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

Jean-Côme Charpentier* Christian Tellechea†

July 13, 2016

*jean-come.charpentier@wanadoo.fr
†unbonpetit@openmailbox.org

Contents

1 Overview

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

In "randomlist", the word "List" must be understound 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:

```
\NewList{MyList}
\NewList{MyList}
```

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:

```
\NewList{def}
```

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 typeseted at the right side and result is showed at the left side):

```
BEGIN{MyList} (empty list)
END{MyList}
```

```
\NewList{MyList}
\ShowList{MyList}
```

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 behaves 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:

`\InsertFirstItem`	`\ExtractFirstItem`	`\SetFirstItem`	`\GetFirstItem`
`\InsertLastItem`	`\ExtractLastItem`	`\SetLastItem`	`\GetLastItem`
`\InsertItem`	`\ExtractItem`	`\SetItem`	`\GetItem`
`\InsertRandomItem`	`\ExtractRandomItem`	`\SetRandomItem`	`\GetRandomItem`

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

Finally, we have `\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:

```
\ShiftList{<list>}{<start>}{<nb>}
```

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

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

Usually, you don't need `\ShiftList`: the other commands, especially `\ExtractList` and `\InsertList` are enough.

2.2.1 Insert commands

`\InsertFirstItem` `\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:

```
BEGIN{MyList} (1 element)
  MyList[0] = First value
END{MyList}
BEGIN{MyList} (2 elements)
  MyList[0] = Second value
  MyList[1] = First value
END{MyList}
BEGIN{MyList} (3 elements)
  MyList[0] = Third value
  MyList[1] = Second value
  MyList[2] = First value
END{MyList}
```

```
\NewList{MyList}
\InsertFirstItem{MyList}{First value}
\ShowList{MyList}
\InsertFirstItem{MyList}{Second value}
\ShowList{MyList}
\InsertFirstItem{MyList}{Third value}
\ShowList{MyList}
```

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{MyList} (4 elements)
  MyList[0] = end
  MyList[1] = special^
  MyList[2] = \textbf {text}
  MyList[3] = {two}{groups}
END{MyList}
```

```
\NewList{MyList}
\InsertFirstItem{MyList}{{two}{groups}}
\InsertFirstItem{MyList}{\textbf{text}}
\InsertFirstItem{MyList}{special^}
\InsertFirstItem{MyList}{end}
\ShowList{MyList}
```

`\InsertLastItem` `\InsertLastItem` is like `\InsertFirstItem` but the insertion is made at the end of the list. As for the previous command, it takes two arguments: the list name and the value to insert. The previous example give:

```
BEGIN{MyList} (4 elements)
  MyList[0] = {two}{groups}
  MyList[1] = \textbf {text}
  MyList[2] = special^
  MyList[3] = end
END{MyList}
```

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

With this command, the list behaves as queue.

`\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:

```
BEGIN{MyList} (4 elements)
  MyList[0] = first
  MyList[1] = \textbf {second}
  MyList[2] = special^
  MyList[3] = end
END{MyList}
BEGIN{MyList} (7 elements)
  MyList[0] = first
  MyList[1] = \textbf {second}
  MyList[2] = \textbf {other!}
  MyList[3] = \textbf {insert!}
  MyList[4] = special^
  MyList[5] = end
  MyList[6] = real end
END{MyList}
```

```
\NewList{MyList}
\InsertLastItem{MyList}{first}
\InsertLastItem{MyList}{\textbf{second}}
\InsertLastItem{MyList}{special^}
\InsertLastItem{MyList}{end}
\ShowList{MyList}
\InsertItem{MyList}{2}{\textbf{insert!}}
\InsertItem{MyList}{2}{\textbf{other!}}
\InsertItem{MyList}{6}{real end}
\ShowList{MyList}
```

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.

`\InsertRandomItem` This is the first command which use random numbers. We will see later how to manage random numbers themselves.

`\InsertRandomItem` command works as the `\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:

```
BEGIN{MyList} (5 elements)
  MyList[0] = fifth
  MyList[1] = first
  MyList[2] = second
  MyList[3] = fourth
  MyList[4] = third
END{MyList}
```

```
\NewList{MyList}
\InsertRandomItem{MyList}{first}
\InsertRandomItem{MyList}{second}
\InsertRandomItem{MyList}{third}
\InsertRandomItem{MyList}{fourth}
\InsertRandomItem{MyList}{fifth}
\ShowList{MyList}
```

`\InsertList` `\InsertList` allows to do in one shot what the `\InsertItem` can do in several ones. The principle of this command is to insert all the items of a list inside a second one. You have to give three arguments: the list which receive, the index to start insertion, and the list to insert. For instance:

```
BEGIN{MyList} (5 elements)
  MyList[0] = first in M
  MyList[1] = second in O
  MyList[2] = third in O
  MyList[3] = fourth in M
  MyList[4] = fifth in M
END{MyList}
BEGIN{OtherList} (2 elements)
  OtherList[0] = second in O
  OtherList[1] = third in O
END{OtherList}
```

```
\NewList{MyList}
\NewList{OtherList}
\InsertLastItem{MyList}{first in M}
\InsertLastItem{MyList}{fourth in M}
\InsertLastItem{MyList}{fifth in M}
\InsertLastItem{OtherList}{second in O}
\InsertLastItem{OtherList}{third in O}
\InsertList{MyList}{1}{OtherList}
\ShowList{MyList}
\ShowList{OtherList}
```

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:

```
\NewList{MyList}
\NewList{OtherList}
\InsertLastItem{MyList}{first}
\InsertLastItem{MyList}{second}
\InsertLastItem{MyList}{third}
\InsertList{MyList}{1}{OtherList}
\ShowList{MyList}
```

```
BEGIN{MyList} (3 elements)
  MyList[0] = first
  MyList[1] = second
  MyList[2] = third
END{MyList}
```

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:

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\ExtractFirstItem{MyList}{MyMacro}
The first element was ``\MyMacro''.

\ShowList{MyList}
```

```
The first element was "TEX".
BEGIN{MyList} (3 elements)
  MyList[0] = is
  MyList[1] = very
  MyList[2] = powerful
END{MyList}
```

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,

```
\NewList{MyList}
\ExtractFirstItem{MyList}{MyMacro}
```

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:

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\ExtractLastItem{MyList}{MyMacro}
The last element was ``\MyMacro''.

\ShowList{MyList}
```

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

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\ExtractItem{MyList}{2}{MyMacro}
The third element was ``\MyMacro''.

\ShowList{MyList}
```

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

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\ExtractRandomItem{MyList}{MyMacro}
``\MyMacro'' was extracted.

\ShowList{MyList}
```

“is” was extracted.
```
BEGIN{MyList} (3 elements)
  MyList[0] = \TeX
  MyList[1] = very
  MyList[2] = powerful
END{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.
```

`\ExtractList` The commands `\Extract...Item` extract one item and store it in a macro. With the command `\ExtractList` we can extract several items and put them in a list. `\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:

```
\NewList{MyList}
\NewList{OtherList}
\InsertLastItem{MyList}{first}
\InsertLastItem{MyList}{second}
\InsertLastItem{MyList}{third}
\InsertLastItem{MyList}{fourth}
\InsertLastItem{MyList}{fifth}
\InsertLastItem{MyList}{sixth}
\ExtractList{MyList}{2}{4}{OtherList}

\ShowList{MyList}
\ShowList{OtherList}
```

```
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}
```

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

```
\NewList{MyList}
\NewList{OtherList}
\InsertLastItem{MyList}{first}
\InsertLastItem{MyList}{second}
\InsertLastItem{MyList}{third}
\InsertLastItem{MyList}{fourth}
\InsertLastItem{MyList}{fifth}
\InsertLastItem{MyList}{sixth}
\ExtractList{MyList}{2}{2}{OtherList}

\ShowList{MyList}
\ShowList{OtherList}
```

```
BEGIN{MyList} (5 elements)
  MyList[0] = first
  MyList[1] = second
  MyList[2] = fourth
  MyList[3] = fifth
  MyList[4] = sixth
END{MyList}
BEGIN{OtherList} (1 element)
  OtherList[0] = third
END{OtherList}
```

2.2.3 Set commands

`\SetFirstItem` The commands `\Set...Item` modify the existing values of list. `\SetFirstItem` modify the first value.

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\SetFirstItem{MyList}{\LaTeX}
\ShowList{MyList}
```

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

If a list is empty, there is the classic error message about empty list.

\SetLastItem \SetLastItem acts like \SetFirstItem but at the end of the list.

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

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

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\SetItem{MyList}{2}{quiet}
\ShowList{MyList}
```

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

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\SetItem{MyList}{4}{isn't it?}
```

gives the error message:

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

\SetRandomItem \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)
  MyList[0] = \TeX
  MyList[1] = snap!
  MyList[2] = really
  MyList[3] = very
  MyList[4] = powerful
END{MyList}
```

```
\NewList{MyList}
\InsertLastItem{MyList}{\TeX}
\InsertLastItem{MyList}{is}
\InsertLastItem{MyList}{really}
\InsertLastItem{MyList}{very}
\InsertLastItem{MyList}{powerful}
\SetRandomItem{MyList}{snap!}
\ShowList{MyList}
```

\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 \SetList. Items are separated with comma. For instance:

```
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,
  powerful}
\ShowList{MyList}
```

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,%
  powerful}
\ShowList{MyList}
```

\CopyList Copy a list in another one. Both lists must exists: this command don't create list since only \NewList can do that. Here is an example:

```
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,very,%
                 powerful}
\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}
```

```
\NewList{MyList}
\SetList{MyList}{\TeX,is,so,cute}
\GetFirstItem{MyList}{MyMacro}
The first element is ``\MyMacro''

\ShowList{MyList}
```

\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}
```

```
\NewList{MyList}
\SetList{MyList}{\TeX,is,so,cute}
\GetItem{MyList}{2}{MyMacro}
The third element is ``\MyMacro''

\ShowList{MyList}
```

Package randomlist offers an other syntax to access to an item: \<namelist>[<index>]. Thus, we can write the previous example like that:

```
The third element is "so"
```

```
\NewList{MyList}
\SetList{MyList}{\TeX,is,so,cute}
The third element is ``\MyList[2]''
```

\GetRandomItem \GetRandomItem give the value of a randomly selected element of a list.

```
\NewList{MyList}
\SetList{MyList}{\TeX,is,so,cute}
\GetRandomItem{MyList}{MyMacro}
The random element is ``\MyMacro''

\ShowList{MyList}
```

The random element is "is"
BEGIN{MyList} (4 elements)
  MyList[0] = \TeX
  MyList[1] = is
  MyList[2] = so
  MyList[3] = cute
END{MyList}

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

```
\NewList{MyList}
\NewList{OtherList}
\SetList{MyList}{X1,X2,X3,X4,X5,X6}
\GetList{MyList}{2}{4}{OtherList}
\ShowList{MyList}
\ShowList{OtherList}
```

BEGIN{MyList} (6 elements)
  MyList[0] = X1
  MyList[1] = X2
  MyList[2] = X3
  MyList[3] = X4
  MyList[4] = X5
  MyList[5] = X6
END{MyList}
BEGIN{OtherList} (3 elements)
  OtherList[0] = X3
  OtherList[1] = X4
  OtherList[2] = X5
END{OtherList}

Contrary to what \ExtractItem do, \GetList 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 product 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. For these commands, the macro name is given without the backslash. 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 Charles Babbage
blah blah blah...
Test for Alfred Aho
blah blah blah...
Test for Edsger Dijkstra
blah blah blah...
Test for Gregory Chaintin
blah blah blah...

```
BEGIN{Pupils} (4 elements)
  Pupils[0] = Alfred Aho
  Pupils[1] = Charles Babbage
  Pupils[2] = Gregory Chaintin
  Pupils[3] = Edsger Dijkstra
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):

Alfred and Ada are computer scientists
Alfred and Adele are computer scientists
Alfred and Grace are computer scientists
Gregory and Grace are computer scientists
Gregory and Adele are computer scientists
Gregory and Ada are computer scientists
Charles and Adele are computer scientists
Charles and Grace are computer scientists
Charles and Ada are computer scientists

```
\NewList{L-Man}
\NewList{L-Woman}
\SetList{L-Man}{Alfred,Charles,Gregory}
\SetList{L-Woman}{Ada,Grace,Adele}
\ForEachRandomItem{L-Man}{Man}{%
  \ForEachRandomItem{L-Woman}{Woman}{%
    \Man{} and \Woman{} are computer
    scientists\par
  }
}
```

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:

un French, ein German, and one English.

```
\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:

You say "un" in France.
You say "ein" in Germany.
You say "one" in England.

```
\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
}
```

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 41) shows 100 lines of three numbers separated by comma. When this file is read, the data base 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?

```
\NewList{Pyth}
\ReadFileList{Pyth}{pythagoras.dat}
\GetItem{Pyth}{10}{triple}
\ReadFieldItem{\triple}{0}{triplea}
\ReadFieldItem{\triple}{1}{tripleb}
Do you know that
$\sqrt{\triplea^2+\tripleb^2}$ is an
integer?
```

To those who check the triple page 41, 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 44) where the first line is obviously a title line. You have just to extract this or these lines to obtain a "classical" database.

Your child Wall Larry has B this term. This is good.

```
\NewList{Class}
\ReadFileList{Class}{pupils.dat}
\ExtractFirstItem{Class}{NULL}
\GetRandomItem{Class}{pupil}
\ReadFieldItem{\pupil}{0}{Name}
\ReadFieldItem{\pupil}{1}{FName}
\ReadFieldItem{\pupil}{2}{Note}
Your child \Name{} \FName{} has
\Note{} this term. This is
\if A\Note very \fi
\if C\Note not \fi
good.
```

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 45) has "|" as field separator. Therefore, the calling syntax becomes:

```
\NewList{Comets}
\ReadFileList[|"]{Comets}{comets.dat}
\ExtractFirstItem{Comets}{NULL}
\ExtractFirstItem{Comets}{NULL}
\GetRandomItem{Comets}{comet}
\ReadFieldItem{\comet}{1}{Name}
\ReadFieldItem{\comet}{2}{Discover}
\ReadFieldItem{\comet}{3}{Year}
\ReadFieldItem{\comet}{4}{Period}
The comet \Name{} was discovered by
\Discover{} in \Year{}.
\unless\ifx\Period\empty
  Its period is \Period{} years.
\fi
```

The comet 2P/Encke was discovered by Encke in 1786. Its period is 3.30 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 $n-1$ and store it in `\<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 `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 `pgffor` package or `\multido` from `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 `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.

```
\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}
  \end{center}
  \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
  \newpage
  \begin{center}
```

```
41      \fbox{\huge\bfseries Answer to the test for \Name{} \FName}
42    \end{center}
43    \textbf{Exercise} \par
44    \if A\Note
45      Use Pythagorean theorem. We have:
46      \[\text{diag}^2=\text{side1}^2+\text{side2}^2.\]
47      Here:
48      \[\Triplec^2=\Triplea^2+\text{side2}^2\]
49      and then
50      \[\text{side2}=\sqrt{\Triplec^2-\Triplea^2} = \Tripleb.\]
51    \else
52      Use Pythagorean theorem. We have:
53      \[\text{diag}^2=\text{side1}^2+\text{side2}^2.\]
54      Here:
55      \[\text{diag}^2=\Triplea^2+\Tripleb^2\]
56      and then
57      \[\text{diag}=\sqrt{\Triplea^2+\Tripleb^2} = \Triplec.\]
58    \fi
59    \newpage
60  }
61  \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 data bases 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 nth 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 LaTeX 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:

- cynical
- magical
- logical
- clinical
- practical

```
\LaTeX{} is:
\RandomItemizeList
  {magical}
  {logical}
  {practical}
  {clinical}
  {cynical}
```

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

LaTeX is:

1. clinical
2. practical
3. logical
4. cynical
5. magical

```
\LaTeX{} is:
\RandomEnumerateList
  {magical}
  {logical}
  {practical}
  {clinical}
  {cynical}
```

5 Package `randomlist` code

5.1 LaTeX's wrapper

5.1.1 Introduction

We start with release number and date.

```
\NeedsTeXFormat{LaTeX2e}[1995/06/01]
\ProvidesPackage{randomlist}
        [2016/07/13 v1.2 Package for random list (JCC, CT)]
```

LaTeX's wrapper has the possibility to use option. There is only one option: the seed one. It requires the `(x)keyval` package.

```
\RequirePackage{xkeyval}
\DeclareOptionX{seed}{\gdef\RL@seed{#1}}
\ExecuteOptions{seed=0}
\ProcessOptionsX
```

We can now call the real randomlist code!

```
\input{randomlist}
```

5.1.2 LaTeX lists

Obviously, LaTeX lists are useful only with LaTeX!

`\RandomItemizeList` Build an itemize list with random placement of items.

```
\NewList{*RandomList*}
\def\RandomItemizeList{%
  \def\RL@Type{itemize}%
  \ClearList{*RandomList*}%
  \@ifnextchar\bgroup{\@randomlist}{\@@randomlist}%
}
\long\def\@randomlist#1{%
  \InsertRandomItem{*RandomList*}{#1}%
  \@ifnextchar\bgroup{\@randomlist}{\@@randomlist}%
}
\def\@@randomlist{%
  \long\edef\RL@body{\noexpand\begin{\RL@Type}}%
  \RLfor \RL@var = 0 to \RL@lenof{*RandomList*}-1 \do{%
    \long\edef\RL@body{%
      \unexpanded\expandafter{\RL@body}%
      \unexpanded\expandafter{%
        \expandafter\item \csname *RandomList*-\RL@var\endcsname
      }%
    }%
  }%
  \long\edef\RL@body{\unexpanded\expandafter{\RL@body}\noexpand\end{\RL@Type}}%
  \RL@body
}
```

`andomEnumerateList` Like `randomitemize` but for enumerate list.

```
\newcommand*\RandomEnumerateList{%
  \def\RL@Type{enumerate}
  \ClearList{*RandomList*}%
  \@ifnextchar\bgroup{\@randomlist}{\@@randomlist}
}
```

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.

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

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

```
\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
```

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

```
\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. \space See User's Manual for further information.^^J
      \@spaces\@spaces\@spaces\@spaces
      Type \space H <return> \space for immediate help.}%
    \errmessage{#2}%
    \endgroup
  }
```

```
\fi
```

We check if we work with an engine which contain at least eTEX.

```
\ifx\numexpr\@undefined
    \begingroup
    \newlinechar`\^^J
    \errhelp{Run under etex, pdftex, xetex, luatex, ... but not under
      tex}%
    \typeout{%
      randomlist error. \space See User's Manual for further information.^^J
      \@spaces\@spaces\@spaces\@spaces
      Type \space H <return> \space for immediate help.}%
    \errmessage{You can't use randomlist under tex without etex extension.}%
    \endgroup
\fi
```

`\@gobble` Redefine `\@gobble` if needed.

```
\ifx\@gobble\@undefined
  \long\def\@gobble#1{}
\fi
```

`\RL@addtomacro` We needs to add some code to some macros sometimes.

```
\def\RL@addtomacro#1#2{\expandafter\def\expandafter#1\expandafter{#1#2}}
```

`\RL@ifempty` Test if something is empty. Execute code according to the answer.

```
\def\RL@ifempty#1{%
  \ifcat\relax\detokenize{#1}\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

PDFLATEX, and XETEXknow the primitives `\pdfsetrandomseed` and `\pdfuniformedeviate` but luaTEX didn't know those primitives. Then we have to process "by hand"!

`\RLsetrandomseed` We define the macro which give the initial seed. If the argument is zero, the value is a mixture of actual time, day, month and year. If the argument is nonzero, we process a new randomseed.

The actual random number is stored in `\RL@random`.

```
\newcount\RL@random
\newcount\RL@random@a
\newcount\RL@random@b
\def\RLsetrandomseed#1{%
  \ifnum#1=0
    \RL@random \numexpr \time + \year * \month * \day \relax
  \else
    \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.

```
\ifx\RL@seed\@undefined
  \RLsetrandomseed{0}
```

```
\else
  \RLsetrandomseed{\RL@seed}
\fi
```

`\RL@nextrand` Process the next random number using Linear Congruentiel Generator with Shrage's metthod.

```
\def\RL@nextrand{%
```

Use the LCG with :

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

For that we take:

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

Then:

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

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

```
  \RL@random@a=\RL@random
  \divide\RL@random@a 127773
  \RL@random@b=\RL@random@a
  \multiply\RL@random@a -2836
  \multiply\RL@random@b -127773
  \advance\RL@random\RL@random@b
  \multiply\RL@random 16807
  \advance\RL@random\RL@random@a
```

If random number is negative add $2^{31} - 1$.

```
  \ifnum\RL@random<0
    \advance\RL@random 2147483647
  \fi
}
```

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

```
\def\RLuniformdeviate#1#2{%
```

Compute the next random number `\RL@random`.

```
  \RL@nextrand
```

Compute `\RL@random` (mod #1).

```
  \RL@random@a=\RL@random
  \RL@random@b=\RL@random
  \divide\RL@random@a \numexpr#1\relax
  \RL@random@b \numexpr\RL@random@b - \RL@random@a * (#1)\relax
  \expandafter\edef\csname #2\endcsname{\number\RL@random@b}%
}%
```

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 separed 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
      \expandafter\@secondoftwo
    \else
      \expandafter\@firstoftwo
    \fi
  }%
  \@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{%
  \if@EmptyListFound
  \PackageError{randomlist}%
```

```
                {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.

```
\long\def\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.
\long version of \@nameedef.

```
\long\def\RL@nameledef#1{%
  \long\expandafter\edef\csname #1\endcsname
}
```

\RL@namelgdef \long version of \@namegdef.

```
\long\def\RL@namelgdef#1{%
  \long\expandafter\gdef\csname #1\endcsname
}
```

\RL@namelxdef \long version of \@namexdef.

```
\long\def\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

```
\long\def\RL@doafterfi#1\fi{\fi#1}
\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@ii@\string#1\expandafter\endcsname\expandafter
```

Second argument (max).

```
  {\number\numexpr#3\expandafter}%
```

Third and fourth arguments since \RL@sgncomp is "<+" or ">-".

```
  \RL@sgncomp
```

fifth argument (variable name).

```
  #1%
}
```

Auxiliary macro:

- #1 recursive macro name (like \RLfor@ii@<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%
    }%
    \fi
  }%
```

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 @ListOfList. A macro is also created for accessing data.

```
\def\NewList#1{%
  \@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 erase 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}%
```

```
      }%
    \fi
  }%
  {\@NoListError{#2}}%
 }%
 {\@NoListError{#1}}%
}
```

\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}}%
}
```

\ShowList Macro for debugging purpose. First we declare some scratch count registers.

```
\newcount\RL@counti
\newcount\RL@countii
\newcount\RL@countiii
\def\ShowList#1{%
```

We show a list only if this list exists!

```
 \@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\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
```

```
        \RLfor\RL@iter=0 to \RL@lenof{#1}-1 \do {%
          #1[\RL@iter] = \expandafter\RL@meaning\csname
          #1-\RL@iter\endcsname
          \par
        }%
      \fi
```

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}}%
}
```

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 shitf is negative, it is a left shift.

```
\def\ShiftList#1#2#3{%
  \@ifIsList{#1}%
  {%
```

No action if shift is zero!

```
    \unless\ifnum#3=0
```

If <start> is negative, raise an error.

```
      \ifnum\numexpr#2<0
        \PackageError{randomlist}%
                     {Negative index number}%
                     {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
                       (<=\RL@lenof{#1})}%
                     {Index must be equal or smaller than length of
                       the list}%
      \else
```

Here we have $0 \leq$ <start> $\leq$ len(<list>).

If <shift> is positive, we process a right shifting: it's alway possible.

```
      \ifnum\numexpr#3>0
        \RLfor\RL@iter = \RL@lenof{#1} to #2 \do{%
          \RL@let{#1-\number\numexpr\RL@iter+#3}{#1-\RL@iter}%
        }%
```

Empty the items out of shift part.

```
        \RLfor\RL@iter = #2 to #2 + #3 - 1 \do{%
          \RL@nameldef{#1-\RL@iter}{}%
        }%
      \else
        \ifnum-#3>\numexpr#2\relax
```

If the negative shifting is to big for index #2 then raise an error.

```
          \PackageError{randomlist}%
                       {Negative shift to big}%
                       {When negative, shift must not be greater than index}%
        \else
```

Elsewhere, process the left shifting.

```
          \RLfor\RL@iter=#2 to \RL@lenof{#1} \do{%
            \RL@let{#1-\number\numexpr\RL@iter+#3}{#1-\RL@iter}%
          }%
        \fi
      \fi
```

Set the list length for both positive and negative shifting.

```
      \RL@nameledef{#1-len}{\number\numexpr\RL@lenof{#1} + #3}%
    \fi\fi\fi
  }%
  {\@NoListError{#1}}%
}
```

`\InsertLastItem` Add an element #2 at the end of the list #1.

```
\long\def\InsertLastItem#1#2{%
  \@ifIsList{#1}
  {%
    \RL@nameldef{#1-\RL@lenof{#1}}{#2}%
    \RL@nameledef{#1-len}{\number\numexpr\RL@lenof{#1}+1}%
  }
  {\@NoListError{#1}}%
}
```

`\InsertFirstItem` Add an element #2 at the beginning of the list #1. For that, shift right all the element and then put #3 at L[0].

```
\long\def\InsertFirstItem#1#2{%
  \InsertItem{#1}{0}{#2}%
}
```

`\InsertItem` Add an element #3 at the position #2 of the list #1. For that, pass from L[0] to L[#2-1] then shift right from L[#2] to L[len] and finally put #3 at L[#2]. To do this, we must have #2 $\geq$ L-len.

```
\long\def\InsertItem#1#2#3{%
```

```
  \@ifIsList{#1}%
  {%
    \ShiftList{#1}{#2}{1}%
    \RL@nameldef{#1-#2}{#3}%
  }%
  {\@NoListError{#1}}%
}
```

\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}%
  }%
```

```
  {\@NoListError{#1}}%
}
```

\ExtractList `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.

```
\def\ExtractList#1#2#3#4{%
```

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

```
  \@ifIsList{#1}{%
    \@ifIsList{#4}{%
      \ifnum#2<\RL@lenof{#1}%
        \ifnum#3<\RL@lenof{#1}%
          \ifnum#2>#3\relax
```

If start > end we build an empty list.

```
            \RL@nameldef{#4-len}{0}%
          \else
```

If start ≤ end we build a real extracted list. We have to be careful because `\ExtractItem` uses the loop variable `\RL@iter`. Then we use another loop variable.

```
            \RLfor\RL@iterextract=0 to #3 - #2 \do{%
              \RL@let{#4-\RL@iterextract}{#1-#2}%
              \ExtractItem{#1}{#2}{RL@temp}%
            }%
            \RL@nameledef{#4-len}{\number\numexpr #3 - #2 + 1}%
          \fi
        \else
          \@OutOfRangeError{#1}{#3}%
        \fi
      \else
        \@OutOfRangeError{#1}{#2}%
      \fi
    }%
    {\@NoListError{#4}}%
  }%
  {\@NoListError{#1}}%
}
```

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

```
\def\GetFirstItem#1#2{%
  \GetItem{#1}{0}{#2}%
}
```

\GetLastItem Get the last element of list #1 and store it in #2.

```
\def\GetLastItem#1#2{%
  \GetItem{#1}{\number\numexpr\RL@lenof{#1}-1}{#2}%
}
```

\GetItem Get the element of rank #2 of list #1 and store it in #3.

```
\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
  }
  {\@NoListError{#1}}%
}
```

\GetRandomItem Get element in a random position of list #1 and store it in #2.

```
\def\GetRandomItem#1#2{%
  \@ifIsListNotEmpty{#1}%
  {%
    \RLuniformdeviate{\RL@lenof{#1}}{RL@temp}%
    \GetItem{#1}{\RL@temp}{#2}%
  }%
  {\@NoListError{#1}}%
}
```

\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.

```
\def\GetList#1#2#3#4{%
```

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

```
  \@ifIsList{#1}{%
    \@ifIsList{#4}{%
      \ifnum#2<\RL@lenof{#1}%
        \ifnum#3<\RL@lenof{#1}%
          \ifnum#2>#3\relax
```

If start > end we build an empty list.

```
            \RL@nameldef{#4-len}{0}%
          \else
```

If start ≤ end we build a real extracted list.

```
            \RLfor\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{#1}{#2}%
      \fi
    }%
    {\@NoListError{#4}}%
  }%
  {\@NoListError{#1}}%
}
```

\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 \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}}%
}
\long\def\RL@setlist#1,{%
```

```
  \def\RL@arg{#1}%
  \unless\ifx\RL@arg\@nnil
    \InsertLastItem{\RL@name}{#1}%
    \expandafter\RL@setlist
  \fi
}
```

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 recusive 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 string is the double quote.

```
\def\RL@SetDelimiters#1#2#3\@nil{%
```

- argument #1 is the field separator;
- argument #2 is the string delimiter;
- argument #3 is the remainder to ignore.

```
  \def\RL@markstrings##1{%
    \let\RL@accu\empty
    \expandafter\RL@markstrings@i##1#2\@nil#2%
    \let##1=\RL@accu
  }%
  \def\RL@markstrings@i##1#2##2#2{%
    \RL@addtomacro\RL@accu{##1}%
    \def\RL@current{##2}%
    \unless\ifx\@nnil\RL@current
      \RL@addtomacro\RL@accu{\RL@string{##2}}%
      \expandafter\RL@markstrings@i
    \fi
  }%
  \def\RL@unmarkstrings##1{%
```

```
    \let\RL@accuA\empty
    \expandafter\RL@unmarkstrings@i##1\RL@string\@nil
    \let##1=\RL@accuA
  }%
  \def\RL@unmarkstrings@i##1\RL@string##2{%
    \RL@addtomacro\RL@accuA{##1}%
    \def\RL@current{##2}%
    \unless\ifx\@nnil\RL@current
      \RL@ifempty{##2}%
        {\RL@addtomacro\RL@accuA{#2}}%
        {\RL@addtomacro\RL@accuA{##2}}%
      \expandafter\RL@unmarkstrings@i
    \fi
  }%
  \def\RL@parsefields##1{%
    \let\RL@accu\empty
    \expandafter\RL@parsefields@i##1#1\@nil#1%
    \let##1=\RL@accu
  }%
  \def\RL@parsefields@i##1#1{%
    \def\RL@current{##1}%
    \unless\ifx\@nnil\RL@current
      \RL@unmarkstrings\RL@current
      \RL@removefirstspaces\RL@current
      \RL@removelastspaces \RL@current
      \expandafter\RL@addtomacro\expandafter\RL@accu\expandafter
        {\expandafter{\RL@current}}%
      \expandafter\RL@parsefields@i
    \fi
  }%
}
```

The macro \ReadFileList uses a handle for the reading file. it needs also a macro to detect \par

```
\newread\RL@hdle
\def\@ppar{\par}
```

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

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

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

```
\def\@ReadFileList[#1]#2#3{%
  \openin \RL@hdle = #3
  \ifeof\RL@hdle
    \PackageError{randomlist}%
      {File #3 doesn't exist}%
      {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.

```
    \RL@ifempty{#1}%
      {%
        \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\@ppar
```

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
  \expandafter#\expandafter1\space#2\@nil{%
    \RL@ifempty{#1}%
}
% Remove all spaces at the start of argument (macro).
\def\RL@removefirstspaces#1{%
  \expandafter\RL@ifspacefirst\expandafter{#1}
    {\expandafter\removefistspace@i#1\@nil#1}
    {}%
}
\expandafter\def\expandafter\removefistspace@i\space#1\@nil#2{%
  \def#2{#1}%
  \RL@removefirstspaces#2%
}
% Store |^^00|'s catcode
\edef\RL@restorecatcodezero{\catcode0=\number\catcode0\relax}
```

then set this catcode to other catcode.

```
% puis le modifie à 12.
\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{\RL@removelastspaces@ii#1^^00}
\def\RL@removelastspaces@ii#1^^00#2\@nil{%
  \RL@ifspacefirst{#2}
    {\RL@removelastspaces@i#1^^00 ^^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
```

Change history

v0.1
General: First release 20
v0.1a
General: Fix bug about braces. 20
v0.1b
General: Add \initrandomlist. 20
v1.0
General: randomlist completly rewritten. 20
v1.1
General: Tons of improvements due to Christian Tellechea. 20
v1.2
General: First public release. 20

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.

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.

```
119,120,169
108,144,180
153,104,185
144,130,194
133,156,205
120,182,218
105,208,233
180,112,212
171,140,221
160,168,232
147,196,245
132,224,260
115,252,277
209,120,241
200,150,250
189,180,261
176,210,274
161,240,289
144,270,306
125,300,325
104,330,346
240,128,272
231,160,281
220,192,292
207,224,305
192,256,320
175,288,337
156,320,356
135,352,377
112,384,400
280,102,298
273,136,305
264,170,314
253,204,325
240,238,338
225,272,353
208,306,370
189,340,389
168,374,410
145,408,433
120,442,458
315,108,333
308,144,340
299,180,349
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

185,672,697
152,714,730
117,756,765
475,132,493
468,176,500

B File `pupils.dat`

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

```
Name,FirstName,Result
Aho,Alfred,A
Babbage,Charles,A
Chaitin,Gregory,B
Dijkstra,Edsger,A
Eckert,John Preper,B
Floyd,Robert,B
G\"odel,Kurt,A
Huffman,David,B
Ichbiah, Jean,A
Joshi,Aravind,C
Knuth,Donald,C
Lovelace,Ada,A
Moore,Gordon,A
Neumann (Von),John,A
Ouserhout,John,B
Pascal,Blaise,A
Ritchie,Dennis,C
Shannon,Claude,C
Thompson,Ken,A
Ullman,Jeffrey,B
Vixie,Paul,B
Wall,Larry,B
Yao, Adrew Chi-Chih,C
Zuse,Konrad,C
```

C File `comets.dat`

This file use lines which aren't data lines and weird separator.

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