схож с алгоритмом вычисления значения хеш-функции в (1):

$$C_1 = E_k(M_1 \oplus iv),\ C_2 = E_k(M_2 \oplus C_1), \ldots, \quad (6)$$
$$C_r = E_k(M_r \oplus C_{r-1}).$$

Контрольной суммой сообщения M служит блок C_r из (6). Как правило, начальная строка iv состоит из одних нулей.

При передаче сообщения вместе с его КС получатель производит последовательность зашифрований (6) и сравнивает вычисленную КС с полученной КС. Совпадение гарантирует аутентичность сообщения. Конечно, это верно лишь в том случае, когда задача восстановления секретного ключа k по данным M и C_r является вычислительно сложной. В противном случае активный противник найдет k и сможет вычислить корректную КС для поддельного сообщения. Другими словами, в основе стойкой аутентификации лежит стойкое шифрование. Описанная конструкция КС называется CBC-MAC.

Пример 8

Вычислим КС сообщения “шифр Цезаря” с помощью CBC-MAC при использовании шифра с параметрами $n = s = 10$, и правилом зашифрования блока $x = x_1||x_2$ на ключе $k = k_1||k_2$ (состоящих из своих половин), определяемым формулой

$$E_k(x) = x_1 \oplus k_1||x_2 \oplus k_2\ ^{14}. \quad (7)$$

Воспользуемся двоичным представлением (5) нашего сообщения и запишем его в виде последовательности блоков длины 10:

$$1011101000||1001101111||1010100101||0011100000||0111111101.$$

Выберем секретный ключ $k = 1011000111$ и, пользуясь формулами (6) и (7), произведем последовательность зашифрований:

$$E_k(1011101000 \oplus 0000000000) =$$
$$= E_k(1011101000) = 0000101111,$$
$$E_k(1001101111 \oplus 0000101111) =$$
$$= E_k(1001000000) = 0010001111,$$
$$E_k(1010100101 \oplus 0010001111) =$$
$$= E_k(1000101010) = 0011101101,$$
$$E_k(0011100000 \oplus 0011101101) =$$
$$= E_k(0000001101) = 1011100101,$$

[12] В англоязычной литературе он называется режимом ECB (*electronic code book*).

[13] От CBC (*cipher block chaining*).

[14] Проверьте, что формула (7) определяет взаимно однозначное преобразование.

$$E_k(0111111101 \oplus 1011100101) =$$

$$= E_k(1100011000) = 0111110000.$$

Таким образом, КС нашего сообщения равна 0111110000. Можно вернуться к буквенной записи, заменив по таблице из примера 4 половину $(01111)_2 = (15)_{10}$ КС буквой P, а другую половину $(10000)_2 = (16)_{10}$ буквой C. Тогда КС сообщения состоит из двух букв PC.

Следует отметить, что конструкция CBC-MAC обеспечивает стойкую аутентификацию лишь сообщений фиксированной длины. Это означает, что если один и тот же ключ используется для аутентификации нескольких сообщений разной длины, то активный злоумышленник сможет построить поддельное сообщение, снабженное корректной КС.

Пример 9

Пусть M — одноблочное сообщение и τ_M — его КС, полученная на ключе k. Если злоумышленник знает пару (M, τ_M), то он может построить подделку $(M \,||\, (M \oplus \tau_M), \tau_M)$, в которой КС получена на том же ключе k и совпадает с τ_M. В самом деле, по определению КС сообщения $M \,||\, (M \oplus \tau_M)$ вычисляется следующим образом:

$$E_k(E_k(M) \oplus (M \oplus \tau_M)) = E_k(\tau_M \oplus (M \oplus \tau_M)) =$$

$$= E_k(M \oplus (\tau_M \oplus \tau_M)) = E_k(M) = \tau_M.$$

Пример 10

Пусть M_1, M_2 — одноблочные сообщения и τ_{M_1}, τ_{M_2} — их КС, полученные на ключе k. Если злоумышленник знает пары (M_1, τ_{M_1}), (M_2, τ_{M2}), то он может построить подделку (M, τ_{M_2}), в которой $M = M_1 \,||\, (M_2 \oplus \tau_{M_1})$. При этом КС, полученная на том же ключе k, совпадает с τ_{M_2}, в чем легко убедиться.

Чтобы избавить систему CBC-MAC от указанного недостатка, в последовательность зашифрований (6) добавляют еще один шаг, на котором КС сообщения вычисляется как значение $f(C_r)$ некоторой функции f. Например, для схемы EMAC $f(C_r) = E_{k_1}(C_r)$, где k_1 — еще один секретный ключ (отличный от k), а в схеме CBC-MAC-Y $f(C_r) = E_k(D_{k_1}(C_r))$.

Как правило, системы аутентификации на основе хеш-функций и блочного шифрования позволяют использовать секретный ключ для аутентификации многих сообщений. Это означает, что активный злоумышленник не сможет построить поддельное сообщение, даже обладая несколькими аутентифицированными на одном ключе сообщениями. Это является достоинством подобных систем, поскольку в значительной мере именно расход секретных ключей определяет стоимость защиты. Другие их достоинства — универсальность (т.е. применимость к сообщению любой длины) и высокая скорость вычисления КС.

Стойкость таких систем оценивается вычислительной сложностью задач, характеризующих возможность "взлома" системы. Если обеспечить противника неограниченными вычислительными ресурсами, то он наверняка добьется успеха. Пусть, например, известно, что шифр обладает тем свойством, что по шифртексту достаточной длины[15] однозначно определяется ключ. Тогда, имея шифртекст, противник может себе позволить перебрать все ключи (даже если их число астрономически велико), на каждом из них расшифровать данный шифртекст и определить единственно возможный открытый текст[16] и соответствующий ключ.

Аналогично, если система аутентификации основана на использовании шифрования, и условия задачи (например, совокупность нескольких пар (текст — КС), полученных на одном ключе) позволяют найти ключ путем перебора, то противник сможет определить ключ и построить подделку. Такие системы называют *практически стойкими*. Фактически их стойкость определяется ограниченными вычислительными возможностями потенциального противника. С ростом таких возможностей стойкость снижается.

Имеются и другие системы аутентификации, стойкость которых определяется с позиции противника, обладающего неограниченными вычислительными ресурсами. Такие системы называются *теоретически стойкими*. Их стойкость оценивается *вероятностью успеха атаки*. Чем больше эта вероятность, тем менее стойка система к рассматриваемой атаке. Теоретически стойкими системы могут быть лишь в рамках соответствующей атаки. Применительно к системам аутентификации речь идет, как правило, об условиях, когда злоумышленник принимает решение о проведении активной атаки на основе наблюдения не более чем одного передаваемого сообщения. Это означает, что система аутентификации для выработки КС каждого сообщения использует новый ключ, выбираемый случайно. Тем самым, такие системы проигрывают в сравнении с практически стойкими системами по расходу ключей. Кроме того, многие подобные системы пригодны лишь для ограниченного множества сообщений, например, для сообщений фиксированной длины. Тем не менее для многих приложений теоретически стойкие системы вполне применимы. Они выигрывают в скорости реализации и, что более важно, *гарантируют стойкость*. Часто такие системы аутентификации называют *кодами аутентификации*.

Замечание 2

Отметим, что и рассмотренные выше практически стойкие системы могут рассматриваться и как коды аутентификации в условиях ограниченной атаки, когда, например, каждый ключ может использоваться для аутентификации лишь одного сообщения. При этом условии система может гарантировать теоретическую стойкость, однако строго обосновать это весьма сложно.

[15] Эта величина называется в криптографии *расстоянием единственности*.

[16] Для этого используется тот или иной *критерий на открытый текст*.

Теоретически стойкие системы аутентификации

Начнем с примера кода аутентификации.

Пример 11

Пусть p — достаточно большое простое число. Будем аутентифицировать сообщения $M \in \{0,1\}^{\le n}$, где n — число, удовлетворяющее неравенству $2^n < p \le 2^{n+1}$, а $\{0,1\}^{\le n}$ — множество двоичных строк, длина которых не превосходит n. Будем представлять M числом из множества $\mathbb{Z}_p = \{0,1,\ldots,p-1\}$. Например, десятичное число 13 представляет строку $M = 1101$. Контрольную сумму сообщения M будем вырабатывать по формуле

$$f(M,k) = (a \cdot M + b) \bmod p, \qquad (8)$$

в которой ключ k представляется парой (a,b), $a,b \in \mathbb{Z}_p$, а результат $a \cdot M + b$ обычных операций над целыми числами берется по модулю p. КС будем записывать двоичной строкой длины $n + 1$. Для вычисления КС каждого сообщения будем использовать новый ключ, выбираемый совершенно случайно[17].

Оценим для такой системы аутентификации шансы на успех активной атаки.

Рассмотрим сначала шансы на успех подмены сообщения. Пусть противник наблюдает передачу сообщения $M \,||\, f(M,k)$ и желает подменить M на $M' \ne M$. Тогда ему следует подменить и КС $f(M,k)$ на КС $f(M',k)$ (иначе поддельное сообщение не будет принято как аутентичное). Противник не знает ключа $k = (a,b)$, но знает, что a и b связаны соотношением (8). Поскольку по условию a и b выбираются совершенно случайно, противнику при вычислении $f(M',k)$ не остается ничего лучшего, чем с равными шансами на успех выбрать в качестве a произвольное значение a' из $\mathbb{Z}_p$. Соответствующее значение b' противник найдет из (8). Пусть $k' = (a',b')$. Тогда поддельное сообщение M' противник снабдит КС, равной

$$f(M',k') = [a' \cdot M' + (f(M,k) - a' \cdot M)] \bmod p =$$
$$= [a'(M' - M) + f(M,k)] \bmod p.$$

Заметим, что при выборе различных значений a' различными окажутся и значения $f(M',k')$. Поэтому с равными шансами КС совпадет с любым элементом из $\mathbb{Z}_p$. Тем самым, шансы противника на успех подмены оцениваются величиной $1/p$.

При проведении атаки типа имитации для противника не имеют значения наблюдаемые сообщения, поскольку каждое передаваемое сообщение аутентифицируется своим ключом. Тогда очевидно, что шансы на успех имитации оцениваются величиной $1/p^2$, поскольку именно столько имеется равновозможных выборов пары (a,b).

[17] Например, с помощью лототрона или игровой рулетки.

Проведенные рассуждения показывают, что против рассматриваемой системы аутентификации противнику целесообразно применять атаку типа подмены. При этом не имеет никакого значения, какими вычислительными ресурсами он обладает. Мы обосновали теоретическую стойкость системы. Если выбрать p примерно равным 10^{80} (а эта величина представляет число атомов во Вселенной), то противник практически не будет иметь шансов на успех активной атаки.

Еще раз подчеркнем, что все сказанное о стойкости системы верно лишь при условии, что ключ используется для аутентификации лишь одного сообщения, так как в противном случае ни о какой стойкости речи быть не может. В самом деле, пусть ключ используется для выработки КС двух различных сообщений. Тогда противник имеет систему уравнений

$$\begin{cases} f(M,k) = (a \cdot M + b) \bmod p, \\ f(M',k) = (a \cdot M' + b) \bmod p, \end{cases}$$

в которой $M, M', f(M,k), f(M',k)$ ему известны. Вычитая из первого уравнения второе, он получает равенство

$$f(M,k) - f(M',k) = a(M - M') \bmod p. \qquad (9)$$

Поскольку по условию $M' \ne M$, в $\mathbb{Z}_p$ найдется число $(M - M')^{-1}$, такое, что

$$(M - M')^{-1} \cdot (M - M') = 1.$$

Умножая на это число[18] обе части равенства (9), противник найдет a и затем из первого уравнения системы — b. Найдя ключ, противник сможет подменить M' любым другим сообщением.

Разработка теоретически стойких систем аутентификации началась в 70-х годах прошлого века под руководством Г.Симмонса в связи с реализацией механизма проверки выполнения международных договоров, например, таких, как договор между СССР и США о запрещении испытаний ядерного оружия. Симмонс ввел понятие кода аутентификации, построил его общую модель и исследовал основные свойства. После Симмонса многие математики занимались исследованием кодов аутентификации. Сегодня для построения эффективно реализуемых стойких кодов аутентификации используются так называемые *универсальные семейства хеш-функций*. Формула (8) определяет одно из таких семейств.

Пример 12

Приведем пример кода аутентификации, который можно использовать для аутентификации сообщений о результатах случайного бросания монеты.

Обозначим через H и T — "орел" и "решку" соответственно. Пусть A бросает монету и передает B сообщение о результате — H или T. При этом получатель сообщения должен иметь возможность проверить его аутентичность. С этой целью A и B выбирают семейство однозначных отображений $\{e_k : S \to M,\ k \in K\}$, называемых *правилами кодирования*, где $S = \{H,T\}$ — множество *состояний источника*, M — множество *сообщений*, а K — множество *ключей*. Для каждого

[18] Которое можно найти с помощью алгоритма Евклида.

$s \in S$ и каждого $k \in K$ должно выполняться равенство $e_k^{-1}(e_k(s)) = s$, где e_k^{-1} — обратное к e_k отображение. Формально отображение $e_k^{-1} : M \to S \cup \{0\}$ задается для $m \in M$ формулой

$$e_k^{-1}(m) = \begin{cases} s \in S, \text{ если } e_k(s) = m, \\ 0, \text{ если } e_k(s) \neq m, \end{cases}$$

для любого $s \in S$.

Здесь 0 — "вспомогательный" элемент, не принадлежащий множеству S.

A и B договариваются о выборе секретного ключа $k \in K$. Теперь A, наблюдая $s \in S$, вычисляет $m = e_k(s)$ и передает m по каналу связи B. Получив m, B вычисляет $e_k^{-1}(m)$ и принимает m как аутентичное сообщение в том и только в том случае, когда $e_k^{-1}(m) \neq 0$. Активный противник, подменяя m на $m' \neq m$, добьется успеха лишь тогда, когда $e_k^{-1}(m') \neq 0$.

Не зная используемого ключа, он сможет добиться успеха лишь с некоторой вероятностью.

Определим наш код аутентификации с помощью *матрицы кодирования*. Так называется прямоугольная таблица W размером $|K| \times |M|$, строки которой занумерованы ключами $k \in K$, а столбцы — сообщениями $m \in M$. На пересечении строки k и столбца m в этой таблице расположим $e_k^{-1}(m)$. Пусть $K = \{1, 2, 3, 4\}$, $M = \{m_1, m_2, m_3, m_4\}$ и

$$\mathrm{W} = \begin{pmatrix} H & T & 0 & 0 \\ 0 & H & T & 0 \\ 0 & 0 & H & T \\ T & 0 & 0 & H \end{pmatrix}.$$

Пусть A и B договорились об использовании ключа $k = 2$. Тогда A, наблюдая $s = H$, передаст B сообщение $m_2 = e_2(H)$. B, получив m_2, вычислит $e_2^{-1}(m_2)$ и восстановит состояние источника $s = H$. Противник, наблюдая m_2 и зная матрицу кодирования, может определить лишь то, что использовался один из двух ключей — $k = 1$ или $k = 2$. Если стороны A и B выбирали ключ совершенно случайно, то противник с равными шансами на успех может "выбрать" один из этих ключей. Если он отдаст предпочтение $k = 1$, то он подменит m_2 на m_1. В противном случае он подменит m_2 на m_3. В первом случае противник не добьется успеха, поскольку $e_2^{-1}(m_1) = 0$, а во втором случае ему "повезет". Итак, шансы на успех подмены оцениваются числом 1/2. Легко убедиться в том, что и шансы на успех имитации оцениваются тем же числом.

Мы пришли к выводу, что для данного кода аутентификации шансы на успех активной атаки оцениваются числом 1/2. Чтобы уменьшить шансы противника на успех, мы должны увеличить размеры матрицы кодирования и ее структуру. Например, для матрицы кодирования

$$\mathrm{W} = \begin{pmatrix} H & 0 & 0 & T & 0 & 0 \\ T & 0 & 0 & 0 & H & 0 \\ H & 0 & 0 & 0 & 0 & T \\ 0 & H & 0 & T & 0 & 0 \\ 0 & T & 0 & 0 & H & 0 \\ 0 & H & 0 & 0 & 0 & T \\ 0 & 0 & T & H & 0 & 0 \\ 0 & 0 & H & 0 & T & 0 \\ 0 & 0 & T & 0 & 0 & H \end{pmatrix}$$

шансы противника уменьшаются до 1/3. Убедитесь в этом самостоятельно. Это рассуждение показывает, какой ценой достигается стойкость.

Заметим в заключение, что последний пример системы аутентификации совсем не похож на предыдущие, поскольку в нем в явном виде не используется контрольная сумма. Более того, передаваемое сообщение не позволяет противнику однозначно определить состояние источника. Такие коды аутентификации называются кодами аутентификации *с секретностью*. Предыдущие примеры относятся к классу кодов аутентификации *без секретности*.

Контрольные вопросы

1. Какие атаки противника относят к активным?
2. Можете ли вы привести примеры активных атак из реальной жизни?
3. Что понимается под аутентификацией информации?
4. Что такое криптографическая контрольная сумма (КС) сообщения?
5. Чем отличается MDC от MAC?
6. Что такое хеш-функция?
7. Что такое криптографическая хеш-функция?
8. Почему хеш-функции, используемые в системах аутентификации, должны быть криптографическими?
9. Выясните, является ли функция из примера 3 криптографической хеш-функцией.
10. Что такое шифрсистема?
11. Чем отличается шифрсистема с открытым ключом от шифрсистемы с секретным ключом?
12. Как образуется цифровая подпись сообщения?
13. Какие способы построения КС сообщения вам известны?
14. Как оценивается стойкость системы аутентификации информации?
15. Какие системы аутентификации называются практически стойкими, теоретически стойкими?
16. Какие достоинства и недостатки практически стойких (теоретически стойких) систем аутентификации вы можете назвать?
17. Пусть C и C' — КС произвольных сообщений M и M', вычисленные с помощью системы CBC-MAC при использовании секретного ключа k. Пусть N — произвольный блок. Проверьте, что сообщения $M||N$ и $M'||(N \oplus (C \oplus C'))$ имеют одинаковые КС, полученные при использовании того же ключа k.
18. Что такое код аутентификации?
19. Можете ли вы привести пример кода аутентификации?
20. Можете ли вы построить пример кода аутентификации для передачи результатов бросания монеты, для которого шансы на успех активной атаки противника оцениваются числом 1/4?

“1С:Школа. Информатика, 10 класс” для обучения в школе и дома

Мария Барская,
Игорь Кузора,
фирма “1С”, Москва

Неотъемлемым элементом современного обучения является использование электронных образовательных ресурсов в учебном процессе. Предмет “Информатика и ИКТ” не является исключением. Образовательный комплекс “1С:Школа. Информатика, 10 класс” составлен на основе комплекта профессиональных электронных образовательных ресурсов, подходит как для обучения дома самостоятельно, так и для работы в классе под руководством учителя. Структура курса соответствует УМК под ред. Н.Д. Угриновича и включает следующие разделы: компьютер и программное обеспечение, информация и представление различных видов информации в компьютере, основы алгебры логики и логические элементы компьютера, основы алгоритмизации и технологии программирования, компьютерные сети и Интернет, информатизация общества.

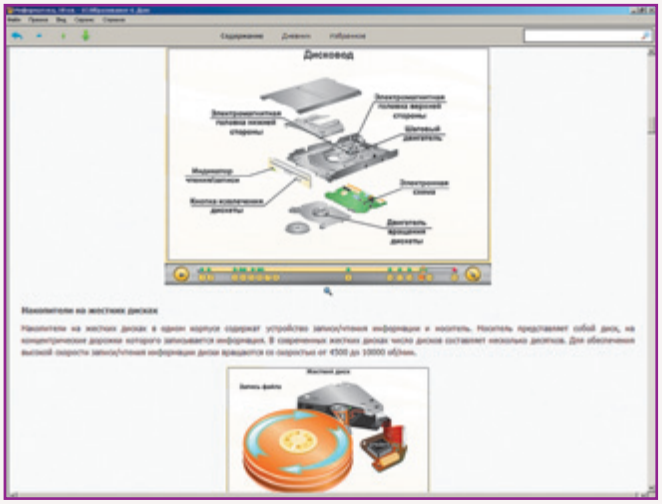


Рис. 1

Образовательный комплекс (ОК) поможет учителю подобрать иллюстративный материал к уроку, подготовить практические задания для выполнения в классе и тесты для проведения контрольной работы, провести практические и контрольные занятия по подготовке к ЕГЭ по темам 10-го класса, подобрать дополнительный материал и провести элективное занятие. Использование ОК предоставит ученику возможность самостоятельно изучить новый материал, на практике отработать новые умения, выполнить контроль знаний, подготовиться к ЕГЭ по информатике по темам 10-го класса и расширить свой кругозор.

В состав авторского и редакторского коллектива ОК вошли преподаватели СУНЦ МГУ Т.С. Богомолова и И.Н. Фалина, сотрудники Института информатизации образования РАО Д.Ю. Усенков и Л.Л. Босова, сотрудники лаборатории обучения информатике Института содержания и методов обучения РАО С.А. Бешенков и Э.В. Миндзаева, специалисты по “1С:Предприятию” В.Э. Бояркин и В.В. Рыбалка.

Каждый раздел ОК разбит на параграфы, в состав которых входят теоретический материал, подборка практических заданий и заданий для самостоятельного решения, тесты для подготовки к ЕГЭ.

Все учебные материалы ОК являются кроссбраузерными и кроссплатформенными. При желании пользователь может экспортировать их и использовать в учебных целях во внешних средах. С помощью системы программ “1С:Образование 4.1. Школа 2.0” их можно использовать для обучения в сетевом режиме и в ОС семейства GNU/Linux.

Теоретический материал (*рис.* 1) представляет собой гипертекст, снабженный анимациями, фотографиями, рисунками и коллажами. Содержание параграфа является исчерпывающим по той или иной теме, дополнительный материал для изучения курса не требуется. Теория сопровождается разобранными примерами, облегчающими понимание нового материала и характеризующими его практическую значимость. Материалы параграфа содержат ссылки

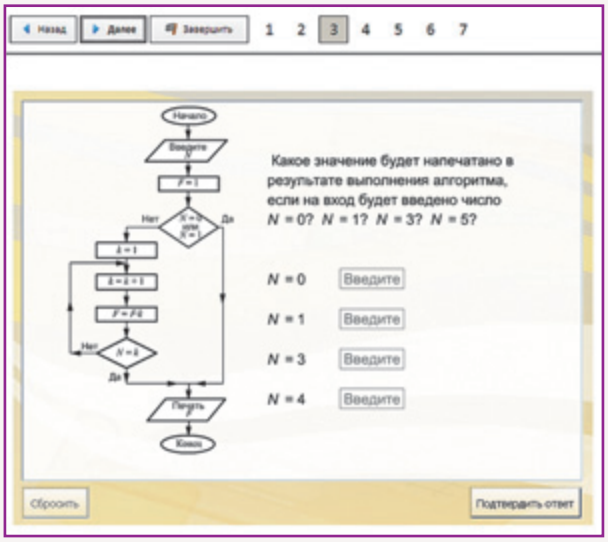


Рис. 2

на биографические справки об ученых, внесших вклад в развитие данной темы.

В состав параграфов включены подборки практических заданий, представляющие собой наборы тестов с решениями и подсказками для закрепления нового материала, и задания для самостоятельного решения (*рис.* 2). Прохождение тестов возможно в произвольном порядке и с неограниченным количеством попыток. После прохождения подборки тестов можно обратиться к системе статистики о времени прохождения, количестве тестовых заданий, количестве попыток и результате по каждому тесту, сводном результате по тестам. В тестах представлены задания различных типов: выбор элементов из множественного списка, установление соответствий между объектами, перенесение элементов в контейнер и ввод ответа в активное поле. Учитель может использовать тестовые задания для проведения тестирования в классе и в качестве домашнего задания на закрепление материала. Используя проектор с экраном, учитель может предложить классу решить тот или иной тест путем группового обсуждения, тем самым повысив интерес и внимание учеников к процессу обучения на уроке.

Для подготовки к ЕГЭ в состав образовательного комплекса включены контрольные измерительные материалы (КИМ) типов А и В, разработанные по актуальным спецификациям ЕГЭ (*рис.* 3). Задания включены в параграфы, изучение материала которых необходимо для их правильного прохождения. Таким образом, к ЕГЭ можно начать готовиться по мере прохождения нового материла. Входящие в состав комплекса задания по ЕГЭ можно разделить на две группы: практические задания с подробными решениями и контрольные задания. Тесты могут быть полезны и учителю для организации классного тренинга по подготовке к ЕГЭ.

Одним из центральных мест в курсе информатики является раздел “Основы алгоритмизации. Технологии программирования”. Он нацелен на развитие алгоритмического мышления учащегося, которое важно практически для всех видов деятельности в современном мире. В ОК “1С:Школа. Информатика, 10 класс” практический материал по алгоритмике и программированию представлен на четырех языках: Pascal, Basic, C и встроенный язык программирования системы программ “1С:Предприятие 8.2”.

Преимущество примеров на встроенном языке “1С:Предприятие 8.2” в том, что все программы записываются на русском языке сродни школьному алгоритмическому языку, однако в отличие от последнего являются исполняемыми программой. Так выглядит пример кода операции сравнения двух чисел:

```
Процедура СравнениеЧисел(Команда)
 Перем a;
 Перем b;
  ВвестиЧисло(a,"Введите a");
  ВвестиЧисло(b,"Введите b");
 Если a < b Тогда Сообщить("a < b");
  ИначеЕсли a > b Тогда Сообщить("a > b")
    Иначе Сообщить("a = b");
 КонецЕсли;
 КонецПроцедуры
```

При этом не требуются дополнительные пояснения, что означают те или иные выражения, функции или команды, которые записываются в других программах с помощью латиницы.

Другими преимуществами преподавания информатики с примерами в “1С:Предприятие 8.2” являются простота и удобство разработки решений различного уровня сложности, а также режим справки на русском языке. На одном диске с образовательным комплексом поставляется учебная версия платформы “1С:Предприятие 8.2”, что позволяет на практике выполнить все примеры и задания.

В 2011/2012 учебном году фирма “1С” приглашает учителей-практиков к совместной апробации данного раздела курса. Апробационным площадкам предоставляется лицензия на образовательное учреждение, проводится подробная консультация по работе с образовательным комплексом. По вопросам участия в апробации обращайтесь по адресу *obr@1c.ru*.

На CD-приложении к журналу и на сайте представлена более подробная информация о порядке апробации, об образовательном комплексе, его демонстрационная версия и методические видеоматериалы (*http://obr.1c.ru/info10*).

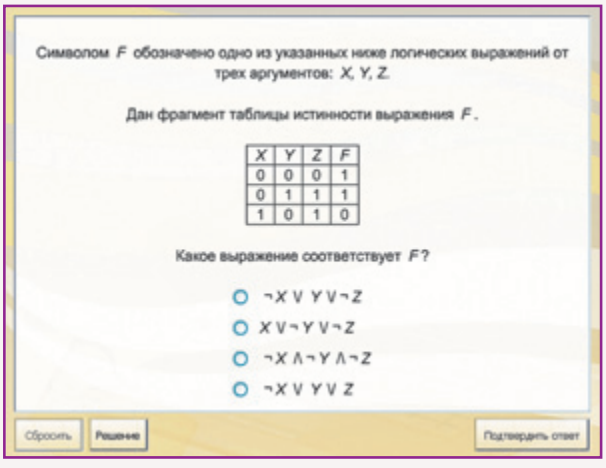
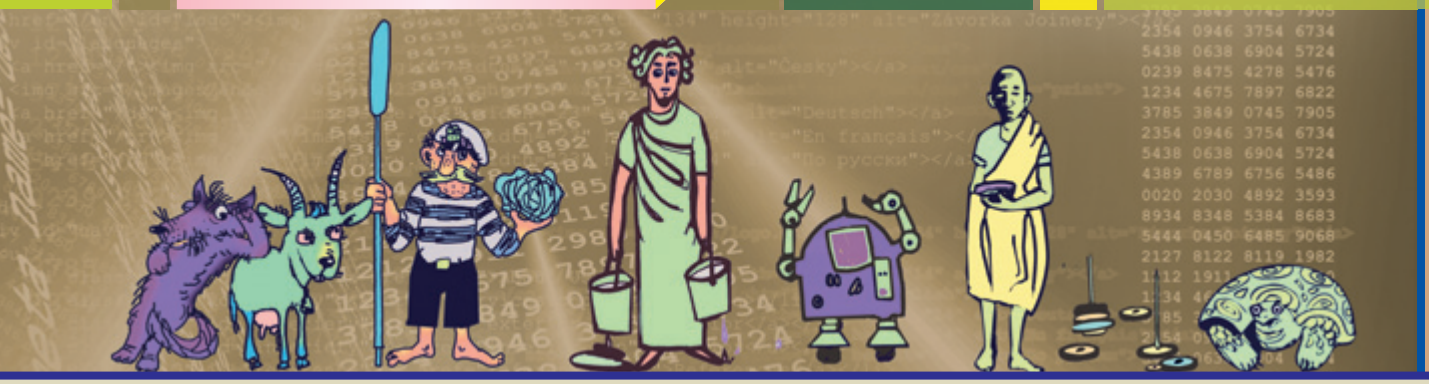


Рис. 3



ИСТОРИЯ ИНФОРМАТИКИ

Как считали на счетах?

Д.М. Златопольский

Вряд ли найдется читатель, не знакомый со старым устройством для вычислений — счетами. А знаете ли вы, как считать на счетах? О том, как это делалось (а иногда — и делается), будет рассказано в данной статье.

Но сначала немного истории. Прообразом известных нам счетов был старинный русский счетный прибор — так называемый “дощаной счет”. Он состоял из одного или двух неглубоких ящиков, поперек которых натянуты веревки или проволоки с надетыми на них косточками[1]. На верхних проволоках было надето 9 или 10 косточек, на нескольких нижних — от 1 до 6. Ящики разделялись перегородкой на два отделения. Перегородка могла быть по всей высоте прибора или только для нескольких нижних неполных рядов.

На *рис.* 1 показана реконструкция одного из вариантов дощаного счета, представленная в Музее истории вычислительной техники гимназии № 1530 г. Москвы (*www.museum.ru/m2744*).

Наиболее старыми русскими счетами являются счеты середины XVII века, хранящиеся в Государственном историческом музее в Москве, которые имеют четыре счетных поля для неполных рядов (*рис.* 2). В конце XVII в. счеты утратили неполные ряды, но имели еще два счетных поля. “Современный” вид счеты приняли в XVIII веке.

Счеты — старинный прибор, который долгое время пользовался популярностью у разных народов. При этом в разных странах счеты выглядели по-разному (*рис.* 4 и 5).

Несколько слов о конструкции русских счетов (см. *рис.* 3). Несколько верхних рядов имели 10 костяшек[2] и использовались для откладывания целых чисел. Имелся также неполный ряд, обычно из четырех костей, под которым находились 2 или 3 полных ряда. Последние использовались для откладывания копеек (при денежных расчетах) или десятых, сотых и тысячных долей чисел (в общем случае), то есть неполный ряд являлся, выражаясь математически, “разделителем целой и дробной частей”.

Для наглядности вычислений костяшки счетов имели двухцветную окраску. Пятая и шестая костяшки на каждой проволоке окрашивались в более темный (черный) цвет, остальные — в

Рис. 1. Дощаной счет

Рис. 2. Счеты с четырьмя полями

Рис. 3. “Современные” счеты

[1] Как писалось в одном древнем руководстве, “А изряднее вместо вервей проволока медная или железная” [1].
[2] Подумайте, пожалуйста, над тем, почему используется (и использовался) этот термин.

Рис. 4. Китайские счеты (суаньпань). Отложено число 627

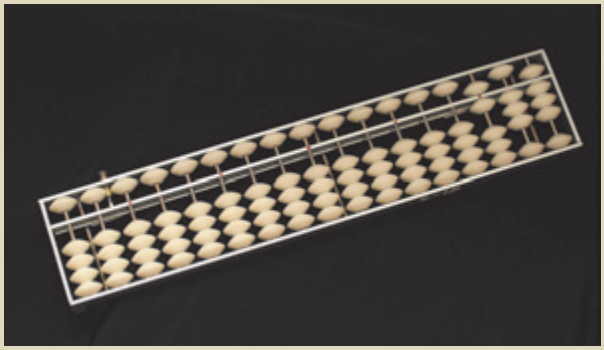


Рис. 5. Японские счеты (соробан). Отложено число 638

светлый. Двухцветная окраска костяшек облегчала откладывание цифр, поскольку, согласитесь, что, например, четыре светлых костяшки и две темных на левой стороне быстрее определяются как цифра 6, чем отсчитывание шести одноцветных костяшек. Такая окраска позволяла также очень быстро определить, какое число набрано на счетах.

Вычисления на счетах

Дальнейшие разъяснения, связанные с методикой вычислений на счетах, проведем с использованием их схематических изображений.

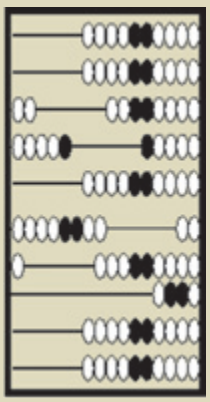


Рис. 6. Отложено первое слагаемое (25 081)

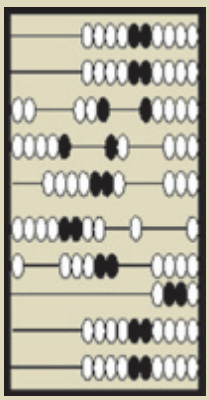


Рис. 7. Набрано второе слагаемое (32 715)

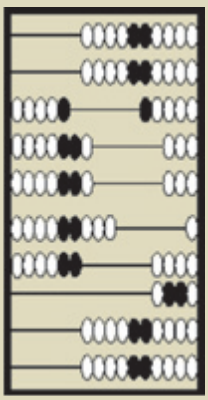


Рис. 8. Получен результат сложения (57 796)

На *рис.* 6 отложено число 25 081. Для сложения с ним числа 32 715 цифры последнего последовательно набирались в каждом разряде из оставшихся “неиспользованными” в соответствующем ряду (*рис.* 7), и набранные костяшки перемещались влево к “цифрам” первого слагаемого; в итоге слева получалось число-результат (*рис.* 8).

Если в каком-то разряде после сложения оказывалось 10 костяшек, то все они сбрасывались (возвращались в исходное, правое, положение), а в старшем разряде добавлялась одна костяшка (см. *рис.* 9 и 10). Ясно, что если при этом в старшем разряде было отложено 9, то в нем проводились аналогичные действия.

Рис. 9. К 7 добавляется 3

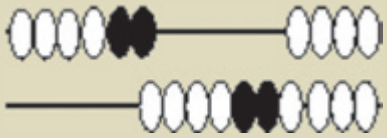


Рис. 10. Результат сложения

Определенная проблема возникает, когда в каком-то разряде сумма цифр превышает 10. Например, такая ситуация показана на *рис.* 11 — к 8 костяшкам на нижней проволоке нужно прибавить 6. Непосредственно добавить нужное количество, как это делалось выше, в данном случае нельзя.

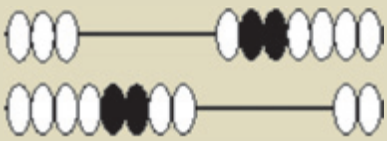


Рис. 11. Небольшая проблема

Как быть? Здесь возможны два варианта решения.

Можно добавить одну костяшку в старшем разряде, а в разряде “с проблемой” — вычесть 4 (10 – 6).

Можно также поступить так:

1) на нижней проволоке добавить имеющиеся справа две костяшки;

2) поскольку в результате в этом разряде получилось 10, то сбросить их все, а в старшем разряде добавить 1 (см. выше);

3) на нижней проволоке добавить еще 6 – 2 = = 4 костяшки.

Теперь о вычитании. Когда в каждом разряде количество уже отложенных костяшек (цифр уменьшаемого) не меньше, чем количество костяшек, которые нужно снять (соответствующих цифр вычитаемого), то задача решается просто — снимаем необходимые костяшки. Пример вычитания из числа 57 796 значения 32 765 показан на *рис.* 12.

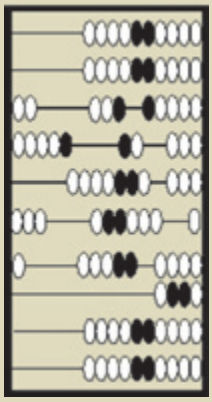


Рис. 12. Вычитание без проблем

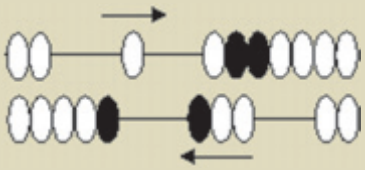


Рис. 13. Вычитание с заимствованием единицы

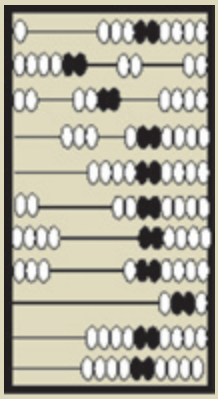


Рис. 14. Пример умножения

Как поступить в случае, когда в каком-то разряде количество отложенных костяшек меньше количества костяшек, которые надо снять при вычитании, вы, конечно, уже догадались — надо "заимствовать единицу" из старшего разряда. Например, когда в нижнем разряде (см. *рис.* 13) требовалось вычесть 7, снималась одна костяшка в старшем (верхнем) разряде, в нижнем — добавлялись 3.

"А умножение?" — спросите вы. Да, на счетах можно было и умножать даже многозначные числа на многозначные, правда, заменяя умножение многократным сложением, а затем суммируя частные произведения с учетом весомости разряда. Например, при умножении 81 на 23 результат умножения 81 на 2 (162) откладывался на неиспользуемых верхних проволоках, а произведение 81 на 3 (243) добавлялось к нему как к числу 1620 (см. *рис.* 14). Конечно, для умножения многозначных чисел требовалось большое число проволочек или еще одни счеты (вспомните о нескольких счетных полях на предшественниках счетов).

Задание для самостоятельной работы

1. Найдите на счетах сумму чисел:
а) 250 863 и 136 014;
б) 84 392 и 54 386.
2. Найдите на счетах разность чисел:
а) 67 842 и 33 022;
б) 56 193 и 38 155.
3. Отложите на счетах число 1967. Прибавьте к нему 1967. К сумме прибавьте 1967. Еще дважды прибавьте 1967. Из полученного результата вычтите 1967. Вычтите еще 4 раза. Сколько получилось? ☺ Если 0, то можете считать себя научившимся проводить вычисления на счетах.

В заключение заметим, что в музее, который упоминался в статье, представлен доклад известного русского математика В.Я. Буняковского, который он сделал 14 февраля 1876 г. на заседании физико-математического отделения Императорской академии наук. В докладе описывается изобретенное ученым вычислительное устройство, которое автор назвал "самосчеты". В.Я. Буняковский пишет: "Мы едва ли ошибемся, утверждая, что ни один из существующих арифметических снарядов, и даже вероятно из тех, которые со временем будут придуманы, не вытеснят из всеобщего у нас употребления простых русских счетов". Прав или нет оказался ученый, судить вам, уважаемый читатель…

Литература

1. *Апокин И.А.*, *Майстров Л.Е.* Развитие вычислительных машин. М.: Наука, 1974.

ЗАДАЧНИК

Ответы, решения, разъяснения к заданиям, опубликованным в газете "В мир информатики" № 159 ("Информатика" № 5/2011)

1. Задача "Семь кошельков"

Напомним, что необходимо было ответить на вопрос, как разложить по семи кошелькам 127 рублевых монет, чтобы любую сумму от 1 до 127 рублей можно было бы выдать, не открывая кошельков (то есть вместе с кошельками)?

Решение

1. Десятичное число 127 в двоичной системе счисления выглядит так: 1111111, а все числа, меньшие 127, в этой системе состоят (естественно) из единиц и нулей.

2. Любое двоичное число от 1 до 1111111 можно получить, складывая двоичные числа 1, 10, 100, 1000, 10 000, 100 000 и 1 000 000 (убедитесь в этом!). Эти числа есть двойка в степени 0, 1, 2, 3, 4, 5 и 6, т.е. десятичные числа 1, 2, 4, 8, 16, 32 и 64.

Значит, для набора любой суммы от 1 до 127 рублей надо иметь семь кошельков: в первый кошелек надо положить одну рублевую монету, во второй — две, в третий — четыре, в четвертый — восемь, в пятый — шестнадцать, в шестой — тридцать две, в седьмой — шестьдесят четыре. Если этим кошелькам присвоить условные номера, соответственно, 1, 2, …, 7, тогда любую сумму от 1 до 127 рублей можно получить следующим образом. Нужно перевести эту сумму в двоичную систему счисления, после чего взять те кошельки, номера которых равны номерам тех разрядов двоичной записи суммы, где представлена цифра 1 (разряды следует нумеровать справа налево, начиная с 1). Например, чтобы

выдать сумму в 109 рублей, надо отдать кошельки с номерами 7, 6, 4, 3 и 1, так как $109_{10} = 1101101_2$.

Ответы прислали:

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

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

— Буханов Василий, Григорьев Кирилл и Юхтенко Илья, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Мнацаканян Ашот, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Согомонян Серине, Воронежская обл., поселок Каменка, средняя школа № 1 им. Героя Советского Союза В.П. Захарченко, учитель **Старикова М.Е.**;

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

— Ядзевичюс Стас, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**

2. Задача “Три министра”

Ответ. Министр иностранных дел Китая — откровенный, России — осторожный, США — скрытный.

Ответы представили:

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

— Антонов Василий и Черепов Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

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

— Владимирова Юлия, Моронцова Анастасия, Семенов Дмитрий, Тимофеев Анатолий и Яковлев Анатолий, основная школа села Именево, Республика Чувашия, Красноармейский р-н, учитель **Тимофеева И.А.**;

— Губаева Зульфия, Республика Татарстан, Актанышский р-н, село Актаныш, средняя школа № 2, учитель **Гилязова Г.М.**;

— Исакова Анна, Круглякова Мария и Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Моисеев Игорь и Шафиева Алина, Республика Башкортостан, г. Стерлитамак, школа № 17, учитель **Орлова Е.В.**;

— Осипова Дарья, Республика Карелия, г. Сегежа, школа № 5, учитель **Меньшиков В.В.**;

— Согомонян Серине, Воронежская обл., поселок Каменка, средняя школа № 1 им. Героя Советского Союза В.П. Захарченко, учитель **Старикова М.Е.**

3. Головоломки “Слова на диагоналях”

Ответы

Головоломка № 1

Слова в строках: СЕАНС, ВРЕМЯ, СХЕМА, АНОДЫ, АЛЬФА. Слово на диагонали — СРЕДА.

Головоломка № 2

Слова в строках: АРХИВ, СДВИГ, КАРТА, БУФЕР, СЕАНС. Слово на диагонали — АДРЕС.

Правильные ответы представили:

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

— Антонов Василий и Черепов Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

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

— Буханов Василий, Григорьев Кирилл и Юхтенко Илья, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Денисов Петр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Согомонян Серине, Воронежская обл., поселок Каменка, средняя школа № 1 им. Героя Советского Союза В.П. Захарченко, учитель **Старикова М.Е.**

4. Статья “Второе дело об украденном компьютере”

Напомним, что необходимо было определить, кто из подозреваемых — Трусище, Балбесище или Бывалыч — украл компьютер, если Балбесище заявил, что компьютер украл Трусище, а в ходе судебного заседания выяснилось, что компьютер украл только один из трех подозреваемых и что именно он дал правдивые показания.

Решение

Рассмотрим возможные варианты.

1. Допустим, компьютер украл Трусище. Тогда получается, что Балбесище сказал правду. Но этого не может быть, так как по условию тот, кто сказал правду, сам и украл.

2. Предположим, что преступление совершил Балбесище. Но тогда тоже получается противоречие — сказанное им утверждение не могло быть правдивым.

Следовательно, компьютер украл Бывалыч.

Правильные ответы прислали:

— Антонов Василий и Черепов Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

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

— Васильева Юлия, Республика Башкортостан, г. Стерлитамак, школа № 17, учитель **Орлова Е.В.**;

— Владимирова Юлия, Моронцова Анастасия, Семенов Дмитрий, Тимофеев Анатолий и Яковлев Анатолий, основная школа села Именево, Республика Чувашия, Красноармейский р-н, учитель **Тимофеева И.А.**;

— Денисов Петр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Хотеев Сергей и Янкин Ян, Москва, гимназия № 1530, учитель **Шамшев М.В.**

Юрий Базылев представил также решение задач, опубликованных в газете "В мир информатики" № 157 ("Информатика" № 3/2011): "Каким днем недели было 20-е число" и "Три сотрудника", а также ребусов по информатике и числового ребуса с символами "*".

Спасибо всем!

Четыре сообщения

Для пяти букв русского алфавита заданы их двоичные коды (для некоторых букв — из двух бит, для некоторых — из трех). Эти коды представлены в таблице:

В	К	А	Р	Д
000	11	01	001	10

Из четырех полученных сообщений в этой кодировке:

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

— только одно прошло без ошибки и может быть корректно декодировано. Какое?

Как проехать из A в B?

Таблица стоимости перевозок устроена следующим образом: числа, стоящие на пересечениях строк и столбцов таблиц, означают стоимость проезда между соответствующими соседними станциями. Если пересечение строки и столбца пусто, то станции не являются соседними.

Для какой таблицы из приведенных ниже выполняется условие: "Минимальная стоимость проезда от станции **A** до станции **B** не больше 6"? (Стоимость проезда по маршруту складывается из стоимостей проезда между соответствующими соседними станциями.)

1)

	A	B	C	D	E
A			3	1	
B			4		2
C	3	4			2
D	1				
E		2	2		

2)

	A	B	C	D	E
A			3	1	1
B			4		
C	3	4			2
D	1				
E	1		2		

3)

	A	B	C	D	E
A			3	1	
B			4		1
C	3	4			2
D	1				
E		1	2		

4)

	A	B	C	D	E
A				1	
B			4		1
C		4		4	2
D	1		4		
E		1	2		

Три брата[3]

Три брата — Ваня, Саша и Коля — учились в разных классах одной школы. Ваня был не старше Коли, а Саша не старше Вани. Назовите имена старшего из братьев, среднего и младшего.

Любительница мороженого

Ирина любит мороженое с фруктами. В кафе был выбор из таких вариантов:

а) пломбир с орехами;
б) пломбир с бананами;
в) пломбир с черникой;
г) шоколадное с черникой;
д) шоколадное с клубникой.

В четырех вариантах Ирине не нравились или тип мороженого, или наполнитель, а в одном варианте ей не нравились ни мороженое, ни наполнитель. Она попросила приготовить из имеющихся продуктов порцию по своему вкусу.

Какое же мороженое и с какими фруктами любит Ирина?

[3] Задача предназначена для учащихся начальной школы и учеников 5–7-х классов.

Задача из старинной рукописи

М.А. Цайгер,
кандидат технических наук

В старинной русской рукописи конца XVI — начала XVII века приведена следующая задача:

"Смотри сице: Ѯ пуд воску, цена ему ѱ рублев ѕ алтына с полуденгой. Что да(с)т ѻ пудов?

В ней есть слова, которые сейчас отсутствуют в русском языке, например, "сице" — в словаре В.И. Даля читаем: "Сице — *нареч., церк.*, так, тако, сяк, таким образом, сим способом", то есть "смотри сице" значит, как говорят иногда, "смотри сюда".

Цена — это не то, что понимается сегодня как стоимость единицы товара, например, килограмма или пуда, а вся уплаченная сумма за некоторое количество килограммов или пудов.

Вы уже догадались, что выражение "Что даст… пудов?" в данном случае означает — "Сколько надо уплатить за указанное количество?".

Для решения задачи надо знать, какая связь между денежными единицами того времени — рублем, алтыном и денгой (см., например, [1]).

Кроме того, обращаю внимание на то, 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[4].

Числовые значения использованных букв установите, ознакомившись со статьей [2] или найдя соответствующую информацию в Интернете или в других источниках.

Литература

1. *Цайгер М.А.* Запись чисел в решетках. / "В мир информатики" № 142 ("Информатика" № 8/2010).

2. Буквенная цифирь. / "В мир информатики" № 125 ("Информатика" № 7/2009).

GAMES.EXE

Компьютерная игра "Крестики-нолики"

Н.А. Насыров,
учитель информатики и математики
средней школы села Сайраново,
Республика Башкортостан

Часть 2. Снабжаем компьютер интеллектом

В первой части статьи [1]) мы разработали компьютерную программу, моделирующую известную игру крестики-нолики, участниками которой были человек и компьютер. Ход компьютера выбирался во вспомогательном алгоритме (процедуре) **`Ход_компьютера`**. В нем случайным образом вычислялись координаты клетки, и если она была свободна, то компьютер ставил туда "крестик". Конечно, при такой тактике вероятность выигрыша компьютера практически нулевая. Но ведь в другие игры компьютер может выигрывать у человека! Неужели это нельзя сделать при игре в крестики-нолики? Ответ — можно. Но как научить компьютер "думать"? Для этого надо в программе предусмотреть не случайный, а какой-то "осмысленный" выбор клетки.

Очень часто "мышление компьютера" — это моделирование мышления человека. Давайте так же поступим и мы.

Подумаем, как играют в крестики-нолики. Очевидно, что игрок анализирует возможные ходы и оценивает их эффективность, выбирая наиболее удачный. Следовательно, мы в первую очередь должны решить, как машина может оценить удачность того или иного хода. Так как клеток на поле только 9, создать иерархию ценностей возможных ходов совсем несложно.

1. Высший приоритет должен быть в ситуации, когда ход в данную клетку может победно завершить игру. Она возникает, если в соответствующем ряду компьютером уже закрыты две клетки (*рис.* 1).

Рис. 1

Как бы поступил человек в этой ситуации? — Он бы поставил свой "крестик" в левую верхнюю клетку. Так же должен пойти и компьютер.

2. Второй по приоритету будет комбинация двух знаков противника и свободной клетки в одном ряду (*рис.* 2). Она означает, что на следующем ходу игра может закончиться проигрышем компьютера. Поэтому, если нельзя победить на этом ходу, опасную позицию следует закрыть.

Рис. 2

Следовательно, компьютер должен пойти в клетку с координатами (1, 2).

[4] Напомним также, что системы счисления, для записи чисел в которых используются *буквы* некоторого алфавита, называют "алфавитными". К числу таких систем, кроме буквенной цифири ("славянская алфавитная система"), относились также ионийская (греческая), финикийская и другие. — **Прим. ред.**

3. На третьем месте в иерархии находится позиция, когда в ряду стоит один знак компьютера и две клетки свободны (*рис.* 3). Значит, потенциально, через два хода можно победить.

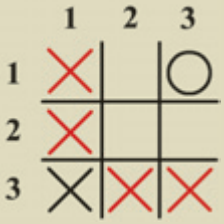


Рис. 3

В таких позициях "крестик" можно поставить в клетки с координатами (1, 1), (2, 1), (3, 2) или (3, 3).

4. Четвертое место в иерархии необходимо отдать комбинации "знак противника плюс две свободные клетки" (*рис.* 4), так как она грозит возможным проигрышем через два хода — в обоих рядах соперник, ставящий "нолик", может выиграть через два хода.

Рис. 4

Поэтому компьютеру необходимо закрыть одну из четырех клеток с красными "крестиками".

5. Пятое место занимает строка из пустых клеток. Здесь до победы далеко, но она вполне возможна.

6. Последняя по ценности позиция соответствует сочетанию знака противника, знака компьютера и пустой клетки (*рис.* 5). Такая комбинация не может закончить игру ни победой, ни проигрышем, поэтому завершать заполнение такого ряда нужно в последнюю очередь.

Рис. 5

Ход компьютера в клетку с координатами (3, 3) не принесет выигрыш его сопернику.

Итак, когда наступает ход компьютера, он должен проверить "ценность" каждой пустой клетки и выбрать клетку с наибольшей ценностью.

При этом нужно учитывать, что клетка всегда принадлежит нескольким (от двух до четырех) рядам, и ее значимость в разных рядах может быть различной. Поэтому характеристикой ценности той или иной клетки должна быть сумма ценностей ее положений в каждом из рядов. После того как эффективность хода в каждую свободную клетку будет вычислена, необходимо определить, какой из них соответствует наибольшее значение эффективности. Именно туда и должен быть поставлен знак.

Проиллюстрируем алгоритм выбора хода компьютером на конкретных примерах.

Прежде всего вспомним, что клетка с "ноликом" имеет значение 10, с "крестиком" компьютера — 100, пустая клетка — 1. Для рассмотренных ценностей позиций составим таблицу оценок ряда клеток в зависимости от суммы значений в клетках (см. внизу страницы).

Рассмотрим ситуацию, показанную на *рис.* 1. Идет игра, ходит компьютер. У него есть 5 свободных клеток, куда он может поставить "крестик". Исследуем каждую из них.

1. Клетка с координатами (1, 1) принадлежит первой строке, первому столбцу и главной диагонали:

— сумма значений первой строки:

$$1 + 1 + 10 = 12,$$

оценка 20 (см. таблицу внизу);

— сумма значений первого столбца:

$$1 + 1 + 1 = 3,$$

оценка 7;

— сумма значений главной диагонали:

$$1 + 100 + 100 = 201,$$

оценка 1000.

Сумма всех оценок рядов:

$$20 + 7 + 1000 = \mathbf{1027}.$$

2. Клетка с координатами (1, 2) принадлежит первому ряду и второму столбцу:

— сумма значений первой строки:

$$1 + 1 + 10 = 12,$$

оценка 20;

— сумма значений второго столбца:

$$1 + 100 + 1 = 102,$$

оценка 50.

Сумма всех оценок рядов: $20 + 50 = \mathbf{70}$.

Таблица 1

Оценки рядов клеток

Ситуация в ряду	Сумма значений в ряду	Оценка ряда
Два знака компьютера в ряду — высший приоритет (*рис.* 1)	201 (100 + 100 + 1)	1000
Два знака соперника в ряду (*рис.* 2)	21 (10 + 10 + 1)	100
Знак компьютера и две пустые клетки (*рис.* 3)	102 (1 + 1 + 100)	50
Знак соперника и две пустые клетки (*рис.* 4)	12 (1 + 1 + 10)	20
Пустой ряд	3 (1 + 1 + 1)	7
В ряду есть и знак компьютера, и знак его соперника (*рис.* 5)	111 (1 + 10 + 100)	3

3. Клетка с координатами (2, 1) принадлежит второй строке и первому столбцу:

— сумма значений второй строки:

$$1 + 100 + 10 = 111,$$

оценка 3;

— сумма значений первого столбца: 1 + 1 + 1 = = 3, оценка 7.

Сумма всех оценок рядов: 3 + 7 = **10**.

4. Клетка с координатами (3, 1) принадлежит третьей строке, первому столбцу и побочной диагонали:

— сумма значений третьей строки:

$$1 + 1 + 100 = 102,$$

оценка 50;

— сумма значений первого столбца:

$$1 + 1 + 1 = 3,$$

оценка 7;

— сумма значений побочной диагонали:

$$1 + 100 + 10 = 111,$$

оценка 3.

Сумма всех оценок рядов: 50 + 7 + 3 = **60**.

5. Клетка с координатами (3, 2) принадлежит третьей строке и второму столбцу.

— сумма значений третьей строки:

$$1 + 1 + 100 = 102,$$

оценка 50;

— сумма второго столбца:

$$1 + 100 + 1 = 102,$$

оценка 50.

Сумма всех оценок рядов: 50 + 50 = **100**.

Имея суммарные оценки рядов каждой клетки, можно сделать вывод, что клетка с координатами (1, 1) имеет наивысшую оценку, равную 1027. Именно туда компьютер поставит “крестик” — и это будет “разумный” ход!

Для лучшего понимания работы схемы поиска клетки с наивысшей оценкой самостоятельно проведите аналогичный анализ ситуации, показанной на *рис.* 2. Убедитесь в том, что компьютер, “подумав”, действительно поставит “крестик” именно в клетку с координатами (1, 2).

Смоделируем все описанные действия в нашей программе.

Для выбора лучшего хода компьютера необходимо следующее.

1. Рассмотреть все клетки поля. Это можно сделать, применив вложенный оператор цикла:

```
нц для i от 1 до 3
 нц для j от 1 до 3
   если поле[i, j] = 1
    то
      …
   все
 кц
кц
```

Величина *i* — номер строки, *j* — номер столбца.

Ясно, что если клетка занята, то определять ее ценность не нужно. Поэтому, прежде чем проводить оценку, проверяем, является ли клетка пустой (`поле[i, j] = 1`).

2. Для каждой клетки, используя табл. 1, подсчитать сумму оценок всех рядов, которым она принадлежит. Как отмечалось, оценка ряда зависит от суммы значений в каждой его клетке.

Эти суммы мы определяли в первой части статьи во вспомогательном алгоритме (процедуре) `Проверка`. Такие же расчеты мы должны провести и здесь в зависимости от значений координат *i* и *j*:

```
выбор
  при i = 1 и j = 1: сумма1 := поле[1, 1] + поле[1, 2] + поле[1, 3]
                     сумма2 := поле[1, 1] + поле[2, 1] + поле[3, 1]
                     сумма3 := поле[1, 1] + поле[2, 2] + поле[3, 3]
  при i = 1 и j = 2: сумма1 := поле[1, 1] + поле[1, 2] + поле[1, 3]
                     сумма2 := поле[1, 2] + поле[2, 2] + поле[3, 2]
  при i = 1 и j = 3: сумма1 := поле[1, 1] + поле[1, 2] + поле[1, 3]
                     сумма2 := поле[1, 3] + поле[2, 3] + поле[3, 3]
                     сумма4 := поле[3, 1] + поле[2, 2] + поле[1, 3]
  при i = 2 и j = 1: сумма1 := поле[2, 1] + поле[2, 2] + поле[2, 3]
                     сумма2 := поле[1, 1] + поле[2, 1] + поле[3, 1]
  при i = 2 и j = 2: сумма1 := поле[2, 1] + поле[2, 2] + поле[2, 3]
                     сумма2 := поле[1, 2] + поле[2, 2] + поле[3, 2]
                     сумма3 := поле[1, 1] + поле[2, 2] + поле[3, 3]
                     сумма4 := поле[3, 1] + поле[2, 2] + поле[1, 3]
  при i = 2 и j = 3: сумма1 := поле[1, 3] + поле[2, 3] + поле[3, 3]
                     сумма1 := поле[2, 1] + поле[2, 2] + поле[2, 3]
  при i = 3 и j = 1: сумма1 := поле[3, 1] + поле[3, 2] + поле[3, 3]
                     сумма2 := поле[1, 1] + поле[2, 1] + поле[3, 1]
                     сумма4 := поле[3, 1] + поле[2, 2] + поле[1, 3]
  при i = 3 и j = 2: сумма1 := поле[3, 1] + поле[3, 2] + поле[3, 3]
                     сумма2 := поле[1, 2] + поле[2, 2] + поле[3, 2]
  при i = 3 и j = 3: сумма1 := поле[3, 1] + поле[3, 2] + поле[3, 3]
                     сумма2 := поле[1, 3] + поле[2, 3] + поле[3, 3]
                     сумма3 := поле[1, 1] + поле[2, 2] + поле[3, 3]
 все
```

После расчета сумм мы должны найти оценку каждого ряда и сумму оценок всех рядов. Оценку каждого ряда удобно провести с использованием вспомогательной функции Оценка:

```
алг цел Оценка(цел сумма) |сумма – сумма значений для некоторого ряда
нач
 выбор
  при сумма = 201: знач := 1000
  |Значение функции (см. табл. 1)
  при сумма = 21: знач := 100
  при сумма = 102: знач := 50
  при сумма = 12: знач := 20
  при сумма = 3: знач := 7
  при сумма = 111: знач := 3
 все
кон
```

С использованием этой функции сумма оценок всех рядов клетки рассчитывается так:

```
сумм_оценка := Оценка(сумма1) +
      Оценка(сумма2) + Оценка(сумма3)+
      Оценка(сумма4)
```

3. Найти координаты клетки, у которой сумма оценок всех рядов максимальная. Эта задача похожа на поиск координат элемента двумерного массива с максимальным значением:

```
макс := 0
нц для i от 1 до 3
 нц для j от 1 до 3
   если массив[i, j] > макс
     то
       макс := массив[i, j]
       |Новое значение максимума
       i_макс := i
       |Запоминаем координаты элемента
       j_макс := j
       |с максимальным значением
   все
 кц
кц
```

В нашем случае:

```
…
если сумм_оценка > макс
 то
  макс := сумм_оценка
  x := i |Запоминаем координаты клетки
  y := j
  |с максимальной суммарной оценкой
все
```

После выбора компьютером оптимального хода в клетку с координатами (x, y) помещается “крестик”:

```
поле[x, y] := 100
```

счетчик ходов увеличивается на 1:

```
n := n + 1
```

и, как и в первом варианте программы, проверяется, не окончилась ли игра. Для этого, напомним, используется функция Проверка.

Весь алгоритм Ход_компьютера с “осмысленным” выбором клетки для размещения своего символа выглядит так:

```
алг Ход_компьютера
нач цел i, j, x, y
 вывод нс, "Ход компьютера – "
 вывод "для продолжения нажмите
        любую клавишу"
 |Приостановка программы до нажатия
 |любой клавиши
 нц
 кц при клав <> 0
 макс := 0
 нц для i от 1 до 3
  нц для j от 1 до 3
   если поле[i, j] = 1
    то
     |Рассчитываем значения сумма1,
     |сумма2, сумма3 и сумма4
      сумма3 := 0; сумма4 := 0
      выбор
        при i = 1 и j = 1:
        сумма1 := поле[1, 1] +
        поле[1, 2] + поле[1, 3]
        сумма2 := поле[1, 1] +
        поле[2, 1] + поле[3, 1]
        сумма3 := поле[1, 1] +
        поле[2, 2] + поле[3, 3]
        … (см. выше)
     все
     |Определяем сумму оценок всех
     |рядов клетки
      сумм_оценка := Оценка(сумма1) +
                     Оценка(сумма2) +
      Оценка(сумма3) + Оценка(сумма4)
     |Сравниваем ее со "старым"
     |максимальным значением
     если сумм_оценка > макс
       то
         макс := сумм_оценка
         x := i
         |Запоминаем координаты клетки
         y := j
        |с максимальной суммарной оценкой
     все
   все
  кц
 кц
 |Помещаем в соответствующий элемент
 |массива поле значение 100
 поле[x, y] := 100
 n := n + 1 |Увеличиваем счетчик ходов
 |Проверяем, не окончилась ли игра
 если Проверка(x, y) = -100
 |Выиграл компьютер
   то
     конец_игры := да
     n := -100
 все
 если n = 9 |Ничья
  то
   конец_игры := да
 все
кон
```

Вспомогательные алгоритмы `Пустое_поле`, `Вывод_поля`, `Ход_игрока`, `Проверка` и `Определение_и_объявление_результата`, а также основная программа останутся без изменений.

Вот и все. Наберите программу и поиграйте в игру. Как и для ее первого варианта, рассмотрите при отладке три случая: ваша победа, ваш проигрыш и ничья. Желаю успехов!

Литература

1. *Насыров Н.А.* Компьютерная игра “Крестики-нолики” / “В мир информатики” № 166 (“Информатика” № 12/2011).

От редакции. Разработанную программу, пожалуйста, пришлите в редакцию. Кроме того, подумайте над тем, как вместо общей схемы программы, использованной в статье (в ее первой части):

```
Подготовка начальной ситуации
на игровом поле (в массиве поле)
Вывод игрового поля
Определение начинающего игру
если начинает компьютер
   то
   нц пока не конец_игры
      Ход_компьютера
      Вывод игрового поля
      если не конец_игры
        то
          Ход_игрока
      Вывод игрового поля
       все
       кц
   иначе
    нц пока не конец_игры
        Ход_игрока
        Вывод игрового поля
        если не конец_игры
          то
           Ход_компьютера
           Вывод игрового поля
        все
   кц
все
Вывод результата игры
```

— применить схему не с двумя операторами цикла, а только с одним.

Все приславшие правильную программу и/или схему будут награждены дипломами. Срок представления программ — ноябрь 2011 года.

ЛОГИКА

Еще раз о задаче С4 из демонстрационного варианта ЕГЭ по информатике и ИКТ 2011 года

В “Информатике” № 10 за 2011 год была рассмотрена методика решения задачи С4 из демонстрационного варианта Единого государственного экзамена по информатике и ИКТ 2011 года [1]. Напомним, что в целом ее суть заключалась в следующем. Известно количество вхождений каждой из цифр от 0 до 9 в строку символов. Необходимо определить, можно ли из такого набора цифр (переставив их при необходимости) получить на экране симметричное число (которое читается одинаково как слева направо, так и справа налево). Ведущих нулей в числе быть не должно, исключение — число 0, запись которого содержит ровно один ноль. Если требуемое число составить невозможно, то программа должна вывести на экран слово “NO”. А если возможно, то в первой строке следует вывести слово “YES”, а во второй — искомое симметричное число. Если таких чисел несколько, то программа должна выводить максимальное из них. Например, пусть на вход подаются следующие символы:

Do not 911 to 09 do.

В данном случае программа должна вывести

YES

91019

В результате анализа было определено условие, при котором симметричное число получить нельзя (а значит, и при котором можно). Это условие получилось достаточно объемным, но зато логичным, полностью соответствующим сделанным рассуждениям.

Прежде всего, симметричное число составить нельзя, когда число цифр с нечетным количеством вхождений в набор больше 1. Кроме того, это нельзя сделать, когда:

1) число нулей — четное, других цифр с четным числом вхождений нет, а ненулевая цифра (она нечетная) единственная, например, когда имеются цифры 0, 3, 0, 0, 0;

2) есть только четное число одних нулей (0, 3, 0, 0, 0);

3) есть только нечетное число нулей, кроме случая одного нуля (см. условие);

4) цифр в строке символов нет вообще.

Соответствующее условие выглядит так:

```
если (всего_нечет_кол + всего_чет_кол = 0) или (всего_нечет_кол > 1) или
     (всего_чет_кол = 1 и mod(кол_цифр[0], 2) = 0 и кол_цифр[0] > 0)
      и (всего_нечет_кол = 0 или (всего_нечет_кол = 1 и кол_цифр[цифра_неч] = 1))
     или (всего_нечет_кол = 1 и mod(кол_цифр[0], 2) = 1  и
         кол_цифр[0] > 1 и всего_чет_кол = 0)
  то
    вывод нс, "NO"
  иначе
   вывод нс, "YES", нс
   |Выводим максимальное симметричное число
   …
```

где *всего_нечет_кол* — количество цифр с нечетным числом вхождений в строку;
всего_чет_кол — то же, с четным числом вхождений;
цифра_неч — цифра с нечетным числом вхождений;
кол_цифр — массив с информацией о количестве вхождений каждой из цифр.

Первые три величины можно определить, обработав информацию в последнем массиве.

Попробуем упростить его. Давайте сформулируем условие, при котором симметричное число составить *можно*.

Во-первых, это можно сделать, когда количество цифр с нечетным числом вхождений равно нулю и при этом есть цифры с четным числом вхождений, но вариант, состоящий только из четного количества нулей, должен быть исключен. Соответствующее условие можно записать так:

```
всего_нечет_кол = 0 и всего_чет_кол > 0 1 и не (всего_чет_кол = 1 и кол_цифр[0] > 0)
```

Во-вторых, симметричное число можно получить, если количество цифр с нечетным числом вхождений равно 1, кроме случаев наборов типа (0, 0, 0, 0, 0) и (0, 3, 0, 0, 0). Соответствующее условие имеет вид:

```
всего_нечет_кол = 1 и
  не (                          |1
    всего_чет_кол = 0           |Цифр с четным числом вхождений нет
     и mod(кол_цифр[0], 2) = 1 |Количество нулей нечетное
     и кол_цифр[0] > 1          |и не равно 1
    или                         |2
      всего_чет_кол = 1         |Есть только одна цифра с четным числом вхождений
      и
      mod(кол_цифр[0], 2) = 0 и кол_цифр[0] > 0  |и это – цифра 0
      и всего_нечет_кол  = 1 |Есть только одна цифра с нечетным числом вхождений
      и кол_цифр[цифра_неч]  = 1 |и это число равно 1
     )
```

С учетом сделанных рассуждений фрагмент программы, относящийся к выводу ответа, будет следующим:

```
если всего_нечет_кол = 0 и всего_чет_кол > 0
       и не (всего_чет_кол = 1 и кол_цифр[0] > 0)
     или всего_нечет_кол = 1 и
     не (всего_чет_кол = 0 и mod(кол_цифр[0], 2) = 1 и кол_цифр[0] > 1
        или всего_чет_кол = 1 и mod(кол_цифр[0], 2) = 0 и кол_цифр[0] > 0
        и всего_нечет_кол = 1 и кол_цифр[цифра_неч] = 1)
 то
   вывод нс, "YES", нс
   |Выводим максимальное симметричное число
    …
 иначе
  вывод нс, "NO"
все
```

Подсчет показывает, что общее число простых условий равно 13, что совпадает с числом применительно к варианту, приведенному в [1].

Попробуем уменьшить это количество, идя, так сказать, от числа нулей в заданной строке. Сформулируем условие, при котором симметричное число составить *можно*. Рассмотрим 3 случая.

1. Когда число нулей нечетное (`mod(кол_цифр[0], 2) = 1`). В этом случае других цифр с нечетным числом вхождений быть не должно (`всего_нечет_кол = 1`) и при этом количество нулей должно быть равно 1 (`кол_цифр[0] = 1`) или в наборе должны присутствовать цифры с четным числом вхождений

(всего_чет_кол > 0). Примеры таких допустимых наборов: 0, 4 0 4, 4 0 0 0 4, примеры недопустимых — 0 0 0, 4 0 4 2, 4 0 0 0 4 5.

2. Когда нулей в наборе нет (кол_цифр[0] = 0). Для формирования симметричного числа в наборе должны присутствовать цифры (всего_нечет_кол + всего_чет_кол > 0) и при этом количество цифр с нечетным числом вхождений не должно быть больше 1 (всего_нечет_кол <= 1). Примеры соответствующих наборов: 4 4 2 4 4 2, 4 4 4 2 4, 7 7 7 7 7.

3. Когда число нулей четное (mod(кол_цифр[0], 2) = 0) и они есть (кол_цифр[0] > 0). В этом случае должна быть только одна цифра с нечетным числом вхождений, но такой цифры должно быть больше одной (всего_нечет_кол = 1 **и** кол_цифр[цифра_неч] > 1), в противном случае в наборе должны присутствовать другие цифры с четным числом вхождений (всего_чет_кол > 0) и при этом количество цифр с нечетным числом вхождений не должно быть больше 1 (всего_нечет_кол <= 1).

Все условие, при котором симметричное число составить можно, выглядит так:

```
mod(кол_цифр[0], 2) = 1 и
всего_нечет_кол = 1 и
(кол_цифр[0] = 1 или всего_чет_кол > 0)
или
кол_цифр[0] = 0 и
всего_нечет_кол + всего_чет_кол > 0 и
всего_нечет_кол <= 1
или
mod(кол_цифр[0], 2) = 0 и
кол_цифр[0] > 0 и
(всего_нечет_кол = 1 и
кол_цифр[цифра_неч] > 1 или
всего_чет_кол > 0 и
всего_нечет_кол <= 1)
```

Общее число простых условий в нем — все то же 13. Поистине “роковое” ☺ число…

В заключение еще раз напомним, что в демонстрационном варианте ЕГЭ 2011 года представлен более компактный вариант условия. Понятность приведенного там варианта предлагаем оценить читателям.

Задания для самостоятельной работы

1. Запишите сложное условие, при котором симметричное число составить нельзя, рассмотрев 3 варианта значения величины *всего_неч*:

1) *всего_неч* > 1;
2) *всего_неч* = 1;
3) *всего_неч* = 0.

2. На языке программирования, который вы изучаете, разработайте программу решения обсуждаемой задачи (в одном из трех рассмотренных вариантов и с использованием условия по п. 1) и проверьте ее работу на следующих наборах цифр:

1) 0;
2) 00;
3) 000;
4) 001;
5) 20201;
6) 20200;
7) 1;
8) 12;
9) 212;
10) 440005;
11) 4400;
12) 440012,

а также в случае, когда цифр нет.

3. Исследуйте, можно ли, используя законы логики:

1) уменьшить число простых условий в любом из трех приведенных составных условий;

2) получить один из четырех вариантов сложного условия, преобразовав любой другой.

Результаты, пожалуйста, присылайте в редакцию.

Литература

1. *Златопольский Д.М.* Задача С4 из демонстрационного варианта ЕГЭ по информатике и ИКТ 2011 года. / “Информатика” № 10/2011.

ЛИЧНОСТИ

Юлиус Эдгар Лилиенфельд, или Как делаются гениальные изобретения

(к 130-летию со дня рождения Ю.Э. Лилиенфельда)

Александр Нитусов

Известно, что основа всей цифровой техники — транзисторы. Но мало кто знает, как произошел переход от ламповых вычислительных машин к транзисторным…

Мощный импульс, данный промышленной революцией науке и образованию, породил и продолжает вызывать к жизни огромную массу открытий и изобретений. Ученый и изобретатель давно уже заняли в обществе почетное место — для нас это естественно и привычно. Из “одиночек Средневековья” они выросли в огромную хорошо организованную “армию”; открытия буквально “поставлены на поток”, и все же, при каждом появлении чего-то нового, “дающего зеленую улицу” множеству вещей, ранее невозможных, имя создателя вызывает невольное восхищение.

Правда, часто возникает вопрос: “А кого, собственно, считать создателем?”. Большинство современных изобретений очень сложны: чтобы их сделать, нужен труд многих, иногда сотен, людей, создававшаяся долгие годы “пирамида” разнообразных знаний, а еще длинная цепь предыдущих открытий, приборы, технологии и т.п. Мы это понимаем, но… все равно хотим видеть

конкретного человека. Так уж мы устроены…

Юлиус Лилиенфельд

В XV веке адмирал Чжень Хэ мог позволить себе изобрести океанский корабль с водонепроницаемыми переборками, просто взглянув на распиленный ствол бамбука (как известно, разделенный перегородками на секции), гарантировав морякам безопасность и открыв тем самым новую страницу в истории мореплавания. Однако уже и он опирался на обширные знания о кораблях и навигации.

Дальше — сложнее. В XIX веке Чарльз Бэббидж и Семен Корсаков смогли изобрести вычислительные машины и устройства со вводом данных на перфокартах потому, что французский инженер Жаккар уже внедрил перфокарты для автоматического управления ткацкими станками. Перфокарты он придумал не сам, а взял из разработки Вокансона — знаменитого создателя механических автоматов, развившего, в свою очередь, идею перфорированных стальных лент, управляющих музыкальными автоматами на башенных часах, и т.д.

Открытие вытекает из открытия, идея из многих предыдущих — сказать, где, собственно, "начинается чье-то начало", не легче, чем найти начало какой-то одной струйки в общем течении реки. Причинно-следственный закон — в классическом виде.

Так что же, в наше время изобретателей нет?

Есть, и немало. Но мы всегда склонны отдать пальму первенства именно *принесшему нам готовый продукт* — будь то знание, устройство или что-то еще, — а не тем "гигантам", на чьих плечах он стоит[5]. Так люди благодарили Прометея, принесшего им огонь, хотя не он его изобрел.

Однако стоит отдать дань благодарности и тем, чье имя редко появляется на первом плане, но без чьих открытий результат был бы невозможен. Речь, конечно, не о тысячах предшественников, а о том, кто "открыл дорогу".

К таким "скромным героям" относится и доктор Лилиенфельд. Именно с его патента началась история полупроводникового транзистора — "этапного элемента электроники".

Юлиус Эдгар Лилиенфельд родился 18 апреля 1881 г. во Львове — столице Галиции, входившей в те годы в состав Австро-Венгерской империи. Его отец Сигизмунд Лилиенфельд, доктор права, работал адвокатом, Юлиус посещал Львовское высшее реальное училище — не гимназию, но учебное заведение, дававшее аттестат зрелости.

Высшее образование он получил уже не во Львове. В зимнем семестре 1899 г. Юлиус поступил в Политехнический институт в Шарлоттенбурге в Берлине. Однако, проучившись один год и пройдя производственную практику на одном из берлинских заводов, Лилиенфельд решил, что его призвание — естественные науки, и в ноябре 1900 г. перевелся в университет Фридриха-Вильгельма, там же в Берлине, где до 1904 г. усердно посещал лекции по философии, математике, химии и физике. Среди его профессоров были такие известные ученые, как Макс Планк и президент Германского физического общества Эмиль Варбург. Последний значительно повлиял на формирование научных интересов (а может быть, и судьбы) Лилиенфельда. Варбург читал лекции по самым передовым темам: кинетической теории газов, электропроводности, разрядам в газах, тепловому излучению, ферромагнетизму и фотохимии. Среди его учеников были физики Джеймс Франк — будущий (1925 г.) Нобелевский лауреат, и Роберт Вихард Поль — будущий конструктор первого экспериментального полупроводникового транзистора. Помимо этого, Лилиенфельд посещал лекции Варбурга и работал под его руководством в Физическом институте в 1903–1904 гг. В 1904 г. он получил диплом университета, в феврале 1905 г. сдал с отличием диссертационный экзамен, а затем и защитил диссертацию. Варбург и Планк входили в экзаменационную комиссию.

В зимний семестр 1905–1906 гг. он начал работать в Институте физики при Лейпцигском университете и вскоре опубликовал статьи по тлеющему разряду и по ртутной лампе низкого давления с высокой выходной мощностью. Исследования тлеющего разряда в институте не велись и явно были продолжением берлинского опыта самого Лилиенфельда. Поддержка профессора Отто Винера — заведующего кафедрой экспериментальной физики — обеспечила исследованиям популяризацию и финансирование от Саксонского королевского научного общества. В 1909 г. он включился в новую актуальную тему института — производство жидкого водорода, началось развитие дирижаблей. Ему повезло — первая машина для сжижения газа дала утечку и взорвалась, жертв не было, но, чтобы не рисковать, институт построил отдельную лабораторию, где Лилиенфельд стал ведущим работником. Он разработал компрессор, выделявший кислород и водород из воздуха, и запатентовал его в Англии (UK Patent № 22930, 1912 г.), а затем некоторое время сотрудничал с графом Фердинандом фон Цеппелином, проектируя дирижабли.

Тогда же он занялся исследованиями электрических разрядов в глубоком вакууме, в 1910 г. представил докторскую диссертацию, а вскоре начал преподавать в университете и параллельно исследовать и разрабатывать рентгеновские трубки. В марте 1915 г. он подал заявку еще на один английский патент на модернизацию трубок, а летом того же года его лекции отменили — началась война, и студентов мобилизовали. Сам он продолжил исследования и значительно модернизировал дозиметрию в рентгенологии. Одновременно он

[5] Исаак Ньютон: "Я видел дальше других потому, что стоял на плечах гигантов".

совершенствовал методы достижения максимально глубокого вакуума в трубках, где производились разряды в десять тысяч вольт. Несмотря на военное положение, в августе 1916 г. Саксонское министерство целевым порядком выделило средства и предоставило ему должность профессора в Лейпцигском университете (весьма редкий случай!).

Обычно Лилиенфельд всегда старался довести свои научные результаты до практической реализации и запатентовать их. Его теоретические научные публикации касаются принципиальных вопросов и не так многочисленны. Может быть, именно поэтому его имя и многочисленные разработки известны в основном специалистам. Тем более необычно выглядит получение им нескольких патентов на устройства, основанные на полевой электронной эмиссии, появившиеся как продукт исследований тока в вакуумных трубках и не имевшие никакого практического значения. В заявке на патент 1922 г. он прямо пишет: "…устройство, испускающее электроны… только за счет электромагнитного поля" (автоэлектронный эффект). Автоэлектронное устройство с контактом, покрытым очищенным веществом, который испускает поток электронов и очищается бомбардировкой электронами, значится в его американском патенте (US Patent № 1,578,045). В 1928 г. он запатентовал устройство с полевой эмиссией, названное им "электронный триод". Устройство можно было использовать для генерирования рентгеновских лучей и как генератор переменного тока (US Patent № 1,979,275).

С 1922 г. он проводил в США все больше времени, а в 1926 г. и вовсе оставил свою должность в Лейпциге. В Германии уже начались националистические выступления, а он познакомился в Нью-Йорке с американкой Беатрис Гинзбург, на которой и женился 2 мая 1926 г. В 1927 г. он получил официальный статус эмигранта и несколько лет работал по временным контрактам. Начал с полутора лет работы в Нью-Йоркском университете, затем в Ergon Research Lab. В 1928 г. он наконец получил стабильную должность в фирме Amrad Inc., производителе радиодеталей в Малдене (Массачусетс). Там он занялся оксидными алюминиевыми пленками и их применением в электролитических конденсаторах. По некоторым свидетельствам, это и привело его к созданию транзистора.

Лилиенфельд считал, что подача напряжения на слабо проводящий материал будет изменять его проводимость, вызывая усиление электрических колебаний. Однако создать работающий полевой транзистор не удалось, поскольку в то время еще не нашлось подходящего материала, необходимого для такого прибора. Именно поэтому большинство ученых того времени усиленно изобретали более сложный биполярный транзистор.

Известно, что немецкий физик Вальтер Шоттки в 1923 г. написал развернутую статью о холодных и горячих электронных разрядах. Вероятно, она оказала влияние на работу Лилиенфельда, т.к. в 1926 г. он писал своему учителю Отто Винеру, что: "…выпрямитель, дающий ток в 3А, можно использовать для зарядки аккумуляторов и применять во многих других низковольтных устройствах".

В 1926 г., когда он еще жил в Бруклине, Лилиенфельд получил патент на выпрямитель переменного тока — первое из разработанных им твердотельных устройств, за которым последовали три патента на модуляцию проводимости твердого вещества поперечным электромагнитным полем. Первый, заявленный в 1926 г. (№ 1745175, выдан в 1928 г.), полевой транзистор — комбинация металла и полупроводника, второй следует из первого, заявлен в 1928 г. (выдан в 1933 г., № 1900018) и представляет собой полевой транзистор, работающий в режиме обеднения. Третий патент — усилитель электрического тока (№ 1877140, выдан в 1932 г.), описывает транзистор на металлической основе (полупроводниковый транзистор "полупроводник/металл") и транзистор Шоттки с p–n-коллектором. Это — первый известный патент, описывающий транзистор с металлической базой.

Пытался ли изобретатель изготовить действующую модель своего транзистора, неизвестно. В любом случае это устройство не могло бы исправно, стабильно работать из-за отсутствия высококачественных полупроводников и поэтому было неприменимо в радиотехнической практике. Тем более обидно, поскольку Лилиенфельд опередил свое время буквально на 15 лет.

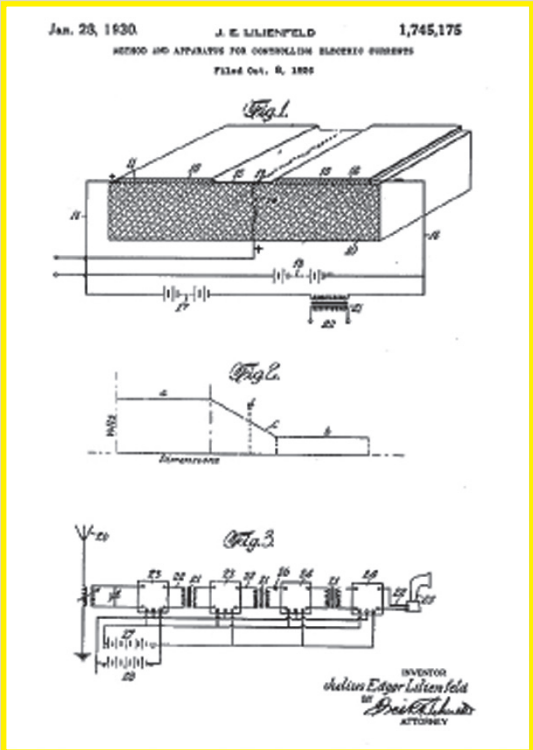

Jan. 28, 1930. J. E. LILIENFELD 1,745,175

Fig. 1.

Fig. 2.

Fig. 3.

INVENTOR
Julius Edgar Lilienfeld
BY
ATTORNEY

Патент Ю.Лилиенфельда на полевой транзистор

В 1938 г. известный немецкий физик Роберт Вихард Поль вместе с Рудольфом Хильшем (из Гёттингенского университета) независимо от Лилиенфельда создал первый в мире действующий "твердотельный" транзистор, с солью — арсенидом галлия — в качестве полупроводника. Разумеется, он был экспериментальным и работал нестабильно. Трудно сказать, знал ли Поль о

патентах Лилиенфельда, но в любом случае они — питомцы одной научной школы.

В 1947 г. в тех же США группа исследователей из Bell Laboratories, руководимая тремя будущими Нобелевскими лауреатами — У.Шокли, У.Бардином и Дж. Браттейном, после нескольких лет напряженных экспериментов создала действующее полупроводниковое устройство, демонстрировавшее относительно устойчивый транзисторный эффект. В США это было объявлено созданием первого в мире транзистора, положившего начало новой эпохе в электронике.

Правда, успех тут же омрачился довольно громким скандалом. Обладавший "непростым" характером Шокли заявил опешившим коллегам, что эта разработка основывалась на его патенте, что он был формальным руководителем группы, поэтому и приоритет, и патент на транзистор должны принадлежать исключительно ему. Начались разбирательства, в ходе которых патентные эксперты Bell Laboratories объявили, что патент Шокли вообще практически повторяет патент Ю.Лилиенфельда и поэтому никаких патентов и приоритетов на полевой транзистор ни он, ни кто другой вообще не получит. О самом Лилиенфельде никто уже не вспоминал, хотя он в это время еще довольно активно работал.

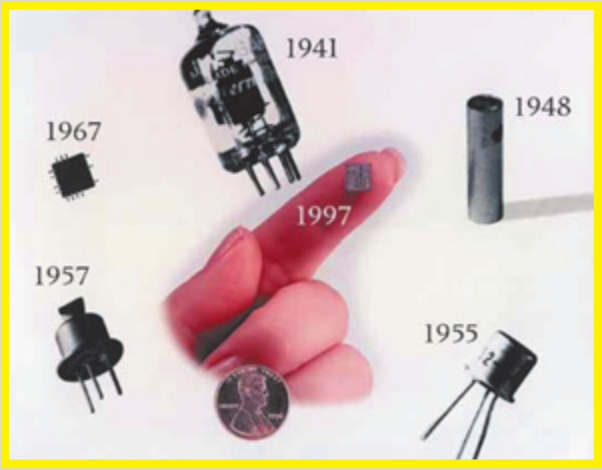
Однако в начале 1950-х гг. все трое все-таки получили Нобелевские премии за изобретение транзистора (уже не полевого). Добросовестный Дж. Браттейн отметил в своей нобелевской лекции, что их работа строилась на опыте Лилиенфельда и Шоттки и на теории проводимости в полупроводниках советских ученых А.Иоффе и И.Френкеля в Ленинграде и В.Давыдова в Киеве.

А в 1949 г. московская студентка Сусанна Мадоян за несколько месяцев своей дипломной практики в Зеленограде под руководством научного сотрудника А.Красилова и при помощи лаборанта Петелина самостоятельно сделала то же самое [1].

Однако это были лишь первые шаги, а серийное производство более-менее надежных транзисторов и в СССР, и в США началось уже в середине 1950-х гг.

За свою жизнь Лилиенфельд получил в общей сложности 15 германских и 60 американских патентов, права на некоторые из них ему пришлось отстаивать до последнего дня. Он скончался 28 августа 1963 г. в США, в г. Шарлотта Амалия, в возрасте 82 лет.

Хотя изобретение Лилиенфельдом транзистора и разработка рентгеновских трубок пришлись на неудачное время (он "чуть-чуть" опередил появление необходимых материалов и технологий) и не принесли ему известности, он, безусловно, заслуживает уважения как талантливый ученый и плодовитый изобретатель.



Эволюция транзистора

Литература

1. *Александр Нитусов*. Сусанна Мадоян — создатель первого советского триода. / "В мир информатики" № 144 ("Информатика" № 10/2010).

"ЛОМАЕМ" ГОЛОВУ

Чайнворд "Соточка"

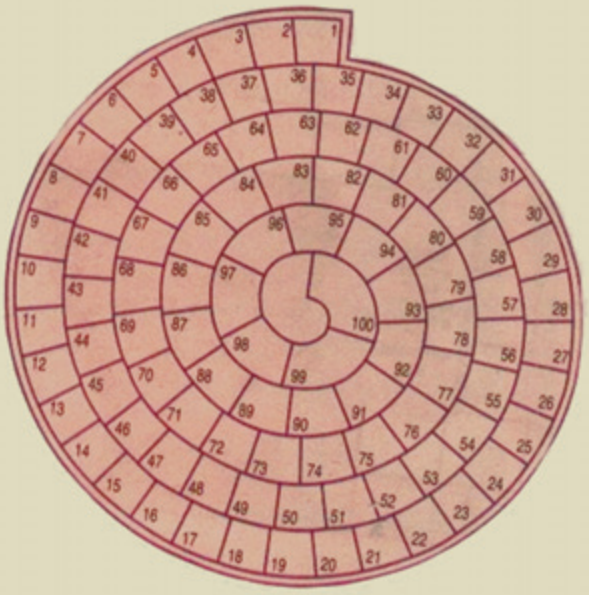
Приведенную ниже головоломку подготовил Денис Гаврилов, ученик Кезской средней школы № 1, Удмуртская Республика, учитель **Ветошкина Н.В.** В нее нужно вписать слова, соответствующие приведенным ниже комментариям. Решите, пожалуйста, ее.

В ответах приводите только слова (изображение приводить не нужно).

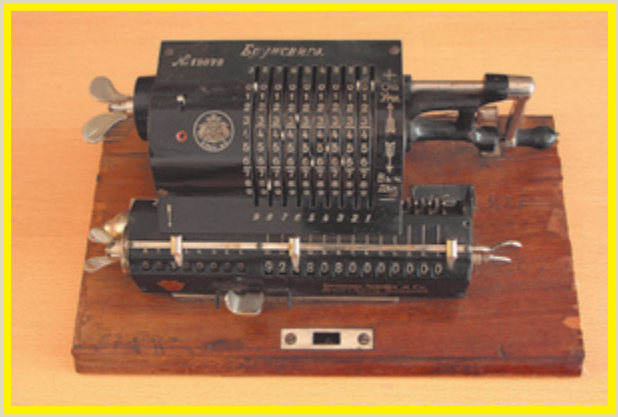
1–11. Небольшое графическое изображение — условное обозначение объекта управления на экране.

11–13. Язык программирования, названный в честь первой женщины-программиста.

13–21. Язык программирования, в командах которого используются обозначения в удобочитаемой форме.



21–26. Страна, в которой было изобретено механическое вычислительное устройство — арифмометр, показанное на рисунке:



26–31. Элемент электронной таблицы.

31–35. В текстовом редакторе Microsoft Word — текст, набранный пользователем до нажатия клавиши Enter.

35–38. Алгоритмическая конструкция, позволяющая многократно повторять одну и ту же последовательность команд.

38–44. Фамилия женщины, упомянутой выше.

44–49. Графический объект в компьютерной графике, как правило, перемещающийся по экрану. Его название происходит от английского слова, в переводе на русский язык обозначающего — “эльф” (сказочный персонаж).

49–54. Поток данных в компьютерных сетях.

54–63. Устройство ввода информации.

63–73. Логическая организация, структура и ресурсы компьютера, которые может использовать программист.

73–79. Устройство, осуществляющее преобразование представления и скорости передачи информации между ЭВМ и внешним устройством.

79–83. Совокупность точек графического изображения.

83–88. Структурный элемент числа в позиционных системах счисления.

88–93. Элемент измерительного, сигнального, регулирующего или управляющего устройства системы, преобразующий контролируемую величину в удобный, например для использования в компьютере, сигнал.

93–97. Один из первых языков программирования.

97–100. Язык программирования, основанный на функциональной парадигме программирования.

Числовой ребус с “ЭММОЙ”

Решите, пожалуйста, числовой ребус, в котором, как обычно в таких головоломках, одинаковыми буквами зашифрованы одинаковые цифры, разными буквами — разные цифры:

МА * МА = ЭММА

Четыре сообщения

Для пяти букв русского алфавита заданы их двоичные коды (для некоторых букв — из двух бит, для некоторых — из трех). Эти коды представлены в таблице:

В	К	А	Р	Д
000	11	01	001	10

Из четырех полученных сообщений в этой кодировке:

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

— только одно прошло без ошибки и может быть корректно декодировано. Какое?

Пять бусинок

Для составления цепочек используются бусины, помеченные буквами: *M*, *N*, *O*, *P*, *S*. В середине цепочки стоит одна из бусин *M*, *O*, *S*. На третьем месте — любая гласная, если первая буква согласная, и любая согласная, если первая гласная. На первом месте — одна из бусин *O*, *P*, *S*, не стоящая в цепочке в середине.

1. Какая из перечисленных цепочек создана по этому правилу?

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

2. Сколько вариантов цепочек можно составить согласно указанным требованиям? (Ответ получите, не выписывая все возможные варианты.)

Ответы с обоснованием, пожалуйста, присылайте в редакцию.

ВНИМАНИЕ! КОНКУРС

Конкурс № 86

В качестве задания этого конкурса предлагаем решить задачу из старинной рукописи, опубликованную в рубрике “Задачник”.

Ответ отправьте в редакцию до 20 сентября 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.