Typeset MMIX programs with TeX

Udo Wermuth

Abstract

A TeX macro package is presented as a literate program. It can be included in programs written in the languages MMIX or MMIXAL without affecting the assembler. Such an instrumented file can be processed by TeX to get nicely formatted output. Only a new first line and a new last line must be entered. And for each end-of-line comment a flag is set to indicate that the comment is written in TeX.

How to read the following program

The text that starts in the next chapter is a *literate program* [2, 1] written in a style similar to *noweb* [7]. Readers who are not familiar with literate programming might find the following remarks useful.

The program is divided into *sections*. Each section has a number that is written in bold at the beginning of the section. A section contains two parts and at least one of them must be present: (1) a *documentation part* with one or more paragraphs, and (2) a *code part* starting with a *headline* that is followed either by ≡ or +≡ and the *replacement code*. The headline has the format "⟨ *Name Number* ⟩".

Example: Sections 9 and 10 of the program have the respective headlines "⟨ List symbols that are special in TeX 9 ⟩ ≡" and "⟨ List symbols that are special in TeX 9 ⟩ +≡". Section 9 has five lines of replacement code and section 10 three.

The *Name* is the name under which the replacement code can be called by other sections. The *Number* is either the number of the section (the case with ≡) or a smaller number when a previously defined replacement code is extended with the new code lines (case +≡). In the second case the code part of the previous section that owns the smaller section number is followed by a line "See also sections ..." and the current section number is somewhere listed in the "...". Also, in the first case the code part is also often followed by a line "This code is used in sections ...". Then the headline of this section is used inside the replacement code of other sections (listed in the "..."), in which the final output of the complete replacement code with all extensions must be inserted. The number in the headline states the first section that contains code for it. So a reader sees in a call of a section where it starts and under this section he finds the other sections that add replacement code.

Example: In section 9 the lines "See also section 10." and "This code is used in section 24." are given. No such line appears in section 10 as it only extends the replacement code of section 9. (Note that section 10 has in its headline the number 9.) In section 24 the reference to section 9 stands for all of the eight code lines stated in sections 9 and 10.

If a section is not used in any other section then it is a *root* and during the extraction of the code a file is created that has the name of the root. This file collects all the code in the sequence of the referenced sections from the code part. The collection process for all root sections is called *tangle*. A second process is called *weave*. It outputs the documentation and the code parts as a TeX document.

Example: The following program has only one root that is defined in section 4 with the headline "⟨ mmix.tex 4 ⟩ ≡". The file that is created by the tangle process is therefore called "`mmix.tex`".

The tangled output in the original WEB system for literate programming is intended to be read only by computers (see [2], p. 116). In the present system output is created that is readable by humans. But changes to the program should only be made in the original source of the literate program.

The following text is the output that TeX has produced from the woven document. (A few edits have been made to follow the style of this journal.)

Contents

Introduction

1. Algorithms in *The Art of Computer Programming* (TAOCP) [3] by Donald E. Knuth are stated in plain English. But every time an implementation is needed a machine language or assembler language is used. In the first three volumes the language is MIX; in Volume 4a the algorithms are implemented in the language of a new computer called MMIX [4]. For the next editions of the TAOCP volumes all the

MIX programs of the first three volumes must be rewritten either in MMIX or in the new assembler language MMIXAL. On his web page `http://www-cs-staff.stanford.edu/~uno/mmix.html`, Knuth asks volunteers to start the conversion of the MIX programs before he has finished Volume 5 and new editions of Vols. 1–3 are created.

2. I decided to be one of the volunteers to do the conversion (although I'm not an MMIXmaster; see [6]). I asked myself the question: How to present the result? The MMIX programs are stored in `mms` files, which allow a very flexible input format. For example, a line that starts with a backslash will be treated as a comment and is ignored during the assembly.

So my idea is to write the `mms` files in a way that they can be processed not only by the assembler but also by TeX. The output of TeX shall reflect the style that is used in the TAOCP volumes to present the MIX and MMIX programs. And TeX can be used to implement a second idea: Not only shall the conversion be done but an analysis of the new implementation shall be added.

A macro package for TeX is developed that is included in the `mms` files. Then TeX is able to process and pretty-print such `mms` files.

3. Other volunteers for the conversion from MIX to MMIX have obviously felt the same need for TeX output. The solution on the MMIX *home page* [6] is a `lex` script `mmixtotex.l` to create a program that reads the `mms` file and outputs a TeX file that can be typeset with a macro package *mmstotex.sty*.

4. Here is the plan for the macro package.

⟨ mmix.tex 4 ⟩ ≡
⟨ Initialization 5 ⟩
⟨ Definitions 24 ⟩
⟨ Useful commands and shortcuts 44 ⟩
⟨ Add an analysis of the algorithm 40 ⟩
⟨ Format the `mms` file 19 ⟩
⟨ Take off 33 ⟩

This is a root.

5. The file `mmix.tex` might be shipped with an `mms` file without this description. Therefore I will be adding plenty of comments to the TeX code to help others to read and understand the macro package.

⟨ Initialization 5 ⟩ ≡
```
% Package to format MMIX programs with TeX
% (and some useful commands to document them)
% Author: Udo Wermuth
% %%%
```
⟨ Description 14 ⟩
```
% %%%
```

See also sections 11, 12, and 13.
This code is used in section 4.

Format of the output

6. Most of the time programs are not written in MMIX but in the MMIX *Assembly Language*, called MMIXAL. MMIXAL allows labels, alphabetic names, etc. and introduces new operations that are called pseudo-ops. As we are only interested in formatting the source lines of a program the details of the extensions provided by MMIXAL are not discussed here. The reference [4] defines not only MMIX but gives all the information about MMIXAL too. A source line of a MMIXAL program has up to five elements. Three elements are of principal interest for the program behavior: (1) an optional label, (2) the operation (or short: the *op-code*), and (3) an expression field. The other two elements are not needed for the execution of the program but for the analysis and the comprehension: (4) optional timing information, i.e., the number of times the statement is executed in a run, and (5) an optional comment.

For the presentation of the program one more element is printed: an optional line number. The line number is printed in italics, the elements 1–3 are output verbatim in a monospaced font, element 4 is written in math mode, and element 5 is formatted by TeX as normal text. Therefore the output shall look like this:

line	label	op-code	expression	time	comment
07	`Maximum`	`SL`	`kk,$0,3`	1	*M1. Initialize.*

7. Following this example, let us state the complete requirements for the output format:

R1 The line number is either empty or has two or three digits; leading zeros are printed. It is written left-aligned in 9 pt italics.

R2 The label is optional. If it is present it is written verbatim in a 10 pt monospaced font.

R3 The op-code is written verbatim in the monospaced font.

R4 The expression field may contain one or more items but it does not contain a blank (except in a string). Like the label and the op-code it is printed verbatim in the monospaced font.

R5 The timing information is optional. If present, it is printed in 9 pt as a math expression centered in its column.

R6 The optional comment is written in a 9 pt roman font. It is written in TeX.

R7 Lines that contain only a comment written in the monospaced font are allowed. Lines with more than one source statement are allowed.

R8 The program source ends with a thick vertical bar in the comment area.

R9 It is possible to add a runtime analysis. The text uses a 10 pt roman font.

R10 The output shall show the name and the source of the MIX program, the name of the author, who programmed the MMIX source, and the date of the conversion.

Preparation

8. The monospaced font is a 10 pt font (see R2), the other fonts have size 9 pt (R1, R5, and R6). The 9 pt fonts (and the 6 pt fonts for subscripts) are not activated in plain TEX. Let's give them names.

⟨ Fonts 8 ⟩ ≡
```
% name 9pt and 6pt fonts
\font\ninerm=cmr9  \font\sixrm=cmr6
\font\ninesy=cmsy9 \font\sixsy=cmsy6
\font\ninei=cmmi9  \font\sixi=cmmi6
\font\nineit=cmti9    \font\ninesl=cmsl9
\font\ninett=cmtt9
\font\ninebf=cmbx9 \font\sixbf=cmbx6
```
This code is used in section 24.

9. The example in section 6 shows that MMIXAL uses characters that have a special meaning in TEX, for example, the dollar sign and hash mark are important symbols in MMIXAL. Therefore the output must be filtered and the functions assigned to the special characters in TEX have to be deactivated.

Plain TEX provides a `\dospecials` command, but let us separate the MMIXAL and TEX special characters.

⟨ List symbols that are special in TEX 9 ⟩ ≡
```
% special in TeX but common in MMIX
\def\mmixdospecials{\do\ \do\$\do\&\do\#%
                    \do\^\do\_\do\%\do\~}
% remaining special characters in TeX
\def\texdospecials{\do\\\do\{\do\}}
```
See also section 10.
This code is used in section 24.

10. To switch off the special meaning of the above listed characters a command from *The TEXbook* [5], p. 380, is used.

⟨ List symbols that are special in TEX 9 ⟩ +≡
```
\def\uncatcodespecials{% redef special chars
  \def\do##1{\catcode`##1=12 }%
  \mmixdospecials\texdospecials}
```

11. In the header the author, the name of the program and the original source are listed. The footer states the date and provides a page number. This fulfills requirement R10.

⟨ Initialization 5 ⟩ +≡
```
% Header and Footer
\headline={\sevenrm Author: \authorH\hfill
           Program: \pgmnameH.mms (\sourceH)}%
\footline={\sevenrm Date: \dateF\hfill
           \sevenbf\folio}%
```

12. The printed document shall not only show the program but also provide the possibility of including an analysis of the algorithm (R9). The analysis is placed in a second file (a plain TEX file) and it is included with an `\input` statement. The name of the file is created from the name of the program extended by the suffix `_aoa` and, of course, with file extension `.tex`.

⟨ Initialization 5 ⟩ +≡
```
% file name for the ``Analysis of Algorithm''
\def\AoAfile{\pgmnameH_aoa.tex}
```

13. The name of the program is by default the name of the MMIXAL file — but the user has the ability to override that name. The name of the author, the source location and the date are initialized with some text, but it is expected that the user specifies them before `mmix.tex` is loaded. The external control sequences are copied and made `\undefined`.

⟨ Initialization 5 ⟩ +≡
```
\def\checkextdata{%
  \ifundef pgmname \def\pgmnameH{\jobname}%
  \else\let\pgmnameH=\pgmname
    \let\pgmname=\undefined
  \fi
  \ifundef author \def\authorH{Unknown}%
  \else\let\authorH=\author
    \let\author=\undefined
  \fi
  \ifundef source \def\sourceH{TAOCP}%
  \else\let\sourceH=\source
    \let\source=\undefined
  \fi
  \ifundef date \def\dateF{\number\year}%
  \else\let\dateF=\date
    \let\date=\undefined
  \fi}
```

14. The values for date, author and source must be declared outside of the package. Let us document this at the beginning of the package.

⟨ Description 14 ⟩ ≡
```
% before the macro package is loaded the
%    following must be \def'ed
%    required: \date, \author, \source
% the program name is taken from the mms-file
%    but it can be overwritten
%    optional: \pgmname
```
See also sections 26, 41, and 53.
This code is used in section 5.

Line numbers

15. The output format states all the information about line numbers, as they are not part of the input file. Of course a counter for the numbers is needed.

⟨ Counters 15 ⟩ ≡
```
% count registers
\newcount\lnocnt % counter for line numbers
```
See also section 25.
This code is used in section 24.

16. Next the width of the column for the line numbers has to be defined. Two cases are stated in the requirements: 2 and 3 digits (see R1).

⟨ Dimensions 16 ⟩ ≡
```
% dimen registers
\newdimen\lnotwodigitswidth   % 2 digits col
\newdimen\lnothreedigitswidth % or 3 digits
```
See also section 22.
This code is used in section 24.

17. Here are the default widths of the columns.

⟨ Set values of dimen-registers 17 ⟩ ≡
```
{\setbox0=\hbox{\nineit 00\tentt\quad}%
 \global\lnotwodigitswidth=\wd0
 \global\lnothreedigitswidth=1.25\wd0 }%
```
See also section 23.
This code is used in section 35.

18. Lines can only be numbered if the space is reserved for the column of numbers: `\colforlnotrue` must be set. Then a number is printed if the flag `\ifnumberlines` is true. A third flag is needed to set the number of digits for line numbers.

⟨ Flags 18 ⟩ ≡
```
% if flags
\newif\ifcolforlno     % true: add col for lno
\newif\ifnumberlines   % true: number the lines
\newif\ifthreedigitlno % true: use 001..999
```
See also section 43.
This code is used in section 24.

19. The output routine for line numbers prints leading zeros (R1).

⟨ Format the `mms` file 19 ⟩ ≡
```
\def\printlinenumber{% with leading 0s
  \ifthreedigitlno % how many digits?
    \hbox to \lnothreedigitswidth{\it
      \ifnum\lnocnt<100 0\fi
      \ifnum\lnocnt<10 0\fi \number\lnocnt
      \hss}%
  \else\hbox to \lnotwodigitswidth{\it
      \ifnum\lnocnt<10 0\fi \number\lnocnt
      \hss}%
  \fi}
```
See also sections 20, 21, 27, 28, 29, 30, 31, 32, 34, 35, and 38.
This code is used in section 4.

20. But before the output routine can be called it must be checked that line numbers shall be printed at all. Therefore the following macro is called to output the line number.

⟨ Format the `mms` file 19 ⟩ +≡
```
\def\numbermmixline{% shall no. be printed?
  \ifcolforlno
    \ifnumberlines
      \global\advance\lnocnt by 1
      \printlinenumber            % yes
    \else\phantom{\printlinenumber}% no
  \fi\fi}
```

21. To have the style of line numbers available if a comment or the text of the analysis of the algorithm needs to reference a line number, one more control sequence is provided. The command is used in text printed in roman type. It gets either a single line number or a range of line numbers and prints this in italics. So a simple solution is implemented which doesn't force the user to type in the italic correction.

⟨ Format the `mms` file 19 ⟩ +≡
```
\def\pgmline#1{% print #1 as line number
  \gdef\argpgmline{#1}% store #1 and look ahead
  \futurelet\next\pgmlinex}
\def\pgmlinex{% check if \next is . or ,
  \if.\next {\it\argpgmline}%      no \/
  \else\if,\next {\it\argpgmline}% no \/
  \else {\it\argpgmline\/}%       add \/
  \fi\fi}
```

Times column

22. The optional column for the timing information (R5) gets its own dimen register.

⟨ Dimensions 16 ⟩ +≡
```
\newdimen\timecolumnwidth % column for time
```

23. To allow entries like "$A - 1$" the column must be wider than three symbols.

⟨ Set values of dimen-registers 17 ⟩ +≡
```
{\setbox0=\hbox{$2M+M$}% 10pt gives white space
 \global\timecolumnwidth=\wd0 }%
```

24. Of course, as the column is optional one more flag needs to be declared.

It is time to collect all definitions in a sorted list. Such a list might be easier to understand if the `mmix.tex` file comes without this documentation, so some sub-entries are created.

⟨ Definitions 24 ⟩ ≡
```
% %%% Definitions
⟨ Counters 15 ⟩
⟨ Dimensions 16 ⟩
⟨ Flags 18 ⟩
\newif\iftimeinfostated % true: add time col
⟨ Fonts 8 ⟩
⟨ List symbols that are special in TeX 9 ⟩
```
See also section 39.
This code is used in section 4.

Setting the output format

25. To identify the size of the field for the line numbers a hint must be given by the author of the program. This hint is a counter called `\mmixtype`. Three cases must be considered according to R1: The values 0 and 1 mean no line numbers are used, 2 and 3 stand for two-digit line numbers, and 4 and 5 for three-digit numbers.

And requirement R5 is also covered: If the value is odd the timing information is present: 0, 2, and 4 format the program without timing information, but 1, 3, and 5 have such information.

And a third bit of information is included: if the number is positive the line numbering starts immediately. Otherwise a command must be given to start the numbering.

⟨ Counters 15 ⟩ +≡
```
\newcount\mmixtype % a value between -5 and 5
% -1,0,1: no line numbers, no space reserved
% absolute value 2,3: add column for 2 digits
% absolute value 4,5: add column for 3 digits
% > 1: start line numbering directly
% <-1: a user command starts numbering
% write a line (‘!’ is the commchar) with
%   even value: label op expr ! comment
%   odd value:  label op expr ! time ! comment
```

26. I decided to set this counter by an “assignment” to the macro package. Therefore, a typical first line looks like the following lines in the comment.

⟨ Description 14 ⟩ +≡
```
% start a programm with all \def’s in one line
%    (n is the \mmixtype explained elsewhere):
%    \def\date{<date>}\def\author{<name>}
%    \def\source{<volume, page>}\input mmix =n
```

27. The value of `\mmixtype` determines the values of all flags. They are set even when `\mmixtype` has a value outside the defined range from −5 to +5. Such an error situation is tested and reported later.

⟨ Format the `mms` file 19 ⟩ +≡
```
% %%% Format
\def\setflagsformmixtype{% analyse \mmixtype
  \ifnum\mmixtype<0
    \mmixtype=-\mmixtype % wait with numbering
  \else
    \numberlinestrue % prep. to number 1st line
  \fi
  \ifnum\mmixtype>1 % activate numbering
    \colforlnotrue
    \ifnum\mmixtype>3 % use 3 digits
      \threedigitlnotrue
  \fi\fi
  \ifodd\mmixtype % timing info is present
    \timeinfostatedtrue
  \fi}
```

28. In the case that `\mmixtype` < -1 the numbering of lines is activated by user commands. They are placed in the comment of a source line.

⟨ Format the `mms` file 19 ⟩ +≡
```
% start and stop line numbering
\let\startnumbering=\numberlinestrue
\let\stopnumbering=\numberlinesfalse
```

Input format

29. How shall the MMIXAL program line of section 6 be entered into the input file? Requirements R2–R4 state that a monospaced font is used for the above defined elements 1–3. As special symbols of TeX might be present the best way to typeset them is to use verbatim mode. My idea is to use a special character that ends the verbatim mode, which is automatically started in every line, and then to format the rest of the line in TeX. Such a flag makes it possible to fulfill the requirements R6 and R7. I call this special character the *commchar* and by default the exclamation mark is used for it.

A single commchar is required if no timing information is present and two are used to identify the timing information. The above stated program line is therefore coded like this:

```
Maximum SL  kk,$0,3  !1!  \step M1. Initialize.
```

The label, the op-code, and the expression must always start at the same column to be properly aligned in the output. The macros shouldn’t destroy other input styles, for example, several MMIXAL statements might be written in one line (see R7).

Note: The control sequence `\step` is one of the useful macros defined later in this package.

⟨ Format the `mms` file 19 ⟩ +≡
```
\def\setcommchar#1{% boundary for verbatim
  \vskip-\baselineskip% for first end of line
  \gdef\commchar{#1}%
  \def\par{\endgraf\verbatim#1}}
```

30. The verbatim mode is defined in a standard way (see *The TeXbook* [5], pp. 380–382).

⟨ Format the `mms` file 19 ⟩ +≡
```
% Verbatim macros
\def\verbatim{\begingroup % ends in \doverbatim
  \setupverbatim
  \doverbatim}
\def\setupverbatim{%
  \def\par{\leavevmode\endgraf\noindent}%
  \catcode‘\‘=\active
  \obeylines
  \uncatcodespecials
  \obeyspaces}
% now make a blank a control space
{\obeyspaces\global\let =\ }%
% and avoid ligatures of ? and ! with ‘
{\catcode‘\‘=\active \gdef‘{\relax\lq}}%
```

31. When the verbatim mode is executed the tests for the line numbering and the timing information are made. The change of `\everypar` and `\par` is reverted in the comment field to allow, for example, a command like `\smallskip`.

Note that a missing second commchar with an odd `\mmixtype` results in a couple of errors (runaway argument) in `\printtimeinfo`, but forgetting the second commchar seems unlikely.

⟨ Format the mms file 19 ⟩ +≡
```
\long\def\doverbatim#1{% #1 is the commchar
  \everypar{\numbermmixline}%
  \def\nextmmixline##1#1{\noindent
    \tentt##1%
    \endgroup % opened in \verbatim
    \printtimeinfo}%
  \iftimeinfostated
    \gdef\printtimeinfo##1#1{% #1<time>#1
      \hbox to \timecolumnwidth{%
        \hss$ ##1 $\hss}\resetpar}%
  \else\global\let\printtimeinfo\resetpar
  \fi
  \nextmmixline}
\def\resetpar{\everypar{}\let\par=\endgraf
  \ignorespaces}
```

32. The exclamation mark seems to work fine as the commchar, but there might be reasons to switch the commchar in a program.

⟨ Format the mms file 19 ⟩ +≡
```
\def\newcommchar#1{% change the commchar
  \gdef\commchar{#1}%
  \def\par{\endgraf\verbatim#1}% new \par
  \obeylines}% end-of-line is the new \par
```

Activation

33. In order to get all macros working together they must be called in a certain sequence. First, the macro `\setflagsformmixtype` has to be executed, which makes the value of `\mmixtype` positive and sets the flag for immediate line numbering. Next, the commchar must be defined. The command starts the verbatim mode after the first `\par` command and reads in the first line of the MMIXAL program. And of course the assignment statement for the `\mmixtype` must be executed. All this is done in the following way:

a) With `\afterassignment` a (not yet opened) group is closed; the counter `\mmixtype` gets the value that appears after the `\input mmix`.
b) A group is opened. It will be closed with the construction in a).
c) With `\aftergroup` the following sequence is prepared:
   - `\setflagsformmixtype` is called;
   - `\setcommchar` is called (in a group);
   - an exclamation mark is given as an argument for `\setcommchar`;
   - `\obeylines` is called;
   - and `\par` starts the verbatim mode.
d) Finally, `\global\mmixtype` is the left side of the assignment statement (see a)).

And here is the place where we call the test for the value of `\mmixtype`. If an error would be reported in the macro `\setflagsformmixtype` the user would see a bunch of tokens that have to be read again. Therefore the test for an error is made just before `\par` at the end of part c) is executed and the verbatim mode starts.

⟨ Take off 33 ⟩ ≡
```
% %%% Start
⟨ Last-minute procedures 36 ⟩
\def\getmmixtype{% needs a ‘‘right side’’
  ⟨ Prepare the environment 37 ⟩
  \afterassignment\egroup% an assignment ends
  \bgroup                          % this group
  \aftergroup\setflagsformmixtype
  \aftergroup\begingroup% closed in last line
  \aftergroup\setcommchar
  \aftergroup!% this is the default commchar
  \aftergroup\obeylines
  \aftergroup\testvalueofmmixtype
  \aftergroup\par
  \global\mmixtype}% now get \mmixtype
\getmmixtype
```
This code is used in section 4.

34. What has to be done to prepare the environment? Answer: set the fonts, initialize the variables (external and internal) and handle TEX comments.

The `\ninepoint` macro of *The TEXbook* [5], pp. 414–415, does more than we need, but it shows how to switch to the 9 pt fonts.

⟨ Format the mms file 19 ⟩ +≡
```
% switch to 9pt fonts
\def\setupfonts{\def\rm{\fam0\ninerm}%
  \textfont0=\ninerm  \scriptfont0=\sixrm
  \textfont1=\ninei   \scriptfont1=\sixi
  \textfont2=\ninesy  \scriptfont2=\sixsy
  % no changes for fam3
  \def\it{\fam\itfam\nineit}%
        \textfont\itfam=\nineit
  \def\sl{\fam\slfam\ninesl}%
        \textfont\slfam=\ninesl
  \def\tt{\fam\ttfam\ninett}%
        \textfont\ttfam=\ninett
  \def\bf{\fam\bffam\ninebf}%
        \textfont\bffam=\ninebf
        \scriptfont\bffam=\sixbf
  \def\oldstyle{\mit\ninei}%
  \rm}
```

35. Here the variables are set to their initial values.

⟨ Format the mms file 19 ⟩ +≡
```
% set the variables to their default values
\def\setvariables{\mmixtype=0 \lnocnt=0
  ⟨ Set values of dimen-registers 17 ⟩
  \colforlnofalse     \numberlinesfalse
  \threedigitlnofalse \timeinfostatedfalse}
```

36. One problem remains: If a TeX comment is placed at the end of the comment field the end-of-line information is not available and the verbatim mode isn't restarted. So the `%` is made active. It gobbles the comment and behaves like the current definition of `\par`.

⟨ Last-minute procedures 36 ⟩ ≡
```
{\obeylines \catcode`\%=\active
 \gdef\handleTeXcomments{\catcode`\%=\active
   {\obeylines \gdef%##1
    {\endgraf\expandafter\verbatim\commchar}}}}%
\def\resetTeXcomment{\catcode`\%=14 }
```
This code is used in section 33.

37. Now we collect the pieces together to prepare the environment.

⟨ Prepare the environment 37 ⟩ ≡
```
\checkextdata   \setupfonts
\setvariables   \handleTeXcomments
```
This code is used in section 33.

38. One task is open: To give a warning message if `\mmixtype` is out of range. I prefer to issue an error message. The following macro is called after `\mmixtype` was made positive.

⟨ Format the mms file 19 ⟩ +≡
```
\def\testvalueofmmixtype{% value must be < 6
  \ifnum\mmixtype>5
    \errhelp\mmixtypeerror
    \errmessage{The number \string\mmixtype
       \space must be between -5 and 5}%
  \fi}
```

39. We append the help message to the definitions.

⟨ Definitions 24 ⟩ +≡
```
% help messages
\newlinechar=`\^^J
\newhelp\mmixtypeerror{%
 mmixtype is the number
   stated after \string\input\space mmix.^^J%
 It must be between -5 and 5.
 Three aspects are coded into it:^^J%
 if it is odd
   time information is given (use two !);^^J%
 if it is -1, 0, 1
   no line numbers are present;^^J%
 if it is -3, -2, 2, 3
   line numbers have two digits;^^J%
 if it is -5, -4, 4, 5
   line numbers have three digits;^^J%
 if it is >1
   immediate numbering of lines is started.}%
```

Last line

40. At the end of the program source the possibility of including a separate file with the analysis of the algorithm shall be given (see R9). The name of the file was already defined above. The text shall be printed in a roman font of size 10 pt. Therefore we must switch back to 10 pt before that section can start.

⟨ Add an analysis of the algorithm 40 ⟩ ≡
```
% %%% Macros for the last line(s)
% typeset the Analysis of the Algorithm
\def\Analysis{\medbreak
  \def\rm{\fam0\tenrm}% back to 10pt
  \textfont0=\tenrm \scriptfont0=\sevenrm
  \textfont1=\teni  \scriptfont1=\seveni
  \textfont2=\tensy \scriptfont2=\sevensy
  % fam3 was not changed
  \def\it{\fam\itfam\tenit}%
         \textfont\itfam=\tenit
  \def\sl{\fam\slfam\tensl}%
         \textfont\slfam=\tensl
  \def\tt{\fam\ttfam\tentt}%
         \textfont\ttfam=\tentt
  \def\bf{\fam\bffam\tenbf}%
         \textfont\bffam=\tenbf
         \scriptfont\bffam=\sevenbf
  \def\oldstyle{\mit\teni}%
  \rm % activate \tenrm
  \noindent{\tenbf Analysis}\par% the headline
  \nobreak\smallskip\noindent}
```
See also section 42.
This code is used in section 4.

41. The section with the analysis is started with the last line of the file. Similar to the first line it follows a special convention.

All programs have to use the control sequence `\eop`. It typesets a thick vertical rule as it is stated in requirement R8. It also stops the line numbering.

⟨ Description 14 ⟩ +≡
```
% use \eop in the comment of the last
%    source line
% end the input file with a line that
%    contains either (! is the commchar)
%    ``!\endprogram\bye'' (even \mmixtype)
%    or ``!!\endprogram\bye'' (odd \mmixtype)
%    or use \endwAoA instead of \endprogram
%       to input a file with an analysis
```

42. At the end of the input file a group is still open that must be closed. And `%` gets back its default meaning. But first, up to two empty hboxes are deleted (that might have been created on the current horizontal line) to avoid a line break in front of this "empty line".

⟨ Add an analysis of the algorithm 40 ⟩ +≡
```
\def\eop{% end of program symbol
  \qquad\vrule height 7pt depth 1pt width 3pt
  \eopusedtrue\stopnumbering}
```

```
\def\clearline{% remove 0--2 empty hboxes
  {\setbox0=\lastbox \setbox0=\lastbox}}
\def\endprogram{\clearline
  \ifeopused\eopusedfalse
  \else\message{^^JWarning: end the program
              with \string\eop^^J}%
  \fi
  \endgroup% opened in \getmmixtype
  \resetTeXcomment}
\def\endwAoA{\endprogram\bigskip
  \Analysis \input\AoAfile}
```

43. ⟨ Flags 18 ⟩ +≡

```
\newif\ifeopused % true: ''\eop'' was used
```

Shortcuts

44. When the analysis is written some commands for often-used idioms reduce the amount of typing.

⟨ Useful commands and shortcuts 44 ⟩ ≡

```
% %%% Useful commands
\def\MIX{{\ninett MIX}}
\def\MMIX{{\ninett MMIX}}
\def\MMIXAL{{\ninett MMIXAL}}
\let\NULL\Lambda % the null link
\def\AVAIL{\hbox{\ninett AVAIL}}% free space
\let\Gets\Leftarrow % get space from AVAIL
\let\implies\Rightarrow % more ''logical''
% units for the analysis: oops and mems
\def\oops{\hbox{$\upsilon$}}\let\oop=\oops
\def\mems{\hbox{$\mu$}}     \let\mem=\mems
% reference to equation numbers of TAOCP
\def\numeq(#1){\hbox{$({\oldstyle#1})$}}
\def\eq(#1){% outputs Eq. (...)
   \hbox{Eq.\thinspace\numeq(#1)}}
\def\Eq(#1){% outputs Equation (...)
   \hbox{Equation \numeq(#1)}}
```

See also sections 45, 46, 48, 49, 50, 51, and 52.
This code is used in section 4.

45. Some commands and shortcuts are needed in the comments to a program. For example, the steps of an algorithm are labeled with the identifying letter of the algorithm, a number, and a phrase. This information is often stated in the comment to a program. (The phrase might be omitted; for example, see [3], Vol. 1, p. 236.)

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
% steps in algorithms
\def\algidphrase#1#2#3.#4.{% #1 #arguments;
  % #2 phrase delimiter; #3 step id; #4 phrase
  $\underline{\hbox{\sl
      \vphantom{y}#3.%
      \ifnum #1>1 \enspace #4#2\fi}}%
  $\space}
\def\step#1. #2.{\algidphrase2.#1.#2.}
\def\steq#1. #2?{\algidphrase2?#1.#2.}
\def\stepid#1.{\algidphrase1-#1.-.}
```

46. Sometimes several lines get a single comment: In [3], Vol. 1, p. 258 and 278 (and in [4], p. 107) a right brace is used to collect the statements for a comment. Place the command `\mlsc` (multiple lines, single comment) in the middle or just above the middle of the lines that get one comment.

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
% place the command in (odd number) or
% just above (even number) the middle
\def\mlsc#1:#2{% #1 #lines; #2 comment
  \smash{\ifodd#1\else\lower.45\baselineskip\fi
    \hbox{$% next line: see \TeX book, p.194
      \openup-1\jot % cancel for \eqalign
      ⟨ Compute \dimen255 from #1 (i.e., #lines) 47 ⟩
      \left.\kern-.5em % empty left brace
      \eqalign{\vrule height\dimen255
                    width 0pt depth 0pt }%
      \right\}% visible right brace
      $\thinspace#2}}}
```

47. The height of the brace is of course roughly the number of lines, which are combined by the brace, multiplied by the `\baselineskip`. I use the formula:

$$\text{number of lines} \times (\texttt{\textbackslash baselineskip} + 3\,\text{pt}) - 8\,\text{pt}.$$

⟨ Compute `\dimen255` from `#1` (i.e., #lines) 47 ⟩ ≡

```
% compute height of brace
\dimen255=\baselineskip
\advance\dimen255 by 3pt
\multiply\dimen255 by #1\relax
\advance\dimen255 by -8pt
```

This code is used in section 46.

48. Next some special constructions: The dot minus (*monus operation* or *saturating subtraction*) is a binary operation defined by $a \dot{-} b = \max(0, a - b)$. It isn't coded like `\doteq` as it is not a relation and some care must be taken with the position of the dot. The notation of the conditional expression is changed in TAOCP, Vol. 4a.

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
% special operations
\def\dm{% dot minus: saturating subtraction
  \mathbin{\mathop{\kern0pt \smash{-}}%
    \limits^{\raise.55ex\hbox{$\textstyle.$}}}}
\def\ite(#1?#2:#3){% if-then-else; Vol.4a, p.96
  (#1\,{\rm?\ }#2{\rm:}\enspace#3)}
```

49. The following shortcuts make special symbols of TEX available for comments.

The plain TEX command for `\l` is redefined here. The original definition is stored in `\lstroke`.

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
% symbols of \TeX
\def\vs{{\tt\char32 }}% visible space
\def\bs{{\tt\char92 }}% backslash
\def\bo{{\tt\char123 }}% open brace
\def\bc{{\tt\char125 }}% close brace
\let\lstroke\l
\def\l\_{{\tt\char95 }}% long underline
\def\h\#{\hbox{${}^\#$}}% high # (hex no.)
```

More shortcuts

50. Here are some shortcuts that I find useful. In a comment short words must be processed in math mode either as roman text or monospaced text. So I define a couple of commands for that.

Often text must be placed in an hbox. And a short cut for an array with a roman or monospaced name and a math mode index is quite useful.

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
% %%% my shortcuts
% output rm or tt in math with 1 to 3 chars
\def\r#1{{\rm #1}}
\def\rr#1#2{{\rm #1#2}}
\def\rrr#1#2#3{{\rm #1#2#3}}
\def\m#1{{\tt #1}}
\def\mm#1#2{{\tt #1#2}}
\def\mmm#1#2#3{{\tt #1#2#3}}
% output of rm or tt text in boxes or arrays
\def\rb#1{\hbox{\rm #1}}% rm box
\def\mb#1{\hbox{\tt #1}}
\def\ra#1[#2]{\hbox{\rm #1[$#2$]}}% rm array
\def\ma#1[#2]{\hbox{\tt #1[$#2$]}}
```

51. I add a comment in front of a subroutine or procedure. A few lines describe the calling sequence, the entry and exit conditions, and changed special or global registers. This is described on page 55 of [4]. I start such comments indented at the column of the op-code and with a `>` that sticks out to the left. To get this alignment in the case when timing information is present some care must be taken.

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
\def\gts{% align 'g' with the op-code col
  \iftimeinfostated
    {% omit time column if \mmixtype is odd
     \ninerm\hskip-\timecolumnwidth
     {\tentt\ }}% add space for 2nd commchar
  \fi
  {\tentt>\space }}
```

Final remarks

52. The following command doesn't produce any output. Nevertheless I find it useful in the analysis of the algorithm. The timing information states how often a source line is executed but for a line with a branch instruction it is also useful to know how often a bad branching decision was made.

Therefore I place the following command directly after the second commchar and state the number of bad decisions.

⟨ Useful commands and shortcuts 44 ⟩ +≡

```
% used to state number of bad decisions
\def\bad#1\bad{\ignorespaces}% no output
```

53. The macros have been presented for a single `mms` file. But the conversion project needs to rewrite many MIX programs. So, to create a book of converted programs, for example, for a complete chapter of TAOCP, the individual files can be `\input` in a main file which is then processed by TeX. (Of course the main file must include a definition like, for example, `\let\goodbye=\bye` and then the redefinition of `\bye`: `\outer\def\bye{\par\vfill\supereject \endinput}`.)

To avoid reloading this package a test is added that determines if the package is already known. And all the counters, fonts etc. are reset to their initial value.

Note that `\endinput` and `\fi` must appear in the same line (see *The TeXbook* [5], p. 214).

⟨ Description 14 ⟩ +≡

```
% %%%
% don't load the file several times
% but reset variables, fonts etc.
\def\ifundef #1 {% see \TeX book, ex. 7.7
  \expandafter\ifx\csname #1\endcsname\relax}
\ifundef mmixisloaded \def\mmixisloaded{true}%
\else\getmmixtype\endinput\fi
```

54. To test the scripts and to give an example of how to use the macro package a small example is shown in the appendix.

Index and List of sections

55. A literate program comes usually with an index of the names of used identifiers — variables, types, functions, procedures, or whatever the used programming language offers. It includes also certain aspects of the program that might be of interest to users or developers who want to change the code. For example, error messages are listed.

The index lists the section numbers, in which the entry appear. The section number, in which an identifier is defined, is written in slanted digits.

56. The second index collects all headlines of the code parts. Here the headlines contain all section numbers that define the replacement code for the section name.

References

[1] Bart Childs, “Thirty years of literate programming and more?” *TUGboat* **31**(2010), 183–188. `http://tug.org/TUGboat/tb31-2/tb98childs.pdf` (accessed: August 4, 2014)
[2] Donald E. Knuth, *Literate Programming*, CSLI Lecture Note No. 27, 1992. `http://www-cs-staff.stanford.edu/~uno/lp.html` (accessed: August 4, 2014)
[3] Donald E. Knuth, *The Art of Computer Programming*, Addison-Wesley, Vol. 1 (3rd ed.), 1997; Vol. 2 (3rd ed.), 1998; Vol. 3 (2nd ed.), 1998; Vol. 4a (1st ed.), 2011. `http://www-cs-staff.stanford.edu/~uno/taocp.html` (accessed: August 4, 2014)
[4] Donald E. Knuth, *The Art of Computer Programming — MMIX: A RISC Computer for the new Millennium*, Vol. 1, Fascicle 1, Addison-Wesley, 2005. `http://www-cs-staff.stanford.edu/~uno/mmix.html` (accessed: August 4, 2014)
[5] Donald E. Knuth, *The TeXbook*, Volume A of *Computers & Typesetting*, Addison-Wesley, 1984.
[6] MMIX home page, hosted by: The MMIX Group at Munich University of Applied Sciences. `http://mmix.cs.hm.edu` (accessed: August 4, 2014)
[7] Norman Ramsey, “Literate programming simplified”, *IEEE Software* **11** (1994), 97–105. `http://www.cs.tufts.edu/~nr/noweb/` (accessed: August 4, 2014)

◇ Udo Wermuth
Babenhäuser Straße 6
63128 Dietzenbach
Germany
`u dot wermuth (at) icloud dot com`

Appendix: An Example

First the input (file `1-3-3I.mms`) is shown. Note that commchars are used only in lines that contain a comment. Line numbers and timing information are given only for the lines that belong to the subroutine.

The value of `\mmixtype` is −3 meaning that (a) the first line is not numbered (the value is negative), (b) line numbers need only two digits (i.e., value is −2 or −3), and (c) timing information is given (so value must be odd).

```
\def\date{04 Aug 2014}\def\source{V1, p.\ 177}\def\author{Udo Wermuth}\input mmix =-3
!!\clearline{\tenbf Program I} ({\tenit Inverse in place\/})% use a lot of ‘‘features’’
!!\clearline\smallskip\timecolumnwidth=2.5em % (some are not necessary in this conversion)
n         GREG      6                   !! Number of elements in the permutation
j         IS        $0                  !! Variables of the algorithm
i         IS        $1
mm        IS        $2                  !! $\mm mm = 8m$
          LOC       Data_Segment
X         GREG      @
          OCTA      0                   !! $X[0]$ is not used
          OCTA      6,2,1,5,4,3         !! The data of Table 1.3.3--3
          LOC       #100
       !!\gts Inverse a permutation in place
       !!\gts Entry condition: $X[1]\,\ldots\,X[n]$ is a permutation of $\{1,\ldots,n\}$
       !!\gts Exit condition: array $X$ contains inverted permutation  \startnumbering
:Invert   SL        mm,n,3              !1! \step I1. Initialize. $m\gets n$.
          NEG       j,1                 !1! $j\gets-1$.
2H        LDO       i,X,mm              !N! \step I2. Next element. $i\gets X[m]$.
          PBN       i,5F                !N!\bad C\bad To I5 if $i<0$.
3H        STO       j,X,mm              !N! \step I3. Invert one. $X[m]\gets j$.
          SR        j,mm,3              !N! \mlsc 2:{$j\gets-m$.} % multi-line comment
          NEG       j,j                 !N!
          SL        mm,i,3              !N! $m\gets i$.
          LDO       i,X,mm              !N! $i\gets X[m]$.
4H        PBP       i,3B                !N!\bad C\bad \steq I4. End of cycle? To I3 if $i>0$.
          SET       i,j                 !C! Otherwise set $i\gets j$.
5H        NEG       i,i                 !N! \step I5. Store final value.
          STO       i,X,mm              !N! $X[m]\gets-i$. \newcommchar. % change the commchar
6H        SUB       mm,mm,8             .N. \step I6. Loop on $m$.
          PBP       mm,2B               .N.\bad 1\bad To I2 if $m>0$. \stopnumbering
* inspect memory locations of array X for the result
          TRAP      0,Halt,0
Main      IS        :Invert             .. \eop
..\endwAoA\bye
```

The last line of the input file ends the source with `\endwAoA`. So a second file with the analysis of the algorithm is needed; it is the file `1-3-3I_aoa.tex`:

```
In step~I3 each slot of the array~$X$ once receives a negative value and in step~I5 it is
filled with a positive number. Using Kirchhoff’s law the number of times step~I6 is executed is
equal to the number of times steps~I5 and~I2 are executed; that is steps~I2 and~I6 have count
$N$. Step~I5 is entered from I4 $C$~times, so I2 goes $N-C$~times to step~I5 and $C$~times
to~I3. And step~I3 goes $N$~times to step~I4, which must return $N-C$~times to~I3.

Of course, $N$ is the number of elements in the permutation and~$C$ is the number of its
cycles. The \mb{PB..}~instructions in lines~\pgmline{04} and~\pgmline{10} are based on the
assumption that in most cases $C\leq N/2$. An analysis of~$C$ shows that its average value is
the harmonic number~$H_n$. So the assumption is correct.

The program needs $4N\mems + (12N+5C+4)\oops$. The execution with the test data, the
permutation $(4 5)(2)(1 6 3)$, gives the statistic for \mb{Invert}: {\tt 78~instructions,
24~mems, 91~oops; 11~good guesses, 7~bad}. (The total run time is 96\oops\ as the \mb{TRAP}
instruction needs 5\oops.) As in this case $N=6$ and $C=3$ the above formula calculates
$(4\times6)\mems=24\mems$ and $(12\times6+5\times3+4)\oops=(72+15+4)\oops=91\oops$ in agreement
with the measured data.
```

And this shows the final output (with simulated headline and footline). I assume that in a TAOCP volume only the numbered lines appear.

Author: Udo Wermuth   Program: 1-3-3I.mms (V1, p. 177)

Program I (*Inverse in place*)

```
    n         GREG    6                         Number of elements in the permutation
    j         IS      $0                        Variables of the algorithm
    i         IS      $1
    mm        IS      $2                        mm = 8m
              LOC     Data_Segment
    X         GREG    @
              OCTA    0                         X[0] is not used
              OCTA    6,2,1,5,4,3               The data of Table 1.3.3–3
              LOC     #100
            > Inverse a permutation in place
            > Entry condition: X[1] ... X[n] is a permutation of {1,...,n}
            > Exit condition: array X contains inverted permutation
01  :Invert   SL      mm,n,3                 1  I1. Initialize.  m ← n.
02            NEG     j,1                    1  j ← −1.
03  2H        LDO     i,X,mm                 N  I2. Next element.  i ← X[m].
04            PBN     i,5F                   N  To I5 if i < 0.
05  3H        STO     j,X,mm                 N  I3. Invert one.  X[m] ← j.
06            SR      j,mm,3                 N  }
07            NEG     j,j                    N  } j ← −m.
08            SL      mm,i,3                 N  m ← i.
09            LDO     i,X,mm                 N  i ← X[m].
10  4H        PBP     i,3B                   N  I4. End of cycle?  To I3 if i > 0.
11            SET     i,j                    C  Otherwise set i ← j.
12  5H        NEG     i,i                    N  I5. Store final value.
13            STO     i,X,mm                 N  X[m] ← −i.
14  6H        SUB     mm,mm,8                N  I6. Loop on m.
15            PBP     mm,2B                  N  To I2 if m > 0.
    * inspect memory locations of array X for the result
              TRAP    0,Halt,0
    Main      IS      :Invert                        ▌
```

Analysis

In step I3 each slot of the array X once receives a negative value and in step I5 it is filled with a positive number. Using Kirchhoff's law the number of times step I6 is executed is equal to the number of times steps I5 and I2 are executed; that is steps I2 and I6 have count N. Step I5 is entered from I4 C times, so I2 goes $N - C$ times to step I5 and C times to I3. And step I3 goes N times to step I4, which must return $N - C$ times to I3.

Of course, N is the number of elements in the permutation and C is the number of its cycles. The `PB..` instructions in lines *04* and *10* are based on the assumption that in most cases $C \le N/2$. An analysis of C shows that its average value is the harmonic number H_n. So the assumption is correct.

The program needs $4N\mu + (12N + 5C + 4)\upsilon$. The execution with the test data, the permutation (45)(2)(163), gives the statistic for `Invert: 78 instructions, 24 mems, 91 oops; 11 good guesses, 7 bad`. (The total run time is 96υ as the `TRAP` instruction needs 5υ.) As in this case $N = 6$ and $C = 3$ the above formula calculates $(4 \times 6)\mu = 24\mu$ and $(12 \times 6 + 5 \times 3 + 4)\upsilon = (72 + 15 + 4)\upsilon = 91\upsilon$ in agreement with the measured data.

Date: 04 Aug 2014   **1**

Udo Wermuth