Rocky Enterprise Linux 9.2 Manual Pages on command 'Sort::Key.3pm'

C:\>man Sort::Key.3pm

Sort::Key(3pm) User Contributed Perl Documentation Sort::Key(3pm)

NAME

Sort::Key - the fastest way to sort anything in Perl

SYNOPSIS

```
use Sort::Key qw(keysort nkeysort ikeysort);

@by_name = keysort { "$_->{surname} $_->{name}" } @people;

# sorting by a numeric key:
@by_age = nkeysort { $_->{age} } @people;

# sorting by a numeric integer key:
@by_sons = ikeysort { $_->{sons} } @people;
```

DESCRIPTION

Sort::Key provides a set of functions to sort lists of values by some calculated key value.

It is faster (usually much faster) and uses less memory than other alternatives implemented around perl sort function (ST, GRT, etc.).

Multi-key sorting functionality is also provided via the companion modules Sort::Key::Multi, Sort::Key::Maker and Sort::Key::Register.

FUNCTIONS

This module provides a large number of sorting subroutines but they are all variations off the "keysort" one:

```
@sorted = keysort { CALC_KEY($_) } @data
```

that is conceptually equivalent to

```
@sorted = sort { CALC_KEY($a) cmp CALC_KEY($b) } @data
```

and where "CALC_KEY($_)" can be any expression to extract the key value from $_ (not only a subroutine call).

For instance, some variations are "nkeysort" that performs a numeric comparison, "rkeysort" that orders the data in descending order, "ikeysort" and "ukeysort" that are optimized versions of "nkeysort" that can be used when the keys are integers or unsigned integers respectively, etc.

Also, inplace versions of the sorters are provided. For instance

```
keysort_inplace { CALC_KEY($_) } @data
```

that is equivalent to

```
@data = keysort { CALC_KEY($_) } @data
```

but being (a bit) faster and using less memory.

The full list of subroutines that can be imported from this module follows:

keysort { CALC_KEY } @array

returns the elements on @array sorted by the key calculated applying "{ CALC_KEY }" to them.

Inside "{ CALC_KEY }", the object is available as $_.

For example:

```
@a=({name=>john, surname=>smith}, {name=>paul, surname=>belvedere});
@by_name=keysort {$_->{name}} @a;
```

This function honours the "use locale" pragma.

nkeysort { CALC_KEY } @array

similar to "keysort" but compares the keys numerically instead of as strings.

This function honours the "use integer" pragma, i.e.:

```
use integer;
my @s=(2.4, 2.0, 1.6, 1.2, 0.8);
my @ns = nkeysort { $_ } @s;
print "@ns\n"
```

prints

```
0.8 1.6 1.2 2.4 2
```

rnkeysort { CALC_KEY } @array

works as "nkeysort", comparing keys in reverse (or descending) numerical order.

ikeysort { CALC_KEY } @array

works as "keysort" but compares the keys as integers (32 bits or more, no

checking is performed for overflows).

rikeysort { CALC_KEY } @array

works as "ikeysort", but in reverse (or descending) order.

ukeysort { CALC_KEY } @array

works as "keysort" but compares the keys as unsigned integers (32 bits or more).

For instance, it can be used to efficiently sort IP4 addresses:

```
my @data = qw(1.2.3.4 4.3.2.1 11.1.111.1 222.12.1.34
              0.0.0.0 255.255.255.0) 127.0.0.1);

my @sorted = ukeysort {
                my @a = split /\./;
                ((($a[0] << 8) + $a[1] << 8) + $a[2] << 8) + $a[3])
             } @data;
```

rukeysort { CALC_KEY } @array

works as "ukeysort", but in reverse (or descending) order.

keysort_inplace { CALC_KEY } @array

nkeysort_inplace { CALC_KEY } @array

ikeysort_inplace { CALC_KEY } @array

ukeysort_inplace { CALC_KEY } @array

rkeysort_inplace { CALC_KEY } @array

rnkeysort_inplace { CALC_KEY } @array

rikeysort_inplace { CALC_KEY } @array

rukeysort_inplace { CALC_KEY } @array

work as the corresponding "keysort" functions but sorting the array inplace.

rsort @array

nsort @array

rnsort @array

isort @array

risort @array

usort @array

rusort @array

rsort_inplace @array

nsort_inplace @array

rnsort_inplace @array

isort_inplace @array

risort_inplace @array

usort_inplace @array

rusort_inplace @array

are simplified versions of its "keysort" cousins. They use the own values as the sorting keys.

For instance those constructions are equivalent:

```
@sorted = nsort @foo;

@sorted = nkeysort { $_ } @foo;

@sorted = sort { $a <=> $b } @foo;
```

multikeysorter(@types)

multikeysorter_inplace(@types)

multikeysorter(\&genkeys, @types)

multikeysorter_inplace(\&genkeys, @types)

are the low level interface to the multi-key sorting functionality (normally, you should use Sort::Key::Maker and Sort::Key::Register or Sort::Key::Multi instead).

They get a list of keys descriptions and return a reference to a multi-key

sorting subroutine.

Types accepted by default are:

```
string, str, locale, loc, integer, int,
unsigned_integer, uint, number, num
```

and support for additional types can be added via the register_type subroutine available from Sort::Key::Types or the more friendly interface available from Sort::Key::Register.

Types can be preceded by a minus sign to indicate descending order.

If the first argument is a reference to a subroutine it is used as the multi-key extraction function. If not, the generated sorters expect one as their first argument.

Example:

```
my $sorter1 = multikeysorter(sub {length $_, $_}, qw(int str));
my @sorted1 = &$sorter1(qw(foo fo o of oof));

my $sorter2 = multikeysorter(qw(int str));
my @sorted2 = &$sorter2(sub {length $_, $_}, qw(foo fo o of oof));
```

SEE ALSO

perl sort function, integer, locale.

Companion modules Sort::Key::Multi, Sort::Key::Register, Sort::Key::Maker and Sort::Key::Natural.

Sort::Key::IPv4, Sort::Key::DateTime and Sort::Key::OID modules add support for additional datatypes to Sort::Key.

Sort::Key::External allows one to sort huge lists that do not fit in the available memory.

Other interesting Perl sorting modules are Sort::Maker, Sort::Naturally and Sort::External.

SUPPORT

To report bugs, send me and email or use the CPAN bug tracking system at <http://rt.cpan.org>.

Commercial support

Commercial support, professional services and custom software development around this module are available through my current company. Drop me an email with a rough description of your requirements and we will get back to you ASAP.

My wishlist

If you like this module and you're feeling generous, take a look at my Amazon Wish List: <http://amzn.com/w/1WU1P6IR5QZ42>

COPYRIGHT AND LICENSE