The ran_toks Package

D. P. Story
Email: dpstory@uakron.edu

processed December 30, 2019

Contents

1 ⟨*package⟩

Description. This short package randomizes a list of tokens. The command, `\ranToks`, takes one argument, which is a list of tokens:

```
\ranToks{⟨name⟩}
{
    {⟨tok1⟩}{⟨tok2⟩}...{⟨tokn⟩}
}
```

The command defines a series of n internal commands, one for each of the tokens. The definitions are essentially randomized. The randomized tokens are accessed through the command `\useRanTok`. For example

```
\useRanTok{1}, \useRanTok{2},..., \useRanTok{n}
```

gives a random listing of the n tokens. These can be arranged on the page as desired.

There is a second construct, designed for more elaborate randomization.

```
\bRTVToks{⟨name⟩}
\begin{rtVW}
    ⟨some content⟩
\end{rtVW}
...
...
\begin{rtVW}
    ⟨some content⟩
\end{rtVW}
\eRTVToks
```

The contents of each of the `rtVW` environments are written to the computers hard drive, then input back in random order, using `\useRanTok`, eg,

```
\useRanTok{1}, \useRanTok{2},..., \useRanTok{n}
```

Other details are left to the readers' imagination.

Requirements. As of this writing, we require only the `verbatim` package and `random.tex`, the package was written by Donald Arseneau.

```
\RequirePackage{verbatim}
```

Input `random.tex`. Input `random.tex` if not already input.

```
\@ifundefined{nextrandom}{\input{random.tex}}{}
```

We redefine `\nextrandom` from `random.tex` to save the initializing seed.

```
\def\nextrandom{\begingroup
  \ifnum\randomi<\@ne % then initialize with time
    \global\randomi\time
    \global\multiply\randomi388 \global\advance\randomi\year
    \global\multiply\randomi31 \global\advance\randomi\day
    \global\multiply\randomi97 \global\advance\randomi\month
    \message{Randomizer initialized to \the\randomi.}%
    \nextrandom \nextrandom \nextrandom
```

Save the initial seed value to `\rtInitSeedValue`.

```
    \xdef\InitSeedValue{\the\randomi}%
  \fi
  \count@ii\randomi
  \divide\count@ii 127773 % modulus = multiplier * 127773 + 2836
  \count@\count@ii
  \multiply\count@ii 127773
  \global\advance\randomi-\count@ii % random mod 127773
  \global\multiply\randomi 16807
  \multiply\count@ 2836
  \global\advance\randomi-\count@
  \ifnum\randomi<\z@ \global\advance\randomi 2147483647\relax\fi
  \endgroup
}
```

The code for this package was taken from the dps package, and modified suitably. We use several token registers and count registers. This can probably be optimized.

```
\newtoks\rt@listIn \rt@listIn={}
\newtoks\rt@newListIn \rt@newListIn={}
\newtoks\rt@listOut \rt@listOut={}
\newcount\rt@nMax
\newcount\rt@nCnt
\newcount\rt@getRanNum
\newif\ifrtdebug \rtdebugfalse
\newif\ifwerandomize \werandomizetrue
\newif\ifsaveseed\saveseedtrue
\newwrite\rt@Verb@write
```

```
\def\rt@nameedef#1{\expandafter\edef\csname #1\endcsname}
```

⟨/package⟩
⟨*altpkgname⟩

1 Alternate package name: ran-toks

CTAN lists this package (ran_toks) as ran-toks, so we'll create a dummy package by that name.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{ran-toks}
 [2019/12/28 v1.0 ran-toks Alt-name (dps)]
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{ran_toks}}
\ProcessOptions
\RequirePackage{ran_toks}[2019/12/28]
```

⟨/altpkgname⟩
⟨*package⟩

2 Commands for controlling the process

`\ranToksOn` `\ranToksOff` These two turn on and turn off randomization.

```
\def\ranToksOn{\werandomizetrue}
\def\ranToksOff{\werandomizefalse}
```

`\useThisSeed` initializes the random number generator. Use this to reproduce the same sequence of pseudo-random numbers from an earlier run. We also set `\saveseedfalse` so we do not write the initial seed to the disk.

```
\def\useThisSeed#1{\saveseedfalse\randomi=#1}
\@onlypreamble\useThisSeed
```

`\useLastAsSeed` initializes the random number generator using the last random seed. If the file `\jobname_rt.sav` does not exist, the generator will be initialized using time and date data.

```
\def\useLastAsSeed{\rt@useLastAsSeed}
\@onlypreamble\useLastAsSeed
```

```
\def\rt@useLastAsSeed{%
  \IfFileExists{\jobname_rt.sav}{%
    \PackageInfo{ran_toks}{Inputting \jobname_rt.sav}%
    \@ifundefined{readsavfile}{\newread\readsavfile}{}%
    \openin\readsavfile=\jobname_rt.sav
    \read\readsavfile to \InitSeedValue
    \read\readsavfile to \lastRandomNum
    \closein\readsavfile
    \randomi=\lastRandomNum
```

When `\useLastAsSeed`, the last becomes the first.

```
        \xdef\InitSeedValue{\the\randomi}
  }{%
    \PackageInfo{ran_toks}{\jobname_rt.sav cannot
      be found, \MessageBreak
      using the random initializer}%
  }%
}
\@ifundefined{aeb@randomizeChoices}{%
    \let\inputRandomSeed\useLastAsSeed
    \let\useRandomSeed\useThisSeed}{}
```

3 Utility commands

A standard `\verbatim` write used in exerquiz and other package in the AeB family.

```
\def\verbatimwrite{\@bsphack
  \let\do\@makeother\dospecials
  \catcode`\^^M\active \catcode`\^^I=12
  \def\verbatim@processline{%
    \immediate\write\verbatim@out
    {\the\verbatim@line}}%
  \verbatim@start}
\def\endverbatimwrite{\@esphack}
\def\rt@IWVO{\immediate\write\verbatim@out}
```

We write only if `\ifsaveseed` is true.

```
\def\InitSeedValue{\the\randomi}
\def\rt@writeSeedData{\ifsaveseed
  \@ifundefined{saveseedinfo}{\newwrite\saveseedinfo}{}
  \immediate\openout \saveseedinfo \jobname_rt.sav
  \let\verbatim@out\saveseedinfo
  \def\rt@msgi{initializing seed value}%
  \def\rt@msgii{last random number used}%
  \uccode`c=`\%\uppercase{%
  \rt@IWVO{\InitSeedValue\space c \rt@msgi}%
  \rt@IWVO{\the\randomi\space c \rt@msgii}}\immediate
  \closeout\saveseedinfo\fi}
```

Save the initial seed value to hard drive.

```
\AtEndDocument{\rt@writeSeedData}%
```

`\rt@populateList{`⟨*n*⟩`}` is a utility command, its argument ⟨*n*⟩ is a positive integer, and it generates a list of the form `\\{1}\\{2}...\\{n}` and is held in the token register `\rt@listIn` This listing is later randomly permuted by `\rt@RandomizeList`.

```
\def\rt@populateList#1{\rt@listIn={}\rt@nCnt\z@
  \@whilenum\rt@nCnt<#1\do{\advance\rt@nCnt\@ne
    \edef\rt@listInHold{\the\rt@listIn\noexpand\\{\the\rt@nCnt}}%
    \rt@listIn=\expandafter{\rt@listInHold}}}
```

`\rt@RandomizeList {`⟨*n*⟩`}` is the command that gets the process of randomizing the input list going. The argument is the number ⟨*n*⟩ of tokens. If `\werandomize` is false, it just returns the input list; otherwise, it calls `\rt@randomizeList` to actually do the work.

```
\def\rt@RandomizeList#1{\global
  \rt@listIn={}\global\rt@newListIn={}\global\rt@listOut={}%
  \rt@nMax=#1\relax\rt@populateList{\the\rt@nMax}%
  \ifwerandomize
    \expandafter\rt@randomizeList\else
    \global\rt@listOut=\expandafter{\the\rt@listIn}\fi
```

Save the list out as `\rt@BaseName-List` for later retrieval. This is the randomized list of integers for this base name.

```
  \global\rt@nameedef{\rt@BaseName-List}{\the\rt@listOut}}
```

`\rt@randomizeList` randomizes the list of consecutive integers, and leaves the results,

```
\\{k_1}\\{k_2}...\\{k_n}
```

in the token register `\rt@listOut`. `\rt@randomizeList` is a loop, looping between itself and `\rt@loopTest`.

```
\def\rt@randomizeList{\let\\=\rt@processi
  \setrannum{\rt@getRanNum}{1}{\the\rt@nMax}%
\ifrtdebug\typeout{\string\rt@getRanNum=\the\rt@getRanNum}\fi
  \rt@nCnt\z@
\ifrtdebug\typeout{LISTING: \the\rt@listIn}\fi
  \the\rt@listIn
  \rt@loopTest
}
\def\rt@loopTest{\advance\rt@nMax\m@ne
  \ifnum\rt@nMax>\z@
    \def\rt@next{%
      \rt@listIn=\expandafter{\the\rt@newListIn}%
      \rt@newListIn={}\rt@randomizeList}%
  \else
    \let\rt@next\relax
    \global\rt@listOut=\expandafter{\the\rt@listOut}%
    \ifrtdebug
      \typeout{Final Result: \string\rt@listOut=\the\rt@listOut}\fi
  \fi\rt@next
}
```

In `\rt@randomizeList`, we `\let\\=\rt@processi` before dumping the contents of `\rt@listIn`. We then go into a loop `\rt@loopTest`. `\rt@getRanNum` is the random integer between 1 and `\rt@nMax`.

```
\def\rt@processi#1{\advance\rt@nCnt\@ne
  \ifnum\rt@nCnt=\rt@getRanNum
    \edef\rt@listOutHold{\the\rt@listOut}%
    \global\rt@listOut=\expandafter{\rt@listOutHold\\{#1}}%
    \ifrtdebug\typeout{Found it: \string\\{#1}}%
    \typeout{New \string\rt@listOut: \the\rt@listOut}\fi
  \else
    \edef\rt@listInHold{\the\rt@newListIn}%
      \rt@newListIn=\expandafter{\rt@listInHold\\{#1}}%
    \ifrtdebug\typeout{\string\rt@newListIn: \the\rt@newListIn}\fi
  \fi
}
```

We perform modular arithmetic when the index of `\useRanTok` is too large.

`\rt@modarith` `\rt@modarith` performs modular arithmetic on its arguments (`#1 mod #2`) and returns the result in the macro `\rt@mod`.

```
\def\rt@modarith#1#2{\count\z@=#1\relax\count\tw@=#1\relax
```

This macro uses `\dimen0` and `\dimen2`, so it should be called within a group.

```
  \advance\count\z@\m@ne\divide\count\z@ #2\relax
  \multiply\count\z@ #2\relax
  \advance\count\tw@-\count\z@
  \edef\rt@mod{\the\count\tw@}}
```

`\rt@badIndex` Warning messages, these are `\rt@badIndex` and `\rt@badTokName`.

`\rt@badTokName`

```
\def\rt@badIndex#1#2{\PackageWarningNoLine{ran_toks}
    {The argument of \string\useRanTok{#1} on line
      \the\inputlineno\space is\MessageBreak
        greater than \string\nToksFor{#2} (\nToksFor{#2}),
        instead will use\MessageBreak
        \string\useRanTok{\rt@mod}, obtained from modular
        arithmetic.\MessageBreak
        You might want to fix this}
}
\def\rt@badTokName#1{%
  \PackageWarningNoLine{ran_toks}
    {The token list `#1' on line \the\inputlineno\space
      is undefined,\MessageBreak
      possibly simply mispelled; check spelling.\MessageBreak
      If undefined, use \string\ranToks\space or \string\bRTVToks/%
      \string\eRTVToks\space\MessageBreak
      to define a list with the name `#1'}%
}
\def\rt@warnTokName#1{%
  \PackageWarningNoLine{ran_toks}
    {The token list `#1' on line \the\inputlineno\space
      is already defined,\MessageBreak
      will overwrite this list}%
}
```

4 The main commands

`\ranToks{`⟨*token-list*⟩`}` takes one argument, `{`⟨*token-list*⟩`}`, a list of tokens. It randomizes them. The randomized listing can be accessed using `\useRanTok`.

```
\def\ranToks#1{\begingroup
  \useRTName{#1}%
  \r@nToks
}
\long\def\r@nToks#1{\rt@nMax\z@\r@ndToks#1\rt@NIL}
\def\rt@NIL{@nil}
```

`\useRTName{`⟨*name*⟩`}` sets the base name (use prior to the use of `\useRanTok`).

```
\newcommand{\useRTName}[1]{\gdef\rt@BaseName{#1}}%
\let\rt@BaseName\@empty
```

`\bRTVToks{`⟨*name*⟩`}` `\bRTVToks` and `\eRTVToks` enclose a series of `rtVW` environments. The single argument is the name of this set of verbatim write "tokens".

```
\newcommand{\bRTVToks}[1]{\rt@nCnt\z@\useRTName{#1}}
```

`\eRTVToks` At the end of the `rtVW` environments, initiated by `\bRTVToks`, the `\eRTVToks` command saves the number of tokens counted, and randomizes the access to the contents of the `rtVW` environments, this done by `\r@nVToks`.

```
\newcommand{\eRTVToks}{\global
  \rt@nameedef{\rt@BaseName Cnt}{\the\rt@nCnt}%
  \expandafter\r@nVToks\expandafter{\rt@BaseName}}
```

`rtVW` `\rtVW` is a verbatim write environment. It saves its contents to the file `\jobname_\rt@BaseName\the\rt@nCnt.cut`. The file is later input back into the source file in a random way.

```
\def\reVerbEnd{\ifhmode\unskip\fi}
```

Insert the hook `\rtVWHook` prior to writing the verbatim content. The default is `\relax`.

```
\def\rtVWHook#1{\def\@rgi{#1}\ifx\@rgi\@empty
  \let\RTVWHook\relax\else\def\RTVWHook{#1}\fi}
\rtVWHook{}
\newenvironment{rtVW}{\global\advance\rt@nCnt\@ne
  \immediate\openout\rt@Verb@write
    \jobname_\rt@BaseName\the\rt@nCnt.cut
  \let\verbatim@out\rt@Verb@write
  \rt@IWVO{\string\RTVWHook}%
  \verbatimwrite
}{%
  \endverbatimwrite
  \immediate\write\rt@Verb@write{\string\reVerbEnd}%
  \immediate\closeout\rt@Verb@write
}
```

`\r@nVToks` randomizes the contents of the `rtVW` environment.

```
\def\r@nVToks#1{\begingroup
  \gdef\rt@BaseName{#1}%
  \expandafter\rt@nMax\@nameuse{#1Cnt}%
  \rt@listIn={}\rt@nCnt=0\relax\let\rt@listInHold\@empty
  \@whilenum\rt@nCnt<\rt@nMax\do{\advance\rt@nCnt\@ne
    \edef\rt@listInHold{%
      \the\rt@listIn{\noexpand\rt@inputVerb{#1\the\rt@nCnt}}}%
    \rt@listIn=\expandafter{\rt@listInHold}}\ifrtdebug
  \typeout{\string\r@nVToks: \the\rt@listIn}\fi
  \expandafter\r@nToks\expandafter{\the\rt@listIn}}
\def\rt@inputVerb#1{\input{\jobname_#1.cut}}
```

`\r@ndToks` is main looping command for `\ranToks` and `\eRTVToks` (through `\r@nVToks`). If the ending token `\rt@NIL` is detected, we break off and go to `\rt@endToks`.

```
\def\rt@PAR{\par}
\long\def\r@ndToks#1{\def\rt@rgi{#1}%
```

If the current argument is `\par`, we skip it

```
  \ifx\rt@rgi\rt@PAR\def\rt@next{\r@ndToks}\else
    \advance\rt@nMax\@ne
    \global\@namedef{rtTok\the\rt@nMax\rt@BaseName}{#1}%
    \def\rt@next{\@ifnextchar\rt@NIL
      {\rt@endToks\@gobble}{\r@ndToks}}\fi\rt@next}
```

`\rt@performRanDefns{`⟨*n*⟩`}` The `\rt@performRanDefns` performs code that is repeated in several other macros: `\rt@endToks`, `\reorderRanToks`, and `\copyRanToks`. It randomizes the list `\rt@RandomizeList`, then assignments the randomized list to the definitions.

```
\def\rt@performRanDefns#1{%
```

Now we randomize the order of the integers 1, 2,... `#1`.

```
  \rt@RandomizeList{#1}\rt@nCnt\z@
```

Now we randomize the definitions. We `\let\\=\rt@ssign`, then let loose the tokens!

```
  \let\\\rt@ssign\the\rt@listOut}
```

`\rt@endToks` The final destination for `\r@ndToks`.

```
\def\rt@endToks{\global
```

Save the number of tokens counted

```
  \rt@nameedef{nMax4\rt@BaseName}{\the\rt@nMax}%
  \rt@performRanDefns{\the\rt@nMax}\endgroup}
```

`\reorderRanToks{`⟨*name*⟩`}` The `\reorderRanToks` command reorders (or re-indexes) the family with name ⟨*name*⟩ (`#1`).

```
\def\reorderRanToks#1{\begingroup\useRTName{#1}\expandafter
  \ifx\csname nMax4#1\endcsname\relax
```

Document author has not run `\ranToks` yet for this basename (`#1`)

```
    \rt@badTokName{#1}\else
```

Good to go. We reorder this list.

```
    \rt@performRanDefns{\@nameuse{nMax4#1}}\fi
\endgroup}
```

`\copyRanToks{`⟨*name1*⟩`}{`⟨*name2*⟩`}` Use this command to copy ⟨*name1*⟩ to ⟨*name2*⟩. This gives a randomization of the same list, without affecting the original order of ⟨*name1*⟩.

```
\newcommand\copyRanToks[2]{\begingroup\expandafter
  \ifx\csname nMax4#1\endcsname\relax
```

Source list is not defined

```
      \rt@badTokName{#1}%
  \else\expandafter
```

Source list is defined

```
    \ifx\csname nMax4#2\endcsname\relax
```

Destination list is not defined, which is good in this instance. This is the case we copy the list.

```
      \useRTName{#2}\global
      \rt@nameedef{nMax4#2}{\@nameuse{nMax4#1}}%
      \rt@nCnt=\csname nMax4#2\endcsname\relax
      \@whilenum\rt@nCnt>\z@\do{\global
        \rt@nameedef{rtTok\the\rt@nCnt#2}%
          {\noexpand\@nameuse{rtTok\the\rt@nCnt#1}}%
        \advance\rt@nCnt\m@ne}%
      \rt@performRanDefns{\@nameuse{nMax4#2}}\else
```

Destination list is defined already, warn the user.

```
      \rt@warnTokName{#2}\fi
  \fi
\endgroup}
```

`\rt@ssign{`⟨*name*⟩`}` makes the assignments that are expanded by `\useRanTok`. We `\let` the assignment `\let\\=\rt@ssign` in `\rt@endToks`, just before we dump out the contents of `\the\rt@listOut`.

```
\def\rt@ssign#1{\advance\rt@nCnt\@ne\global
  \rt@nameedef{rtRanTok\the\rt@nCnt\rt@BaseName}{\noexpand
  \@nameuse{rtTok#1\rt@BaseName}}}
```

4.1 Additional user access commands

`\nToksFor{`⟨*name*⟩`}` expands the the number of tokens whose name is ⟨*name*⟩ (`#1`).

```
\newcommand{\nToksFor}[1]{\expandafter
  \ifx\csname nMax4#1\endcsname\relax
    \textbf{??}\rt@badTokName{#1}\else
    \@nameuse{nMax4#1}\fi}
```

`\rtTokByNum[`⟨*name*⟩`]{`⟨*num*⟩`}` is an internal macro, but it can be used publicly. The argument of it is an integer, eg, `\rtTokByNum{3}` is the third token, as listed in the order given in the argument of `\ranToks`.

```
\newcommand{\rtTokByNum}[2][\rt@BaseName]{\expandafter
  \ifx\csname nMax4#1\endcsname\relax
    \textbf{??}\rt@badTokName{#1}\else
    \@nameuse{rtTok#2#1}\expandafter\ignorespaces\fi}
```

`\useRanTok[`⟨*name*⟩`]{`⟨*num*⟩`}` After `\ranToks` has been executed, the user has access to the randomized tokens through `\useRanTok`. The argument ⟨*num*⟩ is an integer 1 through max.

```
\newcommand{\useRanTok}[2][\rt@BaseName]{\bgroup
  \expandafter\ifx\csname nMax4#1\endcsname\relax
    \rt@badTokName{#1}\else
    \ifnum#2>\nToksFor{#1}\rt@modarith{#2}{\nToksFor{#1}}%
```

If index (`#2`) is greater than array length, use modular arithmetic to resolve the issue, and send a warning to the user.

```
      \rt@badIndex{#2}{#1}\@nameuse{rtRanTok\rt@mod#1}\else
      \@nameuse{rtRanTok#2#1}\fi\fi\egroup}
```

`\displayListRandomly[`⟨*prior*⟩`][`⟨*post*⟩`]{`⟨*name*⟩`}` lists all items in the list as passed by the required argument. For expanding in a list environment, use `\item` as the optional argument. Designed for listing all question in an eqexam document in random order.

```
\newcommand{\displayListRandomly}[1][]{\bgroup\def\rt@prior{#1}%
  \displ@yListRandomly}
\newcommand{\displ@yListRandomly}[2][]{\rt@nCnt\z@
  \expandafter\ifx\csname nMax4#2\endcsname\relax
    \rt@rgi\space\textbf{??}\rt@badTokName{#2}#1%
  \else
```

`\i` `\first` `\last` `\lessone` Within the optional arguments, we define `\i`, `\first`, `\last`, and `\lessone` to do some logic on the arguments. These four macro are defined locally and not available outside the command `\displayListRandomly`.

```
    \def\rt@post{#1}\useRTName{#2}\let\i\rt@nCnt
    \def\first{1}\edef\last{\@nameuse{nMax4#2}}%
    \@tempcnta\last \advance\@tempcnta\m@ne
    \edef\lessone{\the\@tempcnta}%
    \@whilenum\rt@nCnt<\last\advance\rt@nCnt\@ne
    \do{\rt@prior\useRanTok{\the\rt@nCnt}\rt@post}%
  \fi
\egroup}
```

5 Commands that support a DB application

We begin with some utility commands to help parse the argument of `\useProbDBs`.

```
\def\rt@gettonil#1\@nil{\def\to@nilarg{#1}}
\def\rt@ifspc{\ifx\@let@token\@sptoken
  \let\rt@next\rt@xifspc\else
  \let\rt@next\rt@gettonil\fi\rt@next}
\begingroup
\def\:{\rt@xifspc}
```

```
\expandafter\gdef\: {\futurelet\@let@token\rt@ifspc}
\endgroup
\def\rt@strpspcs{\futurelet\@let@token\rt@ifspc}
```

`\useTheseDBs{`⟨*list*⟩`}` Inputs any files included in the comma-delimited list. The base names need only be listed, as the extension is assumed to be `.tex`. The command `\useProbDBs` can only be used in the preamble. Refer to the demo file `mc_db.tex` for an illustration of its intended use.

```
\def\ProbDBWarningMsg#1{\filename@parse{#1}
  \PackageWarning{ran_toks}
  {The file \filename@area\filename@base.\ifx\filename@ext\relax
      tex\else\filename@ext\fi\space cannot be found}}
\def\useTheseDBs#1{\def\rt@dblist{#1}\ifx\rt@dblist\@empty\else
  \let\rt@DB@List\@empty
  \edef\temp@expand{\noexpand\@for\noexpand\@@tmp:=\rt@dblist}%
  \temp@expand\do{\ifx\@@tmp\@empty\else
    \expandafter\rt@strpspcs\@@tmp\@nil\edef\@@tmp{\to@nilarg}%
    \edef\rt@nextDB{\noexpand
      \InputIfFileExists{\@@tmp}{}{\noexpand
      \ProbDBWarningMsg{\@@tmp}}}%
    \toks\tw@=\expandafter{\rt@DB@List}%
    \toks@=\expandafter{\rt@nextDB}%
    \edef\rt@DB@List{\the\toks\tw@\space\the\toks@}\fi
  }\expandafter\rt@DB@List\fi}
\let\useProbDBs\useTheseDBs
```

⟨/package⟩

6 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

S

T

U

V

W

Y

7 Change History