Form Letters with 3-Across Labels Capability

Jackie Damrau
Superconducting Super Collider Laboratory
Dallas, Texas, 75237 USA
214-708-6048; FAX: 214-708-5143
Internet: damrau@sscvx1.ssc.gov

Michael Wester
Department of Mathematics and Statistics
University of New Mexico
Albuquerque, New Mexico, 87131 USA
505-277-4613
Internet: wester@spectre.unm.edu

Abstract

This article discusses a general-purpose program for generating form letters, using either TEX or LATEX. Given three inputs: a preamble file for initializations, a list of blank separated addresses, and a letter template, this program can be used to generate a letter per address and provide personalizations as directed by the template. Sample applications are presented, including one which constructs 3-across mailing labels. Thus, both form letters and mailing labels can be generated from the same list of addresses by simply changing two inputs to the program.

Introduction

With TEX or LATEX, it is not hard to produce a letter to be sent to a single addressee. Nor is it difficult to create multiple letters that follow a similar format by setting up a form in which changeable parameters, such as the name and address, are specified by macros. The form can then be input a fixed number of times, each time preceded by a redefinition of the parameters. However, there are problems with this approach. Modifying the list of addresses or adding new parameters to the form can be cumbersome. Also, serious reformatting may be required to use the individual pieces of information (such as the names and addresses) in other contexts.

An easy-to-use, general-purpose program to generate form letters has been developed that overcomes the above problems. This program, address, is written using TEX constructs and macros and can be executed by TEXing or LATEXing it.

The address program requires three user-supplied files: a preamble for performing initializations (which is optional), a list of addresses separated by blank lines, and a template. On execution, address asks for three file names, and then reads in the addresses one by one. For each address, the individual components are assigned to various macros, after which the template file is `\input`. The template can, therefore, refer to these macros.

The address list need not contain any formatting instructions as individual lines of text within a given address are retained by address. This allows the address file to be pure text and usable by any other program. Nor is it required that an address be simply that; any other data, such as telephone numbers, test scores, or whatever, may be included. Macros are provided to extract both individual lines and individual blocks of data.

In the following sections, instructions on how to set up the necessary files and details on running the program are furnished. In addition, a template for producing 3-across labels is provided which demonstrates a simple, but useful application. Finally, some discussion of the construction of the macros in address is given, explaining some of the difficulties encountered and how they were overcome.

Setting Up the Files

The easiest file to set up is the address list. It consists of blocks of text separated by blank lines. Commonly, the first line of a block will be the name of

an addressee, while the rest of the lines form the address. The address program therefore assigns these segments of text to the macros \Name and \Address, respectively.

Unlike normal TEX input, individual lines of text within an address (which can be of arbitrary length) retain their identity. This is accomplished by concatenating the lines, using \\ as a separator. For LATEX, this is quite convenient since a reference to \Address will result in the expansion of each \\ as a new line. In TEX, a similar effect can be obtained by

```
\def\\{\hfil\break}
```

In the template file, individual lines within \Address can be selected by \AddrLine{n}, where n is a positive integer. In addition, \Laddr will count the number of lines in \Address and \Naddr will refer to the current position in the address list.

It is often desirable to include other information along with the address. The easiest way to do this with address is to divide the lines of text in the address segment into subblocks, each of which can be of variable length. The macro call, \AddrBlock{k}, is provided to select the k^{th} subblock, where a line consisting of --- acts as a separator between subblocks. (Two consecutive lines of --- will not produce an empty subblock, but this effect can be achieved by inserting a line consisting of {} between the lines.)

To extract individual lines from a subblock of \Address, a two-step process is required. \StoreAddrBlock{k}\in\Block will define the contents of the macro \Block to be the k^{th} subblock of \Address. \GetLine{n}\of\Block will then select the n^{th} line of the \Block. This procedure is necessary to circumvent some peculiarities of TEX macro expansion.

One of the more salient features of address is the ability to intelligently parse the \Name and break it up into its components. For example, suppose the name is The Honorable and Mrs. Henry \& Matilda Edward Bo van Frothingham III, Royals. The following macro assignments will then be made when this name is parsed:

\SocialTitle → The Honorable and Mrs.
\FirstName → Henry \& Matilda
\MiddleName → Edward Bo
\LastName → van Frothingham
\Suffix → III
\OtherTitle → Royals

Simpler names will result in some of the above macros having null definitions.

The macros associated with the address program are summarized in the following table.

\DEFAULTtolist	default address list (initially, tolist)
\DEFAULTletter	default letter template (initially, letter)
\Name	first line of the address
\Address	subsequent lines of the address
\SocialTitle	e.g., Dr., Mr., Ms.
\FirstName	first name
\MiddleName	middle names
\LastName	last name
\Suffix	e.g., Sr., Jr., III
\OtherTitle	academic and professional titles
\Naddr	position of the address in the tolist
\Laddr	number of lines in \Address
\AddrLine{n}	n^{th} line of \Address
\AddrBlock{n}	n^{th} subblock of \Address
\StoreAddrBlock{n}\in\B	store the n^{th} subblock of \Address in \B
\GetLine{n}\of\B	n^{th} line of \B
\addspace\A	adds a space after \A if it is not null
\topbox{H}{W}{text}	top aligned box of height H and width W

An example of an address list appears below. Notice that following the address in each instance is a separate block of two lines.

```
Mrs. Apple Thesaurus
Apt. Z
234 Gestalt Lane
Cockermouth, Umbria, U.K.
---
artichokes
jalape\~nos

Harry K. Banana
P.O. Box 29Z46
Kahului, Maui, Hawaii
---
mustard greens
okra
```

A typical (LaTeX) letter template that might be used with the above list of addresses looks like:

```
\Name\\
\AddrBlock{1}

\StoreAddrBlock{2}\in\veggies
Dear \addspace\SocialTitle\LastName:

Welcome to the vegetable of the month
club!  Your introductory offer of
two selections this first month are
\GetLine{1}\of\veggies\ and
\GetLine{2}\of\veggies.  ...
\newpage
```

The macro `\addspace` is used to produce correct spacing in the salutation by adding a proper space after `\SocialTitle` if it contains text and doing nothing if `\SocialTitle` is empty (as would be the case for the second addressee).

The last file needed by address is the preamble. This file is `\input` once and is used to perform initializations, such as setting margins and defining macros. If address is being LaTeXed, the preamble is input before the `\begin{document}` that is executed automatically before processing the addresses. A sample preamble that can be used under TeX or LaTeX is given below. (The definition for `\ifundefined` can be found in Knuth [page 40] and is present in address.)

```
%
% Generic TeX/LaTeX preamble for
%    address.tex .
%
\ifundefined{LaTeX}                % TeX
   \magnification=\magstep1
   \voffset=0in
   \hoffset=0in
   \vsize=9in
   \hsize=6.5in
   \nopagenumbers
   %
   \def\\{\hfil\break}
   \def\newpage{\vfil\eject}
\else                              % LaTeX
   \documentstyle[12pt]{letter}
   \topmargin 0in
   \headheight 0in
   \headsep 0in
   \oddsidemargin 0in
   \textheight 9in
   \textwidth 6.5in
   \pagestyle{empty}
\fi
```

Running the Program

The address program is executed simply by typing *tex address* or *latex address*. This action produces the following set of requests:

```
Enter the filename of the preamble
   [preamble.tex]:
Enter filename of recipients' addresses
   [tolist.tex]:
Enter the filename of the letter
   template [letter.tex]:
```

The filenames in square brackets (`[ ]`) are default values and are accepted by pressing a RETURN. (The default names for the second and third files can be changed by redefining the macros `\DEFAULTtolist` and `\DEFAULTletter` in the preamble.) Of course, the various files should contain commands appropriate to the package actually being used.

address is designed to be reasonably robust. This and TeX's rules for reading input allows some sloppiness in setting up the address list. For example, leading and trailing white space and extra blank lines are all ignored. Also, a % can be used to comment-out text. This last item implies that any line with a % as its first nonblank character will be treated as a blank line.

Making 3-Across Mailing Labels

By taking advantage of the macros defined in address, it is not difficult to design a template that can produce 3-across labels. The following template will create a three-column, 33 labels-per-page format for a standard sheet of 2.75"×1" labels. The resulting output can be sent directly to sheets of labels or onto regular paper, which can then be photocopied onto label sheets.

```
\ifcase\the\Naddr
  \or\topbox{1in}{2.75in}{\Name\\\AddrBlock1}%
  \or\topbox{1in}{2.75in}{\Name\\\AddrBlock1}%
  \or\topbox{1in}{2.75in}{\Name\\\AddrBlock1}\\
     \Naddr=0
\fi
```

The methodology used here is to produce differing output depending on the value of `\Naddr`. The `\ifcase` construct will result in three horizontally aligned boxes, each containing the current contents of `\Name` and the first subblock of `\Address`, as `\Naddr` takes on the values 1, 2 and 3. At the end of the third case, a new line is started and `\Naddr` is reset to 0. Thus, the next time around, address will have incremented `\Naddr` back to 1 and the process will start over. The macro `\topbox{`*H*`}{`*W*`}{text}`, defined in address by

```
\def\topbox#1#2#3{\leavevmode
  \vtop to #1{\hsize=#2 #3\vfil\eject}}
```

will produce a top-aligned box containing the `text` of height H and width W.

The above strategy is easily modified to handle any number of columns and any spacing requirements for a regular gridlike pattern of labels. Depending on the printer settings, the top and left margins established in the preamble (not shown) may need to be changed as well. Using 12-point fonts, it is possible to put 6 lines of text in a one-inch-high label. This can be increased by decreasing the fontsize set in the preamble.

Comments on other labeling schemes. A label-making capability is already available in LaTeX, as well as in other programs; but these do not possess the flexibility nor the ease of use exhibited by address. According to Lamport [page 67], the `\makelabels` command prints a "list of mailing labels, one for each `letter` environment, in a format suitable for xerographic copying onto 'peel-off' labels." However, the two-column format produced does not correspond nicely to 3-across and other common mailing label arrangements nor can it be changed easily.

In printing 3-across mailing labels in Microsoft Word, it is necessary to type three sets of field names, a NEXT instruction telling Microsoft Word to place information from more than one record onto a single copy of a form document, and various other commands and formatting statements. Typically, in programs of this type, a number of steps will be required; and again, the choice of output formats will be quite limited. Also, the data typically cannot be a simple ASCII file but must be converted into the program's possibly multifield internal format.

Comments on the Code

In developing address, certain difficulties had to be overcome. The solutions found may be of benefit to other users. One problem encountered was creating a box of text that had a definite height, as well as a definite width. The `\topbox` macro mentioned above has these features. It is adapted from the definition of LaTeX's `\parbox` command. With respect to TeX's viewpoint, the H in the definition of `\topbox` is really a depth with the height of the box being zero, but these details can normally be ignored.

A second obstacle was obtaining sequential space delimited strings (e.g., words) from a line of text in a robust manner. Macros to do this were needed to build the name-parsing macro (`\breakup`). For example, suppose `\List` is defined by `\def\List{ a1 b2 c3 }`. The first 'word' of `\List` is `a1`, the second is `b2`, and the third is `c3`. One way to select elements in this fashion is to construct macros, `\wcar` and `\wcdr`, that are analogous to the Lisp functions, CAR and CDR. `\wcar\List\nil` should select `a1` and `\wcdr\List\nil` should return `{b2 c3 }`. Moreover, `\wcar` and `\wcdr` of `{ }` and `{}` should be null, and any sequence of multiple spaces should be treated like a single space.

If `\List` is a simple list of tokens (for example, `{ a b c }`), then a token CAR and CDR can be defined as follows:

```
%
% Test for {}.
%
\def\ifnull#1{\ifx#1\empty}
%
% \tcar\List\nil picks off the first non-
%    blank token (which is typically a
%    character or a control sequence) in
%    the \List.  If the \List is blank or
%    empty, then a null string is returned.
%
\def\tcar#1\nil{\ifnull#1
                   \empty
                \else
                   \tCar#1\nil
                \fi}
\def\tCar#1#2\nil{#1}
%
% \tcdr\List\nil removes the first nonblank
%    token in the \List and any preceding
%    blanks.  If the \List is blank or
%    empty, then a null string is returned.
%
\def\tcdr#1\nil{\ifnull#1
                   \empty
                \else
                   \tCdr#1\nil
                \fi}
\def\tCdr#1#2\nil{#2}
```

The general case is trickier and requires auxiliary macros.

```
%
% \ABDReverseExpand{D}{C}{B}{A}
%    first expands A, then expands B, then
%    expands D.
%
\def\ABDReverseExpand#1#2#3#4{%
   \expandafter\expandafter
      \expandafter#1%
         \expandafter\expandafter
            \expandafter#2\expandafter#3#4}
%
% Used to remove leading spaces.
```

```
%
\def\pretrim.#1{#1}
%
% \wcar\List\nil picks off the first word
%    (string of nonblank characters) in the
%    \List.  If the \List is blank or
%    empty, then a null string is returned.
%
\def\wcar#1\nil{%
   \ifnull#1
      \empty
   \else
      \expandafter\wCar\pretrim.#1 \nil
   \fi}
\def\wCar#1 #2\nil{#1}
%
% \wcdr\List\nil removes the first word and
%    any preceding blanks from the \List.
%    If the \List is blank or empty, then a
%    null string is returned.
%
\def\wcdr#1\nil{%
   \ifnull#1
      \empty
   \else
      \ABDReverseExpand
            \ifx\empty\wCdr\pretrim.#1 \nil
         \empty
      \else
         \expandafter\wCdr\pretrim.#1\nil
      \fi
   \fi}
\def\wCdr#1 #2\nil{#2}
```

`\ABDReverseExpand` is a simplification of the example found in Knuth [page 374] in the "Dirty Tricks" appendix.[1]

One more pair of useful Lisplike macros are `\setq` and `\gsetq`, which are defined by

```
\def\setq#1#2{\edef#1{\expandafter#2}}
\def\gsetq#1#2{\xdef#1{\expandafter#2}}
```

These macros allow statements such as

```
\setq\List{\wcdr\List\nil}
```

to function correctly by performing an immediate expansion on the second argument. This particular example results in `\List` being redefined to be `{b2 c3 }`.

Using the preceding as building blocks, it is easy to devise more complex macros. address defines `\wmember\Element\of\List\nil`, which causes `\ifwmember` to be true if the `\Element` is found in the `\List` (false otherwise); and `\wendcarcdr\List\nil\A\B`, which assigns `\A` and `\B` to the CAR and CDR of the `\List`, starting at the *right*. The assignment to macros provided as arguments in the latter case is done for two reasons. It is more efficient in this situation to make both assignments at once. More importantly, since `\wendcarcdr` uses recursion via the `\loop` construct, as well as defining temporary variables, TEX will complain if an attempt to force an immediate expansion of the result is made with `\edef`. Thus, `\wendcarcdr` uses `\xdef` internally to define `\A` and `\B`.

The problem of being unable to store and further manipulate the results of certain macro expansions can be solved in a second way. The `\StoreAddrBlock` macro mentioned earlier is defined by

```
\def\StoreAddrBlock#1\in#2{%
   {\setbox0=\hbox{\AddrBlock#1}%
    \toks0=\expandafter{\Current}%
    \xdef#2{\the\toks0}}}
```

In the first line, `\AddrBlock` completely expands within the `\hbox` and the result is assigned to a box which is subsequently ignored. As a side effect of the expansion, the global macro `\Current` is set to be the value of the reference to `\AddrBlock`. The last two lines of the macro then store this result in the second argument, operating with a token list to prevent premature expansion of any \\'s that may be present. The extra set of braces ensure that the assignments to `\box0` and `\toks0` are local to the macro.

This section concludes with a small, but important point. TEX will append an end-of-line character to any line of input text unless `\endlinechar=-1` is performed. This character will be treated like a space unless the line is blank, in which case it will be converted into a `\par`. Since `\ifx` performs one level of macro expansion on its arguments, one way to read a line from a file and test if it is blank is:

```
\read\file to \Line
\ifx\Line\blank
```

where `\def\blank{\par}` must be done somewhere previous.

Summary

We have designed a general-purpose form-letter generator that runs directly under TEX or LATEX. This program requires three files, supplied by the user: a (optional) preamble, a simple format address list, and a letter template. These files are prompted for interactively, and they default to certain names if none are specified. We have also developed various sample files, including a preamble and template

[1] The comment there incorrectly predicts that such a construction is "probably too lengthy to be of any use."

to produce 3-across labels. Thus, form letters and mailing labels can both be generated from the same address list by just changing two inputs to the program. Of course, there is nothing magical about names and addresses; this program can certainly be adapted to other uses.

This form-letter program has been designed to be both robust and easy to work with. The former means, for example, that excess white space and lines commented out with a % in the address list will not cause havoc. To make the program easy to work with, the files have been designed to be easily modified for various types of results. Thus, the files themselves have been made READABLE so that anyone who wishes to alter them may do so easily. These files will be made publicly available from the TEX repositories at `sun.soe.clarkson.edu` and `ymir.claremont.edu`.

Acknowledgments

Thanks are due to Ellen Golden and Sam Matthews for providing certain crucial help during the development of this program. Also, Tom Stickels is due thanks for the encouragement and sound editing advice provided throughout the development of this paper. Finally, hats off to the reading club for instigating this endeavor.

Bibliography

Knuth, Donald E. *The TEXbook*. Reading, Mass.: Addison-Wesley, 1984.

Lamport, Leslie. *LATEX: A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.

Microsoft Corporation. *Microsoft Word Processing Program Version 3.0 for IBM Personal Computers and Compatibles*. Redmond, Wash.: Microsoft Corporation, 1986.

Appendix: Selected Macro Definitions

```
\newif\ifnotdone
\newif\ifwmember
% ============================================
%
% Allow \par's within \loop constructs.
%
\long\def\loop#1\repeat{\def\body{#1}\iterate}
% --------------------------------------------
\def\AddrLine#1{\GetLine#1\of\Address}
%
\def\AddrBlock#1{%
   \GetBlock#1\of\Address\by{\\---\\}}
%
\def\GetLine#1\of#2{\GetBlock#1\of#2\by\\}
%
% \GetBlock{N}\of\List\by\Delim gets block N of
%    the \List.  The blocks in the \List are
%    assumed to be separated by \Delim's.
%    \Current will hold the last block
%    selected.
%
\def\Current{}
\def\GetBlock#1\of#2\by#3{%
   {\count0=#1
    \toks0=\expandafter{#2#3}%
    \edef\List{\the\toks0}%
    \def\lcar##1#3##2\nil{##1}%
    \def\lcdr##1#3##2\nil{##2}%
    \notdonetrue
    \loop
       \ifx\List\empty
          \notdonefalse
          \gdef\Current{}%
       \else
          \advance\count0 by -1
          \ifnum\count0=0
             \notdonefalse
             \toks0=\expandafter\expandafter
                       \expandafter{%
                          \expandafter
                             \lcar\List\nil}%
             \xdef\Current{\the\toks0}%
             \Current
          \fi
       \fi
       \ifnotdone
          \toks0=\expandafter\expandafter
                    \expandafter{%
                       \expandafter
                          \lcdr\List\nil}%
          \edef\List{\the\toks0}%
    \repeat}}
%
% \wendcarcdr\List\nil\A\B picks off the last
%    word in the \List and places it in \A.
%    The rest of the list (stripped of leading
%    blanks) is placed in \B.
%
\def\wendcarcdr#1\nil#2#3{%
   {\edef\List{#1}%
    \def\carList{}%
    \def\newList{}%
    %
    \notdonetrue
    \loop
       \ifnull\List
          \notdonefalse
          \xdef#2{\carList}%
          \xdef#3{\newList}%
       \else
          \ifx\List\space
             \notdonefalse
             \xdef#2{}%
```

```
            \xdef#3{}%
         \fi
      \fi
      \ifnotdone
         \ifx\newList\empty
            \edef\newList{\carList}%
         \else
            \edef\newList{%
               \newList\space\carList}%
         \fi
         \setq\carList{\wcar\List\nil}%
         \setq\List{\wcdr\List\nil}%
   \repeat}}
%
% \wmember\Element\of\List\nil causes
%    \ifwmember to be true if the \Element is a
%    member of the \List and false otherwise.
%
\def\wmember#1\of#2\nil{
   {\global\wmemberfalse
    \edef\Element{#1}%
    \edef\List{#2}%
    \setq\carList{\wcar\List\nil}%
    %
    \ifnull\Element
       \notdonefalse
    \else
       \notdonetrue
    \fi
    \loop
       \ifnull\carList
          \notdonefalse
       \fi
       \ifnotdone
          \ifx\Element\carList
             \notdonefalse
             \global\wmembertrue
          \else
             \setq\List{\wcdr\List\nil}%
             \setq\carList{\wcar\List\nil}%
          \fi
    \repeat}}
```