Reusing content in LaTeX
Wielokrotne wykorzystanie treści w LaTeXu

Marcin Borkowski

Adam Mickiewicz University

2011-04-29

Problem statement

Assume that we want to prepare a problem book for a math class. It would be nice to put exercises and answers (and possibly other things, like hints, solutions, remarks etc.) all in one place, and then instruct LaTeX to typeset a "student's version" (exercises without solutions), "teacher's version" (exercises with solutions), answer sheet (answers only) etc.

Plan of the presentation

We will first discuss possible methods of achieving such an effect, mention their advantages and drawbacks; then we will look at existing packages offering suitable features; finally, we will examine in more detail the `xrcise` package, mentioning a few caveats important for everyone who would like to write suitable macros themselves.

Emphasis

There are at least two packages on CTAN which enable preparing a problem set (and a third one on its way ;-)), but none of them has all features one might need.

Therefore, we will only briefly describe the existing packages, and we will concentrate on the problems one has to face when programming one's own package (with a few examples from my `xrcise` package).

Design questions

Before choosing the method to store the content to be reused, one has to answer a few questions.

- What size are the chunks of saved text going to be?
- What is there going to be in the chunks of text?
- How many chunks are needed?
- How are we going to specify which "version" we want typeset?
- What input format are we going to use?

Chunk size

What size are the chunks of saved text going to be?

(Smaller than paragraphs, single paragraphs, multiple paragraphs?)

We will have to handle spaces and `\par`'s.

Chunk content

What is there going to be in the chunks of text?

(Ordinary text, text with TEX macros, non-ASCII text (using `inputenc`), `verbatim` text?

Some methods don't like some kinds of content...

One or many?

Do we need one chunk at a time or do we want to accumulate a series of them? If we need more than one, will the chunks have names or numbers? If numbers, are they going to be consecutive ones?

When we only need one chunk at a time, it is enough to use one toks/box register, file or macro. In case of a series, we have to deal with accumulating and maybe automatic numbering (possibly with omissions).

Specifying versions

How are we going to specify which "version" we want typeset?

(Package/class option or macro, command line, config file, "driver" files?)

LaTeX doesn't support command line parameters well. If we want to use this method, it's probably best to use a config file and a wrapper script.

A package option may seem convenient, but changing the file every time we want "the other" version is rather awkward.

A config file might be especially nice when using a revision control system, when we don't have to register the config in it.

Input format

Are we going to require a strict format for specifying chunks to be reused or do we want to allow for some flexibility (like in LaTeX itself)? Are we going to apply macros or environments?

Macros are much easier to code, but may be perceived as "inelegant" and user-nonfriendly (as opposed to environments) when dealing with larger chunks of text.

On the other hand, when using environments, we have to take care of space leaks.

Methods of saving and retrieving information in LaTeX

- token lists,
- macros,
- boxes,
- external files generated on-the-fly,
- hand-made external files.

Token lists

There are 256 token lists (32768 in ε-TeX), identified by a number. They are allocated by the LaTeX kernel, so you can't rely on a toks register having some number. This means that you'd probably want to use only one of them and perhaps accumulate content in it.

I have to admit that I don't have much experience with toks registers (at least not with using them for the purpose discussed)...

Macros

There may be infinitely many (well, at least potentially more than token lists) macros, and they are identified by name and not by number. This means that allocation is much easier.

However, in case of really large amount of stuff, we might run out of memory.

Also, macros (just like toks registers, for that matter...) and verbatim don't mix well.

Boxes

Box registers have limitations similar to toks, but have also some advantages: they are easier to "delimit" than toks or macros, and there is no problem with putting verbatim content into boxes.

Autogenerated files

Automatically (on-the-fly) generated files are very handy (LaTeX itself uses this approach, e.g., with the label/ref mechanism), but *may* cause problems when `inputenc` (or *fragile* commands) are used (unless the user `\protects` them).

For example, assume we want to typeset some part of a document verbatim and then normally (this is very useful when writing documentation on LaTeX packages etc.). A customary way to do it is to save it to an external file, then input it verbatim, then input it normally (this is exactly what the `demo` environment of a package called `sverb` does). However, if we use `inputenc` and non-ASCII characters in the chunk, we get wrong results. (The `filecontents` and `moreverb` packages do it right, but do not have an analogue of the `demo` environment.)

Hand-made external files

This may seem a stupid and/or primitive approach, but it turns out to be very useful (my package `xrcise` uses exactly this method). The main advantage is that it is very easy to specify different "versions" of the file without having to mess up with package options etc., which is especially handy when using a revison control system. It is also (obviously) useful when we have some content to be reused across multiple documents (bibliographies being probably the most obvious example).

In this approach, we prepare at least two files: the main ("master" or "driver") file (possibly in a few versions), and the file with the content to be reused.

Existing packages

General purpose:

- `filecontents`
- `moreverb`
- `sverb`
- `version[s]`
- `optional`
- `extract`

Problem sets:

- `answers`
- `probsoln`

(The above lists are not exhaustive.)

The filecontents, moreverb and sverb packages

These packages allow writing some text verbatim to an external file. (The sverb package doesn't work with inputenc.)

The version, versions and optional packages

These packages allow marking some parts of the document as "belonging" to a particular version and selecting a particular version in the preamble.

The extract package

The extract package creates an "extraction" file during an otherwise normal LaTeX run. This file can contain, for example, all \chapter commands and all \exercise environments from the main file. It can be then processed to obtain a version of the main document, but only with exercises.

The answers package

This package uses on-the-fly generated files. It seems to be not very user-friendly, but very configurable and quite powerful. The package does not really support handling different "versions" (with the only exception of putting all normally saved content in place of appearance in the source file).

The probsoln package

This package supports only "problems" and "solutions" (i.e., no hints, remarks etc), and supports different "versions" (i.e., making e.g. solutions visible or invisible). It can also support typesetting solutions in other part of the document through usage of a hand-made external file. As a bonus, the problems in the external file may be labeled and the user may choose to include only a (specified or random) subset of them. Unfortunately, the package is not as configurable as `answers`.

The xrcise package

Around 10 years ago, I didn't know of any package doing what I needed. And what I needed (well, maybe just *wanted*) was to have a package enabling me to specify exercises consisting of questions and (optionally): labels, source information, answers, solutions, remarks. Then, I wanted to be able to typeset a "teacher's version", including all the material, or "student's version", including only questions and hints. Additionally, I wanted to be able to typeset only the answers, in a compact format (as many answers in one line as can fit)—for example at the end of the "student's version"—so I didn't assume that each answer etc. is a paragraph on its own.

This way, the xrcise package was born. Currently I am reviewing the code and polishing it with the intent of uploading the package to CTAN soon.

The exercise file

All the exercises are contained in a file—call it `exercises.tex`—and have the following format:

```
\begin{exercise}
  \exerciselabel{first}
  \begin{question}
    Please answer the question.
  \end{question}
  \source{Source}
  \answer{42.}
  \begin{solution}
    This is the full solution text.
  \end{solution}
  \begin{remark}
    This question is stupid.
  \end{remark}
\end{exercise}
```

The driver file

The "driver" file may look as simple as

```
\documentclass{article}

\usepackage{xrcise}

\begin{document}

\input{exercises}

\end{document}
```

This typesets only the questions.

Selecting what to typeset and how

Each part of the exercise may be typeset as a "short" or "long" version. For example, to typeset the hint in a "short" version, one has to say `\usecommandversion{hint}{short}`. To typeset the answer in the "long" version, use `\usecommandversion{answer}{long}`. To omit remarks from the output file, use `\usecommandversion{remark}{void}`.

Usually, the "long" version is typeset in a paragraph(s) on its own, and preceded by a label like "Exercise 2" or "Hint:". On the other hand, a short version is typeset "inline" (without starting a new paragraph) and without any labels. (The labels are configurable through `\renewcommand`.)

Possible extensions

As can be easily seen, currently the package is rather crude. It would be nice to have, for example, package options (and corresponding commands) enabling typical scenarios. Another thing I consider is adding support for "hiding" commands like `\section` or `\chapter` in the exercise file. Yet another one would be configurable exercise parts.

I plan to include support for above features within a few weeks and upload the first version to CTAN.

All other feature requests are welcome!

Implementation issues

- Space leaks
- \par leaks
- Empty parts
- Switching commands

Space leaks

Since every environment is usually ended by `\end{foo}` (and relying on the user to type `\end{foo}%` is rather naive), one has to ignore the spaces after `\end`. This is achieved (unsurprisingly) by the `\ignorespacesafterend` command.

Using an `\unskip` at the beginning of the environment is also tricky, since it will result in an error when the environment starts in vertical mode. Thus, one may use `\ifhmode\unskip\fi` instead.

Finally, to get rid of the space after `\begin{foo}` it is enough to say `\ignorespaces` in the definition of the "opening" part of the environment.

\par leaks

In case of a "short" version of the `exercise` environment (for instance, when typestting only answers in a "compact" mode), one has to take care of blank lines between exercises (again, relying on the user not to have any blank lines there is a bad idea). This may be achieved by a simple trick:

```
\newcommand{\convertnextpartovoid}
  {\def\par{\let\par=\endgraf}}

\newenvironment{shortexercise}
  {\ifhmode\unskip\fi
   \refstepcounter{exercisenumber}\ignorespaces}
  {\aftergroup\convertnextpartovoid\ignorespacesafterend}
```

Empty parts of an exercise

Notice that `shortexercise` increments the counter, but does not typeset anything by itself (in fact, not even a space). This is right, since when typesetting e.g. only the answers we do not want to have an "empty" number in case of exercises without the answer, so it is the `\shortanswer` which actually typesets the number.

Switching commands

This part is a bit tricky. We want, for example, `\usecommandversion{answer}{short}` to perform `\let\answer=\shortanswer`. However, since also the right-hand part of the assignment needs to be inserted in `\csname...\endcsname`, we can't rely on `\expandafter`. Thus, we can do the trick in the following way:

```
\newcommand{\usecommandversion}[2]{%
  \edef\dousecommandversion{%
    \noexpand\let
      \expandafter\noexpand\csname #1\endcsname
      =\expandafter\noexpand\csname #2#1\endcsname
  \dousecommandversion
}
```

(In reality, this handles also `\endfoo` command, so that `\usecommandversion` works with both commands and environments.)

A macro-based approach

Another approach could be based on macros. For example, the answer to exercise number 5 might be stored in macro called `\csname answer5\endcsname` and so on. Then one could typeset all the answers by looping over all the exercise numbers, checking whether a suitably named macro exists (to cater for exercises without answers) and if it does, typesetting it in some way.

However, this would require to use commands and not environments (one could also use "fake environments", i.e., commands defined by a construct like `\def\foo #1 \end{foo}{\unskip ...}`, but this will not support `verbatim` text (nor other catcode tricks), and will issue a cryptic error message in case of a typo in `\end{foo}`).

Yet another approach

Use ConTEXt ;-) .

Thank you for your attention!

or

Wake up!