The `lcg` package

Erich Janka
erich.janka@gmail.com

2013/08/09 (v1.3)

Abstract

This Package contains macros to generate (pseudo) random numbers with LaTeX by a simple linear congruential generator. The user can specify a range of integers containing the generated random numbers.

To pass options to the package, the ⟨*key*⟩=⟨*value*⟩ scheme of the `keyval` package is used.

If no options are specified, random numbers will be generated in the interval from 1 to $2^{31}-1$ (= 2147483647). Let S be the smallest and L the largest number of the specified range, then the following inequalities must hold because of limitations of LaTeX:

$$-2^{31}+1 \leq S \leq L \leq 2^{31}-1 \qquad \text{and} \qquad L-S \leq 2^{31}-2$$

The generated random numbers will be stored in a LaTeX counter definable by the user.

1 User Interface

The `lcg` package is loaded with

`\usepackage[⟨list of options⟩]{lcg}`.

The optional Argument is a comma separated list of entries of the kind ⟨*key*⟩=⟨*value*⟩ or ⟨*key*⟩. In the second case the key will be set to a standard value. For example the line

`\usepackage[first=10, last=20]{lcg}`

loads the package and generates the LaTeX counter `rand` which will hold pseudo random numbers from 10 to 20. All available keys and their standard values are introduced below.

`\rand` Each call of the command `\rand` will write a new random number to the counter provided by the user with the key ⟨*counter*⟩ or to the standard counter of this package—`rand`. Now it's possible to do whatever can be done with counters. The command evokes a linear congruential random number generator described below.

`\reinitrand` The command `\reinitrand[`⟨*list of options*⟩`]` has one optional argument which is identical to the argument of `\usepackage`, i. e. the same keys in the comma separated list are allowed. The effect is that specified keys will be set and all others will be reset to their standard values.

`\chgrand` The syntax of the command `\chgrand` is identical to `\reinitrand`. The difference is that `\chgrand` will set the specified keys but won't effect any other key.

1.1 The Options

This section deals with the list of all available keys and their standard values.

`counter` This key sets the name of the LaTeX counter where the random numbers will be stored. If the counter already exists (maybe somebody likes random page numbers), LaTeX will prompt a warning and will use it. If the counter doesn't exist, it will be defined by this package and set to 0.
Standard value: rand

`first` This key sets the left boarder of the range within random numbers will be generated. Its value can be any number from $-2^{31}+1$ to $2^{31}-1$, as long as it is not greater than the value of the key ⟨*last*⟩ and the difference to the value of ⟨*last*⟩ doesn't exceed $2^{31}-2$.
Standard value: 1

`last` This key sets the right boarder of the range within random numbers will be generated. Its value can be any number from $-2^{31}+1$ to $2^{31}-1$, as long as it is not less than the value of the key ⟨*first*⟩ and the difference to the value of ⟨*first*⟩ doesn't exceed $2^{31}-2$.
Standard value: $2^{31}-1$

`seed` The value of this key is the starting value for the algorithm generating the random numbers and must be within the range 1 to $2^{31}-1$. If the value is smaller than 1, the random number generator will be initialized with the time, the page number and the actual line of the file. This key allows reproduction of the sequences of random numbers.
Standard value: 0

`quiet` When using the `lcg` package it sends some lines of information to the screen and the `log`-file:

- The name of the counter holding the generated random numbers
- The lower bound of the range of random numbers
- The upper bound of the range of random numbers
- The initial value of the random number generator

To supress this output this key can be used. If the value starts with the letter y, Y, j or J there will be no output whereas words beginning with the letters n or N won't supress it.
Standard value: y

2 Example

This documentation loaded the package with:

```
\usepackage[first=10, last=20]{lcg}
```

The lines

```
\rand\arabic{rand} \rand\arabic{rand} \rand\arabic{rand}
\rand\arabic{rand} \rand\arabic{rand} \rand\arabic{rand}
\rand\arabic{rand} \rand\arabic{rand} \rand\arabic{rand}
```

generate these numbers (all between 10 and 20): 16 12 11 10 13 18 20 14 14

Now the counter `die` should simulate a die and hold random numbers from 1 to 6. In addition it is demonstrated how to switch off the output to the screen and to the `log`-file. This can achieved with one of the following lines:

```
\reinitrand[last=6, counter=die, quiet]
```

After that, the numbers 1 4 4 3 4 3. are a product of:

```
\rand\arabic{die} \rand\arabic{die} \rand\arabic{die}
\rand\arabic{die} \rand\arabic{die} \rand\arabic{die}
```

As one can see, the key ⟨*first*⟩ has been reset to 1.

The following lines will change the range to -6 to $+6$ without modifying any other option:

```
\chgrand[first=-6]
```

Here the numbers -5 6 0 -6 -1 6 are stored in a user defined counter and brought to paper by:

```
\rand\arabic{die} \rand\arabic{die} \rand{arabic}{die}
\rand\arabic{die} \rand\arabic{die} \rand\arabic{die}
```

At last, random numbers between 1 and 12 will be generated and stored in the standard counter `rand`. The seed will be set to 1234. There will also be a warning because the name of the counter is set to `rand` which was already defined when calling the package:

```
\reinitrand[last=12, seed=1234]

\rand\Roman{rand} \rand\Roman{rand} \rand\Roman{rand}
```

These lines produce: XI V VII. When using other formats than `arabic` for printing, the desired numbers might not appear on the screen because these formats don't support the full range of $2^{31} - 1$.

3 The Linear Congruential Generator

The linear congruential generator used produces a sequence of numbers I_j in the range from 1 to m by following rule:

$$I_{j+1} = aI_j \mod m$$

where I_0 is set to a arbitrary starting value (called “seed”). The quality of this generator depends on the choice of the parameters a and m. Another problem is that when implementing the algorithm as above, the multiplication might leave the range LaTeX can deal with. The solution is Schrage’s method [W. Press et al. *Numerical Recipies in* C. 2nd edition. Cambridge University Press 1992] which allows to perform the multiplications without leaving the given range. This is done by introducing two variables r and q:

$$m = aq + r \qquad \text{with} \qquad q = [m/a] \quad \text{and} \quad r = m \mod a$$

where $[\,\cdot\,]$ denotes the integer part of the argument. If z is an integer and $0 \leq r < q$ and $0 < z < m - 1$, then the following (in)equalities hold:

$$\begin{aligned} 0 \leq a \cdot (z \mod q) &\leq m - 1 \\ 0 \leq [m/a] &\leq m - 1 \\ az \mod m &= \begin{cases} a \cdot (z \mod q) - [z/q] & \text{if the term is } \leq 0 \\ a \cdot (z \mod q) - [z/q] + m & \text{otherwise} \end{cases} \end{aligned}$$

To exploit the whole possible range and guarantee good performance, a and m are set as follows: $a = 7^5 \ = 16807$ and $m = 2^{31} - 1 \ = 2147483647$ and this gives $q = 127773$ and $r = 2836$.

4 The Code

4.1 Checking for possible conflicts

The following lines check if the commands provided by this package are already defined:

```
\@ifundefined{rand}{}
      {\PackageWarning{lcg}{Command ‘rand’ already defined}}
\@ifundefined{r@ndcountername}{}
      {\PackageWarning{lcg}{Command ‘r@ndcountername’
        already defined}}
```

Checking internal commands:

```
\@ifundefined{r@nd}{}
      {\PackageWarning{lcg}{Command ‘r@nd’ already defined}}
\@ifundefined{initr@nd}{}
      {\PackageWarning{lcg}{Command ‘initr@nd’ already defined}}
\@ifundefined{cutr@nger@nd}{}
```

```
       {\PackageWarning{lcg}{Command ‘cutr@nger@nd’ already defined}}
\@ifundefined{@rderr@nd}{}
       {\PackageWarning{lcg}{Command ‘@rderr@nd’ already defined}}
\@ifundefined{ProcessOptionsWithKVr@nd}{}
       {\PackageWarning{lcg}{Command ‘ProcessOptionsWithKVr@nd’
         already defined}}%
\@ifundefined{qui@t}{}
       {\PackageWarning{lcg}{Command ‘qui@t’ already defined}}
\@ifundefined{firstletterr@nd}{}
       {\PackageWarning{lcg}{Command ‘firstletterr@nd’ already defined}}

```

Checking the used counters:

```
\@ifundefined{c@f@rst}{}
       {\PackageWarning{lcg}{Counter ‘f@rst’ already defined}}
\@ifundefined{c@l@st}{}
       {\PackageWarning{lcg}{Counter ‘l@st’ already defined}}
\@ifundefined{c@cr@nd}{}
       {\PackageWarning{lcg}{Counter ‘cr@nd’ already defined}}
\@ifundefined{f@rst}{}
       {\PackageWarning{lcg}{Existing command ‘f@rst’ conflicts
         with counter ‘f@rst’}}
\@ifundefined{l@st}{}
       {\PackageWarning{lcg}{Existing command ‘l@st’ conflicts
         with counter ‘l@st’}}
\@ifundefined{cr@nd}{}
       {\PackageWarning{lcg}{Existing command ‘cr@nd’ conflicts
         with counter ‘cr@nd’}}
```

4.2 Macros for (re)initialization

`init` Set starting values for the parameters and counters to standard values or according to the provides keys

```
\def\initr@nd{%
   \def\r@ndcountername{rand}%
   \newcount \f@rst
   \newcount \l@st
   \newcount \cr@nd
   \pr@keysr@nd%
   \ProcessOptionsWithKVr@nd{Init}%
   \p@stkeysr@nd%
   \@utputr@nd%
}  % end of \def\initr@nd
```

`reinit` Sets the provided keys and resets all other options.

```
\def\reinitrand{\@ifnextchar[\@reinitr@nd{\@reinitr@nd[]}}%
\def\@reinitr@nd[#1]{%
    \pr@keysr@nd%
    \setkeys{Init}{#1}%
    \p@stkeysr@nd%
    \@utputr@nd%
}%    end of \def\reinitrand
```

chgrand

Sets the provided keys and doesn't change any other option.

```
\def\chgrand{\@ifnextchar[\@chgr@nd{\@chgr@nd[]}}
\def\@chgr@nd[#1]{%
    \@tempcnta = \z@
    \@tempcntb = \z@
    \setkeys{Init}{#1}%
    \p@stkeysr@nd%
    \@utputr@nd%
}  % end of \def\chgrand
```

4.3 The keys

use keys

The following lines are from the `geometry` package written by Hideo Umeki (who borrowed it from the `hyperref` package written by Sebastian Rahtz). It enables the usage of the ⟨*key*⟩=⟨*value*⟩ scheme of the keyval package.

```
\def\ProcessOptionsWithKVr@nd#1{%
  \let\@tempa\@empty
  \@for\CurrentOption:=\@classoptionslist\do{%
    \@ifundefined{KV@#1@\CurrentOption}%
    {}{\edef\@tempa{\@tempa,\CurrentOption,}}}
  \edef\@tempa{%
    \noexpand\setkeys{#1}{\@tempa\@ptionlist{\@currname.\@currext}}}
  \@tempa
  \AtEndOfPackage{\let\@unprocessedoptions\relax}}
```

first

```
\define@key{Init}{first}[1]{\f@rst = #1}
```

last

```
\define@key{Init}{last}[2147483647]{\l@st = #1}
```

counter

```
\define@key{Init}{counter}[rand]{\def\r@ndcountername{#1}}
```

seed

```
\define@key{Init}{seed}[\z@]{% seed for random number generator
   \ifnum #1 < \z@%
      \PackageWarning{lcg}{Seed should be > 0 --
              Seed will be initialized with the actual time}%
      \cr@nd = \z@%
   \else%
      \cr@nd = #1
      \typeout{Random number generator initialized to #1}%
   \fi%
}
```

`quiet`

```
\define@key{Init}{quiet}[y]{
   \def\qui@t{\expandafter\firstletterr@nd #1\delimiter}
   \if \qui@t y% nothing to do
   \else\if\qui@t Y \def\qui@t{y}
   \else\if\qui@t j \def\qui@t{y}
   \else\if\qui@t J \def\qui@t{y}
   \else\if\qui@t n \def\qui@t{n}
   \else\if\qui@t N \def\qui@t{n}
   \else
      \PackageWarning{lcg}{Value of key <quiet> must be <y> or <n>}
      \def\qui@t{y}
   \fi\fi\fi\fi\fi\fi
}
```

4.4 Macros called by other macros

`pr@keys` The command `\pr@keysr@nd` is used to define and initialize all parameters (counters) needed by this package (before the keys are evaluated). Random numbers will be generated from `f@rst` to `f@rst` + `l@st` − 1, `cr@nd` will hold the random numbers (full range: 1 to $2^{31} - 1$) and `rand` will hold the random numbers (user defined range). The counters are also initialized to standard values. If the counter `cr@nd` equals zero, the seed will be initialized according to the actual time by the command `\r@nd`:

```
\def\pr@keysr@nd{%
    \f@rst = \@ne        % 1
    \l@st = 2147483647   % 2147483647
    \cr@nd  = \z@        % 0
    \@tempcnta = \z@
    \@tempcntb = \z@
    \def\r@ndcountername{rand}%
    \def\qui@t{n}
} % end of newcommand\def\pr@keysr@nd
```

`p@stkeys` The command `\p@stkeysr@nd` is executed after the keys are evaluated as last step of the initialization. The setting of the counter `l@st` depends on weather the key ⟨*last*⟩ is set or not. and the counter (user defined or standard name) is created.

```
\def\p@stkeysr@nd{%
   \@rderr@nd%                  last < first  -> swap
   \cutr@nger@nd%               range too big -> cut
   \@ifundefined{c@\r@ndcountername}{\newcounter{\r@ndcountername}}%
   {%
      \PackageWarning{lcg}{Using an already existing
           counter \r@ndcountername}%
   }%
\setcounter{\r@ndcountername}{0}%
} % end of \def\p@stkeysr@nd
```

`firstletter` This macro is used to determine the first letter of the value of the key ⟨*quiet*⟩.

```
\def\firstletterr@nd#1#2\delimiter{#1}
```

`@utput` Output to log-file/screen

```
\def\@utputr@nd{%
   \if \qui@t y% do nothing
   \else
     \typeout{Smallest possible random number: \the\f@rst}%
     \typeout{Largest possible random number:  \the\l@st}%
     \typeout{The pseudo random numbers will be stored
      in the LaTeX counter `\r@ndcountername'}%
   \fi
}
```

`@rder` If the value of the key ⟨*last*⟩ is less than the value of ⟨*first*⟩, they will be exchanged.

```
\def\@rderr@nd{%
   \ifnum \l@st < \f@rst%
       \PackageWarning{lcg}{Key `last' less than key `first'
               -- swapped}%
       \@tempcnta = \f@rst
       \f@rst = \l@st
       \l@st = \@tempcnta
   \fi%
}%  end of \def\@rderr@nd
```

`cutr@nge` If the given range of random numbers exceeds the possibilities of LaTeX (the limit is $2^{31}-1$), then the value of the TeX-counter `@tempcnta` will be less than zero and the right border will be adjusted.

```
\def\cutr@nger@nd{%
    \ifnum\l@st<\z@\else
    \@tempcntb =  -2147483646   % -2^31 + 2
    \@tempcnta = \f@rst
    \advance \@tempcntb \l@st
    \multiply \@tempcntb \m@ne
    \advance \@tempcnta \@tempcntb
    \ifnum \@tempcnta < \z@%
       \PackageWarning{lcg}{Range contains too many numbers
                       -- right border reset to largest possible value}%
       \advance \l@st \@tempcnta
    \fi%
    \fi%
}%  end of \checkr@ange
```

4.5 Macros for random number generation

`rand` The command `\rand` calls the internal command `\r@nd` which stores s random number (full range) within the counter `cr@nd`. If the condition

$$\texttt{cr@nd} \le (\texttt{l@st} - \texttt{f@rst} + 1) \cdot \frac{2^{31} - 1}{\texttt{l@st} - \texttt{f@rst} + 1}$$

holds, `cr@nd` will be transformed to the given range:

$$\texttt{f@rst} + \texttt{cr@nd} - (\texttt{l@st} - \texttt{f@rst} + 1) \cdot \frac{\texttt{cr@nd}}{\texttt{l@st} - \texttt{f@rst} + 1}$$

and the result stored in the corresponding counter and otherwise `\rand` calls itself till the condition is satisfied. It's important to notice that the result of the division of two integers is again an integer (the fraction part is lost)!

```
\def\rand{%
   \r@nd%
   \@tempcnta
   \@tempcntb
   \@tempcnta = \f@rst
   \@tempcntb = \l@st
   \multiply \@tempcnta \m@ne
   \advance \@tempcntb \@tempcnta
   \advance \@tempcntb \@ne         %l@st-f@rst+1
   \@tempcnta = 2147483647
   \divide \@tempcnta \@tempcntb
   \multiply \@tempcnta \@tempcntb
   \ifnum \cr@nd > \@tempcnta
      \rand%
   \else
      \setcounter{\r@ndcountername}{\cr@nd}%
      \@tempcnta = \cr@nd
      \divide \@tempcnta \@tempcntb
      \multiply \@tempcnta \@tempcntb
      \multiply \@tempcnta \m@ne
      \addtocounter{\r@ndcountername}{\@tempcnta}%
      \addtocounter{\r@ndcountername}{\f@rst}%
   \fi
} % end of \rand
```

`r@nd` The command `\r@nd` generates pseudo random numbers within the range 1 to $2^{31} - 1$ Schrage's method and stores them in the counter `cr@nd`:

```
\def\r@nd{%
   \ifnum \cr@nd < \@ne%       then  ... initialize generator
      \cr@nd =  \the\time
      \advance  \cr@nd \inputlineno
      \multiply \cr@nd \value{page}
      \advance  \cr@nd \the\year
      \multiply \cr@nd \the\month
      \multiply \cr@nd \the\day
      \advance  \cr@nd \inputlineno
      \if \qui@t y%
      \else
        \typeout{Random number generator initialized to \the\cr@nd}%
      \fi
      \r@nd%
   \else                         %  else ... generate new number
      \@tempcnta = \cr@nd
      \divide \@tempcnta  127773    % \@tempcnta = floor(z/q)
      \@tempcntb = \@tempcnta       % \@tempcntb = floor(z/q)
      \multiply \@tempcnta  -2836   % \@tempcnta = -r*floor(z/q)
      \multiply \@tempcntb -127773  % \@tempcntb = -q*floor(z/q)
      \advance \cr@nd \@tempcntb    % cr@nd = z mod q
      \multiply \cr@nd 16807        % cr@nd = a * (z mod q)
      \advance \cr@nd \@tempcnta    % cr@nd = a*z mod m
      \ifnum \cr@nd < \z@%
         \advance \cr@nd 2147483647 % cr@nd = (a*z mod m) > 0
      \fi
      \global\cr@nd=\cr@nd % persist the change outside current scope
   \fi
}%        end of \r@nd
```

4.6 Initialization

```
\initr@nd      % initialize the package
```