GNU recode, version 3.3

The character set transliterator
Edition 3.3, December 1993

by Francois Pinard

1 What is the purpose of this program

This `recode` program has the purpose of converting files between various character sets and usages. When exact transliterations are not possible, as it is often the case, the program may get rid of the offending characters or fall back on approximations.

Let us coin the term *charset* to represent, without distinction, a character set "per se" or a particular usage of a character set. This program recognizes or produces around 150 such charsets. Since it can convert each charset to almost any other one, many thousands of different conversions are possible.

This tool pays special attention to superimposition of diacritics for French representation. This orientation is mostly historical, it does not impair the usefulness, generality or extensibility of the program.

1.1 Overview of charsets

Recoding is currently possible between most of the charsets described in RFC 1435. See Chapter 3 [RFC 1345 charsets], page 9.

Recode also handles some charsets in more specialized ways. These are:

- usual 7-bit ASCII: without any diacritics, or else: using backspace for overstriking; Unisys' ICON convention; TEX/LATEX coding; easy French conventions for electronic mail;
- 8-bit extensions to ASCII: ISO Latin-1, Atari ST code, IBM's code for the PC, Apple's code for the Macintosh, NeXTSTEP code;
- 6-bit escaped ASCII based on CDC display code: 6/12 code from NOS; bang-bang code from Université de Montréal;
- non-ASCII codes: three flavors of EBCDIC.

The recent introduction of RFC 1345 in GNU `recode` has brought with it a few charsets having the functionnality of older ones, but yet being different in subtle ways. The effects have not been fully investigated yet, so for now, clashes are avoided, the old and new charsets are kept well separate. For example, wizards would be interested in comparing the output of these two commands:

```
recode -vh ibmpc:applemac
recode -vh ibm437:macintosh
```

The first command uses only charsets prior to RFC 1345 introduction. Both methods give different recodings, the first also properly recodes end of lines. These differences are annoying, the fuziness will have to be explained and settle down one day.

1.2 Contributions and bug reports

Even being the `recode` author and current maintainer, I am no specialist in charset standards. I only made `recode` along the years to solve my own needs, but felt it was extendable for the needs of others. Some GNU people liked the program structure and suggested to make it more widely available. I rely on GNU users judgement for what is best to be done next.

Properly protecting GNU `recode` about possible copyright fights is a pain for me and for contributors, but we cannot avoid addressing the issue in the long run. Besides, the Free Software Foundation, which mandates the GNU project, is very sensible to this matter. GNU standards require that I be cautious before looking at copyrighted code. The safest and simplest way for me is to gather ideas and reprogram them anew, even if this might slow me down considerably. For contributions going beyond a few lines of code here and there, the FSF definitely requires employer disclaimers and copyright assignments.

Many users contributed to GNU `recode` already, I am grateful to them for their interest and involvement. Some suggestions can be integrated quickly while some others have to be delayed, I have to draw a line somewhere when time comes to make a new release, about what would go in it and what would go in the next. Also, when you contribute something to `recode`, *please* explain what it is about. Do not take for granted that I know those charsets which are familiar to you. Your explanations could well find their way into this documentation, too.

Mail suggestions, documentation errors and bug reports to `bug-gnu-utils@prep.ai.mit.edu` or, if you prefer, directly to Francois Pinard '`pinard@iro.umontreal.ca`'. Do not be afraid to report details, because this program is the mere aggregation of hundreds of details.

2 How to use this program

The general format of the program call is one of:

```
recode [option]... [charset]
recode [option]... [before]:[after] [file]...
```

The second form is the common case. Each file *file* will be read assuming it is coded with charset *before*, it will be recoded over itself so to use the charset *after*. If there is no such *file*, the program rather acts as a filter and recode standard input to standard output.

The available options are:

`-C`
`--copyright`
: Given this option, all other parameters and options are ignored. The program prints briefly the Copyright and copying conditions. See the file '`COPYING`' in the distribution for full statement of the Copyright and copying conditions.

`-a`
`--auto-check`
: In this special mode, `recode` ignore arguments and most options. It diagnostics itself by analysing connectivity of the various charsets, reporting on standard output, then it exits without recoding any file.

  For each possible pair of different charsets, it prints on standard output how many single steps are needed for achieving the recoding and how many can be saved by step merging. If a recoding cannot be done, the word '`UNACHIEVABLE`' is printed instead. However, this special line is completely suppressed if option `-x` specified some charset to ignore.

  The option `-h`*name* affects the resulting output, because there are more merging rules when this option is in effect. Other options affect the result: `-d`, `-g` and, notably, `-s`.

  There was a time, in GNU `recode` development, when this option was reasonnably interesting. With the greater number of handled charsets, it became *very* slow, while generating a great deal of output. It can be made slightly more practical with `-x.`, which effectively disable most RFC 1345 charsets from the report.

`-c`
`--colons`
: With `texte` Easy French conventions, use the column `:` instead of the double-quote `"` for marking diaeresis. See Section 8.2 [texte], page 31.

`-d`
`--diacritics`

While converting to or from `latex` charset, limit conversion to diacritics only. This is particularily useful when people write what would be valid TEX or LATEX files, if only they were using TEX macros for applying diacritics instead of using the diacriticized characters directly from the underlying character set.

While converting to `latex` charset, this option assumes that all special characters to TEX or LATEX are properly escaped already; backslashes are also transmitted litterally. While converting the other way, this option prevents all attempts at recognizing TEX or LATEX escaped representation of single characters of the other charset. See Section 8.1 [latex], page 31.

`-f`

`--force` This option *will* is necessary for a file to be transformed irreversibly, regardless of the fact a file is recoded over itself or produced on standard output. Beware that in this `recode` version, this option is only recognized, but otherwise ignored: *if it is found that the recoding is not fully reversible, the file replacement is still unconditionnaly done.*

Even if GNU `recode` tries hard at keeping the recodings reversible, it cannot make any promise! In particular, consider:

- Some transformations are known to be fully reversible for all inputs: `recode` seeks for them (also see option `-s`). This is not true for all transformations, however.
- Usually, reversibility depends on file contents and cannot be told beforehand. Further, reversibility is never absolute accross successive versions of the program. Even correcting a small bug in a mapping could induce slight discrepancies later: please keep only reasonnable expectations about reverse recodings.
- Reversibility is easily lost by merging. This is best explained through an example. If you reversibly recode a file from charset '`A`' to charset '`B`', then you reversibly recode the result from charset '`B`' to charset '`C`', you cannot expect to recover the original file by merely recoding from charset '`C`' directly to charset '`A`'. You will instead have to recode from charset '`C`' back to charset '`B`', and only then from charset '`B`' to charset '`A`'.
- Faulty files create a particular problem. Consider an example, recoding from `ibmpc` to `latin1`. End of lines are represented as '`\r\n`' is `ibmpc` and as '`\n`' in latin1. There is no way by which a faulty `ibmpc` file containing a '`\n`' not preceeded by '`\r`' be translated into a `latin1` file, and then back.
- There is another difficulty arising from code equivalences. For example, in a `latex` charset file, the string '`\^\i{}`' could be recoded back and forth though another charset and become '`\^{\i}`'. Even if the resulting file is equivalent to the original one, it is not identical.

`-g`
`--graphics`

This option is only meaningful while getting *out* of the `ibmpc` charset. In this charset, characters 176 to 223 are used for constructing rulers and boxes, using simple or double horizontal or vertical lines. This option forces the automatic selection of ASCII characters for approximating these rulers and boxes, at cost of making the transformation irreversible.

`-h`[*name*]
`--header`[=*name*]

Instead of recoding files, `recode` writes a C source file on standard output and exits. This source is meant to be included in a regular C program: its purpose is to declare and initialize an array, named *name*, which represents the requested recoding. If *name* is not specified, then it defaults to *before*`_to_`*after*, where *before* is the starting charset and *after* is the goal charset.

Even if `recode` tries its best, this option does not always succeed in producing the requested C table. It will however, provided the recoding can be internally represented by only one step after the optimization phase, and if this merged step conveys a one-to-one or a one-to-many explicit table. But this is all fairly technical. Better try and see!

Beware that other options might affect the produced C tables, these are: `-d`, `-g` and, particularily, `-s`.

`-i`
`--sequence=files`

When the recoding requires a combination of two or more elementary recoding steps, this option forces many passes over the data, using intermediate files between passes. This is the default behaviour when files are recoded over themselves. If this option is selected in filter mode, that is, when the program reads standard input and writes standard output, it might take longer for programs further down the pipe chain to start receiving some recoded data.

`-l`[*format*]
`--list`[=*format*]

This option asks for information about all charsets, or about one particular charset. No file will be recoded.

If there is no non-option arguments, `recode` ignores the *format* value of the option, it writes a sorted list of charset names on standard output, one per line. When a charset name have aliases or synonyms, they follow the true charset name on its line, presented in lexicographical order from left to right. This list is over one hundred lines. It is best used with `grep`, as in:

```
recode -l | grep greek
```

There might be one non-option argument, in which case it is interpreted as a charset name, possibly abbreviated to any non ambiguous prefix. This particular usage of the `-l` option is obeyed *only* for charsets having an RFC 1345 style internal description. Even if most charsets have this property, some do not, then option `-l` cannot be used to detail these particular charsets. For knowing if a particular charset can be listed this way, you should merely try and see if this works. The *format* value of the option can be any of:

`decimal` This format asks for the production on standard output of a concise tabular display of the charset, in which character code values are expressed in decimal.

`octal` This format uses octal instead of decimal in the concise tabular display of the charset.

`hexadecimal`
This format uses hexadecimal instead of decimal in the concise tabular display of the charset.

`full` This format requests an extensive display of the charset on standard output, using one line per character showing its decimal, hexadecimal and octal code values, and also a descriptive comment which is indeed the 10646 character name.

When option `-l` is used together with a *charset* argument, the *format* defaults to `decimal`.

`-o`
`--sequence=popen`
When the recoding requires a combination of two or more elementary recoding steps, this option forces the creation of a chain of program instances initiated through the `popen(3)` library call, all operating in parallel. In filter mode, at cost of some overhead, recoded data will be available soon after the program starts, even if many elementary recoding steps are required.

If, at installation time, the `popen(3)` call is said to be unavailable, selecting option `-o` is equivalent to selecting option `-i`.

`-p`
`--sequence=pipe`
When the recoding requires a combination of two or more elementary recoding steps, this option forces the program to fork itself into a few copies interconnected with pipes, using the `pipe(2)` system call. All copies of the program operate in parallel. This method is similar to the method used through option `-o`, but is slightly more efficient. This is the default behaviour in filter mode. If this option is used when files are recoded over themselves, this should save some disk space, at cost of more system overhead.

If, at installation time, the `pipe(2)` call is said to be unavailable, selecting option `-p` is equivalent to selecting option `-o`. If both `pipe(2)` and `popen(3)` are unavailable, selecting option `-p` is equivalent to selecting option `-i`.

`-s`

`--strict` By using this option, the user requests that `recode` be very strict while recoding a file, merely loosing in the transformation any character which is not explicitely mapped from a charset to another. This option renders the recoding less likely reversible, so it also implies option `-f`.

When this option is not used, `recode` automatically tries to fill mappings with inventend correspondances, making them fully reversible in many instances. This filling is not made at random: the algorithm tries to stick to the identity mapping and, when not possible, prefer small permutation cycles. This means that, by default, `recode` may sometimes produce *funny* characters, however these are quite helpful when one changes his/her mind and wants to revert to the prior recoding.

`-t`

`--touch` The *touch* option is meaningful only when files are recoded over themselves. Without it, the timestamps associated with files are preserved, to reflect the fact that changing the code of a file does not really alter its informational contents. When the user wants the recoded files to be timestamped at the recoding time, this option inhibits the automatic protection of the timestamps.

`-v`
`--verbose`

Before doing any recoding, the program will first print on '`stderr`' the list of all intermediate charsets planned for recoding, starting with the *before* charset and ending with the *after* charset. It also prints an indication of the recoding quality, as one of the word '`reversible`', '`one to one`', '`one to many`', '`many to one`' or '`many to many`'.

This information will appear once or twice. It is shown a second time only when the optimization and step merging phase succeeds in creating a new single step.

This option also has a second effect. The program will print on '`stderr`' one message per *file* recoded, so to let the user informed of the progress of its command.

An easy way to know beforehand the sequence or quality of a recoding is by using the command such as:

`recode -v` *before*`:`*after* `< /dev/null`

using the fact that, *so far* in `recode`, an empty input file produces an empty output file.

`-x=`*charset*
`--ignore=`*charset*

This option tells the program to ignore any recoding path through the specified *charset*, so disabling any single step using this charset as a start or end point. This may be used when the user wants to force `recode` in using an alternate recoding path.

charset may be abbreviated to any unambiguous prefix. For convenience, the value '`.`' is an alias for '`RFC 1345`', so the option `-x.` effectively disables *all* RFC 1345 tables at once.

`--help` The program merely prints a page of help on standard output, and exits without doing any recoding.

`--version`

The program merely prints its version numbers on standard output, and exits without doing anything else.

The *before*:*after* argument specifies the start charset and the goal charset. The allowable values for *before* or *after* are described in the remainder of this document. Charsets may have predefined alternate names, or aliases, which are equally acceptable.

In the *before*:*after* argument only, a backslash may be used to quote the next character of a charset name. This might be useful for preventing a colon to be mistakenly interpreted as the separator between *before* and *after*. Rather, the colon could be omitted, because while recognizing a charset name or alias, GNU `recode` ignores all characters besides letters and digits. There is also no distinction between upper and lower case. Charset names or aliases may always be abbreviated to any unambiguous prefix.

One or both of the *before* or *after* keywords may be omitted, but the colon which separates them cannot. An omitted keyword implies the usual or default code in usage on the system where this program is installed. Usually, this default code is `latin1` for UNIX systems or `ibmpc` for MS-DOS machines.

3 Charsets from RFC 1345

In the GNU `recode` distribution, there is a copy of RFC 1345:

> "Character Mnemonics & Character Sets", K. Simonsen, Request for Comments no. 1345, Network Working Group, June 1992.

This document is also available by anonymous ftp at '`nic.ddn.mil`' in directory '`rfc`' as file '`rfc1345.txt`'. This report defines many character mnemonics and character sets.

GNU `recode` implements most of RFC 1345, however:

1. It does not recognize 16-bits charsets: `GB_2312-80`, `JIS_C6226-1978`, `JIS_C6226-1983`, `JIS_X0212-1990` and `KS_C_5601-1987`.
2. It does not recognize those charsets which combine two characters for representing a third: `ANSI_X3.110-1983`, `ISO_6937-2-add`, `T.101-G2`, `T.61-8bit`, `iso-ir-90` and `videotex-suppl`.
3. It interprets the charset `isoir91` as `NATS-DANO` (alias `iso-ir-9-1`, *not* as `JIS_C6229-1984-a` (alias `iso-ir-91`). So better avoid using these two alias names.
4. It interprets the charset `isoir92` as `NATS-DANO-ADD` (alias `iso-ir-9-2`, *not* as `JIS_C6229-1984-b` (alias `iso-ir-92`). So better avoid using these two alias names.
5. It ignores all about code overloading, but still processes correctly the remainder of `dk-us` and `us-dk`.

Keld Simonsen '`keld@dkuug.dk`' did most of RFC 1345 himself, with some funding from Danish Standards and Nordic standards (INSTA) project. He also did the character set design work, with substantial input from Olle Jaernefors. Keld typed in almost all of the tables, some have been contributed. A number of people have checked the tables in various ways. The RFC lists a number of people who helped.

`ANSI_X3.4-1968`
: `ANSI_X3.4-1986`, `ASCII`, `IBM367`, `ISO646-US`, `ISO_646.irv:1991`, `US-ASCII`, `cp367`, `iso-ir-6` and `us` are aliases for this charset. source: ECMA registry

`ASMO_449`
: `ISO_9036`, `arabic7` and `iso-ir-89` are aliases for this charset. source: ECMA registry

`BS_4730`
: `ISO646-GB`, `gb`, `iso-ir-4` and `uk` are aliases for this charset. source: ECMA registry

BS_viewdata
: iso-ir-47 is an alias for this charset. source: ECMA registry

CSA_Z243.4-1985-1
: ISO646-CA, ca, csa7-1 and iso-ir-121 are aliases for this charset. source: ECMA registry

CSA_Z243.4-1985-2
: ISO646-CA2, csa7-2 and iso-ir-122 are aliases for this charset. source: ECMA registry

CSA_Z243.4-1985-gr
: iso-ir-123 is an alias for this charset. source: ECMA registry

CSN_369103
: iso-ir-139 is an alias for this charset. source: ECMA registry

DEC-MCS
: dec is an alias for this charset. VAX/VMS User's Manual, Order Number: AI-Y517A-TE, April 1986.

DIN_66003
: ISO646-DE, de and iso-ir-21 are aliases for this charset. source: ECMA registry

DS_2089
: DS2089, ISO646-DK and dk are aliases for this charset. source: Danish Standard, DS 2089, February 1974

EBCDIC-AT-DE
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-AT-DE-A
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-CA-FR
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-DK-NO
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-DK-NO-A
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-ES
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-ES-A
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-ES-S
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-FI-SE
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

EBCDIC-FI-SE-A
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

`EBCDIC-FR`
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

`EBCDIC-IT`
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

`EBCDIC-PT`
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

`EBCDIC-UK`
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

`EBCDIC-US`
: source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

`ECMA-cyrillic`
: `iso-ir-111` is an alias for this charset. source: ECMA registry

`ES`
: `ISO646-ES` and `iso-ir-17` are aliases for this charset. source: ECMA registry

`ES2`
: `ISO646-ES2` and `iso-ir-85` are aliases for this charset. source: ECMA registry

`GB_1988-80`
: `ISO646-CN`, `cn` and `iso-ir-57` are aliases for this charset. source: ECMA registry

`GOST_19768-74`
: `ST_SEV_358-88` and `iso-ir-153` are aliases for this charset. source: ECMA registry

`IBM037`
: `cp037`, `ebcdic-cp-ca`, `ebcdic-cp-nl`, `ebcdic-cp-us` and `ebcdic-cp-wt` are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

`IBM038`
: `EBCDIC-INT` and `cp038` are aliases for this charset. source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990

`IBM1026`
: `CP1026` is an alias for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

`IBM273`
: `CP273` is an alias for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

`IBM274`
: `CP274` and `EBCDIC-BE` are aliases for this charset. source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990

`IBM275`
: `EBCDIC-BR` and `cp275` are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

`IBM277`
: `EBCDIC-CP-DK` and `EBCDIC-CP-NO` are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

`IBM278`
: `CP278`, `ebcdic-cp-fi` and `ebcdic-cp-se` are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

`IBM280`
: `CP280` and `ebcdic-cp-it` are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM281 EBCDIC-JP-E and cp281 are aliases for this charset. source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990

IBM284 CP284 and ebcdic-cp-es are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM285 CP285 and ebcdic-cp-gb are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM290 EBCDIC-JP-kana and cp290 are aliases for this charset. source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990

IBM297 cp297 and ebcdic-cp-fr are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM420 cp420 and ebcdic-cp-ar1 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990 IBM NLS RM p 11-11

IBM423 cp423 and ebcdic-cp-gr are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM424 cp424 and ebcdic-cp-he are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM437 437 and cp437 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM500 CP500, ebcdic-cp-be and ebcdic-cp-ch are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM850 850 and cp850 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM851 851 and cp851 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM852 852 and cp852 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM855 855 and cp855 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM857 857 and cp857 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM860 860 and cp860 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM861 861, cp-is and cp861 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM862 862 and cp862 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM863 863 and cp863 are aliases for this charset. source: IBM Keyboard layouts and code pages, PN 07G4586 June 1991

IBM864 cp864 is an alias for this charset. source: IBM Keyboard layouts and code pages, PN 07G4586 June 1991

IBM865 865 and cp865 are aliases for this charset. source: IBM DOS 3.3 Ref (Abridged), 94X9575 (Feb 1987)

IBM868 CP868 and cp-ar are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM869 869, cp-gr and cp869 are aliases for this charset. source: IBM Keyboard layouts and code pages, PN 07G4586 June 1991

IBM870 CP870, ebcdic-cp-roece and ebcdic-cp-yu are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM871 CP871 and ebcdic-cp-is are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM880 EBCDIC-Cyrillic and cp880 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM891 cp891 is an alias for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM903 cp903 is an alias for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM904 904 and cp904 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IBM905 CP905 and ebcdic-cp-tr are aliases for this charset. source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990

IBM918 CP918 and ebcdic-cp-ar2 are aliases for this charset. source: IBM NLS RM Vol2 SE09-8002-01, March 1990

IEC_P27-1
: iso-ir-143 is an alias for this charset. source: ECMA registry

INIS iso-ir-49 is an alias for this charset. source: ECMA registry

INIS-8 iso-ir-50 is an alias for this charset. source: ECMA registry

INIS-cyrillic
: iso-ir-51 is an alias for this charset. source: ECMA registry

INVARIANT

ISO_10367-box
: iso-ir-155 is an alias for this charset. source: ECMA registry

ISO_2033-1983
: e13b and iso-ir-98 are aliases for this charset. source: ECMA registry

ISO_5427 iso-ir-37 is an alias for this charset. source: ECMA registry

ISO_5427:1981
: iso-ir-54 is an alias for this charset. source: ECMA registry

ISO_5428:1980
: iso-ir-55 is an alias for this charset. source: ECMA registry

ISO_646.basic:1983
: ref is an alias for this charset. source: ECMA registry

ISO_646.irv:1983
: irv and iso-ir-2 are aliases for this charset. source: ECMA registry

ISO_6937-2-25
: iso-ir-152 is an alias for this charset. source: ECMA registry

ISO_8859-1:1987
: CP819, IBM819, ISO-8859-1, ISO_8859-1, iso-ir-100, l1 and latin1 are aliases for this charset. source: ECMA registry

ISO_8859-2:1987
: ISO-8859-2, ISO_8859-2, iso-ir-101, l2 and latin2 are aliases for this charset. source: ECMA registry

ISO_8859-3:1988
: ISO-8859-3, ISO_8859-3, iso-ir-109, l3 and latin3 are aliases for this charset. source: ECMA registry

ISO_8859-4:1988
: ISO-8859-4, ISO_8859-4, iso-ir-110, l4 and latin4 are aliases for this charset. source: ECMA registry

ISO_8859-5:1988
: ISO-8859-5, ISO_8859-5, cyrillic and iso-ir-144 are aliases for this charset. source: ECMA registry

ISO_8859-6:1987
: ASMO-708, ECMA-114, ISO-8859-6, ISO_8859-6, arabic and iso-ir-127 are aliases for this charset. source: ECMA registry

ISO_8859-7:1987
: ECMA-118, ELOT_928, ISO-8859-7, ISO_8859-7, greek, greek8 and iso-ir-126 are aliases for this charset. source: ECMA registry

`ISO_8859-8:1988`
: `ISO-8859-8`, `ISO_8859-8`, `hebrew` and `iso-ir-138` are aliases for this charset. source: ECMA registry

`ISO_8859-9:1989`
: `ISO-8859-9`, `ISO_8859-9`, `iso-ir-148`, `l5` and `latin5` are aliases for this charset. source: ECMA registry

`ISO_8859-supp`
: `iso-ir-154` and `latin1-2-5` are aliases for this charset. source: ECMA registry

`IT`
: `ISO646-IT` and `iso-ir-15` are aliases for this charset. source: ECMA registry

`JIS_C6220-1969-jp`
: `JIS_C6220-1969`, `iso-ir-13`, `katakana` and `x0201-7` are aliases for this charset. source: ECMA registry

`JIS_C6220-1969-ro`
: `ISO646-JP`, `iso-ir-14` and `jp` are aliases for this charset. source: ECMA registry

`JIS_C6229-1984-a`
: `jp-ocr-a` is an alias for this charset. source: ECMA registry

`JIS_C6229-1984-b`
: `ISO646-JP-OCR-B` and `jp-ocr-b` are aliases for this charset. source: ECMA registry

`JIS_C6229-1984-b-add`
: `iso-ir-93` and `jp-ocr-b-add` are aliases for this charset. source: ECMA registry

`JIS_C6229-1984-hand`
: `iso-ir-94` and `jp-ocr-hand` are aliases for this charset. source: ECMA registry

`JIS_C6229-1984-hand-add`
: `iso-ir-95` and `jp-ocr-hand-add` are aliases for this charset. source: ECMA registry

`JIS_C6229-1984-kana`
: `iso-ir-96` is an alias for this charset. source: ECMA registry

`JIS_X0201`
: `X0201` is an alias for this charset.

`JUS_I.B1.002`
: `ISO646-YU`, `iso-ir-141`, `js` and `yu` are aliases for this charset. source: ECMA registry

`JUS_I.B1.003-mac`
: `iso-ir-147` and `macedonian` are aliases for this charset. source: ECMA registry

`JUS_I.B1.003-serb`
: `iso-ir-146` and `serbian` are aliases for this charset. source: ECMA registry

`KSC5636`
: `ISO646-KR` is an alias for this charset.

`Latin-greek-1`
: `iso-ir-27` is an alias for this charset. source: ECMA registry

MSZ_7795.3
: ISO646-HU, hu and iso-ir-86 are aliases for this charset. source: ECMA registry

NATS-DANO
: iso-ir-9-1 is an alias for this charset. source: ECMA registry

NATS-DANO-ADD
: iso-ir-9-2 is an alias for this charset. source: ECMA registry

NATS-SEFI
: iso-ir-8-1 is an alias for this charset. source: ECMA registry

NATS-SEFI-ADD
: iso-ir-8-2 is an alias for this charset. source: ECMA registry

NC_NC00-10:81
: ISO646-CU, cuba and iso-ir-151 are aliases for this charset. source: ECMA registry

NF_Z_62-010
: ISO646-FR, fr and iso-ir-69 are aliases for this charset. source: ECMA registry

NF_Z_62-010_(1973)
: ISO646-FR1 and iso-ir-25 are aliases for this charset. source: ECMA registry

NS_4551-1
: ISO646-NO, iso-ir-60 and no are aliases for this charset. source: ECMA registry

NS_4551-2
: ISO646-NO2, iso-ir-61 and no2 are aliases for this charset. source: ECMA registry

PT
: ISO646-PT and iso-ir-16 are aliases for this charset. source: ECMA registry

PT2
: ISO646-PT2 and iso-ir-84 are aliases for this charset. source: ECMA registry

SEN_850200_B
: FI, ISO646-FI, ISO646-SE, iso-ir-10 and se are aliases for this charset. source: ECMA registry

SEN_850200_C
: ISO646-SE2, iso-ir-11 and se2 are aliases for this charset. source: ECMA registry

T.61-7bit
: iso-ir-102 is an alias for this charset. source: ECMA registry

dk-us

greek-ccitt
: iso-ir-150 is an alias for this charset. source: ECMA registry

greek7
: iso-ir-88 is an alias for this charset. source: ECMA registry

greek7-old
: iso-ir-18 is an alias for this charset. source: ECMA registry

`hp-roman8`
: `r8` and `roman8` are aliases for this charset. source: LaserJet IIP Printer User's Manual, HP part no 33471-90901, Hewlet-Packard, June 1989.

`latin-greek`
: `iso-ir-19` is an alias for this charset. source: ECMA registry

`latin-lap`
: `iso-ir-158` and `lap` are aliases for this charset. source: ECMA registry

`latin6`
: `iso-ir-157` and `l6` are aliases for this charset. source: ECMA registry

`macintosh`
: `mac` is an alias for this charset. source: The Unicode Standard ver1.0, ISBN 0-201-56788-1, Oct 1991

`us-dk`
: for compatibility with ASCII

4 Charsets based on ASCII

4.1 Usual ASCII

This charset is available in `recode` under the name `ascii`. In fact, it's true name is `ANSI_X3.4-1968` as per RFC 1345, accepted aliases being `ANSI_X3.4-1986`, `ASCII`, `IBM367`, `ISO646-US`, `ISO_646.irv:1991`, `US-ASCII`, `cp367`, `iso-ir-6` and `us`. The shortest way of specifying it in `recode` is `us`.

This documentation used to include ASCII tables. They have been removed since `recode` can now recreate these (and a lot of others) easily:

```
recode -lf ascii                for commented ASCII
recode -ld ascii                for concise decimal table
recode -lo ascii                for concise octal table
recode -lh ascii                for concise hexadecimal table
```

4.2 ASCII extended by Latin Alphabets

This charset is available in `recode` under the name `latin1`. In fact, it's true name is `ISO_8859-1:1987` as per RFC 1345, accepted aliases being `CP819`, `IBM819`, `ISO-8859-1`, `ISO_8859-1`, `iso-ir-100`, `l1` and `latin1`. The shortest way of specifying it in `recode` is `l1`.

This charset corresponds to the ISO Latin Alphabet 1. It is an eight-bit code which coincides with ASCII for the lower half.

This documentation used to include Latin-1 tables. They have been removed since `recode` can now recreate these (and a lot of others) easily:

```
recode -lf latin1               for commented ISO Latin-1
recode -ld latin1               for concise decimal table
recode -lo latin1               for concise octal table
recode -lh latin1               for concise hexadecimal table
```

The following from '`lasko@video.dec.com`' (Tim Lasko), with no date.

ISO Latin-1, or more completely ISO Latin Alphabet No 1, is now an international standard as of February 1987 (IS 8859, Part 1). For those American USEnet'rs that care, the 8-bit ASCII standard, which is essentially the same code, is going through the final administrative processes prior to publication.

ISO Latin-1 (IS 8859/1) is actually one of an entire family of eight-bit one-byte character sets, all having ASCII on the left hand side, and with varying repertoires on the right hand side:

```
Pt 1.   Latin Alphabet No 1  (caters to Western Europe - now approved)
Pt 2.   Latin Alphabet No 2  (caters to Eastern Europe - now approved)
Pt 3.   Latin Alphabet No 3  (caters to SE Europe + others - in draft ballot)
Pt 4.   Latin Alphabet No 4  (caters to Northern Europe - in draft ballot)
Pt 5.   Latin-Cyrillic alphabet  (right half all Cyrillic - processing
                                  currently suspended pending USSR input)
Pt 6.   Latin-Arabic alphabet    (right half all Arabic - now approved)
Pt 7.   Latin-Greek alphabet     (right half Greek + symbols - in draft ballot)
Pt 8.   Latin-Hebrew alphabet    (right half Hebrew + symbols - proposed)
```

4.3 ASCII 7-bits, BS to overstrike

This charset is available in `recode` under the name `ascii-bs`.

The file is straight ASCII, seven bits only. According to the definition of ASCII: diacritics are applied by a sequence of three characters: the letter, one `BS`, the diacritic mark. We deviate slightly from this by exchanging the diacritic mark and the letter so, on a screen device, the diacritic will disappear and let the letter alone. At recognition time, both methods are acceptable.

The French quotes are coded by the sequences: `< BS "` or `" BS <` for the opening quote and `> BS "` or `" BS >` for the closing quote. This artifical convention was inherited in straight `ascii-bs` from habits around `bangbang` entry, and is not well known. But we decided to stick to it so that `ascii-bs` charset will not loose French quotes.

The `ascii-bs` charset is independant of `ascii`, and different. The following examples demonstrate this, knowing at advance that '`!2`' is the `bangbang` way of representing an `e` with an acute accent. Compare:

```
% echo \!2 | recode -v bang:ascii | od -bc
bangbang -> iso-8859-1-1987 -> rfc1345 -> ansi-x3.4-1968  (many to one)
bangbang -> iso-8859-1-1987 -> ansi-x3.4-1968  (many to one)
0000000 351 012
351  \n
```

```
0000002
```

with:

```
% echo \!2 | recode -v bang:ascii-bs | od -bc
bangbang -> iso-8859-1-1987 -> ascii-bs  (many to many)
0000000 047 010 145 012
  '  \b   e  \n
0000004
```

In the first case, the `e` with an acute accent is merely transmitted by the `latin1:ascii` mapping, not having a special recoding rule for it. In the `latin1:ascii-bs` case, the acute accent is applied over the `e` with a backspace: diacriticized characters have special rules. For the `ascii-bs` charset, reversibility is still possible, but there might be difficult cases.

4.4 ASCII without diacritics nor underline

This charset is available in `recode` under the name `flat`.

This code is ASCII expunged of all diacritics and underlines, as long as they are applied using three character sequences, with `BS` in the middle. Also, despite slightly unrelated, each control character is represented by a sequence of two or three graphic characters. The newline character, however, keeps its functionnality and is not represented.

Note that charset `flat` is a terminal charset. We can convert *to* `flat`, but not *from* it.

5 Charsets based on IBM

5.1 EBCDIC code

This charset is the IBM's external binary coded decimal for interchange coding. This is an eight bits code. The following three variants were implemented in GNU `recode` independantly of RFC 1345:

`ebcdic` This charset represents the way Control Data Corporation relates EBCDIC to 8-bits ASCII. GNU `dd ebcdic` conversion is identical.

`ebcdic-ccc`
This charset represents the way Concurrent Computer Corporation (formerly Perkin Elmer) relates EBCDIC to 8-bits ASCII.

`ebcdic-ibm`
This charset is almost identical to the GNU `dd ibm` conversion. For the GNU `dd ibm` table, `recode` said:

```
Codes  91 and 213 both recode to 173
Codes  93 and 229 both recode to 189
No character recodes to  74
No character recodes to 106
```

So I arbitrarily chose to recode 213 by 74 and 229 by 106. This makes the `ebcdic-ibm` recoding reversible, but this is not necessarily the best correction. In any case, I believe GNU `dd` should be corrected, and preferrably, GNU `dd` and GNU `recode` should agree on the correction. So, this table may change once again.

RFC 1345 brings in `recode` 15 other EBCDIC charsets, and 21 other charsets having EBCDIC in at least one of their alias names. You can get a list of all these by executing:

```
recode -l | grep ebcdic
```

5.2 IBM's PC code

This charset is available in `recode` under the name `ibmpc`. There are a few discrepancies between this charset and the very similar RFC 1345 charset `ibm437`, which have not been analyzed yet, so the charsets are being kept separate for now. This might change in the future.

The file was obtained or is aimed towards a PC microcomputer from IBM or any compatible. This is an eight-bit code.

5.3 Unisys' ICON code

This charset is available in `recode` under the name `iconqnx`.

The file is using Unisys' ICON way to represent diacritics with code 25 escape sequences. This is a seven-bit code, even if eight-bit codes can flow through as part of IBM-PC charset.

6 Charsets based on CDC

6.1 Control Data's Display Code

This code is not available in `recode`, but repeated here for reference. This is a 6-bit code used on CDC mainframes.

```
Octal display code to graphic       Octal display code to octal ASCII

00  :    20  P    40  5    60  #     00 072  20 120  40 065  60 043
01  A    21  Q    41  6    61  [     01 101  21 121  41 066  61 133
02  B    22  R    42  7    62  ]     02 102  22 122  42 067  62 135
03  C    23  S    43  8    63  %     03 103  23 123  43 070  63 045
04  D    24  T    44  9    64  "     04 104  24 124  44 071  64 042
05  E    25  U    45  +    65  _     05 105  25 125  45 053  65 137
06  F    26  V    46  -    66  !     06 106  26 126  46 055  66 041
07  G    27  W    47  *    67  &     07 107  27 127  47 052  67 046
10  H    30  X    50  /    70  '     10 110  30 130  50 057  70 047
11  I    31  Y    51  (    71  ?     11 111  31 131  51 050  71 077
12  J    32  Z    52  )    72  <     12 112  32 132  52 051  72 074
13  K    33  0    53  $    73  >     13 113  33 060  53 044  73 076
14  L    34  1    54  =    74  @     14 114  34 061  54 075  74 100
15  M    35  2    55       75  \     15 115  35 062  55 040  75 134
16  N    36  3    56  ,    76  ^     16 116  36 063  56 054  76 136
17  O    37  4    57  .    77  ;     17 117  37 064  57 056  77 073
```

6.2 ASCII 6/12 from NOS

This charset is available in `recode` under the name `cdcnos`.

This is one of the charset in use on CDC Cyber NOS systems to represent ASCII, sometimes named *NOS 6/12* code for coding ASCII. This code is also known as *caret ASCII*. It is based on a six bits character set in which small letters and control characters are coded using a `^` escape and, sometimes, a `@` escape.

The routines given here presume that the six bits code is already expressed in ASCII by the communication channel, with embedded ASCII `^` and `@` escapes.

Here is a table showing which characters are being used to encode each ASCII character.

```
000  ^5  020  ^#  040     060  0  100 @A  120  P  140  @G  160  ^P
001  ^6  021  ^[  041  !  061  1  101  A  121  Q  141  ^A  161  ^Q
002  ^7  022  ^]  042  "  062  2  102  B  122  R  142  ^B  162  ^R
003  ^8  023  ^%  043  #  063  3  103  C  123  S  143  ^C  163  ^S
004  ^9  024  ^"  044  $  064  4  104  D  124  T  144  ^D  164  ^T
005  ^+  025  ^_  045  %  065  5  105  E  125  U  145  ^E  165  ^U
006  ^-  026  ^!  046  &  066  6  106  F  126  V  146  ^F  166  ^V
007  ^*  027  ^&  047  ’  067  7  107  G  127  W  147  ^G  167  ^W
010  ^/  030  ^’  050  (  070  8  110  H  130  X  150  ^H  170  ^X
011  ^(  031  ^?  051  )  071  9  111  I  131  Y  151  ^I  171  ^Y
012  ^)  032  ^<  052  *  072 @D  112  J  132  Z  152  ^J  172  ^Z
013  ^$  033  ^>  053  +  073  ;  113  K  133  [  153  ^K  173  ^0
014  ^=  034  ^@  054  ,  074  <  114  L  134  \  154  ^L  174  ^1
015  ^   035  ^\  055  -  075  =  115  M  135  ]  155  ^M  175  ^2
016  ^,  036  ^^  056  .  076  >  116  N  136 @B  156  ^N  176  ^3
017  ^.  037  ^;  057  /  077  ?  117  O  137  _  157  ^O  177  ^4
```

6.3 ASCII “bang bang”

This charset is available in `recode` under the name `bangbang`.

This is the local code in use on Cybers at Universite de Montreal, which grave and serious people there prefer to name *ASCII code display*. This code is also known as *Bang-bang*. It is based on a six bits character set in which capitals, French diacritics and a few others are coded using an `!` escape followed by a single character, and control characters using a double `!` escape followed by a single character.

The routines given here presume that the six bits code is already expressed in ASCII by the communication channel, with embedded ASCII `!` escapes.

Here is a table showing which characters are being used to encode each ASCII character.

```
000 !!@  020 !!P  040    060 0  100 @   120 !P  140 !@ 160 P
001 !!A  021 !!Q  041 !" 061 1  101 !A  121 !Q  141 A  161 Q
002 !!B  022 !!R  042 "  062 2  102 !B  122 !R  142 B  162 R
003 !!C  023 !!S  043 #  063 3  103 !C  123 !S  143 C  163 S
004 !!D  024 !!T  044 $  064 4  104 !D  124 !T  144 D  164 T
005 !!E  025 !!U  045 %  065 5  105 !E  125 !U  145 E  165 U
006 !!F  026 !!V  046 &  066 6  106 !F  126 !V  146 F  166 V
007 !!G  027 !!W  047 ’  067 7  107 !G  127 !W  147 G  167 W
010 !!H  030 !!X  050 (  070 8  110 !H  130 !X  150 H  170 X
011 !!I  031 !!Y  051 )  071 9  111 !I  131 !Y  151 I  171 Y
012 !!J  032 !!Z  052 *  072 :  112 !J  132 !Z  152 J  172 Z
```

```
013 !!K  033 !![  053 +  073 ;  113 !K  133 [   153 K  173 ![
014 !!L  034 !!\  054 ,  074 <  114 !L  134 \   154 L  174 !\
015 !!M  035 !!]  055 -  075 =  115 !M  135 ]   155 M  175 !]
016 !!N  036 !!^  056 .  076 >  116 !N  136 ^   156 N  176 !^
017 !!O  037 !!_  057 /  077 ?  117 !O  137 _   157 O  177 !_
```

7 Non-IBM micro-computer charsets

7.1 Apple's Macintosh code

This charset is available in `recode` under the name `applemac`. There are a few discrepancies between this charset and the very similar RFC 1345 charset `macintosh`, which have not been analyzed yet, so the charsets are being kept separate for now. This might change in the future.

The file has been obtained or is aimed to a Macintosh micro-computer from Apple. This is an eight bit code. The file is the data fork only.

7.2 Atari ST code

This charset is available in `recode` under the name `atarist`.

This is the character set used on the Atari ST/TT/Falcon. This is similar to `ibmpc`, but differs in some details (includes some more accented characters, the graphic characters are mostly replaced by hebrew characters, and there is a true german `SHARP S` different from `GREEK BETA`).

About the end-of-line conversions: the canonical end-of-line on the Atari is '`\r\n`', but unlike `ibmpc`, the OS makes no difference between text and binary input/output; it is up to the application how to interpret the data. In fact, most of the libraries that come with compilers can grok both '`\r\n`' and '`\n`' as end of lines. Many of the users who also have access to Unix systems prefer '`\n`' to ease porting Unix utilities. So, for easing reversibility, `recode` tries to let '`\r`' undisturbed through recodings.

7.3 NeXT international code

This charset is available in `recode` under the name `NeXTSTEP`.

The NeXTSTEP encoding is an extension to the ISO Latin-1 ASCII encoding used by NeXT. It is identical to Latin-1 for the positions 0-127. In the position 128-255, NeXT added some chars and shuffled them around a little bit (for some unknown reason).

8 Some other charsets

Even if these charsets were originally added to `recode` for handling texts written in French, they find other uses. We did use them lot for writing French diacriticized texts in the past, so `recode` knows how to handle these particularily well for French texts.

8.1 ASCII with LaTeX codes

This charset is available in `recode` under the name `latex` and has `ltex` as an alias. It is used for ASCII files coded to be read by LaTeX or, in certain cases, by TeX.

Whenever you recode from another charset to `latex`, beware that all occurrences of backslashes ('\') are usually translated into the string '`\backslash{}`'. However, in practice, people often use backslashes in the other charset for introducing TeX commands, compromising it: it is not pure TeX, nor it is pure other charset. This translation of backslashes into '`\backslash{}`' can be rather inconvenient, it may be inhibited through the command option `-d`.

8.2 ASCII with easy French conventions

This charset is available in `recode` under the name `texte` and has `txte` for an alias.

This charset is identical to `ascii-bs`, save for French diacritics which are noted using a slightly different convention.

These conventions are used in `texte` and `latexte` charsets, which are seven bits codes. At text entry time, these conventions provide a little speed up. At read time, they slightly improve the readability. Of course, it would better to have a specialized keyboard to make direct eight bits entries and fonts for immediately displaying eight bit ISO Latin-1 characters. But not everybody is so fortunate. In several mailing environment, the eight bit is often willfully destroyed (an horrible Crime that most people do not care to straighten up).

Easy French has been in use in France for a while. I only slightly adapted it (the diaeresis option) to make it more comfortable to several usages in Québec originating from Université de Montréal. In fact, the main problem for me was not to necessarily to invent Easy French, but to recognize the "best" convention to use, (best is not being defined, here) and to try to solve the main pithfalls associated with the selected convention.

8.2.1 Diacritics

French quotes (sometimes called "angle quotes") are noted the same way English quotes are noted in TEX, *id est* by `‘‘` and `’’`.

No effort has been put to preserve Latin ligatures (`ae`, `oe`) which are representable in several other charsets. So, these ligatures may be lost through Easy French conventions.

This is almost the French convention for simplified diacritics entry:

`e’`	Acute accent
`e‘`	Grave accent
`e^`	Circumflex accent
`e"`	Diaeresis
`c,`	Cedilla

In some countries, `:` is used instead of `"` to mark diaeresis. `recode` support one convention on a single call, depending on the `-c` option of the `recode` command.

The convention is prone to loosing information, because the diacritic meaning overloads some characters that already have other uses. To alleviate this, some knowledge of the French language is insufflated into the recognition routines. So, the following subtleties are systematically obeyed by the various recognizers.

- A single quote which follows a `e` does not necessarily means an acute accent if it is followed by a single other one. For example:

`e’`	will give an `e` with an acute accent.
`e’’`	will give a simple `e`, with a closing quotation mark.
`e’’’`	will give an `e` with an acute accent, followed by a closing quotation mark.

  There is a problem induced by this convention if there are English citations with a French text. In sentences like:

  > There’s a meeting at Archie’s restaurant.

  the single quotes will be mistaken twice for acute accents. So English contractions and suffix possessives could be mangled.

- A double quote or colon, depending on `-c` option, which follows a vowel is interpreted as diaeresis only if it is followd by another letter. But there are in French several words that *end* with a diaeresis, the program also recognizes them. See Section 8.2.2 [Ending diaeresis], page 33, for a study of all the problematic cases.
- A comma which follows a `c` is interpreted as a cedilla only if it is followd by one of the vowels `a`, `o` and `u`.

8.2.2 List of words ending with diaeresis

Here is a classification of all cases of a diaeresis at the end of a French word:

- Words ending in "igue"
  - Feminine words without a relative masculine:
    `besaigue" cigue"`
  - Feminine words with a relative masculine: (1)
    `aigue" ambigue" contigue" exigue" subaigue" suraigue"`
- Words not ending in "igue"
  - Ended by "i": (2)
    `ai" congai" goi" hai"kai" inoui" sai" samurai" thai" tokai"`
  - Ended by "e":
    `canoe"`
  - Ended by "u": (3)
    `Esau"`

Notes:

1. There are supposed to be seven words in this case. So, one is missing.
2. Look at the following sentence:
   `"Ai"e! Voici le proble'me que j'ai"`

   or, using the `-c` option:
   Ai:e! Voici le proble'me que j'ai:

   There is an ambiguity between an *ai*", the small animal, and the indicative future of *avoir* (first person singular), when followed by what could be a diaeresis mark. Hopefully, the case is solved by the fact that an apostrophe always precedes the verb and almost never the animal.
3. I did not pay attention to proper nouns, but this one showed up as being fairly evident.

Just to complete this topic, note that it would be wrong to make a rule for all words ending in "igue" as needing a diaerisis. Here are counter-examples:

```
becfigue be‘sigue bigue bordigue bourdigue brigue contre-digue
digue d’intrigue fatigue figue garrigue gigue igue intrigue
ligue prodigue sarigue zigue
```

9 Internal aspects

Suppose that four elementary steps are selected at path optimization time. Then `recode` will split itself into four different tasks interconnected with pipes, logically equivalent to:

step1 <*input* | *step2* | *step3* | *step4* >*output*

9.1 Overall organization

The main driver constructs, while initializing all conversion modules, a table giving all the conversion routines available (*single step*s) and for each, the starting charset and the ending charset. If we consider these charsets as being the nodes of a directed graph, each single step may be considered as oriented arc from one node to the other. A cost is attributed to each arc: for example, a high penality is given to single steps which are prone to loosing characters, a low penality is given to those which need studying more than one input character for producing an output character, etc.

Given a starting code and a goal code, `recode` computes the most economical route through the elementary recodings, that is, the best sequence of conversions that will transform the input charset into the final charset. To speed up execution, `recode` looks for subsequences of conversions which are simple enough to be merged, it then dynamically creates new single steps, of course, use them.

A *double step* is a sequence of two single steps, the output of the first being the special charset `rfc1345` (which is not directly available to the user), the input of the second single step being also `rfc1345`. A special machinery dynamically produces efficient, reversible, mergeable single steps out of these double steps.

The main part of `recode` is written in C, as are most single steps. A few single steps need to recognize sequences of multiple characters, they are often better written in `flex`.

9.2 Adding new charsets

It is easy for a programmer to add a new charset to `recode`. All it requires is making a few functions kept in a single '`.c`' file, adjusting '`Makefile.in`', and remaking `recode`.

One of the function should convert from any previous charset to the new one. Any previous charset will do, but try to select it so you will not loose too much information while converting. The other function should convert from the new charset to any older one. You do not have to select the same old charset than what you selected for the previous routine. Once again, select any charset for which you will not loose too much information while converting.

If, for any of these two functions, you have to read multiple bytes of the old charset before recognizing the character to produce, you might prefer programming it in `flex` in a separate '`.l`' file. Prototype your C or `flex` files after one of those which exist already, so to keep the sources uniform. Besides, at `make` time, all '`.l`' files are automatically merged into a single big one by the script '`mergelex.awk`', which requires sources to follow some rules. Mimetism is a simple approach which relieves me of explaining all these rules!

Each of your source files should have its own initialization function, named `module_`*charset*, which is meant to be executed quickly, once, prior to any recoding. It should declare the name of your charsets and the single steps (or elementary recodings) you provide, by calling `declare_step` one or more times. Besides the charset names, `declare_step` expects a description of the recoding quality (see '`recode.h`') and two functions you also provide.

The first such function has the purpose of allocating structures, preconditionning conversion tables, etc. It is also the usual way of further modifying the `STEP` structure. This function is executed only if and when the single step is retained in an actual recoding sequence. If you do not need such delayed initialization, merely use NULL for the function argument.

The second function executes the elementary recoding on a whole file. There are a few cases when you can spare writing this function:

- Some single steps do nothing else than a pure copy of the input onto the output, in this case, you can use the predefined function `file_one_to_one`, but have a delayed initialization for presetting the field `one_to_one` to the predefined value `one_to_same`.
- Some single steps are driven by a table which recodes one character into another; if the recoding does nothing else, you can use the predefined function `file_one_to_one`, but have a delayed initialization for presetting the `STEP` field `one_to_one` with your table.
- Some single steps are driven by a table which recodes one character into a string; if the recoding does nothing else, you can use the predefined function `file_one_to_many`, but have a delayed initialization for presetting the `STEP` field `one_to_many` with your table.

If you have a recoding table handy in a suitable format but do not use one of the predefined recoding functions, it is still a good idea to use a delayed initialization to save it anyway, because `recode` option `-h` will take advantage of this information when available.

Finally, edit '`Makefile.in`' to add the source file name of your routines to the `C_STEPS` or `L_STEPS` macro definition, depending on the fact your routines is written in C or in `flex`. For C files only, also modify the `STEPOBJS` macro definition.

Table of Contents