NAME

TRANSLIT

Program to transliterate texts in different character sets. The program converts input character codes (or sequences of codes) to a different set of output character codes (or sequences of codes). Intended for transliteration to/from phonetic representation of foreign letters with Latin letters from/to special national codes used for these letters. It supports simple matches, character lists and flexible matches via regular expressions. The new transliteration schemes are easily added by creating simple transliteration tables. Multiple character sets are supported for input and output. It does not yet support UNICODE, but some day it will.

COPYRIGHT

SYNOPSIS

translit [**-i** *inpfile*][**-o** *outfile*][**-d**][**-t** *transtbl* | *transtbl*]

OPTIONS

-i *inpfile*

inpfile is a name of input file to be transliterated. If "**-i**" is not specified, the input is taken from standard input.

-o *outfile*

outfile is an output file, where the transliterated text is stored. If "**-o**" is not specified, the output is directed to the standard output. Program will not overwrite the existing file. If file exists, you need to delete it first.

-d Some information on character codes read from transliteration table file are sent to standard error ("*stderr*"). Useful when developing new transliteration tables.

-t *transtbl*

transtbl is a transliteration table file which you want to use. The "**-t**" option may be omitted if the *transtbl* is specified as the last parameter on the command line. The program first tries to locate *transtbl* file in the current directory, and if not found, it searches the directory chosen at compilation/installation time in "*paths.h*". If no "*transtbl*" is given, the default file name specified in "*paths.h*" is taken. The compile/installation time defaults in "*paths.h*" for the search directory and the default file name can be overiden by setting environment variables: TRANSP and TRANSF, respectively (see below).

ENVIRONMENT VARIABLES

The default path to the directory holding transliteration tables can be overiden by setting environment variable TRANSP. The default name for the transliteration table can be overiden by setting TRANSF environment variable. However, when the transliteration file is given on the command line, it will overide the defaults and environment setting. Here are some examples of setting environment variables for different operating systems:

UN∗X System

If you are using *csh* (C-shell):

```
setenv TRANSP /home/john/translit/
setenv TRANSF koi8-tex.rus
```

If you are using *sh* (Bourne Shell):

```
set TRANSP=/home/john/translit/
export TRANSP
set TRANSF=koi8-tex.rus
export TRANSF
```

VAX-VMS System

```
TRANSP:==SYS$USER:[JOHN.TRANSLIT]
TRANSF:==KOI8-TEX.TBL
```

PC-DOS or MS-DOS

```
SET TRANSP=C:\JOHN\TRANSLIT\
SET TRANSF=KOI8-TEX.TBL
```

Note that the directory path has to include concluding slashes, \ or / .

EXAMPLES

```
cat text.koi8 | translit koi8-tex.rus > text.tex
```

in UN∗X is equivalent to:

```
translit -t koi8-tex.rus -o text.tex -i text.koi8
```

and converts file text.koi8 to file text.tex using transliteration specified in the file koi8-tex.rus.

```
translit -i text.koi8 koi8-cl.rus
```

displays the converted text from file text.koi8 on your terminal. The conversion table is koi8-cl.rus (KOI8 --> Library of Congress).

```
translit -i text.alt -t alt-koi8.rus | translit -o text.tex -t koi8-tex.rus
```

is essentially equivalent to the following two commands in UN∗X or MS-DOS:

```
translit -i text.alt -o junkfile -t alt-koi8.rus
translit -i junkfile -o text.tex -t koi8-ltx.rus
```

and converts the file in ALT character set to a LaTeX file for printing.

```
translit -i russ.txt pho-koi8.rus | translit -o russ.tex koi8-ltx.rus
```

converts file russ.txt from phonetic transliteration to LaTeX file russ.tex for printing.

TRANSLITERATION TABLES

The following transliteration files are available with the current distribution. Consult the comments in the individual files for details.

koi8-tex.rus

Conversion table which changes the file in KOI8 (8 bit character set used by RELCOM news service) to a Plain TeX file for printing with *AMS* WNCYR fonts.

koi8-ltx.rus

Conversion table which changes the file in KOI8 (8 bit character set used by RELCOM news service) to LaTeX file for printing with *AMS* WNCYR fonts.

ltx-koi8.rus

Conversion table for the LaTeX to KOI8 conversion. Note that it will not handle complicated cases, since LaTeX is a program, and only TeX can convert a LaTeX source to the characters. However, it should work OK for simple cases of text only files, and may need some editing for complicated cases.

k8-tavtt.rus

Converts KOI8 to Bill Tavolga cyrttf truetype font mapping.

hex-text.rus
Converts hexcodes to actual codes. Some e-mail programs convert characters with codes larger than 127 to hexadecimal numbers like =AB, =9C, etc. This table converts hexadecimal numbers back to codes.

alt-gos.rus
This is a transliteration data file for converting from ALT (Bryabrins alternativnyj variant used in many popular wordprocessors) to GOSTSCII 84 (approx. ISO-8859-5?)

alt-koi8.rus
This is a transliteration data file for converting from ALT to KOI8. KOI8 is meant to be GOST 19768-74 (as used by RELCOM).

gos-alt.rus
This is a transliteration data file for converting GOSTSCII 84 (approx. ISO-8859-5?) to ALT (Bryabrins alternativnyj variant)

gos-koi8.rus
This is a transliteration data file for converting GOSTSCII 84 (approx. ISO-8859-5?) to KOI8 used by RELCOM KOI8 is meant to be GOST 19768-74

koi8-alt.rus
This is a transliteration data file for converting from KOI8. KOI8 is meant to be GOST 19768-74, to ALT (Bryabrins alternativnyj variant)

koi8-gos.rus
This is a transliteration data file for converting from KOI8 (Relcom). KOI8 is meant to be GOST 19768-74, to GOSTSCII 84 (approx. ISO-8859-5)

koi8-7.rus
This file converts from KOI8 to KOI7.

koi7-8.rus
This file converts from KOI7 to KOI8. Before you attempt the conversion, you might need to perform a simple edit on your file. You MUST read the comments in *koi7-8.rus* before you attempt this conversion.

koi7nl-8.rus
This file assumes that there are only Russian letters (no Latin) in the input file. If you have Latin letters, and you inserted SHIFT-OUT/IN characters, use file *koi7-8.rus*.

koi8-lc.rus
This file converts KOI8 to the Library of Congress transliteration. Some extensions are added.

koi8-php.rus
This file converts KOI8 to the Pokrovsky transliteration.

php-koi8.rus
This file converts from Pokrovsky transliteration to KOI8.

koi8-phg.rus
This file converts from KOI8 to GOST transliteration.

phg-koi8.rus
This file converts from GOST transliteration to KOI8.

pho-koi8.rus
This is a table which will convert from many "phonetic" transliteration schemes to KOI8. It is elaborate and it takes a lot of time to transliterate the file using this table. Some transliterations are hopeless and internally inconsistent (as humans...), so the results cannot be bug free. You might want to modify the file, if your transliteration patterns are different than those assumed in

this file. You may also want to simplify this file if the phonetic transliteration you are converting is a sound one (most are not, e.g., they use e for je and e oborotnoye, ts for c and t-s, h for kha, i for i-kratkoe, etc.).

INTRODUCTION

If you do not intend to write your own transliteration tables, you may skip this description and go directly to the installation and copyright sections. However, you might want to read this material anyhow, to better understand the traps and complexities of transliteration. It is frequently necessary to transliterate text, i.e., to change one set of characters (or composite characters, phonemes, etc.) to another set.

On computers, the transliteration operation consists of converting the input file in some character set to the output file in another character set.

In the simplest case, the single characters are transliterated, i.e, their codes are changed according to some transliteration table. This is called remapping and, assuming the one-to-one mapping, the task can be accomplished by a simple pseudo program:

```
new_char_code = character_map[old_char_code];
```

If the one-to-one correspondence does not exist (i.e., some codes may be present in one set, but do not have corresponding codes in another set), precise transliteration is not possible. In such cases there are 3 obvious possibilities:

1. skip characters which do not have counterparts,
2. retain unchanged codes of these characters,
3. convert the codes to multicharacter sequences.

In some cases, the file can contain more than one character sets, e.g., the file can contain Latin characters (e.g. English text) and Cyrillic characters (e.g. Russian text). If the character codes assigned to characters in different sets do not overlap, this is still a simple mapping problem. This is a case with KOI8 or GOSTCII character tables for Russian, which reserve the lower 127 codes for standard ASCII codes (which include all Latin characters) and characters with codes above 127 for Cyrillic letters.

If character codes overlap, there is a SHIFT-OUT/SHIFT-IN technique in which the meaning of the character sequence is determined by an opening code (or sequence of characters codes). In this case, the meaning of the series of characters is determined by the SHIFT-OUT character (or sequence) which precedes them. The SHIFT-IN character (or sequence) following the series of characters returns the "reader" to the default or previous status. To schemes are used:

```
(char_set_1)(SHIFT-IN[1])(SHIFT-OUT[2])(char_set_2)...
```

or

```
(char_set_1)(SHIFT-OUT[2])(char_set_2)(SHIFT-OUT[1])char_set_1...
```

Since computer keyboards, screens, printers, software, etc., are by necessity language specific (the most popular being ASCII), there is a problem of typing foreign language text which contains letters different than standard Latin alphabet. For this reason, many transliteration schemes use several Latin letters to represent a single letter of foreign alphabet, for example:
zh is used to represent cyrillic letter zhe, \"o may be used to represent the o umlaut, etc.

If there is one-to-one mapping of such sequences to another alphabet, it is also easy to process. However, it is necessary to substitute longest sequences first. For example, a frequently used transliteration for cyrillic letters:

shch	---	letter **shcza**	221 (decimal KOI8 code)
sh	---	letter **sha**	219
ch	---	letter **cze**	222
c	---	letter **tse**	195
h	---	letter **kha**	200

a	---	letter **a**	193

Obviously, in this case, we should proceed first with converting all *shch* sequences to **shcha** letter, then two-character *sh* and *ch*, and then single character **c** and **h**. Generally, for the one-to-one transliteration, the longest sequences should be precessed first, and the order of conversion within sequences of the same length makes no difference. For example, converting the word "shchah" to KOI8 should proceed in a following way:

shchah --> (221)*ah*, (221)*ah* --> (221)(193)*h*, (221)(193)*h* --> (221)(193)(200)

There is a multitude of reasons why transliteration is done. I wrote this program having in mind the following ones:

1) to print cyrillic text using TeX/LaTeX and cyrillic fonts
2) to read KOI8 encoded messages from Russia on my ASCII terminal.

However, I was trying to make it flexible to accommodate other uses.

PROGRAM OPERATION

The program converts the input file to an output file using transliteration rules from the transliteration rule file which you specify with option **-t**. Some examples of transliteration rule files are enclosed. Before program can be used, the transliteration rules need to be specified.

These are given as a file which consist of the following parts described below:

1) File format number (it is 1 at this moment)
2) Delimiters used to enclose a) simple strings, b) character lists, c) regular expressions
3) Starting sequence for output
4) Ending sequence for output
5) Number of input "character sets"
6) SHIFT-OUT/SHIFT-IN sequences for each input character set
7) Number of output "character sets"
8) SHIFT-OUT/SHIFT-IN sequences for each output character set
9) Transliteration table

GENERAL COMMENTS

The transliteration rules file consists of comments and data. The comments may be included in the file as:

a) line comments --- lines starting with ! or # character (# or ! must be in the first column of a line) are treated as comments and are not read in by the program.
b) comments following all required entries on the line. They must be separated by at least one space from the last data entry on the line and need not start with any particular character. These comments cannot be used within multiline sequences.

The data entries consist of integer numbers and strings. The strings may represent:

a) plain strings
b) character lists
c) regular expressions

All strings which appear in the file, are processed through the "string processor", which allows entering unprintable characters as codes. The character code is specified as a backslash "\" followed by at least 2 digit(s) (i.e., \01 produces code=1, but \1 is passed unchanged). The following formats are supported:

\0123 character of octal code 123 (when leading zero present)
\123 character of decimal code 123 (when leading digit is not zero)
\0o123 or \0O123 character of octal code 123
\0d123 or \0D123 character of decimal code 123
\0xA3 or \0XA3 or \0xa3 character of hexadecimal code A3

The allowed digits are 0-7 for octal codes, 0-9 for decimal codes and 0-F (and/or 0-f) for hexadecimal codes. In a situation when code has to be followed by a digit character, you need to enter the digit as a code. E.g., if you want character \0xA3 followed by a letter C, you need to specify letter C as a code

(\0x43 or \103 or \0o103 or \0d67) and type the sequence as, e.g., \0xA3\103. Character resulting in a code 0 (zero) (e.g., \00) is special. It tells: "skip everything what follows me in this string". It does not make sense to use it, since you can always terminate the sequence with a delimiter. When you use an empty string as a matching sequence, remember that it does not match anything.

If the line with entries is too long, you can break it between the fields. If the string is too long to fit a line, you can break it before any nonblank character by the \ (backslash) followed by white space (i.e., new lines, spaces, tabs, etc.). The \ and the following white space will be removed from the string by the string preprocessor. However, you are not allowed to break the individual character codes (and you probably would not do it ever for aestetic purposes). For example:

```
"experi\
mental design"
```

is equivalent to:

```
"experimental design"
```

while:

```
"experimental\
design"
```

is equivalent to:

```
"experimentaldesign"
```

If you need to have \ followed by a space in your string, you need to enter either a backslash or a space following it as an explicit character code, for example:

```
"\\0o40"
```

will produce a \ followed by the space, while the string:

```
"\    "
```

will be empty.

The preprocessor knows only about comments, plain characters, character codes, and continuation lines. However, some characters and their combinations may have a special meaning in lists and regular expressions.

DETAILS OF FILE STRUCTURE

Ad.1) File format number. This is simply a digit 1 on a line by itself at the moment. This entry is included to allow future extensions of the transliteration description file without the need to modify older transliteration descriptions (program will read data according to the current file format number given in the file).

Ad.2) String delimiters. The subsequent 3 lines specify pairs of single character delimiters for 3 types of text data. The line format is:

opening_character   closing_character.

These are needed to mark the beginning/end and the type of the text data. Each string (text datum) is saved starting from the first character after opening delimiter, and ends at the last character before the closing delimiter. If you need to use the closing delimiter within a string, you need to specify it as its code (e.g., if you are using () pair as delimiters, specify ")" as \0x29). The opening delimiter may be the same or different from the closing delimiter.

a) The first line contains characters used to enclose (bracket) a *plain string*. Plain strings are directly matched to input data or directly sent to output. I suggest to stick to " " pair for plain strings. The ASCII code for " is \0d34 = \0x22 = \0o42 if you need it inside the string itself.

b) The second line contains characters to mark the beginning and the end of the *list*. Lists are used to translate single character codes. I suggest [and] delimiters for the list (ASCII code of "]" is: \0d93 = \0x5D = \0o135). The lists may include ranges, for example: [a-zA-Z0-9] will include all Latin letters (small and capital) and digits. Note that order is important: [a-d] is equivalent to [abcd], while [d-a] will result in an error. If you want to include "-" (minus) in the list, you need to place it as the first or the last character. There are only two special characters on the list, the "-" described above, and the "]" character. You need to enter the "]" as its code. E.g., for ASCII character table [*--] is equivalent to [*+,-], is equivalent to [\42\43\44\45]. The order of characters in the list does not matter unless the input list corresponds to the output list (this will be explained later). Empty lists do not make sense.

c) The third line of delimiter specification contains delimiters for *regular expression*s and *substitution expression*s. These strings are used for "flexible" matches to the text in the input file. They are very similar to the ones used in UN*X for searching text in utilities like: grep, sed, vi, awk, etc., though only a subset of full UN*X regular expression syntax is used here. I suggest enclosing them within braces { and } (ASCII code for } is \0d125 = \0x7D = \0o175). Actually, regular expressions can only be used for input sequences, and for output sequences the {} are used to enclose substitution sequences. This will be explained below. The description of the syntax for regular/substitution expressions is adapted from the documentation for the regexp package of Henry Spencer, University of Toronto --- this regular expression package was incorporated, after minute modifications, into the program.

REGULAR EXPRESSION SYNTAX

A regular expression is zero or more branches, separated by ‘|’. It matches anything that matches one of the branches. The ‘|’ simply means "or".

A branch is zero or more pieces, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an atom possibly followed by ‘*’, ‘+’, or ‘?’. An atom followed by ‘*’ matches a sequence of 0 or more matches of the atom. An atom followed by ‘+’ matches a sequence of 1 or more matches of the atom. An atom followed by ‘?’ matches zero or one occurrences of atom.

An atom is a regular expression in parentheses (matching a match for the regular expression), a range (see below), ‘.’ (matching any single character), a ‘\’ followed by a single character (matching that character), or a single character with no other significance (matching that character).

A range is a sequence of characters enclosed in ‘[ ]’. It normally matches any single character from the sequence. If the sequence begins with ‘^’, it matches any single character not from the rest of the sequence. If two characters in the sequence are separated by ‘-’, this is shorthand for the full list of ASCII characters between them (e.g. ‘[0-9]’ matches any decimal digit). To include a literal ‘]’ in the sequence, make it the first character (following a possible ‘^’). To include a literal ‘-’, make it the first or last character. The regular expression can contains subexpressions which are enclosed in a ( ) pair. These subexpressions are numbered 1 to 9 and can be nested. The numbering of subexpressions is given in the order of their opening parentheses "(". For example:

(111)...(22(333)222(444)222)...(555)

Note that expression 2 contains within itself expressions 3 and 4.

These subexpressions can be referenced in the substitution string which is described below in the paragraph below, or can be used to delimit atoms.

Examples:

{[\0d32\0d09]\0d10} --- will match space or tab followed by new line

{[Tt][Ss]} --- will match TS, Ts, tS and ts

{TS | Ts | tS | ts} --- same as above

{[\0d09-\0d15][^hH][^uU][a-zA-Z]*[\0d09-\0d15]} --- all words which do not start with hu, Hu, hU, HU. There is a space between \0d15 and].

Note that specifying expressions like {.*} (i.e., match all characters) does not make much sense, since it would mean here: match the whole input file. However, expressions like {A.*B} should be acceptable, since they match a pair of A and B, and everything in between them, e.g. for a string like: "This is Mr. Allen and this is Mr. Brown." this expression should match the string: "Allen and this is Mr. B".

Remember to put a backslash "\" in front of the following characters: . [( ) | ? + * \ if you want their literal meaning outside the range enclosed in [ ]. Inside the range they have their literal meaning. If you know the syntax of UN*X regular expressions, please note that ^ and $ anchors are not supported and are treated as normal characters (with the exception of ^ negation within [ ]).

SUBSTITUTION EXPRESSIONS

After finding a match for a regular expression in the input text, a substitution is made. It can be a simple substitution where the whole matching string is replaced by another string, or it may reuse a portion or the whole matching string. The subexpressions (the ones enclosed in parentheses) within the regular expression which matched the input text can be referenced in the substitution expression. Only the following characters have special meaning within substitution expression:

& --- will put the whole matching string.
\1 --- will put the match for the 1st subexpression in ( ).
\2 --- will put the string which matched 2nd subexpression, etc.
\9 --- will place in a replacement string the 9th subexpression (provided that there was 9 ( ) pairs in the regular expression)

Only 9 subexpressions are allowed. All other characters and sequences within the substitution expression will be placed in a substitution string as written. To be able to put a single backslash there, you need to put two of them. To be able to place the unchanged codes of the above characters (i.e., to make them literals), you need to precede them with a backslash "\", i.e., to get & in the output string you need to write it as \&. Similarly, to place literal \1, \2, etc., you need to enter it as \\1, \\2, etc. Note that characters .+[]()^, etc. which had a special meaning in the regular expressions, do not have any special meaning in the substitution expression and will be output as written.

Example:

The regular expression:

{([Tt])([Ss])} and the corresponding substitution expression {\1.\2} puts a period between adjoining letters t and s preserving their letter case.

The expression:

{([A-Za-z]+)-[\0x09]*([\0x0A-\0x0D]+)[\0x09]*([A-Za-z,.?;:"\)'‘!]+)[\0x09]}

and the substitution expression {\1\3\2} dehyphenate words (when you understand this one, you are a guru...). For example: con- (NL)cert is changed to concert(NL), where NL stands for New Line. It looks for one or more letters (saves them as substring 1) followed by a hyphen (which may be followed by zero or more spaces or tabs). The hyphen must be followed by a NewLine (ASCII characters 0A-0D hex form various new line sequences) and saves NewLine sequence as a subexpression 2. Then it looks for zero or more tabs and spaces (at the beginning of the line). Then it looks for the rest of the hyphenated word and saves it as substring 3. The word may have punctuation attached. Then it looks again for some spaces or tabs. The substitution expression junks all sequences which were not within (), i.e., hyphen and spaces/tabs and inserts only substrings but in a different order. The \1 (word beginning) is followed by \3 (word end) and followed by the NewLine --- \2. The {\2\1\3} would be probably equally good, though you would need to move the punctuation matching to the beginning of the regular expression.

Ad.3) Starting sequence. This sequence will be sent to the output before any text. It is enclosed in the pair of string delimiters. I use it to output LaTeX preamble. However, it can be empty, if not used. The (sequence) may contain any characters, including new lines, etc.

Example:

```
""          # empty sequence
```

Example:

```
"\documentstyle{article}
\input cyracc
\begin{document}
"
```

is right (note a new line at the end), but

```
"\documentstyle{article}
\input cyracc       # this comment will be included!
\begin{document}"   # while this will not
```

is wrong.

Ad.4) Ending sequence. Similar to 1), but will be appended at the end of the output file.

For example:

```
"\end{document}
"
```

Ad.5) Number of input character sets. For example, in some incarnation of KOI7, there are two character sets: Latin and Cyrillic. Cyrillic character sequence follows SHIFT-OUT character (CTRL-N), \0x0e, and is terminated by SHIFT-IN character (CTRL-O), \0x0f. Another way of looking at it is that Latin characters follow CTRL-O and cyrillic ones follow CTRL-N.

If there is only one character set on input you should specify 0 as a number of input char sets, since the input file obviously does not contain any SHIFT-OUT/IN sequences.

Ad.6) SHIFT-OUT/SHIFT-IN sequences for each input character set. These lines appear only if you specified nonzero number of character sets. These lines contain also "nesting sequences", which will be explained later in this section. You do not use "nesting sequences" frequently, and let us assume for a moment that nesting data are empty strings. The strings or regular expressions specified here are matched with the contents of input text. If match was found, the matching sequence is usually deleted from the input text and:

a) for SHIFT-OUT sequence: the current input character set number is changed to the new one corresponding to the SHIFT-OUT sequence, or

b) for SHIFT-IN sequence: the previous input character set number is restored, (i.e., the one which preceded the SHIFT-OUT sequence for the current set). Note that only the SHIFT-IN sequence for the current set is matched. The SHIFT-IN sequences for other character sets than the current set are not matched. The bracketing of sets is assumed perfect. If the SHIFT-IN sequence for the current set is an empty string, the input set number is changed when SHIFT-OUT sequence of the new set is detected.

For each input character set, you have to specify a line consisting of 6 strings/expressions separated by spaces:

SO-match SO-subs NEST-up NEST-down SI-match SI-subs

where:

SO-match --- the string or regular expression for the SHIFT-OUT sequence for the current character set. If detected, the input character set is changed to this set.

SO-subs --- this is usually an empty string (i.e., the input sequence matching SO-match is removed). But it can be a replacement string or a substitution expression, which will substitute the original matching SHIFT-OUT sequence.

NEST-up --- this string (or a regular expression) is usually an empty string). However, it can be used to count brackets for detection of SHIFT-IN bracket, if SHIFT-IN sequence is not unique. Its use is explained below.

NEST-down --- a counterpart of NEST-up. It is explained later.

SI-match --- when a sequence in an input file matches the string or regular expression given as SI-match for a current input character set, the input character set number is restored to the previous set. Note, that only SI-match for a current set is matched with input characters.

SI-subs --- this is usually an empty string (i.e., input sequence which matched SI-match is removed), but if it is not, the input characters which matched the SI-match are replaced with the SI-subs.

The KOI7 case described above may be specified as:

```
2                       # 2 input sets
""       ""       ""       ""       ""       ""       # Latin(set 1)
"\016"   ""       ""       ""       "\017"   ""       # Cyrillic(set 2)
         or
2                       # 2 sets
"\017"   ""       ""       ""       ""       ""       # Latin(set 1)
"\016"   ""       ""       ""       ""       ""       # Cyrillic(set 2)
```

Before the input is processed, the program is initialized to the character set of the first set. In the above case, it is important, since declaration:

```
2                       # 2 sets
"\016"   ""       ""       ""       ""       ""       # Cyrillic(set 1)
"\017"   ""       ""       ""       ""       ""       # Latin(set 2)
```

would be wrong and would mess up the Latin characters preceding first Cyrillic sequence.

The nesting sequences are used only for specific situations. I needed them to write a transliteration table from LaTeX to KOI8. In LaTeX the { } pair is used for grouping and appears frequently in the text. The sequence of cyrillic characters is also a group in LaTeX. The SHIFT-OUT sequence for Russian letters in LaTeX is (at least in my case): "{\cyr ", and the end of the Russian letters is marked by "}", but the "}" has to be the bracket matching the opening "{" in "{\cyr ", not just any bracket. For this reason, my SHIFT-OUT/IN entry was in this case:

```
"{\cyr "  ""  "{"  "}"  "}"  ""   # Cyrillic codes
```

Whenever the "{\cyr " was found, the program zeroes the counter. It adds +1 to it, when NEST-up sequence (i.e., the "{" here) is found, and subtracts 1 from it, when the NEST-down sequence is found (i.e., the "}"). The checking for a SHIFT-IN sequence (i.e., the "}") for cyrillic set is done only when the counter value is zero (i.e., all pairs inside the cyrillic text are matched. In fact, the process is more complicated than that (the counter for an opened character set is placed on the stack), but these are details you can find in the code itself.

What if the SHIFT-IN and SHIFT-OUT sequence is the same character? Starting from version 1.01 the TRANSLIT will also work in such cases. Let us assume that the SHIFT-IN and SHIFT-OUT sequence is a single character "%" which switches between two character sets. Also, if we want to use it in the text, we have to double it, i.e., "%%" will not be a SHIFT-IN/OUT sequence but will denote a literal percent sign. We can do it in the following way:

```
""        ""     ""  ""  ""        ""       # Latin letters
{%([^%])}  { \ 1}  ""  ""  {%([^%])} { \ 1}  # Cyrillic codes
```

and later in the transliteration table (see below) we should put a line:

```
0    "%%"      0    "%"        # change doubled % to a single one
```

The same effect, for identical SHIFT-IN/OUT sequences, can be accomplished with a -3 character set code and will be described below.

Ad.7) Number of output "character sets". This is analogous to the input case. The characters sent to output may belong to different sets. For example, when the character (or the sequence) from set 2 is followed by the character (or the sequence) from set 1, the program first sends the SHIFT-IN sequence for set 2 (if it is not empty) and then the SHIFT-OUT sequence for set 1 (if it is not empty). If the output character (or sequence) is assigned to set 0, then no SHIFT-IN/SHIFT-OUT sequences are sent to output.

If there is only one set of output characters, you should specify 0. Note that you may have several input sets and several output sets, though this is rare. Usually, you have one input set and many output character sets, or vice versa. Again, if you have only one output set, you do not have any SHIFT-IN/SHIFT-OUT sequences, since those are send to output only when a set number is changed. But you are free to experiment.

Ad.8) SHIFT-OUT/SHIFT-IN sequences for each output character set. It is similar to the input case, however, the NEST-in and NEST-up sequences are not used here. Again, before any text is sent to output, the character set specified as the first one is assumed. If SHIFT-OUT/IN sequences are not used (i.e., you have only one output character set), you will not have any SHIFT-OUT/SHIFT-IN data lines. The KOI8 (single character set containing all Latin and Russian letters) to KOI7 (the set using overlapping codes switched by SHIFT-OUT/IN sequences) conversion could be therefore accomplished by the following table:

```
2                        # 2 output sets
""          ""           # Latin Letters
"\016"      "\017"       # Russian Letters case
```

Ad.9) Transliteration table for individual character or their sequences. It is a core of your transliteration data. There are 4 columns in the transliteration table:

(inp_set_no) (inp_seq) (out_set_no) (out_seq)

These 4 columns are separated by spaces. The (input_set_number) corresponds to the input character set number as specified above for input SHIFT-OUT/SHIFT-IN data, or zero. If zero is used (even if number of input sets is not zero), the (input_sequence) will be always matched, irrespectively of the current input character set imposed by the SHIFT-OUT sequence. This is useful, since some characters are universal (e.g., new lines, spaces, pluses, minuses, etc.) irrespectively of the current character set. The (input_sequence) is the sequence of characters to be matched with characters in the input file, and if found (within the character set specified) it is replaced by the (output_sequence) and sent to output (i.e., the matching is interrupted, the (output_sequence) sent to ouput, the input file pointer is moved to the first character after the matched sequence and matching resumes). The (output_set_number) specifies the output character set. When the output character set changes during transliteration, the appropriate SHIFT-IN sequence of the previous set and the current set's SHIFT-OUT sequence is sent to output. The (output_set_number) may also be zero (even if number of output sets is not zero). In this case, the current output set status is not changed, and no SHIFT-IN/OUT sequences is sent to output. Lastly, the output set code may be -1, -2 or -3. In this case, the substitution is performed within input string that matched but the output sequence is not sent to the output yet. Depending on the code, the following action is performed:

- -1 --- program makes the substitution in the input string (i.e., substitutes the matching string with the input string in the input buffer). It does not send the output sequence to the output, but continues matching input sequences following the currently matched one.
- -2 --- like code -1, but matching is resumed from the first sequence on the list.
- -3 --- like code -1, but matching is resumed from the input SHIFT-OUT/IN sequences.

E.g., if the unprocessed text in the input file is:

mental procedure was not successful since..........

and there was a line in transliteration table:

```
0  "me"   -1  "you"
```

the input text would be changed to:

yountal procedure was not successful since..........

and all remaining matching data would be applied to this text, rather than original text. The -2 code backsteps to the point where the matching of transliteration starts. The -3 code backsteps even further, to the point where the input SHIFT-OUT and SHIFT-IN sequences are matched. Since the order of sequences to match is crucial here, for the case of output set code -1/-2/-3 even one-character input sequences are matched in the order specified. BE CAREFUL HERE. You may create infinite loops. If you use code -2/-3, be sure that the resulting sequence after substitution with the code -2/-3, will not match previous sequences with codes -2/-3.

The (output_sequence) is a sequence which substitutes the corresponding (input_sequence). If (output_sequence) is "" (i.e., empty string) then (input_sequence) is effectively deleted. The (input_sequence)s are compared with input in the order specified unless backstepping -2/-3 code is used (the matching is done from the first sequence again). I use the code -1 e.g., to dehyphenate words when changing to LaTeX. Code -2 is useful if you want to skip next comparisons, and the resulting substitution string will match earlier matching expressions. I do not see many uses for the code -3, but it can be used to resolve "toggle" SHIFT-IN/OUT sequence, as described in an example further below. The order for multicharacter sequences is therefore important (the single character sequences are always compared after all multicharacter sequences, and can be therefore put anywhere). The longer multicharacter sequences should be specified before shorter ones, unless they are some "preprocessing" steps with codes -1/-2/-3. The order may sometimes be crucial. If you need single character sequences matched in a specific order, enter them as regular expressions, i.e., as {c} instead of "c". In short, the multicharacter input sequences and regular expressions are matched to input text in the order specified. For the sake of efficiency, the single character input sequences (with exception of output set code -1/-2/-3) and input lists are handled as a case of remapping and are matched in the order of character codes associated with them. If you specify the same single input character twice for a given input set, the program will complain. The following combinations of input and output sequences are allowed:

Input Sequence	Output Sequence
"*plain string*"	only "*plain string*"
[*list*]	[*list*] or "*plain string*"
{*regular expression*}	{*substitution expression*} or "*plain string*"

When match is found, the matching sequence is removed and substituted with an output sequence. If this results is changing the current output character set, the appropriate SHIFT-IN/SHIFT-OUT pair is sent to the output before the transliterated output sequence. If list is used as the input sequence, you may either use:

a) plain string as output sequence. In this case, if current input character belongs to the input list, it is replaced by the output string. I use it to delete ranges of characters which do not have any corresponding characters in the output set (e.g., some graphics characters). In this case, the order of characters on the input list is not important.

b) if the output string is also a list then it has to contain exactly the same number of characters as the input list. In this case, the 1st character from the input list is replaced by the 1st character from the output list, the 2nd one by the 2nd one, etc. Therefore, the order of characters is important.

Theoretically, if there is one-to-one correspondence between characters in the input set and characters in the output set, you can make the conversion by using a single line consisting of two lists. But it looks ugly... And is difficult to read. And for the program, the substitution takes the same time, if the characters are specified separately, or when they are specified as matching lists. If regular expression is used to match the input characters, the matching sequence may be replaced by a plain string or a substitution string, which was described above.

Examples:

```
2          "CCCP"          0               ""
```

will delete all occurrences of CCCP from the input file (but not Cccp or CCCp) for input set 2.

```
0          "\0xD1"          0          "ya"
```

will replace all occurrences of character of the code \0xD1 with a two letter sequence "ya".

```
0          \0xD1            2          q
```

will replace all characters \0xD1 with a character "q" and output SHIFT-IN/OUT sequence if necessary.

```
2          "q"              0          "\0xD1"
```

will replace letter q (if the current input set is 2) with a code \0xD1.

```
0          "\0xD1"          2          "ya"
```

will replace code \0xD1 with a sequence ya (assuming that SHIFT-OUT and SHIFT-IN sequences for output set 2 are: {\cyr and }, respectively, you will get {\cyr ya}).

If a character is not specified in the transliteration table, it will be output as is, i.e., it corresponds to a line:

```
0          "c"              0          "c"
```

where c is the character. If you want to delete certain characters, you need to explicitly specify this, e.g.:

```
0          [a-z]            0          ""
```

will delete all lower case Latin letters from the text.
Below is an example of solving the identical SHIFT-IN/OUT sequences problem using character set code -3 which I promissed above. Assume, that you have 2 character sets in the input file, but switching between them is accomplished by a "toggle" character. That is, if the toggle character is found, you should switch to the other set. Also, if you want to use the toggle character in the set, you need to double it. Let also assume that we have 2 character codes which will never, ever appear. We can fool the translit by changing toggle character to a unique character and backstepping with character code -3 to check for SHIFT-IN/OUT sequences again. Let the % sign be a toggle character, and that we have two codes (for example codes \ 254 and \ 255) which will never appear in our text. The appropriate entries in the transliteration table may look like:

```
1   {%([ˆ%])}    -3    { \254 \ 1}
2   {%([ˆ%])}    -3    { \255 \ 1}
0   "%%"         0     "%"
```

i.e., when the single % is seen in set 1, produce SHIFT-OUT sequence for set 2; and when a single % is seen in set 2, produce SHIFT-IN sequence for set 1. The appropriate input character set definitions will be:

```
2              # number of input character sets
" \ 255"   ""    ""    ""    ""  ""
" \ 254"   ""    ""    ""    ""  ""
```

However, be warned. I never tried this. If this trick does not work, please let me know.

Before you decide to create your own transliteration file, please examine existing transliteration files. Do yourself (and others) a favor --- put as many comments as possible there. If you allow others to use your transliteration files, please include your name and e-mail address and file creation date.

Program matches the sequences in a specific order:

1) if NEST counter is zero, Match/substitute current set SHIFT-IN sequence

2) If matched, restore previous set number
3) If matched, restore previous set nest counter
4) Match/substitute input SHIFT-OUT sequences
5) If matched, save current set and start new one
6) If matched, zero nest counter for NEST sequences
7) Match/substitute transliteration sequences
8) If matched and code = -1 make substitution in input buffer and continue matching the next sequence.
9) If matched and code = -2 make substitution and goto 7)
10) If matched and code = -3 make substitution and goto 1)
11) Match (no substitution) NEST-up and NEST-down to input buffer
12) If NEST-up matched, increment counter for current set
13) If NEST-down matched, decrement counter for current set
14) If match in 7) send substitute sequence to output
15) If no match in 7) (or code -1) output current input character
16) Advance input pointer to point at new characters
17) If End of File, break
18) Goto 1)

ASCII CHARACTER CODES

dec	hx	oct	ch		dec	hx	oct	ch
0	00	000	^@	NUL	64	40	100	@
1	01	001	^A	SOH	65	41	101	A
2	02	002	^B	STX	66	42	102	B
3	03	003	^C	ETX	67	43	103	C
4	04	004	^D	EOT	68	44	104	D
5	05	005	^E	ENQ	69	45	105	E
6	06	006	^F	ACK	70	46	106	F
7	07	007	^G	BEL	71	47	107	G
8	08	010	^H	BS	72	48	110	H
9	09	011	^I	HT	73	49	111	I
10	0a	012	^J	LF	74	4a	112	J
11	0b	013	^K	VT	75	4b	113	K
12	0c	014	^L	FF	76	4c	114	L
13	0d	015	^M	CR	77	4d	115	M
14	0e	016	^N	SO	78	4e	116	N
15	0f	017	^O	SI	79	4f	117	O
16	10	020	^P	DLE	80	50	120	P
17	11	021	^Q	DC1	81	51	121	Q
18	12	022	^R	DC2	82	52	122	R
19	13	023	^S	DC3	83	53	123	S
20	14	024	^T	DC4	84	54	124	T
21	15	025	^U	NAK	85	55	125	U
22	16	026	^V	SYN	86	56	126	V
23	17	027	^W	ETB	87	57	127	W
24	18	030	^X	CAN	88	58	130	X
25	19	031	^Y	EM	89	59	131	Y
26	1a	032	^Z	SUB	90	5a	132	Z
27	1b	033	^[	ESC	91	5b	133	[
28	1c	034	^\	FS	92	5c	134	\
29	1d	035	^]	GS	93	5d	135	]
30	1e	036	^^	RS	94	5e	136	^

31	1f	037	ˆ_	US	95	5f	137	_
32	20	040		SP	96	60	140	‘
33	21	041	!		97	61	141	a
34	22	042	"		98	62	142	b
35	23	043	#		99	63	143	c
36	24	044	$		100	64	144	d
37	25	045	%		101	65	145	e
38	26	046	&		102	66	146	f
39	27	047	’		103	67	147	g
40	28	050	(		104	68	150	h
41	29	051	)		105	69	151	i
42	2a	052	*		106	6a	152	j
43	2b	053	+		107	6b	153	k
44	2c	054	,		108	6c	154	l
45	2d	055	-		109	6d	155	m
46	2e	056	.		110	6e	156	n
47	2f	057	/		111	6f	157	o
48	30	060	0		112	70	160	p
49	31	061	1		113	71	161	q
50	32	062	2		114	72	162	r
51	33	063	3		115	73	163	s
52	34	064	4		116	74	164	t
53	35	065	5		117	75	165	u
54	36	066	6		118	76	166	v
55	37	067	7		119	77	167	w
56	38	070	8		120	78	170	x
57	39	071	9		121	79	171	y
58	3a	072	:		122	7a	172	z
59	3b	073	;		123	7b	173	{
60	3c	074	<		124	7c	174	|
61	3d	075	=		125	7d	175	}
62	3e	076	>		126	7e	176	˜
63	3f	077	?		127	7f	177	DEL

CONVERSION: DECIMAL<-->OCTAL<-->HEX.

000	000	00	064	100	40	128	200	80	192	300	C0
001	001	01	065	101	41	129	201	81	193	301	C1
002	002	02	066	102	42	130	202	82	194	302	C2
003	003	03	067	103	43	131	203	83	195	303	C3
004	004	04	068	104	44	132	204	84	196	304	C4
005	005	05	069	105	45	133	205	85	197	305	C5
006	006	06	070	106	46	134	206	86	198	306	C6
007	007	07	071	107	47	135	207	87	199	307	C7
008	010	08	072	110	48	136	210	88	200	310	C8
009	011	09	073	111	49	137	211	89	201	311	C9
010	012	0A	074	112	4A	138	212	8A	202	312	CA
011	013	0B	075	113	4B	139	213	8B	203	313	CB
012	014	0C	076	114	4C	140	214	8C	204	314	CC
013	015	0D	077	115	4D	141	215	8D	205	315	CD
014	016	0E	078	116	4E	142	216	8E	206	316	CE
015	017	0F	079	117	4F	143	217	8F	207	317	CF
016	020	10	080	120	50	144	220	90	208	320	D0

017	021	11	081	121	51	145	221	91	209	321	D1
018	022	12	082	122	52	146	222	92	210	322	D2
019	023	13	083	123	53	147	223	93	211	323	D3
020	024	14	084	124	54	148	224	94	212	324	D4
021	025	15	085	125	55	149	225	95	213	325	D5
022	026	16	086	126	56	150	226	96	214	326	D6
023	027	17	087	127	57	151	227	97	215	327	D7
024	030	18	088	130	58	152	230	98	216	330	D8
025	031	19	089	131	59	153	231	99	217	331	D9
026	032	1A	090	132	5A	154	232	9A	218	332	DA
027	033	1B	091	133	5B	155	233	9B	219	333	DB
028	034	1C	092	134	5C	156	234	9C	220	334	DC
029	035	1D	093	135	5D	157	235	9D	221	335	DD
030	036	1E	094	136	5E	158	236	9E	222	336	DE
031	037	1F	095	137	5F	159	237	9F	223	337	DF
032	040	20	096	140	60	160	240	A0	224	340	E0
033	041	21	097	141	61	161	241	A1	225	341	E1
034	042	22	098	142	62	162	242	A2	226	342	E2
035	043	23	099	143	63	163	243	A3	227	343	E3
036	044	24	100	144	64	164	244	A4	228	344	E4
037	045	25	101	145	65	165	245	A5	229	345	E5
038	046	26	102	146	66	166	246	A6	230	346	E6
039	047	27	103	147	67	167	247	A7	231	347	E7
040	050	28	104	150	68	168	250	A8	232	350	E8
041	051	29	105	151	69	169	251	A9	233	351	E9
042	052	2A	106	152	6A	170	252	AA	234	352	EA
043	053	2B	107	153	6B	171	253	AB	235	353	EB
044	054	2C	108	154	6C	172	254	AC	236	354	EC
045	055	2D	109	155	6D	173	255	AD	237	355	ED
046	056	2E	110	156	6E	174	256	AE	238	356	EE
047	057	2F	111	157	6F	175	257	AF	239	357	EF
048	060	30	112	160	70	176	260	B0	240	360	F0
049	061	31	113	161	71	177	261	B1	241	361	F1
050	062	32	114	162	72	178	262	B2	242	362	F2
051	063	33	115	163	73	179	263	B3	243	363	F3
052	064	34	116	164	74	180	264	B4	244	364	F4
053	065	35	117	165	75	181	265	B5	245	365	F5
054	066	36	118	166	76	182	266	B6	246	366	F6
055	067	37	119	167	77	183	267	B7	247	367	F7
056	070	38	120	170	78	184	270	B8	248	370	F8
057	071	39	121	171	79	185	271	B9	249	371	F9
058	072	3A	122	172	7A	186	272	BA	250	372	FA
059	073	3B	123	173	7B	187	273	BB	251	373	FB
060	074	3C	124	174	7C	188	274	BC	252	374	FC
061	075	3D	125	175	7D	189	275	BD	253	375	FD
062	076	3E	126	176	7E	190	276	BE	254	376	FE
063	077	3F	127	177	7F	191	277	BF	255	377	FF

INSTALLATION

Program is given in a source form. It was tried under UN∗X, VMS and MS-DOS systems and ran. The file *readme.doc* contains the details on how to obtain the whole package. You can retrieve this file from anonymous ftp on kekule.osc.edu in the directory /pub/russian/translit. You can also obtain it via e-mail by sending a message:

```
get translit/readme.doc from russian
```

to OSCPOST@osc.edu or OSCPOST@OHSTPY.BITNET.

The source of the program consists of several files:

paths.h must be edited before compilation. It contains its own comments what to do. The defines in this file relate to the operating system you are using and the default path for searching transliteration table.

translit.c
It contains the main program. This was intended to be a portable code.

reg_exp.h
the include file for regular expression matching library of Henry Spencer from the University of Toronto. This regular expression package was posted to comp.sources.misc (volume 3). Also 4 patches were posted (in volumes: 3, 4, 4, 10). I applied the patches to the original code and made small modifications to the code, which are marked in the source code.

reg_exp.c
the regular expression library for compilation and matching of regular expressions.

reg_sub.c
the regular expression substitution routine.

Before you compile this program you have to edit *paths.h*. Read comments in the file. During compilation, all source code should reside in the current directory.
Then you may compile the program under UN∗X as (for example):

```
cc -o translit translit.c reg_exp.c reg_sub.c
```

and copy the program *translit* to some standard directory which is in users' path (for example: /usr/local/bin). Then you need to copy transliteration tables to the directory which you have chosen in *paths.h*. If you get errors, then it is not OK. Please, report them to the author (with all the gory details: error message, line number, machine, operating system, etc.).

Under VMS (VAXes) you need to compile it as:

```
cc translit
cc reg_exp
cc reg_sub
link translit+reg_exp+reg_sub,sys$library:vaxcrtl/lib
```

and before you can use the program, you need to type (or better put into your LOGIN.COM file) a line:

```
translit == "$SYS$USER:[ME.TRA]TRANSLIT.EXE"
```

or whatever is the full path to the *translit* executable image which you created with LINK. Note the quotes and the $ sign in front of program path.

On an IBM-PC I used MicroSoft C 5.1 as:

```
cl /FeTRANSLIT /AL /FPc /W1 /F 5000 /Ox /Gs translit.c reg_exp.c reg_sub.c
```

RULES, CONDITIONS AND AUTHOR'S WHISHES

You can distribute this code and associated files under these conditions:

1) You will distribute all files (even if you think that they are garbage). You may get the complete set from anonymous ftp at kekule.osc.edu in /pub/russian/translit. You can also get the program and associated files via e-mail. To get the instructions for e-mail distribution send a line:

```
send translit/readme.doc from russian
```

to OSCPOST@osc.edu or OSCPOST@OHSTPY.BITNET. You are not allowed to distribute the incomplete distribution. The following files should be present in the distribution:

alt-gos.rus - ALT to GOSTCII table
alt-koi8.rus - ALT to KOI8 table
example.alt.uu - uuencoded example in ALT
example.ko8.uu - uuencoded example in KOI8
example.pho - phonetic transliteration example
example.tex - LaTeX example
gos-alt.rus - GOSTCII to ALT table
gos-koi8.rus - GOSTCII to KOI8 table
koi7-8.rus - KOI7 to KOI8 table
koi7nl-8.rus - KOI7 (no Latin) to KOI8 table
koi8-7.rus - KOI8 to KOI7 table
koi8-alt.rus - KOI8 to ALT table
koi8-gos.rus - KOI8 to GOSTCII table
koi8-lc.rus - KOI8 to Library of Congress table
koi8-phg.rus - KOI8 to GOST transliteration
koi8-php.rus - KOI8 to Pokrovsky transliteration
koi8-ltx.rus - KOI8 to LaTeX conversion
koi8-tex.rus - KOI8 to Plain TeX conversion
order.txt - Order form for ordering the program
paths.h - Include file for translit.c
phg-koi8.rus - GOST transliteration to KOI8
pho-8sim.rus - Simple phonetic to KOI8
pho-koi8.rus - Various phonetic to KOI8
php-koi8.rus - Pokrovsky to KOI8
readme.doc - short description of the files
reg_exp.c - regular expression code by Henry Spencer
reg_exp.h - include for reg_exp.c and reg_sub.c
reg_sub.c - regular expression code by H. Spencer
ltx-koi8.rus - LaTeX to KOI8
translit.c - TRANSLIT main program
translit.ps - TRANSLIT manual in PostScript
translit.1 - TRANSLIT manual in ∗roff
translit.txt - Plain ASCII TRANSLIT manual

2) You may expand/change the files and the program and distribute modified files, provided that you do not delete anything (you can always comment the unnecessary portions out) and clearly mark your changes. Please send the copy of the modified version to the author, though you are not required to do so. I will give you all the credit for your enhancements. I simply wish that there is a single point of distribution for this code, so it is maintained to some extent. If you create additional transliteration definition files, please, send them to the author if you may. I will add them to the program distribution. I want to fix bugs and expand/optimize this code, but I need your help. I need your transliteration files for languages which I do not know or do not use currently. Your suggestions for improving documentation are most welcome (I am not a native English speaker).

3) You will not charge money for the program and/or associated files, except for media and copying costs. If you want to sell it, contact the author first. Bear in mind that the regular expression package by Henry Spencer has some copyright restrictions. But there are other regular expression packages which do not have these restrictions (which are not violated by this offering).

4) I will gladly help you with advice on compiling this software and try to fix bugs when time allows. However, if you want a ready to run executable, you need to order it for a very nominal fee from *JKL ENTERPRISES, INC.* as described in the file *order.txt* which must be a part

of a complete distribution.

AUTHOR

Jan Labanowski, P.O. Box 21821, Columbus, OH 43221-0821, USA. E-mail: jkl@osc.edu, JKL@OHSTPY.BITNET.