4 Conclusion

We have presented a new approach to adapt the page layout of the document styles that are part of the standard LaTeX distributions to the dimensions of A4 paper. The width of marginal notes has been taken into account and a means to get wider marginal notes at the cost of shorter lines in the main body of the text has been provided.

References

[1] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst.* SDU Uitgeverij ('s-Gravenhage, 1988). A Dutch book on layout design and typography.

◇ Nico Poppelier
TEXnique
Washingtondreef 153
3564 KD Utrecht
Poppelier@hutruu53.bitnet

◇ Johannes Braams
PTT Research Neher Laboratories
P.O. Box 421
2260 AK Leidschendam
JL_Braams@pttrnl.nl

Some Macros to Draw Crosswords[*]

B Hamilton Kelly[†]

Abstract

The crossword environment is intended to be used to typeset crossword puzzles for use in newsletters, etc.

Contents

List of Figures

[*]This file is v2.7, dated 16 Oct 89

[†]Especial thanks to my colleague Niel Kempson for many helpful suggestions, and to Frank Mittelbach of the Johannes Gutenberg University of Mainz, who saved me two pages of code!

1 Introduction

As a small diversion from the statistics of computer availability, lists of new software, and the like, Computer Centre Newsletters often include a crossword for the amusement of their readers.[1]

The macros presented in this document provide a LaTeX method of typesetting these, and also assist the composer to ensure that the "grid" all goes together correctly. The grid generated is the more usual form, with black squares separating the "lights" which receive the answers to the clues. Work is in hand to be able to handle the *Mephisto/Azed* type of grid, in which only thicker grid lines separate the lights.

A sample crossword appears as Figure 1; I've left the grid blank for those who want some intellectual exercise: those who don't can cheat by reading the source listing at the end of this article!

The whole crossword, including the `\clue` commands (*q.v.*), is bracketed within the crossword environment. This requires that the user specifies two parameters:

⟨*gridsize*⟩ This is a number which specifies the columns (and rows) in the square grid.

⟨*visible*⟩ This controls whether the answers are to be "filled in"; obviously of no use for publication, but useful whilst composing the crossword. If the parameter provided is the letter '`Y`', then the answers will be typeset; if '`N`' then the lights will be left blank. Any other value[2] provided for this parameter will cause LaTeX to input a yes/no answer by interaction with the user.

An analogous environment is provided especially for typesetting a smaller version of a grid showing, for example, "Last Month's Solution". In this of course, the answers *always* appear, and the clues are not printed. Again it takes two parameters:

⟨*gridsize*⟩ As before, this specifies the number of squares in each axis.

⟨*header_text*⟩ Some text which will be set (in **bold**) above the completed grid.

Figure 2 shows an example of the crossword* environment.

Within the body of these environments appear a succession of `\clue` commands; each of these takes a total of seven (!) parameters:

⟨*clue_number*⟩ The number of the light on the grid, for example `{17}`. See 1.1 below for details of how more complex specifications may be given for multiple lights.

⟨*Across/Down*⟩ This parameter *must* be either the letter '`A`' or '`D`', in upper-case.

⟨*col_number*⟩ The x-coordinate of the first square of the light. The left-most column of the grid is numbered 1.

⟨*row_number*⟩ The y-coordinate of the first square of the light. The top-most row is numbered 1.

⟨*answer*⟩ The answer to the clue (or that part of it which appears in the light numbered ⟨*clue_number*⟩). This must be a string of upper-case letters *only*; no spaces, punctuation, hyphens, etc.

⟨*text*⟩ The text of the clue itself. If you want to use any LaTeX macros in this text, such as `\dots`, each such macro must be preceded by `\noexpand`. This includes such macros as `\&`, to produce an explicit ampersand.

⟨*help*⟩ Anything to appear after the text, in parentheses; this will most usually be used for giving the length of the answer, such as "7" or "2,6,3-3". Also used when the text of the clue is associated with another light when this parameter may say something like "see 14d".

[1] That at RMCS has a bottle of wine as a prize!

[2] The lower-case letters '`y`' and '`n`' are also recognized

1.1 How to Specify Clue Numbers

Sometimes the solution to one clue is split amongst a number of "lights". To cover this eventuality, provide a `\clue` for *each* of the lights involved, with the solution to that light *alone* given as ⟨*answer*⟩. All except the `\clue` corresponding to the first light of the solution should have a null ⟨*text*⟩, and the ⟨*help*⟩ parameter should be something like "see 7d".
If this final parameter is totally empty, no corresponding clue number is printed: this facility would be used when the current clue is the next consecutive light, when it is usual to omit any further reference to the clue number.

The `\clue` for the first light of the solution should provide the *entire* clue as its ⟨*text*⟩, and the ⟨*help*⟩ should say something like "7,3-3". The ⟨*clue_number*⟩ field should consist of the number of that light, followed immediately by the text required to describe the other lights, separated from it by some non-digit character, for example, a space.

For example, suppose the clue "Bill's desired outcome?", has the solution 'ACT OF PARLIAMENT' which is to go into lights 9d and 13a. Then[3]

```
\clue{13}{A}{5}{1}{PARLIAMENT}{}{see 9d}
\clue{9 {\noexpand\rm\&} 13a}{D}{1}{10}{ACTOF}%
     {Bill's desired outcome?}{3,2,10}
```

will produce

13 (see 9d)

amongst the ACROSS clues, and

9 & **13a** Bill's desired outcome? (3,2,10)

amongst the DOWN clues

2 Definition of the Macros

As always, we start by identifying this version of the style file.

```
\typeout{Style option: 'crossword' \fileversion\space\space
          <\filedate> (BHK)}
```

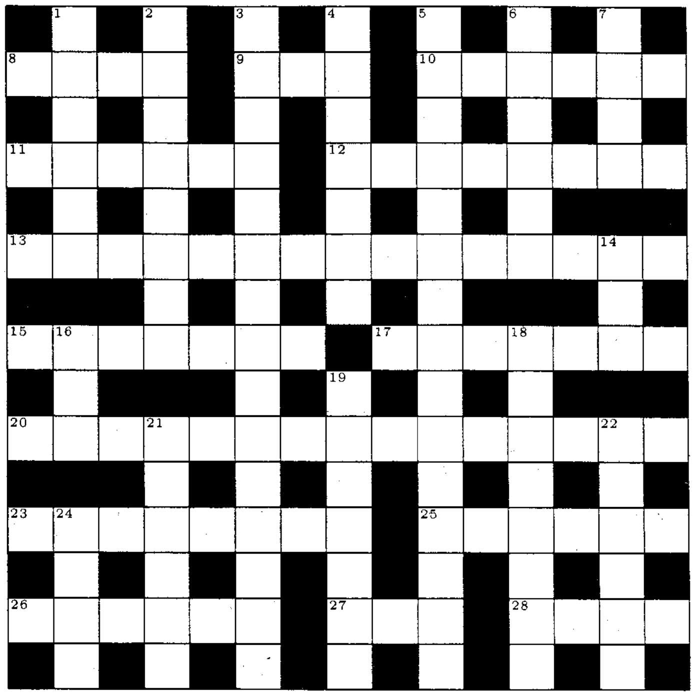
`\ninept` `\@listi` We define a new font size to ensure clues are set at 9pt, no matter what style size option is in effect. This command also defines suitable parameters for list environments set in this size of type.

```
\def\ninept{\@setsize\ninept{11pt}\ixpt\@ixpt
  \abovedisplayskip 8.5pt plus 3pt minus 4pt
  \belowdisplayskip \abovedisplayskip
  \abovedisplayshortskip \z@ plus2pt
  \belowdisplayshortskip 4pt plus2pt minus 2pt
  \def\@listi{\itemsep 0pt
    \parsep \z@ plus 1pt
    \topsep 4pt plus 2pt minus 2pt
}}
```

2.1 Counters and Lengths

`\ifnumberit` `\numberittrue` `\numberitfalse` The crossword environment draws a grid (with black and white squares); each "light" into which a clue's answer is to be written has to be numbered, and this number will be typeset (using `\tiny`) in the top-left corner of the first square of the light.

[3] note the `\noexpand` before the `\rm` for the `\&`



ACROSS

8 Points a thousand tested for witchcraft. (4)

9 Gourmet's triumphant cry on finding middle-cut Pacific salmon! (3)

10 One hundred stride backwards across a Pole. (3-3)

11 Fifties' jazz record about Eastern childs' play. (2-4)

12 Timetable created by editor in synagogue of Spain. (8)

13 The dialect a girl mixed up tangle around symbolic diagram used by maritime student! (15)

15 Wire fastening bent road narrowly. (7)

17 Hammerhead consumes German company and casts a shade. (7)

20 Strange cel alien chops to make a figure with odd sides. (7,8)

23 Sounds like a hoarse editor came to in total! (8)

25 Assert without proof everyone, for example, English. (6)

26 Flourished examination of flowers. (6)

27 Floor covering discards fuming sulphuric acid and returns to nothing. (3)

28 "Latin for a candle" to be silent note about aircraftman? (4)

DOWN

1 Water rush noise disturbs show so! (6)

2 Mischievous child with cloth measure hesitates to assemble rotor. (8)

3 Mercifully inclined to pass round ten? Nay, about short blower! (15)

4 Sort ion? An isotron gives it a new twist! (7)

5 Wildly and boisterously rearrange Billy May Third, roughly. (15)

6 Satirical book or film—give odds about revision of "Dune"? (4-2)

7 & **24** Premier took in a Lord Lieutenant and all played an old game in London street. (4-4)

14 Work expended in power games? (3)

16 A church circle (or part of one). (3)

18 Encircle hindrances under hair in long curls. (8)

19 Boss over otorhinolaryngology department undergraduate. (7)

21 Old dovecote in Parisian museum. (6)

22 Like ornamental fabric, for example, that's in bequest. (6)

24 (See **7**)

Figure 1: A Sample Crossword

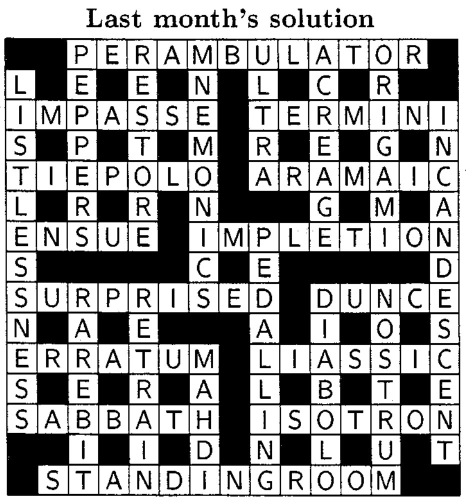


Figure 2: An example of the crossword* environment

This style option also provides the crossword* environment, which is intended to be used to produce "last month's solution" in a smaller grid. There is insufficient room for clue numbers to appear on the grid in this mode, so `\ifnumberit` is used to indicate whether the numbers should be set.

```
\newif\ifnumberit
```

`\gr@dsize` `\p@csize`

The counter `\gr@dsize` is used to hold the width of the grid, as the number of squares in each direction.

To prevent too much run-time arithmetic, the counter `\p@csize` is set to be one count higher than `\gr@dsize`.

```
\newcount{\gr@dsize}
\newcount{\p@csize}
```

`\Down` `\Across`

As we move around the grid, determining whether squares are black or white, we utilize the counters `\Across` and `\Down` to keep track of our location.

```
\newcount{\Down}
\newcount{\Across}
```

2.2 Reading and Writing the Clues

`\tf@acr` `\tf@dwn` `\OpenClueFiles`

Whilst we are determining the appearance of the grid, we copy the text of each of the clues to an auxiliary file, so that the latter may later be read back to generate the clues themselves after the grid has been printed.

This macro opens a new file, with file extension `.acr`, and puts into it the commands necessary to typeset the Across clues. It also opens a `.dwn` file, which is similarly filled with the Down clues.

These files are created in the same manner as table-of-contents (`.toc`) files, etc; thus LaTeX will create file "handles" with names `\tf@acr` and `\tf@dwn`. However, that would ordinarily attempt to *read* the given file first, and also might defer the actual opening; therefore, we start a new group in which we redefine LaTeX's `@input` command and TeX's `\openout` primitive.

```
\def\OpenClueFiles{{\let\OpenOut=\openout
  \def\openout{\immediate\OpenOut}%
  \let\@input=\@gobble
```

```
    \@starttoc{acr}
    \@starttoc{dwn}}
```

Here's the preliminary material that gets inserted into the `.acr` file.

```
    \@writefile{acr}{\string\begin{minipage}[t]{70mm}}
    \@writefile{acr}{ \string\centerline{\string\bf\ ACROSS}}
    \@writefile{acr}{ \string\sloppy}
    \@writefile{acr}{ \string\ninept}
    \@writefile{acr}{ \string\begin{ClueList}}
```

Whilst something similar goes into the `.dwn` file.

```
    \@writefile{dwn}{\string\begin{minipage}[t]{70mm}}
    \@writefile{dwn}{ \string\centerline{\string\bf\ DOWN}}
    \@writefile{dwn}{ \string\sloppy}
    \@writefile{dwn}{ \string\ninept}
    \@writefile{dwn}{ \string\begin{ClueList}}}
```

`\CloseClueFiles` After the grid has been printed, we can close the "clues" files; these will later be read back in (by the `\endcrossword` command) to set the text of the clues below the grid. Before closing, we insert the material that completes the two ClueList environments; firstly across...

```
  \def\CloseClueFiles{%
    \@writefile{acr}{ \string\end{ClueList}}
    \@writefile{acr}{\string\end{minipage}}
```

Then for the down clues.

```
    \@writefile{dwn}{ \string\end{ClueList}}
    \@writefile{dwn}{\string\end{minipage}}
```

Now we can close those files, and make them "invisible" if someone tries to write to them.

```
    \immediate\closeout\tf@acr \let\tf@acr=\relax
    \immediate\closeout\tf@dwn \let\tf@dwn=\relax
    \endgraf
  }
```

2.3 Tabulating the Clues

The auxiliary files contain the texts of the clues, each given as an `\item` for the ClueList environment. This is similar to a description list, except that overlong labels run on into the text rather than sticking out to the left.

`\ClueList` `\ClueListLabel` This sets up the ClueList environment, and defines the appearance of the label.

```
  \def\ClueListlabel#1{\hspace\labelsep {\bf #1}\hss}
  \def\ClueList{\list{}{\labelwidth\leftmargin
    \advance \labelwidth by -\labelsep
    \let\makelabel\ClueListlabel}}
  \let\endClueList\endlist
```

`\PrintClues` The following macro reads in the two files (of Across and Down clues), and sets them alongside each other, separated by a vertical rule. Clues are set in the style of the ClueList environment.

```
  \def\PrintClues{%
    \centerline{%
      \begin{tabular}{ c | c }
        \@input{\jobname.acr}
        &
      \@input{\jobname.dwn}
    \end{tabular}
  }\endgraf
}
```

3 Creating the Grid

The remaining commands are concerned with creating (and, optionally, populating) the crossword grid

`\crossword` The crossword environment takes two parameters: *viz.* the size of the matrix, and the indication of whether the grid is to include the answers. (If the latter is omitted, LaTeX will request it interactively.)

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

We start off with a `\vtop` box and a group to hold everything within the environment, so as to ensure that user-entered text remains with the crossword.

```
  \endgraf\leavevmode
  \vtop\bgroup
```

The crossword environment uses the full-size grid, and has the lights numbered. Furthermore it doesn't have any heading to output (see the crossword* environment).

```
    \unitlength 6mm\numberittrue
    \def\Header{}%
```

We now open the auxiliary files into which the clues are written, and determine (interactively if necessary) whether the answers are to be written into the grid.

```
    \OpenClueFiles
    \TestAnswers{#2}%
```

Finally, we generate the necessary macros to describe the grid as being entirely filled with black squares; for each square, a macro whose name is of the form `\RiCi`, `\RxiiCviii`, *etc.* is created. As the `\clue` commands are read in, these will be redefined to produce the correct appearance when the macros are later expanded.

```
    \SetUpGrid{#1}}
```

`\endcrossword` When all the clues have been processed, we can invoke `\FinishGrid` to draw the grid. The `\FinishGrid` and `\PrintClues` commands draw the grid and tabulate the clues, respectively. By enclosing them in a vertical mode list, we ensure that they remain stuck together on one page!

The crossword environment defines `\Header` to be empty, but the user may give it an explicit definition within the environment; if so, we'll print it just above the grid itself.

```
\def\endcrossword{\endgraf
    \centerline{\Header}%
    \hbox{\FinishGrid}%
```

We can now finish off the auxiliary files and then read them back in to set the text of the clues below the grid.

Finally, we complete the group and the `\vtop` box.

```
    \CloseClueFiles
    \hbox{\PrintClues}%
  \egroup
}
```

`\crossword*` `\endcrossword*` The crossword* environment doesn't need a second parameter to control printing of answers, because it **always** populates the grid with the answers. Instead, its second parameter provides the text to appear above the printed grid. Its actions are as for the crossword environment except that

- It prints the descriptive text above the grid.
- It **always** outputs the answers, without numbers.
- It draws them in a smaller box.
- It doesn't output the clues (it doesn't even open any auxiliary files!)

```
\expandafter\def\csname crossword*\endcsname#1#2{%
  \unitlength 4mm\numberitfalse
  \endgraf\leavevmode
  \vtop\bgroup
    \def\Header{{\bf\strut #2}}%
    \def\answer{Y}%
    \let\tf@dwn=\relax \let\tf@acr=\relax
    \SetUpGrid{#1}}

\expandafter\def\csname endcrossword*\endcsname{\endgraf
    \centerline{\Header}%
    \hbox{\FinishGrid}%
  \egroup
}
```

3.1 Macros used when Populating the Grid

`\laterletter` The following macro calls this to "place" the letters of a solution by defining a macro unique to this square.

```
\def\laterletter#1{\setsquare{\lettersquare{#1}}}
```

`\nextletter` `\nextlet` To determine how much space is required for the light corresponding to an answer, we need to cycle through each of the characters of the answer individually; this macro is called with two parameters — the first indicates the current setting direction (and thus accesses one of the counters `\Across` or `\Down`), whilst the second consists of the characters forming the answer followed by the string `\@nil`. When it is called, this "second" parameter is *not* enclosed in braces, so only the first token in it is accessed. The macro calls itself recursively to process the remaining characters until the `\@nil` has been met.

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

If the next token is `\@nil`, we've finished; the `\let` ensures that its parameter will be discarded (through the LaTeX internal command `\@gobble`) and the recursion will then unravel.

```
  \ifx#2\@nil \let\nextlet=\@gobble
```

Otherwise, we have another letter of the ⟨*answer*⟩ in `#2`, so we call `\letterput` to define the macro corresponding to this square, and count one more position occupied in the current direction.

```
  \else\letterput{#2}\advance#1 by \@ne
```

After we've processed this letter, we want to call this routine recursively to process the remaining letters (if any)...

```
    \let\nextlet=\nextletter
```

These letters cannot possibly require a starting square number, so we use a simpler macro for these later letters.

```
    \let\letterput=\laterletter
  \fi
```

This is where we either exit from the recursion (and `\@gobble` the `#1` parameter) or call the macro recursively to process the next character; the direction has to be passed on as the first parameter for `\nextletter` or `\@gobble`.

```
  \nextlet{#1}}
```

`\blacktest` Later we shall want to check that the square into which we are "putting" a letter is either black (and hence should be changed to contain the character) or has already had the *same* letter put into it by an intersecting light.

We don't want to expand each squares' macros, so these tests have to use the `\ifx` primitive; therefore, we need a macro which has the same substitution text *at the highest level.*

```
\def\blacktest{\blacksquare}
```

`\ifneed@d` `\need@dtrue` `\need@dfalse` As we create the macros corresponding to each occupied square, we have to decide whether it is necessary to actually perform a (re)definition of the macro; the following permits the code to determine whether such definition takes place (within the `\putsquare` macro).

```
\newif\ifneed@d
```

`\ifNoerr` `\Noerrtrue` `\Noerrfalse` If an error is detected during the placement of the occupied squares, there will probably be other errors; to save pouring out yards of error messages, we arrange to suppress all but the first such; the following `\newif` provides this facility.

And of course we start without any errors!

```
\newif\ifNoerr
\Noerrtrue
```

`\Number` `\Plain` To determine whether a square has already been occupied, and if so, by what, we require a couple of new tokens which can be tested for as the result of a macro expansion

```
\newtoks\Number \newtoks\Plain
```

`\blank` `\numbered` The next two definitions can be temporarily `\let` to be the replacements for `\lettersquare` and `\numbersquare` respectively, for use in conjunction with the aforementioned test.

```
\def\blank#1{\Plain}
\def\numbered#1#2{\Number}
```

`\letterinsquare` `\letterinnumbersquare` When we first read the clues, we create macros which are unique to each square; later we shall redefine the macros to which the squares' macros expand to actually perform the setting, but during the first phase we require expansions which correspond to the parameters alone. The following definitions are therefore used during the "filling" phase.

```
\def\letterinsquare#1{#1}

\def\letterinnumbersquare#1#2{#2}
```

`\setsquare` As we scan the answers for each clue, this macro is called with a parameter which specifies a call of either the `\lettersquare` or `\numbersquare` macros (with appropriate parameters).

It starts by assuming that no redefinition of the square's macro will be required.

```
\def\setsquare#1{%
  \need@dfalse
```

This macro is also called during the initialization of the grid, when we populate it with black squares; therefore, we definitely want to perform a definition of the square's macro at this stage.

```
  \ifx#1\blacksquare
    \need@dtrue
```

Otherwise, let's have a look and see what is already in the square's macro. This test will succeed only if the square contains its initial definition (as `\blacksquare`).

```
  \else
    \expandafter\ifx\csname\ther@w\thec@l\endcsname\blacktest
```

in which case we *will* want to redefine the macro for this square

```
      \need@dtrue
```

If the square has already been redefined, it means we've already "put" a letter (or number+letter) into its definition, so we will have to check that this definition doesn't conflict with the new one (which is in #1). We have to expand the macros to perform this test, but don't want that expansion to return anything except the letter which is being (and has been) placed in the square, so we make a temporary replacement for the two macros which might form our #1.

```
    \else
      \let\lettersquare=\letterinsquare
      \let\numbersquare=\letterinnumbersquare
```

Now we *expand* the original definition and the new one; these should be the same!

```
      \expandafter\if\csname\ther@w\thec@l\endcsname#1
      \else
```

If they aren't, it means the compiler of the crossword has made a mistake and has solutions which don't correctly intersect: tell him so (well, the first time anyway). We even provide the user with some help!

```
        \ifNoerr
          \errhelp{Two intersecting lights tried to
                   put different letters^^Jin the same square!
                   You've probably confused their coordinates.^^J
                   Carry on, and examine the printout.}
          \errmessage{Illegal redefinition of square \ther@w\thec@l.
                   Was: \expandafter\meaning\csname\ther@w
                                                  \thec@l\endcsname.
                   Now: \noexpand #1}
          \Noerrfalse
        \fi
      \fi
```

So far, we don't seem to need to change anything, but if the new #1 which has been passed to `\putsquare` specifies that it be numbered, we must subsume any existing definition which *doesn't* have a number.

Again, we make temporary reassignments for the two potential macros, which is where we make use of the two new tokens that we invented.

```
      \let\lettersquare=\blank
      \let\numbersquare=\numbered
```

If the following test succeeds, we know that the new parameter specifies a number as well as a letter. This code could be expanded to check whether the original definition also specified a number, and if so ensure that they are the same, but would anybody really be silly enough to give different numbers for clues starting from the same square?!

```
      \expandafter\ifx#1\Number
        \need@dtrue
      \fi
    \fi
  \fi
```

Now we know whether we must (re)define the macro which is unique to this square. If we must, we expand our `#1`, so as to get the actual number and letter passed in through that parameter, but keep the name of the placement macro itself[4] unexpanded.

```
  \ifneed@d
     \expandafter\edef\csname\ther@w\thec@l\endcsname{\noexpand #1}
  \fi
}
```

3.2 The \clue Command

\clue Well, here it is at last. We start off by extracting the parts (if any) which form the ⟨*clue_number*⟩ parameter. We will therefore have the purely numeric first portion of the ⟨*clue_number*⟩ in `\cluenumber`.

```
\def\clue#1#2#3#4#5#6#7{%
  \findnumber{#1}
```

We now examine the second (⟨*Across/Down*⟩) parameter of the `\clue` command to determine whether this is an Across or Down clue. The clue's ⟨*text*⟩ and ⟨*help*⟩ information is then written[5] to the appropriate auxiliary file, and note taken of the direction in which the ⟨*answer*⟩ should be set into the light of the grid.

Firstly we deal with writes to the `.acr` file, if the ⟨*Across/Down*⟩ parameter is the letter '`A`'. In this case, the counter to be incremented is `\Across`.

```
  \ifx#2A
    \if\@empty#7\relax\else
      \ifx\tf@acr\relax\else
        \@writefile{acr}{  \string\item[#1] #6 (#7)}%
      \fi
    \fi
    \let\Direction=\Across
  \else
```

If this parameter is the letter '`D`', writes go to the `.dwn` file, and the `\Down` counter is incremented.

```
    \ifx#2D
      \if\@empty#7\relax\else
        \ifx\tf@dwn\relax\else
          \@writefile{dwn}{  \string\item[#1] #6 (#7)}%
        \fi
      \fi
      \let\Direction=\Down
    \else
```

If this ⟨*Across/Down*⟩ parameter is not one of the two permitted characters, an error message is issued.

```
      \errhelp{The second parameter of the \string\clue\space
               command must be `A' or `D'}
      \errmessage{Illegal direction (#1) specification
                  for \string\clue.}
    \fi
  \fi
```

The x and y coordinate counters are set from the ⟨*column*⟩ and ⟨*row*⟩ parameters.

```
  \Across=#3 \Down=#4
```

[4] This will be the first token of the parameter itself.

[5] The writes only take place if the output files exist, and the ⟨*help*⟩ parameter is non-empty.

\letterput The \letterput macro is defined anew at the start of each clue to create the correct definition for the *first* square of each light. The \setsquare macro redefines this macro internally for the remaining squares.

```
    \edef\letterput##1{\noexpand\setsquare
                           {\noexpand\numbersquare
                                  {\noexpand\cluenumber}{##1}}}%
```

Now we can call \nextletter which will cycle through all the letters of the ⟨*answer*⟩ until meeting the token \@nil. As each letter is processed, it creates a definition for the current square, initially using the above definition, and then \laterletter for subsequent letters of the ⟨*answer*⟩.

Finally, we ensure that the newlines after \clue commands don't lead to unwanted spaces being typeset.

```
    \nextletter{\Direction}#5\@nil
    \ignorespaces
  }
```

3.2.1 Finding the clue number to be set in the light

\findnumber We mentioned earlier that clues with solutions which occupy more than one light require a special format for specifying their ⟨*clue_number*⟩. If this form is required, the number of the current light is given first, with the remaining text (as it is required to be set) following, separated from the first number by some non-digit character.

The macro \findnumber is called with the entire ⟨*clue_number*⟩ parameter passed to \clue and sets \cluenumber to expand to the first or only number found in that parameter (which should then appear in the first square of the light).

\clueNumber Of course, we require a counter in which to attempt to assemble that number:

```
  \newcount\clueNumber
```

\special@gobble This macro is used by \findnumber to discard the unwanted portion (if any) of a ⟨*clue_number*⟩, including the special termination token.

```
  \def\special@gobble #1\@nil{}
```

The following mechanism to separate the first (or only) number from the remainder was suggested by Frank Mittelbach of the University of Mainz, and replaced about two pages worth of code.

```
  \def\findnumber#1{%
```

We attempt to assign the ⟨*clue_number*⟩ parameter to the \clueNumber counter: only that portion consisting purely of digits will actually be assigned. The remainder, if any, including the special terminator sequence \@nil is then discarded by the \special@gobble command:

```
    \afterassignment \special@gobble \clueNumber=0#1 \@nil
```

If the user did not provide a valid ⟨*clue_number*⟩ (i.e. something starting with a digit), then clueNumber will have zero assigned to it — seems the user ought to be told about this!

This completes \findnumber.

```
    \ifnum\clueNumber=0
      \errhelp{The first parameter of the \string\clue\space command
               must commence with a digit}
      \errmessage{Illegal clue number (#1) specified
                  for \string\clue.}
    \fi
  }
```

`\cluenumber` This macro merely produces the number as saved in `\clueNumber`.

```
\def\cluenumber{\the\clueNumber}
```

3.3 Populating the Crossword Grid

`\blackenrow` For each column in a row we create a black square; effectively `\setsquare` will execute the definition

```
% \def\csname\ther@w\thec@l\endcsname{\blacksquare}
```

so that for row 3 column 6 of a 15×15 grid, for example, we would end up by defining `\def\RiiiCvi{\blacksquare}`.

Before starting the inner loop, we need to save the definition of `\body` which was created for the outer loop. We cannot do this by creating a new block, since that would require that each square be defined globally, which might give rise to save stack overflow problems.

```
\def\blackenrow{\let\savedbody=\body
  \loop\relax\ifnum\Across>\z@
    \setsquare{\blacksquare}%
```

We then shift ourselves back to the next column to the left and iterate. If we've reached the end of this inner loop, we re-establish the definition of `\body`.

```
    \advance\Across by \m@ne
  \repeat
  \let\body=\savedbody
}
```

`\SetUpGrid` This macro creates an empty grid of the appropriate size.

```
\def\SetUpGrid#1{%
```

We firstly make a note of the ⟨*gridsize*⟩ parameter in the `\gr@dsize` counter, from which the width and height of the grid may be computed. We also set the `\p@csize` counter to be one greater than `\gr@dsize` to save our recomputing this quantity many times over.

```
  \gr@dsize=#1
  \p@csize=#1 \advance\p@csize by \@ne
```

Right, this is where we start to generate the grid itself. We start at the bottom edge, because TeX loops are easiest if counting down to zero. Therefore, the `\Down` counter is set equal to the highest row number attainable.

```
  \Down=\gr@dsize
```

We now start a loop, so the following code will be repeated for each row of the grid in turn. As with the rows, we process the columns from highest address to lowest, so the `\Across` counter is also set to the highest column attainable.

```
  \loop
    \Across=\gr@dsize
```

Provided we haven't decremented down to the 0th row, we start off the inner loop to process each column: this is done by invoking a separate macro — the alternative to which would be to enclose the inner loop in a group, which would require the use of global definitions for each square.

```
    \ifnum\Down>\z@
      \blackenrow
```

Afterwards we move ourselves up one row, and iterate for the next row.

```
      \advance\Down by \m@ne
  \repeat
}
```

And that's the end of `\SetUpGrid`!

`\thec@l` `\ther@w` `\thec@l` and `\ther@w` macros generate a string of letters, starting with 'C' (for column) and 'R' (for row) respectively. The remainder of the string consists of the lower-case Roman numeral equivalent to the current value of the appropriate counter. Such all-letter strings are used to create the names for macros which can be unique for each square of the grid.

```
\def\thec@l{C\romannumeral\Across}
\def\ther@w{R\romannumeral\Down}
```

`\TestAnswers` This macro interacts with the user, if necessary, to get a yes or no indication of whether the answers shall be written into the grid. No check is made that the user has entered a valid response, but the use of `\answer` is such that any answer apart from a 'y' (in upper- or lower-case) is treated as if it were 'n'.

`\f@rst` To determine what parameter has been provided, or the response elicited, we will require a little macro to pass on the first token of a list terminated by a full stop.

```
\def\f@rst#1#2.{#1}
```

We commence by lower-casing the given parameter, setting the lower-cased version into the macro `\answer`.

```
\def\TestAnswers#1{\edef\next{\def\noexpand\answer{#1}}%
  \lowercase\expandafter{\next}%
```

We can then extract just the first character

```
  \edef\answer{\expandafter \f@rst \answer .}%
```

The we determine whether it's the letter 'y' or 'n'...

```
  \if\answer y \else \if\answer n \else
```

If the ⟨*visible*⟩ parameter isn't either of these, we ask the user to give us an answer!

```
    \typein[\answer]{Make answers visible? [Y/N]: }\fi
  \fi
```

OK, `\answer` now contains some response; let's upper-case it and extract just its first character

```
  \edef\next{\def\noexpand\answer{\answer}}%
  \uppercase\expandafter{\next}%
  \edef\answer{\expandafter \f@rst \answer .}%
}
```

3.4 Setting the Grid

`\letter` This is the expansion which is used (for `\letterinsquare`, and within the expansion of `\numberedsquare`) when the grid is actually being *typeset* from the stored squares' macros.

```
\def\letter#1{{\put(\Across,-\Down){\makebox(1,1){\tensf #1}}}}
```

`\numberedsquare` This macro is used (for `\numbersquare`) when the lights are being drawn with light numbering enabled. It puts the number (and the letter of the answer too, depending upon the definition of `\letter` which is current) at the current coordinate position specified by `(\Across,\Down)`.

```
\def\numberedsquare#1#2{%
  \put(\Across,-\Down){%
```

To insert the clue number, we generate it within a sub-picture, of size equal to one square.

```
    \begin{picture}(1,1)(0,0)
```

We stick the number in the top-left corner of an (invisible) box which fills the central 81% of the area.

```
      \put(0.05,0.05){\makebox(0.9,0.9)[tl]{\tiny #1}}
    \end{picture}%
  }
```

We also set the letter in the square (depending upon the definition of `\letter` which applies.

```
  \letter{#2}}
```

`\unnumberedsquare` Despite its name, this macro is invoked (through `\numbersquare`) for those squares which would ordinarily carry a light's number, were it not for the fact that the numbers have been suppressed by the crossword* environment.

It just discards the ⟨*clue_number*⟩ given in the first parameter and sets the current letter by invoking `\letter`, which uses the second parameter...

```
\def\unnumberedsquare#1{\letter}
```

`\FinishGrid` Now we can process all the stored macros which define the appearance of each square in the grid, and thus generate the printed version thereof, using LaTeX's picture environment.

This command makes appropriate redefinitions of some macros which produced different effects during the filling of the grid.

```
\def\FinishGrid{%
```

If the customer doesn't want the letters put into the grid, then we need only throw away any parameter to `letter`.

```
  \if\answer Y \else \let \letter=\@gobble \fi
```

As stated in the introduction, when typesetting the grid in a smaller version, there is insufficient space to include the numbers for the lights; by testing `\ifnumberit` we can determine whether a macro which expands to `\numbersquare` shall result in a number being printed or not.

```
  \ifnumberit
    \let\numbersquare=\numberedsquare
  \else
    \let\numbersquare=\unnumberedsquare
  \fi
```

Anything that's been stored as a `\lettersquare` is "set" using the `\letter` macro (which we might have just `\let` equal to `\@gobble`).

```
  \let\lettersquare=\letter
```

`\blacksquare` Any black squares that are still left in the grid are set by means of this command:

```
  \def\blacksquare{%
```

The black square itself is merely a rule of the appropriate dimensions.

```
    \put(\Across,-\Down){\rule{\unitlength}{\unitlength}}}
```

Now we come to the actual body of `\FinishGrid`.

We start off at the bottom-most row of the grid...

```
  \Down=\gr@dsize
```

The whole grid is created in a centered `\hbox` in a picture environment. By offsetting the origin negatively, we can address each row by simply negating the y coordinate; thus column x in the highest row is $(x, -1)$.

```
  \centerline{%
    \begin{picture}(\p@csize,\p@csize)(1,-\p@csize)
```

We now cycle through each of the rows. The first thing we output is a horizontal rule of the full width of the grid, one such rule being generated for each row of the grid, providing the horizontal lines across vertical lights.

```
   \loop\ifnum\Down>\z@
     \put(1,-\Down){\line(1,0){\the\gr@dsize}}
```

Now we are about to cycle across all the columns of the current row; again, it's convenient for us to work backwards to the left...

```
     \Across=\gr@dsize
```

To do this we need an inner loop; this has to be inside a group so as to isolate the effects of its `\repeat` command.

```
     {\loop \ifnum\Across>\z@
```

To set the appropriate object in this square, we merely invoke the macro which has been associated with the square. Thus we'll end up with a black square, or a numbered or plain square, the latter two of which may also contain a letter.

```
        \csname\ther@w\thec@l\endcsname
```

Now we advance to the next column and iterate. That's the end of the inner loop for each of the columns of the current row.

```
        \advance\Across by \m@ne
      \repeat
     }%
```

Now we can decrement down to the next row and iterate through the rows.

```
     \advance\Down by \m@ne
   \repeat
```

We've so far drawn a horizontal line *under* each of the rows; the next `\put` draws a final line *above* the top-most row.

```
   \put(1,0){\line(1,0){\the\gr@dsize}}
```

Similarly, a short loop can draw vertical rules at the left-hand edge of each of the columns, starting with a line on the left of an imaginary column to the right of the whole grid, which will therefore form a line to the right of the final column.

```
   \Across=\p@csize
   \loop\ifnum\Across>\z@
     \put(\Across,0){\line(0,-1){\the\gr@dsize}}
     \advance\Across by \m@ne
   \repeat
```

And that completes the picture.

```
   \end{picture}%
 }%
}
```

Finally, here's the input which produced the crossword in figure 1

```
%       \begin{crossword}{15}{N}
%          \input{grid_1}
%       \end{crossword}
```

where the file `grid_1.tex` reads as follows:

```
\clue{8}{A}{1}{2}{SWAM}{Points a thousand tested for witchcraft.}{4}
\clue{9}{A}{6}{2}{OHO}{Gourmet's triumphant cry on finding middle-cut
                      Pacific salmon!}{3}
\clue{10}{A}{10}{2}{ICECAP}{One hundred stride backwards across a
```

```
                                     Pole.}{3-3}
\clue{11}{A}{1}{4}{BOPEEP}{Fifties' jazz record about Eastern childs'
                                    play.}{2-4}
\clue{12}{A}{8}{4}{SCHEDULE}{Timetable created by editor in synagogue
                                      of Spain.}{8}
\clue{13}{A}{1}{6}{THALASSOGRAPHER}{The dialect a girl mixed up tangle
                                             around symbolic diagram used by
                                             maritime student!}{15}
\clue{15}{A}{1}{8}{HAIRPIN}{Wire fastening bent road narrowly.}{7}
\clue{17}{A}{9}{8}{UMBRAGE}{Hammerhead consumes German company
                                     and casts a shade.}{7}
\clue{20}{A}{1}{10}{SCALENETRIANGLE}{Strange cel alien chops to make a
                                              figure with odd sides.}{7,8}
\clue{23}{A}{1}{12}{AMOUNTED}{Sounds like a hoarse editor came to in
                                       total!}{8}
\clue{25}{A}{10}{12}{ALLEGE}{Assert without proof everyone, for
                                      example, English.}{6}
\clue{26}{A}{1}{14}{FLORAL}{Flourished examination of flowers.}{6}
\clue{27}{A}{8}{14}{NIL}{Floor covering discards fuming sulphuric acid
                                 and returns to nothing.}{3}
\clue{28}{A}{12}{14}{TACE}{``Latin for a candle'' to be silent note
                                    about aircraftman?}{4}
\clue{1}{D}{2}{1}{SWOOSH}{Water rush noise disturbs show so!}{6}
\clue{2}{D}{4}{1}{IMPELLER}{Mischievous child with cloth measure
                                     hesitates to assemble rotor.}{8}
\clue{3}{D}{6}{1}{COMPASSIONATELY}{Mercifully inclined to pass round
                                            ten? Nay, about short blower!}{15}
\clue{4}{D}{8}{1}{TORSION}{Sort ion? An isotron gives it a new
                                    twist!}{7}
\clue{5}{D}{10}{1}{DITHYRAMBICALLY}{Wildly and boisterously rearrange
                                             Billy May Third, roughly.}{15}
\clue{6}{D}{12}{1}{SENDUP}{Satirical book or film---give odds about
                                    revision of ``Dune''?}{4-2}
\clue{7 {\noexpand\rm\&} 24}{D}{14}{1}{PALL}{Premier took in a Lord
                                             Lieutenant and all played an
                                             old game in London street.}{4-4}
\clue{14}{D}{14}{6}{ERG}{Work expended in power games?}{3}
\clue{16}{D}{2}{8}{ARC}{A church circle (or part of one).}{3}
\clue{18}{D}{12}{8}{RINGLETS}{Encircle hindrances under hair in long
                                       curls.}{8}
\clue{19}{D}{8}{9}{STUDENT}{Boss over oto\-rhino\-laryng\-o\-logy
                                     department undergraduate.}{7}
\clue{21}{D}{4}{10}{LOUVRE}{Old dovecote in Parisian museum.}{6}
\clue{22}{D}{14}{10}{LEGACY}{Like ornamental fabric, for example,
                                      that's in bequest.}{6}
\clue{24}{D}{2}{12}{MALL}{}{See {\bf 7}}
```

◇ B Hamilton Kelly
School of Electrical Engineering & Science
Royal Military College of Science
Shrivenham
SWINDON
United Kingdom
Janet: `tex@uk.ac.cranfield.rmcs␣`