The mcexam package

Jorre Vannieuwenhuyze (jorre_v@zoho.com)

December 26, 2017

This package automatically randomly permutes the order of questions and answer options in different versions of a multiple choice exam/test. Next to the exam versions themselves, the package also allows printing a concept version of the exam, a key table with the correct answers or points, and a document with solutions and explanation per exam version. The package also allows writing an R code which processes the results of the exam and calculates the grades.

This package was developed for large-scale randomized multiple choice exams at my department. The functionalities of this package may overlap with the esami package and the AMC program but there were some special requirements at my department which are not included in these packages. In particular the mcexam provides the following features:

- The possible answer permutation patterns are completely flexible and can entirely be specified by the user.
- Typesetting the answer options is completely flexible with the `mcanswers` environment.
- At my university, the students have to fill in a standard one-page answer form and unprocessed results from these forms are directly provided to the examiner. The package writes an `R` code which processes and analyses these results.

It was easier to build this new package that fits our purpose than to unravel and change the existing ones. Nevertheless, in case of interest, it is my intention to contact the developers of the other packages in order to integrate the functionalities into one package.

Specifying questions and answers with the mcexam are very straightforward and similar to ordinary list environments. Imagine that you want to develop a multiple choice exam with three questions, each including three possible answers, which all need to be randomized. With the mcexam package, all you need to do is to include these questions and answers in a list-wise structure within the `mcquestions` environment inside the document body of your favourite LaTeX document-class:

```
\begin{mcquestions}

\question How much is $2+2$?

          \begin{mcanswerslist}
          \answer two
          \answer[correct] four
          \answer five
          \end{mcanswerslist}

\question How much is $5-3$?

          \begin{mcanswerslist}
          \answer 1
          \answer[correct] 2
          \answer 3
          \end{mcanswerslist}

\question How much is $0 \times 2$?
```

```
        \begin{mcanswerslist}[fixlast]
        \answer 1
        \answer 2
        \answer[correct] none of the above
        \end{mcanswerslist}

\end{mcquestions}
```

Depending on the option you specify when loading the package, these questions can be compiled as

- a concept exam which includes all information about the randomizations and solutions;
- each version of the exam itself;
- a key table with the correct answers;
- each version of the exam including the solutions and explanations; and
- an analysis version which includes an analysis of the results of the exam after these are processed by R.

The user of the package can further mark answer options by points instead of 'correct/incorrect', group questions, add an introduction, an explanation and notes to each question, define arbitrary permutation patterns to the answers, and much more. As an example of such an option, note the optional argument `fixlast` in the `mcanswerslist` environment in the last question. This option means that the last answer in this question will never change place across different versions of the exam. Of course, there are many more such options and all these options are described below.

An example exam file is given in `mcexam_example.tex`. Fictitious results for this exam are given in the `mcexam_example_dataset.r` which can be processed in R following the instructions below.

1 Loading the package

The package is loaded as usual by the code statement

```
\usepackage[<options>]{mcexam}
```

The mcexam package requires the following packages to be installed: xkeyval, etoolbox, xstring, environ, pgf, enumitem, longtable and newfile.

The following package options can be included as 'key=value'-pairs:

- `output`: The package option `output` defines the output type of the document. There are five possible output types you can use:

- `output=concept`: This option creates a concept version of your exam with all the questions typeset in the same order as they appear in the script file. For each question, the correct answers or the answer points are given, a table is created with the answer permutations for each version of the exam, the explanation of the solution is shown and the additional notes are shown. `concept` is the default output type.
- `output=exam`: This option creates a version of the exam that can be printed and given to the students during the real exam. It doesn't mark the correct answer and hides explanation boxes and question notes. The version of the exam which needs to be made, can be specified by the `version` option (see below).
- `output=key`: This option creates a key table with the labels of the correct answers for each question. This key table can be published after the exam so that the students can check there answers, without seeing the actual questions and answers themselves.
- `output=answers`: This option creates a document with the questions, the answers and an explanation of the solution. It is meant to be used for feedback to the students. As with the `exam` output, these answer documents have to be made for each version separately because they maintain the randomization order of the questions and answers. The version of the exam which needs to be made, can be specified by the `version` option (see below).
- `output=analysis`: This option provides an analysis of the results, it provides the frequencies of students giving the answer options and it shows statistics like the average points, the p-value, the corrected p-value, the item rest correlation, or Cronbach's alpha. Before using this option, the results need to be processed by R first (see section 4).

• `numberofversions=`n: This option specifies the total number of versions n to be made. By default the number of versions is set to 4. Make sure that the version number (the next package option) does not exceed this number. If the version number does exceed the number of versions, an error will be thrown.

• `version=`v: This option defines the version v which needs to be printed when `output=exam` or `output=answers`. This option takes a number as argument between 1 and the total number of versions, e.g. `version=1`, `version=2`, etc. The default value of this option is `version=1`.

The package also defines the command `\mctheversion` which can be used to print the version number on the document in the correct format (see section 5.2). In the `exam` or `answers` output style, it prints "Version v". In the `concept` output style, it prints "Concept version". In the `analysis` output style, it prints “Analysis”.

- `seed=`*s*: This option sets the seed *s* for the random permutations, which is an arbitrarily chosen positive integer. The seed is not set by default, which means that a different randomization will be obtained each time the document is processed. For that reason, it is advised to always specify the seed, in order to obtain consistent output across different runs.
- `randomizequestions=`*true/false*: This option defines whether the question order should be randomized (true) or not (false). This option is set to true by default.
- `randomizeanswers=`*true/false*: This option defines whether the answer order should be randomized (true) or not (false). This option is set to true by default. If you set both the `randomizequestions` and the `randomizeanswers` options to false, a warning will be thrown. After all, if you don’t want to randomize anything, you don’t really need this package.
- `writeRfile=`*true/false*: This option defines whether an `R`-script should be written including macro’s to process and analyse the students’ answers. If it is set to true, an `R`-script will be written. By default it is set to false. Note that, if this option is set to true, all questions should contain answers within the `mcanswers` or `mcanswerslist` environments, otherwise the program will throw errors.

The package also defines the `\mcexamoptions{key=value,...}` command which can also be used to set or change the package options after the package has been loaded.

2 Setting the questions

2.1 The questions

The most important parts of the exam are, of course, the questions. Within the body of your script file, you place the questions within the `mcquestions`-environment which works similar to list-environments. Each question is preceded by a `\question` command similar to the `\item` command :

```
\begin{mcquestions}
\question This is the first question.
\question This is the second question.
\end{mcquestions}
```

Sometimes, some questions belong together and these questions may not be split when the question order is randomized. To avoid this undesirable behavior, you can include `follow` as an optional argument to the `\question` command. For example,

```
\begin{mcquestions}
\question This is the first question.
\question[follow] This is the second question.
\question The third question.
\end{mcquestions}
```

means that the second question will always directly follow the first question even though they may appear as questions two and three instead of one and two in some versions of the exam.

2.2 The answers

With the `mcexam` package, answers can be set in two ways. The first and probably the most common way to set the answers is a list structure. This can be achieved by the `mcanswerslist` environment which works similar to regular list-environments. Each answer is preceded by an `\answer` command (like `\item` in an `itemize` list). You can include as many answers as you want. For example

```
\question What is the color of the sky?
  \begin{mcanswerslist}
  \answer blue
  \answer green
  \answer red
  \answer yellow
  \answer none of the above
  \end{mcanswerslist}
```

will typeset

1. What is the color of the sky?

    a) blue
    b) green
    c) red
    d) yellow
    e) none of the above

or any of its permutations.

Sometimes, however, such list structures are not what you want. In that case you can use the `mcanswers` environment. Within this environment you need to use the `\answer` command and the `\answernum` command. The `\answer{`*n*`}{`*answer*`}` sets the *answer* itself which is assigned number *n*. It is mandatory to use all numbers between 1 and the total number of answers. For example, if you need three answers you need to include `\answer{1}{...}`,

`\answer{2}{...}`, and `\answer{3}{...}`, otherwise the package will throw errors. The order in which the `\answer` commands appear, however, is not important. You can start with `\answer{3}{...}` as long as you also include `\answer{1}{...}` and `\answer{2}{...}`.

The `\answernum{`*n*`}` simply typesets the answer number n in the correct format (see section 5.2). This command is not mandatory to use, but it is highly recomended to use it because it will automatically typeset the answer number in the correct format.

As an example, the code

```
\question Which is the letter alpha?
          \begin{mcanswers}
          \begin{tabular}{cccc}
          \answer{1}{\Huge$\alpha$} &
          \answer{2}{\Huge$\beta$}  &
          \answer{3}{\Huge$\gamma$} &
          \answer{4}{\Huge$\delta$} \\[0.1\baselineskip]
          \answernum{1} &
          \answernum{2} &
          \answernum{3} &
          \answernum{4} \\
          \end{tabular}
          \end{mcanswers}
```

will produce

1. Which is the letter alpha?

α	β	γ	δ
a)	b)	c)	d)

or something similar with the Greek letters shuffled around.

2.2.1 Permuting the answers

By default, all answers are randomly shuffled within the `mcanswerslist` and `mcanswers` environments. However, this is not always what you want. For that reason, both environments have one optional argument which specifies how answers should be permuted. This option can be one of the following:

- `permuteall`: With this option, all answers are permuted in the different versions of the exam. This is the default option and doesn't need to be given explicitly.
- `ordinal`: This option permutes the answers in order. That is, the answers are given on the exam either in the order or the reversed order as they are

given in the script file.

- `fixlast`: This option permutes all answers except the last given answer. This is handy if the last answer is something like, for example, “None of the above”.
- `permutenone`: This option doesn’t permute the answers and sorts the answers in each version in the same order as they are given in the script file.
- *User specific*: If none of the above options satisfies your needs, you can still enter the possible permutations manually. You achieve this by making a comma separated list of all allowed permutations where each permutation itself is a comma separated list of all answer numbers within two curly braces. For example, if your question contains three answers you can give the option `[{1,2,3},{2,3,1}]`. With this option, the package will either put the answers in the order ‘answer 1, answer 2, answer 3’ or in the order ‘answer 2, answer 3, answer 1’. The package will throw errors if only one permutation is given, if a permutation does not contain all answer numbers, if a permutations contains answer numbers more than once, or if a permutation contains invalid answer numbers. The package will not throw errors if a permutation is given more than once. For example, `[{1,2,3},{1,2,3},{2,3,1}]` will not give errors, it just means that the order ‘1, 2, 3’ is twice more likely to appear than the order ‘2, 3, 1’.

Putting all together, in your script file you can write something like this:

```
\question What is the color of the sky?
  \begin{mcanswerslist}[fixlast]
  \answer blue
  \answer green
  \answer red
  \answer yellow
  \answer none of the above
  \end{mcanswerslist}

\question Which statement is correct?
          \begin{mcanswerslist}[{1,2,3,4},{2,1,4,3}
                               ,{3,4,1,2},{4,3,2,1}]
          \answer The moon is a planet.
          \answer The moon is a star.
          \answer The sun is a planet.
          \answer The sun is a star.
          \end{mcanswerslist}
```

2.2.2 Points

The correct answer (or answers) can be marked by an optional argument of the `\answer` command in both the `mcanswerslist` and `mcanswers` environments, that is

```
\answer[<mark>]
```

in the `mcanswerslist` environment and

```
\answer[<mark>]{<n>}{<answer>}
```

in the `mcanswers` environment. The `<mark>`-option can be either of two modes.

- `correct` or empty (=incorrect, default): With these options, answers are either correct or incorrect. Correct answers are given one point, incorrect answers are given zero points. Note that more than one answer per question can be marked as correct.
- A decimal number: Altenatively, the marks can be decimal numbers which are the points given to each answer. The numbers/points can be negative in case you want to use guess corrections.

Depending on the mode (correct/incorrect or points) the package may give slightly different output in some output types. As soon as the package encounters one decimal as a mark, it switches to points mode. In this mode, `correct` or empty options will not give errors but will be treated as one point and zero points respectively. In case no decimal marks are given and only '`correct`' marks are encountered, the package will give output in correct/incorrect-mode.

As an example, the following questions are valid questions:

```
\question Which is the letter alpha?
          \begin{mcanswers}
          \begin{tabular}{cccc}
          \answer[correct]{1}{\Huge$\alpha$} &
          \answer{2}{\Huge$\beta$}  &
          \answer{3}{\Huge$\gamma$} &
          \answer{4}{\Huge$\delta$} \\[0.1\baselineskip]
          \answernum{1}&
          \answernum{2}&
          \answernum{3}&
          \answernum{4}\\
          \end{tabular}
          \end{mcanswers}
```

and

```
\question What is the color of the sky?
  \begin{mcanswerslist}[fixlast]
  \answer[4] blue
```

```
\answer[-2] green
\answer[1.5] red
\answer[1] yellow
\answer none of the above
\end{mcanswerslist}
```

2.3 Instructions, explanation, and notes

Additional to the answers, the question environment can also include an `mcquestioninstruction`, an `mcexplanation` and an `mcnotes` environment. The `mcquestioninstruction` environment is used for general instructions about the next question or group of questions (using the `follow` option of the `\question` command). It is printed in the exams themselves. The `mcexplanation` enviromnent should be used to store an explanation of the question and the right answer. It is printed in the `concept` and the `answers` versions. The `mcnotes` enviromnent, in contrast, can be used to store additional notes about the question. It is only printed in the `concept` version. An example of their use is as follows:

```
\begin{mcquestioninstruction}
This is a question about the sky.
\end{mcquestioninstruction}

\question What is the color of the sky?

          \begin{mcanswerslist}[fixlast]
          \answer[4] blue
          \answer[-2] green
          \answer[1.5] red
          \answer[1] yellow
          \answer none of the above
          \end{mcanswerslist}

          \begin{mcexplanation}
          If you look up to the sky and there are no clouds,
          you'll see it is blue.
          \end{mcexplanation}

          \begin{mcnotes}
          This question had a large proportion of good answers
          last year.
          \end{mcnotes}
```

Note that the `mcquestioninstruction` comes before the `\question` command while the `mcexplanation` and `mcnotes` environments come after the `\question` command. As a result, an `mcexplanation` or an `mcnotes` environment before the first `\question` will give an error, while an `mcquestioninstruction` after the last `\question` will also give an error.

3 Selective output

Sometimes, you may want to print some text or use some commands in, for example, only one output type or in only one version of the exam. In that case, you should use the `\mcifoutput` command:

```
\mcifoutput[<versions>]{<output-types>}{<text>}
```

This command prints *<text>* in all *<output-types>* given as a comma-separated list. In the `exam` and `answers` output types you can also optionally specify the *<versions>* as a comma-separated list. As an example, the code

```
\mcifoutput{concept,exam}{name:}
```

will print "name:" on the exams and in the concept version but not on the answers document, the key table, or the analysis document. The command

```
\mcifoutput[1,3]{exam}{\newpage}
```

will start a new page on the exams version 1 and 3 but not in the other versions. If the *<versions>* option is empty, the command will print *<text>* on all versions of the `exam` documents. Note that the *<versions>* option is ignored for `concept`, `key` and `analysis` output. This means that code like

```
\mcifoutput[wrong]{exam,concept}{blah}
```

will print "blah" on the concept version because `wrong` is ignored. However, this code will print nothing on the exams because the *<versions>* option is not empty (although wrong). Note that the optional *<versions>* parameter can not be used in the preamble because the version is only set in stone at the `\begin{document}`.

4 Analysis of results

4.1 Processing with R

Typically, multiple choice exams are completed on standard forms which are scanned automatically. The examiner then receives a data file listing all students with their answers. Using this file to grade the students is not an easy task because it firstly requires to map the given permuted answer number to the initial answer number and secondly requires checking whether this answer is right or wrong. The mcexam class allows to do this semi-automatically.

If the `writeRfile` package option is enabled, compiling the exam document in any of the output types automatically generates an `R` file which can be used to process data files with the answers of the students. The `R` code will be written to the file `\jobname.r`. In case more than one `mcquestions` environments are present in the document, the second environment will write its `R`-code to the file `\jobname-B.r`, the third will be written to the file `\jobname-C.r`, etc. The `-B`, `-C`, ... extensions can be overruled by the optional argument of the `mcquestions` environment. For example, if you write

```
\begin{mcquestions}[myfile]
...
\end{mcquestions}
```

the `R`-code of this environment will be written to the file `\jobname-myfile.r`.

In the `R` file, the function `processanswers` is defined which processes the answers. This function takes four arguments:

- `ID`: A vector with the ID numbers of the students.
- `version`: A numeric vector with the version number of the exam for each student in the same order as `ID`.
- `answers`: A numeric data frame with the given answers of the students. The columns refer to the questions, the rows to the students in the same order as `ID`.

  Watch out, if your initial data is not numeric, you must first transform the data to numeric answers. For example, if you have a data frame `X` with answers in capital letters (A-Z) format, you need first to map these letters to the numbers 1-26 by, for example, applying the `match` function to the data frame:

  ```
  apply(answers,2,match,LETTERS).
  ```
- `path`: The path where the function writes information for the analysis output type (see next subsection). By default, the path is the working directory (`getwd()`).

The `processanswers` function creates a new data frame with the students' ID's, the versions, and two variables for each question. The variables `originalQuestion.x`, where `x` refers to the question number, store the answers of the students in the initial enlisting of the script file (before permutations). The variables `pointsQuestion.x` give the points of a student to initial question `x` (in correct/incorrect mode, it is 1 if a student gave a correct answer to question `x` and 0 if the student gave a wrong answer). Additionally, there is also the `total` variable which stores the sum of the points (i.e. the sum of the `pointsQuestion.x` variables).

The `processanswers` function also creates the file `\jobname.ana` which stores information for the analysis output type.

4.2 Compiling an analysis document

After processing the answers in R, you can compile an analysis document with the `output=analysis` package option. This analysis document includes the percentages of students giving each answer option, the (corrected) p-values, the item-rest correlations, and Cronbach's alpha of the questions. First, confirm that the `\jobname.ana` file, which was created by R, is in the same folder as your `.tex` file. In case of multiple `mcquestions` environments, the program will automatically take the extension into account, also when this is specified by the optional argument of the `mcquestions` environment.

5 Lay-out and additional useful commands

5.1 Tweaking the output types

At `\begin{document}`, a set of booleans are defined which depend on the output type and which control how the output is created. These booleans can be changed if you don't like the output of a certain output type. In order to change these booleans, use the

```
\mcsetupConcept{<keys>}
\mcsetupExam{<keys>}
\mcsetupKey{<keys>}
\mcsetupAnswers{<keys>}
\mcsetupAnalysis{<keys>}
```

commands, which take a `key=`*`value`*-list as argument. The keys which can be used are the following:

- `showPerVersion=`*true/false*: If true, only one version is send to the output depending on the `version` option of the package. If false, a general document is made which summarizes all versions and which may show permutation tables for the questions and answers (if these are permuted at least).
- `showQuestionPermutationtable=`*true/false*: Show a permutation table of the questions at the start of the `mcquestions` environment?
- `showQuestionsAnalysis=`*true/false*: Show a the analysis of the questions at the start of the `mcquestions` environment? (Requires an `.ana` file).
- `showQuestionList=`*true/false*: Show the questions?
- `showCorrectAnswers=`*true/false*: Show the correct answers in correct/incorrect-mode?
- `showAnswerPoints=`*true/false*: Show the answer points in points-mode?

- `showExplanation=`*true/false*: Show the explanation of the correct answer?
- `showAnswerPermutationTable=`*true/false*: Show the premutation table of the answers for each question?
- `showAnswersAnalysis=`*true/false*: Show a the analysis of the answers for each question? (Requires an `.ana` file).
- `showNotes=`*true/false*: Show the notes for each question?
- `showKeytable=`*true/false*: Show the key table? Note that the key table always summarizes all versions. It make no sense to set `showKeytable=true` if `showPerVersion=true`.

As an example, if you don't like the standard way of making a document for each version separately with the answers, you can add the following command to the preamble:

```
\mcsetupAnswers{showPerVersion=false
               ,showQuestionPermutationtable=true
               ,showAnswerPermutationTable=true
               }
```

5.2 Changing the appearance of the counters

By default, the number format of the version numbers is `\Roman`, of the questions numbers is `\arabic`, and of the answers numbers is `\alph`. These formats can be changed by redefining the `\mcversionlabelfmt`, the `\mcquestionlabelfmt` and the `\mcanswerlabelfmt` commands with `\renewcommand`. Currently, these commands are defined as

```
\newcommand\mcversionlabelfmt[1]{\Roman{#1}}
\newcommand\mcquestionlabelfmt[1]{\arabic{#1}}
\newcommand\mcanswerlabelfmt[1]{(\alph{#1})}
```

5.3 Changing the appearance of the list structures

The questions, the answers in the `mcanswerslist` environment and the question information in the `concept`, `answers` and `analysis` output styles are put in newly defined lists using the `enumitem` package. The lists are `setmcquestions`, `setmcanswerslist` and `setmcquestioninfo`. You can change the appearance of these lists using the `\setlist` command from the `enumitem` package. For more information, refer to the `enumitem` package documentation. Currently, the lists are defined as

```
\setlist[setmcquestions]{label=\mcquestionlabelfmt{*}.
                        ,ref=\mcquestionlabelfmt{*}
```

```
                          ,itemsep=2\baselineskip
                          ,topsep=2\baselineskip          }
\setlist[setmcanswerslist]{label=\mcanswerlabelfmt{*}
                             ,noitemsep}
\setlist[setmcquestioninfo]{before=\footnotesize\sffamily}
```

5.4 Changing the appearance of different document elements

The package defines several environments which typeset particular content. The `setmcquestioninstruction` environment typesets the question instructions, the `setmcquestion` typesets the question and its answer options, the `setmcanswers` environment typesets the content of the `mcanswers` environment, the `setmcquestionpermutationtable` typesets the permutation table of the questions, the `setmcanswerpermutationtable` typesets the permutation table of the answers, the `setmcquestionanalysistable` and `setmcansweranalysistable` environments typeset the analysis tables in the `analysis` output, and the `setmckeytable` typesets the key table in the `key` output. All these environments can be redefined with the `renewenvironment` command. Currently, they are defined as

```
\newenvironment{setmcquestioninstruction}
  {\noindent}
  {}
\newenvironment{setmcquestion}
  {}
  {}
\newenvironment{setmcanswers}
  {\vspace{\baselineskip}}
  {}
\newenvironment{setmcquestionpermutationtable}
  {\begin{center}
   \footnotesize\sffamily
   \setlength{\tabcolsep}{15pt}     }
  {\end{center}}
\newenvironment{setmcanswerpermutationtable}
  {\begin{center}
   \setlength{\tabcolsep}{10pt}     }
  {\end{center}}
\newenvironment{setmcquestionanalysistable}
  {\begin{center}
   \setlength{\tabcolsep}{10pt}
   \footnotesize\sffamily           }
  {\end{center}}
\newenvironment{setmcansweranalysistable}
```

```
  {\begin{center}\setlength{\tabcolsep}{10pt}}
  {\end{center}}
\newenvironment{setmckeytable}
  {\begin{center}\setlength{\tabcolsep}{10pt}}
  {\end{center}}
```

For example, if you want to keep all questions and answer options together on the same page, you can include them in a minipage as follows:

```
\renewenvironment{setmcquestion}
  {\begin{minipage}[t]{\linewidth-\labelwidth}}
  {\end{minipage}\par}
```

provided that the `calc` package has been loaded.

5.5 Babel

The `babel` package can be used in an `mcexam` document. At the moment, a Dutch implemenation is included. For Dutch exams, you include

```
\usepackage[dutch]{babel}
```

in the preamble of your script file.

6 Revision History

6.1 Versions

2017/05/02 v0.3: First published version.

2017/12/26 v0.4: Added `pgffor` package to fix bug.

6.2 Planned modifications

- Add an error message when you only have one question and the option randomizequestions=true. Now, you should change this option explicitly to randomizequestions=false for one question.
- Find an alternative for longtable which is now the default for tables in the package. Longtable doesn't work with twocolumn.
- Add a function to reproduce the questions in another permutation/version within the same document.
- Add the option to continue the question counter over several mcquestions environments.