Package randomlist
Tools for data base, table and random writing-reading

Jean-Côme Charpentier∗ Christian Tellechea†

September 11, 2017

∗jean-come.charpentier@wanadoo.fr
†unbonpetit@openmailbox.org
## Contents

1 Overview ........................................ 3

2 Array, queue, stack, or list? ............... 3
   2.1 Create, erase and show list .................. 3
   2.2 Writing and reading in a list ............... 4
      2.2.1 Insert commands ......................... 4
      2.2.2 Extract commands ....................... 7
      2.2.3 Set commands ........................... 9
      2.2.4 Get commands ........................... 11

3 Database ........................................ 12
   3.1 Simple database ............................. 12
      3.1.1 \ForEachFirstItem ....................... 13
      3.1.2 \ForEachLastItem ...................... 13
      3.1.3 \ForEachRandomItem .................... 13
   3.2 Database with fields ....................... 14
   3.3 Tricks, things, and other matters ......... 16
      3.3.1 Random number .......................... 16
      3.3.2 Loop .................................. 16
      3.3.3 Internal ................................ 18

4 \LaTeX Lists .................................... 18

5 Package randomlist code ....................... 20
   5.1 \LaTeX’s wrapper ............................. 20
      5.1.1 Introduction ............................ 20
      5.1.2 \LaTeX lists ............................ 20
   5.2 \TeX code .................................. 21
      5.2.1 Introduction and first commands ....... 24
      5.2.2 General list commands .................. 26
      5.2.3 Writing and reading list commands .... 29
      5.2.4 Loop on list ........................... 35
      5.2.5 Database ................................ 35

A File pythagoras.dat .......................... 40
B File pupils.dat .............................. 43
C File comets.dat .............................. 44
1 Overview

The main aim of package randomlist is to work on list, especially with random operation. The hidden aim is to build personal collection of exercises with different data for each pupils. In order to build such exercises, some features about databases are necessary.

In “randomlist”, the word “List” must be understood with two meanings:

- itemize and enumerate \LaTeX environments;
- list as in computer science.

In fact, lists as in computer are not really lists: they are arrays. Some commands allow to deal with these data structures as queues, other commands as stacks and another commands as arrays.

2 Array, queue, stack, or list?

The package give the name “list” to the main data structure. First, we have to declare a new list with command \NewList. There is nothing special about this command. It has a mandatory argument: the name of the list.

Nearly any name is possible. However, don’t use hyphen and number at the end of the name. For instance mylist-1 isn’t a good idea (mylist*1 is a good one). Don’t use fragile commands and special characters. However you can use commands inside list names.

2.1 Create, erase and show list

You can’t create an already existing list. For instance, the code:

\begin{verbatim}
\NewList{MyList}
\NewList{MyList}
\end{verbatim}

give the error message:

! Package randomlist Error: List MyList already exists.

If you want erase a list, use the command \ClearList.

You can’t create a list namelist if the macro \namelist exists. For instance the code:

\begin{verbatim}
\NewList{def}
\end{verbatim}

give the error message:

! Package randomlist Error: Command \def already exists.

As you can see, the source is between horizontal rules and flush right. The result is flush left. When the result isn’t an error message, source and result are side by side.

For the next commands we must be able to see the state of list. The package randomlist offers the \ShowList command which allows to see the whole list. When a list is just created, it is empty, so the \ShowList command shows it like that (source is typeset at the right side and result is showed at the left side):

\begin{verbatim}
BEGIN{MyList} (empty list)
\ShowList{MyList}
END{MyList}
\end{verbatim}
2.2 Writing and reading in a list

Once a list is created, you can write values and, after that, read values.

In fact, these lists can behave like queues, stacks, or arrays according to the used command. There are four kinds of command:

- Insert;
- Extract;
- Set;
- Get.

Each one has four variants to reach some position in the list:

- First;
- Last;
- Index (without prefix);
- Random.

Thus we have the commands:

\texttt{\textbackslash InsertFirstItem \textbackslash ExtractFirstItem \textbackslash SetFirstItem \textbackslash GetFirstItem}  
\texttt{\textbackslash InsertLastItem \textbackslash ExtractLastItem \textbackslash SetLastItem \textbackslash GetLastItem}  
\texttt{\textbackslash InsertItem \textbackslash ExtractItem \textbackslash SetItem \textbackslash GetItem}  
\texttt{\textbackslash InsertRandomItem \textbackslash ExtractRandomItem \textbackslash SetRandomItem \textbackslash GetRandomItem}

Each one has also a "List" variant which acts on several items. That is, we have these commands: \texttt{\textbackslash InsertList}, \texttt{\textbackslash ExtractList}, \texttt{\textbackslash SetList}, and \texttt{\textbackslash GetList}. There is also a command \texttt{\textbackslash CopyList} which is a shortcut for a special \texttt{\textbackslash SetList}.

Finally, we have \texttt{\textbackslash ShiftList} which is somewhere quiet special: it creates empty items or destroy items by shifting inside a list. This macro is rather for internal operation but you can use it. The syntax is:

\texttt{\textbackslash ShiftList\{<list>\}\{<start>\}\{<nb>\}}

When \texttt{<nb>} is positive, then items from index \texttt{<start>} are right shifted. That is, index \texttt{n} becomes index \texttt{n+nb} and index from \texttt{<start>} to \texttt{<start>+<nb>−1} are empty.

When \texttt{<nb>} is negative, then items starting from the end of the list are left shifted and \texttt{<nb>} items (from index \texttt{start}) disappear.

Usually, you don't need \texttt{\textbackslash ShiftList}; the other commands, especially \texttt{\textbackslash ExtractList} and \texttt{\textbackslash InsertList} are enough.

2.2.1 Insert commands

\texttt{\textbackslash InsertFirstItem} \texttt{\textbackslash InsertFirstItem} writes a value at the beginning of a list and shift other values to the end of the list. This command has two arguments: the list name and the value to insert. For example:
As you can see, the list is in the reverse order than the user one. With \InsertFirstItem, list behaves like stack.

The value written in the list can be nearly anything. For instance:

\begin{verbatim}
\NewList{MyList}
\InsertFirstItem{MyList}{First value}
\ShowList{MyList}
\InsertFirstItem{MyList}{Second value}
\ShowList{MyList}
\InsertFirstItem{MyList}{Third value}
\ShowList{MyList}
\end{verbatim}

With this command, the list behaves as queue.

\begin{verbatim}
\NewList{MyList}
\InsertLastItem{MyList}{{two}{groups}}
\InsertLastItem{MyList}{\textbf{text}}
\InsertLastItem{MyList}{special^}
\InsertLastItem{MyList}{end}
\ShowList{MyList}
\end{verbatim}

\Insertitem \Insertitem is like \InsertFirstItem but the insertion is made to a specified position of the list. With this command, the list behaves as an array.

The index starts from zero and if the list has \( n \) elements then the index can’t be greater than \( n \). When a value is inserted in position \( k \), then the previous values from \( k \) to \( n - 1 \) are shifted one position to the right.

\Insertitem takes three arguments: the list name, the position and the value. Here is an example:
As you can see, when the two values is inserted at position 2, the value special^ is shifted from position 2 to position 4. Moreover, it's possible to insert a value to the nonexistent position \( n \) when the length of the list is \( n \): it's the only one possibility to specify a nonexistent index.

\( \text{\textbf{\textbackslash{InsertRandomItem}}} \) This is the first command which use random numbers. We will see later how to manage random numbers themselves.

\( \text{\textbackslash{InsertRandomItem}} \) command works as the \( \text{\textbackslash{InsertItem}} \) one but the position is selected randomly by \TeX. Then there is only two arguments: the list name and the value to insert. Here is an example:

As you can see, the second list is unchanged after operation.
Package randomlist checks that both lists exists and that index is compatible with list. Otherwise an error message will be raised.

It's possible to insert an empty list:

\begin{verbatim}
\NewList{MyList}
\NewList{OtherList}
\InsertLastItem{MyList}{first}
\InsertLastItem{MyList}{second}
\InsertLastItem{MyList}{third}
\InsertList{MyList}{1}{OtherList}
\ShowList{MyList}
\end{verbatim}

2.2.2 Extract commands

\ExtractFirstItem

The four commands \Extract...Item are the inverse one of the four commands \Insert...Item.

\ExtractFirstItem extract the first value of a list and store it in a macro. The other elements of the list are shifted left (the list length decreases by one). This command takes two argument: the list name and the macro name where the value is stored. The last argument is just the name of the macro, i.e. the macro name without the backslash. For example:

\begin{verbatim}
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\ExtractFirstItem{MyList}{MyMacro}
\ShowList{MyList}
\end{verbatim}

The first element was “\TeX”.

\begin{verbatim}
\NewList{MyList}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\ExtractFirstItem{MyList}{MyMacro}
\ShowList{MyList}
\end{verbatim}

The first element was ‘\MyMacro’.

When you extract an element from a list, the list length decreases by one. It explains why it’s forbidden to extract an element from an empty list. If you try it,

\begin{verbatim}
\NewList{MyList}
\ExtractFirstItem{MyList}{MyMacro}
\end{verbatim}

you have the error message:

! Package randomlist Error: List MyList is empty.

\ExtractLastItem

\ExtractLastItem behaves like \ExtractFirstItem but the element extracted is the last one. Thus there is no shifting, there is just a decrementation of the list length.

Here is an example:
The last element was “powerful”.
BEGIN{MyList} (3 elements)
  MyList[0] = \TeX
  MyList[1] = is
  MyList[2] = very
END{MyList}

\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\ExtractLastItem{MyList}{MyMacro}
The last element was ‘‘\MyMacro’’.
\ShowList{MyList}

\ExtractItem \ExtractItem behaves like \ExtractFirstItem but the element extracted is the one indicated by its index. The command takes three argument: the list name, the index of element to extract, the macro used to store the element extracted. Don’t forget that indexes start from zero. Here is an example:

BEGIN{MyList} (3 elements)
  MyList[0] = \TeX
  MyList[1] = is
  MyList[2] = powerful
END{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\ExtractItem{MyList}{2}{MyMacro}
The third element was ‘‘\MyMacro’’.
\ShowList{MyList}

There isn’t anything special. The length of the list decreases by one and elements are shifted accordingly to the extracted one.

\ExtractRandomItem \ExtractRandomItem works like the previous \ExtractItem. Here, the index is selected randomly by the computer. Then there are only two arguments: the list name and the macro to store the extracted element:

BEGIN{MyList} (3 elements)
  MyList[0] = \TeX
  MyList[1] = very
  MyList[2] = powerful
END{MyList}
\ExtractRandomItem{MyList}{MyMacro}
‘‘\MyMacro’’ was extracted.
\ShowList{MyList}

Even the extraction is made on a random index, it’s forbidden to extract something from an empty list. Then, the code:

\NewList{MyList}
\ExtractRandomItem{MyList}{MyMacro}
gives the usual error message:

! Package randomlist Error: List MyList is empty.

8
The commands \texttt{\ExtractList} \texttt{\Extract...Item} extract one item and store it in a macro. With the command \texttt{\ExtractList} we can extract several items and put them in a list. \texttt{\ExtractList} asks for four arguments:

1. the main list;
2. the starting index;
3. the ending index;
4. the list which receive extracted values.

Here is an example:

\begin{verbatim}
BEGIN{MyList} (3 elements)
  MyList[0] = first
  MyList[1] = second
  MyList[2] = sixth
END{MyList}
BEGIN{OtherList} (3 elements)
  OtherList[0] = third
  OtherList[1] = fourth
  OtherList[2] = fifth
END{OtherList}
\InsertLastItem{MyList}{first}
\InsertLastItem{MyList}{second}
\InsertLastItem{MyList}{third}
\ExtractList{MyList}{2}{4}{OtherList}
\ShowList{MyList}
\ShowList{OtherList}
\end{verbatim}

Obviously, \texttt{randomlist} checks list and indexes. You can have the start index and the last index equals. In this case, \texttt{\ExtractList} behaves like \texttt{\ExtractItem} but the extracted value is put in a list rather than in a macro:

\begin{verbatim}
BEGIN{MyList} (5 elements)
  MyList[0] = \LaTeX
  MyList[1] = is
  MyList[2] = very
  MyList[3] = powerful
  MyList[4] = sixth
END{MyList}
BEGIN{OtherList} (1 element)
  OtherList[0] = third
END{OtherList}
\ExtractList{MyList}{2}{2}{OtherList}
\ShowList{MyList}
\ShowList{OtherList}
\end{verbatim}

2.2.3 \textbf{Set commands}

\texttt{\SetFirstItem} The commands \texttt{\Set...Item} modify the existing values of list. \texttt{\SetFirstItem} modify the first value.

\begin{verbatim}
BEGIN{MyList} (4 elements)
  MyList[0] = \LaTeX
  MyList[1] = is
  MyList[2] = very
  MyList[3] = powerful
END{MyList}
\ExtractList{MyList}{2}{4}{otherList}
\ShowList{MyList}
\end{verbatim}
If a list is empty, there is the classic error message about empty list.

\texttt{\textbackslash SetLastItem} \texttt{\textbackslash SetLastItem} acts like \texttt{\textbackslash SetFirstItem} but at the end of the list.

\texttt{\textbackslash SetItem} \texttt{\textbackslash SetItem} acts like the previous commands. It takes three arguments: the list name, the index, the new value:

BEGIN\{MyList\} (4 elements)
\texttt{MyList[0]} = \TeX
\texttt{MyList[1]} = is
\texttt{MyList[2]} = quiet
\texttt{MyList[3]} = powerful
END\{MyList\}

\texttt{NewList\{MyList\}}
\texttt{InsertLastItem\{MyList\}\{\TeX\}}
\texttt{InsertLastItem\{MyList\}\{is\}}
\texttt{InsertLastItem\{MyList\}\{very\}}
\texttt{InsertLastItem\{MyList\}\{powerful\}}
\texttt{setItem\{MyList\}\{2\}\{quiet\}}
\texttt{ShowList\{MyList\}}

If the index doesn’t exist, an error message is showed. Code:

\texttt{NewList\{MyList\}}
\texttt{InsertLastItem\{MyList\}\{\TeX\}}
\texttt{InsertLastItem\{MyList\}\{is\}}
\texttt{InsertLastItem\{MyList\}\{very\}}
\texttt{InsertLastItem\{MyList\}\{powerful\}}
\texttt{setItem\{MyList\}\{4\}\{isn’t it?\}}

\texttt{gives the error message:}

! Package randomlist Error: Index 4 is greater than last index of list MyList.

\texttt{\textbackslash SetRandomItem} \texttt{\textbackslash SetRandomItem} acts like the previous one but the index is selected randomly. The list must be non empty. Here is an example:

BEGIN\{MyList\} (5 elements)
\texttt{MyList[0]} = \TeX
\texttt{MyList[1]} = is
\texttt{MyList[2]} = snap!
\texttt{MyList[3]} = very
\texttt{MyList[4]} = powerful
END\{MyList\}

\texttt{NewList\{MyList\}}
\texttt{InsertLastItem\{MyList\}\{\TeX\}}
\texttt{InsertLastItem\{MyList\}\{is\}}
\texttt{InsertLastItem\{MyList\}\{very\}}
\texttt{InsertLastItem\{MyList\}\{powerful\}}
\texttt{setItem\{MyList\}\{4\}\{isn’t it?\}}
\texttt{SetRandomItem\{MyList\}\{snap!\}}
\texttt{ShowList\{MyList\}}

\texttt{gives the error message:}

! Package randomlist Error: Index 4 is greater than last index of list MyList.

\texttt{\textbackslash SetList} \texttt{\textbackslash SetList} Insert value one by one inside a list could be tiresome especially if you have many values. Package randomlist allows to insert many items in a row using the macro \texttt{\textbackslash SetList}. Items are separated with comma. For instance:

BEGIN\{MyList\} (4 elements)
\texttt{MyList[0]} = \TeX
\texttt{MyList[1]} = is
\texttt{MyList[2]} = very
\texttt{MyList[3]} = powerful
END\{MyList\}

\texttt{NewList\{MyList\}}
\texttt{SetList\{MyList\}\{\TeX, is, very, powerful\}}
\texttt{ShowList\{MyList\}}

\texttt{As you can see, spaces aren’t discarded. A more satisfactory presentation would be:}

\texttt{BEGIN\{MyList\} (4 elements)
\texttt{MyList[0]} = \TeX
\texttt{MyList[1]} = is
\texttt{MyList[2]} = very
\texttt{MyList[3]} = powerful
\texttt{END\{MyList\}}

\texttt{\textbackslash ShowList\{MyList\}}

\texttt{As you can see, spaces aren’t discarded. A more satisfactory presentation would be:}
BEGIN{MyList} (4 elements)
MyList[0] = \TeX
MyList[1] = is
MyList[2] = very
MyList[3] = powerful
END{MyList}
\NewList{MyList}
\SetList{MyList}{\TeX,is,very,\%}
\ShowList{MyList}
\CopyList

\NewList{MyList}
\SetList{MyList}{\TeX,is,very,\%}
\CopyList{MyList}{OtherList}
\ShowList{OtherList}

\NewList{OtherList}
BEGIN{OtherList} (4 elements)
OtherList[0] = \TeX
OtherList[1] = is
OtherList[2] = very
OtherList[3] = powerful
END{OtherList}
\NewList{MyList}
\NewList{OtherList}
\SetList{MyList}{\TeX,is,so,cute}
\CopyList{MyList}{OtherList}
\ShowList{OtherList}

2.2.4 Get commands

\GetFirstItem
The \get...list look for a value in a list. They don't change the list. The index must exist elsewhere an error message will be show. That is, for first, last, and random variant, list must be non empty.
\GetFirstItem put the first value of a list into a macro. The arguments of the command are: the list name, the macro:
The first element is "TeX"
BEGIN{MyList} (4 elements)
MyList[0] = \TeX
MyList[1] = is
MyList[2] = so
MyList[3] = cute
END{MyList}
\GetFirstItem{MyList}{2}{MyMacro}
The third element is "so"
\GetLastItem
\GetLastItem acts like \GetFirstItem but give the last value.
\GetItem
\GetItem acts like the previous one but give the value of the element $k$ where $k$ is the second argument. Pay attention that indexes start from zero. Then the index $k$ maps to the $k+1$st element of the list.
The third element is "so"
BEGIN{MyList} (4 elements)
MyList[0] = \TeX
MyList[1] = is
MyList[2] = so
MyList[3] = cute
END{MyList}
\GetItem{MyList}{2}{MyMacro}
The third element is \"MyMacro\"
\GetItem{MyList}{3}{MyMacro}
The third element is \"MyMacro\"

Package randomlist offers an other syntax to access to an item: \texttt{\langle\texttt{name\_list}\rangle[\langle\texttt{index}\rangle]}.
Thus, we can write the previous example like that:
The third element is "so"
BEGIN{MyList} (4 elements)
MyList[0] = \TeX
MyList[1] = is
MyList[2] = so
MyList[3] = cute
END{MyList}
\SetList{MyList}{\TeX,is,so,cute}
The third element is \texttt{\langle\\langle\texttt{MyList}[2]\rangle\rangle}"\n
11
\GetRandomItem \GetRandomItem give the value of a randomly selected element of a list.

The random element is “is”

\begin{mylist} (4 elements)
\begin{itemize}
  \item MyList[0] = $\LaTeX$
  \item MyList[1] = is
  \item MyList[2] = so
  \item MyList[3] = cute
\end{itemize}
\end{mylist}

\gettlist{\mylist}{\myitem}
The random element is ‘‘\myitem’’

\showlist{\mylist}

\gettlist \gettlist builds a sub-list. Arguments are those of \xtractlist, that is, the read list, the first index, the last index, and the written list.

\begin{mylist} (6 elements)
\begin{itemize}
  \item MyList[0] = X1
  \item MyList[1] = X2
  \item MyList[2] = X3
  \item MyList[3] = X4
  \item MyList[4] = X5
  \item MyList[5] = X6
\end{itemize}
\end{mylist}

\gettlist{\mylist}{2}{4}{\otherlist}

\showlist{\mylist}
\showlist{\otherlist}

Contrary to what \xtractitem do, \gettlist don’t modify the source list.

3 Database

3.1 Simple database

Package \randomlist offers some features about databases. In fact that was the first aim of this package: to be able to produce one assignment for one pupil (with all assignments different).

For \randomlist a database is a list. For instance the next example shows an usual list which is used as a database. We’ll see later real databases with records and fields. For now, our database has records and each record has one single field: the name and first name of our pupils. In order to parse all entries of database \randomlist offers the commands \foreach...item. These command extract one by one all the elements of a list and typeset, for each element, its third argument. Second argument give the macro name where element is stored. Depending how the extraction is made, we have the three commands: \foreachfirstitem, \foreachlastitem, and \foreachrandomitem. In fact, the readind is made with an extraction but, as the work is made in a group, after the \foreach... command, the list is restored.
3.1.1 \ForEachFirstItem
Test for Alfred Aho
blah blah blah...
Test for Charles Babbage
blah blah blah...
Test for Gregory Chaintin
blah blah blah...
Test for Edsger Dijkstra
blah blah blah...
\NewList{Pupils}
\SetList{Pupils}{Alfred Aho,%
Charles Babbage,Gregory Chaintin,%
Edsger Dijkstra}
\ForEachFirstItem{Pupils}{Name}{%
Test for \Name\par
blah blah blah\dots\par\smallskip
}

3.1.2 \ForEachLastItem
\ForEachLastItem acts like \ForEachFirstItem but the reading is made in reverse order:
Test for Edsger Dijkstra
blah blah blah...
Test for Gregory Chaintin
blah blah blah...
Test for Charles Babbage
blah blah blah...
Test for Alfred Aho
blah blah blah...
\NewList{Pupils}
\SetList{Pupils}{Alfred Aho,%
Charles Babbage,Gregory Chaintin,%
Edsger Dijkstra}
\ForEachLastItem{Pupils}{Name}{%
Test for \Name\par
blah blah blah\dots\par\smallskip
}

3.1.3 \ForEachRandomItem
\ForEachRandomItem acts like the previous commands but the reading is made randomly.
In the next example, we can see that the list is restored after the command \ForEachRandomItem:
Test for Edsger Dijkstra
blah blah blah...
Test for Gregory Chaintin
blah blah blah...
Test for Alfred Aho
blah blah blah...
Test for Charles Babbage
blah blah blah...
BEGIN{Pupils} (4 elements)
  Pupils[0] = Alfred Aho
  Pupils[1] = Charles Babbage
END{Pupils}
\NewList{Pupils}
\SetList{Pupils}{Alfred Aho,%
Charles Babbage,Gregory Chaintin,%
Edsger Dijkstra}
\ForEachRandomItem{Pupils}{Name}{%
Test for \Name\par
blah blah blah\dots\par\smallskip
}
\ShowList{Pupils}
You can put a command \ForEach inside another one. There is no limits (but the stacks of TeX):
Charles and Grace are computer scientists
Charles and Adele are computer scientists
Charles and Ada are computer scientists
Alfred and Grace are computer scientists
Alfred and Ada are computer scientists
Alfred and Adele are computer scientists
Gregory and Grace are computer scientists
Gregory and Ada are computer scientists
Gregory and Adele are computer scientists

Actually, there is a bug that don’t allow fragile commands inside lists when they are read with \ForEach...Item commands. I hope that the next version of randomlist will fix this!

3.2 Database with fields

Each record of a database is read as a set of fields. In fact it's a sequence of groups. randomlist allow to read each field with the macro \ReadFieldItem. In order to make life easy, randomlist allow to read whole database from files with the command \ReadFileList.

This command read a field in a record and store it in a macro. It takes three arguments: a whole record or a macro containing the whole record, the rank of the field (starting zero), and a macro to store the value of this field. For instance:

\def\record{{ein}{un}{one}}
\ReadFieldItem{\record}{0}{Zahl}
\ReadFieldItem{\record}{1}{Nombre}
\ReadFieldItem{\record}{2}{Number}
\Nombre\ French, \Zahl\ German, and \Number\ English.

If there are less fields than the indicating rank then an error message is raised:

\def\record{{ein}{un}{one}}
\ReadFieldItem{\record}{3}{Stuff}
give the error message:

! Package randomlist Error: There aren't enough fields in the record.

Remember that fields are numbered starting from zero!

Obviously, the power of \ReadFieldItem comes with list and real database. For instance:

\NewList{Languages}
\SetList{Languages}{{France}{un},% {Germany}{ein},{England}{one}}
\ForEachFirstItem{Languages}{Unit}{% \ReadFieldItem{\Unit}{0}{Country}% \ReadFieldItem{\Unit}{1}{Number}% You say \"Number\" in \Country.\par
}

You say “un” in France.
You say “ein” in Germany.
You say “one” in England.

It’s not very handy to write a whole database inside a \LaTeX source. randomlist allows to load a data base reading a extern file. For that, there is the command \ReadFileList. This
command takes two mandatory arguments: the name of the database and the name of the file.

The file `pythagoras.dat` (page 40) shows 100 lines of three numbers separated by comma. When this file is read, the database contains 100 records with three fields. That is, by default, `randomlist` read CSV files (Comma Separated Values).

Do you know that $\sqrt{147^2 + 196^2}$ is an integer?

To those who check the triple page 40, don’t forget that the 10th rank maps with line number 11, that is, “147,196,245”. Moreover, you have the result to the operation: $\sqrt{147^2 + 196^2} = 245$.

A file could have any structure. In particular, it could have one or several title lines. It’s the case for the file `pupils.dat` (page 43) where the first line is obviously a title line. You have just to extract this or these lines to obtain a “classical” database.

Processing this way, you have got a database with real datas (no title data).

The file could be in another format than CSV. In fact, you can define a field separator (comma by default) and a string delimiter (double quote by default) which allow to put a field separator inside a field. To indicate other symbols than comma and double quote, the command `ReadFileList` accept an optional argument which declare the field separator and the string encloser by two characters.

For instance, the file `comets.dat` (see page 44) has “|” as field separator. Therefore, the calling syntax becomes:
The comet 3D/Biela was discovered by Biela in 1826. Its period is 6.62 years.

Observe that we have two “title lines” to discard. As each line begins by a field separator, the first field of each record is empty. Thus we extract field starting one (not zero). We test if a period is empty because of 18D/Perrine-Mrkos comet.

3.3 Tricks, things, and other matters

3.3.1 Random number

In the package randomlist, the (pseudo) random numbers are processed by the macro \RLuniformdeviate\{<n>\}{<macro>} (choose a random integer number between 0 and \text{n}−1 and store it in \text{<macro>}) and \RLsetrandomseed (set the seed).

When you say nothing, the seed is calculated with the current date (year, month, day, hour and minute). That is, if you run \text{latex} twice with a delay greater than one minute, you will have two different results. Sometime, it’s what you want, sometime it’s annoying.

Under \LaTeX, you can set the seed with the package option seed with the syntax:

\usepackage[seed=<value>]{randomlist}

where <value> is an integer value. If <value> is zero then the seed is calculated using actual time, year, month and day.

Under \TeX you have to use the command \RLsetrandomseed to give the seed value. As for the option, with a zero value, the seed is calculated using actual time, year, month and day. The syntax is simple:

\RLsetrandomseed{<value>}

Of course, this command is available under \LaTeX.

3.3.2 Loop

For complex material with several databases, it could be useful to use external loop such \foreach from \text{pgffor} package or \multido from \text{multido} package. In fact, this is a good idea but not the best! These commands (\foreach and \multido) work inside a group at each loop. With \text{randomlist} that doesn’t work everytime since lists are restored at each loop. For example, you can’t extract element of a list.
It is possible to read the list with \Get...\Item. In this case, you should probably use the command \CountList to know the size of a list. This command takes two arguments: the list name and a macro to store the number of elements. As usual you give only the name of the macro (without the backslash). The big difference between extract and get is that with random reading, you can avoid to have twice (or more) the same element.

A real example is too long to be inserted here. We give only the code source. The file .tex and the result .pdf are part of the package distribution. In this example, we use two databases: one for the pupils and the other one for the pythagorean triples. As we read randomly the pythagorean triples and as we won’t the same test for two pupils, then we don’t use the external loop described in the latter paragraph.

\begin{verbatim}
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[a4paper, margin=2.5cm, noheadfoot]{geometry}
\usepackage{amsmath}
\usepackage[seed=1]{randomlist}
\pagestyle{empty}
\setlength{\parindent}{0pt}
\NewList{Pupils}
\NewList{Triples}
\begin{document}
\ReadFileList{Pupils}{pupils.dat}
\ExtractFirstItem{Pupils}{NULL} % extract title line
\ReadFileList{Triples}{pythagoras.dat}
\ForEachFirstItem{Pupils}{Pupil}
{\ReadFieldItem{\Pupil}{0}{Name}
\ReadFieldItem{\Pupil}{1}{FName}
\ReadFieldItem{\Pupil}{2}{Note}
\ExtractRandomItem{Triples}{Triple}
\ReadFieldItem{\Triple}{0}{Triplea}
\ReadFieldItem{\Triple}{1}{Tripleb}
\ReadFieldItem{\Triple}{2}{Triplec}
\begin{center}
\fbox{\huge\bfseries Test for \Name{} \FName} \\
\textbf{Exercise} \par
\if A\Note
The diagonal of a rectangle is \Triplec~in and a side of this
rectangle is \Triplea~in. What is the length of the other side of
the rectangle?
\else
Find the length of the diagonal of a rectangle that is \Triplea~in
by \Tripleb~in.
\fi
\end{center}
\newpage
\begin{center}
17
\end{center}
\end{document}
\end{verbatim}
\begin{center}
Answer to the test for \Name{} \FName
\end{center}
\textbf{Exercise} \par
\if A\Note
Use Pythagorean theorem. We have:
\[
\text{diag}^2 = \text{side1}^2 + \text{side2}^2.
\]
Here:
\[
\text{TTR}^2 = \text{TTR}a^2 + \text{side2}^2
\]
and then
\[
\text{side2} = \sqrt{TTR^2 - TTRa^2} = \text{TTRb}.
\]
\else
Use Pythagorean theorem. We have:
\[
\text{diag}^2 = \text{side1}^2 + \text{side2}^2.
\]
Here:
\[
\text{diag}^2 = \text{TTR}a^2 + \text{TTR}b^2
\]
and then
\[
\text{diag} = \sqrt{TTRa^2 + TTRb^2} = \text{TTRc}.
\]
\fi
\newpage
\end{document}

Be careful! When we extract triples inside the loop, we must be sure that there is more triples than pupils elsewhere an error about an empty list is raised.

Lines 15, 16, 17 read the databases and extract the title line from pupils.dat. After that, we enter in the main loop (lines 18 to 60).

At the beginning of the loop, we read the fields for the pupil (name, first name and note) and the three fields of the pythagorean triple (lines 20 to 26). It's here that we extract randomly a triple. Since it's an extraction, another pupil will have another triple.

Lines 27 to 39 typeset the test and lines 40 to 59 typeset the answer to the test. We test the note of the pupil to decide the type of exercise: Pythagorean theorem to find the hypotenuse (easy) or Pythagorean theorem to find a side (less easy).

3.3.3 Internal

You can access directly to a list. It's not recommended but...

If the list name is LName (pay attention to the letter case), then the length of the list is \LName-len and the n th element of the list (starting from zero) is \LName-n. When an element is a record with several fields, those fields are inside braces. For example the first element of list Triples (see last example) is: \Triples-0 = \{119\}\{120\}\{169\}. As you can see, inside a list, the characters for separator field and for string delimiter don't exist.

The authors don't see any situation where knowing internal is important. If some users have good idea about it then writing to the authors will be an appreciate initiative!

4 TEX Lists

Package \randomlist offers two other special commands which allow to build random lists.

The first one is \RandomItemizeList which build an itemize list with random placement of items. Each item is a group.
\LaTeX{} is:

\begin{itemize}
\item cynical
\item magical
\item logical
\item clinical
\item practical
\end{itemize}

The second command is for enumerate list. It is \RandomEnumerateList and it acts like the previous one:

\begin{itemize}
\item clinical
\item practical
\item logical
\item cynical
\item magical
\end{itemize}
5 Package randomlist code

5.1 \LaTeX's wrapper

5.1.1 Introduction

We start with release number and date.
\begin{verbatim}
\NeedsTeXFormat{LaTeX2e}[1995/06/01]
\ProvidesPackage{randomlist}
\[2017/09/11 v1.3 Package for random list (JCC, CT)]
\end{verbatim}
\LaTeX's wrapper has the possibility to use option. There is only one option: the seed one. It requires the (x)keyval package.
\begin{verbatim}
\RequirePackage{xkeyval}
\DeclareOptionX{seed}{\gdef\RL@seed{#1}}
\ExecuteOptions{seed=0}
\ProcessOptionsX
\end{verbatim}

We can now call the real randomlist code!
\begin{verbatim}
\input{randomlist}
\end{verbatim}

5.1.2 \LaTeX lists

Obviously, \LaTeX lists are useful only with \LaTeX!

\RandomItemizeList Build an itemize list with random placement of items.
\begin{verbatim}
\NewList{\textit{RandomList}}
def\RandomItemizeList{%
def\RL@Type{itemize}%
\ClearList{\textit{RandomList}}%
@ifnextchar\bgroup{%
InsertRandomItem{\textit{RandomList}}[#1]%
@ifnextchar\bgroup{%
}@randomlist}{@randomlist}%
def@@randomlist{%
\edef\RL@body{\begin{\RL@Type}}%
\RLfor \RL@var = 0 to \RL@lenof{\textit{RandomList}}-1 \do{%
\edef\RL@body{\unexpanded\expandafter{\RL@body}\item \csname \textit{RandomList}-\RL@var\endcsname}%
}%
\edef\RL@body{\unexpanded\expandafter{\RL@body}}%\end{\RL@Type}}%
\RL@body
\end{verbatim}

\RandomEnumerateList Like randomitemize but for enumerate list.
\begin{verbatim}
\newcommand\RandomEnumerateList{%
def\RL@Type{enumerate}%
\ClearList{\textit{RandomList}}%
@ifnextchar\bgroup{%
@randomlist}{@randomlist}%
def@@randomlist{%
\edef\RL@body{\unexpanded\expandafter{\RL@body}}%\noexpand\end{\RL@Type}}%
\RL@body
\end{verbatim}
That's all for the \LaTeX's wrapper!

## 5.2 \TeX code

At the beginning, we have to deal with multiple call and @’s catcode.

\begin{verbatim}
\csname RandomListLoaded\endcsname
\let\RandomListLoaded\endinput
\edef\RLAtCatcode{\the\catcode`\@}
\catcode`\@=11
\end{verbatim}

If we aren’t under \LaTeX then we need some \LaTeX commands. It’s just a copy of \LaTeX2ε code.

\begin{verbatim}
\ifx\@ifnextchar\@undefined
\Definition of \@ifnextchar.
\long\def\@ifnextchar#1#2#3{%
\let\reserved@d=#1%
\def\reserved@a{#2}%
\def\reserved@b{#3}%
\futurelet\@let@token\@ifnch
\def\@ifnch{%
\ifx\@let@token\@sptoken
\let\reserved@c\@xifnch
\else
\ifx\@let@token\reserved@d
\let\reserved@c\reserved@a
\else
\let\reserved@c\reserved@b
\fi
\fi
\reserved@c}
\def\:{\let\@sptoken=} %
\def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
\fi
\end{verbatim}

Definition of \PackageError and some \LaTeX functions when running under \TeX.

\begin{verbatim}
\ifx\PackageError\@undefined
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\def\@nnil{\@nil}%
\alloc@7\write\chardef\sixt@@n\@unused
\def\typeout#1{\immediate\write\@unused{#1}}%
\def\@spaces{\space\space\space\space}
\def\PackageError#1#2#3{%
\begingroup
\newlinechar'\^^J
\edef\RL@temp{#3}%
\expandafter\errhelp\expandafter{\RL@temp}%
\typeout{%
#1 error. See User’s Manual for further information.^^J
@spaces@spaces@spaces@spaces
Type @spacesH <return> @spaces for immediate help.}%
\errmessage{#2}%
\endgroup
\end{verbatim}

\end{verbatim}
We check if we work with an engine which contain at least e\TeX.  
\begin{verbatim}
\begingroup\newlinechar'\^^J\errhelp{Run under etex, pdftex, xetex, luatex, ... but not under \text{\TeX}}\typeout{\randomlist error. See User's Manual for further information.\^^J\spaces\spaces\spaces\spaces\errmessage{You can't use randomlist under \TeX{} without etex extension.}}\endgroup
\end{verbatim}
\@gobble Redefine \@gobble if needed.
\def\RL@ifempty#1{\ifcat\relax\detokenize{#1}\relax\else\expandafter\@firstoftwo\fi}
\RL@addtomacro We needs to add some code to some macros sometimes.
\newcount\RL@random
\newcount\RL@random@a
\newcount\RL@random@b
\def\RLsetrandomseed#1{\ifnum#1=0\global\RL@random \numexpr \time + \year \times \month \times \day \relax\else\global\RL@random \numexpr \ifnum#1<0 -\fi#1 \relax\fi}
If \RL@seed exists – that is, if we run under \LaTeX{} – we process the seed (option of \LaTeX{} package). Otherwise, we use the zero value.
Process the next random number using Linear Congruential Generator with Shrage's method.

Use the LCG with:

$$x_{n+1} = 7^5 \times x_n \pmod{2^{31} - 1}.$$  

For that we take:

- $7^5 = 16807$;
- $2^{31} - 1 = 2147483647$;
- $q = E\left(\frac{2^{31} - 1}{7^5}\right) = 127773$;
- $r = 2^{31} - 1 \pmod{7^5} = 2836$.

Then:

$$x_{n+1} = 7^5(x_n \pmod{q}) - r \times E\left(\frac{x_n}{q}\right).$$

If $x_{n+1} < 0$ then $x_{n+1} = x_{n+1} + 2^{31} - 1$.

Use `\RL@nextrand` to calculate a random integer between 0 (inclusive) and `#1` (exclusive). Store the result in macro `#2`.

```latex
\def\RLuniformdeviate#1#2{%
\RL@nextrand
\global\RL@random@a=\RL@random
\global\divide\RL@random@a 127773
\global\RL@random@b=\RL@random@a
\global\multiply\RL@random@b -2836
\global\multiply\RL@random@a -127773
\global\advance\RL@random\RL@random@b
\global\multiply\RL@random 16807
\global\advance\RL@random\RL@random@a
\ifnum\RL@random<0\global\advance\RL@random 2147483647\fi
\expandafter\edef\csname #2\endcsname{\number\RL@random@b}%
}
```

```tex
\RLuniformdeviate{\RL@random}{\RL@random@a}
\RLuniformdeviate{\RL@random}{\RL@random@b}
```

```latex
\RLuniformdeviate{\RL@random}{\RL@random@b}
\RLuniformdeviate{\RL@random}{\RL@random@a}
```
5.2.1 Introduction and first commands

@ifIsList Test if a list exists and then executes true code or false code. For that the list of list names is stored inside the token register @ListOfList. Each name is separated to the next one by a "\sep" markup.

\newtoks@ListofList
@ifIsList test if the list #1 exists. If yes then it executes the next argument else it executes the third argument. Test must be executed on an expanded argument.

\def@ifIsList#1{%
  \expandafter@ifIsList@\expandafter{#1}%
}\def@ifIsList@#1{%
  \def@@ifIsList##1#1\sep##2\@@ifIsList{%\csname @\ifx\empty##2\empty second\else first\fi oftwo\endcsname\expandafter\@@ifIsList\the\@ListOfList#1\sep\@@ifIsList}
\RL@lenof Shortcut allowing to get the len of a list
\def\RL@lenof#1{\csname #1-len\endcsname}
@ifIsListNotEmpty Test if a list exist and isn’t empty. If double yes then it executes the second argument else it executes the third one.
\newif\if@EmptyListFound\def@ifIsListNotEmpty#1{%
  \global@EmptyListFoundfalse\@ifIsList{#1}{%
    \ifnum\RL@lenof{#1}=0
      \global@EmptyListFoundtrue
    \else
      \expandafter@firstoftwo
    \fi
  \}@secondoftwo
\}@secondoftwo
\@NoListError Error for an unexisting list or an empty list.
\def//@NoListError#1{%
  \if@EmptyListFound
    \@EmptyListError{#1}%
  \global@EmptyListFoundfalse
  \else
    \PackageError{randomlist}{List #1 doesn’t exist}{Maybe you mistyped the list name?}%
  \fi
}\@EmptyListError Error for an empty list.
\def//@EmptyListError#1{%
{List #1 is empty}%
{Ask yourself why this list is empty.}%

\@OutOfRangeError Error for index out of range.
\def\@OutOfRangeError#1#2{%
\PackageError{randomlist}%
{Index #2 is greater than last index of list #1}%
{There aren't enough elements in the list.}%
%
}\RL@nameldef \long version of \@namedef.
\longdef\RL@nameldef#1{%
\long\expandafter\def\csname #1\endcsname %
}%
\RL@nameledef All the macros in randomlist are long ones. It's useless for now since there isn't argument but it's a precaution for the future.
\longdef\RL@nameledef#1{%
\long\expandafter\edef\csname #1\endcsname %
}%
\RL@namelgdef \long version of \@namegdef.
\longdef\RL@namelgdef#1{%
\long\expandafter\gdef\csname #1\endcsname %
}%
\RL@namelxdef \long version of \@namexdef.
\longdef\RL@namelxdef#1{%
\long\expandafter\xdef\csname #1\endcsname %
}%
\RL@let \let between two macros (with just the names)
\def\RL@let#1#2{%
\expandafter\let\csname#1\expandafter\endcsname\csname#2\endcsname %
}%
\RLfor Loop without group. Syntax is \RLfor<var>=<begin>to<end>\do
\def\RLfor#1=#2to#3\do{%
Set the variable.
\edef#1{\number\numexpr#2}%
Set \RL@sgncomp to <+ or >- if variable is greater or less than end
\edef\RL@sgncomp{\ifnum#1<\numexpr#3\relax>+\else<-\fi}%
Call auxiliary macro with five parameters:
\expandafter\RLfor@i
First argument (sub-recursive macro name build with the <variable> name).
\csname RLfor@i\@string#1\expandafter\endcsname\expandafter
Second argument (max).
{\number\numexpr#3}%
}
Third and fourth arguments since \RL@sgncomp is “<+” or “>-”.

\RL@sgncomp

fifth argument (variable name).

\#1%

Auxiliary macro:

• \#1 recursive macro name (like \RLfor@i i<var>);
• \#2 max integer;
• \#3 “<” or “>”;
• \#4 “+” or “-” (incrementation or decrementation);
• \#5 variable name;
• \#6 code to execute.

\long\def\RLfor@i#1#2#3#4#5#6{%  
Define the recursive submacro.
\def\#1{%  
While <var> isn’t greater than max
\unless\ifnum\#5\#3\#2\relax
In order to have a tail recursion.
\RL@doafterfi{%  
Execute the loop code.
\#6%
Increment <variable> by one.
\edef\#5{\number\numexpr\#5\#41\relax}%
And repeat.
\#1%
}\}%
}%

submacro recursive call.
\#1%
}

5.2.2 General list commands
\NewList

The main structure is the list. A list L is a collection of macros L-<n> where <n> is an index (starting from zero) and a macro L-len which store the len of the list, i.e. the last index plus one.

When a new list is created, its name is stored in @List0fList. A macro is also created for accessing data.
\def\NewList1{%  
@ifIsList{#1}{%
If a list with the same name exists then raise an error.

\PackageError{randomlist}
{List #1 already exists}
{Use \string\ClearList.}
%
%
When a list MyName is created, the macros \MyName and \MyName-len are created
and there will be macros \MyName-<n> to store data. Then randomlist prohibit
the name MyName for a list if the macro \MyName already exists.

\ifcsname #1\endcsname
\PackageError{randomlist}
{Command \csname#1\endcsname already exists}
{Creating list #1 defines a \csname#1\endcsname command.}
\else

If everything is fine, create the len macro which store the len of the list (starting
with 0);
\RL@nameldef{#1-len}{0}
append the list name to the list of names @ListOfList;
\@ListOfList\expandafter{\the\@ListOfList#1\sep}
and create the \Mylist[<index>] macro.
\expandafter\def\csname #1\endcsname[##1]{%If index is to big, the macro is \relax (a sort of undefined without error).
\ifnum##1>\csname#1-len\endcsname
\relax
\else
\csname #1-##1\endcsname
\fi
\fi
%
}%
%

\ClearList \ClearList erases a list. It sets the length to zero. There is no need to er ase all
the \Mylist-<index> macros.
\def\ClearList#1{%\@ifIsList{#1}{%Clear the list if it exists.
\RL@nameldef{#1-len}{0}%
}%}
{%\@NoListError{#1}}%
%
\CopyList Copy list #1 in list #2.
\def\CopyList#1#2{%\@ifIsList{#1}{%\@ifIsList{#2}{%\RL@let{#2-len}{#1-len}%\ifnum\RL@lenof{#1}>0
\RLfor\RL@iter=0 to \RL@lenof{#1}-1 \do{%\RL@let{#2-\RL@iter}{#1-\RL@iter}%
\RLfor\RL@iter=0 to \RL@lenof{#1}-1 \do{%\RL@let{#2-\RL@iter}{#1-\RL@iter}%
}
\InsertList Insert List #3 to the list #1 starting at index #2.
\def\InsertList#1#2#3{\% 
\@ifIsList{#1}{\% 
\@ifIsList{#3}{\% 
\ifnum #2>\RL@lenof{#1} \% 
\@@@@OutOfRangeError{#1}{#2} \% 
\else \% 
\ShiftList{#1}{#2}{\RL@lenof{#3}} \% 
\ifnum\RL@lenof{#3}>0 \% 
\RLfor\RL@iter=0 to \RL@lenof{#3}-1 \do{\% 
\RL@let{#1-\number\numexpr\RL@iter+#2}{#3-\RL@iter} \% 
\} \% 
\fi \% 
\fi \% 
\} \% 
\} \% 
\} \% 
\@@NoListError{#3} \% 
\} \% 
\@@NoListError{#1} \% 
\} \% 
\InsertList

\ShowList Macro for debugging purpose. First we declare some scratch count registers.
\newcount\RL@counti \newcount\RL@countii \newcount\RL@countiii \def\ShowList#1{\% 
\@ifIsList{#1}{\% 
\ifhmode\par\noindent\fi \% 
Typeset BEGIN{MyList}. As we typeset braces, we have to use \ttfamily, then put the material inside a group. 
\begingroup \% 
\ifdefined\ttfamily\ttfamily\ttfamily\else\tt\fi \% 
BEGIN\detokenize{{#1}} \% 
Typeset the number of elements. 
(\ifcase\RL@lenof{#1} \% 
\empty list \% 
\or \% 
1 element \% 
\else \% 
\RL@lenof{#1} elements \% 
\fi)\par \% 
Loop to typeset element one after one. 
\ifnum\RL@lenof{#1}>0 \% 
\parindent=1em \% 
\ShowList
\RLfor\RL@iter=0 to \RL@lenof{#1}-1 \do {
    #1[\RL@iter] = \expandafter\RL@meaning\csname #1-\RL@iter\endcsname
    \par
\}

\Typeset END{MyList}.
\noindent\END\detokenize{{#1}}\par
\endgroup
\{\@NoListError{#1}\%
\}

\RL@meaning Like \TeX primitive \meaning without prefix (\long) macro: ->:
\def\RL@meaning#1{\expandafter\RL@meaningi\meaning#1}
\expandafter\def\expandafter\RL@meaningi\expandafter#\expandafter1\string>{}

\CountList Count the number of elements in the list \#1. Store it in \#2.
\def\CountList#1#2{%\@ifIsList{#1}{\RL@nameledef{#2}{\RL@lenof{#1}}}\{\@NoListError{#1}\%
\}

\ShiftList Shift the elements of a list left or right. The syntax is:
\ShiftList{list name}{start}{shift}
where start is the first index to shift and shift the number of shifting. If shift is positive, it is a right shift. If shift is negative, it is a left shift.
\def\ShiftList#1#2#3{%\@ifIsList{#1}{%\unless\ifnum#3=0
If <start> is negative, raise an error.
\ifnum\numexpr#3<0
\PackageError{randomlist}{Negative index number}%
\else
\relax\PackageError{randomlist}{Index must be equal or greater than 0}%
\else
If <start> is greater than the lists length, raise an error.
\ifnum\numexpr#2>\RL@lenof{#1}\relax
\PackageError{randomlist}{Index \number\numexpr #2\relax space too big
\else
\relax\PackageError{randomlist}{Index must be equal or smaller than length of
the list}%
\else
\end{quote}

5.2.3 Writing and reading list commands

\ShiftList Shift the elements of a list left or right. The syntax is:
\ShiftList{list name}{start}{shift}
where start is the first index to shift and shift the number of shifting. If shift is positive, it is a right shift. If shift is negative, it is a left shift.
\def\ShiftList#1#2#3{%\@ifIsList{#1}{%\unless\ifnum#3=0
If <start> is negative, raise an error.
\ifnum\numexpr#3<0
\PackageError{randomlist}{Negative index number}%
\else
\relax\PackageError{randomlist}{Index must be equal or greater than 0}%
\else
If <start> is greater than the lists length, raise an error.
\ifnum\numexpr#2>\RL@lenof{#1}\relax
\PackageError{randomlist}{Index \number\numexpr #2\relax space too big
\else
\relax\PackageError{randomlist}{Index must be equal or smaller than length of
the list}%
\else
\end{quote}
Here we have $0 \leq \langle \text{shift} \rangle \leq \text{len}(\langle \text{list} \rangle)$.

If $\langle \text{shift} \rangle$ is positive, we process a right shifting: it's always possible.

```latex
\texttt{\RLfor} \RL@iter = \RL@lenof{\langle \text{list} \rangle} \texttt{to} \#2 \texttt{do}\{
  \RL@let{\#1-\number\numexpr\RL@iter+\langle \text{shift} \rangle\{\#1-\RL@iter}\%
\}
\}
\texttt{Empty the items out of shift part.}

\texttt{\RLfor} \RL@iter = \#2 \texttt{to} \#2 + \#3 - 1 \texttt{do}\{
  \RL@nameledef{\#1-\RL@iter}\{}
\}
\texttt{If the negative shifting is to big for index \#2 then raise an error.}
\PackageError{randomlist}\{}
  \{\text{Negative shift to big}\%
  \{\text{When negative, shift must not be greater than index}\%
  \else
  \texttt{Elsewhere, process the left shifting.}
\RLfor\RL@iter=\#2 \texttt{to} \RL@lenof{\langle \text{list} \rangle} \texttt{do}\{
  \RL@let{\#1-\number\numexpr\RL@iter+\langle \text{shift} \rangle\{\#1-\RL@iter}\%
\}
\fi
\fi
\texttt{Set the list length for both positive and negative shifting.}
\RL@nameledef{\#1-\RL@lenof{\langle \text{list} \rangle}+\langle \text{shift} \rangle}\%
\fi\fi\fi
\}
\@NoListError{\langle \text{list} \rangle}\}%
```

\InsertLastItem Add an element $\langle \text{shift} \rangle$ at the end of the list $\langle \text{list} \rangle$.
\long\def\InsertLastItem\langle \text{list} \rangle\#2\%
\@ifIsList{\langle \text{list} \rangle}\%
  \RL@nameledef{\#1-\RL@lenof{\langle \text{list} \rangle}}\{}
  \RL@nameledef{\#1-\RL@lenof{\langle \text{list} \rangle}+1}\%
\}\\%\fi\fi\fi
```
```
\InsertFirstItem Add an element $\langle \text{shift} \rangle$ at the beginning of the list $\langle \text{list} \rangle$. For that, shift right all the element and then put $\langle \text{shift} \rangle$ at $L[0]$.
\long\def\InsertFirstItem\langle \text{list} \rangle\#2\%
\InsertItem{\langle \text{list} \rangle}{0}{\langle \text{shift} \rangle}\%
```
```
\InsertItem Add an element $\langle \text{shift} \rangle$ at the position $\#2$ of the list $\langle \text{list} \rangle$. For that, pass from $L[0]$ to $L[\#2-1]$ then shift right from $L[\#2]$ to $L[\#2]$ and finally put $\langle \text{shift} \rangle$ at $L[\#2]$. To do this, we must have $\#2 \geq \text{len}(\langle \text{list} \rangle)$.
\long\def\InsertItem\langle \text{list} \rangle\#2\%
```
```
```
\InsertRandomItem Insert element \#2 in a random position of list \#1.
\long\def\InsertRandomItem#1#2{%  
  \@ifIsList{#1}{%  
    \RLuniformdeviate{\RL@lenof{#1}+1}{RL@temp}%  
    \InsertItem{#1}{\RL@temp}{#2}%  
  }{%  
    \@NoListError{#1}%  
  }%  
}\ExtractFirstItem Extract the first element of list \#1 and store it in \#2.
\def\ExtractFirstItem#1#2{%  
  \@ifIsList{#1}{%  
    \ExtractItem{#1}{0}{#2}%  
  }{%  
    \@NoListError{#1}%  
  }%  
}\ExtractLastItem Extract the last element of list \#1 and store it in \#2.
\def\ExtractLastItem#1#2{%  
  \@ifIsListNotEmpty{#1}{%  
    \RL@let{#2}{#1-\number\numexpr\RL@lenof{#1}-1}%  
    \RL@nameledef{#1-len}{\number\numexpr\RL@lenof{#1}-1}%  
  }{%  
    \@NoListError{#1}%  
  }%  
}\ExtractItem Extract the element at the position \#2 of the list \#1 and store it in \#3.
\def\ExtractItem#1#2#3{%  
  \@ifIsListNotEmpty{#1}{%  
    \RL@let{#3}{#1-#2}%  
    \ShiftList{#1}{#2+1}{-1}%  
  }{%  
    \@NoListError{#1}%  
  }%  
}\ExtractRandomItem Extract element in a random position of list \#1 and store it in \#2.
\def\ExtractRandomItem#1#2{%  
  \@ifIsListNotEmpty{#1}{%  
    \RLuniformdeviate{\RL@lenof{#1}}{RL@temp}%  
    \ExtractItem{#1}{\RL@temp}{#2}%  
  }%
\ExtractList \texttt{ExtractList} extract a list from a list. There are four arguments:

- \#1 is the list from which the extraction is made;
- \#2 is the starting index of extraction;
- \#3 is the ending index of extraction;
- \#4 is the list which receive the extracted list.

In order to do something, \#1 and \#2 must be lists, and indexes \#2 and \#3 must be inside list \#1.

\GetFirstItem \texttt{GetFirstItem} Get the first element of list \#1 and store it in \#2.

\GetLastItem \texttt{GetLastItem} Get the last element of list \#1 and store it in \#2.
\GetItem Get the element of rank \#2 of list \#1 and store it in \#3.
\begin{verbatim}
\def\GetItem#1#2#3{\% 
  \@ifIsListNotEmpty{#1} \%
   \ifnum\numexpr\RL@lenof{#1}-1-#2<0 \%
     \OutOfRangeError{#1}{#2}\%
   \else \%
     \RL@let{#3}{#1-#2}\%
   \fi \%
  \else \%
    \@NoListError{#1}\%
  \fi}
\end{verbatim}

\GetRandomItem Get element in a random position of list \#1 and store it in \#2.
\begin{verbatim}
\def\GetRandomItem#1#2{\% 
  \@ifIsListNotEmpty{#1} \%
   \RLuniformdeviate{\RL@lenof{#1}}{RL@temp}%
   \GetItem{#1}{\RL@temp}{#2}%
  \else \%
    \@NoListError{#1}\%
  \fi}
\end{verbatim}

\GetList \GetList copy a sub-list from a list. There are four arguments:

- \#1 is the list from which the reading is made;
- \#2 is the starting index of extraction;
- \#3 is the ending index of extraction;
- \#4 is the list which receive the readen items.

\begin{verbatim}
\def\GetList#1#2#3#4{\%
  \@ifIsList{#1}{\%
    \@ifIsList{#4}{\%
      \ifnum#2<\RL@lenof{#1} \%
        \@ifIsList{#3}{\% \ifnum#2>#3 \relax\%
          \RL@nameldef{#4-len}{0}\%
        \else \%
          \RL@for\RL@iter=#2 to #3 \do{\%
            \RL@let{#4-\number\numexpr \RL@iter - #2}{#1-\RL@iter}\%
          }\%
          \RL@nameledef{#4-len}{\number\numexpr #3 - #2 + 1}\%
        \fi \%
        \else \%
          \OutOfRangeError{#1}{#3}\%
        \fi\%
      \else \%
        \OutOfRangeError{#4}{#3}\%
      \fi\%
    \else \%
      \@OutOfRangeError{#1}{#3}\%
    \fi\%
  \else \%
    \@OutOfRangeError{#4}{#3}\%
  \fi}
\end{verbatim}

In order to do something, \#1 and \#2 must be lists, and indexes \#2 and \#3 must be inside list \#1.

\begin{verbatim}
\def\GetList#1#2#3#4{\% 
  \@ifIsList{#1}{\% 
    \@ifIsList{#4}{\% 
      \ifnum#2<\RL@lenof{#1}{\% 
        \ifnum#3<\RL@lenof{#1}{\% 
          \ifnum#2>#3 \relax\%
            \RL@nameldef{#4-len}{0}\%
          \else \%
            \RL@for\RL@iter=#2 to #3 \do{\%
              \RL@let{#4-\number\numexpr \RL@iter - #2}{#1-\RL@iter}\%
            }\%
            \RL@nameledef{#4-len}{\number\numexpr #3 - #2 + 1}\%
          \fi\%
        \else \%
          \OutOfRangeError{#4}{#3}\%
        \fi\%
      \else \%
        \OutOfRangeError{#1}{#3}\%
      \fi\%
    \else \%
      \@OutOfRangeError{#4}{#3}\%
    \fi\%
  \else \%
    \@OutOfRangeError{#1}{#3}\%
  \fi\%
\end{verbatim}
\SetFirstItem Set the first element of list #1 with value #2.
\long\def\SetFirstItem#1#2{\SetItem{#1}{0}{#2}}

\SetLastItem Set the last element of list #1 with value #2.
\long\def\SetLastItem#1#2{\SetItem{#1}{\number\numexpr\RL@lenof{#1}-1}{#2}}

\SetItem Set the #2 element of list #1 with value #3.
\long\def\SetItem#1#2#3{\@ifIsListNotEmpty{#1}{\ifnum\numexpr\RL@lenof{#1}-1-#2<0\@OutOfRangeError{#1}{#2}\else\RL@nameldef{#1-#2}{#3}\fi}\@NoListError{#1}}

\SetRandomItem Set element in a random position of list #1 with value #2.
\long\def\SetRandomItem#1#2{\@ifIsListNotEmpty{#1}{\RLuniformdeviate{\RL@lenof{#1}}{RL@temp}\SetItem{#1}{\RL@temp}{#2}}\@NoListError{#1}}

\SetList allow to give multiple values to a list. This function acts like a repetition of \InsertLastItem.
\def\SetList#1#2{\@ifIsList{#1}{\ClearList{#1}\def\RL@name{#1}\RL@setlist#2,\@nil,\@NoListError{#1}}\@NoListError{#1}}
\long\def\RL@setlist#1,{\@NoListError{#1}}
5.2.4 Loop on list

\ForEachFirstItem \ForEachFirstItem typesets #3 for each element of the list #1 extracting the actual first element (stored in #2).  
\long\def\ForEachFirstItem#1#2#3{\%  
\begingroup  
\RLfor \RL@var = 0 to \RL@lenof{#1}-1 \do{\%  
\ExtractFirstItem{#1}{#2}\%  
#3\%  
\}\%  
\endgroup  
}\%}

\ForEachLastItem \ForEachLastItem typesets #3 for each element of the list #1 extracting the actual last element (stored in #2).  
\long\def\ForEachLastItem#1#2#3{\%  
\begingroup  
\RLfor \RL@var = 0 to \RL@lenof{#1}-1 \do{\%  
\ExtractLastItem{#1}{#2}\%  
#3\%  
\}\%  
\endgroup  
}\%}

\ForEachRandomItem \ForEachRandomItem typesets #3 for each element of the list #1 extracting randomly an element (stored in #2).  
\long\def\ForEachRandomItem#1#2#3{\%  
\begingroup  
\RLfor \RL@var = 0 to \RL@lenof{#1}-1 \do{\%  
\ExtractRandomItem{#1}{#2}\%  
#3\%  
\}\%  
\endgroup  
}\%}

5.2.5 Database

\ReadFieldItem Macro \ReadFieldItem read a field in a record.  
A record is a sequence of groups, each group is a field.  
- \#1 is the record (sequence of groups;  
- \#2 is the index of item (starting at zero);  
- \#3 is the macro name which store the field.  
\long\def\ReadFieldItem#1#2#3{\%
Store the field’s index.
\RL@counti \#2\relax

Call the recursive macro
\expandafter\RL@ReadFieldItem\#1\@nil

Store the result in macro \#3.
\expandafter\let\csname#3\endcsname\RL@temp
}

In fact the first recursive call check for a left brace. A record must contain at least
one field otherwise an error message is raised.
\long\def\RL@ReadFieldItem{%
\ifnextchar\bgroup{\RL@@ReadFieldItem}{\RL@@ReadFieldItemError}%
}
\long\def\RL@@ReadFieldItem#1{%
\ifnum\RL@counti=\z@
\def\RL@temp{#1}%
\expandafter\RL@@ReadFieldItemEnd
\else
\advance\RL@counti \m@ne
\expandafter\RL@ReadFieldItem
\fi
}
\long\def\RL@@ReadFieldItemEnd#1\@nil{}%
\long\def\RL@@ReadFieldItemError#1\@nil{%
\PackageError{randomlist}{There aren’t enough fields in the record}{Pay attention that field number starts from zero.}%
}

\ReadFileList First, we look for special delimiters for fields and strings. By default, the delimiter for fields is the comma and the delimiter for strings is the double quote.
\def\RL@SetDelimiters#1#2#3\@nil{%
\let\RL@accu\empty
\expandafter\RL@markstrings@i##1#2#2\@nil#2%
\let##1=\RL@accu
\unless\ifx\@nnil\RL@current
\RL@addtomacro\RL@accu{\RL@string{##2}}%
\expandafter\RL@markstrings@i
\fi
}

\def\RL@unmarkstrings##1{%
\let\RL@accu\empty
\expandafter\RL@unmarkstrings@i##1#2\@nil#2%
\unless\ifx\@nnil\RL@current
\RL@addtomacro\RL@accu{\RL@string{##2}}%
\expandafter\RL@unmarkstrings@i
\fi
}
\def\RL@markstrings##1{%
\let\RL@accu\empty
\expandafter\RL@markstrings@i##1#2#2\@nil#2%
\let##1=\RL@accu
\unless\ifx\@nnil\RL@current
\RL@addtomacro\RL@accu{\RL@string{##2}}%
\expandafter\RL@markstrings@i
\fi
}
\def\RL@unmarkstrings##1{%
\let\RL@accu\empty
\expandafter\RL@unmarkstrings@i##1#2\@nil#2%
\unless\ifx\@nnil\RL@current
\RL@addtomacro\RL@accu{\RL@string{##2}}%
\expandafter\RL@unmarkstrings@i
\fi
}}
The macro \texttt{\ReadFileList} uses a handle for the reading file. It needs also a macro to detect \texttt{\par}

At first, \texttt{\ReadFileList} check for an optional argument giving delimiters. By default, delimiters are comma for field separator and double quote for string delimiter.

\texttt{\def\ReadFileList{[@ifnextchar[{[@ReadFileList}{[@ReadFileList[","}]}}}

- #1 contains the delimiters;
- #2 is the data base name;
- #3 is the file name.

\texttt{\def[@ReadFileList[1][2][3]{[\texttt{\openin\RL@hdle = #3}}
\texttt{\ifeof\RL@hdle = #3}}
\texttt{\PackageError{randomlist}{File \texttt{#3} doesn't exist}]
\texttt{\else
\texttt{\PackageError{randomlist}{Verify its name, its extension, its location, its permissions.}}
\texttt{\else
\texttt{\PackageError{randomlist}{Verify its name, its extension, its location, its permissions.}}
\else
If the optionnal argument is empty then raise an error and take the comma and
the double quote instead.
\PackageError{randomlist}{Optional argument empty: [,"] inserted}{Do not leave an optional argument empty}
\RL@SetDelimiters","@nil
}

Else add double quote to for security.
\RL@SetDelimiters#1"@nil%

The main loop read each line of the file. Don't process anything if the line is
empty (it could be the very end of the file).
\loop
\read\RL@hdle to \RL@buffer
\unless\ifx\RL@buffer@oppar
Mark the string
\RL@markstrings\RL@buffer
and process the fields.
\RL@parsefields\RL@buffer
Save current record with fields, that is, with sequence of groups.
\def\RL@accuA{\InsertLastItem{#2}}%
\expandafter\RL@accuA\expandafter{\RL@buffer}%
\fi
\ifeof\RL@hdle\else
\repeat
\fi

Check for a heading space.
\def\RL@ifspacefirst#1{%\RL@ifspacefirst@i#1A @nil
}
\expandafter\def\expandafter\RL@ifspacefirst@i#1\space#2@nil#1{\RL@ifempty{#1}%
% Remove all spaces at the start of argument (macro).
\def\RL@removefirstspaces#1{%\expandafter\RL@ifspacefirst\expandafter{#1}
\expandafter\RL@ifspacefirst\expandafter{#1}#1A @nil
}
\expandafter\def\expandafter\RL@removefirstspaces@i\space#1@nil#1{\RL@ifempty{#1}%
% Store |~00|'s catcode
\edef\RL@restorecatcodezero{\catcode0=\number\catcode0\relax}
then set this catcode to other catcode.
\catcode0=12

Remove all heading and trailing spaces of argument (macro).
\def\RL@removelastspaces#1{\expandafter\def\expandafter#1\expandafter{\romannumeral\expandafter\RL@removelastspaces@i\expandafter\relax#1^^00 ^^00\@nil}}
\def\RL@removelastspaces@i#1 ^^00\@nil{\RL@ifspacefirst{#1}{\RL@removelastspaces@i#1 ^^00 \@nil}{\expandafter\z@\@gobble#1}}

% Restore |^^00|'s catcode.
\RL@restorecatcodezero

At the very end of the package, we restore the @’s catcode.
\catcode`@=\RLAtCatcode\relax
A  File pythagoras.dat

This file contains Pythagorean triples which have three digits. There isn’t all these triple. In fact the triple are built with the famous formula \((u^2 - v^2, 2uv, u^2 + v^2)\) with \(u\) and \(v\) positive integers such \(u > v\). Here are only the hundred first triples with three digits.

1 119, 120, 169
2 108, 144, 180
3 153, 104, 185
4 144, 130, 194
5 133, 156, 205
6 120, 182, 218
7 105, 208, 233
8 180, 112, 212
9 171, 140, 221
10 160, 168, 232
11 147, 196, 245
12 132, 224, 260
13 115, 252, 277
14 209, 120, 241
15 200, 150, 250
16 189, 180, 261
17 176, 210, 274
18 161, 240, 289
19 144, 270, 306
20 125, 300, 325
21 104, 330, 346
22 240, 128, 272
23 231, 160, 281
24 220, 192, 292
25 207, 224, 305
26 192, 256, 320
27 175, 288, 337
28 156, 320, 356
29 135, 352, 377
30 112, 384, 400
31 280, 102, 298
32 273, 136, 305
33 264, 170, 314
34 253, 204, 325
35 240, 238, 338
36 225, 272, 353
37 208, 306, 370
38 189, 340, 389
39 168, 374, 410
40 145, 408, 433
41 120, 442, 458
42 315, 108, 333
43 308, 144, 340
44 299, 180, 349
45 288, 216, 360
275,252,373
260,288,388
243,324,405
224,360,424
203,396,445
180,432,468
155,468,493
128,504,520
352,114,370
345,152,377
336,190,386
325,228,397
312,266,410
297,304,425
280,342,442
261,380,461
240,418,482
217,456,505
192,494,530
162,532,557
136,570,586
105,608,617
391,120,409
384,160,416
375,200,425
364,240,436
351,280,449
336,320,464
319,360,481
300,400,500
297,440,521
256,480,544
231,520,569
204,560,596
175,600,625
144,640,656
111,680,689
432,126,450
425,168,457
416,210,466
405,252,477
392,294,490
377,336,505
360,378,522
341,420,541
320,462,562
297,504,585
272,546,610
245,588,637
216,630,666
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>185,672,697</td>
</tr>
<tr>
<td>97</td>
<td>152,714,730</td>
</tr>
<tr>
<td>98</td>
<td>117,756,765</td>
</tr>
<tr>
<td>99</td>
<td>475,132,493</td>
</tr>
<tr>
<td>100</td>
<td>468,176,500</td>
</tr>
</tbody>
</table>
B File pupils.dat

This file shows a first line which isn't a data line.

<table>
<thead>
<tr>
<th>Name</th>
<th>FirstName</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aho</td>
<td>Alfred</td>
<td>A</td>
</tr>
<tr>
<td>Babbage</td>
<td>Charles</td>
<td>A</td>
</tr>
<tr>
<td>Chaitin</td>
<td>Gregory</td>
<td>B</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>Edsger</td>
<td>A</td>
</tr>
<tr>
<td>Eckert</td>
<td>John Preper</td>
<td>B</td>
</tr>
<tr>
<td>Floyd</td>
<td>Robert</td>
<td>B</td>
</tr>
<tr>
<td>G&quot;odel</td>
<td>Kurt</td>
<td>A</td>
</tr>
<tr>
<td>Huffman</td>
<td>David</td>
<td>B</td>
</tr>
<tr>
<td>Ichbiah</td>
<td>Jean</td>
<td>A</td>
</tr>
<tr>
<td>Joshi</td>
<td>Aravind</td>
<td>C</td>
</tr>
<tr>
<td>Knuth</td>
<td>Donald</td>
<td>C</td>
</tr>
<tr>
<td>Lovelace</td>
<td>Ada</td>
<td>A</td>
</tr>
<tr>
<td>Moore</td>
<td>Gordon</td>
<td>A</td>
</tr>
<tr>
<td>Neumann (Von)</td>
<td>John</td>
<td>A</td>
</tr>
<tr>
<td>Ouserhout</td>
<td>John</td>
<td>B</td>
</tr>
<tr>
<td>Pascal</td>
<td>Blaise</td>
<td>A</td>
</tr>
<tr>
<td>Ritchie</td>
<td>Dennis</td>
<td>C</td>
</tr>
<tr>
<td>Shannon</td>
<td>Claude</td>
<td>C</td>
</tr>
<tr>
<td>Thompson</td>
<td>Ken</td>
<td>A</td>
</tr>
<tr>
<td>Ullman</td>
<td>Jeffrey</td>
<td>B</td>
</tr>
<tr>
<td>Vixie</td>
<td>Paul</td>
<td>B</td>
</tr>
<tr>
<td>Wall</td>
<td>Larry</td>
<td>B</td>
</tr>
<tr>
<td>Yao</td>
<td>Andrew Chi-Chih</td>
<td>C</td>
</tr>
<tr>
<td>Zuse</td>
<td>Konrad</td>
<td>C</td>
</tr>
</tbody>
</table>
### C File comets.dat

This file uses lines which aren't data lines and weird separators.

<table>
<thead>
<tr>
<th>Comet</th>
<th>Discover</th>
<th>Year</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1P/Halley</td>
<td>Halley</td>
<td>1758</td>
<td>76.09</td>
</tr>
<tr>
<td>2P/Encke</td>
<td>Encke</td>
<td>1786</td>
<td>3.30</td>
</tr>
<tr>
<td>3D/Biela</td>
<td>Biela</td>
<td>1826</td>
<td>6.62</td>
</tr>
<tr>
<td>4P/Faye</td>
<td>Faye</td>
<td>1843</td>
<td>7.55</td>
</tr>
<tr>
<td>5D/Brorsen</td>
<td>Brorsen</td>
<td>1846</td>
<td>5.46</td>
</tr>
<tr>
<td>6P/d’Arrest</td>
<td>d’Arrest</td>
<td>1851</td>
<td>6.54</td>
</tr>
<tr>
<td>7P/Pons-Winnecke</td>
<td>Pons &amp; Winnecke</td>
<td>1819</td>
<td>6.36</td>
</tr>
<tr>
<td>8P/Tuttle</td>
<td>Tuttle</td>
<td>1858</td>
<td>13.58</td>
</tr>
<tr>
<td>9P/Tempel</td>
<td>Tempel</td>
<td>1867</td>
<td>5.52</td>
</tr>
<tr>
<td>10P/Tempel</td>
<td>Tempel</td>
<td>1873</td>
<td>5.38</td>
</tr>
<tr>
<td>11P/Tempel-Swift-LINEAR</td>
<td>Tempel, Swift</td>
<td>1869</td>
<td>6.37</td>
</tr>
<tr>
<td>12P/Pons-Brooks</td>
<td>Pons &amp; Brooks</td>
<td>1812</td>
<td>70.85</td>
</tr>
<tr>
<td>13P/Olbers</td>
<td>Olbers</td>
<td>1815</td>
<td>69.5</td>
</tr>
<tr>
<td>14P/Wolf</td>
<td>Wolf</td>
<td>1884</td>
<td>8.74</td>
</tr>
<tr>
<td>15P/Finlay</td>
<td>Finlay</td>
<td>1886</td>
<td>6.50</td>
</tr>
<tr>
<td>16P/Brooks</td>
<td>Brooks</td>
<td>1889</td>
<td>6.14</td>
</tr>
<tr>
<td>17P/Holmes</td>
<td>Holmes</td>
<td>1892</td>
<td>6.89</td>
</tr>
<tr>
<td>18D/Perrine-Mrkos</td>
<td>Perrine &amp; Mrkos</td>
<td>1896</td>
<td></td>
</tr>
<tr>
<td>19P/Borrelly</td>
<td>Borrelly</td>
<td>1904</td>
<td>6.85</td>
</tr>
<tr>
<td>20D/Westphal</td>
<td>Westphal</td>
<td>1852</td>
<td>61.8</td>
</tr>
</tbody>
</table>
