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

ИНФОРМАТИКА

13 сентября – День программиста! Ура!

Неплохой вопрос для детей, например, на каком-нибудь интеллектуальном конкурсе: почему День программиста отмечается в России 13-го или 12 сентября?
И почему когда-то 13-го, а когда-то 12-го?
(Ответ на второй странице обложки.)

электронная версия журнала
дополнительные материалы
в Личном кабинете
на сайте
www.1september.ru

издательский дом 1september.ru
Первое сентября
сентябрь 2014
ИНФОРМАТИКА Подписка на сайте www.1september.ru или по каталогу «Почта России»: 79066 — бумажная версия, 12684 — CD-версия

НА ОБЛОЖКЕ

13 сентября –
День программиста!

Итак, вы уже догадались, почему? Или просто знали? Согласно Указу № 1034, который подписал 11 сентября 2011 г. Президент Дмитрий Медведев, День программиста отмечается в 256-й день года. Соответственно, в обычные годы он приходится на 13 сентября, а в високосные — на 12-е. Кстати, от идеи учреждения такого праздника, которую предложил Дмитрий Мендрелюк (глава издательского дома "Компьютерра"), до Указа Президента прошло не так много времени. Всего 13 лет. ☺

В НОМЕРЕ

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии
от Издательского дома "Первое сентября"

Уважаемые подписчики бумажной версии журнала!

Дополнительные материалы к номеру и электронная версия журнала находятся в вашем Личном кабинете на сайте www.1september.ru.

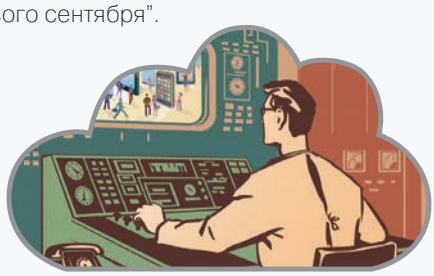
Для доступа к материалам воспользуйтесь, пожалуйста, кодом, вложенным в № 7–8/2014.

Срок действия кода: с 1 июля по 31 декабря 2014 года.

Для активации кода:

- зайдите на сайт www.1september.ru;
- откройте Личный кабинет (создайте, если у вас его еще нет);
- введите код доступа и выберите свое издание.

Справки: podpiska@1september.ru или через службу поддержки на портале "Первого сентября".



ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ

Презентации и исходные файлы к статьям номера

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ
по каталогу "Почта России": 79066 — бумажная версия, 12684 — электронная версия

http://inf.1september.ru

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

РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский
(редактор вкладки "В мир информатики")
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 27 622 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

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

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА "ПЕРВОЕ СЕНТЯБРЯ"
Английский язык – А.Громушкина
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – О.Коротова
Дошкольное образование – Д.Тюттерин
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – О.Волкова
История – А.Савельев
Классное руководство и воспитание школьников – М.Битянова
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
ОБЖ – А.Митрофанов
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Технология – А.Митрофанов
Управление школой – Е.Рачевский
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школа для родителей – Л.Печатникова
Школьный психолог – М.Чибисова

УЧРЕДИТЕЛЬ:
ООО "ЧИСТЫЕ ПРУДЫ"

Зарегистрировано ПИ № ФС77-44341 от 22.03.2011
в Министерстве РФ по делам печати
Подписано в печать:
по графику 11.06.2014,
фактически 11.06.2014
Заказ №
Отпечатано в ОАО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
ул. Полиграфистов, д. 1,
Московская область,
г. Чехов, 142300
Сайт: www.chpd.ru
E-mail: sales@chpk.ru
Факс: 8 (495) 988-63-76

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

Параллельные вселенные истории информатики

Эту идею я услышал от замечательного и увлеченного коллеги в залоге “Вот было бы здорово, а?”. В тот момент даже казалось, что реализовать идею вполне реально, но не сложилось. А ощущение “вот было бы здорово” осталось.

Идея такая: сделать музей истории информатики, который был бы устроен следующим образом. В каждом зале имеются несколько дверей, соответствующих различным решениям, которые были приняты (или могли быть приняты) в тот или иной момент. Ну, к примеру, компания IBM приняла решение сделать архитектуру IBM PC открытой. И реальная история информатики вошла именно в эту дверь. А если бы решение было иным? Понятно, что за другими дверьми — виртуальность. Некая параллельная вселенная истории ☺. Но интересно ведь поразмышлять — что в них могло бы быть?

Идея такого музея интересна тем, что она проявляет известный дуализм самой природы нашей науки. С одной стороны — мы стоим на вполне объективном, материалистическом фундаменте. Теория информации — тут не пофантазируешь. Но это лишь фундамент. Он не полностью определяет то, какие решения (например, технологического плана, бизнес-решения) принимают люди, которые строят на этом фундаменте здание реальной информатики. Если подумать, то даже архитектура фон Неймана — это всего лишь решение (изобретение, открытие — слова можно использовать разные) конкретного человека. А уж бизнес-решения, которые являли к жизни целые направления в развитии технологий, — вовсе неисчерпаемая ветка для размышлений “а если бы…”.

Мне было бы интересно провести детей по такому музею. Даже если бы за “нереальными” дверьми были просто пустые комнаты-аудитории, в которых можно было бы посидеть, подумать, порассуждать.

Еще мне приходит в голову, что интересным познавательным упражнением на знание истории информатики для детей (да и для нас — учителей ☺) было бы попробовать безошибочно пройти именно путь реальности. У меня, например, нет уверенности, что на каком-то этапе я и сам не вошел в виртуальную комнату.

Вот такая идея. Может быть, придет время и она будет реализована.

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

Язык Python глазами учителя

К.Ю. Поляков,
д. т. н., Санкт-Петербург

Введение

В этой статье речь пойдет о языке программирования Python, который приобретает все большую популярность. По данным одного из самых известных рейтингов TIOBE, Python с 2008 года прочно удерживается в восьмерке наиболее популярных языков программирования [1]. Его используют такие известные компании, как Google, Яндекс, Европейская организация по ядерным исследованиям (CERN), Национальное управление по воздухоплаванию и исследованию космического пространства США (NASA) и др.

Python — это скриптовый язык (язык сценариев). В этом качестве он применяется для автоматизации выполнения различных задач во многих программах, например, в *GIMP*, *Blender*, *Cinema 4D*, *Maya*, *Inkscape* и *Scribus*. Python занял свою нишу в игровой индустрии: он используется в играх *Eve Online*, *Civilization IV* и *Battlefield 2*.

Свободно распространяемые реализации языка Python существуют для всех популярных операционных систем (*Windows, Linux, Mac OS X, FreeBSD, Android, iOS* и др.), что сразу снимает проблему лицензирования программного обеспечения.

В университетах разных стран Python постепенно вытесняет языки C и Java, которые долгое время использовались для обучения студентов программированию. В список университетов и колледжей, в которых изучается Python, входят более 30 учебных заведений США, в том числе Массачусетсский технологический институт (MIT) — ведущий мировой центр инженерного образования [2].

Python постепенно “пробирается” и в школы России. Центром его распространения в нашей стране стала Мо-

сква: Python успешно преподают в школе “Интеллектуал”, в школах № 179, 2007, 57, в гимназии № 1543 и некоторых других. Решения на Python разрешены на большинстве региональных и Всероссийских олимпиад по программированию, а также на вебсервисах дистанционной подготовки к олимпиадам: *informatics.mccme.ru*, *codeforces.com*, *acm.timus.ru*.

Не могли обойти вниманием этот вопрос и авторы учебника информатики углубленного уровня для 10–11-х классов [3–4]. На сайте поддержки [5] размещены все материалы для учителя и учащихся, которые изучают Python по этому учебнику: электронные варианты глав по программированию, презентации для проведения уроков, тесты, примеры программ из учебника на языке Python.

Достаточно подробное введение в Python, с точки зрения специалиста по языкам программирования, уже публиковалось в журнале “Информатика” [7]. Задача настоящей статьи — посмотреть на Python с точки зрения учителя, преподающего курс программирования на императивных языках, например на языке Паскаль, и показать достоинства и недостатки Python как языка для обучения программированию. Среди предшествующих материалов этого плана отметим также презентацию Е.В. Андреевой, которая обобщает ее опыт преподавания на Python в школе “Интеллектуал” [8]. Наличие этих материалов позволяет автору не углубляться в подробное объяснение синтаксиса языка, а вместо этого остановиться на проблемах и перспективах его использования в школе, выделить особенности, на которые стоит обратить внимание. Отдельно показаны “подводные камни”, на которые можно наткнуться при переходе с императивных языков (Паскаля, C) на Python.

Отступы — часть синтаксиса

Многие учителя информатики хорошо знают, сколько сил и времени требуется для того, чтобы научить школьников правильно расставлять в программе отступы, выделяющие тело циклов или условных операторов. При программировании на Python этой проблемы не существует: отступы являются частью синтаксиса языка, то есть они обязательны. В Паскале, например, можно написать такой (ошибочный) цикл:

```
i := 0;
while i < 10 do
  writeln ( i );
  i := i + 1;
```

— и это приведет к зацикливанию, потому что оператор `i := i + 1` не входит в тело цикла. В Python такая ошибка в принципе невозможна, потому что все операторы, входящие в блок, должны иметь одинаковые отступы:

```
i = 0
while i < 10:
  print ( i )
  i = i + 1
```

Это правило применимо и к условным операторам:

```
if a > b:
  m = a
  k = k + 1
else:
  m = b
  q = q + 1
```

Использование отступов делает ненужными операторные скобки, ограничивающие блок (фигурные скобки в C-подобных языках, пара `begin-end` в Паскале).

Динамическая типизация

В языке Python используется динамическая типизация переменных. Это означает, что переменные не нужно объявлять. Тип переменной определяется автоматически, когда ей присваивается значение. Одна и та же переменная в разных частях программы может быть целым числом, затем вещественным числом, после этого — символьной строкой, списком (заменяющим массив), кортежем (неизменяемым списком) и словарем:

```
A = 100              # целое
A = 4.5              # вещественное
A = "Привет!"        # строка
A = [1, 2, 3, 4, 5]  # список (массив)
A = (1, "Вася", 3)   # кортеж
A = {"Вася": 12, "Петя": 23} # словарь
```

С одной стороны, эта особенность “снимает оковы” с программиста, а с другой — перекладывает на него ответственность.

Тип возвращаемого значения функции также не проверяется. Иногда это можно использовать “для пользы дела”. Например, функция, решающая линейное уравнение $ax = b$, может выглядеть так:

```
def solve ( a, b ):   # a*x = b
  if a == 0:
    if b == 0: return True
    else: return None
  else:
    return b / a
```

Если $a = b = 0$, решением уравнения является любое число; в этом случае функция возвращает логическое значение `True` (истина). При $a = 0$ и $b \neq 0$ уравнение не имеет решений, поэтому функция возвращает “пустое” значение `None`. Если же $a \neq 0$, решение единственно, и функция возвращает вещественное число b / a.

Итак, динамическая типизация — это хорошо или плохо? Среди программистов, пишущих на Python, известна фраза “*We are all consenting adults here*”, которая переводится примерно так: “Все мы здесь взрослые и по взаимному согласию”. Это значит, что мы получаем полную свободу в обмен на ответственность за свои действия.

Например, такая программа

```
if a > b:
  print ( "OK" )
else:
  this is spam
```

проходит трансляцию и может быть запущена, потому что переменные с именами `this` и `spam` могут существовать (их объявление не требуется!), а `is` — это оператор языка Python. Если же таких переменных нет, то ошибка будет обнаружена только во время выполнения блока после `else`. Поэтому программы на Python требуют тщательного тестирования всех ветвей алгоритма.

Еще более серьезная проблема может быть вызвана опечаткой в имени переменной. Например, в программе

```
x1 = 0
if a > b:
  xl = 1
```

ошибка не обнаруживается даже во время выполнения. Ее тяжело обнаружить и визуально, потому что цифра 1 и латинская строчная буква “эл” выглядят очень похоже. При выполнении условия `a > b` будет создана новая переменная `xl`, равная 1, а значение переменной `x1` не изменится. Конечно, в языке со статической типизацией эта ошибка будет выявлена при трансляции — ведь переменная `xl` скорее всего не объявлена.

Отсутствие проверки типов параметров при вызове функций тоже чревато неожиданными проблемами. Предположим, что мы написали функцию, которая удаляет лишние пробелы в начале символьной строки:

```
def trimLeft ( s ):
  while len(s) and s[0] == ' ':
    s = s[1:]
  return s
```

При правильном вызове мы передаем ей строку:

```
s1 = trimLeft ( "     123    " )
```

— но по ошибке можем передать и целое число (или данные другого типа):

```
s1 = trimLeft ( 123 )    # ошибка!
```

Эта ошибка будет обнаружена только во время выполнения, причем программа завершится аварийно.

Тип значения, возвращаемого функцией, тоже не проверяется. Например, мы забыли написать в предыдущей функции оператор `return`:

```
def trimLeft ( s ):
  while len(s) and s[0] == ' ':
    s = s[1:]
```

Такая функция (в терминах Паскаля это процедура) тоже допустима, просто она вернет пустое значение `None`. Поэтому при вызове

```
s1 = trimLeft ( "     123    " )
```

переменная `s1` получит значение `None`, что может привести к сбою в оставшейся части программы.

Аналогичная ошибка может возникнуть при вызове методов для списка. Например, метод `sort` *не* возвращает отсортированный список, а сортирует имеющийся. При вызове

```
A = [2, 1, 3]
B = A.sort()  # ошибка!
```

в переменную `B` будет записано “пустое” значение None, но транслятор не обнаружит потенциальной проблемы, предполагая, что вы знаете, что делаете. Ведь переменная `B` не объявляется и тип ее неизвестен. Если в Паскале нельзя вызывать процедуру как функцию, то в Python — можно, и это еще один источник ошибок.

Компактность кода

Одно из очевидных достоинств языка Python — компактность программного кода. Например, решение классической задачи — поменять местами значения двух переменных — на языке Паскаль решается в три оператора, например, так:

```
c := a;
a := b;
b := c;
```

— а на Python — в одну строку:

```
a, b = b, a
```

Алгоритм Евклида, который на Паскале записывается в виде

```
while b <> 0 do begin
  c := a mod b;
  a := b;
  b := c
end;
```

на языке Python значительно “ужимается”:

```
while b:
  a, b = b, a % b
```

Рассмотрим еще один класс задач — так называемую “длинную арифметику”, операции с большими целыми числами. Например, задача вычисления факториала числа 100 ($100! = 1 \times 2 \times 3 \times \ldots \times 100$) на Паскале решается с использованием вспомогательного массива для хранения “длинного” числа [4]:

```
const N = 33;
      d = 1000000;
var A: array[0..N] of longint;
    i, k, s, r: integer;
begin
  A[0] := 1;
  for k := 2 to 100 do begin
    r := 0;
    for i := 0 to N do begin
      s := A[i] * k + r;
      A[i] := s mod d;
      r := s div d
    end
  end
  { вывод длинного числа из массива A }
end.
```

В Python целые числа всегда “длинные”, то есть при необходимости объем памяти, отведенный для хранения числа, автоматически увеличивается. Поэтому можно писать просто и коротко:

```
A = 1
for i in range(2,101):
  A = A * i
print ( A )
```

Таким образом, целый класс задач на “длинную арифметику” теряет свою “олимпиадность”, поскольку их решение становится тривиальным.

Нужно отметить, что компактность кода говорит не о том, что Python лучше, чем Паскаль и C, а о том, что Python — это язык более высокого уровня. Он скрывает от программиста реализацию некоторых алгоритмов за счет встроенных средств. Аналогично языки высокого уровня (такие, как Паскаль и C) дают программисту возможность не задумываться о том, как реализуются алгоритмы с помощью команд и регистров процессора.

Локальные и глобальные переменные

Как и в других языках программирования, в Python можно использовать глобальные переменные (переменные модуля) и локальные (переменные функции или процедуры). Из-за динамической типизации при совпадении имен локальных и глобальных переменных могут возникнуть серьезные ошибки, которые достаточно сложно обнаружить.

Если в функции только *используется* глобальная переменная, но ее значение не нужно изменять, никаких дополнительных действий не требуется. Следующая программа правильно выводит значение глобальной переменной `x`, равное 0:

```
x = 0
def f():
  print ( x )
f()
```

Теперь попробуем изменить значение глобальной переменной внутри функции. Пусть состояние программы хранится в глобальной переменной `state`, и при вызове функции `changeState` его нужно изменить:

```
state = 0
def changeState():
  state = 1
changeState ()
print ( state )
```

Эта программа совершенно неожиданно выводит на экран число 0, то есть переменная `state` не изменилась! Это произошло потому, что при выполнении оператора `state = 1` в теле функции была создана новая локальная переменная с именем `state` и в нее записано значение 1. Глобальная переменная при этом не изменилась. Для того чтобы изменять значение глобальной переменной внутри функции, нужно объявить ее с помощью описателя `global`:

```
def changeState():
  global state
  state = 1
```

Заметим, что эта ошибка не обнаруживается ни при трансляции, ни во время выполнения (программа не завершается аварийно). Поэтому поиск и исправление подобных ошибок в программах на Python превращается в захватывающий процесс.

Списки >= массивы

В языке Python нет массивов в привычном понимании этого термина, но зато есть списки, которые можно считать расширением понятия “динамический массив”. Мы можем отдельно работать с каждым элементом списка, а можем выполнять операции со всем списком, например, добавлять и удалять элементы, копировать части списка, сортировать и т.п.

Кроме того, список в Python может объединять значения разных типов: числа, строки, вложенные списки и т.д. Однако этой возможностью на практике пользуются нечасто.

В Python, как в C-подобных языках программирования, нумерация элементов массива (списка) начинается с нуля. Поэтому далее в сравнительных примерах на Паскале используется массив, у которого индексы тоже начинаются с нуля

```
const N = 100;
var A: array[0..N - 1] of integer;
```

Заполнение массива одинаковыми значениями в Паскале выполняется с помощью цикла (здесь и далее `i` — целочисленная переменная):

```
for i := 0 to N - 1 do
  A[i] :=  0;
```

— а в Python может быть записано в краткой форме:

```
A = [0] * N
```

В последней строке `[0]` — это список, состоящий из одного элемента, равного нулю. “Умножение” на *N* означает “сумму” *N* одинаковых списков, которые объединяются в один список.

Теперь заполним массив квадратами последовательных целых чисел, начиная с нуля. Привычный цикл в Паскале

```
for i := 0 to N - 1 do
  A[i] := i * i;
```

на Python можно записать в такой форме

```
A = [i * i for i in range(N)]
```

Это *генератор списка*. Мы видим цикл `for i in range(N)`, который перебирает все целые значения `i` от 0 до *N* – 1. Для каждого из этих значений мы вычисляем `i * i` и из полученных величин составляем список (на это указывают квадратные скобки).

Встроенные функции позволяют легко найти минимум и максимум массива (списка):

```
mi = min(A)
ma = max(A)
```

а также отсортировать его:

```
A.sort()
```

Задача реверса массива (перестановки элементов в обратном порядке) на Паскале решается в виде цикла:

```
for i := 0 to N div 2 do begin
  c := A[i];
  A[i] := A[N - i + 1];
  A[N - i + 1] := c
end;
```

а на Python — в одну строчку:

```
A = A[::-1]
```

Это так называемый "срез", выбирающий все элементы от последнего до первого с шагом –1.

Часто нужно выбрать все элементы массива, удовлетворяющие некоторому условию (например, все положительные), в другой массив. Классическое решение на Паскале со счетчиком

```
count := 0;
for i := 0 to N - 1 do
  if A[i] > 0 then begin
    B[count] := A[i];
    count := count + 1
  end;
```

на Python заменяется одним генератором:

```
B = [x for x in A if x > 0]
```

Для лучшего понимания можно разбить его на части:

```
B = [x
     for x in A
       if x > 0]
```

Мы хотим перебрать все элементы массива A (цикл `for x in A`), оставить только положительные (`if x > 0`) и построить новый список из этих элементов (`[x ...]`).

Теперь предположим, что нужно выбрать все неповторяющиеся элементы массива A в массив B. Решение на Паскале получается довольно длинное. Перебираем все элементы массива A, для каждого проверяем, есть ли уже этот элемент в массиве B, и если нет — добавляем его:

```
count := 0;
for i := 0 to N - 1 do begin
  j := 0;
  while (j < count) and (A[i] <> B[j]) do
    j := j + 1;
  if j = count then begin
    B[count] := A[i];
    count := count + 1
  end
end;
```

Решение на Python элегантно записывается в одну строчку:

```
B = list ( set(A) )
```

Сначала из списка A строится множество (`set`), при этом удаляются все дубликаты. Затем это множество превращаем обратно в список (`list`).

При работе со списками нужно помнить, что *переменная-список — это ссылка*. Программа

```
A = [1, 2, 3]
B = A
```

создает в памяти один список, на который ставится две ссылки: обращаться к этому списку можно по именам A и B:

Поэтому при изменении списка A одновременно изменится и список B. Для того чтобы создать копию списка, можно взять срез по всем элементам:

```
B = A[:]
```

Это приведет к созданию второго независимого списка в памяти:

Кортежи

Кортеж — это неизменяемый список, он записывается с помощью круглых скобок:

```
T = (1, 2, 3)
```

В отличие от списка нельзя добавить, изменить или удалить элемент кортежа, но можно построить новый кортеж и связать его с той же переменной.

Используя кортежи, функция может вернуть несколько значений (объединив их в кортеж). Например, пусть требуется написать функцию, которая находит минимальный элемент массива и его индекс

```
def minInd ( A ):
  m = min(A)
  ind = A.index(m)
  return (m, ind)
```

Оператор `return (m, ind)` вернет кортеж, составленный из значения минимального элемента и его индекса.

При работе с координатами точек удобно хранить информацию о каждой точке в виде кортежа, для двухмерного случая — в виде пары `(x, y)`. Например, можно создать список кортежей — координат точек, по которым строится ломаная линия:

```
L = [(0,0), (0,2), (5,-5), (6, 6)]
```

Символьные строки

В Python нет отдельного типа данных "символ", но есть тип "строка" (`string`). Нумерация символов строки начинается с нуля. Для работы со строками используются *срезы*, которые позволяют выбрать часть символов строки от начального индекса до конечного с некоторым шагом. Если не указан начальный индекс, он считается равным 0, если не указан конечный — выбираются все символы до конца строки.

Нужно запомнить, что *последний указанный индекс не входит в выборку*. Приведем несколько примеров:

```
s = "0123456789"
s1 = s[2:5]       # "234"
s2 = s[:5]        # "01234"
s3 = s[2:]        # "23456789"
s4 = s[2::2]      # "2468"
```

Можно использовать отрицательные индексы, в этом случае к ним добавляется длина строки, то есть индекс `-1` означает то же самое, что и `len(s)-1`:

```
s = "0123456789"
s1 = s[-1]        # "9"
s2 = s[2:-1]      # "2345678"
s3 = s[-5:-2]     # "567"
s4 = s[-5::-2]    # "531"
```

В отличие от многих других языков программирования в Python строки — это неизменяемые объекты. Например, в Паскале можно просто заменить в строке все вхождения одной буквы на другую:

```
for i := 1 to Length(s) do
  if s[i] = 'a' then s[i] := 'b';
```

В Python оператор `s[i] = "b"` не сработает. Вместо этого придется при каждой замене символа строить новую строку и записывать ее в переменную `s`:

```
for i in range(len(s)):
  if s[i] == "a":
    s = s[:i] + "b" + s[i + 1:]
```

Конечно, такой цикл будет выполняться значительно дольше, чем решение на Паскале.

Словари

Одна из самых интересных структур данных в Python — *словарь* или ассоциативный массив (хэш-таблица). Фактически словарь — это пары "ключ – значение", причем в качестве ключей можно использовать любые неизменяемые объекты: числа, строки, кортежи.

Покажем на примере эффективность использования словарей. В учебнике [4] рассмотрена задача обработки файла, в котором в столбик записаны слова (среди них есть повторяющиеся). Требуется составить алфавитно-частотный словарь — вывести все встречающиеся слова в алфавитном порядке, и рядом с каждым словом указать, сколько раз оно встречается. В данном случае нужно построить список пар "слово – количество" и отсортировать его по первому элементу пары (слову).

Решение задачи на Паскале [4] требует значительных усилий, потому что все операции со списком нужно реализовать "вручную":

```
uses WordList;
var F: text;  s: string;
    L: TWordList;  p: integer;
begin
  Assign(F, 'input.txt');
  Reset(F);
  SetLength(L.data, 0);
  L.size := 0;
  while not eof(F) do begin
    readln(F, s);
    p := Find ( L, s );
    if p >= 0 then
     L.data[p].count := L.data[p].count + 1
    else begin
      p := FindPlace ( L, s );
      InsertWord ( L, p, s );
    end
  end;
  Close(F);
  ...
end.
```

К этому добавляется еще модуль `WordList.pas`, занимающий около 50 строк.

Решение на Python отличается краткостью и простотой, которая достигается благодаря использованию готовой структуры данных, идеально подходящей для этой задачи, — словаря:

```
D = {}
for line in open("input.txt"):
  word = line.strip()
  if word:
    D[word] = D.get(word, 0) + 1
```

Сначала создается пустой словарь `D`. Затем в цикле перебираются все строки входного файла, каждая строка-слово "очищается" от пробельных символов в начале и в конце с помощью метода `strip`. Если строка непустая, мы ищем слово в словаре и определяем значение его счетчика. Второй аргумент метода `get` задает значение по умолчанию, которое вернет метод в том случае, когда слово не найдено. Затем значение счетчика увеличивается на 1.

Для вывода результата на экран нужно перебрать все ключи словаря, предварительно отсортировав их с помощью функции `sorted`:

```
for x in sorted(D):
  print ( x, D[x] )
```

Ввод данных

Ввод данных в Python организован менее удобно, чем в Паскале. Это вызвано в первую очередь динамической типизацией.

Пусть требуется ввести число и умножить его на 2. На Паскале мы напишем программу так:

```
var N: integer;
begin
  write ( "Введите число " );
  read ( N );
  write ( N*2 )
end.
```

Можно записать аналогичный код на Python:

```
N = input ( "Введите число " )
print ( N*2 )
```

— но результат получится совсем другой. Оператор `input` вводит с клавиатуры данные, используя переданную ему строку как подсказку для ввода. Так как тип переменной *N* неизвестен из-за динамической типизации, по умолчанию предполагается, что она символьная, поэтому при вводе значения 12 в переменную *N* записывается символьная строка `"12"`. Затем, когда эта строка "умножается" на 2, получается не 24, а `"1212"`.

Для того чтобы ввести именно целое число, результат функции `input` нужно преобразовать в целое значение с помощью функции `int`:

```
N = int ( input("Введите число ") )
print ( N*2 )
```

Теперь при вводе числа 12 мы увидим результат 24.

Ситуация усложняется, если нужно ввести несколько чисел. На Паскале код выглядит очень просто и естественно:

```
write ( "Введите три числа " );
read ( a, b, c );
```

Заметим, что здесь числа можно вводить как в одной строке, так и в разных, нажимая клавишу Enter после каждого числа.

Функция `input`, которая выполняет ввод данных в Python, вводит сразу всю строку. Ее нужно разбить на части (“слова”) и каждое слово отдельно преобразовать в целое число. Разделение строки на слова (по пробелам-разделителям) выполняет метод `split`:

```
s = input("Введите три числа ")
L = s.split()
```

Если ввести строку `"1 2 3"`, в переменную *L* попадет список строк `["1", "2", "3"]`. Теперь каждый элемент списка нужно преобразовать в целое число:

```
a = int(L[0])
b = int(L[1])
c = int(L[2])
```

Эти операции можно записать в краткой форме, используя функцию `map`, которая применяет какую-то другую функцию (в данном примере — `int`) ко всем элементам списка:

```
a, b, c = map(int, L)
```

Можно обойтись вообще без переменной `L`:

```
a, b, c = map(int, s.split())
```

Заметим, что при этом ожидается, что все три значения будут введены в одной строке. Если значений будет меньше или больше, программа завершится аварийно.

Теперь пусть требуется ввести массив из *N* элементов. На Паскале мы запишем цикл:

```
for i := 0 to N - 1 do
  read ( A[i] );
```

— а на Python — одну строчку, хотя более сложную:

```
A = list( map(int, input().split()) )
```

В сравнении с предыдущими примерами, здесь добавляется вызов функции `list`, которая строит список из всех полученных элементов. Обратите внимание, что размер массива нам знать не требуется — программа введет столько элементов, сколько записано во входной строке.

Задачи ЕГЭ

Поскольку пока решения задач ЕГЭ по программированию разрешено писать на любом языке, можно использовать и Python. Рассмотрим задачу С2 из демоварианта ЕГЭ-2014 [9].

Дан целочисленный массив из 20 элементов. Элементы массива могут принимать целые значения от 0 до 10 000 включительно. Опишите на естественном языке или на одном из языков программирования алгоритм, позволяющий найти и вывести максимальное значение среди трехзначных элементов массива, не делящихся на 9. Если в исходном массиве нет элемента, значение которого является трехзначным числом и при этом не кратно 9, то выведите сообщение “Не найдено”.

Решение задачи на Python занимает всего несколько строк. Сначала с помощью генератора выбираем все подходящие значения и строим из них новый массив:

```
a = [x for x in a
   if 100 <= x and x <= 999 and x % 9 != 0]
```

Теперь остается проверить размер массива (найден ли хотя бы один элемент) и вывести результат:

```
if len(a) > 0:
  print ( max(a) )
else:
  print ( "Не найдено" )
```

Заметим, что при этом в принципе нельзя сделать некоторых ошибок, предусмотренных критериями проверки [9]. Например, невозможно неверно задать начальное значение переменной `max`, потому что она вообще не используется.

Рассмотрим теперь задачу С4 из того же демонстрационного варианта [9].

По каналу связи передается последовательность положительных целых чисел, все числа не превышают 1000. Количество чисел известно, но может быть очень велико. Затем передается контрольное значение последовательности — наибольшее число R, удовлетворяющее следующим условиям:

1) *R — произведение двух различных переданных элементов последовательности (“различные” означает, что не рассматриваются квадраты переданных чисел; допускаются произведения различных элементов последовательности, равных по величине);*

2) *R делится на 21.*

Если такого числа R нет, то контрольное значение полагается равным 0. Напишите программу, которая проверяет правильность контрольного значения.

Решения этой задачи на Python короче и понятнее, чем аналогичные эталонные решения на Паскале и Бейсике, приведенные в [9]:

```
N = int(input())
M3 = M7 = M21 = M = 0
for i in range(N):
    x = int(input())
    if x % 21 != 0:
        if x % 3 == 0: M3 = max(M3, x)
        if x % 7 == 0: M7 = max(M7, x)
    if x % 21 == 0  and  x > M21:
        M = max(M21, M)
        M21 = x
    else: M = max(M, x)
R0 = int(input())
R = max(M3*M7, M21*M)
if R == R0: print("Контроль пройден")
else:       print("Контроль не пройден")
```

Таким образом, использование Python при решении задач С2 и С4 позволяет, как правило, сократить программу за счет встроенных возможностей языка и тем самым уменьшить вероятность случайных ошибок.

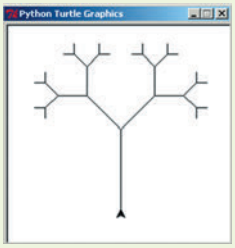
Черепашья графика

В языке Python есть встроенный модуль “черепашьей графики” `turtle`, в котором реализованы

знакомые всем команды исполнителя Черепаха. Исполнитель рисует в отдельном окне, которое открывается при запуске программы. Для примера приведем программу, которая рисует дерево Пифагора с помощью рекурсивной процедуры tree:

```
from turtle import *
def tree ( levels, length ):
  if levels > 0:
    forward ( length )
    left ( 45 )
    tree ( levels-1, length*0.6)
    right ( 90 )
    tree ( levels-1, length*0.6)
    left ( 45 )
    backward ( length )
setheading ( 90 )
tree ( 5, 100 )
```

Вот результат ее работы:



Поскольку Python 3 поддерживает Юникод, в названиях функций можно использовать русские буквы. Поэтому несложно перевести все команды Черепахи на русский язык, построив небольшой вспомогательный модуль turtle_rus:

```
# turtle_rus.py
# -*- coding: utf-8 -*-
from turtle import *
def новый_курс(x): setheading(x)
def влево(x): left(x)
def вправо(x): right(x)
def вперед(x): forward(x)
def назад(x): backward(x)
```

Фактически этот модуль просто вводит новые русские имена для команд Черепахи. Теперь программа для построения дерева Пифагора может быть записана так:

```
# -*- coding: utf-8 -*-
from turtle_rus import *
def дерево ( уровни, длина ):
  if уровни > 0:
    вперед ( длина )
    влево ( 45 )
    дерево ( уровни-1, длина*0.6)
    вправо ( 90 )
    дерево ( уровни-1, длина*0.6)
    влево ( 45 )
    назад ( длина )
новый_курс ( 90 )
дерево ( 5, 100 )
```

Обратите внимание, что здесь импортируется не исходный модуль turtle, а модуль turtle_rus с русскими командами исполнителя. Для того чтобы интерпретатор не запутался в кодировках, в первой строке модуля и программы явно указано, что используется кодировка UTF-8.

Объектно ориентированное программирование

Язык Python поддерживает объектно ориентированный стиль программирования, причем объектная модель унаследована от языка SmallTalk. Все члены класса — общедоступные (открытые, *public*), то есть фактически инкапсуляция (скрытие данных и внутреннего устройства объекта) отсутствует. В отличие от большинства объектно ориентированных языков в Python также нет и защищенных элементов класса (*protected*), которые доступны только классам-наследникам. Отсутствие защиты может привести к тому, что данные объекта могут неожиданно измениться в результате ошибочного присваивания в другой функции.

Например, построим класс dog для хранения информации о собаках:

```
class dog:
  def __init__(self, _age):
    self.age = _age
```

В этом классе один метод — конструктор __init__, который принимает два параметра. Как и у всех методов любого класса, первый параметр конструктора — это ссылка на сам объект, вызвавший метод (он выполняет ту же роль, что и self в Паскале и *Delphi* или this в C). Второй параметр — это возраст собаки, который в конструкторе записывается в поле объекта (self.age = _age). Это поле открытое, поэтому ничто не мешает изменить его напрямую откуда угодно.

Пусть мы создали в программе объект, который описывает пятилетнюю собаку:

```
tuzik = dog(5)
```

Однако в любой посторонней функции можно изменить это поле:

```
tuzik.age = 150
```

Конечно, эта проблема известна и какие-то шаги по ее решению автором языка предпринимались. Например, если имя поля начинается с двух знаков подчеркивания, используется преобразование имен: для доступа к этому полю нужно к имени добавить впереди знак подчеркивания и имя класса. Поле как бы скрывается, но на самом деле доступ к нему сохраняется, только по измененному имени.

Например, если бы мы назвали поле класса dog именем __age:

```
class dog:
  def __init__(self, _age):
    self.__age = _age
```

— то “добраться” до него можно было бы по имени _dog__age:

```
tuzik._dog__age = 150
```

Тут снова приходится вспоминать один из принципов Python: “Все мы тут взрослые и по взаимному согласию”. Снятие ограничений и свобода для про-

граммиста, с одной стороны, и полная ответственность за свои действия с другой.

Еще одна проблема связана с обязательным использованием описателя `self` при обращении к полю объекта в методе класса. В отличие от большинства объектно ориентированных языков опускать `self` нельзя, хотя транслятор об этой ошибке не сообщит. Например, если бы конструктор был записан так:

```
class dog:
  def __init__(self, _age):
    age = _age
```

У объектов класса `dog` не было бы поля `age`. Вместо этого в конструкторе создается локальная переменная `age`, жизнь которой заканчивается при выходе из конструктора. Объект создается без проблем, а ошибка будет обнаружена только при обращении к полю `age` (которого на самом деле нет!). Например, при выводе на экран

```
print ( tuzik.age )
```

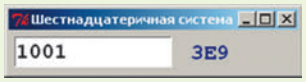
программа завершится аварийно. Понятно, что в этом отчасти тоже "виновата" динамическая типизация переменных в Python.

Графический интерфейс

На Python можно создавать программы с графическим интерфейсом. Для этого есть несколько библиотек: встроенная библиотека `tkinter`, а также более сложные варианты — сторонние библиотеки wxPython (*www.wxpython.org*), PyGTK (*www.pygtk.org*) и PyQt (*www.riverbankcomputing.com/software/pyqt*).

Большинство практических задач можно решить с помощью `tkinter`. Эта библиотека используется для разработки программ с графическим интерфейсом в Python-версии учебника [4]. Для упрощения работы и приближения к привычной многим идеологии *Delphi* авторами разработан модуль-обертка `simpletk`, который можно скачать на сайте поддержки [6].

Напишем простую программу для перевода чисел в шестнадцатеричную систему счисления, которая создает окно с полем ввода и меткой:



Сначала загружаем все функции модуля `simpletk` и создаем объект-приложение `app`:

```
from simpletk import *
app = TApplication("Шестнадцатеричная
                    система")
app.size = (250, 36)
app.position = (200, 200)
```

Затем создаем метку и размещаем ее на форме:

```
f = ("Courier New", 14, "bold")
hexLabel = TLabel(app, text="?",
           font=f, fg="navy")
hexLabel.position = (155, 5)
```

Здесь `f` — это кортеж, который задает свойства шрифта.

Строим обработчик события "изменение текста в поле ввода":

```
def onNumChange(sender):
  hexLabel.text = "{:X}".format(sender.value)
```

Он принимает один параметр — вызвавший объект, переводит введенное число (значение свойства `value`) в шестнадцатеричную систему и изменяет текст метки.

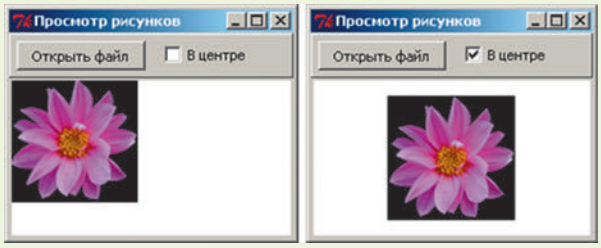
Остается создать и разместить на форме поле для ввода целого числа (объект класса `TIntEdit`)

```
decEdit = TIntEdit(app, width=12, font=f)
decEdit.position = (5, 5)
decEdit.text = "1001"
decEdit.onChange = onNumChange
```

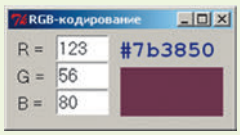
и запустить приложение:

```
app.Run()
```

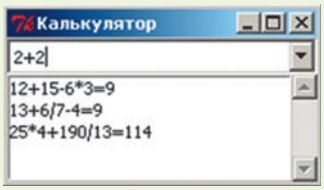
В Python-версии учебника [4] рассмотрены еще несколько программ с графическим интерфейсом. Это программа для просмотра рисунков с диска



программа для перевода RGB-кода цвета в HTML-формат



и два типа калькуляторов:



Функциональное программирование

Популярность языка Python объясняется еще и тем, что он поддерживает разные стили (*парадигмы*) программирования: императивное программирование (как в Паскале и C), объектно ориентированное (как в C++ и Delphi) и *функциональное*. Последний из перечисленных стилей наименее известен учителям информатики, поэтому на нем следует остановиться особо.

Основная идея функционального стиля программирования состоит в том, что вся программа представляет собой вычисление значений функций (в математическом смысле этого слова), которые по

исходным данным получают некоторый результат. В отличие от императивных языков, которые основаны на идее изменения *состояния* программы (значений переменных), в функциональных языках нередко можно обойтись вообще без переменных и без привычных циклов. Среди функциональных языков программирования нужно отметить LISP, Scheme, Haskell, Scala, Erlang, F#.

Циклы обычно заменяют рекурсией. При этом программа получается понятнее и проще. Например, функция, которая определяет сумму цифр числа, на Паскале (в императивном стиле) может быть записана так:

```
function sumDigits(x: integer): integer;
var s: integer;
begin
  s := 0;
  while x <> 0 do begin
    s := s + x mod 10;
    x := x div 10
  end;
  sumDigits := s
end;
```

Рекурсивное решение на Python (без циклов и вспомогательных переменных) значительно проще и понятнее:

```
def sumDigits ( x ):
  if x > 0:  return x % 10 + sumDigits(x//10)
  else:      return 0
```

Если число больше нуля, сумма цифр определяется как последняя цифра (остаток от деления числа на 10) и сумма цифр числа с отброшенной последней цифрой (`x//10`). Если число равно нулю, то и сумма его цифр равна нулю; это условие окончания рекурсии. Обратите внимание, что во втором решении значительно труднее сделать случайную ошибку, то есть код становится более надежным.

Еще один пример: функция, которая возвращает логическое значение `True`, если переданное ей слово — палиндром (читается одинаково в обоих направлениях), и `False`, если слово не является палиндромом. При сравнении нерекурсивного решения на Паскале

```
function isPalindrome (s: string): boolean;
var i: integer;
begin
  isPalindrome := True;
  for i := 1 to Length(s) div 2 do
    if s[i] <> s[Length(s) - i + 1] then
    begin
      isPalindrome := False;
      break
    end
end;
```

и рекурсивного на Python:

```
def isPalindrome ( s ):
  if len(s) > 1:
    return s[0] == s[-1] and
           isPalindrome(s[1:-1])
  else: return True
```

явно выигрывает второе. Его идея очень проста: слово является палиндромом, если равны первый и последний символы, и при этом оставшаяся часть строки (без этих символов) — тоже палиндром. В то же время надо отметить, что рекурсивное решение требует большего расхода памяти (создаются новые строки), а нерекурсивное решает вопрос “на месте”.

К функциональному стилю относятся и генераторы списков, которые мы уже упоминали. Например, генератор

```
B = [x for x in A if x % 3 == 0]
```

выбирает все элементы массива (списка) `A`, которые делятся на 3, и строит из них массив `B`.

В функциональных языках программирования функция — это такой же объект, как число, строка или массив. Функцию можно передавать в другие функции как аргумент и возвращать как результат работы других функций.

Покажем, как можно передать функцию в качестве аргумента в другую функцию. Мы, кстати, это уже делали, когда вводили несколько целых чисел в одной строке. После разделения строки на слова нужно было применить функцию `int` к каждой части, например, так:

```
a, b, c = map(int, s.split())
```

Этот оператор записан в функциональном стиле: первый аргумент при вызове функции `map` — это *функция* `int`, которая должна быть применена ко всем элементам списка.

Таким же способом можно применить к элементам списка любую функцию, в том числе и созданную программистом:

```
def x5 ( x ):
  return x**5
B = list( map(x5, A) )
```

В этом примере ко всем элементам списка `A` применяется функция *x*5, которая возводит число в пятую степень. Затем из полученных чисел строится новый список `B`.

Если функция состоит из одной строки и не нужна в других местах программы, можно не оформлять ее с помощью описателя `def`, а передать функции `map` безымянную функцию, или “лямбда-функцию”, которая описывается с помощью ключевого слова `lambda`:

```
B = list( map( lambda x: x**5 , A) )
```

Приведем еще один пример: нужно возвести в квадрат все элементы массива `A` и найти их произведение. На Паскале можно решить задачу с помощью двух циклов (здесь используется вспомогательный массив `B` и целая переменная `p`):

```
for i := 1 to N do
  B[i] := A[i] * A[i];
p := 1;
for i := 1 to N do
  p := p * B[i];
```

Программа на Python в функциональном стиле выглядит так:

```
from functools import reduce
B = list( map( lambda x: x**2, A ) )
p = reduce( lambda p, x: p*x, B )
```

Сначала все элементы массива A обрабатываются с помощью "лямбда-функции" `lambda x: x**2`, которая возводит число в квадрат. Из полученных значений строится массив B. Затем применяется функция `reduce` из модуля `functools`[1], которая накапливает произведение. Ей передается "лямбда-функция" от двух аргументов `lambda p,x: p*x`. Эта функция на каждом шаге умножает накопленное значение `p` на величину `x`, на место которой по очереди подставляются все элементы массива B.

Как уже было сказано, функция может вернуть другую функцию в качестве результата. Пусть нам нужна функция, которая сравнивает пароль, введенный пользователем сайта, с правильным паролем. Можно написать эту функцию так:

```
def check(s, valid):
  return s == valid
```

Но в этом случае при каждом вызове нужно передавать функции правильный пароль:

```
valid_pass = "123"
OK = check ( s, valid_pass )
...
OK = check ( s, valid_pass )
```

Вместо этого можно создать функцию, которая "запомнит" правильный пароль. Эта функция будет создана другой функцией, которую можно назвать "фабрикой":

```
def checkFact ( valid ):
  def check ( s ):
    return s == valid
  return check
```

Функция `checkFact` ("фабрика") принимает один параметр — правильный пароль. Обратите внимание на возвращаемое значение: эта функция возвращает в качестве результата *свою внутреннюю функцию* `check`, которая запоминает правильный пароль `valid`. Такая внутренняя функция вместе с запомненными данными называется *замыканием* (*closure*). Этот термин означает, что функция-результат "замкнула на себя", то есть запомнила, те данные, которые были получены "фабрикой" при создании этой функции.

Для создания функции проверки вызываем "фабрику", передав ей правильный пароль:

```
valid_pass = "123"
check = checkFact ( valid_pass )
```

Теперь `check` — это переменная класса `function` (функция). При вызове не нужно каждый раз передавать функции правильный пароль, потому что она его запомнила при создании:

```
OK = check ( s )   # сравнивает s с "123"
```

В каких задачах может пригодиться такой прием? Во-первых, для хранения данных, которые должны быть доступны из нескольких процедур и функций, но не должны быть глобальными (из соображений защиты). Пусть в программе выполняется какая-то длительная операция (например, загрузка данных в память) и нужно сделать индикатор загрузки. Для того чтобы не использовать глобальную переменную, можно построить замыкание, которое "запомнит" текущее состояние:

```
def counterFact(start = 0):
  value = start
  def counter(step = 1):
    nonlocal value
    value += step
    return value
  return counter
```

При вызове функции-фабрики `counterFact` ее параметр — начальное значение `start` — запоминается в локальной переменной этой функции `value`. Фабрика возвращает свою внутреннюю функцию `counter`, которая "замыкает" на себя (запоминает) значение `value`. Описатель `nonlocal` означает, что переменная `value` — не локальная (но и не глобальная!), то есть объявлена во внешней функции `counterFact`.

Теперь можно создать функцию-счетчик для индикатора загрузки (по умолчанию начальное значение счетчика равно нулю):

```
load_progress = counterFact ()
```

и использовать ее

```
print ( load_progress() )  # выведет 1
print ( load_progress(2) ) # выведет 3
print ( load_progress(3) ) # выведет 6
print ( load_progress() )  # выведет 7
```

При вызове этой функции без параметра счетчик увеличивается на значение шага по умолчанию, равное 1 (см. заголовок внутренней функции `counter` выше).

Замыкание часто используется при назначении обработчиков событий в программах с графическим интерфейсом. Напишем программу, которая выводит в три специальные области портреты писателей, если включить соответствующий флажок-переключатель:

Если флажок выключен, место портрета заполняется белым фоном.

Для разработки программы будем использовать стандартный модуль `tkinter` с "оберткой" в виде модуля `simpletk` [6]. Пусть первый флажок (объект

[1] В Python 2.x импортировать этот модуль не нужно.

TCheckBox, “отвечающий” за портрет Пушкина) называется cb1, соответствующее место для рисунка (объект TImage) называется image1, а файл с портретом — pushkin.gif. Тогда подключение обработчика события “переключение флажка” выглядит так:

```
def changeImage1(sender):
  if sender.checked:
        image1.picture = "pushkin.gif"
  else: image1.picture = ""
cb1.onChange = changeImage1
```

Но таких пар “флажок – рисунок” в программе несколько, и для каждого флажка придется написать свою функцию-обработчик, причем эти функции отличаются только названием объекта-рисунка и именем файла. Поэтому имеет смысл построить “функцию-фабрику”, которая будет принимать эти два параметра и возвращать новую функцию-обработчик, запоминая все изменяющиеся данные с помощью замыкания:

```
def changer(image, filename):
  def change(sender):
    if sender.checked:
          image.picture = filename
    else: image.picture = ""
  return change
```

Теперь назначение обработчиков флажкам (имеющим имена cb1, cb2 и cb3) выглядит так:

```
cb1.onChange = changer (image1, "pushkin.gif")
cb2.onChange = changer (image2,
                        "lermontov.gif")
cb3.onChange = changer (image3,
                        "bulgakov.gif")
```

Таким образом, с помощью замыкания мы избавились от дублирования кода. Полную программу на Python 3 вы можете найти в приложении к этому номеру.

Библиотеки

Одно из бесспорных достоинств языка Python — разнообразие готовых библиотек, в том числе встроенных. Назовем только некоторые из часто используемых модулей:

- math — математические функции;
- fractions — рациональные дроби;
- decimal — десятичная арифметика;
- random — случайные числа, случайный выбор, случайная перестановка элементов;
- re — регулярные выражения;
- itertools — перестановки, сочетания;
- webbrowser, urllib, http, ftplib — работа с сетью;
- sqlite — работа с базами данных SQLite;
- tkinter — графический интерфейс.

Кроме того, Python-сообществом разработано большое число сторонних библиотек, например,

- NumPy — пакет для научных вычислений (*www.numpy.org*);
- SymPy — библиотека для символьных вычислений (*sympy.org*);
- wxPython, PyGTK и PyQt — библиотеки для создания графического интерфейса;
- pygame — для программирования игр (*www.pygame.org*).

Дистанционное образование

В последние годы быстрыми темпами развивается дистанционное образование. Ведущие сайты дистанционного обучения проводят бесплатные курсы по изучению языка Python. Среди них нужно выделить основательный курс Массачусетсского технологического института (MIT), размещенный на платформе edX [10].

Для желающих научиться программировать игры на Python университетом Райса (США) предлагается курс “Введение в интерактивное программирование на Python” [11]. Специально для этого курса разработана интерактивная веб-среда CodeSkuptor [12] с обширной документацией, которая позволяет не только отлаживать программы на Python, но и наблюдать за изменениями в памяти при их выполнении в режиме визуализации.

Еще один вводный курс по Python размещен на сайте Codecademy [13].

Существует несколько сайтов, где можно отлаживать программы на Python в браузере в интерактивном режиме:

http://www.codeskulptor.org/
http://pythontutor.com/
http://ideone.com/
http://www.compileonline.com/execute_python_online.php
http://www.skulpt.org/

Первые два из перечисленных сайтов поддерживают визуализацию, то есть показывают изменения данных в памяти во время пошагового выполнения программы.

К сожалению, большинство учебных материалов по Python представлено на английском языке. Среди русскоязычных ресурсов следует отметить разработки Д.П. Кириенко [14] и [15].

Заключение

Говоря о достоинствах языка Python с точки зрения обучения школьников программированию, нужно выделить следующее:

- низкий “порог входа”; простейшая программа на Python в отличие от Паскаля и C занимает одну строчку:

```
print ("Привет, Вася!")
```

- понятный синтаксис, отступы как часть синтаксиса языка;
- Python — это язык более высокого уровня, чем C и Паскаль, позволяющий решать задачу на более высоком уровне абстракции;
- развитые структуры данных: списки, словари, множества;

• компактность программ (достигается за счет встроенных средств);

• Python широко применяется в профессиональных разработках, то есть не является чисто учебным языком без перспектив применения в реальной жизни;

• большая библиотека (встроенные модули и разработки сообщества);

• возможность разработки программ с графическим интерфейсом;

• Python поддерживает различные подходы к программированию (императивный, объектно ориентированный, функциональный).

В то же время, есть и достаточно много проблем, сдерживающих использование Python как учебного языка.

Как мы уже говорили, Python предоставляет программисту много свободы, перекладывая на него всю ответственность за возможные ошибки. Такой подход часто полезен для опытных программистов, но может приводить к серьезным проблемам, которые проявляются только во время выполнения некорректных операторов и "вылавливаются" с трудом. В некоторых случаях обычные опечатки могут вызвать логические ошибки в программе, потому что не обнаруживаются транслятором. Одной из причин этого является динамическая типизация (см. примеры выше). Поэтому программы на Python требуют более тщательного тестирования.

В настоящее время существуют две версии Python — 2.x (устаревшая) и 3.x (перспективная), несовместимые между собой. Перевод программ, написанных в версии 2, в версию 3 настолько затруднителен и чреват ошибками, что было решено поддерживать обе версии параллельно.

Python — интерпретируемый язык, поэтому для выполнения программ нужен интерпретатор. Скорость выполнения программ на Python может быть в 100 раз ниже, чем скорость выполнения программ на языке C, при этом Python-программы расходуют больше памяти.

В Python используется неклассическая объектная модель: все члены класса — общедоступные, нельзя сделать закрытые (*private*) или защищенные (*protected*) поля и методы. Тем самым фактически невозможна строгая инкапсуляция.

До сегодняшнего дня не создано надежных RAD-систем для разработки программ на Python с графическим интерфейсом, которые можно было бы использовать в учебных и прикладных задачах.

По этим причинам автор склонен поддержать мнение И.А. Сукина [7]: Python хорош для профессиональных программистов, но его использование в качестве первого языка программирования может быть неудачным решением. Как признаются учителя, преподающие на Python, те, кто учился программировать на Python, не хотят переходить на другие (более низкоуровневые) языки. Научив школьников сортировать массивы вызовом метода `sort`, сложно потом объяснить, зачем написаны целые тома об алгоритмах сортировки. А это может привести к появлению плеяды "программистов-только-на-Python", не готовых к преодолению дополнительных ограничений ради повышения эффективности программы. Фактически учитель попадает в ситуацию, которая хорошо описывается фразой "В Python такие возможности есть, но учить так нельзя!" (Е.В. Андреева). В то же время, было бы полезным изучение Python в качестве второго языка программирования в классах с углубленным уровнем изучения информатики (например, после Паскаля или C).

Автор благодарит В.М. Гуровица за обсуждение материалов статьи и полезные замечания.

Литература

1. TIOBE Index [Электронный ресурс] URL: *http://www.tiobe.com/index.php/content/paperinfo/ tpci/index.html* (дата обращения 18.05.2014).

2. Schools Using Python [Электронный ресурс] URL: *https://wiki.python.org/moin/SchoolsUsingPython* (дата обращения 18.05.2014).

3. *Поляков К.Ю., Еремин Е.А.* Информатика. 10-й класс. Углубленный уровень. В двух частях. М.: Бином, 2013.

4. *Поляков К.Ю., Еремин Е.А.* Информатика. 11-й класс. Углубленный уровень. В двух частях. М.: Бином, 2013.

5. Учебник "Информатика" 10–11-й классы (ФГОС, углубленный уровень) [Электронный ресурс]. URL: *http://kpolyakov.spb.ru/school/probook.htm* (дата обращения 18.05.2014).

6. Язык Python [Электронный ресурс]. URL:*http://kpolyakov.spb.ru/school/probook/python.htm* (дата обращения 18.05.2014).

7. *Сукин И.А.* Python, проглатывающий слона // Информатика, № 2, 2012, с. 22–42.

8. *Андреева Е.В.* Язык программирования Python для преподавания алгоритмизации и программирования в школьном курсе информатики [Электронный ресурс] URL: *http://www.myshared.ru/slide/270634/* (дата обращения 18.05.2014).

9. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2014 года по информатике и ИКТ [Электронный ресурс] URL: *http://fipi.ru/binaries/1517/Inf_ege14.zip* (дата обращения 18.05.2014).

10. Introduction to Computer Science and Programming Using Python [Электронный ресурс] URL: *https://www.edx.org/course/mitx/mitx-6-00-1x-introduction-computer-1841* (дата обращения 18.05.2014).

11. An Introduction to Interactive Programming in Python [Электронный ресурс] URL: *https://www.coursera.org/course/interactivepython* (дата обращения 18.05.2014).

12. CodeSkulptor [Электронный ресурс] URL: *http://www.codeskulptor.org* (дата обращения 18.05.2014).

13. Python [Электронный ресурс] URL: *http://www.codecademy.com/ru/tracks/python* (дата обращения 18.05.2014).

14. Язык программирования Python [Электронный ресурс] URL: *http://server.179.ru/~dk/python.html* (дата обращения 18.05.2014).

15. *Кириенко Д.П.* Программирование на Python [Электронный ресурс] URL: *http://informatics.mccme.ru/course/view.php?id=156* (дата обращения 18.05.2014).

Педагогический университет
«Первое сентября»

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

ДИСТАНЦИОННЫЕ КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

(с учетом требований ФГОС)

До 30 сентября производится прием заявок на 2014/15 учебный год

образовательные программы:

- НОРМАТИВНЫЙ СРОК ОСВОЕНИЯ – **108** УЧЕБНЫХ ЧАСОВ

  Стоимость – 3990 руб.

- НОРМАТИВНЫЙ СРОК ОСВОЕНИЯ – **72** УЧЕБНЫХ ЧАСА

  Стоимость – 3390 руб.

По окончании выдается удостоверение о повышении квалификации установленного образца

Перечень курсов и подробности – на сайте edu.1september.ru

Пожалуйста, обратите внимание:

заявки на обучение подаются только из Личного кабинета, который можно открыть на любом сайте портала www.1september.ru

Язык Python: избранные алгоритмы

К.Ю. Поляков,
д. т. н.,
Санкт-Петербург

Введение

Язык Python в последнее время приобретает все большую популярность как язык для обучения программированию. У него есть как достоинства, так и недостатки, детально рассмотренные в многочисленных публикациях, в том числе и в журнале “Информатика” [1–2].

Python — это более высокоуровневый язык, чем Паскаль и С. Прелесть Python состоит в том, что он позволяет рассматривать решение задачи на более высоком уровне абстракции, не особо отвлекаясь на технические вопросы. Поэтому Python весьма хорош для объяснения сути алгоритмов, и это может быть его серьезным преимуществом перед традиционными языками, используемыми для обучения. В отличие от псевдокода учащиеся могут свободно запускать и модифицировать программы на Python, в том числе и с помощью онлайн-сервисов.

В настоящей статье рассматривается реализация на языке Python некоторых алгоритмов, приведенных в учебнике информатики углубленного уровня для старшей школы [3–4]. Главы по программированию из этого учебника, переработанные для языка Python, размещены в открытом доступе на сайте поддержки [5].

Предполагается, что читатели знакомы с синтаксисом языка Python, по крайней мере в объеме статей [1–2]. Для более детального изучения затронутой темы можно рекомендовать книгу [6]. Программный код в статье написан на современной версии языка Python 3. Рабочие примеры использования всех приведенных в статье функций можно найти в приложении к этому номеру.

Решето Эратосфена

Во многих прикладных задачах, например при шифровании с помощью алгоритма RSA, используются простые числа. Основные вычислительные задачи при работе с простыми числами — это проверка числа на простоту и нахождение всех простых чисел в заданном диапазоне. Для решения второй из этих задач обычно используют метод, предложенный греческим математиком Эратосфеном Киренским (275–194 гг. до н.э.) и известный под названием “решето Эратосфена”:

1) выписать все числа от 2 до N;
2) начать с $k = 2$;
3) вычеркнуть все числа, кратные k, начиная с k^2;
4) найти следующее не вычеркнутое число и присвоить его переменной k;
5) повторять шаги 3 и 4, пока $k^2 \le N$.

Чтобы составить программу, нужно определить, что значит “выписать все числа” и “вычеркнуть число”. Один из возможных вариантов хранения данных — массив логических величин с индексами от 2 до N. Поскольку индексы элементов списков в Python всегда начинаются с нуля, для того чтобы работать с нужным диапазоном индексов (от 2 до N), удобно выделить логический массив из $N + 1$ элементов. Тогда, если число **i** не вычеркнуто, будем хранить в элементе массива A[i] истинное значение (True), а если вычеркнуто — ложное (False). В самом начале нужно заполнить массив истинными значениями:

```
A = [True] * (N + 1)
```

Алгоритм “решето Эратосфена” записывается в виде двойного цикла:

```
for k in range(2, int((N + 1)**0.5) + 1):
  if A[k]:
    for i in range(k * k, N + 1, k):
      A[i] = False
```

Переменная k изменяется в цикле от 2 до $\sqrt{N}$.

Результатом этой функции должен быть список простых чисел, тех, для которых значение соответствующего элемента массива A — истина (True). Такой список в Python удобно строить с помощью генератора:

```
return [i for i in range(2,N + 1) if A[i]]
```

Поскольку в нескольких местах программы встречается значение N + 1, имеет смысл заранее вычислить эту величину. Приведем окончательный вариант функции:

```
def primes(N):
  N1 = N + 1
  A = [True]*N1
  for k in range(2, int(N1**0.5) + 1):
    if A[k]:
      for i in range(k * k, N1, k):
        A[i] = False
  return [i for i in range(2,N1) if A[i]]
```

Интересен и другой вариант решения, который использует структуру данных “множество” (set). Сначала создаем множество всех чисел от 2 до N:

```
P = set(range(2, N + 1))
```

Затем, как и в предыдущем варианте, будем перебирать все числа k в интервале от 2 до $\sqrt{N}$ и, если очередное значение k входит во множество P (еще не вычеркнуто), удалять из множества все элементы, кратные k:

```
for k in range(2, int((N + 1)**0.5) + 1):
  if k in P:
    P -= set(range(k * k, N + 1, k))
```

Напомним, что вызов функции range(k * k, N + 1, k) перебирает все значения от k * k до N (включительно) с шагом k. Новая функция возвращает *множество*, содержащее только простые числа в заданном диапазоне:

```
def primesSet(N):
  N1 = N + 1
  P = set(range(2, N1))
  for k in range(2, int(N1**0.5) + 1):
    if k in P:
      P -= set(range(k * k, N1, k))
  return P
```

Отметим, что эта реализация решета Эратосфена проигрывает по скорости первому варианту примерно в 1,5 раза.

Извлечение квадратного корня

Рассмотрим задачу вычисления квадратного корня из большого целого числа, которое занимает более 64 бит в памяти. Как известно, в Python используется “длинная арифметика”, то есть объем памяти, отводимый под целое число, автоматически увеличивается, если это необходимо. К сожалению, стандартная функция **sqrt** из модуля **math**, вычисляющая квадратный корень, возвращает вещественное число. Возведение целого числа в степень 0,5 тоже дает вещественное число. Поэтому в том случае, когда нужно именно точное целое значение, приходится использовать специальные приемы.

Еще в Древней Греции был известен метод Герона Александрийского, который сводится к многократному применению формулы[1]

$$x_i = \frac{1}{2}\left(x_{i-1} + \frac{a}{x_{i-1}}\right).$$

Здесь a — число, из которого извлекается корень, а x_{i-1} и x_i — предыдущее и следующее приближения. Фактически здесь вычисляется среднее арифметическое между x_{i-1} и a/x_{i-1}. Пусть одно из этих значений меньше, чем $\sqrt{a}$, тогда второе обязательно больше, чем $\sqrt{a}$. Поэтому их среднее арифметическое с каждым шагом приближается к значению корня.

Метод Герона “сходится” (то есть приводит к правильному решению) при любом начальном приближении x_0, не равном нулю. Например, можно выбрать начальное приближение $x_0 = a$.

Приведенная формула служит для вычисления вещественного значения корня. Для того чтобы найти целочисленное значение корня (то есть *мак-*

[1] Эта формула — результат применения метода Ньютона для решения нелинейных уравнений к уравнению $x^2 = a$.

симальное целое число, квадрат которого не больше, чем *a*), можно заменить оба деления на целочисленные, на языке Python это запишется так:

```
x = (x + a // x)//2
```

— или привести выражение в скобках к общему знаменателю для того, чтобы использовать всего одно целочисленное деление:

```
x = (x * x + a)//(2 * x)
```

В итоге получается такая функция для вычисления квадратного корня:

```
def isqrt(a):
  x = a
  while True:
    x1 = (x * x + a)//(2 * x)
    if x1 >= x: return x
    x = x1
```

Здесь наиболее интересный момент — условие выхода из цикла. Цикл с заголовком[2]

```
while True:
  ...
```

представляет собой бесконечный цикл, из которого можно выйти только с помощью оператора **break** или (в функции) с помощью **return**. Мы начинаем поиск с начального приближения $x_0 = a$, которое (при больших *a*) заведомо больше, чем значение корня. Поэтому каждое следующее приближение будет меньше предыдущего. А как только очередное приближение оказывается *больше или равно* предыдущему, мы нашли квадратный корень. Если исходное число не является квадратом целого числа, функция вернет значение корня, округленное в меньшую сторону.

Факторизация чисел

В теории чисел существует одна весьма интересная задача, эффективный алгоритм решения которой позволил бы вскрывать практически все современные шифры, в том числе и RSA. Дано очень большое число *N*, которое представляет собой произведение двух простых чисел. Нужно найти оба сомножителя. Такое разложение на множители называют *факторизацией*.

Заметим, что решение обратной задачи — построить произведения двух больших целых чисел — не вызывает проблем, особенно при использовании современной вычислительной техники.

Общего эффективного решения задачи факторизации не найдено, но в некоторых специальных случаях решить ее можно даже для очень больших чисел. Например, если один из сомножителей — маленькое число, можно делить произведение на все числа, начиная с 2, пока деление не будет выполнено нацело.

Второй особый случай — близкие по величине делители. При этом они оказываются близки к квадратному корню из числа. Тогда для поиска делителей можно использовать метод Пьера Ферма. Идея метода состоит в том, чтобы найти такие числа *x* и *c*, что

$$x^2 - c^2 = (x + c)(x - c) = N,$$

тогда сомножителями числа *N* будут числа $x - c$ и $x + c$. Это значит, что достаточно найти такое *x*, что $x^2 - N$ — полный квадрат некоторого числа.

Будем предполагать, что само число *N* не является полным квадратом, то есть сомножители различны. Поиск начинается со значения $x = \sqrt{N} + 1$ (это минимальное значение *x*, при котором разность $x^2 - N$ положительна) и продолжается до тех пор, пока не будет найдено нужное значение. С каждым шагом *x* увеличивается на единицу:

```
def factor ( A ):
  X = isqrt(A)
  while True:
    X += 1
    Z = X * X - A
    C = isqrt(Z)
    R = Z - C * C
    if R == 0: return (X - C, X + C)
```

Эта функция возвращает кортеж, содержащий два найденных сомножителя. Для вычисления квадратного корня используется функция `isqrt`, рассмотренная в предыдущем пункте. Вызывать функцию можно, например, так:

```
x, y = factor(156194422758106897)
```

Для данного числа мы должны получить сомножители 384 160 001 и 406 586 897.

Быстрая сортировка

В школьном курсе чаще всего изучается только сортировка “пузырьком” и сортировка выбором, имеющие квадратичную сложность (время работы алгоритма при больших размерах массива увеличивается пропорционально квадрату размера массива). К сожалению, объяснять школьникам более эффективные методы сортировки на Паскале или Си достаточно сложно из-за обилия нетривиальных технических деталей их реализации. Использование языка Python позволяет во многом решить эту проблему за счет перехода на более высокий уровень абстракции в записи алгоритма.

Один из самых популярных “быстрых” алгоритмов, разработанный в 1960 году английским ученым Чарльзом Хоаром, так и называется — “быстрая сортировка” (англ. *quicksort*). На естественном языке его можно записать так:

1) выберем некоторый элемент массива `P`;

2) разделим массив на три части: в первую включим элементы, меньшие `P`, во вторую — элементы, равные `P`, а в третью — элементы, большие `P`;

3) отсортируем первую и третью части массива таким же способом.

Этот алгоритм очень красиво и коротко записывается на Python:

```
def quickSort ( A ):
  if len(A) < 2: return A
  L, X, R = partition ( A )
  return quickSort(L) + X + quickSort(R)
```

Это рекурсивная функция, которая дважды вызывает сама себя в последней строке для того, чтобы отсортировать две оставшиеся части массива. Такой подход называется “разделяй и властвуй”

[2] Здесь и далее многоточие обозначает некоторые операторы.

(англ. *divide and conquer*). Рекурсия заканчивается, когда в массиве остается меньше двух элементов, тогда и сортировать нечего, поэтому функция возвращает исходный массив.

Функция `partition` возвращает кортеж из трех списков: список `L` содержит все элементы, меньшие, чем случайно выбранный элемент `P` переданного ей массива; список `X` содержит все элементы, равные `P`, а список `R` — все элементы, большие `P`:

```
def partition ( A ):
  P = random.choice(A)
  L = [x for x in A if x < P]
  X = [x for x in A if x == P]
  R = [x for x in A if x > P]
  return L, X, R
```

Для случайного выбора элемента применяется функция `choice` из модуля `random`.

Сортировка слиянием

Еще один популярный алгоритм “быстрой” сортировки — сортировка слиянием (англ. *merge sort*) — тоже использует подход “разделяй и властвуй”, но по-другому. Массив делится на две половины, каждая из половин сортируется отдельно, а затем два полученных массива сливаются так, чтобы получился один отсортированный массив:

```
def mergeSort ( A ):
  if len(A) < 2: return A
  mid = len(A) // 2
  L = mergeSort(A[:mid])
  R = mergeSort(A[mid:])
  return merge ( L, R )
```

Здесь `mid` — это индекс среднего элемента массива, срез `A[:mid]` обозначает все элементы первой половины (не включая элемент с индексом `mid`), а срез `A[mid:]` — элементы второй половины. В последней строке вызывается функция `merge`, которая сливает два массива:

```
def merge ( a, b ):
  res = []
  while a and b:
    if a[-1] >= b[-1]:
          res.append(a.pop())
    else: res.append(b.pop())
  res.reverse()
  return a + b + res
```

Цикл

```
while a and b:
  ...
```

выполняется до тех пор, пока оба массива непустые. Поскольку для списков в Python эффективно (за постоянное время) выполняются операции добавления и удаления *последнего* элемента, мы построим массив, отсортированный по убыванию (невозрастанию), а в конце развернем его с помощью метода `reverse`. На каждом шаге цикла сравниваем два последних элемента массивов, `a[-1]` и `b[-1]`, и добавляем в конец массива-результата `res` наибольший из них.

После окончания цикла массив-результат переворачивается с помощью метода `reverse`, а затем в его начало добавляется то, что осталось от исходных массивов `a` и `b`:

```
return a + b + res
```

Вспомним, что один из массивов (`a` или `b`) — пустой, потому что закончился цикл. Кроме того, элементы оставшегося массива отсортированы в порядке возрастания (неубывания) и все они не больше, чем любой элемент массива `res`, построенного в цикле. Поэтому список `a+b+res`, который возвращает функция, тоже отсортирован по возрастанию (неубыванию).

Двоичный поиск

Для поиска элемента в небольших массивах обычно применяют *линейный* поиск, который сводится к последовательному просмотру всех элементов. Если число элементов (например, записей в базе данных) очень велико, время линейного поиска может оказаться недопустимо большим.

Поиск можно значительно ускорить, если данные предварительно отсортировать, например, по возрастанию (неубыванию). Тогда применим *двоичный* поиск: берем средний элемент массива и сравниваем его с тем, который мы ищем. Если средний элемент больше, чем нужный, ищем далее только в первой половине массива, если меньше — то только во второй. Таким образом, на каждом шаге область поиска сужается вдвое, и, для того чтобы найти элемент в массиве из N элементов, требуется примерно $\log_2 N$ сравнений (вместо N сравнений при линейном поиске).

Будем предполагать, что массив непустой. Введем переменные `L` и `R`, которые ограничивают “рабочую зону” массива: `L` содержит номер первого элемента, а `R` — номер элемента, *следующего* за последним. Таким образом, нужный элемент (если он есть) находится в части массива `A[L:R]` (она начинается с элемента `A[L]` и заканчивается элементом `A[R - 1]`).

На каждом шаге вычисляется номер среднего элемента “рабочей зоны”, он записывается в переменную `m`. Если `X < A[m]`, то нужный элемент `X` может находиться только левее `A[m]`, и правая граница `R` перемещается в `m`. В противном случае `X` находится правее середины или совпадает с `A[m]`; при этом левая граница `L` перемещается в `m`.

Поиск заканчивается при выполнении условия `L = R – 1`, когда в рабочей зоне остался один элемент. Если при этом `A[L] = X`, то в результате найден элемент, равный `X`, иначе такого элемента нет.

Приведем реализацию этого алгоритма на Python в виде функции:

```
def binSearch(A, X, L, R):
  while L < R - 1:
    m = (L + R) // 2
    if A[m] > X:
          R = m
    else: L = m
  return L if A[L] == X else -1
```

Функция возвращает номер найденного элемента массива или “–1”, если элемент не найден. После завершения цикла срабатывает оператор

```
  return L if A[L] == X else -1
```
равносильный записи
```
  if A[L] == X:
        return L
  else: return -1
```
Интересен также и рекурсивный вариант этой функции, который может оказаться более понятным для объяснения:
```
  def binSearchRec(A, X, L, R):
    if L == R-1:
      return L if A[L] == X else -1
    m = (L + R) // 2
    if A[m] > X:
      return binSearchRec(A, X, L, m)
    else:
      return binSearchRec(A, X, m, R)
```
Если `L` совпадает с `R - 1`, в массиве остался один элемент. Функция возвращает его номер `L`, если он равен искомому `X`, или "–1" в противном случае. Если элементов несколько, находим середину массива. Если этот средний элемент больше, чем `X`, продолжаем поиск в первой половине массиве, иначе — во второй.

Если необходимо, чтобы эти функции работали и для пустого массива, в каждую из них нужно добавить первую строчку
```
  if not A: return -1
```

Стеки

Стек (англ. *stack* — груда, кипа, куча) — это линейный список, в котором элементы добавляются и удаляются только с одного конца ("последним пришел — первым ушел", англ. *LIFO = **L**ast **I**n, **F**irst **O**ut*). Системный стек используется при выполнении программ: в этой области оперативной памяти хранятся адреса возврата из подпрограмм; параметры, передаваемые функциям и процедурам, а также локальные переменные. Кроме того, стек как структура данных полезен при решении многих практических задач, две из которых мы рассмотрим ниже.

Сначала обратимся к проблеме вычисления арифметических выражений. "Обычная" запись, в которой знак операции записывают между операндами, неудобна для автоматических вычислений, потому что невозможно вычислить выражение за один проход, мешают скобки и приоритет операций (умножение и деление выполняются раньше, чем сложение и вычитание).

В 1920 году польский математик Ян Лукашевич предложил *префиксную* форму, которую стали называть *польской нотацией*. В ней знак операции расположен *перед* операндами. Например, выражение `(5 + 15)/(4 + 7 - 1)` может быть записано в виде `/ + 5 15 – + 4 7 1`. Скобок здесь не требуется, так как порядок операций строго определен: сначала выполняются два сложения (`+ 5 15` и `+ 4 7`), затем вычитание и, наконец, деление. Первой стоит последняя операция.

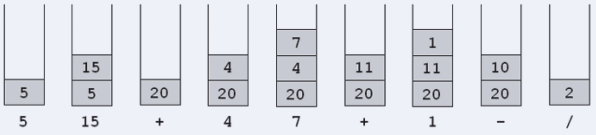
В середине 1950-х годов была предложена *обратная польская нотация*, или *постфиксная* форма записи, в которой знак операции стоит *после* операндов:

`5 15 + 4 7 + 1 – /`

В этом случае также не нужны скобки, и выражение может быть вычислено за один просмотр с помощью стека следующим образом:

- если очередной элемент — число (или переменная), он записывается в стек;
- если очередной элемент — операция, то она выполняется с верхними элементами стека, и после этого в стек вталкивается результат выполнения этой операции.

Покажем, как работает этот алгоритм для выражения `5 15 + 4 7 + 1 – /`, записанного в обратной польской нотации. Предполагается, что стек "растет" снизу вверх.



После окончания вычислений в стеке остается значение заданного выражения, равное 2.

Самый простой вариант организации стека в языке Python — это список. Вершина стека находится в конце списка, потому что в этом случае операции добавления и удаления элементов не требуют перемещения в памяти всех оставшихся элементов. Для добавления элемента используется метод `append`, а для удаления — метод `pop`, возвращающий значение элемента, снятого с вершины стека.

Функция `calcPostfix` принимает список элементов постфиксной записи арифметического выражения в виде символьных строк, например, такой:
```
  data = ["5", "15", "+", "4", "7", "+",
          "1", "-", "/"]
```
— и вычисляет его значение:
```
  def calcPostfix ( data ):
    stack = []                          # 1
    for x in data:                      # 2
      if x in "+-*/":                   # 3
        op2 = stack.pop()               # 4
        op1 = stack.pop()               # 5
        if   x == "+": res = op1 + op2 # 6
        elif x == "-": res = op1 – op2 # 7
        elif x == "*": res = op1 * op2 # 8
        else:          res = op1 // op2 # 9
        stack.append(res)               # 10
      else:
        stack.append(int(x))            # 11
    return stack[0]
```
Разберем эту функцию подробнее. В строке 1 создается пустой список-стек. Затем в цикле перебираем все элементы массива `data` (строка 2). Если очередной элемент — это знак арифметической операции (строка 3), снимаем со стека два верхних элемента (строки 4–5) и выполняем указанную операцию между ними (строки 6–9). Результат снова записывается в стек (строка 10). Если это не операция, то число, и его нужно записать в стек (строка 11).

Функция возвращает значение элемента стека с индексом 0, он должен быть единственным в стеке.

Отметим, что в этом простейшем варианте функция не обрабатывает возможные ошибки в переданной ей постфиксной записи.

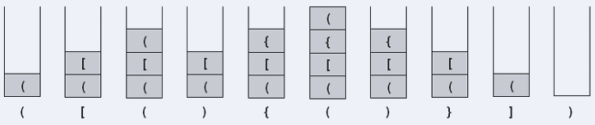
Рассмотрим еще одну задачу, в решении которой используется стек. Дана символьная строка, в которой записано некоторое выражение, использующее скобки трех типов: `()`, `[]` и `{}`. Нужно проверить, правильно ли расставлены скобки. Например, выражение `()[{()[]}]` — правильное, потому что каждой открывающей скобке соответствует закрывающая и вложенность скобок не нарушается. Выражения

`[()  [[[()  [()}  )(  ([)]`

неправильные. В первых трех есть непарные скобки, а в последних двух не соблюдается вложенность скобок.

Задачи, в которых важна вложенность объектов, удобно решать с помощью стека. Нас интересуют только открывающие и закрывающие скобки, на остальные символы можно не обращать внимания.

Исходная строка просматривается слева направо. Если очередной символ — открывающая скобка, нужно втолкнуть ее на вершину стека. Если это закрывающая скобка, то проверяем, что лежит на вершине стека: если там соответствующая открывающая скобка, то ее нужно просто снять со стека. Если стек пуст или на вершине лежит открывающая скобка другого типа, выражение неверное и нужно закончить просмотр. В конце обработки правильной строки стек должен быть пуст. Кроме того, во время просмотра не должно быть ошибок. Работа такого алгоритма для правильного выражения "`([(){()}])`" иллюстрируется на рисунке:



Оформим алгоритм проверки выражения в виде функции, которая возвращает логическое значение `True` (истина), если скобки расставлены правильно, и `False`, если неправильно:

```
def brackets ( s ):
  ...
```

Здесь многоточие обозначает операторы, которые будут написаны далее.

Введем строки `L` и `R`, которые содержат все виды открывающих и соответствующих закрывающих скобок:

```
L = "([{"
R = ")]}"
```

— и создадим пустой стек

```
stack = []
```

В основном цикле перебираем все символы строки `s`, в которой записано скобочное выражение:

```
for c in s:
  ...
```

Сначала мы ищем очередной символ строки `s` (который попал в переменную `c`) в строке `L`, то есть среди открывающих скобок. Если это действительно открывающая скобка, вталкиваем ее в стек:

```
if p in L:
  stack.append(c)
```

Далее ищем символ среди закрывающих скобок. Если нашли, то в первую очередь проверяем, не пуст ли стек. Если стек пуст, выражение неверное, и функция возвращает ложное значение:

```
p = R.find(c)
if p >= 0:
  if not stack: return False
  else:
    top = stack.pop()
    if p!= L.find(top):
      return False
```

Если в стеке что-то есть, снимаем символ с вершины стека в переменную `top`. Затем сравнивается тип (номер) закрывающей скобки `p` и номер открывающей скобки, найденной на вершине стека. Если они не совпадают, выражение неправильное, и функция сразу возвращает `False`.

После окончания цикла нужно проверить содержимое стека: если он не пуст, то в выражении есть незакрытые скобки, и оно ошибочно. Поэтому оператор возврата выглядит так:

```
return len(stack) == 0
```

Приведем полную запись функции на языке Python:

```
def brackets ( s ):
  L = "([{"
  R = ")]}"
  stack = []
  for c in s:
    if p in L: stack.append(c)
    p = R.find(c)
    if p >= 0:
      if not stack: return False
      else:
        top = stack.pop()
        if p!= L.find(top):
          return False
  return len(stack) == 0
```

Очереди

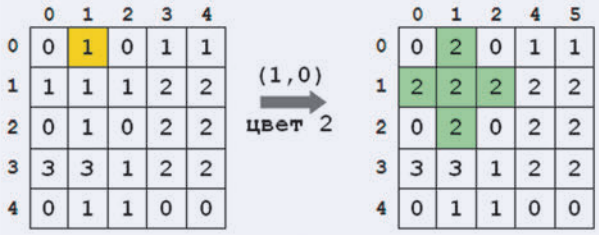
Очередь — это модель данных, для которой выполняется принцип "первым пришел — первым обслужен" (англ. *FIFO = **F**irst **I**n – **F**irst **O**ut*). Для очереди введены две операции:

- добавление нового элемента в конец очереди;
- удаление первого элемента из очереди.

Очередь — это не просто теоретическая модель. Операционные системы используют очереди для организации сообщения между программами: каждая программа имеет свою очередь сообщений. Контроллеры жестких дисков формируют очереди запросов ввода и вывода данных. В сетевых маршрутизаторах создается очередь из пакетов данных, ожидающих отправки.

Одна из задач, при решении которых можно применить очередь, — заливка одноцветной области рисунка. Пусть рисунок задан в виде матрицы `A`, в которой элемент `A[y][x]` определяет цвет пикселя на пересечении строки `y` и столбца `x`. Нужно перекрасить в новый цвет одноцветную область, начиная с пикселя (x_0,y_0). На рисунке показан результат

такой заливки для матрицы из пяти строк и пяти столбцов. Заливка цветом 2 начинается с точки (1,0), которая выделена желтым фоном.



Один из возможных вариантов решения этой задачи использует очередь, элементы которой — координаты пикселей (точек):

```
добавить в очередь точку (x0,y0)
color = цвет начальной точки
while очередь не пуста:
  взять из очереди точку (x,y)
  if A[y][x] == color:
    A[y][x] = новый цвет
    добавить в очередь точку (x - 1,y)
    добавить в очередь точку (x + 1,y)
    добавить в очередь точку (x,y - 1)
    добавить в очередь точку (x,y + 1)
```

Конечно, в очередь нужно добавлять только те точки, которые находятся в пределах рисунка (матрицы A). Заметим, что в этом алгоритме некоторые точки могут быть добавлены в очередь несколько раз. Поэтому решение можно несколько улучшить, помечая точки, уже добавленные в очередь, чтобы не добавлять их повторно.

Пусть изображение записано в виде матрицы A, которая на языке Python представлена как список списков (каждый внутренний список — отдельная строка матрицы). Тогда можно определить размеры матрицы так:

```
YMAX = len(A)
XMAX = len(A[0])
```

Значение YMAX — это число строк, а XMAX — число столбцов.

Определим также цвет заливки:

```
NEW_COLOR = 2
```

Зададим координаты начальной точки, откуда начинается заливка:

```
x0 = 1
y0 = 0
```

— и запомним ее цвет в переменной **color**:

```
color = A[y0][x0]
```

Теперь создадим очередь (как список языка Python) и добавим в эту очередь точку с начальными координатами. Две координаты точки связаны между собой, поэтому в программе лучше объединить их в единый блок, который в Python называется "*кортеж*" и заключается в круглые скобки. Таким образом, каждый элемент очереди — это кортеж из двух элементов:

```
Q = [ (x0,y0) ]
```

Остается написать основной цикл:

```
while Q:                                  # 1
  x, y = Q.pop(0)                         # 2
  if A[y][x] == color:                    # 3
    A[y][x] = NEW_COLOR                   # 4
    if x > 0: Q.append( (x - 1,y) )
    if x < XMAX - 1: Q.append((x + 1,y))
    if y > 0: Q.append((x,y - 1))
    if y < YMAX - 1: Q.append((x,y + 1))
```

Начало очереди всегда совпадает с первым элементом списка (имеющим индекс 0). Цикл работает до тех пор, пока в очереди есть хоть один элемент (строка 1).

В строке 2 первый элемент удаляется из очереди. Как мы уже говорили, элемент очереди — это кортеж из двух элементов-координат, поэтому мы сразу разбиваем его на отдельные координаты x и y, используя множественное присваивание.

Если цвет текущей точки совпадает с цветом начальной точки, который хранится в переменной **color** (строка 3), эта точка закрашивается новым цветом (строка 4), и в очередь добавляются все точки, граничащие с текущей и попадающие на поле рисунка.

Алгоритм заливки можно оформить в виде функции, которая выполняет заливку в матрице A цветом NEW_COLOR, начиная с точки p0 (две координаты объединены в кортеж):

```
def fill ( A, p0, NEW_COLOR ):
  YMAX = len(A)
  XMAX = len(A[0])
  color = A[p0[1]][p0[0]]
  Q = [p0]
  while Q:
   x, y = Q.pop(0)
   if A[y][x] == color:
     A[y][x] = NEW_COLOR
     if x > 0: Q.append ( (x - 1,y))
     if x < XMAX - 1: Q.append ((x + 1,y))
      if y > 0: Q.append ((x,y - 1))
    if y < YMAX - 1: Q.append ((x,y + 1))
```

Вызвать ее можно так:

```
x0 = 1
y0 = 0
NEW_COLOR = 2
fill ( A, (x0,y0), NEW_COLOR )
```

Использование списка для моделирования очереди — не самый быстрый вариант, потому что удаление первого элемента списка выполняется долго (требует перемещения остальных элементов в памяти). Поэтому в практических задачах лучше использовать класс deque из модуля collections. Соответствующий пример приведен в приложении к этому номеру.

Волновой алгоритм

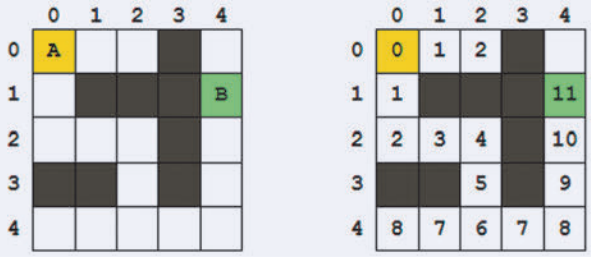
Волновой алгоритм (алгоритм Ли) — это алгоритм поиска кратчайшего пути между двумя точками в лабиринте. Он применяется при компьютерном проектировании печатных плат и микросхем, а также для поиска кратчайшего маршрута в компьютерных играх.

Предположим, что лабиринт имеет размер $H \times W$ клеток (H — высота лабиринта, а W — ширина). Начальную клетку отметим числом 0. Далее отметим числом 1 все клетки, в которые можно перейти из

начальной за один шаг. После того, как отмечены все клетки, куда можно попасть из начальной за d шагов, снова просматриваем все клетки и для каждой, помеченной числом d, отмечаем все свободные соседние клетки поля числом $d + 1$. На псевдокоде волновой алгоритм запишется так:

```
пометить начальную клетку числом 0
while конечная клетка не помечена
  and волна распространяется:
  для каждой ячейки, помеченной числом d:
    пометить все соседние непомеченные
    клетки числом d + 1
 d += 1
```

Если конечная клетка недостижима, например, ограничена стенками со всех сторон, нужно как-то определить момент, когда волна остановится. Для этого можно считать клетки, отметка которых изменилась на очередном шаге цикла. Если ни для одной клетки отметка не изменилась, волна остановилась. Пример распространения волны из точки A в точку B в лабиринте размером 5 × 5 показан на рисунке:



Напишем функцию solveMaze, которая ищет путь в лабиринте с помощью волнового алгоритма. Будем считать, что лабиринт хранится как матрица maze, в каждой ячейке которой может быть записано число “–2” (которое обозначает свободную клетку) или “–1” (стенка).

```
EMPTY = -2  # пустая клетка
WALL = -1   # стенка
```

Кроме самого лабиринта, функция принимает в качестве параметров координаты начальной клетки (кортеж start) и конечной клетки, в которую нужно попасть (кортеж goal).

```
def solveMaze ( maze, start, goal ):
  ...
```

Итак, сначала помечаем начальную клетку числом 0:

```
d = 0
maze[start[0]][start[1]] = d
marked = 1
```

Переменная marked обозначает количество клеток, до которых дошла волна на последнем шаге.

Затем на каждом шаге цикла просматриваем все клетки поля:

```
for i in range(H):
  for j in range(W):
    if maze[i][j] == d:
      marked += (mark(i - 1,j,d)
                 + mark(i + 1,j,d)
                 + mark(i,j - 1,d)
                 + mark(i,j + 1,d))
```

Если очередная клетка с координатами (i,j) отмечена числом d, пытаемся пометить все соседние клетки, они имеют координаты (i - 1,j), (i + 1,j), (i,j - 1) и (i,j + 1). Для этого вызывается внутренняя функция mark, которая проверяет допустимость координат клетки и помечает пустые клетки:

```
def mark ( i, j, d ):
  if (0 <= i < H and 0 <= j < W
      and maze[i][j] == EMPTY):
    maze[i][j] = d + 1
    return 1
  else: return 0
```

Эта функция возвращает 1, если отметка клетки изменилась (волна распространяется), и 0, если этого не произошло.

В конце очередного шага цикла увеличиваем значение d и проверяем, не дошли ли мы уже до конечной клетки (в этом случае функция сразу возвращает полученную длину пути):

```
d += 1
if maze[goal[0]][goal[1]] > 0:
  return d
```

Приведем функцию solveMaze полностью:

```
def solveMaze ( maze, start, goal ):
  def mark ( i, j, d ):
    if (0 <= i < H and 0 <= j < W and
        maze[i][j] == EMPTY):
      maze[i][j] = d + 1
      return 1
    else: return 0
  maze[start[0]][start[1]] = 0
  d = 0
  marked = 1
  while marked > 0:
    marked = 0
    for i in range(H):
      for j in range(W):
        if maze[i][j] == d:
          marked += (mark(i - 1,j,d)
                     + mark(i + 1,j,d)
                     + mark(i,j - 1,d)
                     + mark(i,j + 1,d))
    d += 1
    if maze[goal[0]][goal[1]] > 0:
      return d
  return -1
```

В случае неудачи (если цель не достигнута и при очередном просмотре поля волна не продвинулась) функция возвращает “–1”.

Недостаток этого алгоритма состоит в том, что на каждом шаге основного цикла мы просматриваем все ячейки в двойном цикле. Несложно заметить, что рассмотренная задача очень напоминает задачу о заливке области, рассмотренную в предыдущем пункте. И действительно, быстродействие может быть улучшено, если составить очередь из клеток “переднего фронта” волны, на каждом шаге выбирать первый элемент очереди и помечать все соседние клетки, добавляя их в очередь. Таким образом, просматривать каждый раз *все* ячейки поля уже не нужно. Полу-

чается такая реализация волнового алгоритма с очередью Q:

```
def solveMaze ( maze, start, goal ):
  def mark ( i, j, d ):
    nonlocal Q
    if (0 <= i < H and 0 <= j < W and
        maze[i][j] == EMPTY):
      maze[i][j] = d + 1
      Q.append( (i,j) ) # добавить
                        # клетку в очередь
  Q = [start]   # начальная клетка в очереди
  maze[start[0]][start[1]] = 0
  while Q: # пока очередь не пуста
    i, j = Q.pop(0) # взять первый элемент
    d = maze[i][j] # прочитать его метку
    mark(i - 1,j,d) # пометить соседние клетки
    mark(i + 1,j,d)
    mark(i,j - 1,d)
    mark(i,j + 1,d)
    if maze[goal[0]][goal[1]] > 0:
      return d + 1
  return -1
```

Теперь цикл заканчивается тогда, когда очередь пуста. Это значит, что волне дальше некуда распространяться. Внутренняя функция `mark` не возвращает никакого значения, оно просто не нужно.

Очередь `Q` внутри функции `mark` описана с помощью ключевого слова `nonlocal`:

```
nonlocal Q
```

Это означает, что она не локальная (но и не глобальная!), то есть принадлежит внешней функции.

Остается определить сам кратчайший маршрут. Предположим, что клетки лабиринта (включая целевую) помечены числами, определяющими последовательность распространения волны. Тогда восстановить путь можно с помощью следующего алгоритма:

```
перейти в целевую клетку
d = метка целевой клетки
while не пришли в начальную клетку
  d = d – 1
  выбрать среди соседних клетку,
     помеченную числом d
  перейти в выбранную клетку
     и добавить ее к пути
```

Реализация на языке Python выглядит так:

```
i, j = goal
d = maze[i][j]
path = [goal]
while d > 0:
  d -= 1
  i, j = move ( d )
  path = [(i,j)] + path
```

Координаты целевой клетки находятся в кортеже `goal`, в переменных `i` и `j` хранятся координаты текущей клетки. Путь накапливается в виде списка клеток `path`, в котором каждая клетка представлена как кортеж из двух координат. Поскольку путь “раскручивается” с конца, новая клетка добавляется в начало списка. Внутренняя функция `move` делает очередной шаг и возвращает новые текущие координаты:

```
def move ( d ):
  for k, m in [(i + 1,j), (i - 1,j),
               (i,j + 1), (i,j - 1)]:
    if (0 <= k < H and 0 <= m < W and
        maze[k][m] == d):
      return (k, m)
```

В цикле

```
for k, m in [(i + 1,j), (i - 1,j),
             (i,j + 1), (i,j - 1)]:
  ...
```

перебираются все соседние ячейки, первая координата каждой из них попадает в переменную `k`, вторая — в переменную `m`. Когда нужная клетка найдена, функция `move` возвращает кортеж `(k, m)` — координаты следующей клетки маршрута. Полностью функция `findPath`, которая строит оптимальный путь, выглядит так:

```
def findPath ( maze, goal ):
  def move ( d ):
    for k, m in [(i + 1,j), (i - 1,j),
                 (i,j + 1), (i,j - 1)]:
      if (0 <= k < H and
          0 <= m < W and maze[k][m] == d):
        return (k, m)
  i, j = goal
  d = maze[i][j]
  path = [goal]
  while d > 0:
    d -= 1
    i, j = move ( d )
    path = [(i,j)] + path
  return path
```

При желании можно сделать эту функцию внутренней функцией для `solveMaze`, так чтобы функция `solveMaze` сразу возвращала оптимальный путь.

Литература

1. *Сукин И.А.* Python, проглатывающий слона // Информатика, № 2, 2012, с. 22–42.
2. *Поляков К.Ю.* Язык Python глазами учителя // Информатика, № 9, 2014, с. 4–16.
3. *Поляков К.Ю., Еремин Е.А.* Информатика. 10-й класс. Углубленный уровень. В двух частях. М.: Бином, 2013.
4. *Поляков К.Ю., Еремин Е.А.* Информатика. 11-й класс. Углубленный уровень. В двух частях. М.: Бином, 2013.
5. Язык Python [Электронный ресурс] URL: *http://kpolyakov.spb.ru/school/probook/python.htm* (дата обращения 18.05.2014).
6. *Hetland M.L.* Python Algorithms: Mastering Basic Algorithms in the Python Language, Apress, 2010.
7. Knapsack problem/0–1 [Электронный ресурс] URL: *http://rosettacode.org/wiki/Knapsack_problem/0-1* (дата обращения 18.05.2014).

Окончание читайте в следующем номере.

Общероссийский проект

Школа цифрового века

Издательский дом «ПЕРВОЕ СЕНТЯБРЯ» • Издательство «ПРОСВЕЩЕНИЕ»

Каждый педагогический работник образовательной организации, вошедшей в проект «Школа цифрового века», получает доступ ко всем материалам проекта по принципу «все включено» (без дополнительной платы)

МАТЕРИАЛЫ ПРОЕКТА

- **23 предметно-методических журнала** по всем предметам и направлениям школьной жизни плюс журнал для родителей
- **Модульные дистанционные курсы*** из циклов «Навыки профессиональной и личной эффективности педагога» и «Инклюзивный подход в образовании»
- **Дистанционные 36-часовые курсы**** повышения квалификации с выдачей удостоверения установленного образца
- **Методические брошюры** по всем школьным предметам

Стоимость участия образовательной организации в проекте – 6 тысяч рублей за весь учебный год. Стоимость участия не зависит от количества педагогических работников в образовательной организации

Участие образовательной организации и педагогических работников в проекте удостоверяется соответствующими документами. Для дошкольных организаций предусмотрен свой набор удостоверяющих документов

Срок действия проекта в 2014/15 учебном году: с 1 августа 2014 года по 30 июня 2015 года

Продление участия и прием новых заявок от образовательных организаций продолжается

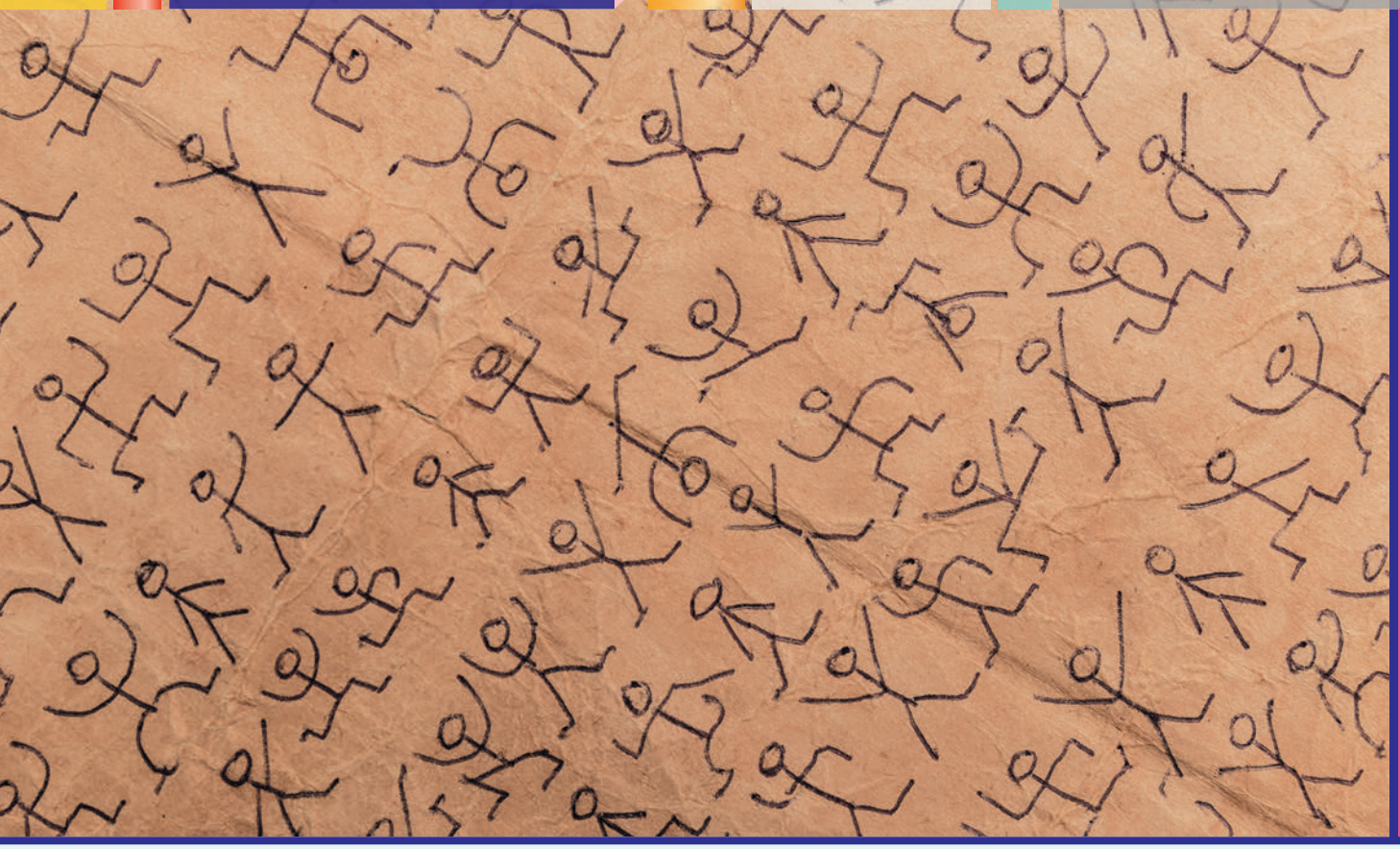
Подробности на сайте

digital.1september.ru

---

* в течение указанного срока предоставляются без ограничения количества курсов

** предоставляется по одному курсу для одного педагогического работника в течение одного учебного года (выбор конкретного курса – на усмотрение педагога)



Криптография

И.А. Калинин,
к. п. н., доцент
Института математики
и информатики МГПУ,
доцент кафедры
информатики
и прикладной
математики

Н.Н. Самылкина,
к. п. н., доцент
Московского
педагогического
государственного
университета,
профессор кафедры
теории и методики
обучения информатике

Сложно даже представить среди компьютерных наук более романтичную тему: тайны, интриги, шпионы, политика, детективы, кино со всеми его штампами... Все мы это видели и слышали. Если не интересоваться вопросом специально — кажется, что это и теперь атрибут деятельности государства, не очень касающийся нас в обычной жизни.

Как ни покажется странным тому, кто этим не занимался, за последние 40 лет ситуация изменилась в корне: теперь это очень даже касается всех нас. Так что будет весьма полезно знать: почему, каким образом и о чем же на самом деле идет речь.

Слово “криптография” греческого происхождения и означает (в прямом переводе) “скрытое письмо”. Так это слово и определялось в словарях: скрытое, то есть доступное и понятное не всем, письмо. Цель такого способа записи: передать сообщение таким образом, чтобы, кроме адресата, его никто не мог прочесть.

Способов такой записи с древности накопилось очень много. Если проанализировать эти способы, то будут видны два основных подхода к решению:

1. Сделать сообщение незаметным.
2. Сделать сообщение непонятным без специальных действий.

Первый подход активно применяется и сейчас, дисциплина, которая этим занимается, — *стеганография*. Мы не будем о нем здесь говорить, поскольку он не может решить большинство задач, которые нас будут интересовать.

Второй же подход превратился сейчас в дисциплину, которая и называется “криптография”.

Теоретическое введение

Зафиксируем несколько терминов, которыми будем постоянно пользоваться:

1. Открытый текст, простой текст — понятное сообщение, которое можно зашифровать.

2. Шифр — способ преобразования сообщения, при котором оно становится “непонятным”.

3. Закрытый текст, шифртекст, криптограмма — текст, получающийся в результате шифрования.

4. Зашифровать сообщение — применить шифр, то есть преобразовать сообщение.

5. Расшифровать сообщение — выполнить обратную операцию, то есть преобразовать зашифрованное сообщение в исходную, понятную форму.

6. Ключ шифра — информация, необходимая, чтобы правильно расшифровать сообщение: конкретный вид шифровальной решетки, таблица подстановки букв и т.п.

До начала XX века криптография была задачей узкой, интересовавшей большей частью государства. Криптографические способы защиты информации активно разрабатывались, применялись и изучались — в первую очередь для того, чтобы читать чужие сообщения.

История криптографии, вместе с простейшими шифрами, подробно и разнообразно рассказывалась многими авторами, в том числе и на страницах “Информатики”, так что повторяться мы не будем, а обратимся к современному состоянию.

Сразу оговоримся: современная криптография — это в первую очередь специализированная (и очень развитая) область математики. Мы не сможем строго (а тем более — доказательно) изложить ее теорию, даже обзорно, — мы просто расскажем об основных подходах и их применении.

Криптография как строгая научная дисциплина возникла не так давно — в 1945 году, в работах Клода Элвуда Шеннона[1], который и заложил основы и современной криптографии, и теории связи. Шеннон впервые предложил рассматривать шифрование некоторого текста как математическое преобразование: функцию, отображающую множество всех возможных текстов-сообщений во множество всех возможных шифрованных текстов.

Этой функцией и будет шифр. В общем виде:

$$M' = A(K, M)$$

То есть шифртекст (M') — это результат вычисления функции-шифра (A) от двух параметров: от ключа и исходного, открытого текста.

То есть появилась возможность не просто угадывать и подбирать какие-то действия, чтобы затруднить чтение противнику или прочесть его текст, а решать задачу в общем виде — изучать свойства таких функций и заранее предполагать их слабые места.

Первое, что можно сделать, — это предъявить требования к функции-шифру.

Мы знаем, что назначение шифра — сохранить сообщение от прочтения кем-то чужим. Но от чего конкретно мы его защищаем? Какие угрозы для нас будут актуальны? Для криптографии ответ на этот вопрос — модель угроз Долева — Яо, то есть *уязвимая среда*. Мы считаем, что сообщения передаются в открытой всем среде, в которой есть добросовестный получатель, а есть злоумышленник. Злоумышленник:

- может перехватить любое сообщение;
- является законным пользователем среды обмена, может вступить в контакт с любым пользователем;
- может получить сообщения от другого пользователя;
- может послать сообщение любому пользователю, маскируясь под любого другого пользователя.

Как видим, злоумышленник может многое — но он не всесилен. Он:

- не может угадать случайное число, если его выбирают из достаточно большого множества;
- если у нас есть идеальный алгоритм шифрования — злоумышленник не может, не имея ключа, расшифровать или зашифровать наше сообщение;
- злоумышленник не имеет доступа к закрытым областям вычислительной среды: например, не может проникнуть в память автономной системы.

Вот для такой среды мы должны создать стойкий шифр — т.е. такой, который не даст злоумышленнику прочесть наше сообщение — даже с учетом того, что он может сообщение перехватить. Есть ли такой шифр в принципе? Ключевая работа Клода Шеннона, о которой мы уже говорили, отвечает: ***ДА***. Такой шифр называется *абсолютно стойким*.

Шеннон доказал, что если:

- ключ случаен,
- не короче, чем шифруемое сообщение,
- используется ровно один раз, —

то существует абсолютно стойкий шифр с таким ключом: никакое время и никакая вычислительная мощность злоумышленнику (если соблюдается модель угроз Долева — Яо) не дадут даже теоретической возможности прочесть сообщение без ключа или вычислить ключ.

Мы не будем излагать математическое доказательство, а приведем описание конкретного шифра, который (доказано математически) абсолютно стоек. Этот шифр называется шифром Вернама.

У отправителя и получателя сообщений должны быть два одинаковых случайных текста (шифроблокнота). Опираясь на известные вам принципы двоичного представления — кстати, введенные опять-таки Клодом Шенноном, — мы можем считать, что это длинная последовательность случайных (действительно случайных — то есть не сочиненных кем-то лично, а действительно случайно распределенных) нулей и единиц. Математически это означает, что этот ключ из всего множества ключей для шифрования попадется для работы с вероятностью $P_k(k) = 1/2^N$, где N — количество битов в шифруемом сообщении.

[1] Статья “Теория связи в секретных системах” (1945) с грифом “секретно”, которую рассекретили и опубликовали только лишь в 1949 году.

Мы берем бит сообщения и бит из блокнота и применяем к ним операцию “исключающее или” (XOR), после чего переходим к следующему биту. Получатель, имея такой же блокнот, получит сообщение, возьмет бит сообщения и бит из блокнота и, сделав то же самое (“исключающее или”), восстановит бит исходного сообщения. После шифрования и расшифровки использованная часть блокнота (например, страница) уничтожается, т.е. каждый бит используется РОВНО один раз.

Почему этот шифр абсолютно стоек? Потому что на выходе шифрования мы получим абсолютно случайный набор бит (блокнот заполнен случайным набором бит) — все тексты, если у нас нет ключа, будут равновероятны. Второго текста, зашифрованного тем же ключом, никогда не встретится — так что если ключ не попадает к злоумышленнику, неважно, что он будет делать с шифровкой, — текста он никогда не увидит.

Конечно, на практике все не так просто:

• Получить действительно абсолютно случайный набор битов сложно. Любое вычисление делает этот набор не случайным — т.е. в принципе воспроизводимым. Применяют сложные физические устройства — и очень берегут их от злоумышленников;

• Блокнот должен быть уничтожен после использования. Поэтому, например, использование компьютера нежелательно — информацию в постоянной памяти трудно бесследно уничтожить. А хранить блокнот надо — потому что мы ведь должны его как-то передать до использования;

• Блокнотами нужно как-то обменяться.

Тем не менее, для особо важных сообщений такая схема применяется — вручную. Со строжайше соблюдаемыми мерами безопасности.

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Вопрос, который сразу возникает: а почему этим шифром не пользуются повсеместно? Потому что:

1. Если вы не знаете, сколько у вас будет сообщений и какой длины, то сколько же места вам понадобится, чтобы хранить такие блокноты? А ведь отправитель и получатель могут не иметь возможности встречаться годами, а передача такого блокнота — дело очень опасное.

2. Если каналов связи, подлежащих защите, много, как будет хранить блокноты центр связи?

3. А если потенциальных получателей сообщения больше двух, что делать? Сколько потребуется работы для шифрования одного сообщения?

4. Как понять, что в сообщение вкралась ошибка при передаче?

И наконец, один из ключевых вопросов: а если отправитель и получатель никогда не встречались и друг друга не видели (и не смогут встретиться) — откуда они возьмут пару шифроблокнотов?

Думаю, вы сможете придумать массу ситуаций реального применения, в которых подобные вопросы станут самыми важными. Что делать? Придумать новые шифры, в которых ситуация с ключами все-таки легче. Но будут ли они абсолютно стойкими? Нет.

Чтобы классифицировать и оценивать шифры, вводят понятие *практической стойкости*. Шифр считается практически стойким, когда, для того чтобы его “взломать” — то есть прочесть сообщение без ключа или подобрать ключ, — потребуется больше ресурсов (времени, денег), чем это будет целесообразно. Например: нет смысла тратить сутки работы суперкомпьютера (или, например, вычислительной сети в несколько тысяч узлов), чтобы прочесть письмо с назначением встречи в кафе, — это не окупится. Нет смысла тратить сутки на подбор ключа для отправки сообщений летящей ракете — она уже к этому времени долетит.

Конечно, все будет упираться в то, насколько хорошо шифр будет проанализирован, — то есть не найдется ли способ взломать его быстро?

На основе такого подхода формулируют требования к функции-шифру:

• применение самого шифра (и дешифрации) должно быть быстрым — т.е. иметь максимум линейный класс сложности относительно длины ключа и сообщения;

• обратная функция без ключа (или подбор ключа) должна иметь экспоненциальный класс сложности по ключу (то есть при достаточно длинном ключе времени потребуется ОЧЕНЬ много).

С математической точки зрения это формулируется так:

1. $F(x, K)$ — легко (фактически надо читать — “достаточно быстро с допустимыми ресурсами”) вычисляется для любого x.

2. Существует $G(x, K)$, такая, что $x = G(F(x, K), K)$ — то есть функция расшифровки, которая легко вычисляется для любого $F(x, K)$.

3. По заданному значению $F(x, K)$ практически (см. выше) невозможно восстановить значение x, не имея ключа K. В некоторых случаях это и вообще невозможно.

4. Невозможно найти пару x и y, таких, что $x \neq y$ и $F(x, K) = F(y, K)$.

Называют такие функции ***односторонними функциями***.

Заметим, что “практически невозможно” означает множество проверок и условий. Например, нельзя допускать, чтобы два близких числа x и y (и сообщения, и ключа) давали два близких значения $F(x, K)$ и $F(y, K)$, — а иначе можно очень легко свести все к небольшому перебору вариантов.

Самой функции для работы, конечно, мало. Полный комплекс, годный для использования, будет включать в себя алгоритмы, реализующие и шиф-

рование, и расшифровку сообщений любой длины. Такой комплекс алгоритмов называется ***криптографической системой*** (***криптосистемой***).

Таких функций, а на их основе криптосистем, разработано много. Большинство из них используют разные перестановки битов в зависимости от ключа. Современные шифры такого рода используют несколько раундов (циклов) перестановок с участием ключа и специально рассчитанных матриц перестановки.

Чаще всего используются в нашей стране AES (***A**dvanced **E**ncryption **S**tandard* — шифр "Rijndael", победивший в конкурсе на стандартный шифр в США) и ГОСТ 28147-89 — российский стандарт. Оба этих шифра обеспечивают высокий уровень защиты (при соблюдении некоторых технических условий). Эти шифры используют для работы ключи длиной до 256 бит, и ни одна реально применимая атака не позволяет снизить затраты на взлом шифра ниже, чем до 2^{180} степени операций. Шифрование в них идет блоками — то есть сообщение разбивается на части и каждая из них шифруется как отдельное "число".

Стоит обратить внимание, что современные популярные шифры широко опубликованы, подробно описаны и до начала использования долго и подробно обсуждаются и испытываются всеми желающими. Шифр применяется только тогда, когда в его стойкости и характеристиках все уверены.

Важные требования к современным шифрам — удобная реализация в виде аппаратного решения и обеспечение высокой скорости работы. Это условие связано с тем, что шифры активно применяются для защиты линий связи и реализуются как составная часть массы программ и аппаратных средств. С точки зрения алгоритмов это означает, что алгоритм состоит из простых битовых операций и перестановок, использует внутренние блоки удобной для процессорного слова длины и т.д. Соблюдение этих условий позволяет организовать, например, защищенную линию связи с роботом или небольшим устройством управления — которое в принципе не может использовать мощный процессор (его нельзя запитать от батарейки, в частности).

Поменять стандарт шифрования очень непросто, и никто не будет делать этого до появления серьезных причин — то есть до потери стойкости. Именно по этой причине был сначала модернизирован, а потом и заменен предыдущий стандарт шифрования — DES.

Описанные криптосистемы решают множество задач, с практической точки зрения обоснованно считаются стойкими, но не решают двух основных проблем:

1. Как участникам безопасно обменяться ключами?

2. Как удостовериться, что отправитель или получатель тот, за кого себя выдает?

Особенно это важно тогда, когда у нас в принципе нет возможности обмениваться данными каким-либо способом, кроме как в недоверенной среде — как в модели Долева — Яо. Думаю, вы все уже заметили, что среда Интернет этой модели замечательно соответствует.

Ответа на вопрос "И что нам тогда делать?" не было до 1976 года. В этом году была опубликована статья Уилфреда Диффи и Мартина Хеллмана "Новые направления в криптографии". Статья содержала революционную идею, получившую название ***асимметричной криптографии***.

Все шифры, которые мы описывали раньше, для шифрования и расшифровки используют один и тот же секретный ключ. То есть они симметричны относительно ключа. А если мы будем использовать разные ключи для шифрования и расшифровки, то сможем реализовать схему Диффи — Хеллмана. Описывают такие схемы обмена с помощью двух традиционных персонажей: Алисы и Боба, абонентов обмена.

Итак, Алисе и Бобу известны некоторые два числа g и p, которые не являются секретными и могут быть известны также другим заинтересованным лицам. Для того чтобы создать неизвестный, кроме них, никому секретный ключ, оба абонента генерируют большие случайные числа: Алиса — число a, Боб — число b. Затем Алиса вычисляет значение (1):

$$A = g^a \bmod p \qquad (1)$$

и пересылает его Бобу, а Боб вычисляет (2):

$$B = g^b \bmod p \qquad (2)$$

и передает Алисе. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (то есть у него нет возможности вмешаться в процесс передачи).

На втором этапе Алиса на основе имеющегося у нее a и полученного по сети B вычисляет значение (3):

$$B^a \bmod p = g^{ab} \bmod p \qquad (3)$$

Боб на основе имеющегося у него b и полученного по сети A вычисляет значение (4):

$$A^b \bmod p = g^{ab} \bmod p \qquad (4)$$

Как нетрудно видеть, у Алисы и Боба получилось одно и то же число (5):

$$K = g^{ab} \bmod p$$

Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник встретится с практически неразрешимой (за разумное время) проблемой вычисления (3) или (4) по перехваченным $g^a \bmod p$ и $g^b \bmod p$, если числа p, a, b выбраны достаточно большими.

На практике схема, в которой злоумышленник не может вмешаться в процесс, — это утопия (и наша модель угроз об этом прямо говорит), но сама принципиальная основа появилась: *можно организовать создание общего секретного ключа*.

Первая реализация этой идеи в виде защищенной криптосистемы была выполнена Рональдом

Ривестом, Ади Шамиром и Робертом Адлеманом и названа RSA.

Первая задача, которая решается в рамках этой системы, — генерация ключей.

1. Выбираются два различных случайных простых числа p и g заданного размера (например, 1024 бита каждое).

2. Вычисляется их произведение $n = p \cdot g$, которое называется *модулем*.

3. Вычисляется значение функции Эйлера от числа n:

$$\varphi(n) = (p-1)(g-1).$$

4. Выбирается целое число $e (1 < e < \varphi(n))$ (), взаимно простое со значением функции $\varphi(n)$. Число e называется *открытой экспонентой* (англ. *public exponent*).

5. Вычисляется число d, удовлетворяющее условию:

$$d \cdot e \equiv 1 \bmod \varphi(n)$$

Число d называется *секретной экспонентой*. Обычно оно вычисляется при помощи расширенного алгоритма Евклида.

Пара $\{e, n\}$ — это *открытый ключ RSA*, он публикуется — то есть сообщается всем участникам.

Пара $\{d, n\}$ — *закрытый ключ RSA* и держится в секрете — то есть никогда никому не передается.

Предположим, Боб посылает сообщение Алисе. Шифруются в этой системе целые числа, от 1 до $n-1$, так что и сообщение должно быть таким числом, обозначим его m. Если оно слишком длинное — придется разбить его на части. Шифруют таким образом:

1. Боб берет *открытый ключ* (e, n) Алисы.

2. Боб берет *открытый текст* m.

3. Боб шифрует сообщение с использованием открытого ключа Алисы:

$c = E(m) = m^b \bmod n$.

А расшифровывают так:

1. Алиса принимает зашифрованное сообщение C.

2. Алиса берет свой *закрытый ключ* (d, n).

3. Применяет закрытый ключ для расшифрования сообщения:

$$m = D(c) = c^d \bmod n.$$

Нужно заметить, что вычисления с такими длинными числами — трудоемкая операция (поэтому-то вычисление ключа, или дешифровка без ключа, настолько трудно — оно дольше на порядки). Сокращая время при передаче реальных сообщений, комбинируют асимметричные и симметричные методы: создают симметричные ключи на один сеанс связи (сеансовые ключи), а потом обмениваются ими с помощью асимметричной схемы. Получается и недолгий обмен, и быстрая передача.

Существенная особенность системы RSA — это возможность *цифровой подписи*.

Цифровая подпись по сути своей — аналог подписи обычной, то есть часть документа, которая подтверждает, что подписавшего документ устраивает (очень может быть, что это документ с пометкой “отказать”). При этом:

1. Нельзя изменить документ так, чтобы это было незаметно (например, подменить текст).

2. Подписавший не может сказать “это не я”.

Уже из самого описания видно, что мы можем отправить сообщение строго определенному абоненту — и его прочитает только он, а отправитель никак не сможет отказаться. Но подпись требуется проверять всем, так что используется своя пара ключей, — а расшифровать сообщение может любой желающий. Как и в случае с передачей большого объема данных, на практике никто документ целиком не шифрует — слишком долго и неудобно. Электронно-цифровая подпись рассчитывается так:

1. Создается *дайджест* сообщения — контрольное число, которое можно быстро рассчитать. Функция вычисления этого числа создается так, чтобы изменение даже одного бита приводило к существенным изменениям числа.

2. Дайджест шифруется — парой из открытого и закрытого ключа подписанта.

Как проверить подпись? Несложно: рассчитать дайджест, расшифровать приложенный открытым ключом подписанта — и сравнить. Совпали — документ подписан, не совпали — документ изменялся.

Считается, что открытый ключ известен всем участникам. Тогда:

1. Нельзя сказать “это не я!” — закрытого ключа ни у кого больше нет.

2. Никто, кроме подписанта, не может поменять документ и подменить подпись: нет закрытого ключа зашифровать дайджест.

RSA — очень известная, но не единственная система шифрования с открытым ключом, позволяющая сформировать ЭЦП. В Российской Федерации официально утвержден государственный стандарт формирования и проверки ЭЦП на основе криптосистемы Эль-Гамаля — одного из продолжений-усовершенствований схемы Диффи — Хелмана[2].

В соответствии с действующим законом об ЭЦП[3], В Российской Федерации, если вы хотите иметь юридически значимую систему (например, для участия в электронных торгах по государственному заказу), нельзя использовать средства, не соответствующие этому ГОСТу.

Практическое использование

С практической точки зрения криптографические средства в той или иной форме входят в состав всех современных операционных систем, постоянно используются для передачи данных, удостоверения личности и так далее. Рассмотрим несколько самых популярных их применений.

[2] “ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи”.

[3] Федеральный закон от 06.04.2011 № 63-ФЗ, с изменениями и дополнениями.

Хэш-функции

Одно из самых частых применений криптографии — хранение паролей. Очевидно, что для выполнения проверки соответствия пароли необходимо где-то хранить. База данных паролей (как бы она ни называлась) будет очень привлекательной целью для злоумышленников, а ее кража — серьезнейшей проблемой.

Чтобы сделать проблему менее острой, используют *криптографические хэш-функции*.

Хэш-функцией вообще называют преобразование, которое ставит в соответствие любому массиву входных данных битовую строку заданной длины (пример такой функции — контрольная сумма). Эту битовую строку мы будем называть ***отпечатком***.

Если такая функция:

1. Необратима, то есть по результату нельзя (точнее, трудно — т.е. преобразование вычислительно стойкое) вычислить прообраз.

2. По сообщению нельзя подобрать другое сообщение с такой же хэш-функцией.

3. Вычислительно невозможно найти пару сообщений с одинаковым отпечатком.

Мы будем называть такую функцию ***криптографической хэш-функцией***.

В современных[4] системах пароли никогда не хранятся в открытом виде. Используя одну из возможных хэш-функций, мы можем получить отпечаток пароля и именно его записать в базу. Тогда для проверки доступа мы просто вычислим "отпечаток" пришедшего пароля и сравним с записью в базе, а вот злоумышленник пароль сможет только подобрать — обратного-то преобразования у него нет.

Это, к сожалению, не делает кражу базы паролей безопасной для жертвы — во-первых, это означает, что она доступна для посторонних, а во-вторых, подбирать пароли тогда можно гораздо быстрее. Серьезно усложнить действия злоумышленнику можно с помощью *сертификатов* — но об этом ниже.

Криптографические хэш-функции используются далеко не только для хранения паролей. Их используют, чтобы проверять на отсутствие изменений сообщения электронной почты, файлы, сертификаты и т.д. Стойкость к коллизиям (второе и третье свойства) делают их очень удобным средством контроля.

Вот примеры таких отпечатков для фразы "демонстрация криптографической хэш-функции":

Отпечаток MD5: 45a3fd25d18bb6396bc9150469c5e2d9

Отпечаток SHA-2: b0bac5f11ea4a8d9ec3d3f6010c8c740c5ae26152e43a0d7b96764efa35e786d

Функции подробно описаны в открытых источниках. Если хотите — вполне можете попробовать доказать их нестойкость.

[4] У грамотных разработчиков.

Инфраструктура открытого ключа. Сертификаты

Одно из самых важных современных применений криптографии — создание средств "удостоверения личности" для недоверенных сред (то есть как раз сред, которые описывает модель угроз Долева — Яо). Таким средством являются *сертификаты*.

По сути, сертификат — это специальное сообщение (например, записанное в файл), которое позволяет удостоверить "личность" предъявителя. Конечно, в сертификате предусмотрены средства, которые не позволят любому желающему такое сообщение сформировать; такое средство — это заверение сертификата в специальном *удостоверяющем центре*.

Сертификат содержит:

1. Реквизиты предъявителя: название, адрес электронной почты или сайта, регион и т.д.

2. Открытый ключ предъявителя.

3. Дайджест — например, SHA-1.

4. Ссылку на удостоверяющий центр и электронно-цифровую подпись сертификата этим центром.

5. Срок действия сертификата (обычно рассчитанный на основе длины ключа).

6. Операции, для которых сертификат предназначен.

Сообщения эти формируются по открытому стандарту X.509[5], и это обеспечивает возможность всем участникам широко использовать этот стандарт для своих систем.

Конечно, мы не принимаем сообщения от любого желающего, создавшего свой центр. В составе современных средств есть набор сертификатов доверенных удостоверяющих центров, который можно пополнять.

Полный список всех центров никем не создается — создается иерархическая структура, в которой один центр доверяет другому. В каждом сертификате прописывается цепочка доверия. В список доверия обычно вносят корневые удостоверяющие центры, — это позволяет их сертификаты делать самыми стойкими, менять редко и распространять широко.

Предпринимается масса мер по защите закрытого ключа удостоверяющего центра — ведь именно от этого зависит уровень доверия ко всем выданным им сертификатам.

Таким образом, предъявление сертификата (то есть отправка сертификата и сообщения, зашифрованного связанным закрытым ключом) — способ удостоверить отправителя даже тогда, когда мы его никогда не встречали и не встретим, например, обменявшись удостоверениями через Интернет. О таких применениях мы и поговорим.

HTTPS

Служба WWW и сервисы на ее основе — самое популярное и распространенное средство решения

[5] RFC 5280, *http://tools.ietf.org/html/rfc5280*.

множества задач. Таким способом, как вы, конечно, знаете, реализуется доступ ко множеству информационных систем — в том числе обрабатывающих очень чувствительную информацию: финансовую, личную, служебную и т.д.

Эту информацию необходимо защищать, учитывая несколько обстоятельств:

1. Доступ к таким системам часто выполняется пользователями, которые физически никогда даже рядом с местом обработки не окажутся и заранее неизвестны никому.

2. Данные будут передаваться по каналам связи, которые от вас (и от пользователей) никак не зависят и никакой проверке не поддаются.

3. Базовый протокол службы — HTTP никаких средств защиты информации от перехвата, изменения и т.п. не имеет, но отказаться от него нельзя. Базовые стандарты представления информации — HTML, например, — тоже никакой защиты не имеют, но работать надо именно с ними.

Несмотря на непростые вводные, средства защиты существуют и активно применяются. Значительная их часть основана как раз на средствах криптографии, и одна из базовых методик для этого — защищенный HTTP, протокол HTTPS — secure.

Его базовая идея очень проста: не будем ничего менять ни в протоколе, ни в страницах — добавим специальный фильтр-посредник, который защитит (зашифрует) данные на выходе и расшифрует на входе. Чтобы никто не знал ключа шифрования, создадим и согласуем его для конкретного сеанса. Все данные будем при обмене шифровать быстрым и надежным методом шифрования с закрытым ключом, а сами данные менять не будем, останется тот же протокол HTTP.

Роль такого фильтра (на самом деле давно встроенного в браузеры и web-серверы) играет специальный протокол защиты данных. Применяются широко сейчас два таких протокола: более старый — SSL (***S**ecure **S**ocket **L**ayer*, слой защищенных сокетов) и более новый — TLS (***T**ransport **L**ayer **S**ecurity*, защита транспортного слоя).

Не вдаваясь в подробности, опишем кратко алгоритм, по которому обеспечивается эта защита:

1. Клиент обращается к серверу на порт 443 (вместо 80 — для обычного HTTP) и передает список возможных алгоритмов и способов шифрования, возможный алгоритм сжатия и случайное число.

2. Сервер присылает назад выбранный им из списка (самый защищенный из доступных) метод, способ сжатия и клиентское случайное число.

3. Сервер присылает клиенту свой сертификат.

4. Клиент генерирует временный случайный ключ для шифрования, шифрует его открытым ключом сервера из сертификата и отсылает серверу.

5. Сервер расшифровывает ключ своим закрытым ключом, добавляет к нему свою часть, шифрует своим закрытым ключом. Сервер шифрует сообщение временным ключом и отсылает клиенту.

6. Клиент получает сообщение, расшифровывает и создает общий ключ шифрования. Клиент отсылает серверу специальное сообщение — хэш всех сообщений, которыми они обменивались, и свой адрес.

7. Сервер пытается его расшифровать и проверить. Если проверка прошла удачно — начинается защищенный обмен.

Обратите внимание:

1. Клиент убедился, что сервер тот, за кого себя выдает, — запросил и проверил сертификат. Можно настроить и сервер так, чтобы он запрашивал и проверял сертификат клиента, но тогда нужно быть уверенным, что у клиента он есть.

2. Ключ шифрования не существовал, пока его не создали — и создали его только на один сеанс обмена.

3. Никакие важные данные в открытом виде не пересылались.

Проверим работу этого протокола на практике.

Практическая работа. Попытка перехвата данных по протоколу HTTPS

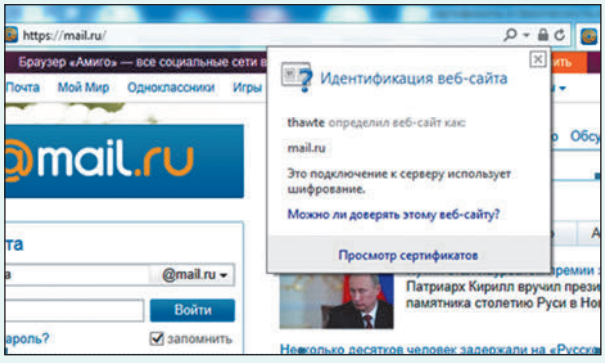
1. Обратимся к сайту *https://mail.ru* (можно и к другому), как и в предыдущей работе, найдем там поля ввода имени и пароля.

2. Запустим программу перехвата[6], настроим фильтр и попробуем обратиться к сайту.

3. Результат мы не показываем, потому что его не будет. Мы использовали для пересылки страницы *mail.ru* защищенный протокол — https, и никакие данные в открытом виде не передавались.

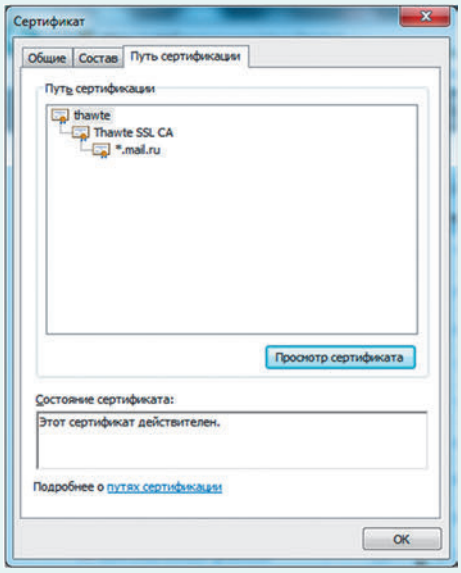
Обратите внимание: при открытии страницы знак “замок” — в адресной строке (см. с. 25).

В разных браузерах он выглядит по-разному, но смысл один — это закрытый канал.



[6] Смотри, например, *Калинин И.А., Самылкина Н.Н.* Защита данных в сетях. “Первое сентября: Информатика”, № 10, 2013.

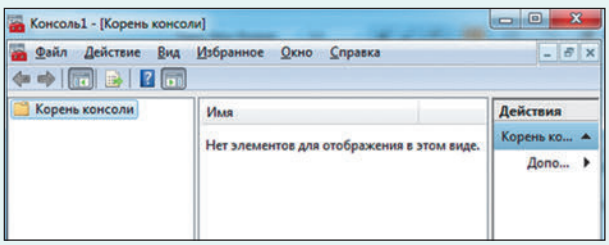
Щелкнем на нем и вызовем свойства сертификата:



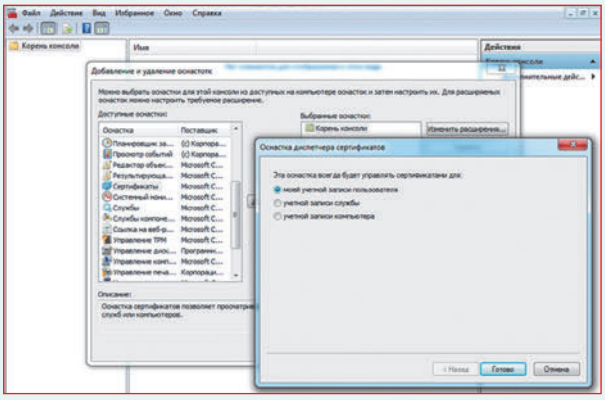
Видно, что сертификат заверен одним из центров Thawte, причем в результате цепочки доверия из трех элементов: корневой центр, промежуточный центр и, наконец, сам сертификат.

Конец этой цепочки обязательно должен быть известен нашему браузеру как доверенный корневой центр. Посмотреть его можно в специальном хранилище:

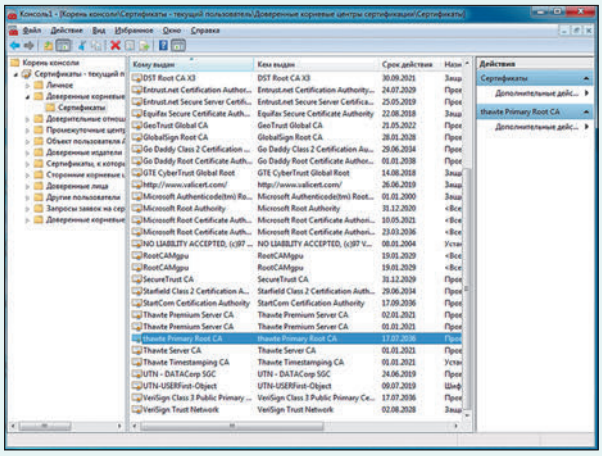
4. — запускаем консоль управления mmc



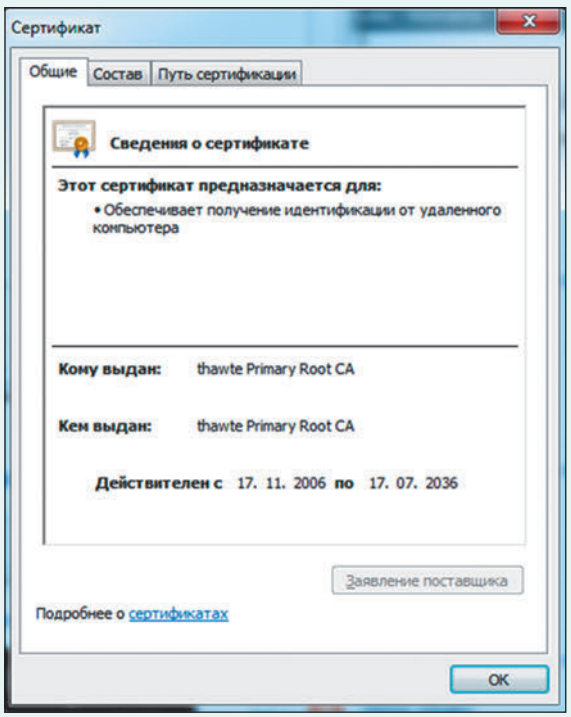
5. — добавляем оснастку (меню “Файл/Добавить оснастку”), указываем, что работать будем с сертификатами своей учетной записи пользователя:



6. — открываем ветку “доверенные корневые сертификаты” и видим там корневой сертификат Thawte.

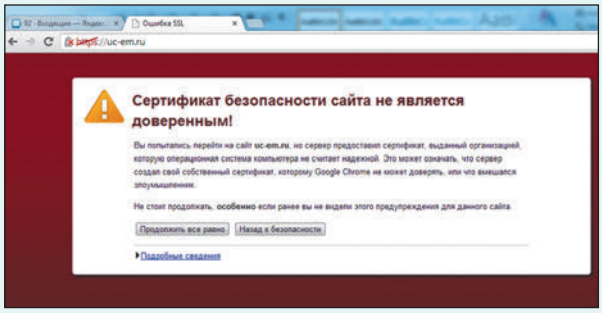


— открываем его и видим его параметры:

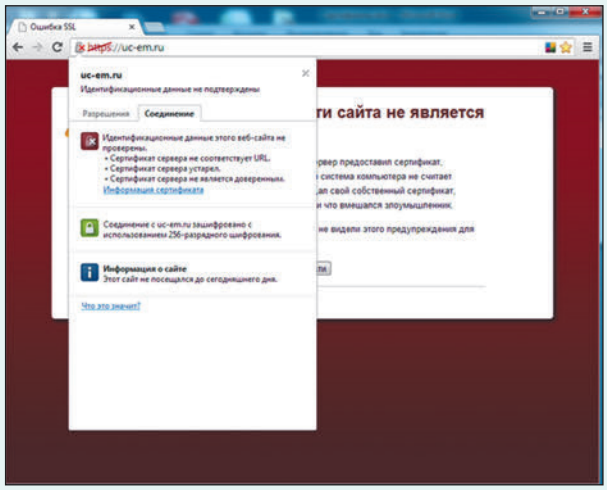


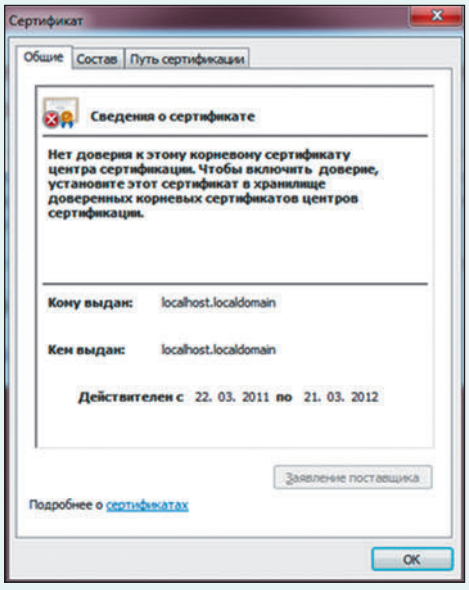
Теперь посмотрим, как будет вести себя система, если сертификат НЕ ПРОЙДЕТ проверку.

7. Обращаемся к сайту *https://uc-em.ru* (московский региональный удостоверяющий центр) и увидим сообщение о проблемах с сертификатом:



Что не так? Щелкнем два раза на замке, вызовем свойства сертификата:





— узел сам себе выдал сертификат;

— узел свое имя определил только для себя — как локальное;

— срок действия сертификата истек — но его не обновили.

Очевидно, разработчики сайта не очень заботились о том, чтобы этот протокол у них работал. Основные задачи организация, очевидно, решает без участия сайта. Возможно (если бы нам требовалось, например, выполнить там оплату), стоит задуматься о том, стоит ли продолжать работу, — это вполне может быть кто угодно.

Откроем обычную версию этого сайта и получим оттуда сертификат этого удостоверяющего центра, — именно этот центр позволяет проверять все сертификаты, например, во время торгов.

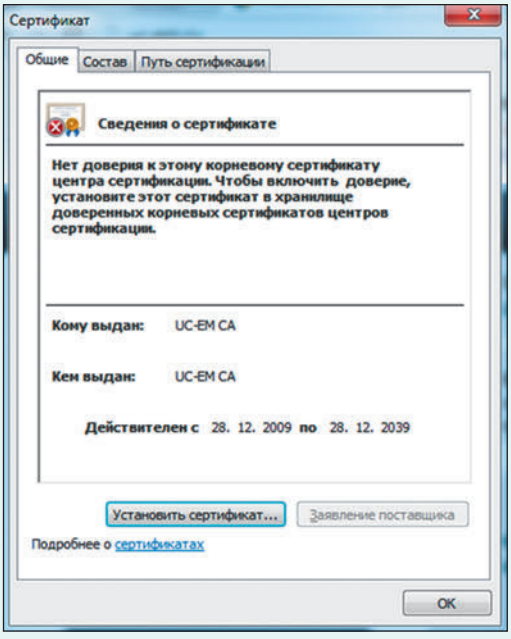
Первое, что нам сообщат при открытии сертификата, — что он поврежден. Дело в том, что его хэш нельзя проверить — алгоритм расчета и шифрования изначально неизвестен. Юридической значимостью в РФ обладают только те ЭЦП, которые сформированы по стандарту ГОСТ Р 34.11/34.10-2001. В стандартную поставку Windows он входить не может.

Придется установить поддержку этого стандарта — то есть программы, которые обеспечат его поддержку. Для личных некоммерческих целей мы можем воспользоваться продуктом ViPNet CSP[7] (*http://www.infotecs.ru/products/catalog.php?SECTION_ID=&ELEMENT_ID=2096*), — для скачивания и последующей активации достаточно указать свои данные на сайте.

В РФ создана целая система корневых удостоверяющих центров, их список можно найти по адресу: *http://www.reestr-pki.ru/tsl.html*.

Посмотрите их сертификаты, а сертификат главного корневого центра мы теперь (имея поддержку ЭЦП по ГОСТ) можем внести в список доверенных.

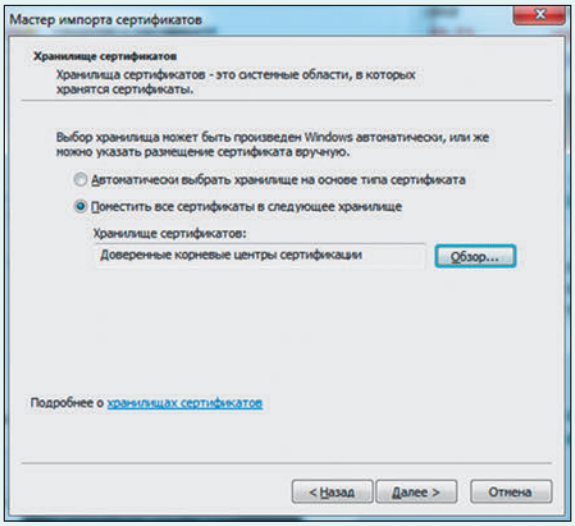
8. Скачиваем сертификат национального удостоверяющего центра с сайта, открываем его:



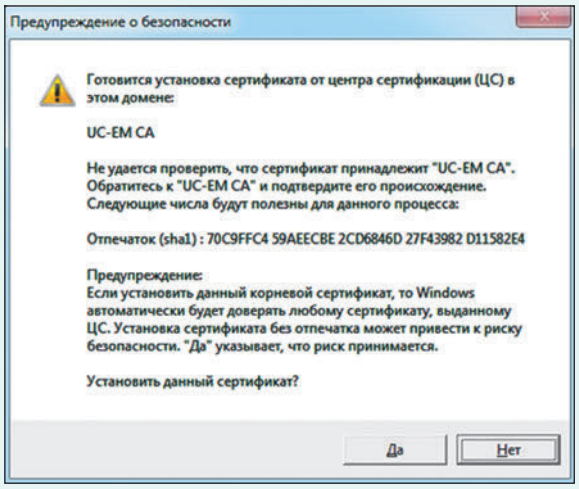
Обратите внимание: сертификат пока считается недоверенным.

[7] Стоит обратить внимание, что заниматься разработкой криптокомпонентов и применять их в нашей стране имеют право только компании, получившие лицензию ФСБ, причем лицензируются и компании, и продукты. Удивительным образом это приводит к тому, что такие продукты бесплатно не предоставляются. Кто-то в любом случае оплачивает лицензию.

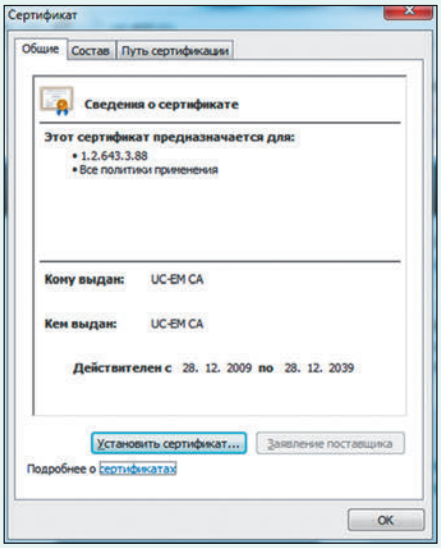
9. Нажимаем кнопку “Установить сертификат” и в окне выбора хранилища выбираем “Доверенные корневые центры”.



10. Подтверждаем установку.



11. При открытии сертификата сообщений об ошибке не будет — несмотря на то, что центр сам себе выдал сертификат.



В каждом регионе РФ есть свой удостоверяющий центр, поэтому вы можете точно так же найти и поставить сертификат “своего” центра.

Хранение закрытого ключа

Обеспечение криптографической надежности самой пары открытого и закрытого ключа — не единственная проблема, которую приходится решать при организации систем электронной подписи и удостоверения с помощью сертификатов. Закрытый ключ, как и сам сертификат, просто сообщение, а это означает, что его вполне можно выкрасть или скопировать.

Для борьбы с такой угрозой используют несколько методов:

1. Использование защищенного хранилища операционной системы. В этом случае сертификат со своим закрытым ключом хранится в специализированном хранилище данных. Хранилище при этом шифруется, и доступ к нему осуществляется только через специализированные средства ОС. Например, такое хранилище в Windows доступно через оснастку “Сертификаты”. Кража такого хранилища бесполезна, но все зависит от стойкости и безошибочности самой ОС.

2. Использование отдельного носителя (флешпамяти или диска), который вставляется только на время использования, а остальное время хранится в сейфе (или в сумочке, например, главного бухгалтера). Авторам представляется бессмысленной идея рассмотрения такого способа хранения.

3. Использование специализированного, защищенного от копирования и несанкционированного чтения носителя. Примером такого носителя может быть носитель Рутокен или его аналог eToken:

Характерные черты таких устройств:

1. Неизвлекаемость закрытых ключей — информация хранится в шифрованном виде (потеря токена при правильной работе не опасна).

2. При каждом доступе к устройству запрашивается специальный код, без ввода которого ключ не будет предоставлен (конечно, если при его инициализации не забыли поменять код по умолчанию).

3. При попытке подобрать код устройство блокируется (причем именно устройство, а не программа у пользователя).

4. Выкрасть ключ с такого устройства программно — чрезвычайно трудно, поскольку ПО для доступа специально защищено.

Сертификаты, средства шифрования и электронно-цифровой подписи — несмотря на свою сложность, одно из тех средств информационных технологий, которые очень активно меняют нашу жизнь, регламентируют обмен, — защищая от нечестных и недобросовестных контрагентов, сокращая затраты времени и средств, обеспечивают конфиденциальный обмен. Это способ организовать “игру по правилам”, что в конечном итоге идет всем на пользу...

Школьники снова играют в CTF, или Компьютерный праздник непослушания

В.В. Ильин,
лицей “Вторая школа”.

В то апрельское воскресенье весь день в лицее “Вторая школа” нарушалось все, что только можно.

Участникам соревнования разрешалось просидеть за мониторами весь день, восемь часов с перерывом на обед. Для решения задач можно было использовать любые электронные средства, в том числе принесенные с собой, спрашивать все что угодно у поисковых систем. Вместо обычного требования “ничего не трогать” участникам предлагалось не только собирать, но и разбирать компьютеры, нажимать кнопки на огромной клавиатуре ногами, использовать интерактивную доску как игровое поле, а главное, производить “деструктивные действия”, взламывать системы защиты, за что, собственно, и начислялись победные очки.

Мы целый день играли в CTF!

Но обо всем по порядку.

```
#include "ctf.h"
```

Компьютерные соревнования CTF (***C**apture **T**he **F**lag*) обычно проводятся среди студентов IT-шников. Командам выдаются компьютеры (сейчас это обычно образ виртуальной машины) с намеренно “дырявыми” сетевыми сервисами. Зада-

Фотографии предоставлены автором

ча команды — оперативно найти и “залатать дыры” в своей системе, а также воспользоваться найденными уязвимостями для проникновения на компьютеры соперников, чтобы получить секретную информацию, специальные коды, которые называют флагами. Захваченные флаги посылаются на сервер жюри, и за них команда получает очки. Естественно, в процессе игры все компьютеры команд должны быть доступны по сети, сервисы — штатно работать, за этим следит специальная проверяющая система.

Студенческие соревнования длятся обычно долго, больше суток. И чтобы участникам было интереснее, и для того, чтобы они могли отдохнуть от напряженной “войны”, им предлагаются дополнительные задачи (tasks, на русском обычно используют сленг “таски”), за которые команда получает дополнительные очки. Задачи при этом бывают самые разные. И “по основному профилю”, например, расшифровать секретное сообщение, и с детективным уклоном — найти спрятанную информацию, и просто на разрядку, скажем, “сфотографироваться, стоя на руках”.

С прошлого года известная уральская хакерская группа (группа студентов — специалистов в области компьютерной безопасности) HackerDom провела соревнование для школьников. Школьникам предлагались только задачи, так называемый “taskbased CTF”. Опыт прошел удачно, и в этом году его решили повторить. В 2014 году к Екатеринбургу подключилась Москва. Играли практически синхронно, в один день, с разницей лишь в несколько часов из-за разных часовых поясов.

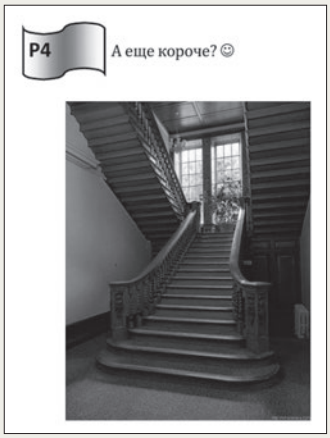
И вышло очень интересно.

При поддержке фирмы 1С во “Второй школе” получился настоящий компьютерный праздник!

В стартапах в области школьных соревнований обычно нет четкого разделения по силам, поэтому все участники получают один набор задач. CTF для школьников только зарождается, и настоящих специалистов в области компьютерной безопасности среди них пока очень немного. Но в то же время очень хотелось вовлечь в процесс даже самых “начинающих новичков”. Мы были готовы к тому, что им будет трудно весь день решать задачи, большую часть из которых они так и не решат. И поэтому заготовили большую дополнительную программу. Чтобы те, кому будет сложно, могли и отдохнуть, и что-то решить, и даже выиграть призы в дополнительном зачете.

Задачи этого соревнования оказались сложнее “ufошных”, о которых мы рассказывали в предыдущей статье, посвященной CTF. Их было пятнадцать, три не поддались никому.

Интернет-площадка соревнований закрыта, но некоторые из них доступны на *http://5kr.mosuzedu.ru/CTFTasks*. Вероятно, вам будет интереснее сначала порешать эти задачи, а потом уже перейти к чтению решений.



Все задачи, кроме одной, были открыты для решения с самого начала соревнования. Расскажем о некоторых из них. Решение остальных можно обсудить в группе ВКонтакте: *http://vk.com/qctfcontest*.

```
void mainTasks()
{
```

Click&Win

Приступая к этому заданию, помните, что кучу монотонных действий куда лучше поручить компьютеру.

По ссылке “Начать!” открывается страничка с лаконичным HTML:

```
<a href=b9daf873d514777f49cbbff9c2c6b8b6
   class="design">Нажми сюда</a>
```

При клике — переход на другую страницу с очень похожим содержимым…

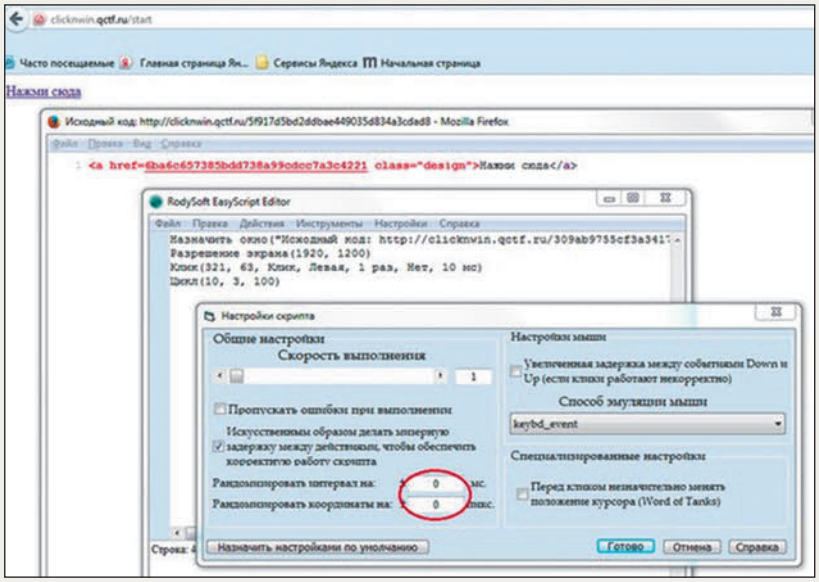
Похоже, это самая простая задача соревнования, с нее и начнем. Из пояснений условия сразу понятно, что надо делать. Нажимать-нажимать и еще раз нажимать на ссылку. Очень долго нажимать! (в результате оказывается 1000 раз).

Нужен автокликер.

На просторах Интернета найден очень понятный, бесплатный, простой и русскоязычный RodySoft EasyScript (*http://rodysoft.ru/easyscript/*).

“Натравливаем” его на страничку — флаг наш.

Только надо отключить “рэндомность” положения кликов (см. картинку на с. 40) в настройках скрипта. И “натравливать” на исходник, чтобы не тратилось время на рендеринг (отрисовку) страницы в браузере.

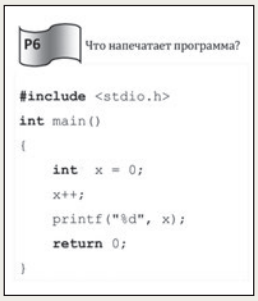


И не стоит сразу ставить слишком большое значение в цикле — мышка захвачена, на время работы скрипта управление компьютером будет сильно затруднено.

Но можно и небольшим скриптом на Питоне, например:

```
===============
import httplib, urllib
body = urllib.urlencode({})
headers = {"User-Agent": "Mozilla/5.0 (X11;
CrOS armv7l 5500.100.6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/34.0.1847.120
Safari/537.36"}
hash = "start"
i = 0
while 1:
  i += 1
 conn = httplib.HTTPConnection("clicknwin.
  qctf.ru")conn.request("GET", '/'+hash,
  body, headers)
  response = conn.getresponse()
  data = response.read()
  conn.close()
  if len(data)==79:
    hash = data[8:41]
  else:
      print data
      print i
  break
=================
```

Вообще без браузера и кликов!



Ping me

Получите файл на сервере

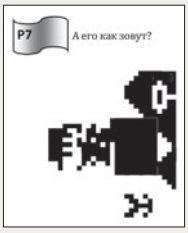
По ссылке страничка с формой, предлагающая ввести IP.

Результат работы скрипта представлен на рисунке справа.

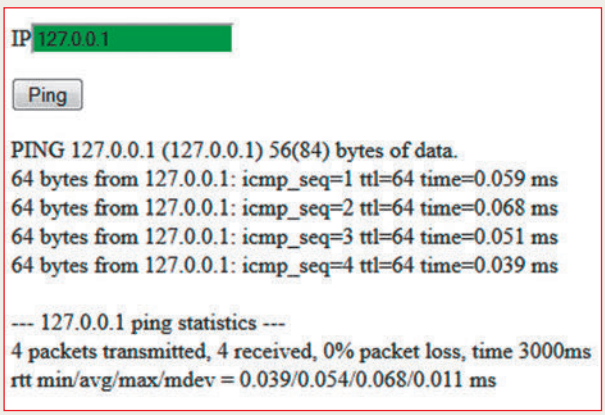
В принципе классическая задача на web-взлом. Похоже, что сервер просто выполняет команду `ping` с переданной строкой — адресом. А что, если ему передать не только IP-адрес, а чуть больше, скажем, 127.0.0.1; ls?

Но поле ввода окрашивается красным — фильтрация. Ничего, кроме IP-адреса, ввести нельзя. Или… можно?! Анализируем HTML:

```
...
  <script>
    var  ip  =/^((25[0-5]|2[0-4]\d|[01]?\
d\d?)\.){3}(25[0-5]|2[0-4]\d|[01]?\d\
d?)$/i;
  function check(id, reg){
      var value = $("#" + id).val();
      if (reg.test(value) ){
     $("#"+id).css("background","green");
            return true;
      }else{
       $("#"+id).css("background","red");
            return false;
      }
  }
  </script>
</head>
<body>
    <form action="index.php"
       method="post" id="f1">
...
```



Фильтруется при помощи JavaScript, на стороне клиента, то есть у нас.



Сохраняем страничку у себя (с расширением html) и правим атрибут `action` у формы на полный адрес скрипта

```
<form action="http://ping.qctf.ru/
              index.php" method="post"
              id="f1">
```

И переписываем функцию проверки:

```
function check(id, reg){
      return true;
}
```

чтобы она всегда возвращала true — успех.

Теперь посылка формы с содержимым 127.0.0.1; ls становится возможной и приносит нам не только вывод команды `ping`, но и список файлов в текущей директории (`ls` — это linux-аналог команды `dir`).

Вторым запросом пытаемся получить содержимое файла с флагом:

```
127.0.0.1; cat flagname –
```

где `flagname` — имя файла с флагом, полученное предыдущим запросом.

Не получается. На уровне HTML стоит фильтрация на длину передаваемых данных:

```
<p><label for="ip">IP</label>
    <input maxlength="15" type="te…
```

Снова правим сохраненный HTML, убирая фильтрацию:

```
<input maxlength="15" type="te…
```

Теперь запрос выдает флаг.

Можно дать и запрос `;cat *.txt`, при этом не нужно править HTML — длины в 15 символов хватит. Команда `ping` отработает некорректно, но мы увидим содержимое команды `cat`.

А можно на втором этапе и просто вбить полученное имя файла в адресную строку браузера — флаг наш!

Hungary games

При переходе по ссылке — сообщение:

Этот сайт доступен только из определенной страны.

Hungary — Венгрия!

Срочно ~~летим в Будапешт~~ ищем и устанавливаем венгерский прокси, его можно найти в поисковике, а установить — в любом браузере. Теперь все запросы от нашего браузера будут идти на венгерский сервер, а уже оттуда — в Екатеринбург. В Екатеринбурге теперь нас примут за венгров. (На самом деле не все так просто. Часто прокси-сервера пересылают запрашиваемому ресурсу информацию о клиенте, и найти злоумышленника становится возможным, и нужно искать правильный абсолютно анонимный прокси, но в данном случае подходит практически любой.)

Заметим, что москвичи имели технические проблемы при решении этой задачи из-за фильтрации МГТС. Мы были готовы к этому и просили участников по возможности “захватить” 3G-интернет с собой. Еще одно непослушание — использование нефильтрованного Интернета на территории школы.

Quiz

Через два часа после начала соревнований был открыт доступ к задаче Quiz. Участникам предлагалось войти в чат и принять участие в викторине. Каждые две минуты организаторы предлагали вопрос.

Многие вопросы были достаточно стандартными на поиск информации:

1-й вопрос: Расшифровать DQG BRX, EUXWXV?

2-й вопрос: Когда был зарегистрирован домен urfu.ru? (дд.мм.гггг)

3-й вопрос: Раскодируйте: — — — --- --- --- — — -

4-й вопрос: Назовите операционную систему, автором которой является Эндрю Таненбаум.

5-й вопрос: Назовите CTF-команду, которая находилась на пятом месте в рейтинге сайта *ctftime.org* 18 июня 2012 года.

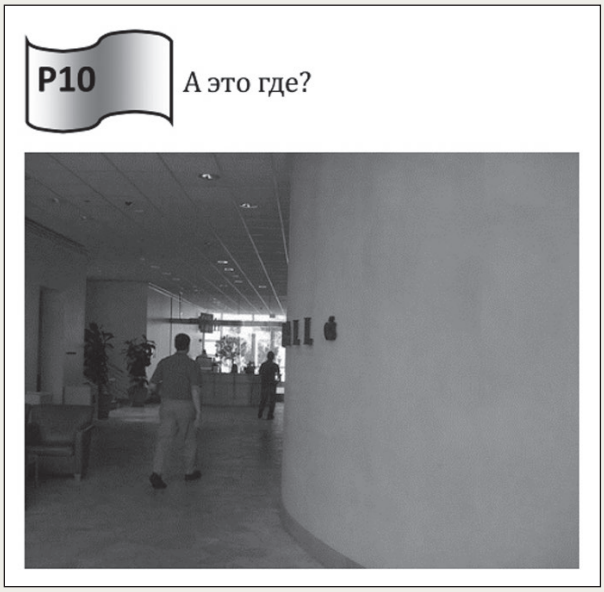
6-й вопрос: Расшифровать 0KLRj9C20LXQu9C+INCyINGD0YfQtdC90LjQuCAtINC70LXQs9C60L4g0LIg0LHQvtGOLg==

7-й вопрос: Кто изображен на фото *http://shkolazhizni.ru/img/content/i73/73586.jpg*?

8-й вопрос: Кому принадлежит цитата “Проще просить прощение, чем получать разрешение”?

9-й вопрос: Напишите корни уравнения $x^3 + 4.85x^2 — 2.618125x + 0.14625 = 0$

10-й вопрос: Назовите автора QDOS.



11-й вопрос: Сколько кустов содержится в реестре Microsoft Windows 7?

12-й вопрос: В каком году в СССР появился первый узел FidoNet?

13-й вопрос: Кто изображен на фото *http://www.osp.ru/FileStorage/NEWS/Computerworld_Online/2014-01/0113_14/13154584/Computerworld_Online_0114-1_(8911).jpg*?

14-й вопрос: Какая технология приходит на смену BIOS?

15-й вопрос: Кто изображен на фото *http://inf-sub.moy.su/_pu/0/97870265.jpg*?

(Ответы и предполагаемые способы решения — в конце статьи.)

Интересно то, что этот конкурс проходил одновременно в двух городах. После первого верного ответа команда получала флаг и на остальные вопросы ей отвечать было бесполезно. Таким образом, флаги по этой задаче получили многие команды, но у сильных команд появилось преимущество во времени. Они тратили на получение флага по этой задаче всего несколько минут, а другим приходилось "ждать своей очереди" полчаса.

Шифр индюка

Нами было перехвачено зашифрованное сообщение: "TFQCZvq4nl]7soD++" (без кавычек). Также удалось найти программу, при помощи которой был зашифрован текст. Найдите, что было зашифровано. Ответ начинается с "QCTF_".

По ссылке — программа-шифратор.

Запускаем и начинаем исследовать, давать различные комбинации. Выясняем, что шифр символа не зависит от положения в строке и контекста, а только от кода самого символа. Задача оказывается весьма простой. Шифруем весь алфавит (и строчные и прописные) и, смотря на результат в условии, восстанавливаем исходное сообщение.

Следующая задача немного посложнее.

API

*Нами было перехвачено зашифрованное сообщение: "H[0%hLR7*jX5^V]" (без кавычек). Также удалось найти программу, при помощи которой был зашифрован этот текст. Найдите, что было зашифровано. Ответ начинается с "qctf".*

Программа предлагает ввести текст и ключ шифрования и выдает зашифрованный текст.

Так же, как и в предыдущей задаче, пробуем, зашифровывая различные комбинации символов.

Анализ проб показывает, что каждый символ кодируется в один символ шифра, но значение теперь зависит и от положения шифруемого символа в строке.

Так же замечаем, что код шифра зависит только от самого кода, его положения и, естественно, ключа. При этом в качестве ключа можно вводить только числа — строки в качестве ключа не изменяют текст.

Как и в предыдущей задаче, мы знаем результат шифрования и флага и начало ключа.

Пытаемся перебрать ключи брутфорсом (просто перебирая числа последовательно) так, чтобы строка `qctf` была зашифрована как `H[0%`. Это утомительно делать вручную. Можно написать скрипт. Напишем его, например, на языке командной строки Windows, bat-языке:

Файл *testkeys.bat*

```
@echo off
echo Keys: > keys.txt
for /l %%x in (1, 1, 200) do (
     echo. >> keys.txt
     echo %%x >> keys.txt
     (echo qctf & echo %%x) | api.exe >>
      keys.txt
)
```

Скрипт сформирует файл *keys.txt*, в котором поиском строки `H[0%` можно найти ключ:

```
….
HQ7%
93
Hello, this is simple crypter
Write data for crypt: Would you like
  to write key? If not, write zero.
H[0%
94
…
```

Язык командной строки Windows достаточно сложен и архаичен. Конечно, по-настоящему компьютерно грамотному IT-шнику нужно знать скриптовый язык командной строки, но нам кажется перспективнее в этом русле изучать язык linux-шеллов. Тем не менее часто бывает нужно быстро автоматизировать работу именно в Windows, поэтому основы bat-языка знать тоже желательно.

Дадим несколько пояснений к тексту скрипта.

Первая строка отключает вывод самих команд — мы будем видеть только результат их выполнения.

Вторая строка нужна для очистки файла *keys.txt* (на случай, если придется его запускать несколько раз, а это естественно при отладке).

Строка цикла выполняется с ключом `/I`, соответственно, строка `(1, 1, 200)` будет развернута в последовательность от 1 до 200 с шагом в единицу (второй параметр в скобках). Соответственно, переменная цикла `%%x` будет принимать значения от 1 до 200 (мы надеемся, что ключ не очень велик).

`echo. >> keys.txt` выводит пустую строку, фактически делает перевод строки, при этом вывод дозаписывается в файл. Таким же образом в следующей строке в файл дозаписывается значение очередной пробы-ключа.

Особый интерес представляет собой последняя строка.

Работа с программой-шифратором интерактивна, но это можно обойти. Команды (`echo qctf` и `echo %%x`) формируют необходимые строки, их вывод передается программе, эмулируя действия пользователя.

Аналогичным скриптом можно теперь символ за символом получить весь флаг.

Файл *getflag.bat*

```
@echo off
echo . > findflag.txt
for %%x in (a b c d e f g h i j k l m n
            o p q r s t u v w x y z 1 2 3
           4 5 6 7 8 9 0 — _ = + ! @) do (
     echo %%x >> 1.txt
    (echo qctf%%x & echo 93) | api.exe
      >> findflag.txt
)
```

Запускаем его несколько раз и при этом расширяем строку `qctf` найденными символами.

В принципе можно написать “волшебный” скрипт, цикл в цикле, который сделает всю работу автоматически. Но отладить его незнакомому с bat-языком школьнику будет очень сложно.

Секрет индейцев

Маленький и большой индейцы закодировали одно и то же сообщение.

У маленького получилось 4499718262561273479485870492282866223510002895156309708657088822517869864987541598166264974214341699905465376440599377,

у большого — 3201925414048142601347782532152964120499943162901520072664731450351145132860925718802635829604783264244683363414324082.

Какое сообщение они закодировали?

Что это? Может быть, огромные числа? Обычно такие используются для шифрования, но здесь вроде зашифрован сам текст. Попробуем перевести в шестнадцатеричную систему счисления.

На Питоне встроенная “длинная арифметика”!

Команда

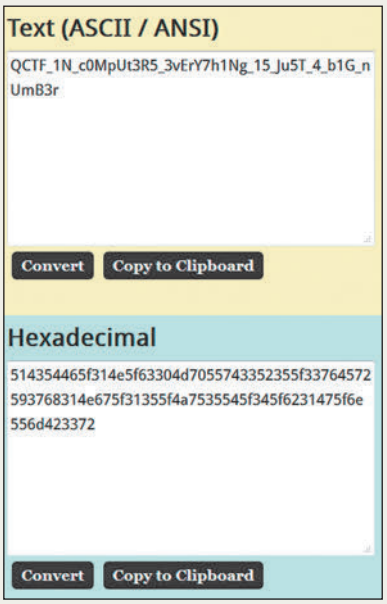
```
hex(3201925414048142601347782
532152964120499943162901520072
664731450351145132860925718802
6358296047832642446833634143240
82)
```

мгновенно выдает результат:

```
'0x514354465f314e5f63304d705
5743352355f33764572593768314
e675f31355f4a7535545f345f623
1475f6e556d423372'
```

Что это? Может быть, просто последовательность кодов?

Воспользуемся, например, сервисом *http://ascii-2hex.com*.



И получим флаг: на этот раз “говорящий” на специальном хакерском языке.

А почему индейцев два? Предлагаем вам самостоятельно попробовать второго!

И на этот раз не обошлось без задачи-шутки.

Only Chrome

Нам не жалко флагов — вы только попросите.

По ссылке, если использовать браузер Chrome, откроется страничка, в которую подгружается JavaScript, приведем лишь его фрагмент:

```
Var_0x8645=["\x68\x65\x6C\x70","\x54\
x68\x65\x72\x65\x20\x61\x72\x65\x20\x66\
x65\x77\x20\x66\x75\x6E\x63\x74\x69\x6F\
x6E\x73\x20\x77x63\x74\x65\x64\x53\x63\
x72\x69\x70\x74\x48\x6F\x73\x74","\x68\
x61\x73\x4F\x77\x6E\x50\x72\x6F\x70\x65\
x72\x74\x79","\x69\x6E\x64\x65\x78\x4F\
x66","\x73\x6C\x69\x63\x65","\x3B","\
x50\x65\x72\x6D\x69\x73\x73\x69\x6F\
x6E\x20\x64\x65\x6E\x69\x65\x64\x21","\
x61\x70\x70\x6C\x79"];funcs=[_0x8645[0
]];function help(){console[_0x8645[2]]
(_0x8645[1]);}  ;function  get_me_flag()
{var  _0xb880x3  =[_0x8645[4][_0x8645[3]]
(0)];  _0xb880x3[_0xb880x3[_0x8645[5]]]=
_0x8645[6][_0x8645[3]](0);_0xb880x3[_0xb8
80x3[_0x8645[5]]]=_0x8645[7][_0x8645[3]]
(0);_0xb880x3[_0xb880x3[_0x8645[5]]]=_
0x8645[8][_0x8645[3]](0);_0xb880x3[_0xb8
80x3[_0x8645[5]]]=_0x8645[9][_0x8645[3]]
(0);var _0xb880x4 =_0xb880x3[_0x8645[10]]
(_0xb880x3); for(_0xb880x8=0;_0xb880x8<_0
xb880x4[_0x8645[5]];
```

Обфускация! Как известно, исходный код HTML и JavaScript невозможно сделать недоступным, но можно обфусцировать, превратить его в очень трудночитаемый. При этом обратная операция даже очень хорошими средствами хорошего результата обычно не дает: максимум, что мы обычно получаем при этом, — нормальное форматирование с отступами и, может быть, читаемые названия функций:

```
funcs = ['help'];
function help() {
   console['log']('There are
few functions which allowed
to you. But which is right?');
};
function get_me_flag() {
  var _0xb880x3 = ['Q'
     ['charCodeAt'](0)];
     _0xb880x3[_0xb880x3
```

```
['length']] =
'C' ['charCodeAt'](0);
    _0xb880x3[_0xb880x3
['length']] = 'T'['charCodeAt'](0);
…
```

Но, может быть, этого хватит? Пробуем изменить код странички, вызываем функцию `get_me_flag()` — безрезультатно. Смотрим на код еще раз. Оказывается, что функций с подобным названием в коде много…

От жюри приходит подсказка: “нельзя просто взять и вызвать функцию”. Замечаем, что из всех “кандидатов” только одна функция, которая принимает параметр:

```
function give_me_flag(_0xb880x43) {
…
```

Но какой именно параметр передать?

Оказывается, решает задачу вызов этой функции с параметром-строкой `'please'`!

Как до этого догадаться? В хакерском деле интуиция очень важна! Например, победители соревнования, команда 179#1, догадались!

Кстати, флаг для задачи-шутки “Ремингтонист”, которой мы закончили статью в прошлом номере, слово “COOL”.

Предлагалось расшифровать фразу “`trdcv ytfcvbh iuhbnmk 0okm`”. Как решить эту задачу при помощи выключенного компьютера?

Элементарно! “Ремингтонист” — наборщик на печатной машинке, перед нами клавиатура!

```
} // Разбор задач завершен
```

Но, как мы уже сказали, понятно, что задачи даже такого уровня (сложные в статье не разобраны, с ними можно познакомиться на гитхабе разработчиков, ссылка в конце статьи) вызвали трудности у многих школьников.

И чтобы праздник состоялся и для них, мы придумали много оффлайновых развлечений.

```
void offline()
{
```

Наибольшим успехом пользовался, пожалуй, админский уголок, в котором были подготовлены задачи по “железу”.

RealAdmin Corner

Желающим предлагалось пять тасков:

Task A1. How long

Даны большой моток витой пары и линейка 15 см, требуется измерить длину мотка с погрешностью 3 см.

***Решение*:** находим на проводе маркировки вида 0000FT, 0002FT, считаем разность между концами мотка, переводим в футы, прибавляем расстояние от маркировок до концов, померенное линейкой.

Рекомендации проводящим: очень желательно настойчиво намекать посмотреть на маркировку провода. Затем заставить учесть еще и расстояние от маркировки до концов. Не принимать решения, когда школьники наматывают кабель на руку или на парты. Мы давали моток длины 13 м, для устрашения можно дать что-нибудь гораздо масштабнее (но тогда оно в размотанном состоянии будет валяться на большой площади). Реально школьники при правильном способе дают ответ с разбросом в 15 см.

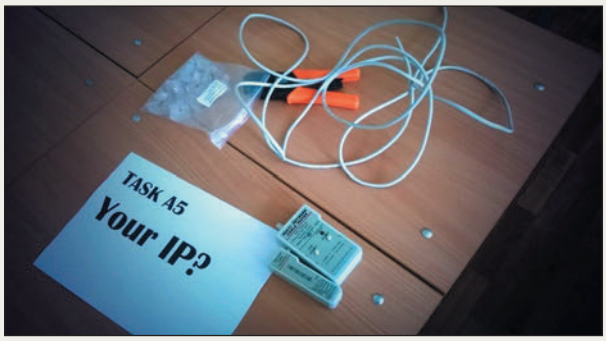
Task A2. Your IP?

Даны кусок витой пары, обжиматор, контакты и устройство для прозвона витой пары. Требуется обжать оба конца в одинаковой последовательности, чтобы устройство для прозвона показало четыре зеленые лампочки.

Сначала предполагалась более сложная схема с получением IP по обжатому проводу, но потом решили сделать попроще.

Стоит объяснить, почему снимать изоляцию с восьми маленьких проводков не надо. У нас был плохой обжиматор: если пользоваться им грубо, то он перерубает изоляцию некоторых маленьких

проводков. Может быть, стоит давать для этого конкурса хороший обжиматор?



Task A3. Secret No?

Даны системный блок, монитор, ВГА-кабель, два кабеля питания, клавиатура, отвертка. Найти флаг.

Решение: развинчиваем системный блок с "глухой" стороны, находим там флешку, грузимся с нее в ДОС, находим файл *secret.bas* и запускаем при помощи QBasic или просто анализируем код. Можно усложнить: убрать автоматическую загрузку с флешки в БИОСе, поставить пароль на БИОС.

Многие школьники догадываются раскрутить системный блок с неправильной стороны, но кому-то не везет. Стоит давать возможность гуглить основы работы с ДОСом.

Task A4. Password?

Даны системный блок со снятой крышкой и с кучей комплектующих внутри, монитор, клавиатура.

Решение: собираем компьютер, грузимся, находим файл с флагом.

Эта задача оказалась самой сложной. Команды очень долго ее решают, а при неправильной сборке могут что-нибудь сжечь. Надо либо пристально наблюдать за тем, что и куда втыкают школьники, либо заранее воткнуть правильно все питание.

Task A5. Blindly*

(*вслепую)

Даны системный блок и клавиатура, найти флаг.

Решение: на клавиатуре циклически переставлены клавиши QCTF и 2014.

Первые два конкурса мы разместили в одном классе, последние три — в другом. На каждый класс нужно по проверяющему.

В течение пяти часов, за которые конкурсы были доступны, отбоя от желающих не было (на площадке было 30 школьников 7–11-х классов).

Сразу после обеда по школе были развешаны задачи-картинки. Здесь, в журнале мы решили поступить аналогично, "развесили" картинки по всей статье. Ответы в конце статьи.

Начался

PictureQuest

Командам, как и в основной игре, нужно было назвать жюри флаг, ответ на краткий вопрос по каждой картинке.

Предупреждаем сразу о небольшом проколе в организации — современные технологии позволяют загуглить картинку мобильным устройством буквально одним нажатием. Так что, если соберетесь сделать что-то подобное, немного изменяйте изображения перед печатью.

Честно говоря, VideoGames corner, который мы проводили под конец, удался не очень хорошо. Планировались парные игры на планшетах, но в результате было решено использовать интерактивную доску. Для того чтобы было интересно не только тем, кто играет, но и зрителям. Единственной игрой, которая предлагалась участникам в тот раз, была Angry Birds.

Заметим, что, несмотря на усталость, никто из младших участников не пытался просто поиграть на своих ноутбуках, и все участвовали в общих играх.

Ну а в конце, когда уже все младшие устали, началось настоящее безумие.

IT-Sport Corner

Длилось оно, правда, совсем недолго, но взбодрило всех перед финалом.

Участникам была предложена огромная Твистер-клавиатура.

Играли креативно, ведущие придумывали правила на ходу. Команды были самые разнообразные: не только "Эскейп левой ногой", но и "Контрол Альт Шифт" или даже "Вызвать Меню Пуск".

В общем, подошли к процессу творчески. Особенно эффектно вышло, когда ведущий скомандовал: "Альт носом!". К сожалению, фотографий той замечательной композиции не сохранилось.

```
}
void HallOfFlame()
{
```

Но, конечно, были ребята, которые потратили на решение все восемь часов — минута в минута.

Скажем даже, секунда в секунду. **Команда московской команды 179#1 из 179-й школы в составе Дмитрий Галов, Артем Хачатрян, Роман Лозко и Георгий Тимощенко** решила свою последнюю задачу на последних секундах **и стала абсолютным победителем соревнования в общем зачете двух городов**.

Второе место за ребятами из СУНЦа Екатеринбурга**: Викентием Котвицким, Алексеем Стафеевым, Станиславом Федяниным** и **Антоном Матвеевым**.

```
}
void about()
{
```

В заключение

Очень хочется поблагодарить всех тех, без кого праздник бы не состоялся.

Прежде всего это, конечно, команда HackerDom, которая, собственно, и подготовила основное соревнование;

учителей, помогавших в проведении мероприятия; выпускников “Второй школы” Гожева Алексея и Янбулатова Рамиля; а также спонсора, фирму 1С, предоставившую подарки всем участникам соревнования, и родительский комитет “Второй школы”, благодаря которым никто из участников не ходил голодным, а победители получили мобильные аккумуляторы, оригинальные часы, портативные колонки и другие ценные призы.

```
}
//TO DO
```

И, конечно, у нас, организаторов, есть желание повторить!

Из главного: таски стоит подготовить чуть проще, а оффлайновые конкурсы — разнообразнее. И мы надеемся сделать это в скором времени. Уверены, что нас в следующий раз будет значительно больше.

Следите за новостями на *http://5kr.mosuzedu.ru/CTFtasks* и группы ВКонтакте *https://vk.com/schoolctf*. Всем зарегистрированным пользователям сайта и подписчикам группы будет разослано приглашение, когда мы окончательно определимся со временем следующего компьютерного “Праздника непослушания”, ближайшего школьного CTF.

```
//COMMENTS
```

Ответы и предполагаемые способы решения на вопросы таска Quiz:

1. AND YOU, BRUTUS? Способ решения: шифр Цезаря — расшифровка.
2. 16.12.2008. Способ решения: сервис *who is*.
3. SOS. Способ решения: азбука Морзе.
4. Minix. Способ решения: Google.
5. Leet More. Способ решения: сервис *web.archive.org*.
6. Тяжело в учении — легко в бою. Способ решения: декодер base64.
7. Ада Лавлейс. Способ решения: поиск по картинкам.
8. Грейс Хоппер. Способ решения: Google.
9. Solutions: $x = -5.34495$, $x = 0.0634053$, $x = 0.431545$. Способ решения: сервис *wolframalpha.com*.
10. Tim Paterson. Способ решения: Google.
11. Способ решения: можно в Google-картинках быстро найти скриншот по запросу “Реестра Windows 7”.
12. 1990. Способ решения: Google.
13. Steve Wozniak. Способ решения: поиск по картинкам.
14. UEFI. Способ решения: Google.
15. Charles Antony Richard Hoare. Способ решения: поиск по картинкам.

Ответы и подсказки PictureQuest:

1. Красивый логотип был у Apple когда-то! Особенно классно смотрится на этом фоне логотип спонсора 1С (внизу) — Как решать, неизвестно, просто надо осознать, что не знаешь и хочешь узнать ответ.

2. Если в двоичной системе записать числа, то можно из 0 и 1 простроить два треугольника (они там все плотные, “без дырок”).

3. HACK! С кодированием — количество символов — номер буквы в английском алфавите. Сложная задача.

4. Внизу нечетко читается адрес — гуглим: вторая ссылка — офис IBM.

5. Pacman.

6. Для сишников… * (еле видна) — указатель ++ на 4 байта в современных системах.

7. Mario, повернутый на 90° против часовой стрелки.

8. Whitespace с подсветкой.

9. У Тьюринга машина Тьюринга.

10. Яблоко на стене.

С другими материалами QCTF School 2014 можно познакомиться в архиве по адресу *https://github.com/HackerDom/qctf-school-2014*, а связаться с разработчиками задач — *http://vk.com/qctfcontest*.

ДЛЯ ЭРУДИТОВ

Как считали чукчи?

М.А. Цайгер,
кандидат технических наук

В своей повести “Чукотка” Т.З. Сёмушкин описал любопытный случай счета оленей чукчами. Этот рассказ привел в своей книге “Мир чисел” известный специалист по арифметике И.Я. Депман. Вот что писал Сёмушкин:

“Проезжая однажды по кочевым стойбищам, я заметил на склоне горы небольшое стадо оленей. Сидя на нарте, я легко пересчитал его. Оленей было сто двадцать восемь. Когда я спросил хозяина, владельца стада, сколько у него оленей, он не мог мне ответить.

— Мы не считаем. Но если хоть один олень пропадет из стада, глаза мои узнают сразу.

— А можешь ты посчитать?

— Если тебе нужно, посчитаю. Только долго буду считать. Поезжай пока в яранг, а потом я принесу счет.

В яранге мы успели попить чаю, закусить, переговорить с хозяином обо всем, а часа через два пришел наш “подсчетчик”. Он назвал цифру — сто двадцать восемь. Старик крайне удивился такому множеству оленей.

— Наверно, ты ошибся. Так много оленей никогда у нас не было.

Старик решил проверить. Он знал каждого оленя и поэтому немедленно, не выходя из яранги, занялся подсчетом. Для этого он разулся и через три часа сообщил, что подсчет произведен правильно. Натуральный “арифмометр”[1], состоящий из пальцев рук и ног, был для подсчета такой цифры недостаточен. Старику-оленеводу потребовались для этой цели все члены семьи, состоящей из пяти человек, кроме того, он пригласил двух человек из соседней яранги”.

Почему старик, считая оленей (заметьте, не выходя из яранги — чукотской юрты), разулся и пригласил в помощь членов своей семьи и даже двух человек из соседней яранги?

Думаю, что причина в том, что считающий знал всех своих оленей, что называется, “в лицо”, они были для него не отвлеченные предметы, а знакомые “личности” со своими индивидуальными признаками.

Попробуйте посчитать своих двоюродных братьев и сестер. Мои дедушки и бабушки родились и жили еще в царской России, когда в семье бывало помногу детей. Поэтому я насчитал у себя восемнадцать двоюродных братьев и сестер. Нынешнему поколению школьников считать будет легче, так как у их дедушек и бабушек детей было меньше. Но я убежден, что многие из вас, ни разу не считавшие этих своих родственников, пойдут по такому пути: у тети Маши такие-то дети, у дяди Коли — такие-то и так далее. Только после установления всех своих двоюродных братьев и сестер вы начнете считать их общее количество. Такова логика обращения с персональными существами.

Полагаю, что чукча, считавший оленей по памяти, мысленно разводил их по загонам, которые идентифицировал с сидящими людьми. Каждый человек имеет двадцать пальцев (рук и ног), и

[1] Арифмометр — механическое счетное устройство. В данном случае автор имеет в виду “устройство для подсчета”. — **Прим. ред.**

чукча разводил своих оленей по этим конкретным загонам, считая, что каждый загон может вместить двадцать животных. Такие-то и такие-то олени шли в загон такого-то члена семьи, и тогда считающий знал, сколько оленей осталось в общем стаде, которых осталось поместить в загоны. Так он увидел, что своих членов семьи ему не хватило, и послал за соседями. Этот процесс оказался длительным — для подсчета потребовалось три часа.

Думаю, что такой способ счета у чукчей определялся особенностями считаемых предметов. Если бы чукче предложили посчитать спички в коробке, которые для него не отличались друг от друга, ему не понадобилось бы разуваться и приглашать других членов семьи: он просто выложил бы спички в кучки (в зависимости от того, как было принято считать — десятками или двадцатками) и затем посчитал бы количество кучек.

Возможно, те из вас, кто хорошо знаком с обычаями чукчей или других народов с подобным образом жизни, со мной в чем-то не согласятся и выскажут свое мнение, основанное на личных наблюдениях. Это было бы интересно. Возможно, вам известны также аналогичные приемы подсчета общего числа предметов и т.п. Пишите в редакцию журнала.

ИСТОРИЯ ИНФОРМАТИКИ

Перфоленты

Прохорова Александра,
ученица гимназии № 1530 г. Москвы

Одним из видов носителей информации в электронно-вычислительных машинах (ЭВМ) первых поколений была так называемая “перфолента” (сокращение от “перфорированная лента”). Как правило, она представляла собой бумажную ленту с отверстиями (*рис.* 1). Использовались также перфоленты на основе кино- и фотопленки.

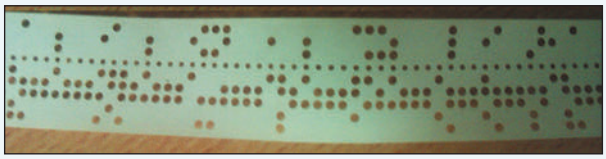


Рис. 1

Информация на ленте представлялась в двоичной системе счисления. Отверстие соответствовало 1, его отсутствие — 0.

Использовались 5-рядные (*рис.* 2), 6-рядные, 7-рядные и 8-рядные (см. *рис.* 1) перфоленты.

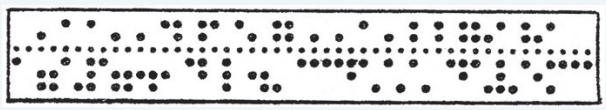


Рис. 2

В одной колонке 5-рядной ленты могли быть закодированы числа от 0 (00000) до 31 (11111), 6-рядной — от 0 (000000) до 63 (111111), 7-рядной — от 0 (0000000) до 127 (1111111), 8-рядной — от 0 (00000000) до 255 (11111111).

Например, на одной из первых отечественных ЭВМ — модели М1 — использовалась 5-рядная перфолента. Основная информация — программа, которая представляла коды команд и адреса ячеек памяти с данными, набивалась в 1-м, 2-м и 4-м рядах ленты. Таким образом, на одной позиции ленты могло быть указано 3-разрядное двоичное число (или десятичное от 0 до 7). Однако благодаря тому, что код команды и адрес числа получались из нескольких таких значений, то есть использовалась двоично-восьмеричная система счисления, значения кода и адресов могли быть и гораздо большими. Кроме того, на ленте кодировалась команда о записи кода очередной команды и адресов в память ЭВМ (об этом “говорило” отсутствие отверстия в 5-м ряду) и команда “стоп” в конце перфоленты (она задавалась отверстием в 1-м ряду).

Оператор набирал текст программы на “обычной” клавиатуре. Эта информация преобразовывалась в двоичную систему счисления и с помощью специального устройства — перфоратора (см. *рис.* 3[2]) наносилась на ленту.

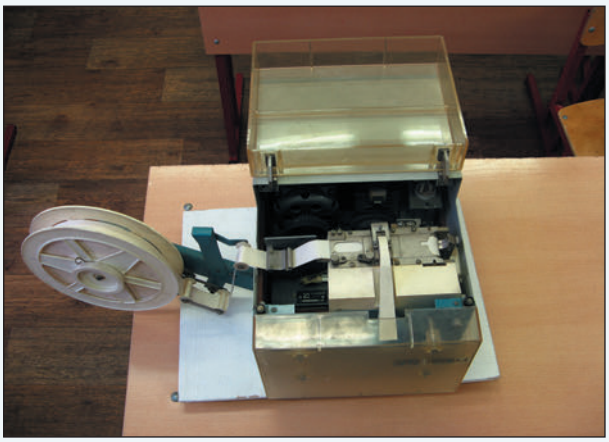


Рис. 3

Когда набивалась информация, то на перфоленте могли быть ошибки. Если оказывалось лишнее отверстие, то его можно было заклеить. Но если нужно было еще одно отверстие, то использовалось специальное приспособление. Лента клалась внутрь этого приспособления и в нужном месте вручную пробивалась стерженьком (см. *рис.* 4).

[2] Представлены фотографии экспонатов Музея истории вычислительной техники гимназии № 1530 г. Москвы (*www.museum.ru/m2744*).

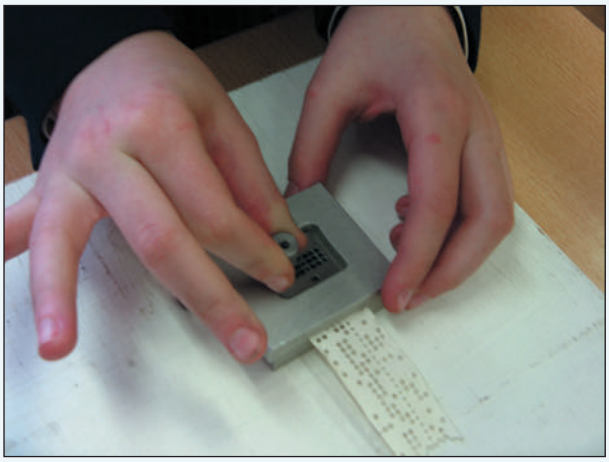


Рис. 4

Но был еще и другой “исправитель”. На нем можно было не только пробивать отверстия, но и разрезать и склеивать ленту (см. *рис*. 5). То есть он представлял собой целый комплекс для работы с перфолентами.

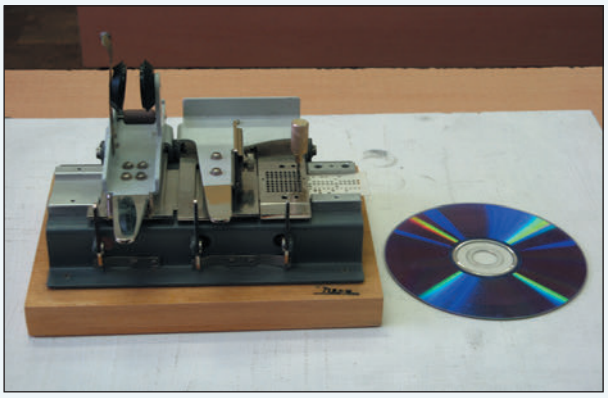


Рис. 5

Ввод информации с перфоленты в ЭВМ осуществлялся с помощью фотосчитывателя (*рис*. 6).

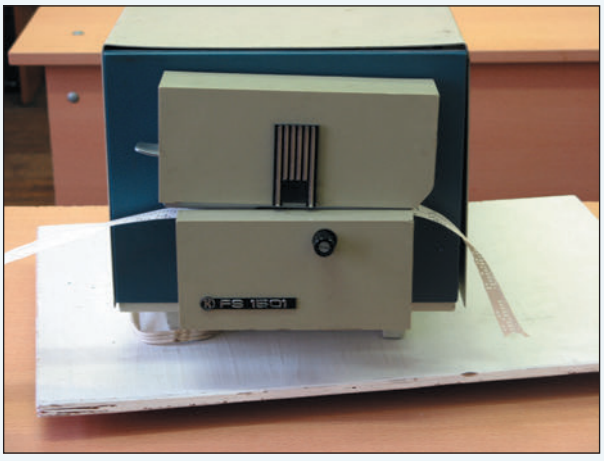


Рис. 6

В нем размещалась перфолента. При ее перемещении для каждой колонки с помощью восьми лучей света “считывалась” и вводилась в ЭВМ информация на перфоленте (если свет прошел сквозь ленту, значит, в этом ряду имеется отверстие, то есть 1, если не прошел — то отверстия нет, то есть 0).

В заключение — немного истории (хотя, конечно, перфоленты в компьютере — это тоже история).

В 1725 году француз Базиль Бошо впервые предложил новый способ управления ткацким станком с помощью перфорированной бумажной ленты.

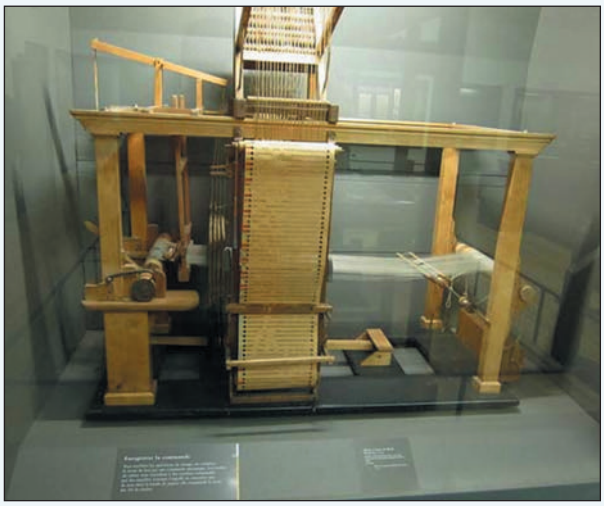


Рис. 7. Ткацкий станок Б.Бошо на выставке в Музее искусств и ремесел в Париже

В 1846 году перфорированные ленты были использованы изобретателем химического телеграфа Александром Бэйном. Расположенные на ней соответствующим образом дырочки-пробивки обозначали точки и тире азбуки Морзе. На таком аппарате можно было передавать до 252 символов за 52 секунды (около 300 слов в минуту), т.е. в пять-шесть раз больше, чем при ручной работе посредством телеграфного ключа. Телеграф назывался химическим потому, что бумажная лента смачивалась смесью аммиачной селитры и ферроцианида калия.

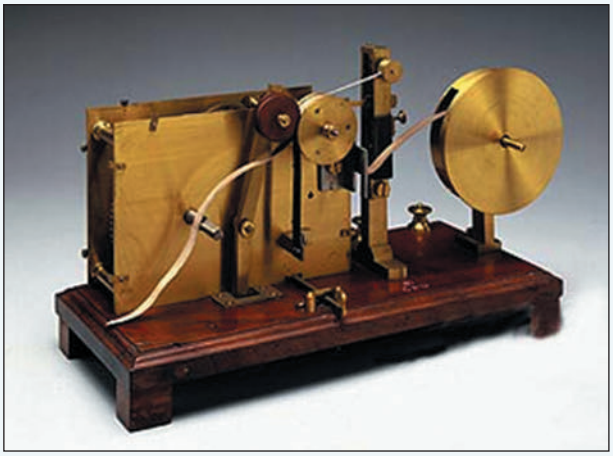


Рис. 8. Химический телеграф Бэйна, 1850 г.

Англичанин сэр Чарльз Уитстон продолжил работу по совершенствованию телеграфа, которую начал Александр Бэйн. В результате сэр Чарльз разработал первое промышленное автоматическое устройство телеграфа, в котором применялись бумажные ленты в качестве средства для подготовки, хранения и передачи данных. На бумажной ленте

сэра Чарльза использовались два ряда отверстий для представления кода Морзе.

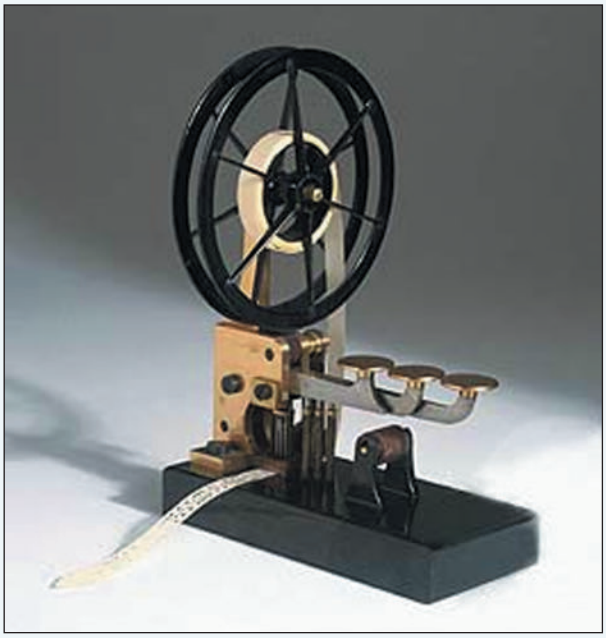


Рис. 9. Устройство Ч.Уитстона с перфолентой, 1869 г.

Русский изобретатель Петр Павлович Княгининский создал первый в мире “автомат-наборщик” — литеронаборную машину с программным управлением от перфоленты. Его машина “читала” текст, представлявший собой бумажную ленту с комбинациями отверстий, соответствующими каждой букве и знаку (так называемые “депеши”), и автоматически (с помощью электромагнитного механизма) производила набор металлических литер. В 1870 году машина Княгининского была доставлена в Петербург и демонстрировалась на Мануфактурной выставке. Позже со своей машиной Княгининский был в Москве. К его машине проявляли интерес, но никто не поддержал изобретателя в его стремлении найти ей практическое применение. Умер П.П. Княгининский в нищете, а машина его бесследно исчезла. Между тем идея Княгининского была весьма прогрессивна. Принцип автоматизации набора с помощью перфорированной ленты был использован в 90-х годах XIX века при создании строкоотливной наборной машины.

ЗАДАЧНИК

Ответы, решения, разъяснения к заданиям, опубликованным в разделе “В мир информатики” ранее

Статья “Четыре задачи”

Напомним, что были предложены четыре задачи, связанные, как показывает их решение, с темой “Измерение информации (содержательный подход)”. Учитывая, что эта тема широко используется в ЕГЭ по информатике и является важной в “нашем” ☺ предмете, обсудим решение задач подробно.

Задача 1

Ира и Катя играют в игру — “угадайку”, используя карточки:

Ира загадывает одну из карточек, а Катя должна отгадать ее. На вопрос можно отвечать только ответами “Да” или “Нет”. Помогите Кате составить вопросы так, чтобы быстрее отгадать загаданную карточку.

Решение

Следует задавать вопросы так, чтобы ответы “Да” и “Нет” делили количество возможных вариантов на две части. Для того чтобы быстрее отгадать единственную карточку, необходимо ставить вопросы так, чтобы эти части были равны друг другу.

Минимальное количество вопросов равно трем.

Вопросы о цвете, форме и содержании карточки можно задать в любом порядке.

Задача 2

В записи пароля использовали цифры, русские и латинские буквы. Пароль состоит из пяти символов. Известны все символы, кроме одного. Точно известно, что среди перечисленных ниже букв имеется необходимая.

Q А Ё I Ъ В Ю К Ц Я

Отвечая на какой из перечисленных вопросов “Да” или “Нет”, сообщат один бит информации?

1) эта буква существует в русском алфавите?
2) эта буква есть среди перечисленных?
3) эта буква существует в латинском алфавите?
4) это цифра?
5) эта буква существует и в русском, и в латинском алфавитах?

Вопрос	Ответ “Да”	Ответ “Нет”
Эта буква существует в русском алфавите?	8 элементов: А Ё Ъ В Ю К Ц Я	2 элемента: Q I
Эта буква есть среди перечисленных?	10 элементов (точно известно, что среди перечисленных букв имеется необходимая)	0 элементов
Эта буква существует в латинском алфавите?	5 элементов: Q A I B K	5 элементов: Ё Ъ Ю Ц Я
Это цифра?	Если учитывать, что здесь есть цифры в шестнадцатеричной системе счисления, то получится два элемента: A B	8 элементов: Q Ё I Ъ Ю Ц Я К
Эта буква существует и в русском, и в латинском алфавитах	3 элемента: A B K	7 элементов: Q Ё I Ъ Ю Ц Я

Решение

Один бит — это количество информации, уменьшающее неопределенность в два раза. Все вопросы поставлены в такой форме, что ответы на них могут быть “Да” или “Нет”. Ответами “Да” или “Нет” множество вариантов разделится на два подмножества. Количество элементов в полученных подмножествах может быть как одинаковым, так и различным. Рассмотрим, на какие подмножества разбивают исходное множество предложенные вопросы (см. таблицу выше).

Исходное множество делится на два равных подмножества (5 и 5 элементов) при ответе на третий вопрос. Неопределенность уменьшилась в два раза, то есть сообщили 1 бит информации. (Интересно, что при ответе на второй вопрос неопределенность не изменяется.)

Ответ: “Эта буква существует в латинском алфавите?”.

Задача 3

Каким минимальным количеством бит можно закодировать состояние игры в “крестики-нолики” на поле 5 × 5 клеток, кодируя каждую клетку в отдельности?

Решение

Всего на поле 5 × 5 = 25 клеток. Каждая клетка во время игры может быть в трех состояниях: пустая, крестик, нолик. Для кодирования одного из трех вариантов необходимо два бита, а для всех клеток — 25 × 2 = 50 бит.

Ответ: 50 бит.

Задача 4

Игральные карты выкладывают по принципу — после черной карты обязательно должна следовать красная, а после красной — черная. Цепочку составляют только из карт-картинок (валет, дама, король, туз) всех мастей. Сколько возможно вариантов цепочек из:

1) трех карт?

2) четырех карт?

Решение

Все карты различны. Так как цвет первой карты может быть любым, то на первое место можно положить одну из 16 карт (четыре карты-картинки по четыре масти каждой). После первого хода будет определен цвет второй карты. Следовательно, на второе место можно положить одну из 8 карт. На третье место можно положить одну из 7 оставшихся карт, совпадающих по цвету с первой картой. На четвертое место — одну из 7 карт, совпадающих по цвету со второй. Количество вариантов цепочки из трех карт: 16 × 8 × 7 = 896; из четырех карт: 16 × 8 × 7 × 7 = 6272.

Ответ: 896 вариантов цепочек из трех карт и 6272 варианта цепочек из четырех карт.

Ответы представили:

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

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Богданова Алина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гололобов Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Землянская Надежда, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Новиков Сергей и Хромченкова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Хорькова Анна, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

Числовые ребусы в троичной системе

Ответы

Ребус 1

$$\begin{array}{r} 1 \\ +\ 1 \\ \hline 2 \end{array}$$

Ребус 2

```
  1
+ 2
-----
1 0
```

Ребус 3

```
  2
+ 2
-----
1 1
```

Правильные ответы представили:

— Абрамкин Дмитрий и Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Антипов Анатолий и Кучук Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Богданова Алина, Зарипова Ксения и Юматова Ксения, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гололобов Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Мищенко Маргарита, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Новиков Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Федосеева Анастасия, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Шигаев Никита, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**

Задание “Шесть вопросов” (рубрика “Поиск информации”)

Ответы

1. Хвойное дерево, плакучую разновидность которого вывели в Никитском ботаническом саду, — кедр.

2. Домашнее прозвище Владимира Маяковского — “Щенок”.

3. Герой романа американского писателя Джона Апдайка “Кролик разбогател” торгует автомобилями марки “Тойота”.

4. Русский вариант испанского имени Диего — Яков.

5. Советский разведчик Рудольф Абель добывал секреты американской атомной бомбы под псевдонимом “Марк”.

6. Сухопутный гигант, близкими родственниками которого являются ламантин и дюгонь, — слон.

Ответы представили:

— Акулова Мария и Зернова Алина, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Берибесова Анастасия, Дедова Анастасия и Лесных Любовь, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Бородюк Анна и Василенко Татьяна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гаязова Фатима, Михайлов Иван и Хорькова Анна, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Герасимова Наталья и Костина Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

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

— Книгина Софья и Подольская Арина, г. Рязань, школа № 44, учитель **Марцинкевич Е.Е.**;

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

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Лёвина Татьяна и Цыплаков Евгений, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Трептау Татьяна, Вадьковская средняя школа, Брянская обл., Погарский р-н, учитель **Цыганкова И.Ю.**;

— Якушов Александр, г. Орел, лицей № 4 им. Героя Советского Союза Г.Б. Злотина, учитель **Чапкевич И.М.**

Задание «Да будет “свет”!» (февральский выпуск, рубрика “Для эрудитов”)

Напомним, что был предложен ряд фактов, для каждого из которых следовало определить, правдивый он или нет.

Ответы приведены в виде таблицы.

Номер вопроса	Ответ
1	Да, верный
2	Нет, только в 1899 году, П.Н. Лебедевым
3	Нет — Фотиния или Фотина
4	Нет — Америго Веспуччи
5	Да, верный
6	Да, верный
7	Да, верный
8	Да, верный

Ответы прислали:

— Акулова Мария, Бойко Юрий, Зернова Алина и Цикавий Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Бородюк Анна и Василенко Татьяна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Васина Светлана и Хомутова Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Евграфова Ксения, Калинкина Мария и Усова Евгения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Есипова Мария, Круглякова Мария, Левченко Елена, Храмцов Филипп и Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Иванова Виолетта и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Кротов Олег и Телегин Дмитрий, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Лёвина Татьяна, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Немова Софья и Подольская Арина, г. Рязань, школа № 44, учитель **Марцинкевич Е.Е.**;

— Трептау Татьяна, Вадьковская средняя школа, Брянская обл., Погарский р-н, учитель **Цыганкова И.Ю.**;

— Якушов Александр, г. Орел, лицей № 4 им. Героя Советского Союза Г.Б. Злотина, учитель **Чапкевич И.М.**

Задача "Шарики в коробочках"

Напомним условие: "Имеется пять коробочек: белая, черная, красная, синяя и зеленая. Также есть по два шарика для каждого из указанных цветов. В каждой коробочке лежит по два шарика, причем цвета коробочки и шариков могут и не совпадать. Известно, что: 1) ни один шарик не лежит в коробочке того же цвета, что и он сам; 2) в красной коробочке нет синих шариков; 3) в коробочке нейтрального цвета (то есть белого или черного) лежит один красный и один зеленый шарик; 4) в черной коробочке лежат шарики холодных тонов (зеленый и синий цвета); 5) в одной из коробочек лежат один белый и один синий шарик; 6) в синей коробочке находится один черный шарик. Какого цвета шарики лежат в каждой коробочке?".

Решение

Составим таблицу, в которой буквами обозначим цвета шариков и поставим знак "–" в клетках, соответствующих "невозможным" вариантам размещения шариков, и знак "+" в клетках, соответствующих известному размещению (в скобках указан номер известного факта из условия):

Коробочка	Шарики									
	Б	Б	Ч	Ч	К	К	С	С	З	З
Белая	– (1)	– (1)	–		+ (3, 4)		–		–	+ (3, 4)
Черная	–	–	– (1)	– (1)	–	–	+ (4)	–	+ (4)	–
Красная			–		–	–	– (2)	– (2)		–
Синяя			+ (6)		–		– (1)	– (1)		–
Зеленая			–		–		–		– (1)	– (1)

Из фактов 3 и 4 следует также, что один красный и один зеленый шарики находятся в белой коробочке. Эта информация, а также следующие из нее "невозможные" варианты размещения в следующей таблице выделены красным цветом:

Продолжая анализ, можно получить такой ответ: в зеленой коробочке находятся белый и синий шарики, в синей — черный и красный, в красной — белый и черный, в белой — красный и зеленый, в черной — зеленый и синий шарики.

Правильные ответы прислали:

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Барановская Татьяна и Жукова Ирина, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Васина Светлана, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Дегтярь Анатолий и Новиченко Владимир, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Иванова Виолетта и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Кротов Олег и Телегин Дмитрий, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Лежнева Александра, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Мусатов Максим и Тонт Анастасия, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Бойко Юрий и Цикавий Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Якушов Александр, г. Орел, лицей № 4 им. Героя Советского Союза Г.Б. Злотина, учитель **Чапкевич И.М.**

Кроссворд (мартовский выпуск)

Ответы

По горизонтали:

1. Трассировка. 6. Три. 8. Фон. 9. Корпус. 10. Азия. 13. Паскаль. 14. Сеть. 16. Счетчик.

17. Остаток. 19. Принтер. 24. Число. 25. Дек. 26. Евклид. 28. Воскресенье. 29. Ось.

По вертикали:

1. Тип. 2. Сноска. 3. Рок. 4. Вирус. 5. Акустика. 6. Триггер. 7. Запись. 11. Палитра. 12. Кластер. 15. Шаг. 18. Индекс. 20. Цикл. 21. Код. 22. Киев. 23. Плюс. 26. Ель. 27. Икс.

Ответы прислали:

— Булатова Анна и Калугин Сергей, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**;

— Дегтярь Анатолий, Елисеева Анастасия и Новиченко Владимир, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Землянская Надежда, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Игнатьева Екатерина и Мезина Ольга, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Крысанов Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Машонина Ирина, г. Фрязино Московской обл., школа № 4, учитель **Сенюта Е.И.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Никишин Дмитрий, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Михайлов Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Нуретдинова Лилия, средняя школа села Сулеево им. Р.Г. Галеева, Республика Татарстан, Альметьевский р-н, учитель **Валиева Д.И.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Цикавий Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Головоломка “Получить число 1957”

Напомним, что требовалось в равенстве

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0 = 45$$

расставить знаки сложения и умножения между цифрами 1, 2 …, 9, не изменяя порядка их последовательности, чтобы результат арифметических действий равнялся бы числу 1957 (между некоторыми цифрами можно не ставить знак, объединяя их в “многозначное” число).

Решение

$$1 \cdot 234 \cdot 5 + 67 + 8 \cdot 90.$$

Ответы представили:

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Аксененко Ирина и Чумаков Илья, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Коробов Сергей, Марков Алексей и Яснов Федор, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Михайлов Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Павлова Ирина, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Полуэктова Анна, Рафикова Гульнара и Шептунова Елена, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

В ряде ответов использован знак вычитания (или знак “–” перед числом).

Задание “Сколько лет человеку?”

Напомним условие: “Предложите кому-нибудь умножить число его лет на 2 и к произведению прибавить 4, затем полученную сумму умножить на 5, к этому произведению прибавить 12 и полученную сумму умножить на 10. После объявления результата предложенных арифметических действий вы можете объявить число лет. Как это сделать?”.

Ответ

Надо от объявленного числа отнять 320, затем полученный результат разделить на 100.

Пример. Пусть число лет некоторого человека равно 37. Умножая это число на 2 и прибавляя 4, получим 78. Умножая это число на 5, имеем 390; прибавив 12, находим 402; умножая на 10, получаем 4020. Вычитая из этого числа 320, имеем 3700. Если это число разделить на 100, то получим число 37.

Обоснование. Пусть число лет некоторого человека равно m. После выполнения всех предложенных арифметических действий получится число

$$((2m + 4) \times 5 + 12) \times 10.$$

Это число можно записать в виде $100m + 320$. Если от него отнять 320, то получится число $100m$. Разделив последнее число на 100, будем знать искомое число лет m.

Правильные ответы представили:

— Абросимов Игорь, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Аксененко Ирина, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Леженников Тарас и Мищенко Маргарита, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Михайлов Иван и Назарова Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Павлова Ирина, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Числовой ребус “Чему равны три ТРЕТИ?”

Решение

Запишем ребус в “столбик”:

```
   Т Р Е Т Ь
+  Т Р Е Т Ь
   Т Р Е Т Ь
  -----------
   Ц Е Л О Е
```

Прежде всего можем сказать, что **Т** = 1 или **Т** = 2 (трем **Т** не может быть равно, так как это возможно только при **Е** = 0, а из последнего разряда следует, что **Е** > 0). Значит, из разряда десятков в разряд сотен “в уме” ничего не переходит. Учитывая это, исследуем возможные значения **Ь** и соответствующие значения **Е** (по последнему справа разряду), **Л** (по разряду сотен) и **Р** (по разряду тысяч):

Ь	**Е**	**Л**	**“В уме” из разряда сотен**	**Р**	**Допустим ли вариант?**
1	3	9	0	1	Нет (**Ь** = **Р**)
2	6	8	1	5	Да
3	9	7	2	9	Нет (**Р** = **Е**)
4	2	6	0	4	Нет (**Ь** = **Р**)
5	5				Нет (**Ь** = **Е**)
6	8	4	2	2	Да
7	1	3	0	7	Нет (**Ь** = **Р**)
8	4	2	1	1	Нет (**Т** = 1 или **Т** = 2)
9	7	1	2	5	Да

Проверка трех допустимых вариантов показывает, что последний вариант не подходит, а два оставшихся — дают следующие решения ребуса:

15 612 + 15 612 + 15 612 = 46 836

32 836 + 32 836 + 32 836 = 98 508

Ответы прислали:

— Абросимов Игорь, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Антипов Анатолий, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Живило Андрей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Иванова Ксения и Мухина Светлана, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.** (Тарас нашел оба решения, разработав для этого компьютерную программу);

— Марун Виталий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

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

— Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Якушов Александр, г. Орел, лицей № 4 им. Героя Советского Союза Г.Б. Злотина, учитель **Чапкевич И.М.**

Еще раз обратим внимание на то, что, публикуя числовые ребусы, редакция имеет в виду их решение методом рассуждений (из всех приславших ответы обоснование привели Анатолий Антипов и Александр Якушов).

Статья “Песочные часы”

Напомним, что предлагалось решить две задачи, связанные с песочными часами.

Задача 1

Есть двое песочных часов: на 4 минуты и на 9 минут. Как определить промежуток времени в 7 минут?

Задача 2

Есть двое песочных часов: на 4 минуты и на 11 минут. Как определить промежуток времени в 2 минуты?

Время, затрачиваемое на переворачивание часов, не учитывать.

Ответы приведем в виде арифметического выражения, в котором числа — первые множители в произведениях в левой части соответствуют типу часов, вторые множители — количеству их установок (переворачиваний), символ “–” говорит о том, что учитывается разность времени:

Задача 1: $4 \times 4 - 9 \times 1 = 7$.

Задача 2: $2 \times 11 - 4 \times 5 = 2$.

Правильные ответы прислали:

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Аксененко Сергей, Иванов Иван, Перова Валентина и Яковлева Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Воронова Анжелика и Хомутов Андрей, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Диков Андрей и Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Довгань Алексей и Мокеева Елена, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Семенов Андрей и Турков Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Симоненко Елизавета и Тегипко Анна, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Филиппов Михаил, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Якушов Александр, г. Орел, лицей № 4 им. Героя Советского Союза Г.Б. Злотина, учитель **Чапкевич И.М.**

Задача "Переправа на остров"

Напомним условие: "Трое мальчиков пошли на рыбалку, взяв с собой лодку, выдерживающую нагрузку до 100 кг. Как перебраться мальчикам с берега реки на остров, если их массы равны 40, 50 и 70 кг?".

Решение

Обозначим для краткости: Б — берег, О — остров, 40 — мальчик массой 40 кг, 50 — мальчик массой 50 кг, 70 — мальчик массой 70 кг.

Тогда схема переправы будет такой:

1. Б → О: плывут 40 и 50 (на Б остался 70).
2. О → Б: возвращается 40 (на О остался 50).
3. Б → О: плывет 70 (на Б остался 40).
4. О → Б: возвращается 50 (на О остался 70).
5. Б → О: плывут 40 и 50.

Правильный ответ представили:

— Абдуллина Лина, Ельницкая Анна, Колесникова Людмила, Мусатов Тимофей, Сатыбалдина Ксения и Тонт Анастасия, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Авдеева Анастасия, Семенов Андрей и Турков Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Андрющенко Александр, Остроухова Валерия, Пономаренко Анастасия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Бирюкова Светлана, Григоренко Василий, Григоренко Дмитрий и Круглякова Мария, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Воронова Анжелика и Зеленова Ксения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Диков Андрей и Филимонова Галина, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Живило Андрей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Иванов Денис, Семенов Александр и Токмурзин Данил, Свердловская обл., Красноуфимский р-н, Тавринская средняя школа, учитель **Ярцев В.А.**;

— Крысанов Виктор и Малахова Юлия, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Марун Виталий, Орликова Алина и Юферева Светлана, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

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

— Якушов Александр, г. Орел, лицей № 4 им. Героя Советского Союза Г.Б. Злотина, учитель **Чапкевич И.М.**

Задача "Точные квадраты"

Напомним, что следовало ответить на вопрос, при каких основаниях систем счисления k являются точными квадратами числа:

1) 121_k; 2) $12\,321_k$.

Ответ

1) учитывая, что точный квадрат есть произведение некоторого числа на самого себя, рассмотрим такое произведение для числа 11 (записанного в некоторой системе счисления):

$$\begin{array}{rcccc} & & 1 & & 1 \\ \times & & 1 & & 1 \\ \hline & & 1 & & 1 \\ & 1 & 1 & & \\ \hline & 1 & ? & & 1 \end{array}$$

Далее возникает вопрос: "В какой системе счисления можно получить результат умножения, равный 121 (числу из условия), то есть когда 1 + 1 = 2?" Ответ здесь такой — при любом основании системы $k > 2$. Полученное условие и является решением задачи;

2) при любом $k > 3$, так как при этом $12\,321_k = 111^2_k$.

Правильные ответы прислали:

— Бородюк Анна и Василенко Татьяна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Диков Андрей, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Живило Андрей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Зеленова Ксения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Мищенко Маргарита, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Федосеева Анастасия, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

Новая задача

Определите, при каких основаниях систем счисления k является точным квадратом число $123\,454\,321_k$?

Задача "Доминошки на шахматной доске"

Напомним, что следовало ответить на вопрос: "Можно ли покрыть шахматную доску доминошка-

ми 1 × 2, чтобы свободными остались только клетки $a1$ и $h8$?”.

Ответ

Каждая доминошка покрывает одно черное и одно белое поле (расположенные рядом), поэтому доминошки закрывают равное количество черных и белых клеток. При “выкидывании” указанных в условии полей черных полей останется на два меньше, чем белых. Поэтому оставшуюся часть доминошками не закрыть.

Задачи для самостоятельной работы, предложенные в статье “Круги Эйлера (диаграммы Эйлера — Венна)”, правильно решили:

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гололобов Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Лазаренко Нина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Лежнева Александра, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

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

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Ответы к задачам

Задача 1. Не знают ни английского, ни немецкого языка — 16 человек, знают хотя бы один из указанных языков — 41 человек, знают только один английский язык — 20 человек, знают только по одному иностранному языку — 35 человек.

Задача 2. Во все три игры умеют играть 4 мальчика.

Решение задачи “Число-исполин” прислали:

— Андреев Алексей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

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

— Лежнева Александра, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Лопатин Дмитрий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Мищенко Маргарита, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Шилов Валерий, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Задачу “2014 год и степени двойки” решили:

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Гололобов Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Лазаренко Нина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Яснов Федор, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**

Задачи “Генерал и рядовой Байтиков”, “Лишний солдат” и “Бидоны с медом” правильно решили:

— Авдеева Полина, Аксененко Ирина и Чумаков Илья, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

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

— Лежнева Александра, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Леоненко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Лопатин Дмитрий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Мазанова Екатерина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Мальчугина Екатерина, Миноцкий Ян и Умаров Тимур, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**

Головоломки судоку, опубликованные в февральском выпуске, правильно решили:

— Абрамов Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Григоренко Василий, Григоренко Дмитрий и Круглякова Мария, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Волков Владимир и Глушаков Андрей, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Дронов Даниил и Крысанов Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Евтюхин Максим, г. Рязань, школа № 44, учитель **Марцинкевич Е.Е.**;

— Коростелев Иннокентий и Ланской Дмитрий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Кротов Олег и Миноцкий Ян, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Леженников Тарас, Краснодарский край, г. Приморско-Ахтарск, школа № 22, учитель **Корнеева М.В.**;

— Ломов Петр и Орлик Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Мусатов Максим и Мусатов Тимофей, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**;

— Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**,

а опубликованные в мартовском выпуске:

— Айзетуллова Ландыш, Чувашская Республика, г. Канаш, Канашский педагогический колледж, преподаватель **Кириллова Л.Н.**;

— Булатова Анна и Калугин Сергей, средняя школа села Ириновка, Новобурасский р-н Саратовской обл., учитель **Брунов А.С.**;

— Волков Владимир и Сластенина Елена, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Григоренко Василий, Григоренко Дмитрий, Дугин Евгений и Цаплина Анастасия, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Дронов Даниил и Пеньков Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Евграфов Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Евтюхин Максим, г. Рязань, школа № 44, учитель **Марцинкевич Е.Е.**;

— Загидуллина Алсу, Рахматуллина Альфира и Файзуллина Алина, Адельшинская средняя школа, Чистопольский р-н Республики Татарстан, учитель **Фатхутдинова А.А.**;

— Клипперт Илона и Одинцова Екатерина, г. Челябинск, школа № 124, учитель **Юртаева Г.Ю.**;

— Коростелев Иннокентий и Закутский Юрий, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Краснова Диана, Свердловская обл., г. Ревда, школа № 10, учитель **Игошева А.А.**;

— Ломов Петр и Федоров Андрей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Машонина Ирина, г. Фрязино Московской обл., школа № 4, учитель **Сенюта Е.И.**;

— Мусатов Тимофей, Челябинская обл., г. Златоуст, школа № 9, учитель **Мусатова И.Б.**

Профессор и студенты

Профессор написал на доске шесть утверждений:

Сегодня на моей лекции будет меньше 10 студентов.
Сегодня на моей лекции будет меньше 20 студентов.
Сегодня на моей лекции будет больше 20 студентов.
Сегодня на моей лекции будет меньше 30 студентов.
Сегодня на моей лекции будет больше 30 студентов.

На лекцию пришло *N* студентов, после чего профессор написал для каждого своего утверждения, верное оно или нет. Получилось, что четыре утверждения — неверные. Сколько студентов пришло на лекцию? Укажите все возможные ответы.

Автор задачи — Г.А. Гальперин

Верблюды и дядя Федор

В загоне зоопарка было пять верблюдов и работник зоопарка дядя Федор. Каждый верблюд плюнул три раза и получил два плевка от своих “товарищей”. Сколько плевков получил дядя Федор? (Верблюды при плевках не промахиваются и выбирают цель для плевка внутри загона. Дядя Федор, как человек воспитанный, не плюется.)

Подсчет пальцев

Один шестиклассник о себе написал так: “Пальцев у меня 24, на каждой руке 5, а на ногах 12”. Как же это могло быть?

ID-номер

При заключении договора на предоставление услуг каждому абоненту — пользователю сети присваивается уникальный 12-значный идентификационный номер — так называемый “ID-номер”, позволяющий идентифицировать пользователя не только по, например, его имени (логину) или IP-адресу. ID-номер предназначен также для пополнения баланса пользователя через различные платежные системы.

Предположим, что ID-номер представляет собой последовательность символов и используется посимвольное кодирование. При этом каждый символ кодируется минимально возможным количеством бит, а весь номер кодируется минимально возможным количеством байт. Исследуя количество требуемой памяти для описанного кодирования, получили зависимость:

Кол-во символов в номере	1	2	3	4	5	6	7	8	9	10	…
Кол-во байт для кодирования номера	1	2	2	3	4	4	5	6	6	7	

Как бы вы описали алфавит для такого кодирования ID-номера (количество символов и т.д.)?

Задачу предложила ***Е.А. Мирончик,*** *учитель информатики лицея № 111 г. Новокузнецка Кемеровской обл.*

Четыре приятеля

Четыре приятеля — Миша, Коля, Саша и Дима — проживали по следующим адресам: Лесная, 37; Цветочная, 25; Лесная, 25. Узнайте, в каком доме и на какой улице жил каждый из мальчиков, если известно, что Миша и Коля жили на одной улице, Саша и Коля жили в домах с одинаковыми номерами, а Миша и Дима были родными братьями.

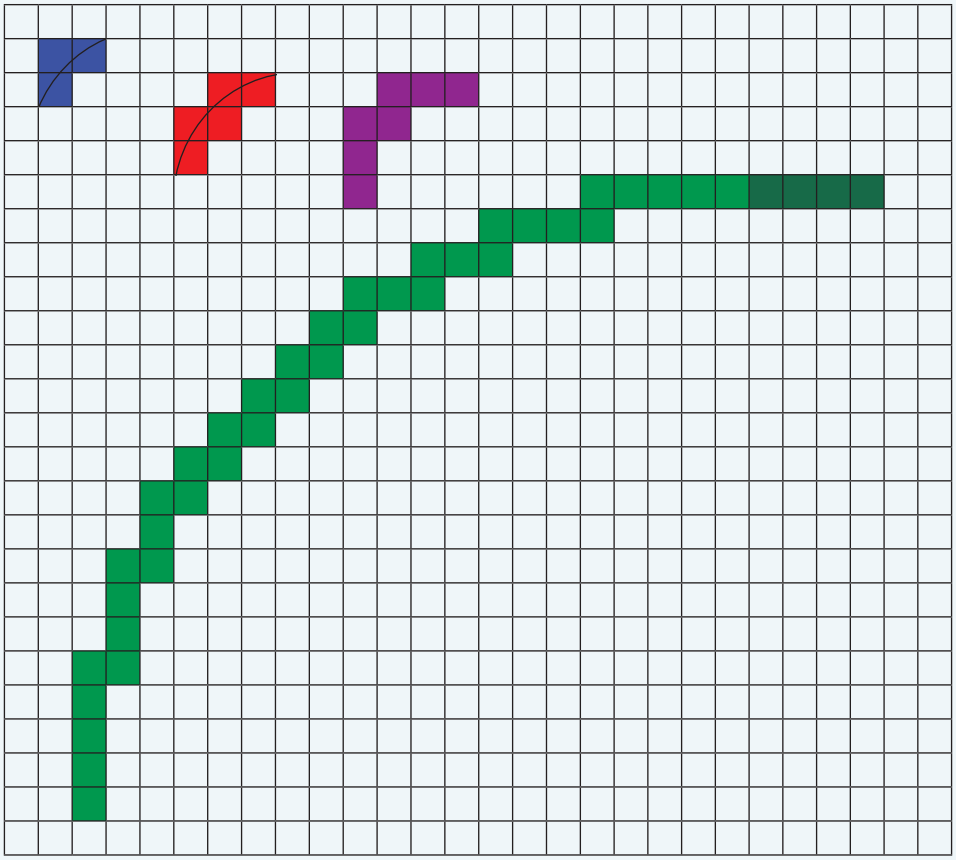
“ЛОМАЕМ” ГОЛОВУ

Дуга из пикселей

Известно, что на экране монитора компьютера изображение формируется из отдельных точек — пикселей. Это можно смоделировать как построение изображения из квадратиков мелкой сетки (см. рисунок).

Митя решил изобразить четверть окружности радиусом 100 пикселей, которая не задевает вершины пикселей и не касается их сторон. Один конец этой дуги “смотрит” строго вниз, другой — строго вправо. Изображение дуги состоит из тех пикселей, по которым дуга проходит.

Сколько пикселей понадобится на эту дугу?



Получить верное равенство

Как, переложив одну спичку, сделать так, чтобы равенство стало верным?

Задание предназначено для учащихся 1–7-х классов.

Числовой ребус “ОДЕЯЛО из ЛОСКУТОВ”

Решите, пожалуйста, числовой ребус:

ЛОСКУТ + ЛОСКУТ + ЛОСКУТ = ОДЕЯЛО

— в котором, как принято в таких головоломках, одинаковыми буквами зашифрованы одинаковые цифры, разными буквами — разные цифры.

Числовые ребусы в троичной системе

В приведенных ниже ребусах зашифрованы числа, записанные в троичной системе счисления.

Одинаковым буквам соответствуют одинаковые цифры. Звездочкой (“*”) может быть любая цифра.

1.

```
    *
+   * *
  -----
    М М
```

2.

```
    *
+   * 0
  -----
    * В
```

3.

```
    *
+   *
  -----
  В В
```

Суперлабиринт “Термины информатики”

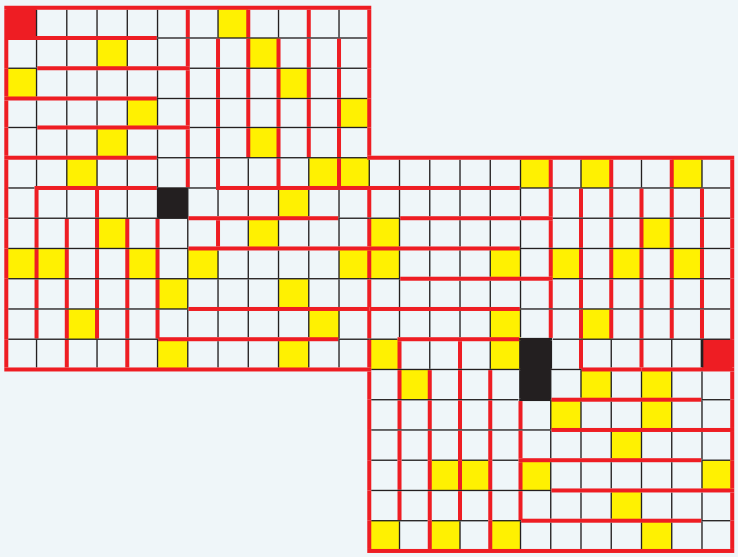
А.А. Мячев, научный консультант
ГБОУ ЦДОД “Эврика”, г. Москва

В приведенную ниже “фигурную” таблицу необходимо вписать 66 терминов, связанных информатикой, по заданным комментариям к ним. Первый термин начинается с клетки в верхнем левом углу, последний заканчивается в клетке справа, также закрашенной красным цветом. Последняя буква каждого термина (клетки для таких букв выделены желтым цветом) является первой в следующем термине. Писать термины можно по всем направлениям (слева направо, сверху вниз, справа налево и снизу вверх) внутри направляющих линий, оформленных красным цветом.

Комментарии к терминам

1. Универсальное устройство, выполняющее последовательность операций, предписанных программой.

2. Матрица близко расположенных пикселей видеоизображения.



3. Программа для подготовки текстовых документов или графических изображений.

4. Схема расположения клавиш клавиатуры и метод их кодирования.

5. Устройство, осуществляющее преобразование представления и скорости передачи информации между компьютером и внешним устройством.

6. Позиция в записи числа.

7. Периферийное устройство для чтения информации с диска и записи ее на диск.

8. Результат работы пользователя с текстовым редактором.

9. Устройство ввода информации в компьютер — шаровой манипулятор.

10. Носитель информации в ЭВМ первых поколений в виде узкой бумажной или подобной полоски.

11. Уникальный номер ячейки памяти.

12. Часть текста, ограниченная пробелами или знаками препинания.

13. Поэтапная отмена внесенных изменений в программу или в документ.

14. Электронная…

15. В текстовом редакторе Microsoft Word — текст, набранный до нажатия клавиши Enter.

16. Алгоритмическая конструкция, обеспечивающая повторение одних и тех же операций.

17. Переносной персональный компьютер.

18. Минимальный элемент изображения (без последней буквы).

19. Часть окна текстового редактора, используемая для установки полей, отступов и т.п.

20. Внутренняя организация электронно-вычислительной машины.

21. Мультипликация для имитации движений.

22. Система обозначений и правил для передачи информации.

23. Второй экземпляр (файла и др.).

24. Значок — ссылка на объект.

25. Стандартное устройство ввода информации в компьютер.

26. Язык программирования, названный в честь первой женщины — программиста.

27. Совокупность символов, используемых в языке.

28. Изображение, воспроизводящее визуальные свойства каких-либо поверхностей или графических объектов.

29. Совокупность четко определенных правил для решения задачи за определенное число шагов.

30. Модулятор-демодулятор.

31. Процесс измерения параметров несущей частоты по заданному закону.
32. Элемент электронной таблицы.
33. Участник сеанса связи.
34. Электронная схема с двумя устойчивыми состояниями.
35. Место хранения информации в процессоре.
36. Последовательность чисел, подчиняющаяся какому-либо закону.
37. Единица, равная 2,54 см.
38. Отношение размеров на экране к натуральным размерам в документе.
39. Программа для просмотра веб-страниц.
40. Механический соединитель различных частей устройства.
41. Устройство для вывода информации в персональном компьютере.
42. Прямоугольник, ограничивающий меню или т.п.
43. Характерный признак объекта.
44. Последовательность символов, предназначенная для чтения человеком.
45. Международная служба передачи сообщений.
46. Совокупность электронных документов, которые воспринимаются как единое целое, оформленных с помощью языков HTML, Java и др.
47. Обобщенный термин, характеризующий поток информации, передаваемый по каналу связи.
48. Так называют буферную (промежуточную) память.
49. Трафарет, форма представления объекта, а также способ обозначения сразу нескольких имен файлов с помощью специальных символов.
50. Процесс написания текста на компьютере, а также полный комплект чего-либо.
51. Изменение направления движения на противоположное.
52. Компьютер в сети, предоставляющий свои услуги другим компьютерам.
53. Повторитель, приемопередатчик (от английского слова).
54. Процесс поэлементного преобразования изображения в электрический сигнал или воспроизведение изображения на экране.
55. Прототип; наиболее близкий по своим характеристикам объект по отношению к данному объекту.
56. Тип графической визуализации информации.
57. Область памяти компьютера для “мусора” (ненужных файлов).
58. Процедура разрешения типовых конфликтных ситуаций.
59. Технический документ для учета видов производимых работ, а также школьный документ.
60. Наука о законах и формах мышления.
61. Программа, написанная на языке программирования Java и встраиваемая в страницу HTML.
62. Устройство отсчета времени.
63. Совокупность особенностей работы технических средств с неизменным алгоритмом функционирования.
64. Двумерный массив данных.
65. Древнеримское счетное устройство.
66. Инструмент графического редактора.

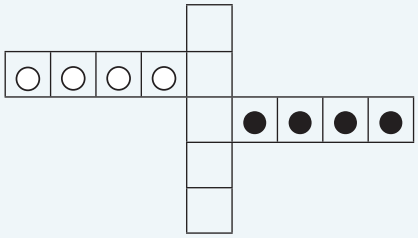
Литература

1. *Златопольский Д.М.* Интеллектуальные игры в информатике. СПб.: БХВ-Петербург, 2004.
2. *Мячев А.А.* Чайнворд по информатике: от абака до компьютера. *http://www.slideshare.net/aamchv/ss-29767607*.
3. *Воройский Ф.С.* Информатика. Новый систематизированный толковый словарь. М.: Физматлит, 2003.

От редакции. Ответы (в виде перечня терминов с их порядковыми номерами), пожалуйста, присылайте в редакцию. Можно приводить не все термины.

Переставить шашки

На клетчатом поле из 13 клеток размещены белые и черные шашки. Разработайте последовательность перемещения шашек для обмена местами шашек разного цвета.



В ответе приведите основные этапы последовательности перемещения (ситуацию после группы перемещений, решающей частную задачу).

Пляшущие человечки

В детективном романе Артура Конан Дойла сыщик Шерлок Холмс расшифровывает письмо, которое было написано с использованием кода “пляшущие человечки” (см. *рис*. 1).

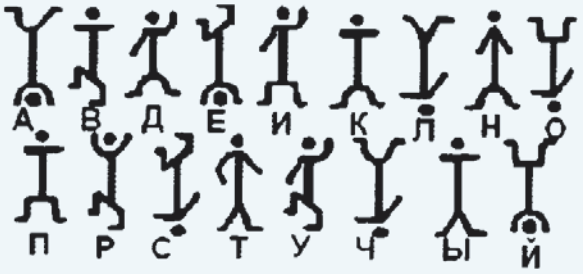


Рис. 1

Определите, какой текст на *рис*. 2 написан с использованием этого кода.

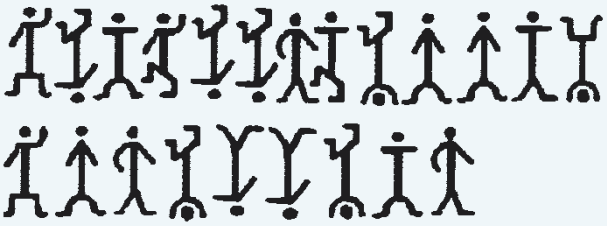


Рис. 2

Два судоку

Решите, пожалуйста, две японские головоломки “судоку”:

1) простую:

8		2		6				
3	1					4		
	4				5	3	2	
			4		2			
6		4				1		9
			6		9			
	8	9	5				4	
		5					9	1
				3		8		2

2) сложную:

8		7	2				5	6
			5				7	
	3	9		6		2		1
9	5					6		
1	7			9	8			
	4	8						
						4		
		1		8				
					6	5	3	

Ответы (можно не на все головоломки) присылайте в редакцию.

ВНИМАНИЕ! КОНКУРС!

Конкурс № 111 “Переправы”

Тур 1

Три человека со стиральной машиной хотят переправиться через реку. Лодка вмещает либо двух человек и стиральную машину, либо трех человек. Беда в том, что стиральная машина тяжелая, поэтому погрузить ее в лодку или вытащить из нее можно только втроем. Смогут ли они переправиться?

Ответ (с обоснованием) отправьте в редакцию до 1 октября по адресу: 121165, Москва, ул. Киевская, д. 24, “Первое сентября”, “Информатика” или по электронной почте: *vmi@1september.ru*. Пожалуйста, четко укажите в ответах свои фамилию и имя, населенный пункт, номер и адрес школы, фамилию, имя и отчество учителя информатики.

Конкурс будет проводиться в несколько туров, а его итоги будут подводиться с учетом всех туров в целом.

Итоги конкурса № 110 будут подведены в следующем выпуске.

КРЕПКИЙ ОРЕШЕК

Как всегда, в этой рубрике проводится разбор задач, решение которых вызвало трудности.

Задача “Перестановка двойки”

Напомним условие: “Некоторое число оканчивается на 2. Если эту цифру перенести в начало числа, то оно удвоится. Найдите наименьшее такое число”.

Приведем начало решения

Пусть искомое число имеет вид $x2$, или $10x + 2$, где x — некоторое число. Соответствующее ему число с двойкой в начале назовем “новое число”.

Ясно, что оно (*новое число*) оканчивается на 4. Однако проверка всех таких чисел на соответствие условию — дело очень трудоемкое. Забегая вперед, скажем, что для нахождения искомого числа придется проверить более чем 10^{10} (!) чисел.

Будем рассуждать так.

Если x — однозначное число, то *новое число* $= 20 + x$. Тогда, согласно условию, можем записать:

$2 \cdot (10x + 2) = 20x + 4 = 20 + x$, или $19x = 16$.

Если x — двузначное число, то *новое число* $= 200 + x$. Тогда имеем:

$20x + 4 = 200 + x$, или $19x = 196$.

При трехзначном x:

$20x + 4 = 2000 + x$, или $19x = 1996$.

Итак, для решения задачи нам нужно найти минимальное *целое* x, являющееся корнем линейных уравнений вида $19x = 16$, $19x = 196$, $19x = 1996$, …, $19x = 1(99…9)6$. Для этого мы можем использовать электронную таблицу Microsoft Excel или подобную программу (хотя, опять забегая вперед, скажем, что для “ручного” поиска понадобится проверить всего 10 чисел; но лучше пусть все сделает компьютер ☺). Общий вид листа такой:

	A	B	C
1		Правая часть уравнения	*x*
2	20	16	
3	200	196	
4	2000	1996	
…			
10	2000000000	1999999996	
11	2E+10	19999999996	
12	2E+11	2E+11	
…			
21	2E+20	2E+20	

Можно также решить задачу методом рассуждений, как это сделал Тарас Леженников, Краснодарский край, г. Приморско-Ахтарск, школа № 22 (учитель **Корнеева М.В.**), который будет награжден дипломом.

Рассуждения такие.

Последняя цифра исходного числа — это 2, значит, последняя цифра “нового” числа — это 4. А это значит, что предпоследняя цифра исходного числа — 4.

Так как при этом переноса “в уме” не было, вторая цифра “нового” числа равна $4 \cdot 2 = 8$ (она же — третья справа цифра исходного числа x).

Следующая цифра “нового” числа равна 6 ($8 \cdot 2 = 16$, 1 переходит в следующий разряд). Записываем 6 в число x.

Очередная при просмотре справа налево цифра — 3 ($6 \cdot 6 = 12 + 1$ из предыдущего разряда), 1 — “в уме”.

Описанные действия должны продолжаться до тех пор, пока последняя цифра числа после удвоения и возможного добавления 1 от предыдущего разряда не станет единицей.

Предлагаем читателям получить ответы для всех рассмотренных задач и прислать их в редакцию (фамилии всех приславших будут опубликованы).

Задача “Статистика”

Напомним условие: “В классе 25 человек, каждого из которых в зависимости от роста можно отнести к высоким, средним или низким, из них 3 двоечника и 5 отличников. Низких в классе 8 человек, высоких — 9, есть двоечники всех размеров, а среди “середняков” (не двоечников и не отличников) — 3 ученика среднего роста. Сколько отличников среднего и низкого роста, если самый высокий в классе тоже отличник?”.

Благодаря Маргариту Мищенко, Краснодарский край, г. Приморско-Ахтарск, школа № 22 (учитель **Корнеева М.В.**), и Надежду Землянскую, Челябинская обл., г. Златоуст, школа № 9 (учитель **Мусатова И.Б.**), приславших правильные ответы, приведем начало решения.

Составим таблицу, в которую запишем известную информацию, а также значения, которые можно получить, учитывая общее число учеников в классе (эти значения выделены цветом):

	Отличников	**“Середняков”**	**Двоечников**	***Всего***
Высоких	≥1		.	9
Среднего роста		≥3	.	8
Низких			.	8
Всего	5	17	3	25

Так как по условию есть двоечники всех “размеров”, то можем добавить в таблицу три новых значения:

	Отличников	**“Середняков”**	**Двоечников**	***Всего***
Высоких	≥1		1	9
Среднего роста		≥3	1	8
Низких			1	8
Всего	5	17	3	25

Предлагаем читателям продолжить анализ и получить искомые значения (ответы, пожалуйста, присылайте в редакцию).

Задача “Цепочки цифр”

Напомним, что следовало установить, какие из приведенных ниже цепочек символов:

(1) 00100120010012300100120010012340010012001001230010120010012345

(2) 00100120010012300100120010012340010012001001230010012001012345 6

(3) 00100120010012300100120010012340010012001001230010012001012345

(4) 00010012001001230010012001001234001001200100123001001200101234 5

(5) 00100120010012340010012001001230010012001001230010012001012345

(6) 00100120010012340010012001001230010012001001230010012001000134 5

соответствуют следующему правилу ее построения — первая строка состоит из одной цифры 0, а каждая из последующих цепочек создается такими действиями: в очередную строку сначала записывается цифра, на 1 большая последней цифры в предыдущей строке, к ней слева дважды подряд приписывается предыдущая строка. Пример для четырех первых цепочек:

0

001

0010012

001001200100123.

Решение

Анализ зависимости количества символов в строке L от ее номера n (см. таблицу ниже) показывает, что $L(n) = 2^n - 1$.

Номер строки, *n*	**Длина строки, *L***
1	1
2	3
3	7
4	15

Исследуем заданные цепочки. Видно, что первая цепочка короче остальных (у нее 62 знака). У правильной последовательности такой длины быть не может. Поэтому первую цепочку отбрасываем.

Так как длина остальных строк 63, то это 6-я строка ($63 = 2^6 - 1$). По правилам, в n-й строке последняя цифра должна быть равна $n - 1$, то есть в нашем случае — 5. Значит, цепочка номер 2 не подходит.

Также по правилам построения цепочек в конце должны стоять подряд цифры от 0 до 5 — в 6-й строке это нарушено, поэтому ее отбрасываем.

Далее, начало у всех строк должно быть 001, то есть не подходит и 4-я строка.

Итак, остались 3-я и 5-я цепочки. Чтобы найти нужную (а она — единственная, так как оставшиеся варианты разные), нужно отбрасывать последнюю цифру, делить оставшуюся часть пополам и сравнивать половины; если они одинаковые, то повторять эти действия до тех пор, когда либо обнаружится расхождение, либо будет получена цепочка “0” (после “001”).

Предлагаем читателям провести соответствующий анализ и найти решение.

журнал

Информатика – Первое сентября

1-е полугодие 2015 года

ПОДПИСКА

на сайте www.1september.ru и в почтовых отделениях РФ

Индекс	Название издания	Периодичн. в полугодие	1 месяц		6 месяцев	
			Каталожная цена (руб.)	Подписная цена (руб.)	Каталожная цена (руб.)	Подписная цена (руб.)
Название блока в разделе «Журналы»	**ПЕРВОЕ СЕНТЯБРЯ. ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА** (499)249-31-38					
79066	**Информатика – Первое сентября. Бумажная версия** С электронными приложениями и презентациями. *В июне не выходит.* *Подписка на июнь не принимается* *(-) 160 г 64 стр.*	5	340.00		1700.00	
12684	**Информатика – Первое сентября. Электронная версия на CD (полная копия бумажной версии)** *В июне не выходит.* *Подписка на июнь не принимается* (-) *75* г	5	118.80		594.00	
сайт 1september.ru	**Информатика – Первое сентября. Электронная версия**	5	–		–	300.00

Подписку принимают во всех отделениях связи Российской Федерации, включая Крым и Севастополь, а также на сайте www.1september.ru

При оформлении подписки **на сайте** оплата производится по квитанции в отделении банка или электронными платежами on-line

VISA MasterCard QIWI WebMoney