The ran_toks Package

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

processed June 6, 2021

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⟩}{%
  {⟨tok_1⟩}{⟨tok_2⟩}...{⟨tok_n⟩}
}
```

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
\newif\ifrt@InputUsedIDs\rt@InputUsedIDsfalse
\newwrite\rt@Verb@write
```

Convenience commands.

```
\def\rtcsarg#1#2{\expandafter#1\csname#2\endcsname}
\def\rt@nameedef#1{\expandafter\edef\csname #1\endcsname}
```

`usedbapp` The code to support a DB application has grown, so much so, it desirves a option so as to include the code only if needed.

```
\DeclareOption{usedbapp}{\let\rtPkgInpt\rt@PkgInpt}
\def\rt@PkgInpt{\InputIfFileExists{rt-dbapp.def}
  {\PackageInfo{ran_toks}{Inputting rt-dbapp.def}}
  {\PackageInfo{ran_toks}{Cannot find rt-dbapp.def}}
}
\let\rtPkgInpt\relax
\AtEndOfPackage{\rtPkgInpt}
\ProcessOptions\relax
```

```
⟨/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
}
```

`\rt@modarith`

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

Warning messages, these are `\rt@badIndex` and `\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{}
\newwrite\wrtprobids
\newif\ifviewIDs\viewIDsfalse
\newif\ifxDBUnique\xDBUniquefalse
\def\wrtProbIds#1{\immediate\write\wrtprobids{\string
  \rtcsarg\string\gdef{#1}{used}}}
\def\rtVWId#1{\ifviewIDs\noindent#1\fi
  \ifxDBUnique\ifrt@InputUsedIDs\wrtProbIds{#1}\fi\fi
}
```

```
\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}%
  \rt@IWVO{\string
    \rtVWId{\rt@BaseName-\the\rt@nCnt}\string\relax}%
  \verbatimwrite
}{%
  \endverbatimwrite
  \immediate\write\rt@Verb@write{\string\reVerbEnd}%
  \immediate\closeout\rt@Verb@write
}
```

(2021/05/29) In support of nested `\bRTVToks/\eRTVToks` constructs, we `\let` `\rtVWi` and `\rtVWii` to `\rtVW`, and `\endrtVWi` and `\endrtVWii` to `\endrtVW`.

```
\let\rtVWi\rtVW % dps5-29
\let\endrtVWi\endrtVW
\let\rtVWii\rtVW % dps5-29
\let\endrtVWii\endrtVW
```

`\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}}}% J14
    \rt@listIn=\expandafter{\rt@listInHold}}\ifrtdebug
  \typeout{\string\r@nVToks: \the\rt@listIn}\fi
  \expandafter\r@nToks\expandafter{\the\rt@listIn}}
```

`\rt@inputVerb{`⟨*db-name*⟩`-`⟨*num*⟩`}` This is the command that inputs a DB Test problem, it inputs it from the file named `\jobname_{`⟨*db-name*⟩`-`⟨*num*⟩`}`. As we input, we make a record of the problem we input by expanding `\rt@recordAsUsed{`⟨*db-name*⟩`-`⟨*num*⟩`}`, which itself expands to \rtcsarg\gdef{⟨*db-name*⟩-⟨*num*⟩}{used}. This is necessary when we are choosing more than one item from a given DB Test file; it must be recorded immediately so that later it cannot be used again, if possible.

```
\def\rt@inputVerb#1{\rt@recordAsUsed{#1}\input{\jobname_#1.cut}}
%\def\rt@recordAsUsed#1{\ifxDBUnique\rtcsarg\gdef{#1}{used}\fi}
\def\rt@recordAsUsed#1{\rtcsarg\gdef{#1}{used}}
```

\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
```

Source list is defined

```
    \expandafter
    \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.

`\uniqueXDBChoicesOn`
`\uniqueXDBChoicesOff`
`\makeInfoAWarning`

We provide two commands to control the feature of try to select unique choices across multiple renditions of the same source file. `\uniqueXDBChoicesOn`, turns on this feature; the default is `\uniqueXDBChoicesOff` make no changes to how `\useRanTok` operates. One other command we define here is `\makeInfoAWarning`; this command applies only when `\uniqueXDBChoicesOn` is expanded. In the macro `\xdb@unique` which is expanded when `\uniqueXDBChoicesOn` is expanded first, there is one line that reports information to the log as `\PackageInfo`. By expanding `\makeInfoAWarning` we change `\PackageInfo` to `\PackageWarning`, which gives it greater visibility in the log and the log report.

```
\newcommand{\uniqueXDBChoicesOn}{\xDBUniquefalse
  \PackageWarning{ran_toks}
  {The \string\uniqueXDBChoicesOn\space requires the\MessageBreak
  \texttt{usedbapp} option}}
\newcommand{\uniqueXDBChoicesOff}{\let\xdbunique\relax\xDBUniquefalse}
\let\xdbunique\relax
\newcommand{\makeInfoAWarning}{\def\pkgNotifType{\PackageWarning}}
\def\pkgNotifType{\PackageInfo}
\newif\ifrt@recording \rt@recordingtrue % dps
```

Now for the definition of `\useRanTok`.

```
\newcommand{\useRanTok}[2][\rt@BaseName]{\bgroup\expandafter
  \ifx\csname nMax4#1\endcsname\relax
    \rt@badTokName{#1}\global\let\rt@next\relax
  \else
    \ifnum#2>\@nameuse{nMax4#1}%
      \rt@modarith{#2}{\@nameuse{nMax4#1}}%
      \rt@badIndex{#2}{#1}\edef\Indx{\rt@mod}%
    \else
      \edef\Indx{#2}%
    \fi
    \xdef\rt@orig@Indx{\Indx}%
```

If `\xdbunique` is `\relax`, `\useRanTok` executes as it did in the past (no change in behavior); otherwise, we expand `\xdb@unique` which attempts to avoid duplicate choices based on the DBs input by `\useProbDBs`.

```
    \ifx\xdbunique\relax
```

```
      \ifrt@recording\rt@recordAsUsed{#1-\Indx}\fi
      \xdef\rt@next{\noexpand\@nameuse{rtRanTok\Indx#1}}%
    \else
      \xdb@unique{#1}%
    \fi
  \fi
  \egroup
  \rt@next
}
```

`\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][]{\begingroup
  \def\rt@prior{#1}\displ@yListRandomly
}
\newcommand{\displ@yListRandomly}[2][]{\@tempcntb\z@  % dps5-29
  \expandafter\ifx\csname nMax4#2\endcsname\relax
    \rt@rgi\space\textbf{??}\rt@badTokName{#2}#1%
  \else
    \rt@recordingfalse
```

`\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\@tempcntb
    \def\first{1}\edef\last{\@nameuse{nMax4#2}}\@tempcnta\last
    \advance\@tempcnta\m@ne
    \edef\lessone{\the\@tempcnta}\@whilenum\@tempcntb<\last
      \advance\@tempcntb\@ne
```

There is one example of this command getting confused and printing the wrong number of items. Here, we pass the optional argument of `\useRanTok` and that cleared up the problem.

```
      \do{\rt@prior\useRanTok[#2]{\the\@tempcntb}\rt@post
    }\fi
  \endgroup
}
⟨/package⟩
⟨*dbapp⟩
```

5 A DB application

This (optional) section supports an application of ran_toks to the eqexam package; though, conceptually, the commands of this section may be applied in other settings. In this application, the document author has a series of DB test files (TEX files), each file contains `\bRTVToks`/`\eRTVToks` constructs, which contain a series of `rtVW` environments of verbatim content. In this application, the verbatim content are `problem`/`problem*` environments of eqexam.

The following verbatim listing is taken from the preamble of `mc-db.tex`, which illustrates the layout of how to apply the commands of this section.

```
\examNum{1}
\numVersions{4}
\forVersion{a}
% initial seeds for each of the four versions of this document
\vA{\useThisSeed{54356}}
\vB{\useThisSeed{577867}}
\vC{\useThisSeed{6746788}}
\vD{\useThisSeed{856785}}

\uniqueXDBChoicesOn  % Try to avoid duplicate questions in multi-version doc
\InputUsedIDs        % Input history of previous versions to current version
\viewIDstrue         % To view the IDs of problems used
...
\useTheseDBs{db1,db2,db3,db4}
```

If `\ifxDBUnique` is true and if `eqexam` is loaded, we open `\wrtprobids` which is used to write the problem IDs of the problems already chosen in earlier version of this source file. The name of this file is `\jobname-ver\selVersion.cut`; eg, `mc-db-verA.cut`, `mc-db-verB.cut`, etc.

```
\def\rt@OpenProbIds{\@ifpackageloaded{eqexam}
  {\immediate\openout\wrtprobids\jobname-ver\selVersion.cut}{}}
```

We open the file `\jobname-ver\selVersion.cut` when `\InputUsedIDs` is expanded in the preamble.

```
%\def\rt@ABD{\ifxDBUnique\expandafter\rt@OpenProbIds\fi}
\def\rt@ABD{\@ifundefined{eq@nVersions}{}
  {\ifnum\eq@nVersions>\@ne\expandafter\rt@OpenProbIds\fi}}
```

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}
```

`\useProbDBs{`⟨*list*⟩`}` Is an alias of `\useTheseDBs`.

```
\let\useProbDBs\useTheseDBs
```

`\viewDB{`⟨*name*⟩`}` Typeset the entire contents of a DB Test file. The argument ⟨*name*⟩ is the name of the DB Test file (as in `\bRTVToks{DB1}`, here `DB1` is the ⟨*name*⟩. The DB test files should be input using `\useProbDBs`.

```
\def\viewDB#1{\useRTName{#1}\rt@nCnt\z@
  \edef\nSTOP{\@nameuse{nMax4\rt@BaseName}}%
  \loop\advance\rt@nCnt\@ne
    \rtTokByNum{\the\rt@nCnt}%
  \ifnum\rt@nCnt<\nSTOP\repeat
}
```

`\getR@nIndx` The macro `\getR@nIndx` executes with each entry of `\@nameuse{#1-List}`. For an index value of `\Indx`, the macro goes through the arguments to the `\Indx`'th argument and reads the value of the argument at that point. It returns the argument of the `\Indx`'th as `\ranIndx`; eg, if `\Indx=1`, then `\ranIndx=3`, for the above example.

```
%% uses \@tempcnta and \Indx
\def\getR@nIndx#1{\def\argi{#1}%
  \ifx\argi\rt@STOP
    % no match, something is wrong
    \edef\ranIndex{-1}\else
    \advance\@tempcnta\@ne
    \ifnum\Indx=\@tempcnta
      \def\ranIndx{#1}\fi
  \fi
}
```

`\xdb@unique{`⟨*name*⟩`}` An add-on command to `\useRanTok`. The command attempts to create a unique choice of a problem over several versions of the same document. This may not be possible if there are not enough choices to satisfy the number of declared versions.

The `\xdb@unique` seems to work when the `eqexam` document has multiple parts (more then one `exam` environments). The id files for the parts are all combined;

ideally, for multiple part exams, the second part draws form a set of DB test files different from the first part, as long as there are enough problems to choose from.

Note that, if there is not a unique choice for a question from the designated DB test file, `\xdb@unique` reverts to the original choice so there may be duplicates across versions of the document.

```
\def\rt@NoAltChoice#1#2{\PackageWarning{ran_toks}
 {Cannot find an alternative to #1-#2,\MessageBreak
  will use it but it may be a duplicate\MessageBreak
  question}}
\def\xdb@unique#1{\@tempcnta\z@
 \def\rt@STOP{\relax}%
```

We use the randomized list for the ⟨*name*⟩

```
\@nameuse{#1-List} is the randomized list: eg,
\\{3}\\{2}\\{4}\\{5}\\{1}
```

```
 \let\\\relax\edef\x{\@nameuse{#1-List}}%
 \toks@=\expandafter{\x}\let\\\getR@nIndx
 \the\toks@\\\rt@STOP
```

We take as the default choice the original choice

```
 \xdef\rt@next{\noexpand
   \@nameuse{rtRanTok\rt@orig@Indx#1}}%
```

Begin to look at the results of `\the\toks@\\\rt@STOP`,

```
 \ifnum\ranIndx>\m@ne
```

If `\ranIndx` is -1, we use the original index.

```
   \edef\rt@orig@ranIndx{\ranIndx}%
   \expandafter
   \ifx\csname#1-\ranIndx\endcsname\relax
```

This question has not been chosen earlier, so we'll use it.

```
     \xdef\rt@next{\noexpand
       \@nameuse{rtRanTok\Indx#1}}%
   \else
```

The question has been chosen in an earlier version of the document. Find the next higher unused one (cycle search).

```
     \@tempcntb\z@
     \rt@nCnt\rt@orig@Indx\relax
```

As we move into the `\@whilenum` loop, we take as the default the original index. The loop may overwrite the definition of `\rt@next`.

```
     \xdef\rt@next{\noexpand\rt@NoAltChoice{#1}{\rt@orig@Indx}\noexpand
       \@nameuse{rtRanTok\rt@orig@Indx#1}}%
     \@whilenum\@tempcntb<\@nameuse{nMax4#1}\do{%
       \advance\@tempcntb\@ne
       \advance\rt@nCnt\@ne
```

If the count is at the `nMax4` value, we start over from the beginning.

```
       \ifnum\rt@nCnt>\@nameuse{nMax4#1}\rt@nCnt\@ne\fi
```

We search through `\toks@` again, so we have the initialize the dependent variables: `\Indx`, the index to search for; `\@tempcnta` the counter that is used by `\getR@nIndx`; nothing has changed `\let\\\getR@nIndx` should still be in effect.

```
    \edef\Indx{\the\rt@nCnt}\@tempcnta\z@
    \the\toks@\\\rt@STOP
```

If `\ranIndx` is -1, we use the original index.

```
    \ifnum\ranIndx>\m@ne
     \expandafter
     \ifx\csname#1-\ranIndx\endcsname\relax
      \pkgNotifType{ran_toks}{#1-\rt@orig@ranIndx\space
       has already been used,\MessageBreak
       will use #1-\ranIndx}%
      % exit the \@whilenum loop
      \@tempcntb\@nameuse{nMax4#1}%
      \advance\@tempcntb\@ne
     \fi
    \fi
    \xdef\rt@next{\noexpand\@nameuse{rtRanTok\Indx#1}}%
   }% do
   \ifnum\@tempcntb=\@nameuse{nMax4#1}\relax
    \xdef\rt@next{\noexpand
     \rt@NoAltChoice{#1}{\rt@orig@ranIndx}\noexpand
     \@nameuse{rtRanTok\rt@orig@Indx#1}}%
   \fi
  \fi
 \fi
}
```

`\uniqueXDBChoicesOn` Here is the operational definition of `\uniqueXDBChoicesOn`; when executed in the preamble, an attempt is made the select only problems that have not already been chosen in any prior renditions of the same source file.

```
\renewcommand{\uniqueXDBChoicesOn}{\xDBUniquetrue
 \let\xdbunique\xdb@unique}
```

`\InputUsedIDs` Input the user ID (CUT) files. These are files that document which questions were used for the various versions of the exam.

```
\newif\ifrt@InputUsedIDs\rt@InputUsedIDsfalse
\def\InputUsedIDs{\rt@InputUsedIDstrue
 \bgroup
  \setcounter{eq@count}{0}%
  \let\rt@InputUsedIDs\@empty
  \let\rt@InputUsedIDsFIs\@empty
  \@whilenum \value{eq@count}<\eq@nVersions\relax\do
  {%
   \stepcounter{eq@count}%
   \g@addto@macro\rt@InputUsedIDs{\if\selVersion}%
   \g@addto@macro\rt@InputUsedIDsFIs{\fi}%
   \edef\x{\Alph{eq@count}}%
   \edef\y{\noexpand\g@addto@macro\noexpand
```

```
        \rt@InputUsedIDs{\x\expandafter\noexpand
        \csname else\endcsname\noexpand\rt@IIFE}}\y
      \edef\x{{\x}}\expandafter
      \g@addto@macro\expandafter\rt@InputUsedIDs\expandafter{\x}%
    }% do
    \expandafter\g@addto@macro\expandafter
      \rt@InputUsedIDs\expandafter{\rt@InputUsedIDsFIs}%
    \egroup
  \rt@InputUsedIDs
  \AtBeginDocument{\rt@ABD}%
}
\@onlypreamble\InputUsedIDs
```

A convenience command used by `\InputUsedIDs`.

```
\def\rt@IIFE#1{\InputIfFileExists{\jobname-ver#1.cut}
  {\PackageInfo{ran_toks}{Inputting \jobname-ver#1.cut}}
  {\PackageInfo{ran_toks}{Cannot find \jobname-ver#1.cut}}}
```

```
⟨/dbapp⟩
⟨*package⟩
⟨/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.

7 Change History

v1.0b (2013/07/29)
  General: Added `\displayListRandomly` 11
v1.0c (2013/08/03)
  General: Save the initial seed value to `\rtInitSeedValue`. 2
v1.0d (2013/08/03)
  General: Added conditional input of random.tex 2
v1.0e (2016/02/06)
  General: Added optional argument to `\displayListRandomly` 11
  Fixed a bug, when the first two tokens `#1` are the same, we get an incorrect decision 8
v1.1 (2017/05/04)
  General: Added second optional argument to `\displayListRandomly` 11
  Save out list for later use 5
v1.2 (2019/12/28)
  General: Added dummy package ran-toks 3
  `rtVW`: Defined `\rtVWHook` 7

v1.3 (2021/01/14)
  General: Added `usedbapp` option 3
    Added several commands and macro to continue to support a DB application. 13

v1.3.1 (2021/01/19)
  General: Added `\ifrt@recording` 11

v1.3.2 (2021/05/25)
  General: Insert optional argument for `\useRanTok` within `\displ@yListRandomly` . 12

v1.3.3 (2021/05/29)
  General: In support of nested `\bRTVToks`/`\eRTVToks` constructs, replace the counter `\rt@nCnt` with `\@tempcntb` 11
  `rtVW`: Assign `\rtVWi` and `\endrtVWi` 8

v1.4 (2021/06/06)
  General: Rounded up v1.3.3 to v1.4 for publication 2