The 'arrayjobx' package
Management of arrays in (LA)TEX

Zhuhan Jiang
School of Computing and Mathematics
University of Western Sydney
Sydney
Australia
email: zhuhan@scm.uws.edu.au

(Documentation: Denis Girou* (CNRS/IDRIS – France) – Denis.Girou@idris.fr)

Version 1.04
May 3, 2010
Documentation revised May 31, 2000, edited lightly May 3, 2010 by Michael Sharpe

Abstract

This package provides array data structures in (LA)TEX, in the meaning of the classical procedural programming languages like Fortran, Ada or C, and macros to manipulate them. Arrays can be mono or bi-dimensional.

This is useful for applications which require high level programming techniques, like algorithmic graphics programmed in TEX. This version no longer conflicts with some macros in the `amsmath` package. It should be used for all new work.

Contents

1 Introduction **2**

2 Command reference **2**

3 Examples **7**
3.1 Basic examples 7
3.1.1 Plot labels 9
3.1.2 Checkboard drawing 10
3.2 Advanced examples 12
3.2.1 Example with recursion usage 12
3.2.2 Structured dynamic diagrams on a grid 13

*All errors and misunderstandings are mine.

3.2.3 Another example of a structured dynamic diagram on a grid 18
3.2.4 Management of heaps and linked lists 20
3.2.5 Associative arrays 29

4 Thanks **37**

1 Introduction

One of the big advantages of the (LA)TEX system over common interactive software for text processing is that it offer, also, a programming language, which give, to people who have some knowledge in the algorithmic and programming fields, an exceptional flexibility and power.

Nevertheless, TEX is a rather specific programming language, based on macro expansion, which implements a lot of unusual constructs but lacks some that are very familiar in the classical procedural languages, as *arrays* to store and retrieve arbitrary pieces of data, stored in a structured way. The main reason why they were not integrated in TEX was perhaps that arrays did not seem very useful in a language focussed on text processing (but METAFONT, for instance, does has them). Nevertheless, one of the few applications where this is straightforward to use them is to program a mailing system, where nearly the same information is to be formatted several times, depending of values which can be simply retrieved from an array of data. This was the goal of the 'formlett' package [7], written in 1993-1995 by Zhuhan Jiang for dealing with mass letters, invoices and similar tasks of some duplicative nature. It implemented a small but powerful set of macros to manage arrays, which have been extracted to form the present 'arrayjobx' package. The arrays can be mono or bi-dimensional[1] and they are dynamically allocated (so we do not have to declare their dimension statically.)[2].

Array structures are, at the opposite of text management, often very useful in graphic programming. This is why the (AI)DraTEX package from Eitan Gurari [4] (see also [5]) integrate such functionality (but which is bundled inside this package and could not be extracted easily from it), and this is also the case for the METAPOST package [6] (but this one do not use TEX as programming language)[3]. This fact explain why most of our examples in this documentation will concern the area of *graphic programming* (here using the PSTricks package from Timothy van Zandt [9]).

2 Command reference

`\newarray` : Define a new array.

Syntax: `\newarray\`⟨*ArrayName*⟩

Example: `\newarray\Values`

`\delarray` : Delete an array.

Syntax: `\delarray\`⟨*ArrayName*⟩

Example: `\delarray\Values`

Remarks:

[1] You can use more than two dimensions, but not in the meaning of the classical programming languages (see page 5)

[2] Stephan von Bechtolsheim [1, Volume III, paragraph 20.3, page 136] also demonstrated such macros for array management in the third volume of his huge book, but it was limited to mono-dimensional arrays.

[3] Another required feature, used in conjunction with the managements of arrays, is a generic loop mechanism. TEX offer the `\loop` macro and LATEX the `\whiledo` macro of the 'ifthen' package (and also the internal `\@for`, `\@whilenum`, etc. macros), but a more high-level structure is to be preferred, as the ones defined too in the (AI)DraTEX or METAPOST packages. We will use in our examples the 'multido' package [8], also written by Timothy van Zandt, but others are available, like the 'repeat' package written by Victor Eijkhout [2].

1. obviously, elements of a deleted array could not be accessed later,
2. take care that the elements of a deleted array are not themselves deleted.
   So, `\delarray\Data \newarray\Data \Data(I)` can produce strange behavior. Just avoid reusing deleted arrays.

`\ArrayName` : Store or get the content of the array element specified by the index/indices. In the last case, the content is inserted at the current point.

Syntax: `\`⟨*ArrayName*⟩`(I)={`⟨*Content*⟩`}` or more generally
`\`⟨*ArrayName*⟩`(I1,...,In)={`⟨*Content*⟩`}` to store a value

`\`⟨*ArrayName*⟩`(I)` or more generally `\`⟨*ArrayName*⟩`(I1,...,In)` to retrieve a value

Examples:
```
\Data(6)={Paul Adams}
\Values(19)
```

`\readarray` : Store consecutive values in an array, starting from the indice one.

Syntax: `\readarray{`⟨*ArrayName*⟩`}={`⟨*Content1*⟩`&...&`⟨*ContentM*⟩`}`

Example: `\readarray{Actors}{Louise Brooks&Marlene Dietrich&Clark Gable}`

Remarks:

1. the values must be separated by the **&** character,
2. take care that the trailing spaces are significant, so the previous definition is different from the following one:
   `\readarray{Actors}{Louise Brooks & Marlene Dietrich & Clark Gable}`
3. you can use (LA)TEX macros inside the values.

`\Values(3)` = 'C'
`\Actors(2)` = 'Marlene Dietrich'
`\Actors(2)` = ' Marlene Dietrich '
`\Actors(4)` = '*Ida Lupino*'

```
\newarray\Values
\readarray{Values}{A&B&C&D}
\verb+\Values(3)+ = `\Values(3)'
\newarray\Actors
\readarray{Actors}{Louise Brooks&Marlene Dietrich&Clark Gable}
\verb+\Actors(2)+ = `\Actors(2)'
\readarray{Actors}{%
Louise Brooks & Marlene Dietrich & Clark Gable}
\verb+\Actors(2)+ = `\Actors(2)'
\Actors(4)={\textit{\underline{Ida Lupino}}}
\verb+\Actors(4)+ = `\Actors(4)'
```

If you really need to trim the unnecessary left and right spaces, you must apply a special action, like this one:

```
\def\BS{\texttt{\symbol{`\\}}}

\newarray\Strings
\readarray{Strings}{a& b&c c  &    d dd  ddd   }

\multido{\iString=1+1}{4}{%
```

```
  \checkStrings(\iString)%
  \BS\texttt{Strings(\iString)}=‘\cachedata’\qquad}

\makeatletter

% A \TrimSpaces macro adapted from Michael J. Downes <epsmjd@ams.org>
% (posted on c.t.t. June 19, 1998)
% \number‘\x reads past one following space (expanding as it goes)
\long\def\TrimSpaces#1{\expandafter\TrimSpaces@i\number‘\^^00#1| |}
% Remove the "0" produced by \number‘\^^00, and " |" at the end.
\long\def\TrimSpaces@i 0#1 |{\TrimSpaces@ii\empty#1|}
% " |" was removed by \TrimSpaces@i, now remove a trailing "||" or "| |"
\long\def\TrimSpaces@ii #1|#2|{#1}

\makeatother

\multido{\iString=1+1}{4}{%
  \checkStrings(\iString)%
  \BS\texttt{Strings(\iString)}=‘\TrimSpaces{\cachedata}’\qquad}
```

\Strings(1)=‘a’ \Strings(2)=‘ b’ \Strings(3)=‘c c ’ \Strings(4)=‘ d dd ddd ’
\Strings(1)=‘a’ \Strings(2)=‘b’ \Strings(3)=‘c c’ \Strings(4)=‘d dd ddd’

`\check` : Get the content of the array element specified by the indice(s) and store the result in the macro `\cachedata`

Syntax: `\check`⟨*ArrayName*⟩`(I)` or more generally `\check`⟨*ArrayName*⟩`(I1,..,In)`

Example: `\checkActors(2)`

`\cachedata` : Macro where the content is stored after a `\check` request.

`\ifemptydata` : True if the last `\check` request has given an empty result.

\Values(2) = ‘B’
\Actors(3) = ‘Clark Gable’
\Actors(5) not defined.
\Actors(3) = ‘Clark Gable’
\Actors(5) not defined

```
% Plain TeX usage
\checkValues(2)%
\verb+\Values(2)+ = ‘\cachedata’
\checkActors(3)%
\verb+\Actors(3)+ = ‘\cachedata’
\checkActors(5)%
\ifemptydata
  \verb+\Actors(5)+ not defined.
\fi
% LaTeX usage
\newcommand{\IsEmptyElement}[2]{%
\ifthenelse{\boolean{emptydata}}{#1}{#2}}
\checkActors(3)%
\verb+\Actors(3)+ = \IsEmptyElement{not defined}{‘\cachedata’}
\checkActors(5)%
\verb+\Actors(5)+ \IsEmptyElement{not defined}{‘\cachedata’}
```

\ifnormalindex : See below (Default: \normalindexfalse).

\dataheight : Counter containing the number of elements in the first dimension, if arrays are bi-dimensional.

Syntax: \dataheight=⟨*Number*⟩

Remarks:

1. arrays are monodimensional when \dataheight $\leq$ 1,
2. if \normalindexfalse (which is the default value), we have:

$$\backslash\langle ArrayName\rangle(I_1,...,I_n) = \backslash\langle ArrayName\rangle(I_n+(I_{n-1}-1)*\texttt{\dataheight}+\cdots+(I_1-1)*\texttt{\dataheight}^{n-1})$$

and if \normalindextrue, we have:

$$\backslash\langle ArrayName\rangle(I_1,...,I_n) = \backslash\langle ArrayName\rangle(I_1+(I_2-1)*\texttt{\dataheight}+\cdots+(I_n-1)*\texttt{\dataheight}^{n-1})$$

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J

\Letters(1,2)=‘B’
\Letters(2,1)=‘F’

	1	2
1	A	F
2	B	G
3	C	H
4	D	I
5	E	J

\Letters(1,2)=‘F’
\Letters(2,1)=‘B’

```
\newarray\Letters
\readarray{Letters}{A&B&C&D&E&F&G&H&I&J}
\dataheight=5

% Default is \normalindexfalse
\verb+\Letters(1,2)=+‘\Letters(1,2)’
\verb+\Letters(2,1)=+‘\Letters(2,1)’
\normalindextrue
\verb+\Letters(1,2)=+‘\Letters(1,2)’
\verb+\Letters(2,1)=+‘\Letters(2,1)’
```

`\ifexpandarrayelement`: Boolean macro to allow or not the element to be evaluated before to be stored in the array (Default: `\expandarrayelementfalse`).

Syntax: `\expandarrayelementtrue` or `\expandarrayelementfalse`

Remark: take care to the possible side effects if you store some macros as values of some array elements without evaluating them, as they can change of content later... (see the following examples).

```
\newarray\Data
\newcount\CounterP            % Plain TeX usage
\CounterP=3
\newcounter{CounterL}         % LaTeX usage
\setcounter{CounterL}{3}
\def\Town{Madrid}
\Data(1)={\the\CounterP}
\Data(2)={\the\value{CounterL}}
\Data(3)={\Town}

\expandarrayelementtrue
\Data(4)={\the\CounterP}
\Data(5)={\the\value{CounterL}}
\Data(6)={\Town}

\CounterP=5
\setcounter{CounterL}{5}
\def\Town{Roma}

\multido{\iData=1+1}{6}{%
  \BS\texttt{Data(\iData)}=‘\Data(\iData)’\\}
```

`\Data(1)`=‘5’
`\Data(2)`=‘5’
`\Data(3)`=‘Roma’
`\Data(4)`=‘3’
`\Data(5)`=‘3’
`\Data(6)`=‘Madrid’

Some other macros exists for mono-dimensional arrays (and only for them), but have little additional interest:

`\arrayx` : Store or get the content of the array element specified by the index. In the last case, the content is inserted at the current point.[4]

Syntax: `\arrayx{`⟨*ArrayName*⟩`}(I)={`⟨*Content*⟩`}` to store a value

`\arrayx{`⟨*ArrayName*⟩`}(I)` to retrieve a value

Examples: `\arrayx{Actors}(6)={Joan Crawford}`

`\arrayx{Actors}(3)`

`\clrarray` : Clear the content of the array element specified. A following inquiry on this element will give an empty content.

Syntax: `\clrarray{`⟨*ArrayName*⟩`}(I)`

Example: `\clrarray{Actors}(2)`

[4]The macro name `\array` conflicted with macros in `amsmath`, and is changed as of version 1.04. (Modification suggested by Michael Sharpe.)

`\testarray` : Get the content of the array element specified by the index and store the result in the macro `\temp@macro`[5].

Syntax: `\testarray{`⟨*ArrayName*⟩`}(I)`

Example: `\testarray{Actors}(1)\typeout{Actors(1)=‘\temp@macro’}`

3 Examples

We will first show some basic and easy examples, before looking at more advanced ones to solve some complex problems, which require more knowledge of TEX programming techniques.

3.1 Basic examples

The immediate thing for which we can use arrays is to store and retrieve information:

```
\newarray\Actors
\newarray\Dates
\newarray\Sexes

\readarray{Actors}{Louise Brooks&Marlene Dietrich&Clark Gable}
\readarray{Dates}{1906--1985&1902--1992&1901--1960}
\readarray{Sexes}{2&2&1}

\begin{description}
  \item[\Actors(1)] : \Dates(1)
  \item[\Actors(2)] : \Dates(2)
  \item[\Actors(3)] : \Dates(3)
\end{description}
```

Louise Brooks : 1906–1985

Marlene Dietrich : 1902–1992

Clark Gable : 1901–1960

But we can also use a general loop macro, as the one provided by the ‘multido’ package, for more powerful usage and a management independent of the number of elements:

```
\newcommand{\NumberActors}{3}

\begin{description}
  \multido{\iActor=1+1}{\NumberActors}{%
    \item[\Actors(\iActor)] : \Dates(\iActor)}
\end{description}
```

Louise Brooks : 1906–1985

Marlene Dietrich : 1902–1992

Clark Gable : 1901–1960

This allow various usage in the formatting of texts. A common usage is for a *mailing* process, when we must compose some similar letters to various people (as we said previously, this was the reason for which these macros were developed in the ‘formlett’ package). Here, the usage of a programming language allow a great flexibility, using some conditionals to format the text differently or even to insert different pieces of text according to the person:

```
\newcounter{iActor}

```

[5] In LATEX, this macro should be used inside a package or between the `\makeatletter` ··· `\makeatother` macro pair.

```
\newcommand{\AccordingSexe}[2]{%
\checkSexes(\the\value{iActor})%
\ifthenelse{\equal{\cachedata}{1}}{#1}{#2}}

\whiledo{\value{iActor} < \NumberActors}{%
  \stepcounter{iActor}%
  \fbox{%
    \begin{minipage}{0.985\textwidth}
      Dear \AccordingSexe{Mr}{Mrs} \Actors(\the\value{iActor}),

      I would like to tell you how I admire the great \AccordingSexe{actor}{actress}
      you are, etc.
    \end{minipage}}\\[5mm]}
```

Dear Mrs Louise Brooks,
I would like to tell you how I admire the great actress you are, etc.

Dear Mrs Marlene Dietrich,
I would like to tell you how I admire the great actress you are, etc.

Dear Mr Clark Gable,
I would like to tell you how I admire the great actor you are, etc.

Nevertheless, people who know a little TEX as a programming language know that it behaviour is full of pitfalls... For instance, the following example, which format a table according to the content of entries stored in external arrays, can't work:

```
\begin{tabular}{|l|c|}
  \hline
  \multicolumn{1}{|c}{\textbf{Actors}} & \multicolumn{1}{|c|}{\textbf{Dates}} \\ \hline
  \multido{\iActor=1+1}{\NumberActors}{\Actors(\iActor) & \Dates(\iActor) \\ \hline}
\end{tabular}
```

This is because there is an implicit grouping of each entry in a tabular environment and that it do not work with the grouping of the `\multido` loop. So, this must be program in a different way, storing all the content of the table before really inserting it:

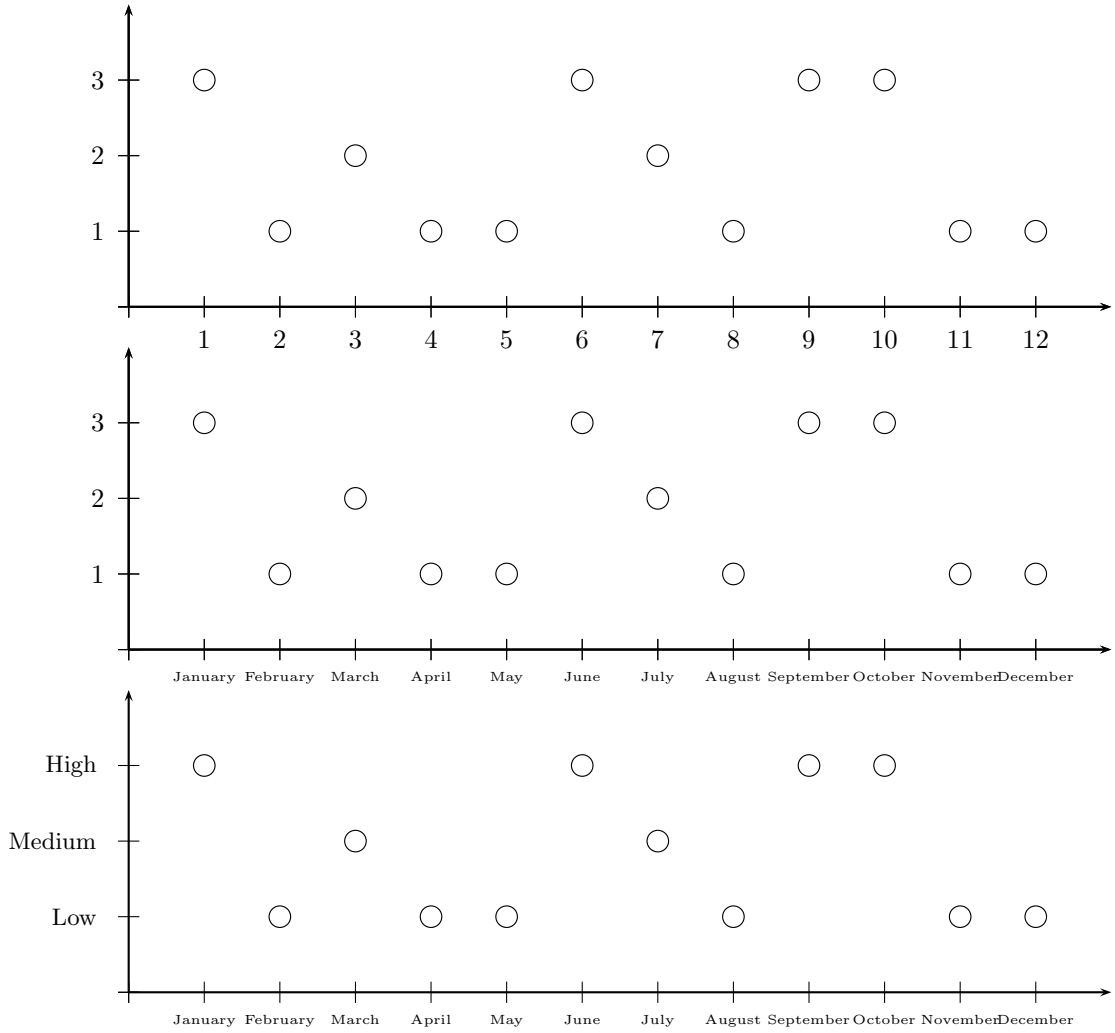
Actors	Dates
Louise Brooks	1906–1985
Marlene Dietrich	1902–1992
Clark Gable	1901–1960

```
\begin{tabular}{|l|c|}
  \hline
  \multicolumn{1}{|c}{\textbf{Actors}} &
    \multicolumn{1}{|c|}{\textbf{Dates}} \\ \hline
  \let\ListActors\empty
  \begingroup
    \let\Actors\relax
    \let\Dates\relax
    \let\\\relax
    \let\hline\relax
    \multido{\iActor=1+1}{\NumberActors}{%
      \xdef\ListActors{\ListActors
        \Actors(\iActor) & \Dates(\iActor) \\ \hline}}
  \endgroup
  \ListActors
\end{tabular}
```

3.1.1 Plot labels

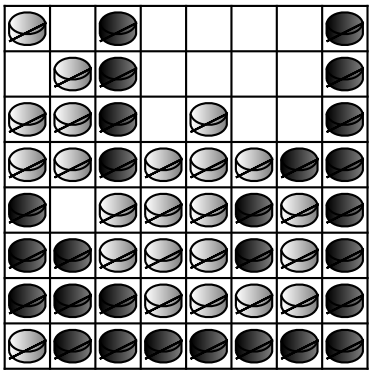
A basic usage in graphic programming is to use arrays to retrieve the labels to put on a drawing, as here to label the simple plot below. (Note the change from the original documentation. The axis labels used with `\psaxes` must now be numeric, so it is necessary to bypass the automatic axis labeling.)

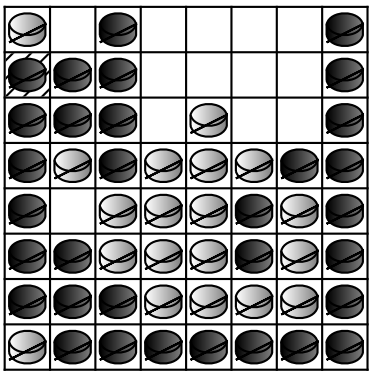
```
\savedata{\Data}[(1,3)(2,1)(3,2)(4,1)(5,1)(6,3),(7,2)(8,1)(9,3)(10,3)(11,1)(12,1)]

\newcommand{\PlotData}[1]{%
\begin{pspicture}(-0.5,-0.5)(13,4)
  \psaxes[labels=none]{->}(13,4)
  \dataplot[plotstyle=dots,dotstyle=o,dotsize=0.3]{\Data}
  #1\end{pspicture}}
\PlotData{\psaxes[showorigin=false]{->}(13,4)}

\newarray{\Months}
\readarray{Months}{January&February&March&April&May&June&July&August&September&%
October&November&December}
\newarray{\Levels}
\readarray{Levels}{Low&Medium&High}
%\renewcommand{\pshlabel}[1]{\tiny\Months(#1)}
\PlotData{\psaxes[labels=y,showorigin=false]{->}(13,4)%
\multido{\i=1+1}{12}{\rput[B](\i,-12pt){\tiny\Months(\i)}}}

%\renewcommand{\psvlabel}[1]{\small\Levels(#1)}
\PlotData{\multido{\i=1+1}{12}{\rput[B](\i,-12pt){\tiny\Months(\i)}}%
\multido{\i=1+1}{3}{\rput[r](-12pt,\i){\small\Levels(\i)}}}
```



3.1.2 Checkboard drawing

Of course, a bi-dimensional array directly allow to store the state of a checkboard of one of the numerous ones existing, like the reversi game that we will show here (in fact, the only more difficult part to understand in this example concern the definition of the elementary piece, but as this is not related at all to the usage of 'arrayjobx', this point can be forgotten by most of the readers).





```
\def\Black{B}
\def\White{W}

\def\Piece#1#2{{%
\psset{unit=0.8}%
\pspicture(0,-0.1)(1,0.8)
  \pscustom[fillstyle=gradient,gradmidpoint=0,gradangle=90,
            gradbegin=#2,gradend=#1]{%
    \psbezier(0,0.2)(0.1,-0.2)(0.9,-0.2)(1,0.2)
    \psline(1,0.2)(1,0.5)
    \psbezier(1,0.5)(0.9,0.9)(0.1,0.9)(0,0.5)
    \psline(0,0.5)(0,0.2)}
    \psbezier(0,0.5)(0.1,0.1)(0.9,0.1)(1,0.5)
\endpspicture}}

\def\PieceBlack{\Piece{black}{gray}}
\def\PieceWhite{\Piece{white}{gray}}

\def\CheckboardHook{}

\newarray\Checkboard

\def\ShowCheckboard{%
\dataheight=8
\pspicture(8,8)
  \psgrid[subgriddiv=0,gridlabels=0](8,8)
  \CheckboardHook
  \multido{\iColumn=1+1,\iColumnPos=0+1}{8}{%
    \multido{\iRow=1+1,\iRowPos=7+-1}{8}{%
      \checkCheckboard(\iRow,\iColumn)%
      \ifx\cachedata\Black
        \rput(\iColumnPos.5,\iRowPos.5){\PieceBlack}
      \else
        \ifx\cachedata\White
          \rput(\iColumnPos.5,\iRowPos.5){\PieceWhite}
        \fi
      \fi}}
\endpspicture}

\readarray{Checkboard}{%
W& &B& & & & &B&%
 &W&B& & & & &B&%
W&W&B& &W& & &B&%
W&W&B&W&W&W&B&B&%
B& &W&W&W&B&W&B&%
B&B&W&W&W&B&W&B&%
B&B&B&W&W&W&W&B&%
W&B&B&B&B&B&B&B&%
}

\psset{unit=0.6}
\ShowCheckboard

\Checkboard(2,1)={B} % Black move
\def\CheckboardHook{%
\psframe[linestyle=none,fillstyle=hlines](0,6)(1,7)}
% White pieces changed by the black move
\Checkboard(3,1)={B}\Checkboard(4,1)={B}
\Checkboard(2,2)={B}\Checkboard(3,2)={B}
\vspace{1cm}
\ShowCheckboard
```

Note also that, if TEX is obviously not well suited to implement a program to *really* play at a game like reversi or chess, nevertheless it can be used to solve internally some non trivial algorithmic problems, where the usage of arrays help a lot. For such example (on the coloration of the Truchet's tiling), see [3].

3.2 Advanced examples

This section will show slightly more difficult to really complex examples, and is mainly for *advanced users* or for people who want to be able to program complex tasks themselves.

3.2.1 Example with recursion usage

```
\makeatletter

% The recursion macro used (from David Carlisle)
\def\Recursion#1{%
#1\relax
  \expandafter\@firstoftwo
\else
  \expandafter\@secondoftwo
\fi}

\newcount\IndexRecursion
\IndexRecursion=\z@

\def\PiFrac#1{{%
\Recursion
  {\ifnum#1>\@ne\relax}
  {\@tempcnta=#1
   \advance\@tempcnta\m@ne
   \advance\IndexRecursion\@ne
   \PiValues(\IndexRecursion)
     +\frac{\strut\displaystyle 1}{\strut\displaystyle\PiFrac{\@tempcnta}}}
  {\advance\IndexRecursion\@ne
   \PiValues(\IndexRecursion)}}}

\makeatother

\newarray\PiValues
\readarray{PiValues}{3&7&15&1&292&1&1&1&2&1&3&1&14}

\(\pi\approx\PiFrac{4}\approx\PiFrac{13}\)

\textbf{\(\pi\) as a continued fraction}
```

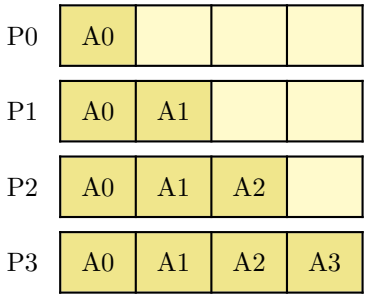
$$\pi \approx 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1}}} \approx 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{14}}}}}}}}}}}}$$

π as a continued fraction

3.2.2 Structured dynamic diagrams on a grid

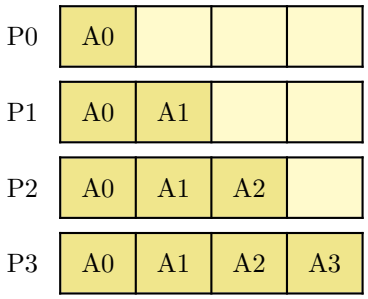
A very common case is when we must use some strutured data which are to be drawn on a kind of abstract grid.

To draw a diagram like this one (such example illustrate the exchanges of data between processors when using applications on multiprocessors computers in parallel programming):




```
\psset{linestyle=none,fillstyle=solid,subgriddiv=0,gridlabels=0}
\begin{pspicture}(-1,0)(4,4)
 % Line 0
 \psframe[fillcolor=Khaki](0,3.1)(1,3.9)
 \psframe[fillcolor=LemonChiffon](1,3.1)(4,3.9)
 \rput(0,3.1){\psgrid[yunit=0.8](4,1)}
 \rput(-0.5,3.5){P0}
 \rput(0.5,3.5){A0}
 % Line 1
 \psframe[fillcolor=Khaki](0,2.1)(2,2.9)
 \psframe[fillcolor=LemonChiffon](2,2.1)(4,2.9)
 \rput(0,2.1){\psgrid[yunit=0.8](4,1)}
 \rput(-0.5,2.5){P1}
 \rput(0.5,2.5){A0}\rput(1.5,2.5){A1}
 % Line 2
 \psframe[fillcolor=Khaki](0,1.1)(3,1.9)
 \psframe[fillcolor=LemonChiffon](3,1.1)(4,1.9)
 \rput(0,1.1){\psgrid[yunit=0.8](4,1)}
 \rput(-0.5,1.5){P2}
 \rput(0.5,1.5){A0}\rput(1.5,1.5){A1}\rput(2.5,1.5){A2}
 % Line 3
 \psframe[fillcolor=Khaki](0,0.1)(4,0.9)
 \rput(0,0.1){\psgrid[yunit=0.8](4,1)}
 \rput(-0.5,0.5){P3}
 \rput(0.5,0.5){A0}\rput(1.5,0.5){A1}
   \rput(2.5,0.5){A2}\rput(3.5,0.5){A3}
\end{pspicture}
```

is easy but rather painful, as we must manage a lot of coordinates. Nevertheless, even if we introduce a little more abstraction level using some minor programming with a loop structure:

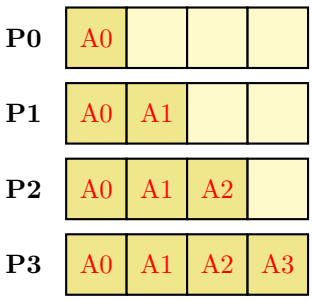



```
\psset{linestyle=none,fillstyle=solid,subgriddiv=0,gridlabels=0}
\begin{pspicture}(-1,0)(4,4)
 \multido{\iLoop=1+1,\iRow=0+1,\iRowPos=3+-1}{4}{%
   \psframe[fillcolor=Khaki](0,\iRowPos.1)(\iLoop,\iRowPos.9)
   \psframe[fillcolor=LemonChiffon]
           (\iLoop,\iRowPos.1)(4,\iRowPos.9)
   \rput(0,\iRowPos.1){\psgrid[yunit=0.8](4,1)}
   \rput(-0.5,\iRowPos.5){P\iRow}
   \multido{\iColumn=0+1,\iLabel=1+1}{\iLoop}{%
     \rput(\iColumn.5,\iRowPos.5){A\iColumn}}}
\end{pspicture}
```

we succeed to heavily reduce the number of lines of the code in this specific case, but we gain nothing in genericity. If we have several such diagrams to draw, with each that must be done differently according to the empty and full cells of the array, we must proceed in a completely different way, defining a *generic object* which can obtain the different values of a data structure previously filled. For such tasks, the 'arrayjobx' package allow to build very efficient and

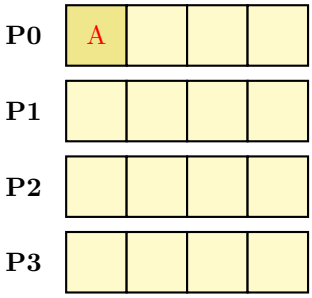
powerful tools.

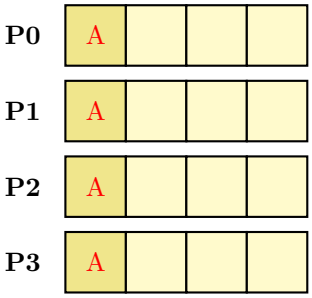
```
% The grid
\def\ArrayGrid#1#2#3{%
% #1=array of data, #2=number of rows, #3=number of columns
\psset{dimen=middle,xunit=0.8}
\dataheight=#3
\pspicture(-1,-#2)(#3,0)
\multido{\iRow=1+1,\iRowMinusOne=0+1}{#2}{%
  \rput(-1,-\iRow){\LineGrid{#1}{\iRow}{\iRowMinusOne}{#3}}}
\endpspicture}}

% One line of the grid
\def\LineGrid#1#2#3#4{%
\rput(0.5,0.4){\AttributeLabel{P#3}}
\multido{\iColumn=1+1}{#4}{%
  \expandafter\csname check#1\endcsname(#2,\iColumn)%
  \rput(\iColumn,0){%
    \ifemptydata
      \def\ColorBackground{\ColorBackgroundEmpty}%
    \else
      \ifx\cachedata\space
        \def\ColorBackground{\ColorBackgroundEmpty}%
      \else
        \def\ColorBackground{\ColorBackgroundFull}%
      \fi
    \fi
    \psframe[fillstyle=solid,fillcolor=\ColorBackground](1,0.8)
    \rput(0.5,0.4){\AttributeElement{\cachedata}}}}}
```




```
% Default attributes
\def\ColorBackgroundEmpty{LemonChiffon}
\def\ColorBackgroundFull{Khaki}
\def\AttributeElement#1{\textcolor{red}{#1}}
\def\AttributeLabel#1{\bf #1}

% Example 1
\newarray\ArrayA

% Data
\readarray{ArrayA}{%
A0&  &  &  &%
A0&A1&  &  &%
A0&A1&A2&  &%
A0&A1&A2&A3}

\ArrayGrid{ArrayA}{4}{4}
```

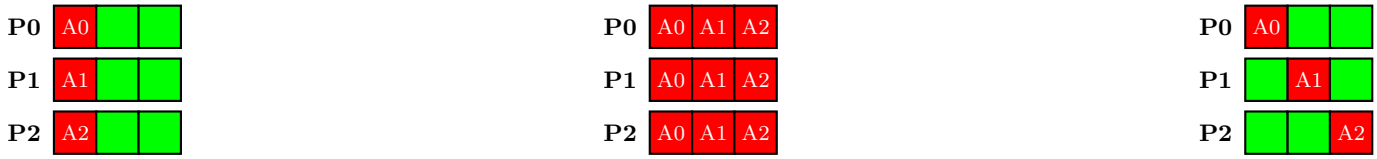
```
% Example 2
\newarray\ArrayB

% Data
\readarray{ArrayA}{%
A& & & &%
 & & & &%
 & & & &%
 & & & }
\readarray{ArrayB}{%
A& & & &%
A& & & &%
A& & & &%
A& & & }

\ArrayGrid{ArrayA}{4}{4}\hfill\ArrayGrid{ArrayB}{4}{4}
```



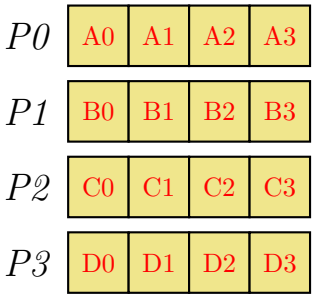


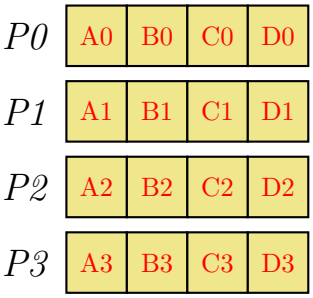

```
% Example 3
\newarray\ArrayC
```

```
%
% Data
\readarray{ArrayA}{%
A0&  &  &%
A1&  &  &%
A2&  &  }
\readarray{ArrayB}{%
A0&A1&A2&%
A0&A1&A2&%
A0&A1&A2}
\readarray{ArrayC}{%
A0&  &  &%
  &A1&  &%
  &  &A2}

% Attributes
\def\ColorBackgroundEmpty{green}
\def\ColorBackgroundFull{red}
\def\AttributeElement#1{\footnotesize\textcolor{white}{#1}}
\def\AttributeLabel#1{\small\bf #1}

\psset{unit=0.7}
\ArrayGrid{ArrayA}{3}{3}\hfill\ArrayGrid{ArrayB}{3}{3}\hfill\ArrayGrid{ArrayC}{3}{3}
```




```
% Example 4

% Data
\readarray{ArrayA}{%
A0&A1&A2&A3&%
B0&B1&B2&B3&%
C0&C1&C2&C3&%
D0&D1&D2&D3}
\readarray{ArrayB}{%
A0&B0&C0&D0&%
A1&B1&C1&D1&%
A2&B2&C2&D2&%
A3&B3&C3&D3}

% Attributes
\def\AttributeElement#1{\textcolor{red}{#1}}
\def\AttributeLabel#1{\Large\it #1}

\ArrayGrid{ArrayA}{4}{4}\hfill\ArrayGrid{ArrayB}{4}{4}
```



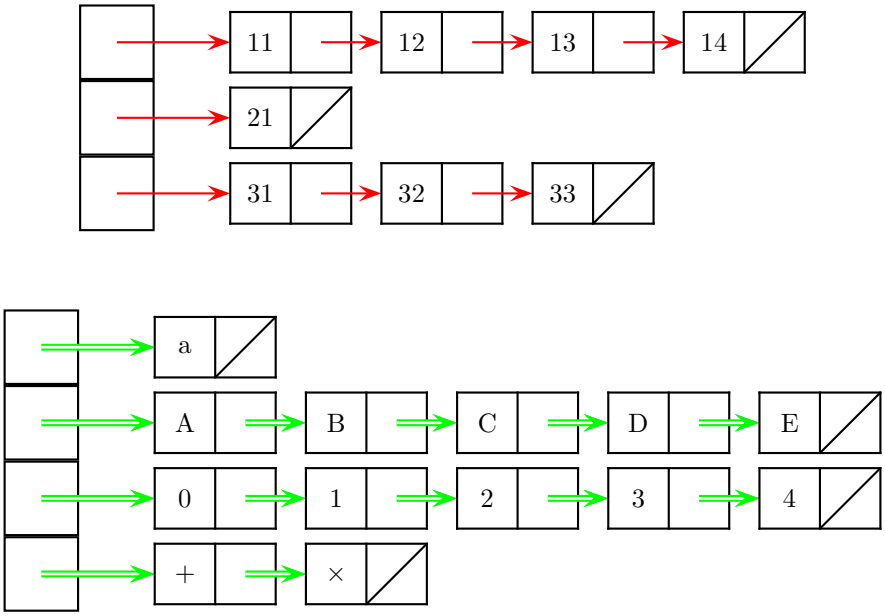


3.2.3 Another example of a structured dynamic diagram on a grid

The preceding technique is in fact relevant for a lot of problems. Here we give another example about the drawing of linked lists (but do not fail to note that these ones have here no internal existence).

```
\makeatletter

\newpsstyle{LinkedListsArrowStyle}{linecolor=red,arrowscale=2} % Default arrow style

\def\LinkedListsDraw{\@ifnextchar[\LinkedListsDraw@i{\LinkedListsDraw@i[]}}

\def\LinkedListsDraw@i[#1]#2#3{{%
\setkeys{psset}{#1}
\dataheight=#3
%
\pst@cnta=#3
\multiply\pst@cnta\tw@
\advance\pst@cnta\tw@
\pspicture(0,-#2)(\pst@cnta,0)
  \multido{\iRow=1+1}{#2}{%
    \rput(0,-\iRow){%
      \psframe(1,1)
      \psline[dimen=middle,style=LinkedListsArrowStyle]{->}(0.5,0.5)(2,0.5)}
    \multido{\iColumn=1+1}{#3}{%
      \checkLinkedListsTable(\iRow,\iColumn)%
      \ifemptydata
      \else
        \pst@cnta=\iColumn
        \advance\pst@cnta\@ne
        \pst@cntb=\@ne
        \checkLinkedListsTable(\iRow,\pst@cnta)%
        \ifemptydata                    % We test if it is empty
        \else
```

```
          \ifnum\multidocount=#3          % We test if it is the last one
          \else
            \pst@cntb=\z@
          \fi
        \fi
        %
        \pst@cnta=\iColumn
        \multiply\pst@cnta\tw@
        \rput(\the\pst@cnta,-\iRow){%
          \LinkedListsDraw@ii{\LinkedListsTable(\iRow,\iColumn)}{\the\pst@cntb}}
    \fi}}
  \endpspicture}}%}

\def\LinkedListsDraw@ii#1#2{{%
\rput(0,0.1){%
  \psset{unit=0.8}
  \psgrid[subgriddiv=0,gridlabels=0](2,1) % Element
  \rput(0.5,0.5){#1}                      % Label for this element
  \ifnum#2=\@ne
    \psline(1,0)(2,1)                     % Mark of the end of the list
  \else
    \psline[style=LinkedListsArrowStyle]{->}(1.5,0.5)(2.5,0.5) % Link to the next element
  \fi}}}

\makeatother

\newarray\LinkedListsTable
\readarray{LinkedListsTable}{%
11&12&13&14&%
21&&&&%
31&32&33&&}

\LinkedListsDraw{3}{4}

\vspace{1cm}
\newpsstyle{LinkedListsArrowStyle}{linecolor=green,doubleline=true}

\readarray{LinkedListsTable}{%
a&&&&&%
A&B&C&D&E&%
0&1&2&3&4&%
+&$\times$&&&&}

\LinkedListsDraw{4}{5}
```



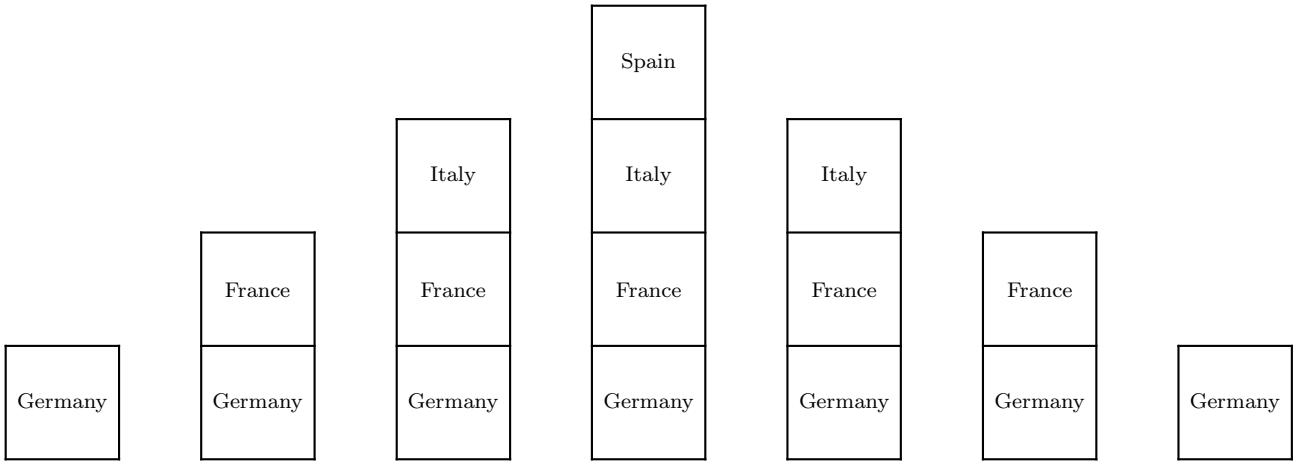
3.2.4 Management of heaps and linked lists

Here we will demonstrate a really more complex usage, but very useful. In fact, if the 'arrayjobx' package primarily allow to manage only arrays, it allow a lot more with some efforts. Above it, we can implement the common internal data structures frequently used in programming, like heaps, simple or double linked lists, trees, associative arrays, etc. Here we will show how to define macros to build and manage heaps and simple linked lists.

```
\makeatletter

\def\PstDebug{\z@}              % For debugging, set \def\PstDebug{1}

% Code for heaps management
% -------------------------

\def\HeapMaxDepth{100}          % No more than 100 elements in the heap

\newarray\Heap

% Add an element at top of the heap
\def\HeapPush#1{%
\multido{\iElem=1+1}{\HeapMaxDepth}{%
  \checkHeap(\iElem)%
  \ifemptydata
    \ifnum\PstDebug=\@ne\typeout{Push Heap(\iElem)=#1}\fi% Debug
```

```
    \Heap(\iElem)={#1}%
    \multidostop
  \fi}}

% Get (in the \cachedata macro) and delete the element at top of the heap
\def\HeapPop{%
\Multido{\iElem=\HeapMaxDepth+-1}{\HeapMaxDepth}{%
  \checkHeap(\iElem)%
  \ifemptydata
  \else
    \ifnum\PstDebug=\@ne\typeout{Delete Heap(\iElem)=\cachedata}\fi% Debug
    \Heap(\iElem)={}%
    \multidostop
  \fi}}

% Print the current state of the heap
\def\HeapPrint{%
\checkHeap(\@ne)%
\ifemptydata
  \typeout{The heap is empty!}% Empty heap
\else
  \multido{\iElem=\HeapMaxDepth+-1}{\HeapMaxDepth}{%
    \checkHeap(\iElem)%
    \ifemptydata
    \else
      \typeout{Heap(\iElem)=\cachedata}%
    \fi}
\fi
\typeout{}}

% Draw the current state of the heap
\def\HeapDraw{%
% To compute the size of the picture
\Multido{\iSize=0+1}{\HeapMaxDepth}{%
  \checkHeap(\the\multidocount)%
  \ifemptydata
    \multidostop
  \fi}
%
\ifnum\iSize=\z@
  % Empty heap
  \typeout{The heap is empty!^^J}%
\else
  \pspicture(1,\iSize)
    \psgrid[subgriddiv=0,gridlabels=0](1,\iSize)
    \multido{\iPos=0+1}{\HeapMaxDepth}{%
      \checkHeap(\the\multidocount)%
```

```
      \ifemptydata
        \multidostop
      \else
        \rput(0.5,\iPos.5){\cachedata}
      \fi}
  \endpspicture%
\fi}

\makeatother

\psset{unit=1.5}\footnotesize

\HeapPush{Germany}\HeapPrint\HeapDraw
\hfill
\HeapPush{France}\HeapPrint\HeapDraw
\hfill
\HeapPush{Italy}\HeapPrint\HeapDraw
\hfill
\HeapPush{Spain}\HeapPrint\HeapDraw
\hfill
\HeapPop\typeout{Popped element: `\cachedata'}\HeapPrint\HeapDraw
\hfill
\HeapPop\typeout{Popped element: `\cachedata'}\HeapPrint\HeapDraw
\hfill
\HeapPop\typeout{Popped element: `\cachedata'}\HeapPrint\HeapDraw
\hfill
\HeapPop\typeout{Popped element: `\cachedata'}\HeapPrint\HeapDraw
```



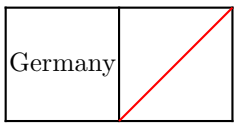

```
\makeatletter

\def\PstDebug{\z@}%                    For debugging, set \def\PstDebug{1}
```

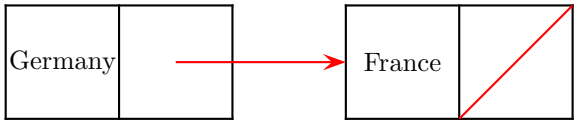
```
4
5  % Code for linked lists management
6  % --------------------------------
7
8  \def\LinkedListMaxDepth{100}%          No more than 100 elements in the list
9
10 \newarray\LinkedList
11
12 \dataheight=2%                         Two cells by element: the content and the pointer
13
14 % Add an element at the end of the list
15 % In fact, we define it here as a simple heap, but the difference is that
16 % we will be able to get and to delete any element in it
17 \def\LinkedListAdd#1{%
18 \edef\@tempa{\@ne}% Pointer initialization
19 \multido{\iElem=2+1}{\LinkedListMaxDepth}{%
20   \checkLinkedList(\@tempa,2)%        We got the content of this element
21   \ifemptydata
22     % No pointed element, so we will insert the new one here
23     \LinkedList(\iElem,1)={#1}%
24     % We update the pointer for the last preceding element
25     \expandarrayelementtrue%          We must evaluate the content of the counter
26     \LinkedList(\@tempa,2)={\iElem}%
27     \expandarrayelementfalse
28     \ifnum\PstDebug=\@ne
29       \typeout{Add LinkedList(\@tempa)->\iElem}% Debug
30       \typeout{\space\space\space\space LinkedList(\iElem)=#1}% Debug
31     \fi
32     \multidostop
33   \else
34     \edef\@tempa{\cachedata}%
35   \fi}}
36
37 % Get (in the \cachedata macro) the next element in the list
38 \def\LinkedListGetNext#1{%
39 \checkLinkedList(1,2)%                 We got the pointer for the first element
40 \edef\@tempa{\@ne}%
41 \ifx\cachedata\empty
42 \else
43   \edef\@tempc{#1}%
44   \Multido{}{\LinkedListMaxDepth}{%
45     \edef\@tempb{\@tempa}%
46     \edef\@tempa{\cachedata}%
47     \checkLinkedList(\cachedata,1)%   We got the pointed element
48     \ifx\cachedata\@tempc
49       % This is the element for which we want the next one: we got it pointer
50       % to the next element in the list, then the content of it
51       \checkLinkedList(\@tempa,2)%
```

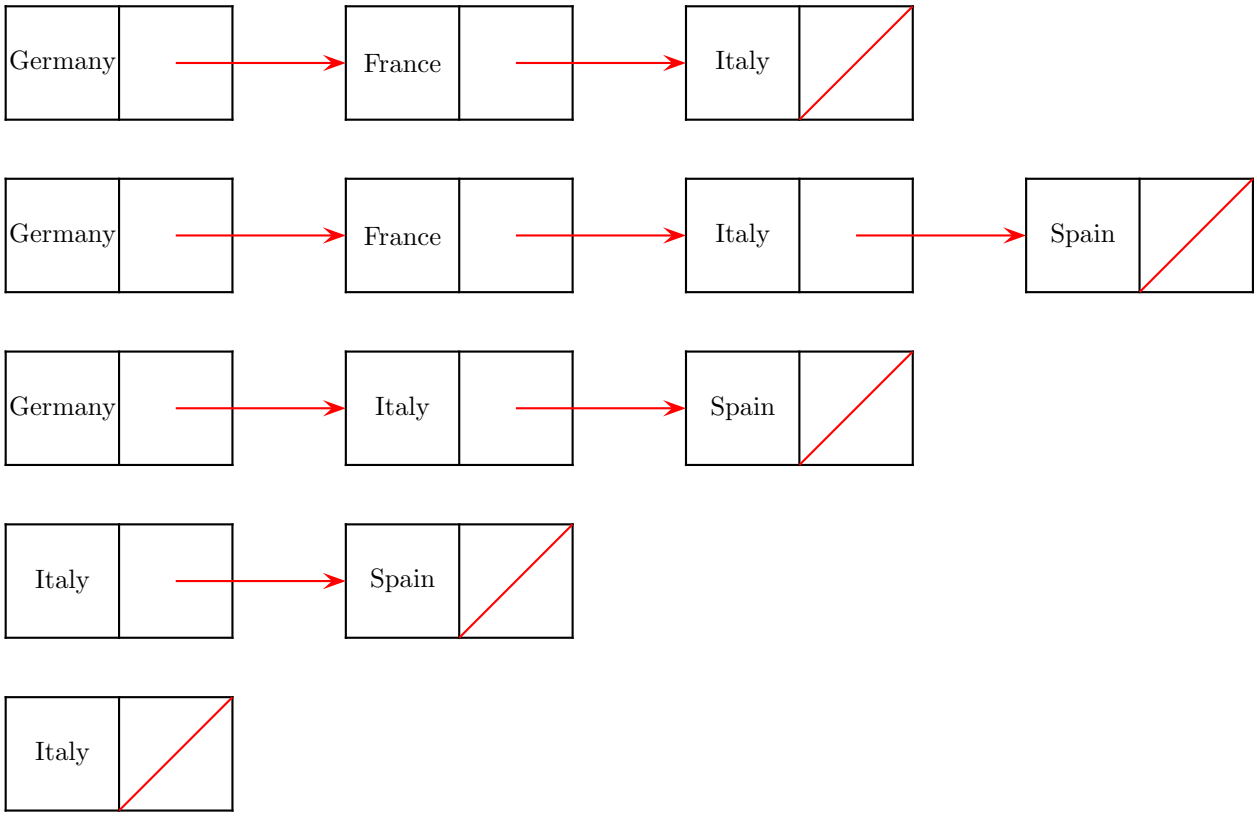
```
      \ifemptydata
        \typeout{Element ‘#1’ has no successor!^^J}%
      \else
        \checkLinkedList(\cachedata,1)%
      \fi
      \multidostop
    \else
      \checkLinkedList(\@tempa,2)%    We got the pointer to the next element
      \ifemptydata
        % We have reach the end of the list without finding the element
        \typeout{Element ‘#1’ not found!^^J}%
        \multidostop%                 No pointed element: end of the list
      \fi
    \fi}
\fi}

% Delete an element in the list (the code is very close to \LinkedListGetNext)
\def\LinkedListDelete#1{%
\checkLinkedList(1,2)%                We got the pointer for the first element
\edef\@tempa{\@ne}%
\ifx\cachedata\empty
\else
  \edef\@tempc{#1}%
  \multido{}{\LinkedListMaxDepth}{%
    \edef\@tempb{\@tempa}%
    \edef\@tempa{\cachedata}%
    \checkLinkedList(\cachedata,1)%   We got the pointed element
    \ifx\cachedata\@tempc
      % This is the element to delete: we will update the pointer of
      % the preceding element to the value of the pointer of the deleted one
      \checkLinkedList(\@tempa,2)%
      \expandarrayelementtrue
      \LinkedList(\@tempb,2)={\cachedata}%
      \expandarrayelementfalse
      \ifnum\PstDebug=\@ne
        \typeout{Delete LinkedList(\@tempa)=#1}% Debug
        \typeout{\space\space\space\space\space\space\space %
                        LinkedList(\@tempb)->\cachedata}% Debug
      \fi
      \multidostop
    \else
      \checkLinkedList(\@tempa,2)%    We got the pointer to the next element
      \ifemptydata
        % We have reach the end of the list without finding the element
        \typeout{Element ‘#1’ not found and so not deleted!^^J}%
        \multidostop%                 No pointed element: end of the list
      \fi
    \fi}
```

```
\fi}

% Print the current state of the list
\def\LinkedListPrint{%
\checkLinkedList(1,2)%                We got the pointer for the first element
\ifx\cachedata\empty
  % Empty list
  \typeout{The list is empty!}%
\else
  \multido{}{\LinkedListMaxDepth}{%
    \edef\@tempa{\cachedata}%
    \checkLinkedList(\cachedata,1)%   We got the pointed element
    \typeout{LinkedList=\cachedata}%
    \checkLinkedList(\@tempa,2)%      We got the pointer to the next element
    \ifemptydata
      \multidostop%                   No pointed element: end of the list
    \fi}
\fi
\typeout{}}

% Draw the current state of the list
\def\LinkedListDraw{{%
\psset{subgriddiv=0,gridlabels=0}
\checkLinkedList(1,2)%                We got the pointer for the first element
\ifx\cachedata\empty
  % Empty list
  \typeout{The list is empty!^^J}%
\else
  % To compute the size of the picture
  \Multido{\iSize=3+3}{\LinkedListMaxDepth}{%
    \checkLinkedList(\cachedata,2)%   We got the pointer to the next element
    \ifemptydata
      \multidostop
    \fi}
  %
  \checkLinkedList(1,2)%              We got the pointer for the first element
  \pspicture(\iSize,1.5)
    \multido{\iPos=0+3}{\LinkedListMaxDepth}{%
      \edef\@tempa{\cachedata}%
      \checkLinkedList(\cachedata,1)% We got the pointed element
      \rput(\iPos,0){%
        \psgrid(2,1)
        \rput(0.5,0.5){\cachedata}
        \ifnum\iPos=\z@
        \else
          \rput(-1.5,0.5){\psline[linecolor=red,arrowscale=2]{->}(1.5,0)}
        \fi}
      \checkLinkedList(\@tempa,2)%    We got the pointer to the next element
```

```
      \ifemptydata
        \rput(\iPos,0){\psline[linecolor=red](1,0)(2,1)} % End of the list
        \multidostop%                  No pointed element: end of the list
      \fi}
  \endpspicture%
\fi}}

\makeatother

\psset{unit=1.5}

\LinkedListPrint\LinkedListDraw

\LinkedListAdd{Germany}\LinkedListPrint\LinkedListDraw

\LinkedListAdd{France}\LinkedListPrint\LinkedListDraw

\LinkedListAdd{Italy}\LinkedListPrint\LinkedListDraw

\LinkedListAdd{Spain}\LinkedListPrint\LinkedListDraw

\LinkedListGetNext{France}\typeout{Next element after ‘France’ is: ‘\cachedata’}%
\LinkedListGetNext{Germany}\typeout{Next element after ‘Germany is: ‘\cachedata’}%
\LinkedListGetNext{Spain}
\LinkedListGetNext{Unknown}

\LinkedListDelete{Unknown}

\LinkedListDelete{France}\LinkedListPrint\LinkedListDraw

\LinkedListDelete{Germany}\LinkedListPrint\LinkedListDraw

\LinkedListDelete{Spain}\LinkedListPrint\LinkedListDraw

\LinkedListDelete{Italy}\LinkedListPrint\LinkedListDraw
```





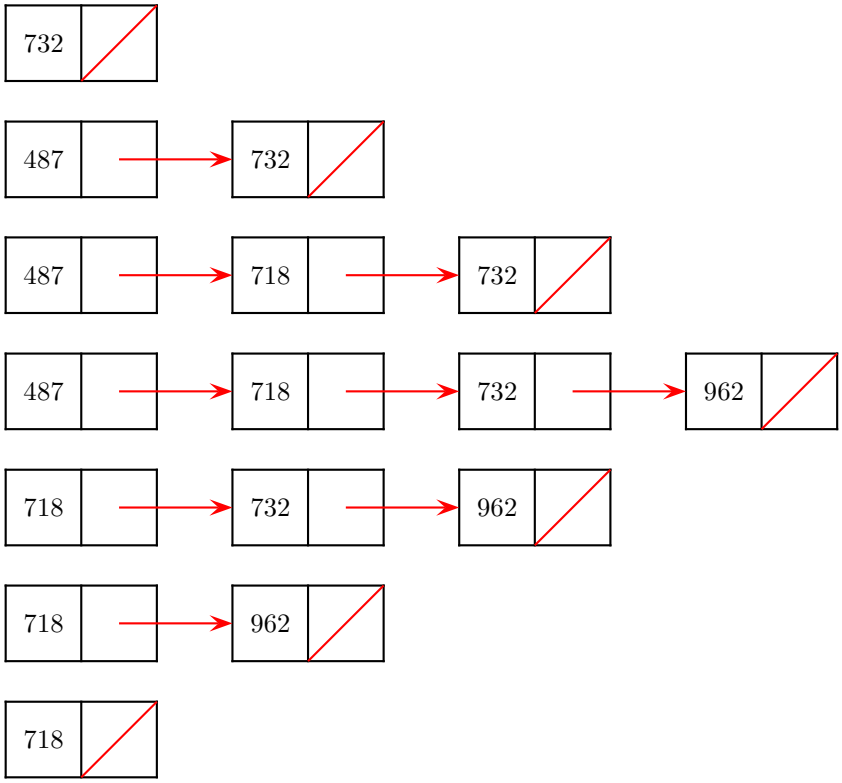


We can write an extended version of the inclusion of an element, where we do not still store each element at the end of the list, but insert it at a special position. For instance, we can decide to use the list to sort the elements, inserting a new one at it sorted position. In the next example, we will manage integer numbers in this way (of course, we can do the same thing with arbitrary strings, but it would be more difficult to program in TEX, mainly if we want to be able to use accentuated letters).

```
\makeatletter

% Add an element at the end of the list, linked in it ordered place
% (this version work only with contents as numbers)
\def\LinkedListSortedAdd#1{%
% Initializations
\edef\@tempa{\@ne}%
\edef\@tempb{\@ne}%
\edef\@tempc{\z@}%
%
\multido{\iElem=2+1}{\LinkedListMaxDepth}{%
  \checkLinkedList(\@tempa,2)% We got the pointer of this element
```

```
  \ifemptydata
    % No pointed element, so we will insert the new one here
    \LinkedList(\iElem,1)={#1}%
    % We update the pointer for the last element
    \expandarrayelementtrue
    \LinkedList(\@tempb,2)={\iElem}%
    \ifnum\PstDebug=\@ne
      \typeout{Add LinkedList(\@tempb)->\iElem}% Debug
      \typeout{\space\space\space\space LinkedList(\iElem)=#1}% Debug
    \fi
    % We update the pointer of the new element to the position
    % of the next greater one, if it exist
    \ifnum\@tempc=\z@
    \else
      \LinkedList(\iElem,2)={\@tempc}%
      \ifnum\PstDebug=\@ne
        \typeout{Update LinkedList(\iElem)->\@tempc}% Debug
      \fi
    \fi
    \expandarrayelementfalse
    \multidostop
  \else
    \edef\@tempa{\cachedata}%
    \checkLinkedList(\cachedata,1)%
    \ifnum#1<\cachedata
      \edef\@tempc{\@tempa}% New element less than this one
    \else
      \edef\@tempb{\@tempa}% New element greater or equal than this one
    \fi
  \fi}}

\makeatother

\LinkedListPrint

\LinkedListSortedAdd{732}\LinkedListPrint\LinkedListDraw

\LinkedListSortedAdd{487}\LinkedListPrint\LinkedListDraw

\LinkedListSortedAdd{718}\LinkedListPrint\LinkedListDraw

\LinkedListSortedAdd{962}\LinkedListPrint\LinkedListDraw

\LinkedListDelete{487}\LinkedListPrint\LinkedListDraw

\LinkedListDelete{732}\LinkedListPrint\LinkedListDraw
```

```
59
60 \LinkedListDelete{500}
61
62 \LinkedListDelete{962}\LinkedListPrint\LinkedListDraw
63
64 \LinkedListDelete{718}\LinkedListPrint\LinkedListDraw
```



3.2.5 Associative arrays

To finish, we will give a complete solution to a classical problem: to count the number of occurrences of the various letters in a sentence. For this, we will first build some macros to deal with *associative* arrays, as they have been popularized by scripting languages like `AWK` and `Perl`.

```
1 \makeatletter
2
3 % Internally, we use two "standard" arrays to define one associative array
4 \newarray\AssociativeArray@Names
5 \newarray\AssociativeArray@Values
6
7 \newcount\AssociativeArrayNbValues
```

```
\AssociativeArrayNbValues=\z@

\newif\ifAssociativeArray@ElementFound

% To store one element
\def\AssociativeArray(#1)=#2{%
\expandarrayelementtrue
\AssociativeArray@ElementFoundfalse
\edef\@tempa{#1}%
\Multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
  \checkAssociativeArray@Names(\iValue)%
  \ifx\@tempa\cachedata
    % This element already exist: we replace it current value
    % \checkAssociativeArray@Values(\iValue)% Debug
    % \typeout{In #1, replace `\cachedata'\space by `#2'}% Debug
    \AssociativeArray@Values(\iValue)={#2}%
    \AssociativeArray@ElementFoundtrue
    \multidostop
  \fi}
\ifAssociativeArray@ElementFound
\else
  % New element
  \advance\AssociativeArrayNbValues\@ne
  \AssociativeArray@Names(\AssociativeArrayNbValues)={#1}%
  \AssociativeArray@Values(\AssociativeArrayNbValues)={#2}%
\fi}

% To get one element
\def\checkAssociativeArray(#1){%
\edef\@tempa{#1}%
\edef\@tempb{999999}%
\Multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
  \checkAssociativeArray@Names(\iValue)%
  \ifx\@tempa\cachedata
    % We have found it by name
    \edef\@tempb{\iValue}%
    \multidostop
  \fi}
% We have now to get it value
\checkAssociativeArray@Values(\@tempb)}

% Simple macro to print all the associative array
\def\printAssociativeArray{%
\multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
  \checkAssociativeArray@Names(\iValue)%
  \iValue: \BS\texttt{FirstNames(\cachedata)}=`\AssociativeArray@Values(\iValue)'\space}}
```

```
54
55 \makeatother
56
57 \let\FirstNames\AssociativeArray
58 \let\checkFirstNames\checkAssociativeArray
59 \let\printFirstNames\printAssociativeArray
60
61 \FirstNames(Crawford)={Joan}
62 \FirstNames(Tierney)={Gene}
63 \FirstNames(Lake)={Veronika}
64
65 \printFirstNames
66
67 \checkFirstNames(Lake)
68 \verb+\FirstNames(Lake)+=‘\cachedata’
69
70 \checkFirstNames(Monroe)
71 \ifemptydata
72   \verb+\FirstNames(Monroe) undefined!+
73 \else
74   \verb+\FirstNames(Monroe)+=‘\cachedata’
75 \fi
```

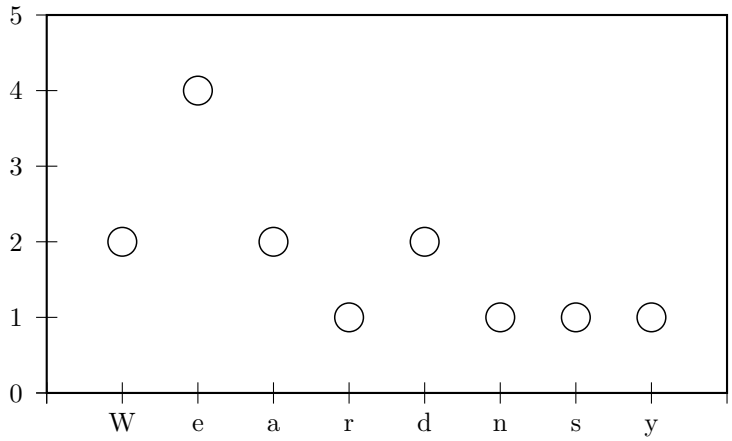
1: `\FirstNames(Crawford)`=‘Joan’ 2: `\FirstNames(Tierney)`=‘Gene’ 3: `\FirstNames(Lake)`=‘Veronika’
`\FirstNames(Lake)`=‘Veronika’
`\FirstNames(Monroe) undefined!`

Now, we can define the macros to read a sentence, to cut it letter by letter, to count the occurrences of each of them, and finally to draw a summary plot of the results.

```
1  \makeatletter
2
3  % A \PerChar style macro adapted from Juergen Schlegelmilch
4  % (<schlegel@Informatik.Uni-Rostock.de> - posted on c.t.t. January 27, 1998)
5  \def\PerChar#1#2{\DoPerChar#1#2\@nil}
6
7  \def\DoPerChar#1#2#3\@nil{%
8  #1#2%
9  \edef\@tempa{#3}%
10 \ifx\@tempa\empty
11 \else
12   \DoPerChar#1#3\@nil
13 \fi}
14
15 % The action to do for each character: we increase the counter
16 % for this character by one
```

```
\def\ActionPerChar#1{% #1=character
\checkAssociativeArray(#1)%
\ifemptydata
  \@tempcnta=\@ne
\else
  \@tempcnta=\cachedata
  \advance\@tempcnta\@ne
\fi
\AssociativeArray(#1)={\the\@tempcnta}}

% To store in an associative array the number of occurrences by characters
% (spaces are not counted)
\def\StatSentence#1{% #1=sentence
\AssociativeArrayNbValues=\z@
\PerChar{\ActionPerChar}{#1}}

% To draw a plot of the number of occurrences of the characters of a sentence
\def\DrawOccurrences#1{% #1=sentence
\StatSentence{#1}
%
\pst@cnta=\AssociativeArrayNbValues
\advance\pst@cnta\@ne
%
% To know the maximum of the numbers
\pst@cntb=\z@
\Multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
  \checkAssociativeArray@Values(\iValue)%
  \ifnum\cachedata>\pst@cntb
    \pst@cntb=\cachedata
  \fi}
\advance\pst@cntb\@ne
%
% The drawing itself
\pspicture(-0.5,-0.5)(\pst@cnta,\pst@cntb)
  \psaxes[axesstyle=frame,labels=y](\pst@cnta,\pst@cntb)
  \multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
    \Draw@OneOccurrence{\iValue}}
\endpspicture}

% To draw the value for one occurrence
\def\Draw@OneOccurrence#1{% #1=Letter
\checkAssociativeArray@Values(#1)%
\psdot(#1,\cachedata)
\rput[B](\iValue,-0.5){\AssociativeArray@Names(#1)}}

\makeatother
```

```
63
64 \let\Names\AssociativeArray
65 \let\checkNames\checkAssociativeArray
66
67 \psset{dotstyle=o,dotsize=0.4}
68
69 \DrawOccurrences{We are Wednesday}
```

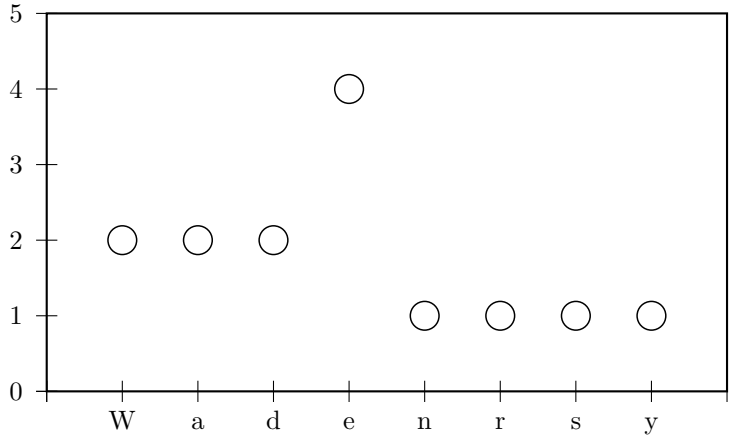


We will now write another version, using another array to store indexes, to allow us to sort the letters found in the sentence read.

```
1  \makeatletter
2
3  % To get in #2 the ASCII code of the #1 character
4  \def\Character@AsciiCode#1#2{\chardef#2=`#1\relax}
5
6  % A new "standard" array to store the sorted indexes
7  \newarray\AssociativeArray@Indexes
8
9  % A simple macro to print the array of indexes (for debugging)
10 \def\PrintIndexes{%
11 \multido{\iIndexes=\@ne+\@ne}{\AssociativeArrayNbValues}{%
12   \checkAssociativeArray@Indexes(\iIndexes)%
13   \BS\texttt{indexes(\iIndexes)}=`\cachedata'\space}}
14
15 % We redefine the macro to store an element in the associative array,
16 % to allow to sort it elements, using an additional array of indexes
17 \def\AssociativeArray(#1)=#2{%
18 \expandarrayelementtrue
```

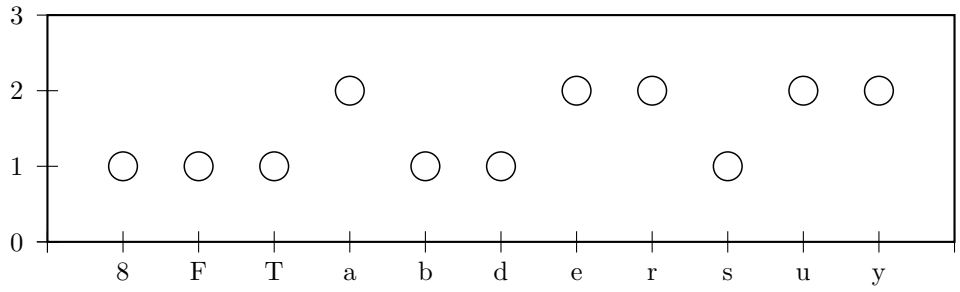
```
\AssociativeArray@ElementFoundfalse
\edef\@tempa{#1}%
\edef\@tempb{#2}%
\Multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
  \checkAssociativeArray@Names(\iValue)%
  \ifx\@tempa\cachedata
    % This element already exist: we replace it current value
    % \checkAssociativeArray@Values(\iValue)% Debug
    % \typeout{In #1, replace `\cachedata'\space by `#2'}% Debug
    \AssociativeArray@Values(\iValue)={\@tempb}%
    \AssociativeArray@ElementFoundtrue
    \multidostop
  \fi}
\ifAssociativeArray@ElementFound
\else
  % New element: we must insert it at it sorted position
  \@tempcnta=\@ne
  \expandafter\Character@AsciiCode\@tempa\@tempx
  \Multido{\iValue=\@ne+\@ne}{\AssociativeArrayNbValues}{%
    \checkAssociativeArray@Names(\iValue)%
    \expandafter\Character@AsciiCode\cachedata\@tempy
    \ifnum\@tempx<\@tempy
      % The new element must be before this one: we will exchange their values
      \checkAssociativeArray@Indexes(\iValue)%
      \@tempcntb=\cachedata
      \advance\@tempcntb\@ne
      \AssociativeArray@Indexes(\iValue)={\the\@tempcntb}%
    \else
      \advance\@tempcnta\@ne
    \fi}
  %
  \advance\AssociativeArrayNbValues\@ne
  \AssociativeArray@Names(\AssociativeArrayNbValues)={\@tempa}%
  \AssociativeArray@Values(\AssociativeArrayNbValues)={\@tempb}%
  \AssociativeArray@Indexes(\AssociativeArrayNbValues)={\the\@tempcnta}%
\fi}

% We change the way we draw each number of occurrences
\def\Draw@OneOccurrence#1{%
\checkAssociativeArray@Indexes(#1)%
\pst@cnta=\cachedata
\checkAssociativeArray@Values(#1)%
\psdot(\the\pst@cnta,\cachedata)
\rput[B](\the\pst@cnta,-0.5){\AssociativeArray@Names(#1)}}

\makeatother
```

```
\DrawOccurrences{We are Wednesday}

\PrintIndexes
```



\indexes(1)=‘1’ \indexes(2)=‘4’ \indexes(3)=‘2’ \indexes(4)=‘6’ \indexes(5)=‘3’ \indexes(6)=‘5’ \indexes(7)=‘7’ \indexes(8)=‘8’

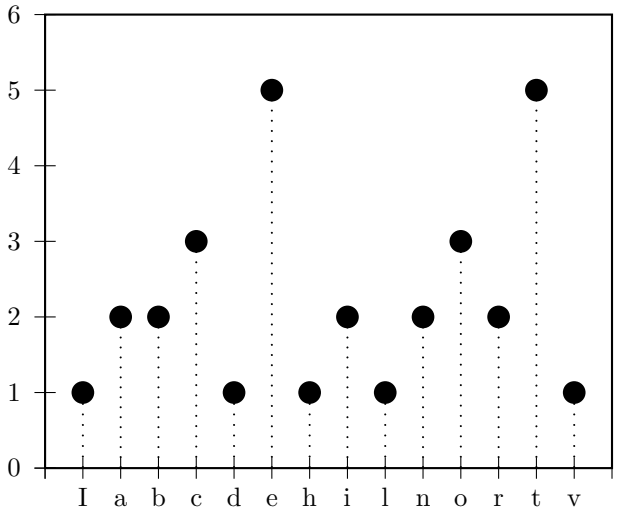
```
\DrawOccurrences{Tuesday 8 February}
```



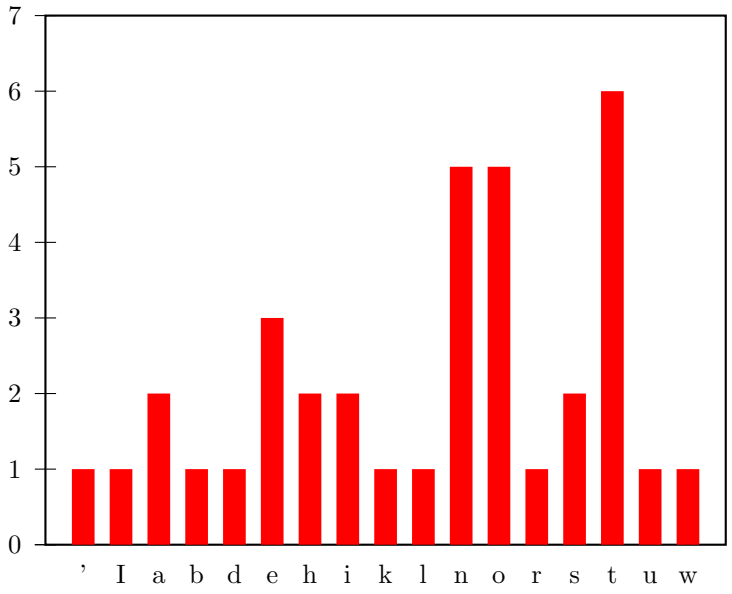
Now, we can easily redefine the internal macro which plot each occurrence, to obtain various kinds of graphics:

```
\makeatletter

\def\Draw@OneOccurrence#1{%
\checkAssociativeArray@Indexes(#1)%
\pst@cnta=\cachedata
\checkAssociativeArray@Values(#1)
\psline[linestyle=dotted](\the\pst@cnta,0)(\the\pst@cnta,\cachedata)
\psdot(\the\pst@cnta,\cachedata)
\rput[B](\the\pst@cnta,-0.5){\AssociativeArray@Names(#1)}}

```

```
\makeatother

\psset{xunit=0.5,dotstyle=*,dotsize=0.3}
\DrawOccurrences{I do not believe that it can be correct}
```




```
\makeatletter

\SpecialCoor

\def\Draw@OneOccurrence#1{%
\checkAssociativeArray@Indexes(#1)%
\pst@cnta=\cachedata
\checkAssociativeArray@Values(#1)%
\pspolygon*[linecolor=red](! \the\pst@cnta\space 0.3 sub 0)
                          (! \the\pst@cnta\space 0.3 sub \cachedata)
                          (! \the\pst@cnta\space 0.3 add \cachedata)
                          (! \the\pst@cnta\space 0.3 add 0)
\rput[B](\the\pst@cnta,-0.5){\AssociativeArray@Names(#1)}}

\makeatother

\psset{xunit=0.5,ticks=y}
\DrawOccurrences{I don't know a better solution than this one}
```



4 Thanks

I would like to thank Rolf Niepraschk <niepraschk@ptb.de> to have carefully read a preliminary version of this document and to have sent me several good advices to clarify and improve some points.

References

[1] Stephan von Bechtolsheim, *TEX in Practice*, four volumes, Springer-Verlag, New York, 1993 (see also CTAN:macros/tip/arraymac.tip).

[2] ‘repeat’, by Victor Eijkhout, University of Illinois, Urbana-Champaign, USA, CTAN:macros/generic/eijkhout/repeat

[3] Philippe Esperet and Denis Girou, *Coloriage du pavage dit de Truchet*, Cahiers GUTenberg, Number 31, December 98, pages 5-18, in french (see http://www.gutenberg.eu.org/pub/GUTenberg/publicationsPS/31-girou.ps.gz).

[4] (AI)DraTEX, by Eitan M. Gurari, Ohio State University, Columbus, USA, CTAN:graphics/dratex (see also http://www.cis.ohio-state.edu/~gurari/systems.html).

[5] Eitan M. Gurari, *TEX and LATEX: Drawing and Literate Programming*, McGraw-Hill, New-York, 1994.

[6] METAPOST, by John D. Hobby, AT&T Bell Laboratories, USA, CTAN:graphics/metapost (see also http://cm.bell-labs.com/who/hobby/MetaPost.html).

[7] ‘formlett’, by Zhuhan Jiang, University of New England, Armidale, Australia, CTAN:macros/generic/formlett.sty (see also http://maths.une.edu.au/~zhuhan/util.html#computer).

[8] ‘multido’, by Timothy van Zandt, INSEAD, Fontainebleau, France, CTAN:macros/generic/multido

[9] PSTricks, by Timothy van Zandt, INSEAD, Fontainebleau, France, CTAN:graphics/pstricks (see also http://www.tug.org/applications/PSTricks).