КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БОЧАРОВ Б.П.,
ВОЕВОДИНА М.Ю.

UNICODE и формат MARC

В одной из библиотек, обратившихся за помощью в редакцию журнала, было принято вполне разумное решение использовать бесплатную автоматизированную информационно - библиотечную систему KOHA (http://koha.org) совместно с уже работающей программой Liber. Первая проблема, с которой столкнулась библиотека, – перенос информации из одной системы в другую. На первый взгляд, эта задача кажется тривиальной, тем более, что обе системы (как утверждают разработчики) поддерживают формат MARC 21. Однако простая процедура экспорта из Liber и импорта в систему KOHA не привела к ожидаемым результатам. Причина неудачи: в системе KOHA поддерживается только кодировка UNICODE (точнее, кодировка UTF-8).

Попытка произвести перекодировку файла в формате MARC из кодировки dos в кодировку UTF-8 и экспортировать полученный файл в программу KOHA тоже закончилась неудачей. Судя по всему, перекодировка информации (с увеличением длины поля) испортила структуру записей, и подсистема импорта не смогла их прочитать.

Формат MARC предусматривает два способа обработки информации:

1. можно использовать длину записи и смещение полей относительно начала записи;

2. можно считывать информацию, используя ограничители конца записи и конца поля.

Следует отметить, что эти два способа практически независимы друг от друга, такой способ организации записи позволяет осуществлять двойной контроль целостности информации. В данном случае нам не повезло: программа KOHA импортирует записи с помощью первого способа чтения информации в формате MARC. Использование второго способа, при котором можно игнорировать все числовые значения длин полей, скорее всего, позволило бы корректно импортировать информацию.

Это утверждение может быть легко проверено, у нас уже были готовые программы работы с файлами в формате MARC (см. [1-3]). Мы переводили запись MARC в текстовый файл специального вида, используя для чтения информации маркеры конца записи и конца поля (второй способ). Затем текстовые записи изменялись и переводились в формат MARC, при этом все необходимые длины и смещения аккуратно пересчитывались.

В результате получилась такая процедура перекодировки:

• перевод файла MARC в текстовый файл;

• перекодировка текстового файла из ASCII в UTF-8 (используется любой стандартный перекодировщик);

• перевод полученного текстового файла в формат MARC.

Файлы, перекодированные с помощью этой процедуры, вполне корректно воспринимались программой KOHA.

В принципе, исходную задачу можно считать решенной. Однако смущает тот факт, что процедура перекодировки автоматизирована не полностью.Кроме того, вполне естественно иметь собственный перекодировщик, который удачно впишется в набор программ для работы с файлами формата MARC.

Разработка перекодировщика не представлялась сложной (как впоследствии и оказалось), и мы приступили к написанию программы.

Приведем для начала краткую информацию о кодировке UNICODE, которую легко найти в Интернете (отметим, что наше первоначальное представление о стандарте UNICODE, как о системе кодирование каждого символа двумя байтами, сильно устарело).

Юнико́д, или Уникóд (англ. Unicode) — стандарт кодирования символов, позволяющий представить знаки практически всех письменных языков. Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода» (англ. Unicode Consortium), объединяющей крупнейшие IT-корпорации. Применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей: в документах Unicode могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становятся ненужными кодовые страницы.

Стандарт состоит из двух основных разделов: универсальный набор символов (UCS, Universal Character Set) и семейство кодировок (UTF, Unicode Transformation Format). Универсальный набор символов задаёт однозначное соответствие символов кодам — элементам кодового пространства, представляющим неотрицательные целые числа. Семейство кодировок определяет машинное представление последовательности кодов UCS.

Первая версия Юникода представляла собой кодировку с фиксированным размером символа в 16 бит, то есть общее число кодов было 2^{16} (65536). Отсюда происходит практика обозначения символов четырьмя шестнадцатеричными цифрами, например, U+0410. При этом в Юникоде планировалось кодировать не все существующие символы, а только те, которые обязательны в повседневном обиходе.

Редко используемые символы должны были размещаться в «области символов для частного использования» (Private Use Area), которая первоначально занимала коды U+D800…U+F8FF.

В дальнейшем, однако, было принято решение кодировать все символы и в связи с этим значительно расширить кодовую область. Одновременно с этим, коды символов стали рассматриваться не как 16-битные значения, а как абстрактные числа, которые в компьютере могут представляться множеством разных способов.

Поскольку в ряде компьютерных систем (например, Windows NT) уже были реализованы 16-битные символы, было решено всё наиболее важное кодировать только в пределах первых 65536 позиций (так называемая англ. Basic Multilingual Plane, BMP). Остальное пространство используется для «Дополнительных символов» (англ. Supplementary Characters): систем письма вымерших языков или очень редко используемых китайских иероглифов, математических и музыкальных символов.

Для совместимости со старыми 16-битными системами была изобретена система UTF-16, где первые 65536 позиций отображаются непосредственно как 16 - битные числа, а остальные представляются в виде «суррогатных пар»:

первый элемент пары из области U+D800…U+DBFF,

второй элемент пары из области U+DC00…DFFF).

Для суррогатных пар была использована часть кодового пространства, ранее отведённого для «символов для частного использования».Поскольку в UTF-16 можно отобразить только $2^{20}+2^{16}$ (1114112) символов, то это и было выбрано в качестве окончательной величины кодового пространства Юникода.

Хотя кодовая область Юникода была расширена за пределы 2^{16} уже в версии 2.0, первые символы в «верхней» области были размещены только в версии 4.0.

UTF-8 — это представление Юникода, обеспечивающее наилучшую совместимость со старыми системами, использовавшими 8 - битные символы. Текст, состоящий только из символов с номером меньше 128, при записи в UTF-8 превращается в обычный текст ASCII.

И наоборот, в тексте UTF-8 любой байт со значением меньше 128 изображает символ ASCII с тем же кодом. Остальные символы Юникода изображаются последовательностями длиной от 2 до 6 байтов (на деле, только до 4 байт, поскольку использование кодов больше 2^{21} не планируется), в которых первый байт всегда имеет вид 11xxxxxx, а остальные — 10xxxxxx.

В процедуре разработки перекодировщика условно выделим четыре этапа.

1. Получение таблицы перекодировки, т.е. файла в котором каждому символу ASCII соответствует представление этого симво-ла в UTF-8 (каталог part1).

2. Разработка перекодировщика из ASCII в UTF-8 (каталог part2).

3. Разработка перекодировщика из UTF-8 в ASCII (каталог part3).

4. Создание «готового продукта» – перекодировщика файлов в формате MARC из ASCII в UTF-8 и обратно (каталог part 4).

Все файлы, относящиеся к каждому этапу, находятся в отдельном каталоге приложения.

Приступим к подробному описанию процедуры разработки перекодировщика.

Этап 1 (каталог part1)

Программа g1.awk генерирует символы ASCII с кодами от 32 до 255. Каждый символ записывается в отдельную строку.

```
BEGIN{
  for(i=32; i<=255; i++)
    printf("%c\n",i);
}
```

Программа вызывается на выполнение с помощью командного файла go1.bat.

```
..\gawk --re-interval -f g1.awk > src.zzz
copy src.zzz win_utf.zzz
copy src.zzz dos_utf.zzz
```

В результате генерируется файл src.zzz (коды ASCII). Мы собираемся перекодировать информацию из двух русских кодировок ASCII – Windows (CP 1251) и Dos Text. Скопируем файл src.zzz два раза для перекодировки в формат UTF-8.

Для перекодировки символов ASCII в UTF-8 воспользуемся любым стандартным перекодировщиком. Нами использован плагин к программе FAR.

Файл win_utf.zzz перекодируем из Windows (CP 1251) в UTF-8, а файл dos_utf.zzz из Dos Text в UTF-8.

Программа g2.awk строит таблицу соответствия между символами ASCII и их представлениями в формате UTF-8. Символы, которые в двух форматах совпадают, в таблицу соответствия не заносятся.

```
{
  u = $0;
  getline a < "src.zzz";
  if(u != a) print a u;
}
```

В результате работы программы получается файл, в каждой строке которого записаны символ в формате ASCCI (первая позиция) и представление символа в UTF-8 (остальные позиции). Например, строка символов xC: обозначает, что русской букве «x» в кодировке Windows (десятичный код 245) в формате UTF-8 соответствуют два символа «C» (код 209) «:» (код 133).

Командный файл go2.bat дважды выполняет программу, в результате генерируется таблица перекодировки из Windows в UTF-8 (win_utf.tab) таблица перекодировки из DOS в UTF-8 (dos_utf.tab).

```
..\gawk --re-interval -f g2.awk win_utf.zzz > win_utf.tab
..\gawk --re-interval -f g2.awk dos_utf.zzz > dos_utf.tab
```

Отметим два свойства полученных таблиц.

1. Размер таблиц разный, в файле dos_utf.tab - 67 строк, а в файле win_utf.tab – 115.
2. Все символы ASCII, независимо от исходной кодировки, представлены в UTF-8 двумя символами. Исключение составляет знак «№», который кодируется тремя символами.

Этап 2 (каталог part2)

На этом этапе рассмотрим перекодировщик из ASCII в UTF-8. Исходный текст может быть в кодировке Windows и Dos. Программа называется to_utf.awk.

Сначала читается таблица перекодировки (в шаблоне BEGIN, перед чтением первой строки исходного файла). Вид таблицы задается с помощью переменной src, значение которой задается в строке вызова.

```
BEGIN{
```

Если переменная src имеет значение “d”, то перекодировка будет осуществляться из Dos в UTF-8 и имя таблицы перекодировки – dos_utf.tab.

```
  if(src == "d") fn = "dos_utf.tab";
```

Если переменная src имеет значение “d”, то перекодировка будет осуществляться из Windows в UTF-8 и имя таблицы перекодировки – win_utf.tab.

```
  if(src == "w") fn = "win_utf.tab";
```

Если переменная src не равна “d” или “w”, то перекодировка осуществляться не будет.

```
  if(fn != "")
```

Читаем таблицу перекодировки в цикле. В переменную s считываем строку из таблицы.

```
  while((getline s < fn) > 0){
```

Выделяем первый символ (ASCII) из строки s и записываем в переменную s1.

```
    s1 = substr(s,1,1);
```

Удаляем из строки s первый символ, оставшиеся символы код UTF-8.

```
    sub(/^./,"",s);
```

В массив a_trn записываем код символа в ASCII (как индекс) и код в UTF-8 (как значение элемента массива).

```
    a_trn[s1] = s;
```

Завершаем цикл чтения таблицы и шаблон BEGIN.

```
  }
}
```

Приступаем к последовательной перекодировке строк исходного файла. Если таблица перекодировки не считана, то строка в выходной файл записывается без изменений.

```
{
 if(fn == ""){
  print $0;
  next;
 }
```

Приступаем к перекодировке. Устанавливаем начальные значения переменных tmp (исходная строка) buf (выходная строка).

```
 tmp = $0;
 buf = "";
```

Цикл перекодировки (останавливается, когда исходная строка полностью перекодирована).

```
 while(tmp != ""){
```

В переменную s записываем первый символ исходной строки.

```
  s = substr(tmp,1,1);
```

Если символ s записан в массив a_trn (как индекс), то в выходную строку записывается значение элемента массива с этим индексом, в противном случае символ записывается без изменений.

```
  s1 = s;
  if(a_trn[s] != "") s1 = a_trn[s];
  buf = buf s1;
```

Удаляем из исходной строки первый символ и переходим к следующему шагу цикла. Если исходная строка (переменная tmp) пуста, то перекодированная строка (buf) записывается в выходной файл. Программа переходит к обработке следующей строки.

```
  sub(/^./,"",tmp);
 }
 print buf;
}
```

Для тестирования программы создадим два ASCII - файла (в кодировках Dos и Windows). В эти файлы запишем все символы, которые можно ввести с помощью клавиатуры.

Информация в файлах tst_dos.txt и tst_win.txt одна и та же, они отличаются только кодировкой.

```
`1234567890-=\
~!@#$%^&*()_+|
qwertyuiop[]
QWERTYUIOP{}
asdfghjkl;'
ASDFGHJKL:"
zxcvbnm,./
```

```
ZXCVBNM<>?
ё1234567890-=\
Ё!"№;%:?*()_+/
йцукенгшщзхъ
ЙЦУКЕНГШЩЗХЪ
фывапролджэ
ФЫВАПРОЛДЖЭ
ячсмитьбю.
ЯЧСМИТЬБЮ,
```

Командный файл go.bat вызывает перекодировщик два раза.

```
..\gawk --re-interval -f to_utf.awk -v src=d tst_dos.txt
>d_utf.txt
..\gawk --re-interval -f to_utf.awk -v src=w tst_win.txt
>w_utf.txt
```

Отметим, что полученные файлы d_utf.txt (результат перекодировки из Dos в UTF-8) и w_utf.txt (результат перекодировки из Windows в UTF-8) одинаковые. Файлы для проверки перекодировщика tst_dos.txt и tst_win.txt.

Этап 3 (каталог part3)

На этом этапе рассмотрим перекодировщик из UTF-8 в ASCII. Выходной файл может быть в кодировке Windows и Dos.

Программа называется to_asc.awk.

Чтение таблиц перекодировки осуществляется так же, как на предыдущем этапе.

Различия заключаются в том, что вид таблицы перекодировки задает переменная res и массив a_trn заполняется наоборот – код в UTF-8 записывается как индекс, а код в ASCII как значение элемента массива.

```
BEGIN{
  if(res == "d") fn = "dos_utf.tab";
  if(res == "w") fn = "win_utf.tab";
  if(fn != "")
    while((getline s < fn) > 0){
      s1 = substr(s,1,1);
      sub(/^./,"",s);
      a_trn[s] = s1;
    }
}
```

Если таблица перекодировки не считана, то строка в выходной файл записывается без изменений.

```
{
  if(fn == ""){
    print $0;
    next;
  }
```

Устанавливаем начальные значения переменных tmp (исходная строка) и buf (выходная строка).

```
  tmp = $0;
  buf = "";
```

Начало цикла перекодировки (останавливается, когда переменная tmp становится пустой строкой).

```
  while(tmp != ""){
```

Проверяем, не начинается ли исходная строка с символа, который кодируется в UTF-8 тремя байтами.

Обязательно контролируем длину исходной строки (должна быть не меньше трех). Если входная строка начинается с символов, записанных в массив a_trn, то записываем соответствующий символ ASCII в выходную строку, удаляем три символа из исходной строки и переходим к следующему шагу цикла.

```
    s = substr(tmp,1,3);
```

```
   if(length(s) == 3)
     if(a_trn[s] != "") {
       buf = buf a_trn[s];
       sub(/^.{3}/,"",tmp);
       continue;
     }
```

Так же обрабатываем «двухбайтовые» символы UTF-8.

```
   s = substr(tmp,1,2);
   if(length(s) == 2)
     if(a_trn[s] != "") {
       buf = buf a_trn[s];
       sub(/^.{2}/,"",tmp);
       continue;
     }
```

Исходная строка начинается с символа, который отсутствует в таблице перекодировки и записывается в выходную строку без изменений.

```
   buf = buf substr(tmp,1,1);
   sub(/^./,"",tmp);
  }
  print buf;
}
```

Для проверки работы программы переписываем из каталога part2 файл d_utf.txt (или w_utf.txt), называем его utf.txt и запускаем командный файл go.bat.

```
..\gawk --re-interval -f to_asc.awk -v res=d utf.txt >tst_
dos.txt
..\gawk --re-interval -f to_asc.awk -v res=w utf.txt >tst_
win.txt
```

Исходный файл перекодируется в tst_dos.txt (в кодировке Dos) и tst_win.txt (в кодировке Windows). Эти файлы совпадают с соответствующими файлами в каталоге part2.

Этап 4 (каталог part4)

На этом этапе рассмотрим перекодировщик файлов в формате MARC. Для этого нам потребуются еще две программы:

iso2text.awk – перевод файлов формата MARC в текстовый вид;

text2iso.awk – перевод текстовых файлов в формат MARC.

Логика работы этих программ и примеры их использования подробно описаны в [1–3], поэтому приведем их тексты без комментариев.

Программа iso2text.awk

```
BEGIN{
  FS = "-";
  RS = "";
}

{
  tmp = $1;
  m1 = substr(tmp,6,7);
  m2 = substr(tmp,18,7);
  printf("xxxxx%sxxxxx%s\n",m1,m2);
  sub(/^.{24}/,"",tmp);
  i = 0;
  while(tmp != ""){
    nfld[i++] = substr(tmp,1,3);
    sub(/^.{12}/,"",tmp);
  }
  for(i=2; i<=NF; i++)
    if($i != "") print nfld[i-2] $i;
  print "***** End of record N " NR;
}
```

Программа text2iso.awk

```
BEGIN{
  iso_FS = "-";
  iso_RS = "";
  getline m0;
  m1 = substr(m0,6,7);
  m2 = substr(m0,18,7);
  curNfld = 0;
}
/^\*\*\*\*\*/{
  spr = "";
  adr = 0;
  rec = "";
  for(i=0; i<curNfld; i++){
    l = length(fld_val[i]);
    spr = spr sprintf("%3.3d%4.4d%5.5d",fld_
num[i],l,adr);
    rec = rec fld_val[i];
    adr += l;
  }
  spr = spr iso_FS;
  rec = rec iso_RS;
  adr = length(spr) + 24;
  l = length(spr) + length(rec) +24;
  printf("%5.5d%s%5.5d%s%s%s",l,m1,adr,m2,spr,rec
);
  delete fld_val;
  delete fld_num;
  getline m0;
  m1 = substr(m0,6,7);
  m2 = substr(m0,18,7);
  curNfld = 0;
  next;
}
```

```
{
  vfl = $0 iso_FS;
  nfl = substr(vfl,1,3);
  sub(/^.{3}/,"",vfl);
  fld_num[curNfld] = nfl;
  fld_val[curNfld] = vfl;
  curNfld++;
}
```

Исходный файл MARC dos.mrc (в кодировке Dos).

Командный файл asc2utf.bat осуществляет перекодировку исходного фай-ла в UTF-8.

```
..\gawk --re-interval -f iso2text.awk dos.mrc >dos.txt
..\gawk --re-interval -f to_utf.awk -v src=d dos.txt
>utf.txt
..\gawk --re-interval -f text2iso.awk utf.txt >utf.mrc
```

Последующие действия:

1. файл dos.mrc переводится в текстовый вид (dos. txt);
2. полученный файл перекодируется в UTF-8 (utf.txt).
3. перевод файла utf.txt в формат MARC (файл utf.mrc).

Командный файл utf2asc.bat осуществляет обратную перекодировку файла utf.mrc в Dos.

```
..\gawk --re-interval -f iso2text.awk utf.mrc >utf1.txt
..\gawk --re-interval -f to_asc.awk -v res=d utf1.txt
>dos1.txt
..\gawk --re-interval -f text2iso.awk dos1.txt >dos1.mrc
```

Файлы dos.mrc и dos1.mrc, dos.txt и dos1.txt, utf.txt и utf1.txt совпадают, что подтверждает работоспособность перекодировщика.

Литература

1. Бочаров Б.П., Воеводина М.Ю. AWK - универсальная программа работы с текстовыми файлами // Библиотеки учебных заведений.– 2002.– N 4.– с. 39-53.

2. Бочаров Б.П., Воеводина М.Ю. Глобальная корректировка БД с использованием программы AWK // Библиотеки учебных заведений.– 2003.– N 7.– с. 37-59.

3. Бочаров Б.П., Воеводина М.Ю. Формирование отчетов в электронных каталогах // Библиотеки учебных заведений.– 2003.– N 10.– с. 42-60.