Романовский Иосиф Владимирович **УДК 004.04**

ЕЩЕ НЕСКОЛЬКО ПРИМЕРОВ ИСПОЛЬЗОВАНИЯ AWK

Аннотация

В статье описан накопленный автором опыт использования языка AWK при решении разнообразных практических задач.

Ключевые слова: язык AWK, язык скриптов, регулярное выражение, числа Фибоначчи.

В статье В.А. Бухваловой и В.В. Бухваловой [1] приводится много интересных и важных приложений языка AWK [2]. Их впечатления от использования этого простого и эффективного средства программирования совпадают с моими, я тоже много раз успешно использовал AWK. Но области применений были другими, и мои программы оказываются более похожими на традиционные. Думаю, что разумно пополнить список примеров задачами «других жанров».

1. Степени числа 2. Все знают, как важны для нас степени числа 2. Степени до 10 помнят, наверное, почти все, дальше… уже сложнее. А используемые нами «числительные приставки» «мега», «гига», а теперь уже и «тера», заставляют считать все дальше и дальше (не говоря уже об описанной в «Занимательной математике» Перельмана истории об изобретателе шахмат).

Обеспечим свои потребности «с запасом» и сосчитаем, например, первые 400 степеней.

```
BEGIN{ a = "1";
  r2["0"] = "0"; f["0"] = 0;  ro["0"] = "1";
  r2["1"] = "2"; f["1"] = 0;  ro["1"] = "3";
  r2["2"] = "4"; f["2"] = 0;  ro["2"] = "5";
  r2["3"] = "6"; f["3"] = 0;  ro["3"] = "7";
  r2["4"] = "8"; f["4"] = 0;  ro["4"] = "9";
  r2["5"] = "0"; f["5"] = 1;  ro["5"] = "1";
  r2["6"] = "2"; f["6"] = 1;  ro["6"] = "3";
  r2["7"] = "4"; f["7"] = 1;  ro["7"] = "5";
  r2["8"] = "6"; f["8"] = 1;  ro["8"] = "7";
  r2["9"] = "8"; f["9"] = 1;  ro["9"] = "9";
  for (i = 1; i < 401; i++) {
    n = length(a); b = ""; flag = 0;
    for (j=n; j>0; j—) {
      c = substr(a,j,1);
      if ( flag == 1) d = ro[c]; else d = r2[c];
      b = d b; flag = f[c]
    }
    if (flag == 1) a = "1" b; else a = b
    print i " " a
  }
}
```

Я думаю, что программу читатели могут разобрать сами. В ней виден перебор всех значений степени от 0 до 400, а при каждом значении удваивается число, записанное в строке **a**.

В общем-то, это арифметика младшего класса, но когда я писал эту программу, то поленился использовать машинную арифметику и трудолюбиво выписал таблицу для результатов удвоения однозначного числа, простого и с добавкой единицы. В примере 2 арифметика будет сделана чуть-чуть лучше.

Результат (400 строк) не стоит приводить полностью, выпишем начало и некоторые особые числа с комментариями и небольшим форматированием длинных чисел

```
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
10 1024                        кило
20 1048576                     мега
30 1073741824                  гига
40 1099511627776               тера
50 1125899906842624            пета
60 1152921504606846976         экса
64 18446744073709551616        число в [3] на единицу меньше
70 1180591620717411303424      зетта
80 1208925819614629174706176 йотта
100 12676 50600 22822 94014 96703 20537 6
200 16069 38044 25899 02755 41962 09234 11626 02522 20299 37827
    92835 30137 6
300 20370 35976 33448 60862 68445 68840 93781 61051 46839 36659
    36250 63614 04493 54381 29976 33367 06183 39737 6
400 25822 49878 08690 85896 55919 17200 30118 74329 70579 28292
    23512 83065 93565 40647 62201 68411 94629 64535 32801 37831
    43590 31719 72747 49337 6
```

2. Числа Фибоначчи. Эти числа, известные с XIV в., легко получающиеся друг из друга: 0, 1, 1, 2, 3, 5, 8 и т. д., каждое следующее равно сумме двух предыдущих, хорошо известны. Находить их почти так же легко, как последовательные степени двойки. Обратите внимание, как легко, в сравнении с ремесленным «табличным» вариантом первого примера, выполнены арифметические действия над разрядами чисел.

Вот программа, в которую я добавил немного комментариев

```
BEGIN{ n = 1000;  # найдем тысячу чисел
   Fpre = "0"; Fcur = "1"; Knxt = 1; # начальные установки
   for (i=2; i < n+1; i++) { # цикл расчета
     Knxt++; Fnxt = ""; over = 0;
     Rcur = length(Fcur); Rpre = length(Fpre);
     for (j = Rpre; j>0; j—) {
       v = over + substr(Fcur,Rcur,1) + substr(Fpre,Rpre,1);
         if (v > 9) {over = 1; v -= 10;} else over = 0;
         Fnxt = "" v Fnxt;
         Rcur—; Rpre--;
       }
     if (Rcur > 0) {v = over + substr(Fcur,Rcur,1)}
     else v = over;
     if (v > 0) Fnxt = "" v Fnxt;
     print "F(" Knxt ") = " Fnxt;
     Fpre = Fcur; Fcur = Fnxt; # сдвиг имеющихся чисел
   }
}
```

Обратите внимание на операторы `v = over + substr(Fcur,Rcur,1)}` и `Fnxt = " " v Fnxt`. В них неявно выполняется «приведение типов»: в первом случае к числу прибавляется строка, которая автоматически преобразуется в число, во втором случае к пустой строке прицепляется целое число, которое автоматически переводится в строковый формат. На тип преобразования в обоих случаях влияет тип первого слагаемого.

А вот результаты вычислений (мы возьмем, конечно, не все)

```
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
F(7) = 13
F(8) = 21
F(9) = 34
F(10) = 55
F(11) = 89
F(12) = 144
F(100) = 354224848179261915075
F(200) = 280571172992510140037611932413038677189525
F(300) = 22223 22446 29420 44552 97398 93461 90996 72066 66939
         09649 97649 90979 600
F(400) = 17602 36806 45013 96646 82269 45392 41125 07703 84383
         30449 21918 86725 99289 65753 45044 21601 9675
F(500) = 13942 32245 61697 88013 97243 82870 40728 39500 70256
         58769 73072 64108 96294 83255 71622 86329 06915 57658
         87622 25212 94125
F(1000)= 43466 55768 69374 56435 68852 76750 40625 80256 46605
         17371 78040 24817 29089 53655 54179 49051 89040 38798
         40079 25516 92959 22593 08032 26347 75209 68962 32398
         73322 47116 16429 96440 90653 31879 38298 96964 99285
         16003 70447 61377 95166 84922 8875
```

3. Дешифровщики-самоучки. Иногда при использовании электронной почты письма поступают в какой-нибудь странной кодировке. Получается подстановочный шифр – каждая буква заменяется какой-то другой, мы знаем такие шифры по художественной литературе («Золотой жук» Эдгара По и «Пляшущие человечки» Конана Дойля). У меня в архиве сохранилась программа, оставшаяся от времен, когда необходимость таких перекодировок возникала часто.

```
BEGIN{
fr="бвчздецъйклмнопртуфхжигюыэ|щшьасБВЧЗДЕЦЪЙКЛМНОПРТУФХЖИГЮЫЭЯЩШЬАС"
to="АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя"
#to="А–ДЕ–И-КЛМНО-РСТУ–Ч–Ы–Яа–де–и-клмно-рсту–ч–ы–я"
}
{ res = ""
  for(i=1;i<=length($0);i++)
    { k = index(fr,substr($0,i,1));
      if (k>0) res = res substr(to,k,1)
      else res = res substr($0,i,1)
    }
  print res
}
```

Здесь во вступительной части определены три строки, одна из которых превращена в комментарий. Строки `fr` и `to` определяют связь между зашифрованным и правильным тек-

стом. Эти строки используются в цикловой части для преобразования каждой строки из первой кодировки во вторую. Но я обращаю ваше внимание на третью строку: она представляет собой рабочую заготовку строки **to**. Часто можно угадать некоторые буквы: по их положению в тексте, по связи с цифрами и др. Вот эти догадки я и записывал в строку, а потом стало ясно, что у нас получаются сплошные куски русского алфавита.

4. Кэб профессора Харди. Начнем с истории, которую я прочел в очень интересной книге Дж. Литтлвуда «Математическая смесь». В Интернете есть пересказ этой истории, сделанный писателем Ч.П. Сноу. В ней участвуют знаменитый английский математик Г. Харди и его тоже знаменитый ученик-индиец С. Рамануджан, который по приезде в Англию очень часто болел.

Харди приехал в больницу Пугни, как было у него заведено, на такси,[1] *и пошел в комнату, где лежал Рамануджан. Харди всегда было трудно начать разговор, и он произнес первое, что пришло ему в голову: «Номером моего такси было 1729. По-моему, довольно непримечательное число». Рамануджан тут же воскликнул: «Нет, Харди, нет! Вы не правы! Ведь это наименьшее число, которое можно двумя разными способами представить в виде суммы двух кубов».*

На одной из наших олимпиад по программированию участникам было предложено найти другие числа, обладающие таким свойством в каком-то начальном интервале.

Жюри требовался правильный ответ. Оказалось, что проще всего найти их с помощью AWK. Прежняя моя программа была слишком сложной, и сейчас я не смог разобраться в том, что написал, поэтому предлагается новое решение, которое понятнее и хорошо пользуется особенностями ассоциативных массивов.

```
BEGIN{n = 100; nBound = 1000000;
   for (i=1; i<n-1; i++) {
     for (j=i+1; j<n; j++) arr[i*i*i+j*j*j]++;
   }
   for (k=1; k < nBound; k++)
     if (k in arr) { if (arr[k] > 1) print k }
}
```

Смотрите, здесь первый цикл находит числа, представляемые в виде суммы двух кубов, и создает для них счетчики количества представлений в массиве **counter**. Таких чисел не очень много – порядка пяти тысяч.

Второй цикл просматривает все числа до верхней границы **nBound**, выбирает те, для которых определен счетчик, и печатает те из них, у которых счетчик больше 1. Получилось 43 числа

```
 1729    110656    314496    525824    955016
 4104    110808    320264    558441    984067
13832    134379    327763    593047    994688
20683    149389    373464    684019
32832    165464    402597    704977
39312    171288    439101    805688
40033    195841    443889    842751
46683    216027    513000    885248
64232    216125    513856    886464
65728    262656    515375    920673
```

Подумайте, как изменить эту программу, чтобы она выдавала и сами способы представления отобранных чисел.

[1] В воспоминаниях самого Харди сказано «taxi-cab». Число 1729 теперь называется «числом Харди-Рамануджана».

5. Как разрезать длинный файл на отдельные куски. А это уже другая тематика, не вычисления, а работа с файлами. Пусть у нас есть длинный текстовый файл, который нужно сохранить в виде отдельных кусков. Это может быть длинная рукопись, или результат каких-то расчетов, полученных при запуске программы, или архив электронной переписки. Разрезать такой файт на куски очень просто. Нужно вставить в текст специальные *управляющие строки*, начинающие каждый кусок в формате **<УНИКАЛЬНОЕ СЛОВО> <ИМЯ ФАЙЛА>**. Например,

```
FILE part17.txt
```

Затем получившийся длинный файл пропустим через следующую AWK-программу

```
{ if ($1 == "LE") fn = $2 else print $0 >> fn }
```

Требуется ли объяснять эту программу? Читается входной поток, и по первому слову во входной строке определяется, не управляющая ли это строка. Если да, то второе слово становится значением параметра **fn**. А этот параметр используется для перенаправления выходного потока как имя файла, к которому добавится данная печать.

6. Использование регулярных выражений. Регулярные выражения – замечательная конструкция для задания поисковых выражений (см. [4]). Я с ней познакомился благодаря этому языку, а сейчас они уже распространились довольно широко.[1] Готовился к печати один из первых наших сборников, который мы решили издать с помощью системы $\TeX$. И один из авторов в довольно длинной статье стал вперемешку использовать кириллицу и латиницу. Выглядели буквы одинаково, но программа видела различие очень хорошо, указывала на ошибку, а место, занятое буквой, оставляла пустым.

И вот тогда меня спасла AWK-программа, всего в одну строчку

```
{ if (match($0,/[A-Za-z][A-пр-я]/) != 0 ) print NR " " $0 }
```

Эта программа находит в предлагаемом тексте строчки, в которых есть примыкающие друг к другу буква из латиницы и буква из кириллицы, как, например, в слове «**examрЛе**», где нарочно в английском слове стоит явно «чужая» буква. Такие строчки выводятся на печать вместе с порядковым номером.

Поиск осуществляет стандартная функция, первый параметр которой – это анализируемая строка, а второй – разыскиваемый образец. Образец, если он задан регулярным выражением, помещается между наклонными чертами. У меня этот образец состоит из двух символов. В квадратных скобках перечисляются все возможные значения этого символа: первый символ латинский, второй – русский (именно русский, а не кириллица, которая шире). Если символы идут подряд, их можно задавать диапазоном. В латинице заглавные буквы идут (в стандарте ASCII) от буквы **A** до буквы **Z**, потом идет несколько «небукв», а потом подряд от буквы **a** до буквы **z** тоже подряд. В кириллице мы тогда пользовались так называемой альтернативной кодировкой, в которой для русских букв было тоже два диапазона (какие?). Альтернативная кодировка с некоторыми изменениями сохранилась в компьютерах как кодовая страница 866. В ней русский диапазон правильнее было бы задавать как **[А-пр-ё]**. А если бы речь шла об используемой в Windows кодовой странице 1251, мы должны были бы написать **[А-яЁё]**.

Для того чтобы найти пары кириллица-латиница, нужно переставить в регулярном выражении квадратные скобки (что и было сделано). Можно было бы соединить оба случая в регулярном выражении

```
/[A-Za-z][A-пр-я]|/[A-пр-я][A-Za-z]/
```

[1] Например, регулярные выражения можно использовать при поиске в Total Commander и в очень удобном текстовом редакторе Notepad++.

(вертикальная черта соответствует логическому «ИЛИ»), но для того, чтобы упростить последующую «работу над ошибками», такое решение не рекомендуется.

7. Перевод данных из одного скучного формата в другой. Это очень частая ситуация, о которой много говорится в [1]. У меня пример из другой области.

Математик И.В.Р. улучшает свою персональную страницу и решил выложить на нее несколько своих статей, приготовив их тексты в формате pdf. Поможем И.В.Р. в создании страницы. Готовые pdf собраны в специальной папке, содержимое которой легко получается командой **dir *.pdf > list1.txt**

```
10/11/2009  05:03 PM       187,674 p62_kuhn.pdf
10/12/2009  08:25 AM       152,474 p65_jvr_sud.pdf
10/11/2009  10:36 PM       179,865 p66_contour.pdf
10/30/2009  09:58 AM       747,273 p67_cyb.pdf
10/12/2009  11:05 AM       242,556 p73_jvr_khri.pdf
10/10/2009  08:46 AM       162,536 p94_bakelm.pdf
10/14/2009  09:40 AM       250,705 pA1_jvr_lae.pdf
10/19/2009  10:59 AM       325,899 pA4_jvr_oap.pdf
```

В названиях файлов закодированы годы публикации, причем **p62** – это 1962 г., а **pA4** – это 2004. Статьи выписаны в календарном порядке по возрастанию, а мне хочется расположить их в обратном порядке. Поэтому мы сначала прочтем их и сохраним, а потом прочтем сохраненный список в обратном порядке, преобразуем и выдадим на печать.

Получаем такую программу

```
BEGIN{
 print "<HTML>"
 print "<TITLE>Список статей И.В.Р.</TITLE>"
 print "<H2>Список статей И.В.Р.</H2>"
 print "<OL>"
 k = 1;
}
{
 if (NF > 4) fn[k] = $5; k++;
}
END{
 for (j=1; j<k; j++) {
  a0 = fn[k-j]; a1 = substr(a0,2,1);
  if (a1 == "B" ) a1 = "201"
  else if (a1 == "A" ) a1 = "200"
  else a1 = "19" a1;
  a1 = a1 substr(a0,3,1);
  print "<LI> <B>" a1 "</B> <a href=" a0 ">Назв.</a>";
 }
 print "</OL>"
 print "</HTML>"
}
```

Осталось только вписать вместо **Назв.** правильные названия статей и использовать список настоящего размера.

```
<HTML>
<TITLE>Список статей И.В.Р.</TITLE>
<H2> Список статей И.В.Р.</H2>
<OL>
<LI> <B>2004</B> <a href=pA4_jvr_oap.pdf>О вычислении спектра Морса </a>
<LI>  <B>2001</B>  <a  href=pA1_jvr_lae.pdf>Расчет  корреляционной
```

```
размерности...</a>
<LI> <B>1994</B> <a href=p94_bakelm.pdf>A simple proof of Birkhoff-von
Neumann... </a>
<LI> <B>1973</B> <a href=p73_jvr_khri.pdf>Решение дискретных минимаксных...
</a>
<LI> <B>1967</B> <a href=p67_cyb.pdf>Оптимизация стационарного
управления...</a>
<LI> <B>1966</B> <a href=p66_contour.pdf>Задача о круговой расстановке...
</a>
<LI> <B>1965</B> <a href=p65_jvr_sud.pdf>О существовании независимых...
</a>
<LI> <B>1962</B> <a href=p62_kuhn.pdf>О сведении игры с полной памятью...
</a>
</OL>
</HTML>
```

Это вписывание оказалось не очень приятным, и мы сделали хитрее – составили список названий статей и переписали TeX-файлы этих статей в отдельную папку. Затем для каждой статьи выполнили AWK-программу

```
BEGIN{print FILENAME ";"}
{ if (NR < 10) print $0 >> titles.txt }
```

В файле **titles.txt** накопились начала файлов (в них заведомо были названия статей). При этом перед каждым таким началом стоит имя файла, завершающееся знаком “;”. Этот файл было несложно обработать вручную, оставив от каждой статьи одну строчку, в которой название файла и название статьи разделены знаком “;”. Полученный файл обработали программой (сравните ее с первоначальным вариантом)

```
BEGIN{
 print "<HTML>"
 print "<TITLE>Список статей И.В.Р.</TITLE>"
 print "<H2>Список статей И.В.Р.</H2>"
 print "<OL>"
 k = 1;
}
{ if (NF > 0) fn[k] = $0; k++; }
END{
 for (j=1; j<k; j++) {
  ar = fn[k-j]; r = index(ar,";");
  a0 = substr(ar,1,r-1); b0 = substr(ar,r+1);
  a1 = substr(a0,2,1);
  if (a1 == "B" ) a1 = "201"
  else if (a1 == "A" ) a1 = "200"
  else a1 = "19" a1;
  a1 = a1 substr(a0,3,1);
 print "<LI> <B>" a1 "</B> <a href=" a0 ">" b0 "</a>";
}
print "</OL>"
print "</HTML>"
}
```

Получившийся файл **index_p.html** почти полностью удовлетворял требованиям системы (по правилам имена файлов в теге **<a>** должны заключаться в кавычки, но интернет-эксплорер обошелся без кавычек).

Литература

1. *Бухвалова В. А., Бухвалова В. В.* Применение языка AWK для оперативной обработки экономической информации // наст. изд. С. 3–13.

2. The GNU Awk User's Guide, http://www.gnu.org/software/gawk/manual/html_node (дата обращения 25.06.2013).

3. *Перельман Я.И.* Живая математика, изд. 8-е, М:, «Наука», 1967.

4. *Фридл Дж.* Регулярные выражения. СПб.: «Питер», 2001.

SOME MORE EXAMPLES OF USING AWK LANGUAGE

Abstract

Here we describe the author own experience in applying AWK language to various practical problems.

Keywords: AWK, Script Language, Regular Expression, Fibonacci numbers.

Романовский Иосиф Владимирович, доктор физико-математических наук, профессор, кафедра исследования операций математико-механического факультета СПбГУ, josephromanovsky@gmail.com.