Some Macros to Draw Crosswords

B Hamilton Kelly†

October 6, 2008

Abstract

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

Contents

1 Introduction 3 Creating the Grid 7

1.1 How to Specify Clue Numbers . . . . . . . . . . . . . . 2

2 Definition of the Macros 3

2.1 Counters and Lengths . . 3

2.2 Reading and Writing the Clues . . . . . . . . . . . . . . 5

2.3 Tabulating the Clues . . . . . . . . . . . . . . . . . . . . 6

3 Creating the Grid 7

3.1 Macros used when Populating the Grid . . . . . . . . . 8

3.2 The \clue Command . . 11

3.2.1 Finding the clue number to be set in the light . . . . 12

3.3 Populating the Crossword Grid . . . . . . . . . . . . . 13

3.4 Setting the Grid . . . . . . . . . . . . . . . . . . . . . 15

List of Figures

1 A Sample Crossword . . . . . . . . . . . . . . . . . . . . . . . . 4

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”

---

*This file is v2.8b, dated 1996/12/06
†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!
¹That at RMCS has a bottle of wine as a prize!
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:

\langle \text{gridsize} \rangle \text{ This is a number which specifies the columns (and rows) in the square grid.}

\langle \text{visible} \rangle \text{ 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:

\langle \text{gridsize} \rangle \text{ As before, this specifies the number of squares in each axis.}

\langle \text{header_text} \rangle \text{ Some text which will be set (in bold) above the completed grid.}

Here is an example of the `crossword` environment:

\hspace{1cm}

\textbf{Last month’s solution}

\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textsc{perambulator} & \textsc{leen} & \textsc{cr} & \\ \hline
\textsc{impasse} & \textsc{termini} & \textsc{sptmregn} & \\ \hline
\textsc{tepolo} & \textsc{aramaic} & \textsc{lrrngma} & \\ \hline
\textsc{ensue} & \textsc{implie} & \textsc{tion} & \\ \hline
\textsc{sced} & \textsc{surprised} & \textsc{dunce} & \\ \hline
\textsc{naeaios} & \textsc{erratum} & \textsc{liassic} & \\ \hline
\textsc{eralbbie} & \textsc{sabbath} & \textsc{isotron} & \\ \hline
\textsc{itdlulut} & \textsc{standingroom} & \\ \hline
\end{tabular}

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

\langle \text{clue_number} \rangle \text{ 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.}

\langle \text{Across/Down} \rangle \text{ This parameter must be either the letter ‘A’ or ‘D’, in upper-case.}

\langle \text{col_number} \rangle \text{ The } x \text{-coordinate of the first square of the light. The leftmost column of the grid is numbered 1.}

\langle \text{row_number} \rangle \text{ The } y \text{-coordinate of the first square of the light. The top-most row is numbered 1.}

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

\(^2\text{The lower-case letters ‘y’ and ‘n’ are also recognized.}\)
\textit{text} The text of the clue itself. If you want to use any \LaTeX{} macros in this text, such as \texttt{\textbackslash dots}, each such macro must be preceded by \texttt{\textbackslash noexpand}. This includes such macros as \texttt{\&}, to produce an explicit ampersand.

\textit{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”.

1.1 How to Specify Clue Numbers

Sometimes the solution to one clue is split amongst a number of “lights”. To cover this eventuality, provide a \texttt{\clue} for each of the lights involved, with the solution to that light \texttt{alone} given as \texttt{\answer}. All except the \texttt{\clue} corresponding to the first light of the solution should have a null \texttt{\textit{text}}, and the \texttt{\textit{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 \texttt{\clue} for the first light of the solution should provide the entire clue as its \texttt{\textit{text}}, and the \texttt{\textit{help}} should say something like “7,3-3”. The \texttt{\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

\begin{verbatim}
\clue{13}{A}{5}{1}{PARLIAMENT}{see 9d}
\clue{9 \noexpand\& 13a}{D}{1}{10}{ACTOF}\% 
\{Bill’s desired outcome?\}{3,2,10}
\end{verbatim}

will produce

13 \hspace{1em} (see 9d)

amongst the ACROSS clues, and

9 \& 13a Bill’s desired outcome? \hspace{1em} (3,2,10)

amongst the DOWN clues

2 Definition of the Macros

\texttt{\ninept} 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.

1 (\*package)
2 \texttt{\def\ninept{\@setsize\ninept{11pt}\ixpt\@ixpt}}
3 \texttt{\abovedisplayskip 8.5pt plus 3pt minus 4pt}

\texttt{\noexpand} before the \texttt{\rm} for the \texttt{\&}
Figure 1: A Sample Crossword
2.1 Counters and Lengths

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.

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

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.

Next paragraph is no longer true. I changed the coding to allow more than 6 crosswords to be used in one document. Using the approach outlined below the system would run out of write channels pretty soon. (FMi)

[ These files are created in the same manner as table-of-contents (.toc) files, etc; thus \TeX 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 \TeX’s @input command and \TeX’s \openout primitive. ]

\newwrite{\tf@acr}
\newwrite{\tf@dwn}
Here’s the preliminary material that gets inserted into the .acr file.

Whilst something similar goes into the .dwn file.

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...

Then for the down clues.

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

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.
\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} 
}\\%}

3 Creating the Grid

The remaining commands are concerned with creating (and, optionally, populat-
ing) the crossword grid

\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.
\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* environ-
ment).
\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,
\RxiiCi\vi, 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\tfdwn=\relax \let\tacr=\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.

86 \def\nextletter#1#2{% 
   If the next token is \@nil, we’ve finished; the \let ensures that its parameter will be discarded (through the \TeX internal command \@gobble) and the recursion will then unravel.

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

   Otherwise, we have another letter of the \langle\textit{answer}\rangle in #2, so we call \letterput to define the macro corresponding to this square, and count one more position occupied in the current direction.

88  \else\letterput{#2}\advance#1 by \@one

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

89  \let\nextlet=\nextletter

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

90  \let\letterput=\laterletter

91  }% 

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.

92 \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.

93 \def\blacktest{\blacksquare}

94 \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).

94 \newif\ifneed@d

95 \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.

95 \newif\ifNoerr

96 \Noerrtrue

9
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.

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.

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.

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).

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.

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).

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

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!

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.

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?!

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.

\(^4\)This will be the first token of the parameter itself.
3.2 The \clue Command

\clue Well, here it is at last. We start off by extracting the parts (if any) which form the \langle\text{clue\_number}\rangle parameter. We will therefore have the purely numeric first portion of the \langle\text{clue\_number}\rangle in \text{cluenumber}.

\begin{verbatim}
138 \def\clue#1#2#3#4#5#6#7{%
139   \findnumber{#1}

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

Firstly we deal with writes to the \texttt{.acr} file, if the \langle\text{Across/Down}\rangle parameter is the letter 'A'. In this case, the counter to be incremented is \texttt{\Across}.

\begin{verbatim}
140 \ifx#2A
141   \if\@empty#7\relax\else
142     \@writefile{acr}{ \item\[#1\] #6 (#7)}%
143   \fi
144 \fi
145 \let\Direction=\Across
146 \else
147 \fi
148 \let\Direction=\Down
\end{verbatim}

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

\begin{verbatim}
148 \ifx#2D
149   \if\@empty#7\relax\else
150     \@writefile{dwn}{ \item\[#1\] #6 (#7)}%
151   \fi
152 \fi
153 \let\Direction=\Down
154 \else
155 \fi
\end{verbatim}

If this \langle\text{Across/Down}\rangle parameter is not one of the two permitted characters, an error message is issued.

\begin{verbatim}
156 \errhelp{The second parameter of the \string\clue\space command must be 'A' or 'D'}
157 \errmessage{Illegal direction (#1) specification for \string\clue.}
158 \fi
159 \fi
\end{verbatim}

The \texttt{x} and \texttt{y} coordinate counters are set from the \langle\text{column}\rangle and \langle\text{row}\rangle parameters.

\Across=#3 \Down=#4

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

\begin{verbatim}
163 \edef\letterput##1{\noexpand\setsquare
\end{verbatim}

\textsuperscript{5}The writes only take place if the output files exist, and the \langle\text{help}\rangle parameter is non-empty.
Now we can call \nextletter which will cycle through all the letters of the \langle answer \rangle 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 \langle answer \rangle.

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 \langle clue number \rangle. 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 \langle clue number \rangle 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:

\special@gobble

This macro is used by \findnumber to discard the unwanted portion (if any) of a \langle clue number \rangle, 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{\afterassignment \special@gobble \clueNumber=0#1 \@nil

We attempt to assign the \langle clue number \rangle 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 \langle clue number \rangle (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\string\space command must commence with a digit}\errmessage{Illegal clue number (#1) specified for \string\clue.}
This macro merely produces the number as saved in \clueNumber.

\def\cluenumber{\the\clueNumber}

3.3 Populating the Crossword Grid

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

\def\blackenrow{\let\savediterate\iterate
\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.

Again with 2e this has to be \iterate.

\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 \gridsize counter, from which the width and height of the grid may be computed. We also set the \p@csizer counter to be one greater than \gridsize to save our recomputing this quantity many times over.

\gridsize=#1
\p@csizer=#1 \advance\p@csizer 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=\gridsize

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.
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.

\def\blackenrow{\ifnum\Down>\z@
  \blackenrow
\fi}

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

\def\repeat{
  \blackenrow
  \advance\Down by \m@ne
  \repeat}

And that’s the end of \SetUpGrid!

\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’.

\def\f@rst#1#2.{#1}
\def\TestAnswers#1{
  \edef\next{\def\noexpand\answer{#1}}%
  \lowercase\expandafter{\next}
  \edef\answer{\expandafter \f@rst \answer .}
}

\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’.

\def\f@rst#1#2.{#1}
\def\TestAnswers#1{
  \edef\next{\def\noexpand\answer{#1}}%
  \lowercase\expandafter{\next}
  \edef\answer{\expandafter \f@rst \answer .}
}

\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’.

\def\f@rst#1#2.{#1}
\def\TestAnswers#1{
  \edef\next{\def\noexpand\answer{#1}}%
  \lowercase\expandafter{\next}
  \edef\answer{\expandafter \f@rst \answer .}
}

\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’.

\def\f@rst#1#2.{#1}
\def\TestAnswers#1{
  \edef\next{\def\noexpand\answer{#1}}%
  \lowercase\expandafter{\next}
  \edef\answer{\expandafter \f@rst \answer .}
}
\section*{3.4 Setting the Grid}

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

\begin{verbatim}
213 \def\letter#1{{\put(\Across,-\Down){{\makebox(1,1)\{sffamily #1\}}}}}
\end{verbatim}

\texttt{\textbackslash numberedsquare} This macro is used (for \texttt{\numberedsquare}) 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 \texttt{\letter} which is current) at the current coordinate position specified by (\Across, \Down).

\begin{verbatim}
214 \def\numberedsquare#1#2{\put(\Across,-\Down){{\makebox(0.9,0.9)[tl]\{\tiny #1\}}}}
\end{verbatim}

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

\begin{verbatim}
\begin{picture}(1,1)(0,0)
\put(0.05,0.05){{\makebox(0.9,0.9)[tl]\{\tiny #1\}}}
\end{picture}
\end{verbatim}

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

\begin{verbatim}
220 \letter{#2}}
\end{verbatim}

\texttt{\textbackslash unnumberedsquare} Despite its name, this macro is invoked (through \texttt{\numberedsquare}) 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 (\texttt{clue\_number}) given in the first parameter and sets the current letter by invoking \texttt{\letter}, which uses the second parameter...

\begin{verbatim}
221 \def\unnumberedsquare#1{\letter}
\end{verbatim}

\texttt{\textbackslash 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 \texttt{picture} environment.

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

\begin{verbatim}
222 \def\FinishGrid{\%
223 \if\answer Y \else \let \letter=\@gobble \fi
224 \%
225 \if\numberit
226 \let\numberedsquare=\numberedsquare
227 \fi
228 \end{verbatim}
\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, \mbox{\(-1\)}).

\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}}\repeat

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@\endcsname

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@\ther\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}

\begin{singlespace}
\begin{verbatim}
\begin{crossword}{15}{N}
\input{grid-1}
\end{crossword}
\end{verbatim}
\end{singlespace}
283 \clue{6}{D}{13}{1}{ORIGAMI} (Maybe magic endlessly raised before one with Japanese artistic skill){7}
284 \clue{7}{D}{1}{2}{LISTLESSNESS} (Enroll fewer once before head suffering a languid malaise){12}
285 \clue{10}{D}{15}{3}{INCANDESCENT} (White-hot Peruvian reorganized objects covering small coin){12}
286 \clue{15}{D}{9}{7}{PEDALLING} (Penny led a rearranged fish engaging in cyclic activity!){9}
287 \clue{17}{D}{3}{9}{RAREBIT} (Welsh toast? An unusual drill!){7}
288 \clue{18}{D}{5}{9}{RETRAIN} (Keep in mind, take in, and acquire new skills){7}
289 \clue{19}{D}{11}{9}{DIABOLO} (Twice a high ball thrown up round a two-headed top){7}
290 \clue{20}{D}{13}{9}{MOSTRUM} (No good man with spirit offers quack remedy){7}
291 \clue{22}{D}{7}{11}{MAHDI} (I would shortly overact, sent up, and the downfall of Gordon){5}
292 ⟨grid0⟩
293 ⟨∗grid1⟩
294 \clue{8}{A}{1}{2}{SWAM} (Points a thousand tested for witchcraft.){4}
295 \clue{9}{A}{6}{2}{OHO} (Gourmet’s triumphant cry on finding middle-cut Pacific salmon!){3}
296 \clue{10}{A}{10}{2}{ICECAP} (One hundred stride backwards across a Pole.){3-3}
297 \clue{11}{A}{1}{4}{BOPEEP} (Fifties’ jazz record about Eastern child’s play.){2-4}
298 \clue{12}{A}{8}{4}{SCHEDULE} (Timetable created by editor in synagogue of Spain.){8}
299 \clue{13}{A}{1}{6}{THALASSOGRAPHER} (The dialect a girl mixed up tangle around symbolic diagram used by maritime student!){15}
300 \clue{15}{A}{1}{8}{SCALENETRIANGLE} (Strange cel alien chops to make a figure with odd sides.){7,8}
301 \clue{16}{A}{10}{12}{AMOUNTED} (Sounds like a hoarse editor came to in total!){8}
302 \clue{17}{A}{1}{10}{TACE} (‘‘Latin for a candle’’ to be silent note about aircraftman?){4}
303 \clue{18}{A}{1}{14}{FLORAL} (Flourished examination of flowers.){6}
304 \clue{20}{A}{6}{14}{NIL} (Floor covering discards fuming sulphuric acid and returns to nothing.){3}
305 \clue{22}{A}{8}{14}{TORSION} (Sort ion? An isotron gives it a new twist!){7}
306 \clue{23}{A}{10}{1}{DITHYRAMBICALLY} (Wildly and boisterously rearrange Billy May Third, roughly.){15}
307 \clue{24}{A}{1}{12}{TACE} (‘‘Latin for a candle’’ to be silent note about aircraftman?){4}
308 \clue{25}{D}{1}{2}{SWGOSH} (Water rush noise disturbs show so!){6}
309 \clue{26}{D}{4}{1}{IMPELLER} (Mischiefly child with cloth measure hesitates to assemble rotor.){8}
310 \clue{27}{D}{6}{1}{COMPASSIONATELY} (Mercifully inclined to pass round ten? Nay, about short blower!){15}
311 \clue{28}{D}{8}{1}{TORSION} (Sort ion? An isotron gives it a new twist!){7}
312 \clue{29}{D}{10}{1}{DITHYRAMBICALLY} (Wildly and boisterously rearrange Billy May Third, roughly.){15}
313 \clue{30}{D}{12}{1}{SENDUP} (Satirical book or film---give odds about
 revision of ‘‘Dune’’?{4-2}
\begin{crossword}{15}{?}
\input{grid1}
\end{crossword}
\section*{A Sample Crossword}
\textit{(Chamber's Twentieth Century dictionary)}
\begin{crossword}{15}{?}
\input{grid0}
\end{crossword}
\begin{crossword*}{15}{Solution to grid 0}
\input{grid0}
\end{crossword*}