Reflections on \globaldefs in plain TEX

Udo Wermuth

Abstract

This article discusses a single integer parameter of the program TEX: It looks at the history of the invention of \globaldefs, describes the behavior of this internal parameter and tries to list useful applications. It also warns about constructions that might lead to faulty results. Moreover, it explains why one must do a careful verification of someone else's code if that code should be reused under a setting of \globaldefs.

1 Introduction

The core design model of TEX, the box/glue/penalty model, hasn't changed since its creation in April 1977 [9, p. 142]. And one of the model's realizations hasn't changed much either: After the debugging in March 1978, the code of the line-breaking algorithm was only once overhauled (June 1980) [9, ch. 3] and received in November 1982 an important fix in one of its computations [7, pp. 274–276]. But nevertheless the program TEX mainly grew because of the addition of new *internal parameters* and *primitives*, i.e., commands that are implemented in the program and aren't defined as macros. Luckily, we are able to follow how TEX evolved as TEX's author, Donald E. Knuth, decided to keep a detailed log file about all changes that he applied since March 10, 1978; see [5]. This log contains the descriptions of bug fixes as well as enhancements like *generalizations*. The above mentioned changes to the line-breaking algorithm are the entries #461 and #554.

Knuth stated that he wanted to create a typesetting language but early users pushed him to add more and more programming features [9, p. 648]. Obviously, his intention was to keep the language TEX concentrated on its application domain: typesetting texts.

A new parameter. One generalization is listed in the aforementioned log as entry #623, dated January 20, 1983: "Add a new \globaldefs feature." Knuth describes that a generalization often occurred when people presented him with new applications [7, p. 281]: "When I couldn't handle the new problem nicely with the existing TEX, I usually would end up changing the system." From other notes in the log file one sees that he was working on *The TEXbook* chapters 16 and 17 and on Appendix G; these are the chapters about typesetting math. The parameter \globaldefs has neither a direct influence on the typesetting of mathematics nor is it used in the text of *The TEXbook*. Moreover, the entry in the log doesn't carry a name of another person as is done with other entries (see, for example, entry #631). It seems that no one individual suggested the creation of this parameter.

I guess that the idea for this generalization was born during the work on entry #621, which is labeled "a cleanup for consistency or clarity". In [7, p. 245] this type of enhancement is described as: "Here I changed the rules of the language to make things easier to remember and/or more logical." The cleanup of January 19 touches TEX's procedure *prefixed_command* [4, §1211] which got, with #623, the code for the parameter \globaldefs in §1214.

Thus, I surmise the question "Why was the parameter \globaldefs added to TEX?" should receive an answer like "because this generalization was an ad hoc idea about a programming feature that could easily be implemented" instead of "because a detailed study showed that this programming feature lets TEX gain a lot of capabilities to solve certain problems more nicely."

Please note, I use "ad hoc" without implying that the implementation was spontaneous or unplanned. Knuth defined the parameter \globaldefs one day after #621. Thus he had time to think about it and to design its implementation.

The creation of \globaldefs, as documented in entry #623 of [5], didn't implement the functionality that this feature has today. On June 7, 1983, Knuth states in entry #710 that he made a cleanup for the rules of \globaldefs. One month later he had to apply a fix to this feature; see entry #748. It's a fix of type "forgotten function" which is described as [7, p. 245]: "Here I didn't remember to do everything I had intended, when I actually got around to writing a particular part of the code." Section 2 explains the impact of these two changes.

Not much coverage in the literature. *The TEXbook* [3] mentions \globaldefs in only three places. No exercise applies this parameter nor is it used in the format file plain.tex. A brief description appears between two blocks with syntax rules on page 275. Pages 206 and 215 indirectly explain fix #710; the impact of #748 is mentioned on page 238.

Well, *The TEXbook* cannot explain everything in detail but other books about TEX might fill the gap. Unfortunately, all books that I own either don't mention \globaldefs or they just repeat what is written in *The TEXbook* on page 275. At last I found on CTAN one that uses it: [1, p. 306] applied \globaldefs mainly to save keystrokes.

doi.org/10.47397/tb/44-1/tb136wermuth-globaldefs

So what are the use cases of \globaldefs? Under which conditions does the saving of keystrokes create a *real* benefit? And is it only used to save keystrokes or are there other useful applications?

Another place to search for use cases is the archive of *TUGboat* [10]. I found only one article by another author; the text describes how to combine TEX-formatted labels with PostScript files; see *TUGboat* **13**:3, page 332. Nowadays it is difficult to say why the parameter is used as the code to which \globaldefs is applied isn't completely known. But it's very likely that its use merely avoids the input of a handful of \globals. Again, \globaldefs is used to save keystrokes.

One can find \globaldefs in code on CTAN too. As an example, look at twimac.tex by Knuth [6]. It's a macro package to support the program TWILL [8] and contains \globaldefs four times. He used this program in 1985 to create the extremely useful mini-indexes for *TEX: The Program* [4].

Contents. The rest of this section briefly reviews the concepts known as "local" and "global". Section 2 describes what \globaldefs does including all special cases. Section 3 looks at the applications of a non-default setting for \globaldefs; it lists four use cases. Another non-default setting is analyzed in section 4. At its end I formulate a fifth use case with a *recommendation* for the use of \globaldefs. Section 5 discusses what can happen if \globaldefs with different settings are nested. At the end of this section I extend the recommendation of section 4. Section 6 briefly examines technical aspects inside of TEX, especially a possibility to save memory. The article ends with some personal remarks in section 7.

Local and global. It is well known that TEX obeys with assignments and macro definitions the group level in which it executes the command.

Example 1: Local assignment restored

```
\dimen0=10.01pt
{\dimen0=20.02pt A: \the\dimen0 }% a group
\quad B: \the\dimen0
```

TEX output

A: 20.02pt B: 10.01pt ▯

(The rectangle in the gutter — here at the end of the part "TEX output" — marks the end of the example.) TEX prints 20.02pt and then 10.01pt because the second assignment occurs inside a group. The new value is only *locally* known, i.e., only inside this group level. One must apply the prefix \global to change \dimen0 inside the group with a *global* effect, i.e., to keep the new value after the '}'. Then the code prints 20.02pt twice.

Example 2: Global assignment kept

```
\dimen0=10.01pt
{\global\dimen0=20.02pt A: \the\dimen0 }%
\quad B: \the\dimen0
```

TEX output

A: 20.02pt B: 20.02pt ▯

But, let's note that not all statements that appear as if they were an assignment are such to TEX. For example, \openout associates a stream number to a file name and accepts an equal sign in its syntax but isn't an assignment. It cannot be prefixed by \global. Well, it doesn't need to be prefixed as the association is by default global.

There are a few commands and quantities which TEX treats with global effect. For example, changing a box dimension, i.e., an assignment to \wd, \ht, or \dp of a box, is always global in the sense that the box dimension is changed for this box permanently. Only if the box number is restored at the end of a group the restored box has its former dimensions.

Some primitives that always act globally are simple commands without the form of an assignment. For example, a switch of the interaction mode from the default \errorstopmode to \batchmode is a global change. See page 277 of *The TEXbook* [3] for the syntactic rule ⟨global assignment⟩ that collects all the statements that act globally. One can use the prefix \global in front of \errorstopmode or \batchmode but it doesn't make a difference.

2 What does the parameter do?

The value of the integer parameter is simply changed by an assignment of the form \globaldefs = n, in which n is any valid TEX integer. But the parameter \globaldefs acts only in three different ways because TEX only checks if $n > 0$, $n = 0$, or $n < 0$.

- With $n > 0$ TEX starts to execute commands listed under ⟨simple assignment⟩ (see [3, p. 276]) and ⟨macro assignment⟩ (see p. 275) — in short: all assignments, arithmetic commands, and all control sequence definitions including \font and \read — as if the prefix \global was specified. Thus, TEX doesn't care if \global occurs in the code or not. It operates on these statements as if it saw the prefix.

Note that TEX must execute the code to assign the prefix; for example, a \def inside a \def doesn't become global during the definition. Similarly, the \relax equivalent of a \csname/\endcsname construct in a test or after \expandafter\show, etc., which is otherwise an undefined control sequence, does not become a global \relax as it isn't executed.

- With $n = 0$ \globaldefs is neutral or switched off; this is the default value of the TEX program.

Thus, a statement that's influenced by \global (except those that always act globally) has to receive the prefix \global to change a value not only inside the current group level; see example 2.

• With $n < 0$ \globaldefs switches the prefix \global off: \global has no impact on the following command and operates like \relax. But as a prefix it must still be used only with ⟨simple assignment⟩ and ⟨macro assignment⟩. For example, you cannot write \global\begingroup as \begingroup doesn't accept \global and therefore TEX raises an error.

The primitives \gdef and \xdef behave like \def and \edef, respectively; this was the cleanup in #710 of [5]. Thus, we can treat \gdef and \xdef as equivalent to \global\def and \global\edef.

The always-global commands listed as ⟨global assignment⟩ in [3] keep their global effect.

Get \globaldefs' value. As an internal parameter \globaldefs' value can be shown, printed, or assigned to an integer register or parameter. That is, \showthe\globaldefs shows the current value on the terminal; \number\globaldefs typesets its value (use it in math mode as the value can be negative), and, for example, \count9=\globaldefs stores its value in the count register number 9.

A special case. When TEX scans the tokens in the preamble of an \halign or \valign it collects them for the templates of the rows or columns, respectively. TEX processes a few tokens during this scan: (1) the alignment character, '&'; (2) the expand token, \span; (3) the parameter character, '#'; (4) the end-of-preamble tokens \cr and \crcr; and (5) the \tabskip token with its following glue specification. Everything else belongs to the templates.

This means that \global cannot be applied to assignments to \tabskip in the preamble as TEX puts this \global into the template. It also means that the \tabskip cannot be placed in a group to make the assignment local: Even in such a case the new \tabskip value in the preamble is extracted and used for the space inserted between the following columns or rows. But at the end of the alignment TEX forgets all non-global changes to \tabskip. The last change to \tabskip in the preamble never determines its value after the alignment except when the alignment starts under \globaldefs > 0. Then all assignments in the alignment are global; with fix #748 of [5], including the ones to \tabskip in the preamble, as probably anticipated by the users.

3 Use cases for \globaldefs=1

Two use cases for \globaldefs=1 were already mentioned. In section 1 we learned that it can be used to avoid the repetitive input of \global. Section 2, subsection "A special case", showed that it's required to make a \tabskip in a preamble global.

UC1: global \tabskip. As explained in section 2 a \tabskip assignment in the preamble cannot use \global. Inside the table entries the value can be globally modified: use \global\tabskip followed by a glue specification. It has no effect on the rôle of \tabskip in the preamble: The white space between columns or rows of the alignment isn't changed.

To be honest, I hesitate to call this a use case for \globaldefs=1 as it is quite extreme to apply this setting to have global assignments to \tabskip in the preamble. Without \globaldefs=1 just insert \noalign{\global\tabskip=\tabskip} after the preamble's \cr to make the preamble's last value of \tabskip global; no other statement is affected.

UC2: saving keystrokes. Sure, a \globaldefs=1 followed by at least four statements that should otherwise be prefixed by \global and a \globaldefs=0 can save at least 2 keystrokes as the prefixes aren't required in the input: 2×13 keystrokes vs. 4×7. Let's write it explicitly although I use just two assignments not four; you see later why.

Example 3: Saving keystrokes with \globaldefs=1

```
\globaldefs=1 \count9=123 \dimen9=123.45pt
\globaldefs=0
```

One should observe that the second assignment to \globaldefs is a global assignment too. But this assignment can be avoided as all statements are global except the \globaldefs=1; so, inside a group, TEX restores only the value of \globaldefs. Now one saves a keystroke if there are just two assignments.

Example 3 continued: Simpler input

```
{\globaldefs=1 \count9=123 \dimen9=123.45pt }
```

One shouldn't do this without need. (If you have to create the group you don't save keystrokes.) Check [1] and the first occurrence in [6] and you see that the authors are *forced* to open a group because of a catcode change; it might be hidden in an \obeylines. The setting \globaldefs=1 allows to enter the code that belongs to the outer level of the macro package as if it were not in a group. Often in such a group, only \def is used; thus, one saves only a 'g' and not a '\global' in the statements.

But keystroke savings aren't the only point. If the catcode change isn't needed anymore one can easily remove the group and the \globaldefs=1. Then the code integrates well with the rest. (See, for example, file ctwimac.tex in the directory of twimac.tex; the first \globaldefs=1 of the latter isn't used in the former anymore.)

The application of \globaldefs=1 in front of a loop is similar. Here is an example using the prefix \global (such a case appears in [6]):

Example 4: Several \global in a loop

```
\newcount\nn \newcount\maxnn \maxnn=200
\global\nn=100
\loop \global\count\nn=0 \global\dimen\nn=0pt
   \global\skip\nn=0pt \global\muskip\nn=0mu
   \global\toks\nn={}%
\ifnum\nn<\maxnn \global\advance\nn by 1 \repeat
```

To limit the scope of \globaldefs=1 a group encloses the whole \loop/\repeat construction.

Example 4 continued: Loop with \globaldefs=1

```
\newcount\nn \newcount\maxnn \maxnn=200
{\globaldefs=1 \nn=100
 \loop \count\nn=0 \dimen\nn=0pt \skip\nn=0pt
    \muskip\nn=0mu \toks\nn={}%
 \ifnum\nn<\maxnn \advance\nn by 1 \repeat}
```

One advantage is that the material between \loop and \repeat is more compact and might be easier comprehended. A side effect in this scenario is that the \body in the macro \loop becomes global too.

UC3: global expand. Here \globaldefs=1 is applied to a token register, a macro, or a TEX file that contains definitions and assignments. When TEX expands the register with \the, executes the macro, or inputs the file all statements receive \global. Such a construction can be used, for example, inside the output routine to make the data in the register or the macro available to the outer level. (For a better but more advanced example, read "Processing by TEX" in [8, p. 7] together with [6], if you can understand high level descriptions of output routines.)

Example 5: Apply \global to a collection

Here is a very simple example of how statements that were collected in a token register are executed inside a group with \globaldefs=1. Assignments to \hsize and \vsize via \setsize and macros to get their product are placed into the register. I omit all error checking.

TEX input

```
{\catcode`\_=11 \newcount\area_sqmm
 \newtoks\area_cmds % the collection
  \global\area_cmds={\area_sqmm=0 }% initialize
 \gdef\set_area(#1){% #1: dimen w/o unit mm
  \ifnum\area_sqmm=0 \area_sqmm=#1\relax
  \else \multiply\area_sqmm by #1\fi}
 \gdef\setsize#1#2mm{% #1: h/v; #2: dimen w/o mm
  \area_cmds=\expandafter{\the\area_cmds
   \csname#1size\endcsname=#2mm\set_area(#2)}}
 \gdef\prtarea{\message{Area: \the\area_sqmm
   sqmm.}\area_cmds={\area_sqmm=0 }}% reset
}\setsize h176mm\setsize v250mm% fill collection
{\catcode`\_=11 \globaldefs=1 \the\area_cmds}
\prtarea % shows 176mmx250mm = 44000sqmm
```

Without the \globaldefs the \hsize and \vsize settings aren't global and \area_sqmm would be zero in the message. We can add \global to the definitions, but then the collection cannot be executed without \globaldefs=-1 for a local application.

UC4: keep code and output in sync. Journals like *TUGboat* publish TEX input and also its typeset output. To stay in sync *TUGboat* suggests to use the following construction; here demonstrated with additional code from me. The first example of this article was more or less coded as

```
\verbatim[\inputfromfile{example1.tex}]
\endverbatim\exout \input example1.tex \exend
```

where the macro \exout outputs "TEX output" in boldface and opens a group that ends in \exend. The first line reads example1.tex and typesets its contents verbatim. Line 2 executes the code in this file inside the \exout/\exend group. In this group and in front of the \input a register value from a previous example might be entered or a parameter implied from the current topic like \parfillskip is set. Example 1 doesn't need anything of this kind.

Here is an example in which the environment of \exout might start with \globaldefs=1. In this example a file is opened for writing. Just one line is written to this file and this line contains a macro that was defined inside the example. TEX executes the \write delayed, i.e., the file gets the line with the next \shipout; see [3, pp. 226–227].

Example 6: Code and its output

First, we look at the contents of the file example6.tex.

TEX input

```
\toks9={Hello world!}\openout5=ex6outfile.tex
\def\textforex6{\number\pageno: \the\toks9 }%
\write5{\textforex6}\closeout5
```

This code cannot be executed inside a group. The \write waits for the next page break and when it occurs TEX expands the token list of this \write and stumbles over \textforex because outside of the group it's undefined. One could add \immediate to \openout, \write, and \closeout; or use \gdef for \textforex and \global in front of the \toks assignment. This destroys the example if the author wants to keep the code as simple as possible.

We don't want to change anything in the input file example6.tex that contains the code of the example. And we want to keep the code of \exout (and \exend). Thus we must make the assignment and the definition global, i.e., we must use \globaldefs.

Example 6 continued: Code used for execution

```
\exout\globaldefs=1 \input example6.tex \exend
```

We are only allowed to do this as we know the code in the file. It doesn't work always; see example 11.

4 Use cases for \globaldefs=-1

Of course, the use of \globaldefs=-1 can be easily replaced if the code to which it should be applied doesn't contain other settings of \globaldefs. As \relax passes prefixes on to the next token we can code \let\global=\relax and the code \long \global\def still generates a \long\def.

But there's a difference between an deactivated \global by \globaldefs=-1 and the above \let. If you scan tokens one by one and compare them against \relax then the new \global executes a wrong branch of the test. The solution consists of a replacement text that uniquely identifies \global as well as \gdef and \xdef.

Example 7: Avoiding \globaldefs=-1

```
\let\CPglobal=\global \let\CPgdef=\gdef
   \let\CPxdef=\xdef
% use for them unique replacement texts
\def\global{\relax\relax}\def\gdef{\relax\def}%
   \def\xdef{\relax\edef}%
... % code with inactive \global, \gdef, \xdef
% restore \global, \gdef, \xdef with the copies
\let\global=\CPglobal \let\gdef=\CPgdef
   \let\xdef=\CPxdef
```

Use cases from section 3. Obviously, one cannot save keystrokes with the setting \globaldefs=-1 as only existing \global tokens are affected. So there are no use cases that correspond to UC1 or UC2. UC3 can be turned into a "local expand" variant if the collection contains the prefix \global. In the scenario of UC4 \globaldefs=-1 can only be used if the code doesn't contain *necessary* \global. Thus, it helps in some sense only for badly written code.

Example 8: Case where \global is necessary
Is testscript.sh's first line a so-called *shebang line*, signaling that it is a *Bourne shell script*?

TEX input

```
\def\uncatcodespecials{% see The TeXbook, p. 380
   \def\do##1{\catcode`##1=12 }\dospecials}
\edef\shebangline{\string#!/bin/sh}%Bourne shell
% the code with \read in a group
\newread\infile \openin\infile=testscript.sh
\def\readin{{\uncatcodespecials \endlinechar=-1
 \global\read\infile to \lineofinfile}}\readin
\ifx\lineofinfile\shebangline
   \message{Bourne shell}\fi \closein\infile
```

If the \global\read isn't global because of an active \globaldefs=-1 then the \ifx doesn't produce a reliable result as \lineofinfile is either undefined or contains data from another assignment.

Reuse unknown code. One might think a good use case for the parameter \globaldefs with a negative number is to limit the effect of a file that contains macros. But of course, \globaldefs=-1 must not eliminate a necessary \global. One must be careful if it should be applied to reuse code.

For example, assume that calmacros.tex contains macros for calendrical computations like the Day of Repentance and Prayer for a given year. (It's celebrated eleven days before the first Sunday of Advent, so its month is November.) Another file consists of macros that belong to the same domain; it's very likely that the files share, for example, register names. Assume that in the second file the Day of Repentance and Prayer is required and that the package calmacros.tex provides a macro with the name \CalcRepPrayDay to compute that day.

Example 9: Avoiding global changes with \input

```
\newcount\repprayday {\globaldefs=-1
\input calmacros \repprayday=\CalcRepPrayDay2023
\globaldefs=0 \global\repprayday=\repprayday}
```

The main file has then access to \repprayday but all macro names, register names etc. of calmacros.tex are gone when the group ends. Nevertheless, example 8 warns us: the result might be wrong!

Even if the code doesn't throw an error one cannot trust the result. In calmacros.tex computations might occur inside a group and the final result made available to the outside only through an assignment prefixed by \global. This doesn't happen if \globaldefs=-1. Thus, the result value is restored when the group ends; the result becomes a "random" value.

One must verify that a file with macros that one wants to reuse in a group with \globaldefs=-1 contains at most unnecessary \globals in the code paths that are called.

Another more concrete example from this text:

Example 10: Which number is output after "A:"?

```
\globaldefs=-1 {\input example3 }%
A: $\number\globaldefs$
```

The result is either 0 or −1; it depends if the black box example3.tex refers to the version with the two \globaldefs or to the one with the group. Moreover, we also cannot answer the question if we use \globaldefs=1 instead of \globaldefs=-1.

What has been found out is known [2] but it should become common knowledge.

UC5: reuse known code. To eliminate the effect of \global one might execute code inside a group with the setting \globaldefs=-1. But one must verify that it doesn't deactivate a necessary \global.

In general follow this recommendation: Never apply \globaldefs $\neq 0$ to code that you don't completely know or fully understand in all its details because you might get a random result.

5 Nested \globaldefs

A programmer might ask if code can be written in such a way that it protects itself against bad results if someone reuses this code inside a group with a non-zero \globaldefs. Of course, there is a simple solution: Start your code with \globaldefs=0. This cancels a \globaldefs=-1 and with \globaldefs=1 only the \globaldefs=0 becomes global.

Let's assume that we want to execute all statements under the setting of \globaldefs except if this would cause an error. Then the protection with \globaldefs=0 is a valid solution for \globaldefs=-1 if the reused code has only necessary \globals. For \globaldefs=1 it isn't a solution.

Protection against \globaldefs=1. Let's state the goal precisely. A programmer wants to protect code so that it still executes correctly if someone takes this code and places it inside a group starting with {\globaldefs=1. To simplify the discussion }\globaldefs=0 \global\globaldefs=0 is used at the end of the group; thus a change of \globaldefs' value inside the group isn't important. The solution to start the code with \globaldefs=0 is not considered to be valid. Only the code that *must be protected* because otherwise the original code does something wrong *should be protected*. Everything else should be executed using \globaldefs=1.

I admit it sounds like an unrealistic scenario. But we might learn from it.

Example 11: Try to protect against \globaldefs=1
The following code should be protected to allow execution in a group that sets \globaldefs=1. It's artificial code to keep the size of the often repeated example small.

TEX input

```
\dimen9=1000pt \count9=0
{{\catcode`\e=3 ex^2e}%
 \global\advance\count9 by 2 }%
\divide\dimen9 by\count9 \count8=\count9
```

It's clear why this code cannot be executed under a \globaldefs=1 without throwing an error. The catcode change becomes globally active and thus an undefined control sequence \advanc is reported later.

A setting \globaldefs=-1 or 0 must be placed in front of the catcode change to keep it local in its group. Let's apply −1.

Example 11 continued: A failed attempt

```
\dimen9=1000pt \count9=0
{{\globaldefs=-1 \catcode`\e=3 ex^2e}%
 \global\advance\count9 by 2 }%
\divide\dimen9 by\count9 \count8=\count9
```

This code throws an error too. As TEX executes the new assignment globally, \globaldefs=-1 survives the end of the inner group and deactivates the \global in front of the \advance. Thus, TEX restores \count9 at the next }, finds a division by 0 in the last line, and reports "Arithmetic overflow".

Thus, the code for the protection needs an improvement. The \global\advance must stay global. We can put a \globaldefs=0 in front of that statement. But as another group ends, the next line isn't globally executed. Should we use \globaldefs=0 instead of \globaldefs=-1 in the inner group?

Example 11 continued: A successful attempt?

```
\dimen9=1000pt \count9=0
{{\globaldefs=0 \catcode`\e=3 ex^2e}%
 \global\advance\count9 by 2 }%
\divide\dimen9 by\count9 \count8=\count9
```

This code runs without generating an error and it protects the code. But the last statements are still protected although that shouldn't happen. In essence it's more or less equivalent to the initial solution which was rejected above because of this effect.

We must find another solution: Let's keep the \globaldefs=-1 in the inner group to signal that the catcode change must be local; but its influence must be stopped when this group ends. As the code must work with initial values 0 or 1 for \globaldefs its value should be reset after the other group.

Example 11 continued: The final attempt

```
\dimen9=1000pt \count9=0
\edef\SAVEglobaldefs{\number\globaldefs}%
{{\globaldefs=-1 \catcode`\e=3 ex^2e}%
 \globaldefs=0 \global\advance\count9 by 2 }%
\globaldefs=\SAVEglobaldefs
\divide\dimen9 by\count9 \count8=\count9
```

The current value is captured in a macro and then restored at the correct place. This does what was requested above. But, in this version, two of the original four lines contain additional code and the other two lines are now accompanied by new lines. The amount of code is nearly doubled.

Extension of the recommendation. I don't suggest that programmers protect their code against an execution with a non-zero setting of \globaldefs; at least not with a \globaldefs=0 at the start of the file. But we should extend the recommendation stated in UC5: You are responsible to protect the code that you reuse from generating erroneous output because of your setting of \globaldefs. And you are responsible to ensure that \globaldefs' value outside of the group that you opened is restored if that is required.

6 Technical advantages

Up to now we looked at the parameter \globaldefs to see what advantages its application has in the

input of a user. But, of course, there are technical payoffs too. The first is obvious: An input file needs fewer bytes, i.e., it needs less storage space and might load faster, if at least two `\global` are saved for each `\globaldefs=1`. Is there anything more about the introduction of `\globaldefs`? Can it save memory? Knuth worked hard in the late 1970s and early 1980s to get TEX into the then-available memory space of the then-available computers.

Let's do a little experiment with the two versions of example 4: Execute the two code snippets with `\tracingstats=1` in front of the code and with an `\end` after it. Next, compare the statistics at the end of the `log` files. The `\globaldefs` variant saves twelve memory words on my system.

The effect seen in example 4 doesn't occur always. For other scenarios the number of memory words doesn't change. For example, create two files. Once "`\tracingstats=1 \global\count3=4 \end`" and the second replaces the global assignment by "`\globaldefs=1 \count3=4`". Except for the value of *buffer size* the statistics are identical. The result is the same if `\count` is replaced by `\dimen`, `\skip`, `\muskip`, or `\toks` with appropriate right hand sides. Even a block with all five register types has the same number of memory words.

The opportunities to apply `\globaldefs` are quite rare. Thus, we cannot hope that it helps to save any significant amount of memory in a project.

7 Personal remarks

I confess that I haven't used `\globaldefs` often in my TEX projects. I used it when I was forced to do so in an unusual macro project (*TUGboat* **43**:1, p. 63) to protect the code from the problems of section 5. (I suggest on p. 72 to use `\let\globaldefs=\undefined` as the protection cannot be perfect.) There are other TEX primitives that I seldom use, for example, `\valign`; but I thought I had used this primitive more often. It is part of the typographic language that Knuth wants to put into the foreground. I assumed Knuth had a good reason to add `\globaldefs` to the program TEX. At least its addition makes the language more complicated to learn.

I don't deny that UC2 is useful: Saving keystrokes is a nice feature and example 4 looks much better with `\globaldefs=1`. But its use could be easily avoided here as well as in UC3. Only UC4 needs `\globaldefs` if code shouldn't be changed. But an alternative with changed code isn't hard to create: Use `sed` (or similar) to insert `\global` automatically into the code and write a new file that's used after `\exout`. I'm convinced that UC5 is of limited use. And I would never apply it to `\newread` as in `thumbpdf.sty` on CTAN; it isn't my programming style to use such tricks. So I asked: Has a TEX without `\globaldefs` problems that must be solved with this parameter? Is it important to save keystrokes?

The reader might say, "Wasn't the introduction of `\xdef` in #370 of [5] similar, as it just saves one `\global`?" No, I think it does more: The language becomes easier to learn with the pairs `\def`/`\gdef` and `\edef`/`\xdef`. And Knuth was asked to implement this change.

I wrote this article to understand `\globaldefs` better. I learned: All listed use cases have other solutions without complicated tricks. Moreover, nesting `\globaldefs` can create problems as described in section 5. And even with the results of section 6, I wonder why `\globaldefs` became a part of TEX.

References

[1] Paul W. Abrahams, Kathryn A. Hargreaves, Karl Berry, *TEX for the Impatient*, 2003. `ctan.org/tex-archive/info/impatient/book.pdf`

[2] David Carlisle, comment on `tex.sx`, 2022-06-30. `tex.stackexchange.com/questions/649425/is-it-safe-to-use-globaldefs-for-setting-global-pgfkey-value-pairs/649437#comment1618526_649425`

[3] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[4] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[5] Donald E. Knuth, "The Errors of TEX", *Software — Practice and Experience* **19** (1989), 607–685; reprinted as Chapters 10 and 11 in [7], 243–339. The log, i.e., Chapter 11, is still updated: `ctan.org/tex-archive/systems/knuth/dist/errata/errorlog.tex`

[6] Donald E. Knuth, `twimac.tex`. `ctan.org/systems/knuth/local/lib/twimac.tex`

[7] Donald E. Knuth, *Literate Programming*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 27, 1992.

[8] Donald E. Knuth, "Mini-Indexes for Literate Programs", *Software — Concepts and Tools* **15** (1994), 2–11; reprinted as Chapter 11 in [9], 225–245.

[9] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999.

[10] *TUGboat*, archive of all publicly available articles. `tug.org/TUGboat/contents.html`

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`