A NATURAL DATA STRUCTURE FOR TEXT FORMATTING

(submitted to Software Practice and Experience)
A NATURAL DATA STRUCTURE FOR TEXT FORMATTING

Juris REINFELDS*
CERN
Geneva, Switzerland

Abstract

A simple recursive data structure is defined and shown to apply to every stage of the text layout problem. Hence a single mechanism can be used to generate text layouts from the simplest sequence of equally spaced characters to context dependent-spacing, overlapped and partially overlapped characters, mathematical equations and tables. This approach should simplify the design of text processors which currently use several different mechanisms and sometimes produce conflicting effects.

*Permanent address : Department of Computing Science
University of Wollongong
WOLLONGONG, NSW 2500 Australia
1. **INTRODUCTION**

Of text processors which apply to a full range of output devices it seems that the most widely used systems are nroff/troff [2] [1], Script [3] and TEX [4]. Each system provides an extensive set of operators to shape the environment of the formatting process and to place text on the output page. The set of operators is introduced without any attempt to define any underlying mechanisms or data structures from which these operators could be derived.

This paper describes a data structure and a mechanism which seem fundamental to text formatting so that many of the otherwise independent or ad-hoc text formatting operators can be derived systematically and simply. Together with a macro mechanism and a mechanism for setting and using marks and traps on the output page [5] one has a simple basis for the text formatting problem. Only a few additional mechanisms are needed to derive the much larger set of text layout operators that are currently regarded as basic [6].

2. **CHARACTERS**

A printable character has two important attributes. It has a visible form that is actually printed and it has a size that is described by three numbers: height, depth and width.

![Character Diagram](image)

so that the height and depth define the base line and the left-hand-end of the base line defines the base point of the character. The blank space is a printable character without a visible form. Nevertheless it has a height and depth to define its base point and it has a width like any other character. Characters are normally printed with their base lines aligned and with base points of each successive character touching the right-end of the base line of the previous character.

Printers used to place narrow slivers of lead between characters which appear as variably sized blank space characters and Knuth [4] places "glue" there. Glue like Ether has magical properties and it seems unwise to overload the semantics of the blank space character.

Another way to look at the problem is to imagine that the placement of each character consists of two steps: move the base point of the character to any arbitrary point \((x, y)\) on the current output page and print the character there (in its box of height \(h\), depth \(d\) and width \(w\)). A character printable by this mechanism is therefore a data object that contains a code describing its visible shape, a pair of numbers describing the move and three numbers describing its size. We may visualise it as follows:
The size of a character is normally determined at font definition time and it does not change from print job to print job. Therefore it makes sense to store the h, d and w of characters in a lookup table rather than with each character. This is an implementation detail that has to do with efficiency. Conceptually each character has a well defined size and the h, d and w of any character can be found whenever needed.

2.1 Implicit Moves

The move \((x,y)\) is not a property of the character. It is determined by the text formatting process. A character by itself exists in isolation and needs no \((x,y)\). The text formatting process generates a sequence. It places each next character in a specific relationship to the previous character and only the previous character.

Moves may be described absolutely with respect to the output medium or relatively with respect to the previous character. For absolute moves one has to introduce an environment that defines the location, size and other properties of the output medium on which the formatter has to place characters. For relative moves one introduces the notion of the current point. The text processor transfers a sequence of moves and characters to the output. No matter how a move is defined a relative displacement can always be found to determine the move uniquely. This may be envisioned as the movement of the current point on the output page.

The current point is the location \((x,y)\) of the right end of the base line of the most recent output character. The action that prints a character also moves the current point to the right hand end of the base line of the character. Note that the simplest form of typewriter output includes sufficient space in the width of each character so that the automatic movement of the current point in character placement is sufficient for the generation of a sequence of output characters. In that case the implicit move \((x,y)\) would be empty \((0,0)\) and once again it is an implementation detail whether and how to represent all the redundant zeros. The important point is that the automatic carriage advancement is just one special case of a more general mechanism which as we shall see also provides for the construction of equations, tables, footnotes and other structures. Using the full generality of this mechanism the text processor may use the move associated with each character to provide context dependent spacing or a simpler text processor may ignore these moves altogether.

2.2 Explicit Moves

All text formatters provide various commands to generate explicit moves. The "space" command usually moves the current point to the left margin and down a specified number of lines, the "indent" command makes a
horizontal move at the beginning of each output line and so on. All these commands are special cases of the general move command

\[ \text{move } x \ y \]

which moves the current point to location \((x, y)\). The general move command is a special case of the above character concept. It is a character whose printable character part is empty, namely \(h = d = w = 0\) and we have the representation

\[
\begin{array}{c|c|c}
 x & y & \\
 \hline
 \text{Character box of zero dimension} & \text{h=0} & \text{d=0} \quad \text{w=0} \\
\end{array}
\]

which is of course different from the blank space whose width is not zero and which because of base-line alignment problems may be given non-zero height and depth.

An explicit move command is therefore nothing else than a character of zero size: it is a pure motion of the current point.

3. LINE SHEETS AND LINE DIVERSIONS

Text formatters generate a sequence of output characters until these characters fill or almost fill an output line. Then the formatter may be required to adjust these characters in some way to satisfy some environmental requirement such as the adjustment of the right-margin of the text. Only then the processed line is placed as a whole output object in the sequence of completed output lines. Let us therefore investigate what kind of data object may represent an output line and how such an output line may be formed.

The previous section established that any desired layout of visible text can be produced by a suitable sequence of characters, some of which are empty and represent moves.

Suppose that we are given a sheet of output medium that is of unspecified vertical size, whose width equals some number called the line length \((\ell)\) with an origin \((0,0)\) somewhere in the left-hand margin and with the current point at the origin \((x=y=0)\). Let us call such a sheet a "line-sheet".

\[
\begin{array}{c}
(0,0) \quad x \text{ increases to the right} \\
\hline
y \text{- increases downwards} \quad \text{line length } \ell \\
\end{array}
\]
Characters are processed from input to output using the "move the current point to \((x,y)\) then place the character" mechanism. Precisely how many characters should be placed in the sequence with regard to user generated moves, word boundaries, hyphenations and other requirements is a problem on a different level. For this discussion let us assume that the sequence of characters that should be contained in the line is given. Then we have in the general case

where the sequence of characters \(c_1, c_2, \ldots, c_n\) defines a height, depth, width and base point for the whole line as follows:

The origin \((0,0)\) of the output sheet is the base point of the whole line because the current point starts there. If the line construction algorithm involves no implicit displacement of the first diversion then the base point of the line coincides with the base point of the first diversion. This is normally the case.

The width of the whole line is equal to the line length if the characters are right-adjusted by the line completion mechanism. For shorter lines the width of the line is the distance spanned by its characters. This can be calculated from the moves and widths of the characters. It is the maximum horizontal excursion of the current point.

The height plus depth of the whole line is equal to the maximum vertical excursion of the heights and depths of the characters constituting the line. The separation into height and depth is made to locate the base point of the line.

The line is placed in the sequence of completed output lines by first moving the current point of the output page to a desired spot \((x,y)\) and then by placing the whole line there with the base point of the line at \((x,y)\). The whole line is therefore an object that aside from internal structure is identical in form to the character introduced in the previous section. We have
If we hide the internal structure of the line in its interior the structure simplifies to

![Diagram of a whole line of processed output text]

This structure is similar to Knuth's box [4]. It is also similar in concept to the notion of a diversion of output text in nroff/troff [2]. Let us therefore call these structures diversions. So far we have shown that output characters, output moves and output lines are diversions.

4. PAGE-SHEETS AND PAGE DIVERSIONS

Output lines are sequenced vertically one after the other to form an output page. A page is a diversion that is made up of a vertical sequence of output lines.

Assume a page-sheet similar to the line-sheet. The page starts with the current point (and therefore its base point) at the origin (0,0). Before the line is placed, a typical page construction algorithm will move the current point to conform with the environmentally specified indent horizontally and with the next line vertically. Hence the base point of the page will not typically coincide with the base point of the first line. The width of a page is determined by the maximal horizontal excursion of the current point. The height and depth of the page is determined by the vertical excursion of the constituent lines above and below the origin (0,0).

The nesting of diversions is recursive. At each level of nesting two sheets are introduced: a line-sheet and a page-sheet each with its own current point.

The text formatter may provide horizontal adjustment of the constituent diversions at the line level and vertical adjustment at the page level because on each sheet all moves \((x_i, y_i)\) of the constituent diversions are available to the formatter until the sheet is converted into a diversion. As a diversion the structure has a fixed size \((h, d, w)\) and it participates in further adjustment processes as a whole.

4.1 User Constructed Diversions

The author of the input text may require a figure, a footnote, a piece of horizontal text that should not be split into two lines, or more than one line of text that should not be split between pages.
All such facilities can be made available by the provision of control commands that permit the construction of named diversion which can be interpolated by name one or more times later. The above mechanism suffices for the construction of named diversions. Each level of diversion obtains two new sheets, a line-sheet and a page sheet where both sheets are of indefinite extent. The environmentally specified indents and page length apply only to the actual output page but for any other page the user may set these parameters to any desired value. The default is indefinite page length and no indent.

The formatter looks at input text in two modes: text mode and control mode. In the text mode control commands are distinguished by the control command-brackets defined by the escaped characters (using the backslash as the escape character)

\: <control command> \;

Between the control command brackets the formatter is in the control mode so that

* control brackets are redundant
* text must be bracketed by escaped quotes thus
  \ this is text and not commands \n* semicolons are used as statement separators
* commas or blank spaces separate arguments of commands. Commas must be used if blank spaces are contained in the argument.

For diversion definition let us introduce the diversion definition superbrackets which include the diversion name for unique pairing of opening and closing brackets:

\:diversion_begin name-of-diversion\;  
\:diversion_append name-of-diversion\;  
\:diversion_end name-of-diversion\;

so that we can define a new diversion named divdiv with the following input text:

\:diversion_begin divdiv;  
<diversion text>  
diversion_end divdiv\;  
or one can append more lines to an existing diversion named divdiv

\:diversion_append divdiv ;  
<more text for divdiv>  
diversion_end divdiv;

A diversion specification suspends the construction of the current diversion (the currently not yet completed line-sheet and page-sheet). The text processor starts a new pair of sheets with the current-point at (0,0) and endless vertical extent. Otherwise the new sheets inherit the environment of the previous level of diversion. The page length of the environment (pL) is effective only on the outermost level of final output.
Upon completion the diversion is stored by the text processor and may be called by the user whenever it is needed. The text processor resumes the processing of the previous level of diversion construction. Note that named diversion construction requires no new mechanism for its implementation.

The text processor must provide commands which interpolate or delete named diversions. The text processor should provide functions which take the diversion name as argument and produce the height, depth or width of the diversion as a result. There, should also be functions that make accessible the x or y of the move associated with a diversion.

4.2 Interpolation of Named Diversions

A named diversion is interpolated like any other diversion by moving the current point to a specified location (x,y) and by placing the diversion there (the line construction mechanism of the formatter ensures that the placement of the diversion moves the current point to the right-hand-end of the base line of the diversion). Hence to interpolate a named diversion the user has to specify the diversion name as command name and the point (x,y) as arguments. By default the omission of arguments indicates placement at the current point. For example if we have created a diversion named divdiv which should be interpolated at the current point we would simply write the command

\:divdiv:;

To interpolate the same diversion with its base point horizontally 5cm from the base point of the current line and vertically one line-width below the current point we would write

\:divdiv 5_cm \ y + 1\ line:;

where the current point's location at (x,y) is maintained by the formatter. Internally in the formatter, the page construction mechanism of a typical formatter may be expressed as a diversion call onto the page sheet of a diversion named linediv

\:linediv \ in+ti-x \ d+height(linediv):;

where the formatter maintains \h \d \w as the height, depth and width of the most recently placed diversion and the environment contains two indentation variables in for permanent indent and ti for temporary indent. This mechanism uses "natural line spacing", stepping ahead by the actual depth and height of adjacent lines. A more rigid method would be to step one line at a time where the environment would provide a spatial definition of "line":

\:linediv \ in+ti-x \ line:;

Note however that user diversions are placed on the line sheet. The user cannot place diversions directly on the page sheet. Only the formatter itself has access to the page sheet.

4.3 The Decomposition of a Diversion

It is often desirable to interpolate a portion of a diversion. In many cases a diversion needs to be interpolated only once and then deleted. The construction and use of a footnote is a typical example. A footnote
diversion is constructed when the first reference to it is encountered. It is interpolated at the bottom of the page if there is enough space for it. If there is not enough space, the footnote is either deferred to the next page or split between two or more pages. It needs to be interpolated only once but the interpolation may be interrupted after any one of its lines and resumed later.

Remember that a named diversion consists of a sequence of one or more lines to which we can append more lines. Let us introduce an approximate inverse of the append operation: the consume operation. Definition:

The consume operation takes the name of a diversion and a pair of values \((x,y)\) as arguments and it places the first line of the named diversion with base-point at \((x,y)\). It deletes the first line from the named diversion adjusting its parameters accordingly. The named diversion may therefore be consumed one output line at a time.

One may of course append to a named diversion one or more lines at a time. Hence a named diversion may be used as a queue of lines. Diversion construction always takes place on two sheets where a line is constructed first and environmentally specified indents are attached automatically by the formatter as the line is placed on the page. The break command terminates the current line immediately so that if the current page construction and the diversion divdiv have compatible environments then the commands

```
\:break \; consume divdiv \;
```

will first terminate the current line and put it on the output page and will then put the first line of divdiv on the output page. The missing arguments \((x,y)\) by default imply that the line is to be placed "at the current point" which is the base point \((0,0)\) of the new line-sheet. If the line length of divdiv is shorter than the line length provided by the environment at consume time then the lines of divdiv can still be transferred one line at a time by the statement

```
\:while not empty (divdiv) do
    consume divdiv ; break
od;
```

or by the inclusion of a horizontal move of the current point to current line length (environmental variable \(\ddot{u}\)):

```
\:while not empty (divdiv) do
    consume divdiv ; move_x \ddot{u}
    od;
```

From this example it is obvious that the operation of "break" is nothing else than a horizontal move to line length but if we inspect the widths of the resulting lines we will find that the break operation produces a shorter line while the explicit move produces a line whose width is equal to line length.
5. EXAMPLES

It is easy to use diversions to design hierarchically structured components to represent mathematical equations and other structures where at each level of the hierarchy only whole diversions participate in any adjustment or orientation process. For example, in mathematical equations the numerator or the denominator may contain fractional expressions of any complexity but at the current level they are centered over each other as whole diversions. The formatting of mathematical expressions poses four general problems:

i) subscripts or superscripts or both

$$y^2_i \quad x_{ij} \quad a^2 \quad k_{ij}^2 \quad x_{kk}^{\text{mm}}$$

ii) linear expressions with "proper" spacing

$$f(x,y) = 2ax + cy - 16xy$$

iii) numerator over denominator of any length or level

$$\frac{ax + by - cz}{a^2 + b^2 + c^2}$$

iv) large signs such as $$\sqrt{a^2 + b^2 + c^2 + d^2}$$

In the case of large signs diversions cannot help much if the output hardware is not willing to cooperate and provide suitable characters but the other cases are easily solved via the diversion mechanism.

5.1 Subscripts and Superscripts

In the simplest form indexing involves a vertical half-line movement up or down. This is specified simply as

```plaintext
x:\move\ y\ 0.5\_line\ \; i j\ :\ move\ y\ -0.5\_line\ ;
```

to produce

$$x_{ij}$$

For more than the occasional unsophisticated index one needs proper indexing commands so that

```plaintext
x:\sub i j \ ; \quad produces \quad x_{ij}
```

```plaintext
a:\ sup \ 2 \ ; \quad produces \quad a^2
```

```plaintext
k:\ subsup \ i j \ 2 \ ; \quad produces \quad k_{ij}^2
```

The indexing commands need not be intrinsic formatter commands, they may be provided as system macros. A macro is a named sequence of input (unprocessed) text which may include control commands and other macro and diversion calls.
A macro may have arguments which are processed before the macro body and stored as diversions. The indexing macros may be written somewhat like this

```
\:macro_begin sub;
local dist;
environ_save; font index; point =-2;
dist := max(height($1), \d);
$1 \ x \ y + dist;
move_y \ y - dist;
environ_restore;
macro-end sub
```

where the control command brackets (which define the extent of the control mode) enclose the whole macro. The arguments in left to right order are denoted \$1, \$2,..., \$9. The formatter maintains the current point as \$(x, y)$ and the height, depth and width of the most recently placed diversion as \$h, \$d and \$w. The statement "point =-2" decrements the environment's point size by two points. The environ-save saves the state of the environment so that all changes made subsequently are annulled by an environ-restore later. Note that the font cannot be restored explicitly unless the pre-index font is saved in a local variable. For the superscript the macro could be written as

```
\:macro_begin sup 
\:local dist \n\:environ_save; font_index; point =-2 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:dist := \max(h, depth (\$4)) 
\:move_y \ y \ y - dist 
\:move_y \ y \ y + dist 
\:environ_restore 
\:macro_end sup
```

Where even with the inclusion of many pairs of command brackets the macro has the same structure as the subscript given previously. The subsup macro should place the subscript and superscript on top of each other

```
\:macro_begin subsup;
local distsub, distsup;
environ_save; font_index; point =-2;
distsub := max(height($1), \d);
distsup := max(\h, depth(\$2));
$1 \ x \ y + distsub;
move width($1) \ y - distsub;
$2 \ x \ y - distsup;
move width($2) \ y + distsup
move_x \ max(width($1), width($2));
macro_end subsup
```
where the purpose of the macro is to place the indices over each other and to leave the current point on the base line of the indexed item. For example, the command

\textbackslash x \textbackslash: \textbackslash subsup \textbackslash i \textbackslash j \textbackslash 2 \textbackslash;

will produce the following movements of the current point

![Diagram showing movement of current point](image)

5.2 Proper Spacing in Mathematical Expressions

For a proper layout of mathematical expressions and equations a mechanism has to be provided for the grouping of mathematical symbols and symbol groups. For example, extra space should be provided around binary operators to write:

\[ f(x,y) = 2ax + cy - 37xy \]

rather than

\[ f(x,y) = 2ax + cy - 37xy \]

The spacing of equation components is a very subjective matter and no two mathematicians will agree on what rules govern such spacing. Knuth [4] provides one style of spacing in TEX but the mechanisms to alter this style to satisfy the author's own desires are limited.

In the simplest instance one could simply introduce system macros named with escaped binary operator symbols such as \textbackslash &= \textbackslash & - \textbackslash to provide some extra space around them.

\textbackslash:macro\_begin \textbackslash + \textbackslash;
move\_x width(\textbackslash ' + \textbackslash ');
\textbackslash '+ \textbackslash '
move\_x width(\textbackslash ' + \textbackslash ');
macro\_end \textbackslash + \textbackslash ;

where regardless of font size the macro call \textbackslash + will leave before and after itself a space equal to its own width. The following input text:

\[ f(x,y) = 2ax\textbackslash + cy\textbackslash - 37xy \]

will produce a reasonably spaced equation:

\[ f(x,y) = 2ax + cy - 37xy \]
5.3 Numerators and Denominators

Algebraic fractions are normally written in two dimensional form with a numerator and denominator centered with respect to a division line equal in length to the longer expression. The numerator or denominator can contain more algebraic fractions to any level of nesting. The formatting of such expressions is a formidable problem for any text processor, typist or typesetter. With the diversion mechanism the problem reduces to the following definition:

An algebraic fraction is a diversion consisting of three lines. The lines, which themselves are diversions, are the numerator the division line and the denominator. The first line of the diversion is the division line so that the base point of the algebraic fraction is at the division line level. The second line is the numerator and the third is the denominator although their sequence is of no importance.

If the algebraic fraction formatter is provided as a system macro with the name "numdiv" then it takes two arguments: the numerator followed by the denominator. For example:

\[ \text{numdiv a}\div b \ 2ab \ ];

produces the fraction

\[
\frac{a + b}{2ab}
\]

where the system macro takes the following steps:

(i) produce a diversion for the division line consisting of a sequence of minus signs or some better suited division line characters whose length is equal to the max (width (numerator), width (denominator)). This forms a diversion one character high and as wide as the wider of the two expressions.

(ii) place this diversion as the first line of the new diversion so that the base point of the diversion line is the base point of the whole diversion.

(iii) the height and depth of the denominator and numerator are known because they are diversions. Hence the moves to center the numerator and denominator over and under the division line can be calculated.

\:macro \begin \numdiv;
local \dist, \divlin;
\dist := \max (\text{width ($1$)}, \text{width ($2$)});
diversion \begin \divlin;
\text{while (dist > 0) do}
\quad \divlin
\quad \dist -= \text{width ($\divlin - \divlin$)};
\text{od;}
\end \numdiv;
diversion_end divlin;
  divlin;
  \$1 \times \text{width(divlin)} - \text{width(}$1$) \times \text{depth(}$1$);
  \$2 \times \text{width(}$1$) - \text{width(}$2$) \times \text{depth(}$1$) + \text{height(}$2$);
  \text{move} \times \text{width(}$2$) + \text{width(divlin)} \times \text{y-height(}$2$);
macro_end numdiv \;

where the current point moves as shown below

![Diagram showing the movement of the current point](image)

this macro is called as

\[ \text{:numdiv} \quad a + b \quad 2ab \]

to produce

\[ \frac{a + b}{2ab} \]

or as

\[ \text{:numdiv} \quad \text{:numdiv} \quad a + b \quad 2ab \; \text{:x:sup}2; \; + \; \text{:y:sup}2; \; + \; z; \; \text{:sup}2; \; \]

to produce

\[ \frac{a + b}{2ab} \]

\[ \frac{x^2 + y^2 + z^2}{2ab} \]

while for more complicated expressions we can say

\[ \text{:numdiv} \quad \text{fraction1} \quad \text{fraction2} \; \]

where

\[ \text{:diversion_begin} \quad \text{fraction1} \; \]

\[ \text{:numdiv} \quad a + b \quad 2ab \; \]

\[ \text{:diversion_end_div} \quad \text{fraction1} \; \]

\[ \text{:diversion_begin} \quad \text{fraction2} \; \]

\[ \text{:numdiv} \quad \text{x:sup}2; \; + \; \text{y:sup}2; \; + \; z; \; \text{sup}2; \; 3xy; \]

\[ \text{:diversion_end} \quad \text{fraction2} \; \]

gives the following output
\[ \frac{a + b}{2ab} \]
\[ \frac{x^2 + y^2 + z^2}{3xy} \]

The algebraic fraction is therefore constructed in the most natural way. It locates the division line and centers the numerator and denominator over and under it.

5.4 Multicolumn Output

For multicolumn output we can choose a suitably short line length and construct each column as a separate diversion named column1, column2,\ldots An output device which can move the paper backwards outputs column1 then returns to the top of the page for column2 and so on until the end of the last column completes the page and there is no problem.

Output devices which cannot back up on the output page cannot be provided with multicolumn output by most text formatters. One has to write a preprocessor such as "colr" in nroff/troff\(^2\) to collect the corresponding lines from all columns into single lines.

With the consume operation this problem is trivially solved by the following command:

\[
\text{\"while\ not empty (column1) do}\ \\
\text{conserve column1; \"column gap chars\"}\ \\
\text{conserve column2; \"column gap chars\"}\ \\
\text{...}\ \\
\text{conserve column;}\ \\
\text{od};
\]

6. REFERENCES


