Wire Service Translation Software
for the
Boston Community Information System

by

Victor J. Diniak

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Science in Electrical Science and Engineering
at the
Massachusetts Institute of Technology
May 1986

© Massachusetts Institute of Technology 1986

Signature of Author

Department of Electrical Engineering and Computer Science
May 9, 1986

Certified By

Professor David K. Gifford
Thesis Supervisor

Accepted By

Professor David Adler
Chairman, Department Committee
# TABLE OF CONTENTS

Introduction.................................................................3

Chapter 1: Raw Data Format
  Raw Associated Press Data Format...........................................5
  Raw New York Times Data Format...........................................8

Chapter 2: Formatting Process Overview
  Two Step Associated Press Formatting Process.........................11
  Two Step New York Times Formatting Process..........................12

Chapter 3: The Code
  3.1 The Formatting Drivers.................................................14
  3.2 The Formatting Procedures.............................................21
  3.3 Associated Press Utility Functions....................................45
  3.4 New York Times Utility Functions.....................................52
  3.5 General Utility Functions.............................................69

Conclusion...........................................................................73

Appendix A
  Explanation of Fixed Spacing Codes........................................76

Appendix B
  Single Tabular Data Examples...............................................77

Appendix C
  Double Tabular Data Examples.............................................78

Appendix D
  Sample Articles..................................................................79

Appendix E
  Complete Listing of the Code................................................97

Appendix H
  High Speed Wire Service ASCII Table....................................119

Bibliography......................................................................120
INTRODUCTION

The wire service organizations such as the Associated Press, United Press International, New York Times news service, etc... are a major factor in the collection and swift distribution of hard news, sports, human interest stories, etc... in the United States and other parts of the world. Virtually all television, radio, and print news organizations have access to at least one of the wire services.

Stories from the wire services act as either background information for writers and reporters in news organizations or are used as filler stories, straight off of the wire. In any case, the news organizations that utilize these stories most often use teletype equipment or specialized newsgathering/typesetting equipment to read the stories off of the wire. In the interest of standardizing the equipment in some way, the wire services have agreed upon several sets of specifications for the protocols involved in the transmission of stories over the news wires.

The agreed upon specifications govern the way that data is sent over the wire. More specifically, the specifications define certain codes to be used to identify different types of data. These codes help teletype and typesetting equipment reconstruct the stories into the form in which they were originally written before being placed on the wire. The codes can also be decoded by a general computer system (with special software) to reconstruct the articles into their original form.

The purpose of this project is to decode the codes used by the Associated Press and the New York Times wire services to produce readable articles in their intended form from raw articles that contain the text of the article and the embedded typesetting codes. The specific practical purpose of the project is to update the front end software of the Boston Community Information System project (CIS) so that the articles stored in the CIS database are in an attractive, readable form rather than in the raw wire service format.

This paper will begin with a description of the raw data formats. It will then provide an
overview of the formatting processes, followed by detailed descriptions of the code used to implement the processes. Finally, the paper will close with examples of formatted articles and recommendations for future revisions of the software.
CHAPTER 1: RAW DATA FORMAT

RAW ASSOCIATED PRESS DATA FORMAT

Articles are passed to the AP formatter in an input buffer in the form of a character array. Within this buffer there are variable length fields, each corresponding to one line of the article. Fields are terminated by a line feed (LF) character (high speed wire service ascii 11). The "High-speed Wire Service Transmission Guidelines" (February 1, 1979) imply that the fields or lines can take on one of three forms: Textual, Tabular material (single tabular, double tabular), and Quad material (quad left, quad center, and quad right).

TEXTUAL LINES:

Textual Lines are the simplest of the possible AP lines. The textual line has no special lead characters. It is simply a string of characters terminated by a line feed character.

QUAD MATERIAL:

Quad material is the first and simplest of the interesting cases. It comes in one of three forms: quad left, quad center, and quad right. The general format of quad material is almost identical to that of textual lines. The only difference is that quad material contains a special quad identification character immediately preceding the terminating line feed character.

Quad Left:

Quad left lines are the least interesting of the three quad formats. The special quad identification character for a quad left line is the QL (quad left) character (high speed wire service ascii 60 ("<" character)).

It was said that quad material is almost identical to textual material. To be more specific, all textual material is quad material. Textual lines are simply quad lines without the special quad identification character. Without the character, the line essentially defaults to quad left.

Quad left lines are formatted so that the line starts at the left most edge of the page. This is how textual lines appear. This definition will become more clear when compared
against the other two types of quad material, quad center and quad right.

**Quad Center:**

Quad center is the first interesting case and most commonly used type of quad material. The special quad identification character is the QC (quad center) character (high speed wire service ascii 61 ("=" character)).

A quad center line is simply formatted so that it is centered on the page. Common uses of quad center in a newspaper, for example, are bylines, headings, etc...

**Quad Right:**

Quad right is the second interesting type of quad material and probably the least commonly used. The special quad identification character is the QR (quad right) character (high speed wire service ascii 62 (">" character)).

A quad right line is formatted right justified instead of left justified. By this, it is meant that the entire line is shifted over to the right so that the last character in the line is in the rightmost column on the screen. Quad right examples are rare.

**Example of textual and quad material:**

This is the longest line of text material in the article.

This is quad left.

This is quad center.

This is quad right.

**TABULAR MATERIAL:**

Tabular material is the most interesting of the data formats and is the main reason for this project. This type of material comes in two forms: single tabular lines and double (side-by-side) tabular lines.

**Single Tabular:**

This is the most widely used type of tabular line. A single tabular line starts with a Tab Line Indicator (TLI) character (high speed wire service ascii 8). Next comes a field of textual data followed by a Tab Field Indicator (TFI) character (high speed wire service ascii 31). Following the TFI character is a field of tabular data with the line terminated by a line
feed (LF) character.

| TLI text field | TFI Tabular field | LF |

The problem with this type of line is that it may or may not be sent in justified format. If it is sent in justified format, the tabular field will contain the data to be displayed, separated by standard spaces. It will also contain variable amounts of fixed spacing codes (EM, EN, THIN space, EM leader, EN leader). These codes correspond to spaces of varying widths. They exist because in typeset text, not all character are the same width. Hence, to right and left justify columns, variable width spacing is needed to compensate for the varying numbers of variable width characters. (see Appendix A for a description of fixed spacing).

To further complicate matters, the wire service includes extra spacing, called monitor spacing, to drive monitors on the typesetting equipment. The theory is that the monitor spacing is transparent to the typesetting equipment and the typesetting (fixed) spacing is transparent to the monitors. Where specialized typesetting and monitoring equipment is used, this scheme works fine. However, in an environment that does not use the special equipment, the result can be as harmless as columns not lining up to as harmful as garbage appearing on the screen in place of the fixed spacing codes.

Examples of single tabular lines are numerous, including stock market listings, horse racing results, scoreboard sections of the sports page, etc (see Appendix B for specific examples).

**Double (side-by-side) Tabular Lines:**

Double Tabular lines are an extension of the concept of the single tabular line. A double tabular line starts with three TLI characters. Next comes the left textual field followed by a TFI character and the left tabular field. The left tabular field is terminated by a Center Field Separator (CFS) character (high speed wire service ascii 28). The right textual field follows the CFS character. The right textual field ends in a TFI character and is followed by the right tabular field. Finally, the line is terminated by a line feed (LF) character.
Double tabular lines are faced with all of the same problems that plague single tabular lines. The existence of fixed and monitor spacing codes result in a number of problems ranging from columns not lining up to garbage appearing on the screen.

The main uses of double tabular lines are sports box scores which are typically found in the scoreboard section of the sports page. They also have uses in producing certain types of tables. (See Appendix C for specific examples of double tabular lines).

It was said that articles are passed to the AP formatter in an input buffer in the form of a character array. Each line in the article is a field in this array that is terminated by a line feed character. The first five or six lines of the article contain the header of the article. The last line of the header contains the single word, "text:." The article (input buffer) ends in a '"0' character.

**RAW NEW YORK TIMES DATA FORMAT**

Articles are passed to the New York Times formatter in an input buffer in the form of a character array. Within this buffer there are variable legnth fields, each corresponding to one line of text in the article. Each field is terminated by a line feed (LF) character. While not explicitly stated, the "Wire Service Transmission Guidelines for Slow-Speed Wires" (February 27, 1976) imply that the lines can take on one of two forms: textual lines and tabular lines.

**TEXTUAL LINES:**

Textual lines are the simpler and by far the most commonly used type of line in a New York Times article. They are simply a string of characters terminated by a line feed (LF) character.

**TABULAR LINES:**

Tabular material is the more interesting of the two New York Times data formats. New York Times tabular lines are also more challenging to format than the previously described Associated Press tabular lines. Like the Associated Press tabular material, New
York Times tabular material comes in two forms: single tabular lines and double (or side-by-side) tabular lines.

**Single Tabular Lines:**

This is the more widely used type of tabular line. A single tabular line starts with a single Tape Feed (TF) character. Next comes a field of textual data followed by a field of tabular data. There is no set code separating the textual and tabular fields such as the TFI character in Associated Press single tabular lines.

Single tabular lines are sent in justified format. The tabular field contains the data to be displayed as well as monitor spacing codes and fixed spacing codes. The significance and problems caused by these codes was already explained in the discussion of the raw Associated Press data format.

In summary, the format of a single New York Times tabular line is:

```
TF textual field tabular field LF
```

**Double Tabular Lines:**

Double tabular lines are once again an extension of single tabular lines. A New York Times double tabular line starts with three tape feed (TF) characters. Next comes the left textual field and left tabular field, followed by a single tape feed character. The right textual field and right tabular field follow the internal tape feed character. Finally, the line is terminated by a line feed (LF) character.

Once again, the right and left sides of double tabular lines are sent in justified format. Hence, they contain varying numbers of variable width fixed spacing characters to compensate for the various numbers of variable width letters in any given line. These fixed spacing codes cause the same problems that the fixed spacing codes in Associated Press articles cause. The main purpose of this project is to rid the articles of these codes and reformat the article so that they appear in their originally intended forms without the use of variable width letters and variable width spacing codes.
In summary, the New York Times format for double tabular lines is:

```
TF TF TF L-textual L-tabular TF R-textual R-tabular LF
field    field    field    field    field
```

Examples of single tabular lines and double tabular lines can be found in the appendices of this paper.

It was said that articles are passed to the New York Times formatter (the function tabbern) in the form of a character array. Each line in this article is a field in this array, terminated by a line feed character. The first five or six lines of the article contain the header of the article. The last line of the header contains the single word "text::" The article (input buffer) ends with a '\0' character.
CHAPTER 2: FORMATTING PROCESS OVERVIEW

TWO STEP A.P. FORMATTING PROCESS

The formatting process for Associated Press articles (the function "tabbera") involves two passes through each article. The purpose of the first pass is to perform a trial formatting of the article to determine a set of parameters that will be used in the actual formatting of the article (second pass through the article).

The first pass determines several values that are stored away in an integer array called const[]. The pass is performed by the function, fconst(name,const). Fconst returns the following values in the integer array const[].

const[1]=width of the longest textual line in the article.
const[3]=number of spaces from the right most edge of the screen to place the least significant digit of the right most column of tabular data.
const[4]=analogous to const[3], only it involves double tabular data. It is an adjustment factor used to adjust the width of each side of the formatted double tabular line.

With the first pass completed, the function, tabbera, now has the spacing information it needs to produce a formatted article with quad material correctly quadded and tabular material correctly lined up in columns.

Tabbera proceeds to read the article, line by line, using the function getaln(inbuf,l,const). Tabbera determines the type of each line (text, quad right, quad center, quad left, single tabular, double tabular), and dispatches the line to an appropriate formatting procedure. The procedures are as follows:

- **textual:**
  - quad left: quadl(l,t)
  - quad right: quadr(l,t,h,p)
  - quad center: quadc(l,t,h)
  - single tabular: tabber(l,t,tp)
  - double tabular: staber(l,t,tg)

Each of these procedures returns the formatted line in the character array t[] which is
then passed to function, `a__stuff(outbuf,t,const)`. `a__stuff` stuffs the line into the output buffer. When the end of the input buffer is reached, a "\0" character is placed at the end of the output buffer. The formatting complete, the output buffer is then passed to the outside routine that called `tabbera`.

Detailed descriptions of each of the functions as well as a detailed description of `tabbera` follow in the next chapter of this paper.

**TWO STEP NEW YORK TIMES FORMATTING PROCESS**

The formatting process for New York Times articles (the procedure "tabbern") involves two passes through each article. The first pass translates the article to a form that can be formatted by procedures written to format the Associated Press articles. The first pass also performs a trial formatting of the article to determine a set of parameters that will be used in the actual formatting of the article. The second pass does the actual formatting of the article.

The first pass determines several values that are stored away in an integer array called `const[]`. This pass is performed by the function `nconst(name,const)` which is called from within `tabbern`. `Nconst` returns the following values in the integer array `const[]`:

- `const[3]` = number of spaces from the right most edge of the screen to place the right most character in the right most column of a formatted single tabular data line.
- `const[4]` = adjustment factor which adjusts the width of each side of a formatted double tabular data line.

With the first pass complete, the function `tabbern(inbuf,outbuf)` then has the information it needs to produce a formatted article with tabular material correctly lined up in columns.

`Tabbern` proceeds to read the article, line by line, using the function `ngetaln(src,targ,const)`. `Tabbern` determines the type of each line (textual, single tabular, double tabular), and dispatches the line to an appropriate formatting procedure.
These procedures are as follows:

- **textual:**
  - single tabular: \texttt{tabber(l,t,tp)}
  - double tabular: \texttt{staber(l,t,tg)}

Each of these procedures returns the formatted line in the character array \( t[] \) which is then passed to the procedure, \texttt{a__stuff}, for stuffing into the output buffer. When the end of the input buffer is reached, a '0' character is placed at the end of the output buffer. The formatting complete, the output buffer is then passed to the outside routine that called \texttt{tabbern}.

Detailed descriptions of each of the functions as well as a detailed description of \texttt{tabbern} follow in the next chapter of this paper.
CHAPTER 3: THE CODE

3.1 The Formatting Drivers

TABBERA(INBUF,OUTBUF)

Tabbera is the main driving function of the AP formatting process. Its "black-box" function is to take as arguments an input buffer (inbuf[]) and an output buffer (outbuf[]). The input buffer is a character array containing the article to be formatted and is terminated by a \0 character. The output buffer as also a character array whose initial contents are unknown. When tabbera is finished, the input buffer remains untouched while the output buffer contains the formatted copy of the article, terminated by a \0 character.

\begin{verbatim}
tabbera(inbuf,outbuf) char inbuf[], outbuf[];
{
    int h;
    char 1[310], t[85];
    int const [7];
    int tp=78;
    int tg=38;

    This first section of code simply declares and initializes the variables that will be used by tabbera. The integer variable, h, will be used as a temporary register. The character array 1[] is used to hold the line of the input buffer that is being formatted. The character array, t[], is used to hold the formatted version of the line in 1[] before stuffing the line into the output buffer.

    The integer, tp, is the column on the screen of the right most digit of the right most column in the formatted single tabular lines. It is initialized to its default value, 78. The integer, tg, is the width of each side of formatted side-by-side tabular line. It is initialized to its default value, 38. Finally, const[] is an integer array used to hold pseudo-global variables whose values are needed by several functions.

    const [5]=0;
    const [6]=0;
    fconst(inbuf,const);
    const [5]=0;
    const [6]=0;
\end{verbatim}
Const[5] is the index of the next output buffer position to stuff. Const[6] is the index of the next input buffer position to read from. This section of code sets both indices to zero. It then calls the function fconst(inbuf,outbuf) which performs the trial formatting of the article, leaving key values in the integer array const[]. (see the description under "Two Step AP Formatting Process" and the description of fconst for details). Once fconst is finished, const[5] and const[6] are reset to zero to prepare for the final formatting of the article.

```
    tp=78-const[3];
    tg=38-const[4];

    if (tg <= 0)
        tg= 38;
```

This block of code changes the default values of tp and tg to reflect what was learned about the article by the first pass (performed by fconst). The whole purpose of the first pass is to make the formatted article easier to read. The general assumption is that table 1 below is easier to read than table 2. Table 2 would be formatted using the default values of tp and tg while table 1 is formatted with the updated values.

Table 1

<table>
<thead>
<tr>
<th></th>
<th>Rice</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>.362</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>6</td>
<td>11</td>
<td>9</td>
<td>.114</td>
<td></td>
</tr>
<tr>
<td>Jones</td>
<td>11</td>
<td>113</td>
<td>10</td>
<td>.372</td>
<td></td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th></th>
<th>Rice</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>.362</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>6</td>
<td>11</td>
<td>9</td>
<td>.114</td>
<td></td>
</tr>
<tr>
<td>Jones</td>
<td>11</td>
<td>113</td>
<td>10</td>
<td>.372</td>
<td></td>
</tr>
</tbody>
</table>

In addition to aesthetics, this scheme reduces the overall size of the article by reducing unnecessary amounts of blank space that would need to be stored along with the readable characters.

```
e3:     while (getaln(inbuf,1,const) != NULL)
{ 
```

The section of code labeled e3: is the section that reads each line, determines what type of line the line is, and then dispatches the line to an appropriate formatting function before stuffing the line into the output buffer. The "while (getaln(inbuf,1,const)!=NULL)" statement
ensures that each line is evaluated. Getaln reads the next line from the input buffer and stuffs the line into l[].

The first step in determining the type of a given line is to examine the first character. If the character (l[0]) is a TLI (Tab Line Indicator), then the line is some type of tabular line. If not, it is a textual or a quad line.

```c
if (l[0] != TLI)
{
    h=ffndr(l)-1;
    if (l[h] == QC)
    {
        h=h-1;
        quadc(1,t,h);
        a_stuff(outbuf,t,const);
    }
    else if (l[h] == QL)
    {
        quadl(1,t);
        a_stuff(outbuf,t,const);
    }
    else if (l[h] == QR)
    {
        if (h>=const[l])
        {
            if (h>= (MAXW-6))
                quadr(1,t,h,(h+1));
            else
                quadr(1,t,h,(h+5));
        }
        else
            quadr(1,t,h,const[l]);
        a_stuff(outbuf,t,const);
    }
    else
    {a_stuff(outbuf,1,const);
    }
}
```

This section of code formats quad and textual material. Since the first character of the line is not a TLI, the program must next determine what kind of quad or text line the line is. This information is in the character immediately preceding the LF (Line Feed) character that terminates the line.

The function, ffndr(l), the position (index) of the LF character in line l[] (see the
description of Iffndr). The quad identification character is therefore in l[llffndr(l)-1), hence h=llffndr-1.

The "if" block beginning with, "if (l[h]==QC)," handles quad center material. If the line is a quad center line, it is dispatched to the function quadc(l,t,h) which formats the line from the array l[] into the array t[] (see the description of quadc for details). The formatted line is then stuffed into the output buffer by a__stuff.

The "if" block beginning with, "if (l[h]==QL)," handles quad left material. Tabbera passes these lines to the function quadl(l,t) which formats the line from the array l[] to the array t[]. A__stuff then stuffs the formatted line into the output buffer.

The "if" block beginning with, "if (l[h]==QR)," handles quad right material. Quad right material is more complicated than quad left or quad center. The function, fconst, placed the width of the widest textual line in const[1]. Fconst, however, neglects quad material in its search of line widths. While unlikely, it is possible for a quad right line to be longer than the widest textual line. The block of code associated with the statement, "if (h>const[1])," handles this special case. What it does is to quad right the line stored in the array l[] and places it into the array t[]. However, in doing this it tries to carry out the "spirit" of quad right while preventing the line from taking up more than 80 characters (the maximum possible line width).

If the width of the quad right line is less than the maximum textual line width, there is no problem, and tabbera simply passes the line to quadr for simple application of the quad right rule. Finally, the formatted line is stuffed into the output buffer by a__stuff.

If the line is not a quad line, it is a textual line. Textual lines require no formatting and are stuffed into the output buffer by a__stuff.

If the first character of the line, l[0], is a TLI character, then the line is a tabular line. The next block of code handles this situation.

```c
else if (l[1] != TLI)
{
    tabber(l,t,tp);
}```
The two types of tabular lines are single tabular lines and double (side-by-side) tabular lines. Single tabular lines start with one TLI character. Double tabular lines begin with three TLI characters.

This block of code checks for more than one TLI character at the start of the line. If there is more than one, the line is a double tabular line and is dispatched to staber(l,t,tg) for formatting before being stuffed into the output buffer by a_stuff. If there is only one TLI character, the line is a single tabular line and is formatted by tabber(l,t,tp) before being stuffed into the output buffer.

e1: h=const[5];
    outbuf[h]='\0';
    return;

After all of the lines have been formatted and stuffed into the output buffer, all that remains is to attach the '\0' character to the end of the output buffer and then return. This final block of code (labeled e1:) performs this task.

**TABBERN(INBUF,OUTBUF)**

Tabbern is the main driving function of the New York Times formatting process. Its "black-box" function is to take as arguments an input buffer (inbuf[]) and an output buffer (outbuf[]). The input buffer is a character array containing the article to be formatted and is terminated by a '\0' character. The output buffer is also a character array, but whose initial contents are unknown. When tabbern is done, the output buffer contains the formatted copy of the article, terminated by a '\0' character.

tabbern(inbuf,outbuf)
char inbuf[], outbuf[];
{
    int h;
    char l[310], t[85];
    int const[8];
    int tp=78;
    int tg=38;

    This first section of code simply declares and initializes the variables that will be used by tabbern. The integer variable, h, will be used as a temporary register. The character array l[] is used to hold the line of the input buffer that is being formatted. The character array t[] is used to hold the formatted version of the line in l[] before stuffing the line into the output buffer.

    The integer, tp, is the column on the screen of the right most character of the right most column in the formatted single tabular line. It is initialized to its default value, 78. The integer, tg, is the width of each side of a formatted side-by-side tabular line. It is initialized to its default value of 38. Finally, the integer array const[] is used to hold pseudo-global variables whose values are used by several functions called by tabbern.

    const[5]=0;
    const[6]=0;
    const[7]=0;
    nconst(inbuf,const);
    const[6]=0;
    const[7]=0;

    Const[5] is the index of the output buffer position to stuff. Const[6] is the index of the next input buffer position to read. Const[7] is the value of const[6] prior to the last call to the function ngetaln. This block of code initializes these three indices to zero. It then calls the function nconst(inbuf,const). This function performs the first pass through the input buffer. Recall that this first pass is a trial formatting of the article. When the pass is completed, const[3] contains the number of spaces that can be removed from the default value of tp, the pointer to the right most character in the right most tabular column of a formatted tabular line. Const[4] contains the number of spaces that can be subtracted from the default value of tg, the width of each side of a double tabular line.
In addition to calculating the values of const[3] and const[4], nconst changes some of the codes in the article to make the article look like an associated press article. With these changes, the article can now be formatted with the same procedures that were developed to format AP articles. Before proceeding, const[6] and const[7], which were changed by nconst, are reset to zero in preparation for the final formatting of the article.

```plaintext
tp=78-const[3];
tg=38-const[4];
if (tg <= 0)
  tg= 38;
```

This block of code changes the default values of tp and tg to reflect what was learned about the article (input buffer) by the first pass (performed by nconst. The purpose of these changes is to make the article easier to read. The general assumption is that table 1 below is easier to read than table 2. Table 2 is formatted with the default values of tg and tp. Table 1 uses the updated values.

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
</tr>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Jones</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
</tr>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Jones</td>
</tr>
</tbody>
</table>

This scheme, in addition to aesthetics, reduces the overall size of articles by eliminating unnecessary amounts of blank space.

```plaintext
e3: while (ngetaln(inbuf,1,const) != NULL)
{
  if (1[0] != TLI)
  {
    a_stuff(outbuf,1,const);
  }
  else if (1[1] != TLI)
  {
    tabber(1,t,tp);
  }
```
This section reads each line, determines the type of the line, and then dispatches the line to an appropriate formatting function before stuffing the line into the output buffer. The while statement ensures that each line of the input buffer is evaluated. Ngetaln reads the next line from the output buffer and stuffs it into l[].

The first step in determining the type of a given line is to examine the first character. If the character, l[0], is a TLI, then the line is some type of tabular line. If l[0] is not a TLI, the line is a textual line, requires no formatting, and is stuffed into the output buffer.

If the first character is a TLI, then the second character must be checked to determine what type of tabular line is in l[]. If l[1] is also a TLI, the line is a double tabular line and is dispatched to staber for formatting. If l[1] is not a TLI, the line is a single tabular line and is formatted by tabber.

After all of the lines have been formatted and stuffed into the output buffer, all that remains is to attach a '\0' to the end of the output buffer and return the output buffer to the calling procedure.

### 3.2 The Formatting Procedures

**QUADL(L,T)**

Quadl(l,t) is the formatting function for quad left material. Its function is to format the line in the character array l[] into the character array t[], left justified. While it may not be totally correct, this function attempts to carry out the "spirit" of quad left by removing any
lead spaces that may occur in the unformatted line. The result is that the first readable character is formatted into position t[0].

```c
quad1(l,t)
char l[], t[];
{
    int i=0;
    int e=0;
    aclear(t);

    while (1[e]==SP)
    {
        e=e+1;
    }

    This first while loop finds the first printable character in the the unformatted line.

    while (1[e] != QL)
    {
        t[i]=1[e];
        i=i+1;
        e=e+1;
    }
    t[i]=LF;
    t[i+1]='\0';
    return;

    This final block of code stuffs characters from l[] into t[] until the quad left identification character is found in l[]. It then attaches a line feed (LF) character and a '\0' to t[] and returns the formatted line in t[].

QUADC(L,T,H)

Quadc(l,t,h) is the formatting function for quad center material. Its purpose is to format the line in the character array l[] into the character array t[] such that the line is centered in t[] and terminated with a LF character and a '\0' character.

```c
quadc(l,t,h)
char l[], t[];
int h;
```
This first block of code declares and initializes the variables that will be used by quadc(l[],t[],h). Recall that l[] is the unformatted line, t[] is the target array, and h is the position to the immediate left of the line feed character in the unformatted line. tmps is a temporary variable that will determine the number of spaces to place on either side of the line when formatting. d and e are pointers to positions in t[] and l[], respectively. Finally, off is an adjustment factor used to offset the formatted line slightly to the left of center. This is done because most of the textual lines do not take up the entire line width (MAXW characters) so offsetting quad center lines makes them appear to be centered a little better.

```c
int tmps;
int d;
int e;
int off=9;
```

This next block of code determines where (in t[]) to start filling in l[]. The filling in is done from the back towards the front. The variable e is set to the index of the character preceeding the line feed in the unformatted line. Quadc will try to center the unformatted line in a line of width MAXW-off. Hence, the number of spaces to each side of the formatted line should be half of the extra space (MAXW-off-e). tmps is therefore set to half of this extra space. The target array is cleared, and the variable d, the index of the right most character in the formatted line is then set to the line width (MAXW-off) less the space (tmps). The terminating line feed and \'\0\' characters are then added for convenience.

```c
while (e >= 0)
{
    if (d<0)
        return;
    t[d]=1[e];
    d=d-1;
    e=h;
    tmps=(((MAXW-off)-e)/2;
    aclear(t);
    d=(MAXW-off)-tmps;
    t[d+1]=LF;
    t[d+2]='\0';
```
All that remains is to stuff the unformatted line into the target array, $t[]$. The above while loop handles this, checking to make sure neither index ever gets below zero. The formatted line is returned in $t[]$.

**QUADR(L,T,H,P)**

```c
quadr(l,t,h,p)
char l[], t[];
int h, p;
{
    int e;
    int i;
    aclear(t);
    i=h-1;
    e=p;
    while (e>=0)
    {
        t[e]=l[i];
        if (i<=0)
            return;
        e=e-1;
        i=i-1;
    }
    return;
}
```

`Quadr(l,t,h,p)` is the formatting function for quad right material. Its function is to format the line in the character array $l[]$ into the character array $t[]$, such that the right most character in the formatted line is $t[p]$. In common terms, `quadr` slides the line over to the right margin.

The code is fairly self-explanatory. The argument $h$ contains the index of the quad right identification character in the unformatted line. The last printable character in $l[]$ is therefore $l[h-1]$. The function works from the end of the unformatted line toward to beginning, stuffing characters into the target array until all of the characters have been transferred or until there is no more room left in the target array, whichever comes first.

The call to the function `aclear(t)` clears the target array before any characters are
stuffed into it. Aclear(t) also attaches the terminating line feed and '0' characters. The line is
then formatted and returned in t[].

**TABBER(L,T,TG)**

Tabber is the formatting function for single tabular data. It takes the unformatted line
in the character array l[] and formats the line into the character array t[] such that the right
most character of the right most tabular column in the formatted line is in t[tg].

Recall the structure of the unformatted line from the discussion of the raw Associated
Press data format.

TLI textual field TFI tabular field LF

The strategy employed in formatting single tabular lines is to start by stuffing the
textual field into the target array, t[]. This stuffing is done by copying characters from l[] into
t[], from the start of the textual field towards the tabular field until the TFI character is
reached. The procedure then finds the end of the tabular field, and formats the tabular field
from its end to its beginning until TFI is again reached. By keeping the widths of tabular
columns constant and keeping tg constant in all calls to tabber, all single tabular lines will
format such that their tabular data line up in columns, right justified within the columns.

```c
tabber(l, t, tg)
char l[], t[];
int tg;
{
    int tp;
    int i=0;
    int a=1;
    int b;
    int j;
    int c;
    int tga=0;

    j=tg;
    /* clear the target array */
    aclear(t);

    tp=tg;
    t[tg+1]=LF;
    t[tg+2]='\0';
```
This first block of code declares and initializes the various variables that will be used. The arguments l and t are character arrays. l[] contains the unformatted line. When finished, t[] will contain the formatted line. Tg is the position (index) in t[] of the right most character of the right most formatted tabular column. The integer variables a, b, and c are all temporary index variables to positions in l[]. The integer variables i, j, tp, and tga perform similar roles in t[].

Aclear(t) sets the target array t[] to all blank spaces. Since the right most character in t[] will be at t[tg], t[tg+1] is set to LF (line feed), and t[tg+2] is set to '\0'. This eliminates the need to carry around extra spaces between t[tg] and the line feed character added by aclear(t) at position t[MAXW].

```
while (l[a] != TFI)
{
    if (i>MAXW)
        return (MAXW-1);
    if (l[a] == LF)
        return (MAXW-1);
    if ((decode(l[a])) == 0)
        a=a+1;
    else
    {
        t[i]=l[a];
        a=a+1;
        i=i+1;
        tga=tga+1;
    }
}
```

This while loop fills the target array with printable characters (as defined by the decode(a) function) from the unformatted array until the TFI character is reached in the unformatted array. There are several potential problems that this loop takes into account. The first is the possibility that due to some error the TFI character is missing. If this happens, the LF character will trigger the error condition and the line will be passed to the target array, unformatted, hence keeping the program running. This is the desired result because this condition is probably quite rare and hence is not worth correcting. If the line feed (LF) character is missing (which should never happen since it was detected by getaln prior to
calling tabber), an error condition exists. This is really the problem of two lines running
together. Tabber solves the problem by simply passing MAXW characters of l[i] into t[i],
unformatted.

Inspection of the code reveals that the function returns a value to the calling procedure.
As one may recall, this value is used by the function "fconst" in determining how to adjust the
location of the right most character of the right most formatted tabular column in the target
array. Since the error lines are not formatted, they should not be considered in this
calculation. The value returned in the error condition (MAXW-1) is equal to the default value
used by fconst and will not interfere with fconst's calculation.

The while loop looks at each character of the unformatted array. If the character isn't
TFI and if the two error conditions do not exist, decode(a) is called to see if the character is
printable. If it isn't printable, the character is ignored and the loop gets the next character. If
the character is printable, it is stuffed into the target array, and the source index, a, the
target index, i, and a special target index, tga, are each incremented. This loop continues until
l[a] is the TFI character. When this happens, t[i] points to the space immediately to the right
of the last character in the textual field. Tga is equal to i. With these values set, the next
step is to find the index of the end of the unformatted line (end of the unformatted tabular
field).

```
b=a;
while (l[b] != LF)
  b=b+1;
```

This block of code finds the end of the unformatted line. The source array index pointer,
b, is initially set equal to the integer, a, which is the index of the TFI character. b is then
incremented and l[b] tested to see if it is the LF character. The loop continues until l[b] is the
line feed character. Tabber assumes that the LF character exists because it was detected by
getaln prior to the call of tabber. At the conclusion of this block, l[b] is the end of the
unformatted tabular field.

The procedure now has the locations of the start and end of the unformatted tabular
field (a and b, respectively). It also knows the end of the formatted textual field in the target array (position i) and the index of the the right most character of the right most tabular column in the target array (position tg). The task that remains is to fill the tabular field into the target array between positions i and tg, formatted such that columns end in distinct preset screen columns that are fixed constant throughout the entire formatting process so that the columns in all of the single tabular lines line up correctly.

**Summary of what is known:**

**Unformatted line:**

```
TLI textual field  TFI  tabular field  LF
```

**Formatted line:**

```
t[i]:  textual field  space for tabular field  LF \0
```

The task that remains is to format the tabular field of the unformatted line ([a] to [b]) into the space in the formatted line reserved for the tabular field (t[i] to t[tg]). One can imagine several different solutions to this problem. The solution employed by tabber is based on several basic rules.

The tabular field of the formatted line should be broken into a series of columns of width CW. Each formatted tabular column must have its right most character lying in the right most position of one of these columns in t[].

**Example:** format this line (CW=4)

```
23 274  64  12
```

```
\|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
In addition to this restriction of where formatted tabular columns may lie within the formatted tabular field, there must be at least one space separating formatted tabular columns. Finally, it is assumed that the unformatted tabular field consists of strings of printable characters (as defined by the function prntbl(a)) separated by varying numbers of fixed and monitor spacing codes.

The strategy that is employed by tabber is to start at the back of the array and format one unformatted tabular column into the formatted array at a time until there is either no more room in the space reserved for the formatted tabular field or until the TFI character is reached, signifying that all columns have been formatted. This reduces the problem of formatting a number of varying width unformatted tabular columns to the problem of formatting one column into a "window" in the formatted field, sliding the window to the left, and repeating the process until there are no more windows or there are no more columns to format.

This strategy involves three steps. Step one is to find the right most printable character of a column and stuff it into the right most position in the window. Step two is to find and stuff the remaining characters in the column into the window and add the leading space. Step three is to slide the window (towards the front of the formatted line) into the next legal window space and loop back to step one until the process is complete.

```
j=tg;
tp=tg;
c=b-1;
```

This block of code sets up pointers in the formatted and unformatted arrays. j is the index of the next position in the formatted line to fill. Tp is the pointer to the right most edge of the current "window" being filled. c is the index of the character in the unformatted line that is being considered.

```
start: if (l[c] == TFI)
        return (j-tga);
```
if (prntbl(l[c]) == 0)
{
    c=c-1;
    goto start;
}

t[tp]=1[c];
j=tp-1;
c=c-1;

if (j <= tga)
    return 0;

This block of code accomplishes the first step of the process. Starting from index c, this code works its way from l[c] to l[0] until it finds the next printable character. If l[c] is the TFI character, all of the columns have been formatted and tabber is done. It returns j-tga which is the amount of space between the end of the textual field and the start of the tabular field.

If the character, l[c], is not printable, it is ignored. When the first printable character is found, it is placed in the right most position of the current window in the formatted tabular field, t[tp]. The "next-formatted-position" index, j, and the "next-unformatted-position" index, c, are then decremented. Before proceeding to find the rest of the characters in this current formatted window, j is tested against tga to see if there is any room left in the formatted tabular field. If not, the procedure is done and returns zero. If there is more room, the procedure continues.

middle: if (1[c] == TFI)
    return (j-tga);
if ((prntbl(l[c])) == 0)
{
    while (tp >= j)
        tp=tp-CW;
    if (tp <= tga)
        return (j-tga);
    c=c-1;
    goto start;
}
else
{
    t[j]=1[c];
c=c-1;
j=j-1;
if (j <= tga)
    return 0;
  goto middle;
}

This final block of code performs the second and third step of the process. Step two was to find and format the remaining characters in the column. \( l[c] \) is tested to see if it is TFI. If so, formatting is complete, and the procedure returns \( j-tga \), the amount of space between the end of the formatted textual field and the start of the formatted tabular field.

If \( l[c] \) is not TFI, it is then tested to see if it is printable. If so (the else block), \( l[c] \) is written into \( t[j] \). \( j \) and \( c \) are then decremented, and \( j \) is tested against \( tga \) to see if there is any more room in the tabular field of the formatted line. If there isn't any more room, formatting is complete (even if unformatted data is lost), and tabber returns 0. If there is more room, the procedure loops back to the label middle: to get the next character.

If \( l[c] \) is not printable (the if block), the procedure assumes that all of the characters of the current unformatted column have been found and formatted into the current window in \( t[…] \). It is then time for the third step of the process. The third step is the sliding of the "window" in the formatted tabular field.

Recall that \( t[j] \) points to the blank space immediately preceding the last space that was filled (the left most edge of the tabular column). The next available "window" in the formatted tabular field is found by decrementing \( tp \) (the index of the right most edge of the current window) by \( CW \) until \( tp \) is to the left of \( j \) (\( tp<j \)). Once this new window is found, \( tp \) is checked to see if it is in the space reserved for the tabular field. If so, \( c \) is decremented and control goes back to label start: for step one. If \( tp<tga \), there are no more legal windows in the formatted tabular field space and so formatting is complete.

Summary of Various Pointers

unformatted line:

\[
\begin{array}{|c|c|c|c|c|}
\hline
1[]: & TLI & \text{textual field} & TFI & \text{tabular field} & LF \\
\hline
\end{array}
\]
formatted line:

\[
\begin{array}{c|c|c}
\text{textual field} & \text{tabular field} & LF \\
\hline
32 & & \\
\end{array}
\]

**STABER(L,T,TG)**

Staber(l,t,tg) is the formatting function for double (or side-by-side) tabular data. It takes an unformatted line in the character array \( l[] \) and formats it into the character array \( t[] \) such that each side of the formatted array is \( tg + 1 \) characters wide.

Recall the structure of the unformatted line from the discussion of the raw data:

\[
\text{TLI TLI TLI L-text TFI L-tab CFS R-text TFI R-tab LF field field field field field}
\]

Staber performs its function by first formatting the left side (left textual field and left tabular field) of the unformatted line, and then formatting the right side (right textual field and right tabular field). It employs a strategy similar to that used by tabber.

In formatting the left side, staber starts from the left edge of the left textual field and stuffs the left textual field, character by character, into \( t[] \). This stuffing is done by copying characters from \( l[] \) into \( t[] \), from the left edge of the textual field toward the tabular field until the TFI character is reached. When TFI is found, the procedure finds the end of the left tabular field (the CFS character) and works backwards, formatting one column of the tabular field at a time until the TFI is again reached.

When the left side is formatted, staber then formats the right side, going forward through the right textual field and then backward through the right tabular field. As is the case with single tabular data, by keeping the tabular column widths constant and \( tg \) constant
in all calls to staber, all double tabular data will line up in columns, right justified within each column. The right justification within each column is a result of the procedure’s working backward through the tabular fields.

```c
staber(int l[], t[], int tg)
char l[], t[];
int tg;
{
    int tp, tl;
    int i=0, ri=0;
    int a=3, ra;
    int b, rb;
    int j, rj;
    int c, rc;
    int cl, dl, bl;
    int kgsa, kgsb, ksa;

    j=MAXW;
    rj=MAXW;

    kgsa=MAXW-1;
    kgsb=MAXW-1;
    ksa=MAXW-1;

    bl=tg;
    cl=tg+3;
    dl=cl+tg;

    This first block of code declares and initializes the various variables that will be used. The arguments l and t are character arrays. l[] contains the unformatted line and t[], upon completion, will contain the formatted line. The argument tg is the width-1 of each side of the formatted line. It is also the index of the right most character of the left side of the formatted line, also known as the index of the right most character of the right most tabular column in the formatted left tabular field.

    The variables a, b, c, ra, rb, and rc are all temporary variables used to index certain characters in l[]. The variables i, j, ri, rj, tp, and tl serve similar roles in t[]. Ksa, kgsa, and kgsb temporarily store values relevant to the spaces between different fields in the formatted array. This information is returned by the function in a form that is used by fconst in determining what the argument tg should be.
```
The variables \( bl, cl, \) and \( dl \) serve special roles. Recall that the argument \( tg \) is the (width-1) of each side of the formatted line. Since \( t[0] \) is the first character of the formatted line and \( t[tg] \) the last character of the left side, each side is \( tg + 1 \) characters wide. \( bl \) is simply \( tg \) so that \( t[bl] \) is the last character in the left side of the formatted line. \( cl = tg + 3 \). It is intended that there be three spaces between the left and right sides of the formatted line, therefore \( t[cl] \) is the first character in the right side of the formatted line. It is intended that \( dl \) be analogous to \( bl \), hence \( t[dl] \) is the last character in the right side of the formatted line. The maximum value of \( tg \) that is used by the calling functions is 38, therefore the maximum value of \( dl \) is 79. Hence, a formatted line should always be able to fit on an 80 column display.

```c
aclear(t);
t[dl+1]=LF;
t[dl+2]='\0';
tp=bl;
```

This block of code fill the target array with spaces and adds a terminating LF (line feed) character as well as a '\0' character. It also initializes a pointer, \( tp \), which is used later in the procedure.

```c
while (l[a] != TFI)
{
    if (i>MAXW)
        return ksa;
    if (l[a] == LF)
        return ksa;
    if ((decode(l[a])) == 0)
        a=a+1;
    else
    {
        t[i]=l[a];
        a=a+1;
        i=i+1;
    }
}
```

This block of code stuffs the left textual field from \( l[] \), the source array, into \( t[] \), the target array. Recall that the source index, \( a \), is initialized to three since \( l[0], l[1], \) and \( l[2] \) are each TLI characters. The target index, \( i \), is initialized to zero.
The code starts with l[3] and moves to the right, examining each character until it reaches the TFI character. If the character l[a] is printable (as defined by the function decode(a)), the character is stuffed into t[i] and i is incremented. If for some reason i becomes greater than the maximum line width (MAXW) or if the line feed (LF) character appears in the left textual field, an error condition exists. Staber returns the part of the unformatted array that has already been stuffed into t[]. Staber also returns the integer, MAXW-1. Recall that this integer returned by staber is used by fconst in determining the proper value of tg. Ksa=MAXW-1 is a number so big that it will be ignored by fconst. This is the desired result since the line is an error line and should not be considered a valid double tabular line. Hence, the line should have no bearing on how valid double tabular lines are formatted, and should not be considered when determining the proper value of tg.

When l[a] is TFI, the entire left textual field has been formatted into t[]. The next step in the process is to find the end of the left tabular field.

\begin{verbatim}
b=a;
while (l[b] != CFS)
{
    if (l[b]==LF)
        return ksa;
    b=b+1;
}
\end{verbatim}

This block of code finds the end of the left tabular field. The while loop starts from the TFI character and moves through the left tabular field until it finds the CFS (center field separator) character which terminates the left tabular field. If a LF is present in the left tabular field, the line is an error line. This is not really possible since the unformatted line is terminated by a LF. What is really in error is the CFS is missing. Either way, the line is incorrect and should not be formatted as a double tabular line. Returning the value of ksa=MAXW-1 causes fconst to ignore the line and prevents it from being considered in the calculation of tg by fconst.

The procedure now knows l[a] and l[b], the positions of the start and the end of the unformatted left tabular field. It also knows t[i] and t[bl], th positions of the start and the end
of the space in the target array that has been reserved for the formatted left tabular field.

The next step is to format the left tabular field from \$l\$ into \$t\$. Staber uses the same strategy outlined in the function, tabber, to accomplish this task. The solution is based on several rules. First, the space reserved for the formatted left tabular field (\$t[i]\$ to \$t[bl]\$) should be broken into a series of columns of width SCW. Next, each formatted tabular column must have its right most character in the right most position of one of these tabular columns in \$t\$.

![Diagram of formatted left tabular field]

**Example:** format this line \(SCW=4\)

```
23  274  64  12
```

In addition to this restriction of where formatted tabular columns may lie in the formatted left tabular field, there must also be at least one space separating each formatted tabular column.

**Example:** format this line \(23 12 14 .273\)

```
{ 2 3 1 2 1 4 . 2 7 3  }
```

Finally, it is assumed that the unformatted left tabular field consists of strings of printable characters (as defined by the function, prntbl(a)), separated by varying numbers of fixed and monitor spacing codes, none of which are considered printable.

The strategy that is employed by staber is to start at the back of the unformatted tabular field and format one unformatted tabular column at a time into a window in the formatted line until there is either no more room in the formatted tabular field space or until
the TFI character is reached, signifying that all of the columns in the left tabular field have been formatted. This reduces the problem of formatting a varying number of varying width unformatted tabular columns to the problem of formatting one column into a window in the formatted tabular field, sliding the window, and repeating the process until all the columns have been formatted.

This strategy involves a three step process. The first step is to start from the end of the unformatted left tabular field and move towards the front until the first printable character is found. This character is then stuffed into the right most position in the current window before proceeding to step two. The second step is to find and format the rest of the characters in the window and add the leading space. The final step is to slide the window over the the next legal window position and go back to step one until all of the columns have been formatted.

c=b;
j=tp;

These two lines set up some pointer in the formatted and unformatted lines. The variable c is the index of the present character in the unformatted line that is being examined. The variable j is the index of the next position in the formatted line to fill. Tp was initially set to bl which is the index of the right most character in the right most tabular column in the left tabular field. Hence, j is set to the right most position in the first (right most) window.

```
start:  if (1[c]==TFI)
  {
    kgsa=j-i;
    goto rside;
  }
if (prntbl(1[c])==0) {
  c=c-1;
  goto start;
}
t[tp]=1[c];
j=tp-1;
c=c-1;
if (j<=i) {
  kgsa=1;
  goto rside;
```
This block of code accomplishes step one of the three step process. Starting from the pointer c, this code works its way from l[c] to l[0] until it finds the next character that is printable (as defined by the function prntbl(a)). If the character l[c] is the TFI character, all of the columns in the left tabular field have been formatted. The procedure saves the number of spaces between the end of the left textual field and the start of the left tabular field (j-i) in kgsa. This value is used later in the procedure. The procedure then continues to the label, rsde:, where the right side of the unformatted line is formatted.

If l[c] is not printable, it is ignored and the procedure examines the next character. When the first printable character is found, it is placed in the right most position the current window of the formatted tabular field (t[tp]). The "next-formatted-position" pointer, j, and the "next-unformatted-position" pointer, c, are then decremented. Before proceeding to find the rest of the characters that belong in the current window, j is tested against i to see if there is any room left in the formatted tabular field. If not, kgsa is set to 1 and the program skips to rsde to format the right side. If there is more room, the procedure continues.

```
middle: if (l[c] == TFI) {
    kgsa=j-i;
    goto rsde;
} if ((prntbl(l[c])) == 0) {

    while (tp >= j)
        tp=tp-SCW;
    c=c-1;
    if (tp <= i) {
        kgsa=1;
        goto rsde;
    }
    goto start;
} else {
    t[j]=l[c];
    c=c-1;
    j=j-1;
```
if (j <= i) {
    kgsa=1;
    goto rside;
} goto middle;
}

This block of code performs steps two and three of the three step process. Step two was to find and format the rest of the characters in the current tabular column. l[c] is tested to see if it is the TFI. If so, the left tabular field is completely formatted. The space between the left textual field and left tabular field, j-i, is stored in kgsa for future use. The procedure then goes to rside to format the right side of the unformatted line.

If l[c] is not the TFI, it is then tested to see if it is printable (as defined by prntbl(a))). If so (the else block of the code), l[c] is written to t[j]. The variables j and c are then decremented, and j is tested against i to see if there is any more room in the space reserved for the formatted left tabular field. If there isn’t any more room, kgsa is set to 1, and the program goes to rside to format the right side of the unformatted line. If there is more room, the procedure loops back to middle to get the next character.

If l[c] is not printable (the if block of the code), the procedure assumes that all of the characters of the current unformatted column have been found and formatted into t[]. The third step of the process must now be performed. The window in the formatted tabular field must be slid to the next open window position.

Recall that t[j] currently points to the next space to fill in the formatted line. This is a blank space immediately to the left of the column that was just formatted. The next available window is found in the "while" loop by decrementing tp by SCW until tp is less than j. The window is then checked to see if it lies in the space reserved for tabular data in the formatted array. If the window lies in the textual field, kgsa is set to 1, and the procedure goes to rside to format the right side. If the window lies in the allotted space, the program loops back to start to get the next column.
Summary of Various Pointers

unformatted line:

\[ \begin{array}{cccccccccc}
\text{TLI} & \text{TLI} & \text{TLI} & \text{L-text} & \text{TFI} & \text{L-tab} & \text{CFS} & \text{R-text} & \text{TFI} & \text{R-tab} & \text{LF} \\
\end{array} \]

left side | right side

formatted line:

\[ \begin{array}{ccccccc}
\text{i} & \text{bl} & \text{cl} & \text{dl} \\
\end{array} \]

\[ \begin{array}{ccccccc}
\text{left textual} & \text{left tabular} & \text{LF} \\
\end{array} \]

At this point the left side of the unformatted line has been formatted into the target array. The procedure must now format the right textual field and right tabular field into the space between \( t[c_l] \) and \( t[d_l] \). It uses the same process that was used in formatting the left side.

\[ \text{rside: } r_a=b; \]
\[ r_i=c_l; \]
\[ r_j=\text{MAXW}-1; \]
\[ t_l=d_l; \]

\[ \text{while } (l[r_a] != \text{TFI}) \]
\[ \{ \]
\[ \text{if } (r_i>\text{MAXW}) \]
\[ \text{return } \text{ksa}; \]
\[ \text{if } (l[r_a] == \text{LF}) \]
\[ \text{return } \text{ksa}; \]
\[ \text{if } ((\text{decode}(l[r_a])) == 0) \]
\[ r_a=r_a+1; \]
\[ \text{else} \]
\[ \{ \]
\[ t[r_i]=l[r_a]; \]
\[ r_a=r_a+1; \]
\[ r_i=r_i+1; \]
\[ \} \]
\[ \} \]
This block of code stuffs the right textual field into the target array, starting from position \( t[cl] \). It is completely analogous to the code already described with the following variable transformations:

\[
\begin{align*}
    i & \Rightarrow ri \\
    j & \Rightarrow rj \\
    a & \Rightarrow ra \\
    b & \Rightarrow rb \\
    c & \Rightarrow rc \\
    tp & \Rightarrow tl
\end{align*}
\]

Now that the right textual field has been formatted, the next step is to find the end of the right tabular field.

\[
\begin{align*}
    rb & = ra; \\
    \text{while} \ (l[rb] \neq LF) & \ \\
        \quad \quad rb & = rb + 1;
\end{align*}
\]

This code simply increments \( rb \) until \( l[rb] \) contains the LF character which terminates the line (and the right tabular field).

\[
\begin{align*}
    \text{rc} & = rb; \\
    \text{restart: if} \ (l[rc] == TFI) & \ \\
        \quad \quad \{ & \ \\
            \quad \quad \quad kgsb = rj - ri; & \ \\
            \quad \quad \quad ksa = \text{min}(kgsa, kgsb); & \ \\
            \quad \quad \quad \text{return} \ ksa; & \ \\
        \quad \quad \} & \ \\
    \quad \quad \text{if} \ (\text{prntbl}(l[rc]) == 0) & \ \\
        \quad \quad \quad \{ & \ \\
            \quad \quad \quad \text{rc} = \text{rc} - 1; & \ \\
            \quad \quad \quad \text{goto} \ \text{restart}; & \ \\
        \quad \quad \} & \ \\
    \quad \quad \text{else} & \ \\
        \quad \quad \{ & \ \\
            \quad \quad \quad t[t1] = l[rc]; & \ \\
            \quad \quad \quad rj = \text{t1} - 1; & \ \\
            \quad \quad \quad \text{rc} = \text{rc} - 1; & \ \\
        \quad \quad \} & \ \\
    \quad \quad \text{if} \ (rj \leq ri) & \ \\
        \quad \quad \quad \text{return} \ ksa;
\end{align*}
\]

This block finds the first printable character of the current column that is being formatted. Starting from the index \( rc \) (which is intitialized to the end of the line), the code works its way from \( l[rc] \) to \( l[ri] \) until it finds the next printable character. If \( l[rc] \) is the TFI,
all of the columns have been formatted. The procedure places \( r_j - r_i \) in \( kgsb \). This is the number of spaces between the formatted right textual field and the formatted right tabular field. Recall that \( kgsa \) contains the same value for the left side. The procedure is finished formatting. The character array \( t[] \) holds the formatted line. The procedure returns the smaller value of \( kgsa \) and \( kgsb \).

If \( l[rc] \) is not printable, it is ignored and the next character is checked. When the first printable character is found, it is placed in \( t[tl] \), the right most position in the current formatting window. The "next-formatted-position" index, \( r_j \), and the "next-unformatted-character" index, \( r_c \), are then decremented. Before proceeding to find the rest of the characters, \( r_j \) is tested against \( r_i \) to see if there is any more room left in the right tabular field space. If no more room is left, formatting is complete and \( \text{staber} \) returns \( ksa \). This condition should only exist if the line is too long since the width of the formatted right and left sides is chosen to avoid the problem. The value returned by \( \text{staber} \) is used by \( \text{fconst} \) to choose the optimal width. \( \text{fconst} \) uses the maximum possible width in its test formatting of data. If this condition occurs with the maximum possible width, it will occur with some smaller (optimal) width. Such lines are too long to be formatted with this process and should not be considered in \( \text{fconst} \)'s calculations. \( ksa \) is initially set to \( \text{MAXW-1} \) which will be ignored by \( \text{fconst} \) because it is too big.

```
rmidle: if (l[rc] == TFI)
    kgsb=rj-ri;
    ksa=min(kgsa,kgsb);
    return ksa;
if ((prntbl(l[rc])) == 0)
{
    while (tl >= rj)
        tl=tl-SCW;
    rc=rc-1;
    if (tl <= ri)
        return ksa;
    goto rstart;
} else
```
This final block of code finds the rest of the characters of the given column, adds the lead space, slides the current window over, and loops back to rstart to get the next column.

l[rc] is tested to see if it is the TFI. If so, formatting is complete. Kgsb is set to rj-ri, the space between the right textual field and right tabular field. The smaller value of kgsb and kgsa (the analogous value for the left side) is returned by the calling function.

If l[rc] is not the TFI, it is tested to see if it is printable. If so, the character is stuffed into t[rj]. Rj and rc are then decremented. Before continuing, rj is compared to ri to see if any more space exists in the formatted right tabular field. If no more space exists, the line is too long and the program returns ksa which will trigger the error condition in fconst (this condition was previously described).

If l[rc] is not printable, the procedure assumes that all of the characters of the unformatted column have been found and formatted into t[]. The window must now be slid to the next available tabular field window position.

Recall that t[rj] is the space immediately preceding the formatted tabular column. The next available window in the formatted right tabular field must be to the left of t[rj]. It is found by decrementing tl (the pointer to the right most position in the current window) by SCW until tl is to the left of rj (tl<rj). Tl is then checked to see if it is in the space allotted for the formatted right tabular field. If so, rc is decremented and the procedure goes back to rstart to format the next column. If there isn’t any more room, once again the line is too long and ksa is returned to trigger the error condition in fconst.

This whole discussion of the error condition triggered by ksa seems complicated, but it
really isn’t. The value returned by staber is used only during the trial formatting of the article by fconst. During this trial formatting, fconst does not care what is returned in t[]. Fconst bases its decision on the optimal value of the staber argument, tg, by trying to format double tabular lines with tg equal to its maximum possible value of 38. Fconst finds the maximum amount of space it can reduce from tg without causing these line overflow conditions to occur. Staber returns the minimum of the space between the right textual field and right tabular field, and the space between the left textual field and left tabular field. Fconst returns the minimum over all calls to staber of this minimum. This is presumably the number of spaces that can be subtracted from the maximum width of a formatted side by side tabular line without causing the error condition. The second pass through an article by tabbera or tabbern where the actual formatting occurs ignores this value that is returned by staber and looks strictly at t[].
3.3 Associated Press Utility Functions

FCONST(NAME, CONST)

Fconst performs the first pass of the two step AP formatting process. Its purpose is to perform a trial formatting of the article stored in the character array input buffer called name[]. In formatting, it uses the default values of tp and tg. As was previously described in the section on tabbera, tp is the screen column of the right most character in the right most column of a single tabular line. The default value of tp is 78. Tg is the width of each side of a formatted double tabular line. The default value of tg is 38.

Fconst has four main functions:

1. To store in const[1] the width of the widest textual line in the article.
2. To store in const[2] the average width of a textual line.
3. To store in const[3] the number of spaces that tp can be decreased (slid to the left) and still produce a readable, formatted, single tabular line. To put it another way, fconst formats all single tabular lines and looks at the amount of space between the textual field and tabular field of each formatted line. It looks at every line and and returns in const[3] the minimum amount of space over all of the lines.

   i.e.

   line 1  text field 1   tab field 1
         space 1

   line 2  text field 2   tab field 2
         space 2

   space 2 is less than space 1, therefore const[3] is space 2.

4. To store in const[4] a value analogous to const[3] only both the left field and right field must be considered. This value is used to determine the width of each side of the final formatted line.

fconst(name, const)
int const[];
char name[];
This first section of code declares and initializes the variables that will be used by fconst. The argument, name[], is a character array which is the dummy name for the input buffer. The argument, const[], is an integer array in which the results of fconst will be stored. l[] is a character array used to hold the line of the input buffer that is currently being formatted. t[] is another character array which serves no real purpose but to act as a place holder for the formatting functions used by fconst.

Ki is an integer variable used to store the number of lines in the article. It is used in a division at the end of the function and is initialized to one to prevent a division by zero error. Ks is the running value of const[3] (previously described). It is initialized to 78, a value which is bigger than any value corresponding to a valid single tabular line. Hence, any single tabular line in the article will reduce this value. Kd is the running value of const[4]. It is initialized to 40 which is bigger than any value that is possible for a valid double tabular line. Hence, any double tabular line will change this value.

Kt is the running value of const[1], the maximum textual line width. It is initialized to zero so that any textual line will change its value. Ka serves two purposes. Its running value is the total number of characters in all of the textual lines in the article. When divided by ki (number of textual lines), ka becomes the average textual line width which will become const[2]. As a running total, ka is initialized to zero. The two remaining variables, k and h, are temporary integer variables.

```c
f2: while (getaln(name,1,const) != NULL)
{
```
Fconst reads each line and performs a trial format on each line. The while loop ensures that each line is read. The first step of the formatting is to determine the type of a line. The first step in doing this is to look for a TLI character as the first character in the line, l[0].

```c
if (l[0] != TLI)
{
    h=1ffndr(1)-1;
    if (l[h]==QC || l[h]==QL || l[h]==QR)
    {
        else
        {
            if (h>kt)
                kt=h;
            ki=ki+1;
            ka=ka+h;
        }
    }
}
```

If l[0] is not a TLI character, the line is either a text line or a quad line. The above block of code checks to see if the line is a quad line, and if so, ignores it. If the line is a text line, the code updates kt if the line is greater that the previously longest line. The code also increments the line counter, ki, and updates the running total of textual line characters, ka.

```c
else if (l[1] != TLI)
{
    k=tabber(l,t,78);
    if (k <= ks)
        ks=k;
}
else
{
    k=staber(l,t,38);
    if (k<=kd)
        kd=k;
}
```

If l[0] is a TLI, the line is a tabular line. The next step is to determine whether it is a single tabular line or a double tabular line, and then to dispatch to the appropriate formatting routines (tabber and staber).

```c
Tabber and staber have been written to return an integer in addition to the formatted line. Tabber returns the number of spaces between the end of the textual field and the star of the tabular field in the formatted single tabular line. Staber returns either the number of
```
characters between the left textual field and the left tabular field, or the number of characters
between the right textual field and right tabular field, whichever is smaller.

**single tabular line:**

```
  text field / n spaces / tabular field
```

*tabber returns n*

**double tabular line:**

```
  L text/n space/L tab/R text/m spaces/R tab
```

*staber returns minimum(n,m)*

Fconst checks the second character in the line, l[1]. If it too is a TLI, the line is a double
tabular line and is dispatched to staber for formatting. If the value returned by staber is
smaller than the previous running value of const[4] (which is stored in the temporary, kd), kd
is updated.

If l[1] is not a TLI, the line is a single tabular line and is dispatched to tabber. If the
value returned by tabber is smaller than the previous running vale of const[3] (stored in ks),
ks is updated.

```c
fl:  if (ki==1)
    ;
  else
    ki=ki-1;
  ka=ka/ki;
  const[1]=kt;
  const[2]=ka;
  const[3]=ks;
  const[4]=kd;
  return;
}
```

Once all of the lines have been read and formatted, all that remains is to put the correct
values into const[] and return. The maximum textual line width, kt, is stuffed into const[1].
Similarly, ks is stuffed into const[3] and kd into const[4]. All that remains is a calculation of
the average textual line width. Ki, which was initially set to 1, is adjusted (if necessary) and
the average line width is ka/ki. This value is then stuffed into const[2] and the function
returns.
GETALN(src, targ, const)

getaln(src, targ, const)
char src[], targ[];
int const[];
{
    int i=0;
    int j;
    j=const[6];
    while (src[j] != LF)
    {
        if (src[j]=='\0')
            return NULL;
            targ[i]=' ';
        else
            targ[i]=src[j];
        i=i+1;
        j=j+1;
        if (i>=300)
            return NULL;
    }
    i=i-1;
    while (targ[i]==SP)
    {
        i=i-1;
    }
    targ[i+1]=LF;
    targ[i+2]='\0';
    j=j+1;
    const[6]=j;
    return 1;
}

Getaln is the function used by tabbera and fconst to read a line of text from the input buffer (character array src[]) and return that line in the character array targ[]. It expects to find the index of the next input buffer character to read in const[6], and upon exit leaves the new index of the next input buffer character to read in const[6].

Recall that the input buffer is one long character array, terminated by a '\0' character. The array contains a number of fields, each terminated by a line feed (LF) character. Each field corresponds to one line of text (in the raw article).

Getaln uses two integer variables as pointers to the next src[] location to read from and
the next targ[] location to stuff. The src[] pointer is j and is initialized to const[6]. The targ[] pointer is i and is initialized to zero.

The reading and stuffing takes place in the first while loop. The loop reads a src[] character and sees if it is the last character of the array ('\0'). If so, getaln returns NULL. If not, getaln checks to see if the character is one of the so called nuisance characters (ENL, EML, Upper Rail, Lower Rail) and if not it stuffs the character into targ[i] and increments i and j. If the character is a nuisance character, a space is stuffed into the target array and i and j are incremented. This reading and stuffing continues until the src[] character is a LF character.

When the LF is reached, the second while loop starts from the end of the target array and goes back to the last character that is not a space. The LF character is placed immediately following this last non-space character, and a '\0' is tacked on the end. This operation serves two functions. First, it eliminates unnecessary spaces at the end of a line. Most importantly, it ensures that quad material will be properly detected and formatted. Recall that quad lines have a quad identification character preceeding the LF character at the end of the line. Tabbera counts on the quad identification character being immediately to the left of the LF with no spaces separating the two. A typographical error at the wire service that inserts a space between the quad identification character and the LF would trick tabbera into thinking the quad line is a textual line, and the line would not be formatted as expected. The second while loop corrects this problem.

Finally, j is incremented and stuffed into const[6] (so the next call to getaln knows where to start reading from the src[] array), and the function returns a 1 in addition to the line in targ[].

One can imagine a case in which, due to some error, there were no more LF or '\0' characters in the input buffer. Getaln would appear to keep reading forever (or until the pointers got too large). Getaln solves this problem by assuming the error condition and returning NULL after 300 characters are read without encountering a LF.
LFFNDR(S)

```c
lffndr(s)
char s[];
{
    int i;
    for (i=0;i<= 300;i=i+1)
    {
        if (s[i] == LF)
            return i;
    }
    return 0;
}
```

Lffndr(s) is used by tabbera and fconst to find the index of the line feed (LF) character at the end of a line. In effect, lffndr returns the length of the line in the character array s[]. lffndr assumes that an error exists if the LF does not occur within the first 300 characters of the line and subsequently returns 0.
3.4 New York Times Utility Functions

NCONST(NAME, CONST)

Nconst performs the first pass of the two pass New York Times formatting process. It has two basic functions. The first is to perform a trial formatting of the article stored in the character array name[], much like that performed by its associated press counterpart, fconst. The purpose of this trial formatting is to gather some information concerning how wide single tabular lines are and how wide double tabular lines are within the article. This information is used by tabbern to adjust certain key variables away from their default values.

The two variables in question are tp and tg in the function tabbern. As was previously described, tp is the screen column of the right most character in the right most tabular column of a formatted single tabular line. Its default value is 78 which is the space immediately to the right of the left most edge of an 80 column display (columns 0 through 79). Tg is the width of each side of a formatted side by side tabular line. Its default value is 38.

Observe that each formatted single tabular line is of the form:

```
  textual field    space    tabular field
```

Nconst returns in const[3] the number of spaces in that single tabular line in the entire article that has the minimum number of spaces. By subtracting this number of spaces from tp, the tabular field of every single tabular line is slid to the left by that number of spaces. As was described in the discussion of tabbern, this makes for a more readable output. Nconst uses the default value of tp in its trial formatting to get the value it returns in const[3].

Also observe that each formatted double tabular line is of the form:

```
  L-tab space 1 L-text R-tab space 2 R-text
  field       field      field      field
```

Space 1 and space 2 are a function of how wide each side of the formatted double tabular line is. This is the variable tg in tabbern. Reducing the size of tg reduces the amount of extra space that must be carried around. Nconst uses the default value of tg, 38, and
formats all of the double tabular lines. It then returns in const[4] the smallest amount of space used over the entire article.

In addition to returning these two values in const[], nconst has a second, more important function. This is to translate the input buffer from its New York Times format to the Associated Press format. This translation was already described in the overview of the NYT formatting process. To review, however, it involves translating the tape feed (TF) characters at the start of every tabular line (one for single tabular lines, three for double tabular lines) into tab line indicator (TLI) characters. The TF character in the middle of a double tabular line must also be translated to a center field separator (CFS) character. Finally, the start of the tabular fields in both the single and double tabular lines must be found and marked by a tab field indicator (TFI) character. Once this translation has been performed, the article can be formatted using many of the same formatting routines used to format Associated Press articles.

Nconst doesn't actually do the formatting of lines. Rather, like tabbern, it simply determines the type of a line and dispatches the line to an appropriate formatting procedure.

```c
nconst(name,const)
  int const[];
  char name[];
  {
    char l[310];

    int ks=78;
    int kd=40;
    int k=0;

    This first section of code declares and initializes the variables that will be used by nconst. The character array argument name[] is the input buffer. The integer array const[] will be used to hold indices within name[] as well as to return the results of nconst's pass through the article.

    The character array l[] is used to hold the line of the article that is currently being formatted. The formatting procedures do not return an actual formatted line so no target array (such as t[] in fconst) is needed.
```
Ks is the variable which will hold the running value of const[3] during the formatting process. It is initialized to its default value of 78 which is guaranteed to be bigger than the value of any correctly formatted single tabular line. Hence, any single tabular line in the article will change this value. Kd is the running value of const[4] during the formatting process. It too is initialized to a value that is bigger than any value corresponding to a valid double tabular line. Hence, any double tabular line in the article will change its value. K is a temporary variable.

```c
f2: while (ngeta1n(name,1,const) != NULL)
{
   if (l[0] != TF)
   ;
   else if (l[1] != TF)
   {
      k=ntabs(1,78,name,const);
      if (k <= ks)
         ks=k;
   }
   else
   {
      k=nstab(1,38,name,const);
      if (k<=kd)
         kd=k;
   }
}
```

This section of code performs the trial formatting of the article. The while loop reads each line of text, determines the line’s type, and dispatches the line to an appropriate formatting function.

To determine a line’s type, the first character of the line, l[0], is examined. If the first character is not a TF (tape feed), the line is a textual line and is ignored by nconst. If l[0] is a TF, the second character is checked.

If the second character is also a TF, the line is a double tabular line and is dispatched to nstab for formatting. Nstab translates the three TF characters at the start of the line into TLI’s. It also marks the start of the tabular fields with TFI’s and translates the internal TF character into a CFS. Finally, nstab returns either the number of spaces between the left textual field and left tabular field or the number of spaces between the right textual field and
right tabular field, whichever is smaller. If this value is smaller than the previous running value of const[4] (kd), the value is stored away in kd.

If l[0] is a TF but l[1] is not, the line is a single tabular line and is dispatched to ntabs for formatting. Ntabs translates the TF character at the start of the line to a TLI. It also marks the start of the tabular field with a TFI, and returns the number of spaces between the textual and tabular fields. If this number is smaller then the previous running value of const[3] (ks), the number is stored away in ks.

Inspection of the discussion and code of ntabs and nstab show that the translation occurs in the input buffer (name[]), not in the individual line (l[]). The index of the first character of the line within the input buffer was left in const[7] by the call to ngetaln that produced the line in l[].

```c
f1:    const[1]=78;
      const[2]=78;
      const[3]=ks;
      const[4]=kd;
      return;
}
```

This last block of code ties up the loose ends and returns its results in const[]. This block places the final running value of const[3] (ks) into const[3] and the final running value of const[4] (kd) in const[4] before returning.

Const[1] and const[2] are set equal to 78. Const[1] and const[2] are unused places in const[]. They carry over from the development of fconst and are being reserved for possible future use. They are set equal to 78 so that their value is at least known.

**NGETALN(SRC,TARG,CONST)**

```c
ngetaln(src,targ,const)
char src[], targ[];
int const[];
{
    int i=0;
    int j;
    j=const[6];
    while (src[j] != LF)
{  
if (src[j]=='\0')  
  return NULL;
targ[i]=' ';  
else  
targ[i]=src[j];
  i=i+1;
  j=j+1;
  if (i>300)  
    return NULL;
}
targ[i]=LF;
targ[i+1]='\0';
j=j+1;
const[7]=const[6];
const[6]=j;
return 1;
}

Ngetaln(src,targ,const) is the procedure used to read a line from the input buffer during the New York Times formatting process. It is similar to the procedure getaln(src,targ,const) which is used to read a line from the input buffer during the Associated Press formatting process.

Ngetaln reads a line of text from the character array src[] to the character array targ[]. It expects to find the index of the next input buffer character to read in const[6]. Upon exit it leaves this initial value of const[6] in const[7] and leaves the new index of the next input buffer character to read in const[6].

Recall that the input buffer is a character array terminated by a '\0' character. The array contains a number of fields, each terminated by a line feed (LF) character. Each field corresponds to a line of text.

Ngetaln uses two integer variables as pointers to the next src[] location to read and the next targ[] location to stuff. The src[] pointer is j which is initialized to const[6]. The targ[] pointer, i, is initialized to 0.

The reading and stuffing takes place in the while loop. The loop reads a src[] character and sees if it is the last character of the array ('\0'). If so, it returns NULL. If not, ngetaln
checks to see if the character is a nuisance character. If so, it stuffs a space into the target array. If not, it stuffs the character into targ[i] and increments i and j. This reading and stuffing continues until the src[] character is a LF character. When this happens, a LF and a \0 are tacked onto the end of targ[]. j is then incremented and stored in const[6] as the new next-src-pointer after the old value of const[6] has been stored in const[7]. Const[7] is, therefore, the index in src[] of the first character in targ[].

One can imagine a case in which, due to some error, there were no more LF or \0 characters left in the input buffer. Ngetaln would appear to keep reading forever. Ngetaln solves this problem by assuming the error condition after 300 characters have been read without encountering a LF or \0, and returns NULL.

NTABS(L,TG,NAME,CONST)

Ntabs is the pre-formatting procedure for New York Times single tabular data lines. By this, it is meant that it is the procedure called by nconst to trial format New York Times single tabular lines and to translate the lines from the New York Times to the Associated Press format.

Recall the structure of the unformatted line from the discussion of the raw New York Times data format:

TF textual field tabular field LF

Ntabs assumes that the textual field is a collection of words each separated by one space (SP) character. It also assumes that the tabular field is a collection of strings of characters, such that each string contains only printable characters (as defined by the function prntbl), and such that the strings are separated by one or more non-printable characters, once again as defined by the function, prntbl (such characters include the space character as well as the fixed spacing characters). Finally, nstab assumes that the textual field and tabular field are separated by at least one space (SP) character followed by at least one monitor or fixed spacing character (as defined by the function tfer(a)).
These seem to be reasonable assumptions in that words in a sentence are usually separated by only one space and columns in a table are usually separated by at least one space. Also, in order to distinguish the end of the textual field from the start of the tabular field, there should be some character or collection of characters. The only characters that are known to exist in the tabular line are monitor space characters and fixed spacing characters (in varying numbers). If there is only one space between the textual and tabular fields, then who is to say what belongs in either field.

The strategy employed by ntabs is almost identical to the strategy employed by tabber. The only difference is that ntabs does not have the TFI character to distinguish between the textual and tabular fields. What it does, then, is to find the end of the textual field, add the TFI character, and then format the tabular field in exactly the same method used by tabber. The code of this procedure was derived from the code of tabber.

The entire process of ntabs will not be described. It can be picked up by reading the discussion of tabber. What will be described is the steps taken to make the New York Times format look like the Associated Press format so that the process used by tabber will work correctly.

```
ntabs(1,tg,name,const)
char 1[]; name[];
int const[];
int tg;
{
    int tp;
    int i=0;
    int a=1;
    int b;
    int j;
    int c;
    int tga=0;

    j=tg;
    name[const[7]]=TLI;
    tp=tg;

    The key to this first block of code is to notice that name[const[7]] is set equal to TLI.
```

Recall that const[7] is the index in the input buffer (name[]) of the first character of the line in
By changing this character in the input buffer to TLI, it identifies it as a single tabular line to be formatted by tabber in the second pass through the article. The only other identification needed is the TFI character that terminates the textual field.

```c
al:   if (l[a]!=SP)
    {
        if (i>MAXW)
            return (MAXW-1);
        if (l[a] == LF)
            return (MAXW-1);
        if ((decode(l[a])) == 0)
            a=a+1;
        else
            {
                a=a+1;
                i=i+1;
                tga=tga+1;
            }
        goto al;
    }
else
    {
        if (tfer(l[a+1]) == 1)
        {
            a=a+1;
            name[const[7]+a]=TFI;
            l[a]=TFI;
            goto bl;
        }
        else
            {
                a=a+1;
                i=i+1;
                tga=tga+1;
                goto al;
            }
    }
```

This block of code does the equivalent of stuffing the textual field into the target array (although the target array doesn’t really exist - it is simply a figment of the imagination of the array index i). The block reads characters, starting from the beginning of the unformatted line, and "stuff" them into the imaginary target array until the first occurrence of a space followed by another space or a fixed spacing code is found (this is the character l[a]). When this happens, the procedure assumes that this is where a TFI belongs and proceeds to change
the character l[a] into a TFI, both in the array l[] and in the input buffer, name[]. The index, a, in the array l[] corresponds to the index, const[7]+a, in the input buffer since l[0] is name[const[7]]).

The translation of the New York Times format into the Associated Press format is complete. All that remains is to format the tabular field and return the number of spaces between the end of the formatted textual field and the start of the formatted tabular field. The code is identical to the code of tabber with the exception of all references to the actual writing of characters into a target array. Since no target array is returned, no target array is needed. For a complete explanation of the formatting of the tabular field, see the description of tabber. The remaining code of ntabs is listed below for a comparison to that of tabber.

```c
bi:
    b=a;
    while (l[b] != LF)
        b=b+1;

    /* l[b] = end of the line */
    /* fill array in from back to front until reach TFI */
    j=tg;
    tp=tg;
    c=b-1;

    /* find the first printable character of a column */

    start: if (l[c] == TFI)
        return (j-tga);
    if (prntbl(l[c]) == 0)
    {
        c=c-1;
        goto start;
    }

    j=tp-1;
    c=c-1;

    if (j <= tga)
        return 0;

    /* keep filling in until find the end of the column */

    middle: if (l[c] == TFI)
        return (j-tga);
    if ((prntbl(l[c])) == 0)
    {
    
```
while (tp >= j)
    tp = tp - CW;
if (tp <= tga)
    return (j - tga);
c = c - 1;
goto start;
} else {
    c = c - 1;
j = j - 1;
if (j <= tga)
    return 0;
goto middle;
}

**NSTAB(L,TG,NAME,CONST)**

Nstab(l,tg,name, const) is the pre-formatting procedure for double (side-by-side) tabular lines. By this it is meant that it is the formatting procedure called by nconst in the first pass through a New York Times article to translate the double tabular line to the Associated Press format and to trial format the line with the width of each side of the formatted line equal to the default width. This default width is passed to nstab in the argument tg. Nstab is also passed the line to be formatted (l[]), as well as the entire input buffer (name[]) and the array of integers, const[]. Recall that the line in l[] starts at name[const[7]]. This is because the call to ngetaln that placed the line in l[] also placed the index of the start of l[] in const[7].

Recall the structure of the unformatted line from the discussion of the raw NYT data format:

```
TF TF TF L-text L-tab TF R-text R-tab LF
field field field field
```

Nstab has two problems to solve. It must format the line and return either the number of spaces between the left textual field and left tabular field or the number of spaces between the right textual field and right tabular field. It should perform this formatting with each side of the formatted line being tg characters wide. Nstab's second problem is to translate the
unformatted line to look like the AP unformatted line:

```
TLI TLI TLI L-text TFI L-tab CFS R-text TFI R-tab LF
field field field field field
```

Nstab makes the same assumptions used by ntabs concerning the textual and tabular fields. Nstab assumes that the textual fields consist of words separated by one space (SP) character. Nstab also assumes that the tabular fields consist of strings of printable characters (as defined by prntbl), separated by at least one non-printable character (as defined by prntbl). Finally, nstab assumes that the textual and tabular fields are separated by a space followed by at least one monitor or fixed spacing code (as defined by the function, tfer(a)). These assumptions are considered reasonable by the same argument given in the discussion of ntabs.

The procedure used by nstab is almost identical to that used by staber to format Associated Press double tabular lines. In fact, the code for nstab evolved from staber. The general idea is to disguise the New York Times line to look like an Associated Press line, and then use the process for formatting the line already developed for Associated Press double tabular lines. This discussion will focus on the steps taken to translate the line from the NYT to AP format and will leave the emphasis on the formatting process to the discussion of staber.

```c
nstab(1,tg,name,const)
char l[] , name[];
int const[];
int tg;
{
    int tp, tl;
    int i=0, ri=0;
    int a=3, ra;
    int b, rb;
    int j, rj;
    int c, rc;
    int cl, dl, bl;
    int kgsa, kgsb, ksa;

    j=MAXW;
    rj=MAXW;

    kgsa=MAXW-1;
    kgsb=MAXW-1;
    ksa=MAXW-1;
}
```
bl=tg;
c1= tg+3;
d1= cl+tg;
name[const[7]]= TLI;
name[const[7]+2]= TLI;

This first block of code performs the initial translation of the first three characters in the line from TF characters to TLI characters. Note that this translation occurs in the input buffer. Nstab knows the index of the start of the line within the input buffer because it was placed in const[7] by the call to ngetaln which produced I[] from the input buffer (name[]). These three characters will tell tabbern that the second pass through the article should use the formatting function, staber, to format the line.

   tp=bl;
   al: if (1[a] != SP)
      {
         if (i>MAXW)
            return ksa;
         if (1[a] == LF)
            return ksa;
         if ((decode(1[a])) == 0)
             a=a+1;
         else
             {
               a=a+1;
               i=i+1;
             }
         goto al;
      }
      else
      {
         if (tfer(1[a+1]) == 1)
            {
               a=a+1;
               name[const[7]+a]= TFI;
               1[a]= TFI;
               goto bl;
            }
         else
            {
              a=a+1;
              i=i+1;
              goto al;
            }
      }
This section of code stuffs the left textual field into the imaginary target array. Imaginary is used to describe this array since it doesn't really exist. Nstab does not return a target array. It simply translates the input buffer to the Associated Press format and returns either the number of spaces that would be between the left textual field and left tabular field or the number of spaces that would be between the right textual and right tabular fields if the line was formatted, whichever is smaller. Since no array is returned, the formatting involves the indices only.

This section of code "stuffs" characters from the unformatted line into the "imaginary" array until the first occurrence of a space followed by a monitor or fixed spacing code is reached (as defined by tfer). When this happens, nstab assumes that the left textual field has been completely stuffed and places the TFI character both in the unformatted line at the current position \( l[a] \) and in the input buffer at its respective position \( \text{name[const[7]+a]} \).

With the start of the left tabular field found, nstab must now find the end of the left textual field. This is marked by a TF symbol which must be translated to a CFS symbol so that staber can properly format the line in the second pass.

```c
bl: b=a;
    while (l[b] != TF)
    {
        if (l[b]==LF)
            return ksa;
        b=b+1;
    }
    name[const[7]+b]=CFS;
    1[b]=CFS;
```

The TF symbol has been found at position \( t[b] \). \( T[b] \) is therefore changed to a CFS and the change is also made in the input buffer. The next step is to format the left tabular field. At this point the left side of the unformatted line looks exactly like the left side of an Associated Press unformatted line. Hence, the exact same code employed by staber is used to format this field. The process will not be described here, but can be found in the discussion of staber. The code is listed below for continuity and comparative purposes.

```c
/* 1[b] = CFS */
/ * fill array in from center (bl) to front until reach TFI */
  c=b;
  j=tp;
start:  if (l[c] == TFI)
    { 
      kgsa=j-i;
      goto rside;
    }
  if (prntbl(l[c]) == 0)
    { 
      c=c-1;
      goto start;
    }
  j=tp-1;
c=c-1;
  if (j <= i)
    { 
      kgsa=1;
      goto rside;
    }
middle: if (l[c] == TFI)
    { 
      kgsa=j-i;
      goto rside;
    }
  if ((prntbl(l[c])) == 0)
    { 
      while (tp >= j)
        { 
          tp=tp-SCW;
          c=c-1;
          if (tp <= i)
            { 
              kgsa=1;
              goto rside;
            }
      goto start;
    }
else
    { 
      c=c-1;
      j=j-1;
      if (j <= i)
        { 
          kgsa=1;
          goto rside;
        }
      goto middle;
    }

The left side of the unformatted line has been completely translated from the New York
Times format to the Associated Press format. It has also gone through a complete trial formatting. Now nstab must translate and format the right side. Once again, this involves disguising the right side to look like an AP line and using the process employed by staber to format the right side of double tabular lines. The first step is to format the right textual field and in doing so, mark the end of the right textual field with a TFI character. Once this has been done, the entire line has been translated to the AP format and the rest of the formatting is a simple application of the process employed by staber.

```c
rside:   ra=b;
   ri=c1;
    rj=MAXW-1;
    tl=d1;

/* fill the array from CFS with printable material until reach TFI */

a2:      if (1[ra] ! = SP)
{   
   if (ri>MAXW)
      return ksa;
   if (1[ra] == LF)
      return ksa;
   if ((decode(1[ra])) == 0)
      ra=ra+1;
   else
   {   
      ra=ra+1;
      ri=ri+1;
   }
   goto a2;
}
  else

   {   
      if (tfer(1[ra+1]==1)
      {   
         ra=ra+1;
         1[ra]=TFI;
         goto b2;
      }
      else
   {   
         ra=ra+1;
         ri=ri+1;
         goto a2;
      }  
   }
```
This block of code stuffs the right textual field into the imaginary target array. It simply reads characters, starting from the CFS character at the end of the left tabular field and start of the right textual field, until it comes upon the first occurrence of a space (SP) character followed by a monitor or fixed spacing character code (as defined by tfer(a)). When this happens, nstab assumes the current character (l[ra]) is the end of the right textual field and writes a TFI character into l[ra] and into its respective position in the input buffer, name[const[7]+ra].

Translation to the Associated Press format is complete. The remainder of the code simply formats the right tabular field with the same process (and same code) used by staber. This process will not be described here. It's description may be found in the discussion of staber. The code is listed below for reference purposes.

```c
b2:    rb=ra;
while (l[rb] != LF)
      rb=rb+1;

/* l[rb] = end of the line */
/* fill array in from back to front until reach TFI */
rc=rb;
restart: if (l[rc] == TFI)
{    kgsb=rj-ri;
    ksa=min(kgsa,kgsb);
    return ksa;
}
if (prntbl(l[rc]) == 0)
{    rc=rc-1;
    goto restart;
}
else    
{    rj=tl-1;
    rc=rc-1;
}
if (rj <=ri)
    return ksa;
rmidle: if (l[rc] == TFI)
{    kgsb=rj-ri;
    ksa=min(kgsa,kgsb);
```
return ksa;
}
if ((prntbl(l[rcl]) == 0)
{
    while (t1 >= rj)
        t1=t1-SCW;
    rc=rc-1;
    if (t1 <= ri)
        return ksa;
    goto rstart;
}
else
{
    rc=rc-1;
    rj=rj-1;
    if (rj <= ri)
        return ksa;
    goto rmiddle;
}

**TFER(A)**

```c
Tfer(a) detects whether the character it is passed is a monitor or a fixed spacing character. It returns a one if the character is a monitor or a fixed spacing code and zero otherwise. Tfer is used only in the context of New York Times articles. It is such a simple function because the fixed spacing codes have been translated to varying numbers of blank spaces by the process that reads the story off of the wire. Tfer is used rather than a simple comparison within the calling function in case the input process for New York Times articles is ever changed.
```
3.5 General Utility Functions

A_STUFF(TARG, SRCE, STRT)

```c
A_stuff(targ, srce, strt)
char targ[], srce[];
int strt[];
{
    int i;
    int j;
    j=strt[5];
    i=0;
    while (srce[i]!='\0')
    {
        targ[j]=srce[i];
        i=i+1;
        j=j+1;
    }
    targ[j]='\0';
    strt[5]=j;
    return j;
}
```

A_stuff is used by tabbera to stuff a formatted line into the output buffer. The formatted line is stored in the character array, srce[]. The last character of the formatted line is a '\0' character. A_stuff assumes that strt[] is an integer array and that strt[5] contains the index of the next location in the output buffer (character array targ[]) to stuff.

A_stuff uses two temporary variables as pointers to specific elements in targ[] and srce[]. j is the pointer to the next space to stuff in the target array, targ[]. j is initialized to the value in strt[5]. i is the pointer to the next character in srce[] to stuff into targ[]. i is initialized to zero.

The while loop stuffs one character of srce[] into targ[] at a time, incrementing i and j each time through the loop until srce[] has been completely stuffed into targ[]. The next location in targ[] to stuff is targ[j]. This location is temporarily filled with a '\0' character and strt[5] is set equal to j, the new "next-output-buffer-position-to-fill" index. A_stuff returns j for any calling function that may want to use it (even though its value is also in strt[5]).
ACLEAR(T)

```c
aclear(t)
char t[];
{
    int i;
    for (i=0;i<=MAXW-1;i=i+1)
        t[i]=SP;
    t[MAXW]='\n';
    t[MAXW+1]='\0';
}
```

Aclear(t) is used by many of the formatting functions (tabber, staber, etc...) to initialize the target array to MAXW-1 blank spaces (SP=blank space) followed by a line feed (LF) and '\0' character. It is used by the formatting functions to clean the target array prior to formatting the line into the target array.

DECODE(A)

```c
decode(a)
char a;
{
    if (a >=' ') { 
        if (a == ENL)
            return 0;
        else if (a == EML)
            return 0;
        else if (a == UP_RAL)
            return 0;
        else
            return 1;
    } else
        return 0;
}
```

Decode is used by the formatting functions (tabber, staber) to determine if a given character should be printed in the textual field of a formatted line or if it should be considered a non-printable character and ignored. The main characters in question are the fixed spacing characters (EM space (EM), EN Space (EN), THIN Space, EM Leader (EML), EN Leader
and the upper rail character (UP_RAL). This function considers a space (SP) a printable character.

Inspection of the high speed wire service ascii table (Appendix H) shows that the ascii values of all of the printable characters come after the space character. In this set, the only characters that should not be printed are the EM Leader (EML=ascii 42), EN Leader (ENL=ascii 35), and the upper rail (UP_RAL=ascii 94) characters. Decode returns 1 if the character should be printed and 0 otherwise.

PRNTBL(A)

```c
prntbl(a)
char a;
{
    if (a > SP)
    {
        if (a == ENL)
            return 0;
        else if (a == EML)
            return 0;
        else if (a == UP_RAL)
            return 0;
        else
            return 1;
    }
    else
        return 0;
}
```

Prntbl(a) is used by the formatting functions (staber, tabber) to determine if a given character should be printed in the tabular field of a formatted line or if it should be considered a non-printable character and ignored. As with the function decode(a), the main characters in question are the fixed spacing characters (EM space, EN space, THIN space, EM leader, EN leader, Upper Rail, etc...). In uses of this function, regular spaces (SP) are not considered printable characters.

Inspection of the high speed wire service ascii table shows that all of the printable characters occur after the space character (SP). In this set, the only characters that should
not be printed are the EM leader (EML), EN leader (ENL) and the upper rail (UP_RAL) characters. Prnbl(a) returns 1 if its argument, a, should be printed and 0 otherwise.

**MIN(A,B)**

```c
min(a,b)
int a,b;
{
    if (a <= b)
        return a;
    else
        return b;
}
```

Min(a,b) simply returns either a or b, whichever is less. It is used by staber in determining which value to return, the space between the left textual field and left tabular field or the space between the right textual field and right tabular field.
CONCLUSION

This project started out with the goal of translating New York Times and Associated Press wire service articles from their raw format, which includes the article plus embedded typesetting codes, into readable articles, formatted to appear on the screen as they would appear in a newspaper. This goal was further defined, in the discussion of the New York Times and Associated Press raw data formats, into five problem areas. Four of the five problems dealt with specific types of data lines in the raw article. The final problem dealt with the entire article.

The first problem was to format any textual lines that may exist in the raw article. This problem was solved before the project was started in that textual lines do not need any special formatting. They are simply passed from the input buffer to the output buffer.

The second problem dealt with quad material. Also referred to as quad left, quad center, and quad right lines, this type of data was marked by a special quad identification character at the end of the line. The quad material was correctly formatted by the procedures quadl, quadc, and quadr such that formatted quad left lines had their first character at the left margin of the screen, formatted quad center lines were centered on the screen, and formatted quad right lines had their last character at the right margin of the screen.

The third problem was concerned with single tabular data lines. This type of data was marked by a single tabular data line identification character at the start of the line. The line begins with a field of textual data and ends with a field of tabular data. Such lines were to be formatted such that the columns in the tabular fields of all of single tabular lines in the article line up correctly.

Single tabular lines were formatted by the procedure, tabber. This procedure offered a pretty good solution to the single tabular data problem with some limitations. The first limitation is that the process assumes that each tabular line in the article contains the same number of tabular columns. Should a given line not have the same number of tabular columns
as all of the rest of the lines, the columns in the line will still line up with the columns in the other lines. However, the columns may not line up in their correct relative columns in the article.

The second limitation of single tabular data lines is that columns that employ fractions may be treated as two separate columns. This is due to the fact that there is probably a space between the whole number and the fraction in a piece of data such as 1 1/2.

The third limitation of the single tabular data line formatting process is that the process generally assumes that the relative columns in the article are of the same width. By this it is meant that the third column in the first single tabular line is the same width as the third column in the nth single tabular line. The choice of the column width parameter, CW, can be made larger to reduce the effects of this problem, however, this will increase the number of blank spaces between columns and increase the effect of the next limitation.

The fourth and final limitation that will be mentioned is that if there are many columns in the tabular field of the single tabular line, it is possible for some of the raw data to be ignored because there simply isn’t enough room in the tabular field of the line to place all of the columns. This problem can be reduced by decreasing the column width parameter, however, this increases the effect of the third limitation.

The fourth problem dealt with double (or side-by-side) tabular data lines. These lines are identified by three tabular line identification characters at the start of the line. Double tabular lines look like two single tabular lines placed side-by-side. The left side of the line has a textual field followed by a tabular field. The right side of the line has the same structure. These lines were to have been formatted such that the tabular columns in all the left tabular fields of all of the lines line up correctly and such that the tabular columns in the right side do the same.

The problem of the double tabular lines was solved by the formatting procedure, staber. However, this process has the same limitations that were outlined for single tabular lines.

The final problem that this project attempts to solve is to remove any stray typesetting
codes that may cause garbage to appear on the screen. The characters that seem to cause the most trouble are the EN leader (ascii 35, "#" character), the EM leader (ascii 42, "*" character), the upper rail (ascii 94, "^" character), and the lower rail (ascii 64, "@" character). These characters are translated to spaces by the procedures getaln (for Associated press articles) and ngetaln (for New York Times articles).

As a first cut to the problem that was outlined, the formatting processes tabbera (for Associated Presss articles) and tabbern (for New York Times articles) offer, with certain limitations, a fairly good solution to the problem. In retrospect, however, the processes are not as efficient as they perhaps could have been. The main recommendation that can be made considering a revision of this project is to make better use of the first pass through the article in both tabbera and tabbern.

With the exception of tabber and staber, the formatting procedures employed by tabbera and tabbern essentially copy characters from the source (unformatted) line to the target (formatted) line. The only difficult formatting tasks are performed by staber and tabber. Within these procedures, the difficult task is formatting the tabular field(s) of the line.

The process of formatting the tabular field(s) in a single or double tabular line is the same in both passes through the article. This implies that it really only needs to be done once.

The first pass through the article could be restructured to format every line into an intermediate buffer. The second pass would then essentially copy each textual and quad line (quad lines, when formatted, look like textual lines) into the output buffer. The second pass would format tabular lines by simply sliding the formatted tabular fields within the lines instead of actually performing the complicated operation of finding and formatting each column that was outlined in the discussion of tabber, staber, ntabs, and nstab. This change reduces the second pass to essentially copying characters instead of performing comparisons, character decodings, etc..., and would probably increase the performance of the process.
APPENDIX A
EXPLANATION OF FIXED SPACING CODES

The typesetting and graphic arts industries deal with, among other things, the printed word. Since these industries frequently deal with things that fit on a single sheet of paper, the basic units of measurement, the inch, foot, yard, mile, meter, kilometer, etc... are simply too large to be convenient units of measurement. The industries have therefore developed their own units of measurement. The basic units of measurement are the point and the pica. There are twelve points in one pica and six picas in one inch.

The discussions of the raw data structure (Chapter 1) referred to variable width fixed spacing codes embedded in the tabular lines. The fixed spacing characters exist because in typeset text, not all characters are the same width. For instance, the characters m and w are wider than the character i. To right and left justify the margins in a typeset article, spaces of varying widths are needed to compensate for the varying numbers of variable width characters on a line.

Typeset text is classified by its point size. This refers to the size of the widest letter, m or w, on the printed page. The point size of the type determines the size of the fixed spacing characters.

There are six basic fixed spacing characters, only three of which are relevant to this project. The three that are relevant are the EM or mutton, the EN or nut, and the THIN. The other three are the THICK, the MIDDLE or MID, and the HAIR.

An EM is a square space of the point size of the type. For example, an EM in sixty point type is a square of 60 points by 60 points. All of the other fixed spaces are defined in terms of the EM. An EN is 1/2 of an EM. A THICK is 1/3 of an EM. A MID is 1/4 of an EM. A THIN is 1/5 of an EM. Finally, a HAIR is 1/6 of an EM.

NOTE:

The source of this information is the book, Designing with Type: A Basic Course in Typography, by James Craig.
APPENDIX B

Single Tabular Data Examples

The following are examples of single tabular data lines in a raw Associated press article.

The * represents the TLI (Tab Line Indicator) character, and the & represents the TFI (Tab Field Indicator) character. Finally, the # represents the LF (line feed) character.

```
* Larry Bird f & 12 23 43 .785 #
* Robert Parish c & 11 25 12 .345 #
```

```
* Boston & cldy 75 43 .006 #
* New York & rain 55 45 .456 #
* Washington & sunny 85 65 #
* Orlando & sunny 95 70 .010 #
* Los Angeles & smoggy 95 80 .016 #
* New Orleans & rain 90 80 .585 #
```

The following are the same single tabular data lines in the raw New York Times data format. The % character represents the TF (Tape Feed) character. The # once again represents the LF (Line Feed) character.

```
% Larry Bird f 12 23 43 .785 #
% Robert Parish c 11 25 12 .345 #
```

```
% Boston cldy 75 43 .006 #
% New York City rain 65 45 .456 #
% Washington sunny 85 65 #
% Orlando sunny 95 70 .010 #
% Los Angeles smoggy 95 80 .016 #
% New Orleans rain 90 80 .585 #
```
APPENDIX C
Double Tabular Data Examples

The following are examples of double tabular data lines in a raw Associated Press article. The * represents the TLI (Tab Line Indicator) character, the & represents the TFI (Tab Field Indicator) character, the @ represents the CFS (Center Field Separator) character, and the # represents the LF (Line Feed) character.

```
*** Smith   & 12 34 43 .221 @ Jones    & 11 32 54 .333 #
*** Adams   & 11 21 6 .111 @ Simons   & 10 12 15 .985 #
*** Heef    & 85 12 37 .431 @ Flynn    & 12 34 55 .454 #
*** Muncey  & 10 11 12 .121 @ Gladys   & 13 23 45 .886 #
*** Sides   & 11 22 33 .444 @ Nickie    & 12 12 12 .121 #
```

The following are the same examples of double tabular data only now they are in the raw New York Times data format. The % represents the TF (Tape Feed) symbol and the # represents the LF (Line Feed) character.

```
%%% Smith    12 34 43 .221 % Jones    11 32 54 .333 #
%%% Adams   11 21 6 .111 % Simons   10 12 15 .986 #
%%% Heef     85 12 37 .431 % Flynn   12 34 55 .454 #
%%% Muncey  10 11 12 .121 % Gladys  13 23 45 .886 #
%%% Sides   11 22 33 .444 % Nickie   12 12 12 .121 #
```
APPENDIX D

Sample Articles

The pages that follow contain sample articles in both their raw and their formatted form. There are many garbage characters in the raw articles. These are the typesetting codes that are visible. There are also typesetting codes that are control characters, but these codes are not visible. The codes that are visible can be decoded as follows:

- `<` quad left identification code
- `=` quad center identification code
- `>` quad right identification code

- `^` upper rail
- `@` lower rail
- `#` EN leader
- `*` EM leader
**College Football Standings**

*By The Associated Press*

### Atlantic Coast Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maryland</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>192</td>
<td>92</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>291</td>
</tr>
<tr>
<td>G. Tech</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>136</td>
<td>55</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>235</td>
<td>118</td>
</tr>
<tr>
<td>Virginia</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>172</td>
<td>150</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>262</td>
<td>217</td>
</tr>
<tr>
<td>Clemson</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>167</td>
<td>122</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>231</td>
<td>202</td>
</tr>
<tr>
<td>N. Carolina</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>129</td>
<td>154</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>224</td>
<td>223</td>
</tr>
<tr>
<td>N. Caro. St.</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>121</td>
<td>189</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>186</td>
<td>305</td>
</tr>
<tr>
<td>Duke</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>94</td>
<td>174</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>193</td>
<td>252</td>
</tr>
<tr>
<td>Wk. Forest</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>99</td>
<td>174</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>212</td>
<td>249</td>
</tr>
</tbody>
</table>

### Big Eight Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oklahoma</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>270</td>
<td>39</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>346</td>
<td>93</td>
</tr>
<tr>
<td>Nebraska</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>232</td>
<td>87</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>398</td>
<td>136</td>
</tr>
<tr>
<td>Colorado</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>140</td>
<td>78</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>211</td>
<td>154</td>
</tr>
<tr>
<td>Okla. St.</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>121</td>
<td>105</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>232</td>
<td>154</td>
</tr>
<tr>
<td>Iowa St.</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>105</td>
<td>221</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>155</td>
<td>318</td>
</tr>
<tr>
<td>Kansas</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>118</td>
<td>184</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>294</td>
<td>281</td>
</tr>
<tr>
<td>Missouri</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>117</td>
<td>219</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>206</td>
<td>342</td>
</tr>
<tr>
<td>Kansas St.</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>53</td>
<td>223</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>101</td>
<td>295</td>
</tr>
</tbody>
</table>

### Big Ten Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iowa</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>249</td>
<td>119</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>412</td>
<td>142</td>
</tr>
<tr>
<td>Michigan</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>241</td>
<td>60</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>315</td>
<td>75</td>
</tr>
<tr>
<td>Illinois</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>212</td>
<td>206</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>275</td>
<td>303</td>
</tr>
<tr>
<td>Ohio St.</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>221</td>
<td>153</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>315</td>
<td>205</td>
</tr>
<tr>
<td>Mich. St.</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>215</td>
<td>169</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>244</td>
<td>202</td>
</tr>
<tr>
<td>Minnesota</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>176</td>
<td>183</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>273</td>
<td>227</td>
</tr>
<tr>
<td>Purdue</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>185</td>
<td>240</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>287</td>
<td>306</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>126</td>
<td>206</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>231</td>
<td>263</td>
</tr>
<tr>
<td>Indiana</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>136</td>
<td>260</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>251</td>
<td>340</td>
</tr>
<tr>
<td>N. Caro. St.</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>88</td>
<td>253</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>170</td>
<td>332</td>
</tr>
</tbody>
</table>

*MORE*
College Football Standings
By The Associated Press

Atlantic Coast Conference

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maryland</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td></td>
<td>6</td>
<td>0</td>
<td>192</td>
<td>92</td>
</tr>
<tr>
<td>Ga. Tech</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td></td>
<td>8</td>
<td>2</td>
<td>136</td>
<td>55</td>
</tr>
<tr>
<td>Virginia</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
<td>8</td>
<td>2</td>
<td>172</td>
<td>150</td>
</tr>
<tr>
<td>Clemson</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
<td>6</td>
<td>5</td>
<td>167</td>
<td>122</td>
</tr>
<tr>
<td>N.Carolina</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td></td>
<td>6</td>
<td>5</td>
<td>129</td>
<td>154</td>
</tr>
<tr>
<td>N.Carol. St.</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td></td>
<td>3</td>
<td>8</td>
<td>121</td>
<td>189</td>
</tr>
<tr>
<td>Duke</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td></td>
<td>4</td>
<td>7</td>
<td>94</td>
<td>174</td>
</tr>
<tr>
<td>Wk.Forest</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td></td>
<td>4</td>
<td>7</td>
<td>99</td>
<td>174</td>
</tr>
</tbody>
</table>

Big Eight Conference

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oklahoma</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td></td>
<td>7</td>
<td>0</td>
<td>270</td>
<td>39</td>
</tr>
<tr>
<td>Nebraska</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td></td>
<td>8</td>
<td>2</td>
<td>232</td>
<td>87</td>
</tr>
<tr>
<td>Colorado</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
<td>7</td>
<td>4</td>
<td>140</td>
<td>78</td>
</tr>
<tr>
<td>Okla.St.</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
<td>8</td>
<td>3</td>
<td>121</td>
<td>105</td>
</tr>
<tr>
<td>Iowa St.</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td></td>
<td>5</td>
<td>6</td>
<td>105</td>
<td>221</td>
</tr>
<tr>
<td>Kansas</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td></td>
<td>6</td>
<td>6</td>
<td>118</td>
<td>184</td>
</tr>
<tr>
<td>Missouri</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td></td>
<td>1</td>
<td>10</td>
<td>117</td>
<td>219</td>
</tr>
<tr>
<td>Kansas St.</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td></td>
<td>1</td>
<td>10</td>
<td>53</td>
<td>223</td>
</tr>
</tbody>
</table>

Big Ten Conference

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iowa</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
<td>7</td>
<td>1</td>
<td>249</td>
<td>119</td>
</tr>
<tr>
<td>Michigan</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td></td>
<td>9</td>
<td>1</td>
<td>241</td>
<td>60</td>
</tr>
<tr>
<td>Illinois</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td></td>
<td>6</td>
<td>4</td>
<td>212</td>
<td>206</td>
</tr>
<tr>
<td>Ohio St.</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td></td>
<td>8</td>
<td>3</td>
<td>221</td>
<td>153</td>
</tr>
<tr>
<td>Mich. St.</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td></td>
<td>7</td>
<td>4</td>
<td>215</td>
<td>169</td>
</tr>
<tr>
<td>Minnesota</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td></td>
<td>6</td>
<td>5</td>
<td>176</td>
<td>183</td>
</tr>
<tr>
<td>Purdue</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td></td>
<td>5</td>
<td>6</td>
<td>185</td>
<td>240</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td></td>
<td>5</td>
<td>6</td>
<td>126</td>
<td>206</td>
</tr>
<tr>
<td>Indiana</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td></td>
<td>4</td>
<td>7</td>
<td>136</td>
<td>260</td>
</tr>
<tr>
<td>Nthwstrn MORE</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>88</td>
<td>253</td>
</tr>
</tbody>
</table>

All conference totals:

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

All conference games:

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Article 2: unformatted version

type: Associated Press national sports news
priority: regular
date: 12-08-85 2220EST
category: Sports
subject: FBC--Standings, 7 Takes, 0070
text:

College Football Standings

By The Associated Press

Atlantic Coast Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maryland</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>192</td>
<td>92</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>291</td>
<td>174</td>
</tr>
<tr>
<td>Ga. Tech</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>136</td>
<td>55</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>235</td>
<td>118</td>
</tr>
<tr>
<td>Virginia</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>172</td>
<td>150</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>262</td>
<td>217</td>
</tr>
<tr>
<td>Clemson</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>167</td>
<td>122</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>231</td>
<td>202</td>
</tr>
<tr>
<td>N.Carolina</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>129</td>
<td>154</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>224</td>
<td>223</td>
</tr>
<tr>
<td>N.Car.St.</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>121</td>
<td>189</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>186</td>
<td>305</td>
</tr>
<tr>
<td>Duke</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>94</td>
<td>174</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>193</td>
<td>252</td>
</tr>
<tr>
<td>Wk.Forest</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>99</td>
<td>174</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>212</td>
<td>249</td>
</tr>
</tbody>
</table>

Big Eight Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oklahoma</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>270</td>
<td>39</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>346</td>
<td>93</td>
</tr>
<tr>
<td>Nebraska</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>232</td>
<td>87</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>398</td>
<td>136</td>
</tr>
<tr>
<td>Colorado</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>140</td>
<td>78</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>211</td>
<td>154</td>
</tr>
<tr>
<td>Okla.St.</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>121</td>
<td>105</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>232</td>
<td>154</td>
</tr>
<tr>
<td>Iowa St.</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>105</td>
<td>221</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>155</td>
<td>318</td>
</tr>
<tr>
<td>Kansas</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>118</td>
<td>184</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>294</td>
<td>281</td>
</tr>
<tr>
<td>Missouri</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>117</td>
<td>219</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>206</td>
<td>342</td>
</tr>
<tr>
<td>Kansas St.</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>53</td>
<td>223</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>101</td>
<td>295</td>
</tr>
</tbody>
</table>

Big Ten Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T</th>
<th>Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iowa</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>249</td>
<td>119</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>412</td>
<td>142</td>
</tr>
<tr>
<td>Michigan</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>241</td>
<td>60</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>315</td>
<td>75</td>
</tr>
<tr>
<td>Illinois</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>212</td>
<td>206</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>275</td>
<td>303</td>
</tr>
<tr>
<td>Ohio St.</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>221</td>
<td>153</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>315</td>
<td>205</td>
</tr>
<tr>
<td>Mich. St.</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>215</td>
<td>169</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>244</td>
<td>202</td>
</tr>
<tr>
<td>Minnesota</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>176</td>
<td>183</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>273</td>
<td>227</td>
</tr>
<tr>
<td>Purdue</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>185</td>
<td>240</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>287</td>
<td>306</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>126</td>
<td>206</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>231</td>
<td>263</td>
</tr>
<tr>
<td>Indiana</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>136</td>
<td>260</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>251</td>
<td>340</td>
</tr>
<tr>
<td>Nrthwstrn</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>88</td>
<td>253</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>170</td>
<td>332</td>
</tr>
</tbody>
</table>

MORE
College Football Standings
By The Associated Press

Atlantic Coast Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maryland</td>
<td>6</td>
<td>0</td>
<td>0 192</td>
<td>92</td>
<td>8</td>
<td>3</td>
<td>0 291</td>
<td>174</td>
</tr>
<tr>
<td>Ga. Tech</td>
<td>5</td>
<td>1</td>
<td>0 136</td>
<td>55</td>
<td>8</td>
<td>2</td>
<td>1 235</td>
<td>118</td>
</tr>
<tr>
<td>Virginia</td>
<td>4</td>
<td>3</td>
<td>0 172</td>
<td>150</td>
<td>6</td>
<td>5</td>
<td>0 262</td>
<td>217</td>
</tr>
<tr>
<td>Clemson</td>
<td>4</td>
<td>3</td>
<td>0 167</td>
<td>122</td>
<td>6</td>
<td>5</td>
<td>0 231</td>
<td>202</td>
</tr>
<tr>
<td>N.Carolina</td>
<td>3</td>
<td>4</td>
<td>0 129</td>
<td>154</td>
<td>5</td>
<td>6</td>
<td>0 224</td>
<td>223</td>
</tr>
<tr>
<td>N.Caro.St.</td>
<td>2</td>
<td>5</td>
<td>0 121</td>
<td>189</td>
<td>3</td>
<td>8</td>
<td>0 186</td>
<td>305</td>
</tr>
<tr>
<td>Duke</td>
<td>2</td>
<td>5</td>
<td>0 94</td>
<td>174</td>
<td>4</td>
<td>7</td>
<td>0 193</td>
<td>252</td>
</tr>
<tr>
<td>Wk.Forest</td>
<td>1</td>
<td>6</td>
<td>0 99</td>
<td>174</td>
<td>4</td>
<td>7</td>
<td>0 212</td>
<td>249</td>
</tr>
</tbody>
</table>

Big Eight Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oklahoma</td>
<td>7</td>
<td>0</td>
<td>0 270</td>
<td>39</td>
<td>10</td>
<td>1</td>
<td>0 346</td>
<td>93</td>
</tr>
<tr>
<td>Nebraska</td>
<td>6</td>
<td>1</td>
<td>0 232</td>
<td>87</td>
<td>9</td>
<td>2</td>
<td>0 398</td>
<td>136</td>
</tr>
<tr>
<td>Colorado</td>
<td>4</td>
<td>3</td>
<td>0 140</td>
<td>78</td>
<td>7</td>
<td>4</td>
<td>0 211</td>
<td>154</td>
</tr>
<tr>
<td>Okla.St.</td>
<td>4</td>
<td>3</td>
<td>0 121</td>
<td>105</td>
<td>8</td>
<td>3</td>
<td>0 232</td>
<td>154</td>
</tr>
<tr>
<td>Iowa St.</td>
<td>3</td>
<td>4</td>
<td>0 105</td>
<td>221</td>
<td>5</td>
<td>6</td>
<td>0 155</td>
<td>318</td>
</tr>
<tr>
<td>Kansas</td>
<td>2</td>
<td>5</td>
<td>0 118</td>
<td>184</td>
<td>6</td>
<td>6</td>
<td>0 294</td>
<td>281</td>
</tr>
<tr>
<td>Missouri</td>
<td>1</td>
<td>6</td>
<td>0 117</td>
<td>219</td>
<td>1</td>
<td>10</td>
<td>0 206</td>
<td>342</td>
</tr>
<tr>
<td>Kansas St.</td>
<td>1</td>
<td>6</td>
<td>0 53</td>
<td>223</td>
<td>1</td>
<td>10</td>
<td>0 101</td>
<td>295</td>
</tr>
</tbody>
</table>

Big Ten Conference

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
<th>W</th>
<th>L</th>
<th>T Pts</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iowa</td>
<td>7</td>
<td>1</td>
<td>0 249</td>
<td>119</td>
<td>10</td>
<td>1</td>
<td>0 412</td>
<td>142</td>
</tr>
<tr>
<td>Michigan</td>
<td>6</td>
<td>1</td>
<td>1 241</td>
<td>60</td>
<td>9</td>
<td>1</td>
<td>1 315</td>
<td>75</td>
</tr>
<tr>
<td>Illinois</td>
<td>5</td>
<td>2</td>
<td>1 212</td>
<td>206</td>
<td>6</td>
<td>4</td>
<td>1 275</td>
<td>303</td>
</tr>
<tr>
<td>Ohio St.</td>
<td>5</td>
<td>3</td>
<td>0 221</td>
<td>153</td>
<td>8</td>
<td>3</td>
<td>0 315</td>
<td>205</td>
</tr>
<tr>
<td>Mich. St.</td>
<td>5</td>
<td>3</td>
<td>0 215</td>
<td>169</td>
<td>7</td>
<td>4</td>
<td>0 244</td>
<td>202</td>
</tr>
<tr>
<td>Minnesota</td>
<td>4</td>
<td>4</td>
<td>0 176</td>
<td>183</td>
<td>6</td>
<td>5</td>
<td>0 273</td>
<td>227</td>
</tr>
<tr>
<td>Purdue</td>
<td>3</td>
<td>5</td>
<td>0 185</td>
<td>240</td>
<td>5</td>
<td>6</td>
<td>0 287</td>
<td>306</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>2</td>
<td>6</td>
<td>0 126</td>
<td>206</td>
<td>5</td>
<td>6</td>
<td>0 231</td>
<td>263</td>
</tr>
<tr>
<td>Indiana</td>
<td>1</td>
<td>7</td>
<td>0 136</td>
<td>260</td>
<td>4</td>
<td>7</td>
<td>0 251</td>
<td>340</td>
</tr>
<tr>
<td>Nthwstrn</td>
<td>1</td>
<td>7</td>
<td>0 88</td>
<td>253</td>
<td>3</td>
<td>8</td>
<td>0 170</td>
<td>332</td>
</tr>
</tbody>
</table>

MORE
### Article 3: unformatted version

- **type:** Associated Press state and regional news
- **priority:** regular
- **date:** 12-09-85 0856EST
- **category:** Racing results and entries
- **subject:** Entries Latonia Race Course, ltet ktteqevyyeq
- **author:** The Associated Press

**text:**

#### 1st_3,200, cl, mdn 3YO up, 6 1/2f.<
- Maids Lady 122 Bekina 122
- Flmng Mnrc 120 Marys Duflbg 120
- Hardys Hony 122 Sugar Bugger 122
- Diane K. 115 Syncptd Dnrc 122
- Little Dunmr 122 Flaming Star 122
- Char KelGal 113 Grand Commd 120
- Keki Angel 122 Mrs. Honey 113
- Miss FoolinM 120 Just Impossbl 120

#### 2nd_3,500, cl, 3YO up, 11-16mi.<
- Chatham Hall 121 Nates Toy 115
- a-Ryl Cntrctr 118 R. Amber 115
- Regal Richrd 110 With Dispatch 113
- Japson 121 a-Vaders Frc 115
- Hot Salsa 114 Garish Gren 115
- Young Naskr 115 Round NTan 115
- b-Hill Hopper 111 Tucks Tout 121
- b-Canard 115 Impulso 115
- a-Coupled. b-Coupled.

#### 3rd_5,500, mdn 3YO up, lmi.<
- Bold Sparky 121 Duris 119
- Brians Astra 112 Duke OfWkfld 119
- Never BKnwn 119 Tillitfreezsvr 119
- Proud Sailor 114 Major Kidd 119
- Rule TheValy 112 Robins Flame 112
- Jun Lee 114 Tireless 119

#### 4th_4,900, cl, 3YO up, 11-16mi.<
- Lady Longsht 121 a-Prncs Mntg 115
- Call MeNita 115 Its Coalie 119
- a-Always BTr 113 Trojan Godds 115
- West Moon 115 Mchl AndDwn 113
- Rose Nix 115 Dutchs LnStr 113
- a-Coupled.
### Article 3: formatted version

**type:** Associated Press state and regional news  
**priority:** regular  
**date:** 12-09-85 0856EST  
**category:** Racing results and entries  
**subject:** Entries Latonia Race Course  
**author:** The Associated Press  
**text:**

**1st_3,200, cl, mdn 3YO up, 6 1/2f.**

<table>
<thead>
<tr>
<th>Horse</th>
<th>Weight</th>
<th>Odds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maids Lady</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>Flmng Mrch</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>Hardys Hony</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>Diane K.</td>
<td>115</td>
<td>122</td>
</tr>
<tr>
<td>Little Dunmr</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>Char KelGal</td>
<td>113</td>
<td>122</td>
</tr>
<tr>
<td>Keki Angel</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>Miss FoolinM</td>
<td>120</td>
<td>122</td>
</tr>
</tbody>
</table>

**2nd_3,500, cl, 3YO up, 11-16mi.**

<table>
<thead>
<tr>
<th>Horse</th>
<th>Weight</th>
<th>Odds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chatham Hall</td>
<td>121</td>
<td>115</td>
</tr>
<tr>
<td>a-Ryl Cntrctr</td>
<td>118</td>
<td>115</td>
</tr>
<tr>
<td>Regal Richrd</td>
<td>110</td>
<td>113</td>
</tr>
<tr>
<td>Japson</td>
<td>121</td>
<td>115</td>
</tr>
<tr>
<td>Hot Salsa</td>
<td>114</td>
<td>115</td>
</tr>
<tr>
<td>Young Naskr</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>b-Hill Hopper</td>
<td>111</td>
<td>121</td>
</tr>
<tr>
<td>b-Canard</td>
<td>115</td>
<td>115</td>
</tr>
</tbody>
</table>

a-Coupled. b-Coupled.

**3rd_5,500, mdn 3YO up, 1mi.**

<table>
<thead>
<tr>
<th>Horse</th>
<th>Weight</th>
<th>Odds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold Sparky</td>
<td>121</td>
<td>119</td>
</tr>
<tr>
<td>Brians Astra</td>
<td>112</td>
<td>119</td>
</tr>
<tr>
<td>Never BKwn</td>
<td>119</td>
<td>119</td>
</tr>
<tr>
<td>Proud Sailor</td>
<td>114</td>
<td>119</td>
</tr>
<tr>
<td>Rule TheValy</td>
<td>112</td>
<td>119</td>
</tr>
<tr>
<td>Jun Lee</td>
<td>114</td>
<td>119</td>
</tr>
</tbody>
</table>

**4th_4,900, cl, 3YO up, 11-16mi.**

<table>
<thead>
<tr>
<th>Horse</th>
<th>Weight</th>
<th>Odds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lady Longsht</td>
<td>121</td>
<td>115</td>
</tr>
<tr>
<td>Call MeNita</td>
<td>115</td>
<td>119</td>
</tr>
<tr>
<td>a-Always BTr</td>
<td>113</td>
<td>115</td>
</tr>
<tr>
<td>West Moon</td>
<td>115</td>
<td>113</td>
</tr>
<tr>
<td>Rose Nix</td>
<td>115</td>
<td>113</td>
</tr>
</tbody>
</table>

a-Coupled.
Article 4: unformatted version

type: Associated Press state and regional news
priority: urgent
date: 12-06-85 1805EST
category: Racing results and entries
subject: Results Fairgrounds-5-Add, fg5 vssvzbywf

text:
5th_8,100, cl, 3YO up, 6f.<
Heart And Soul (GMelancon)  
11.10 6.00 4.80
Twin Sister (Faul)  
9.00 6.00
Chuto Way (Poyadou)  
7.80
Off 4:54. Time 1:13. $3 Exacta (1-10) paid $49.20.

Article 4: formatted version

type: Associated Press state and regional news
priority: urgent
date: 12-06-85 1805EST
category: Racing results and entries
subject: Results Fairgrounds-5-Add, fg5 vssvzbywf

text:
5th_8,100, cl, 3YO up, 6f.
Heart And Soul (GMelancon)  
11.10 6.00 4.80
Twin Sister (Faul)  
9.00 6.00
Chuto Way (Poyadou)  
7.80
Off 4:54. Time 1:13. $3 Exacta (1-10) paid $49.20.
Article 5: unformatted version

type: Associated Press state and regional news
priority: urgent
date: 12-06-85 2116EST
category: Racing results and entries
subject: Results Fairgrounds-11-Add, fg1lvssvzbvztzvt
text:

†11th_6,800, cl, 3YO up, 11-16mi.<
Mickey Lynne (Ardoin) 9.00 6.90 4.20
Tyro Red (Broussard) 8.10 4.80
Iva's Choice (Romero) 5.10

Off 7:45. Time 1:48. Scratched_Lady Clontarf, Change The Power, Saber Shin, I Merrily, Popsalot, Chatham High. $3 Exacta (2-10) paid $45.90. $5 Quinella (2-10 & 10-2) paid $54.50. $3 Trifecta (2-10-11) paid $245.70. Attendance 7,174. Handle $1,091,167.

Article 5: formatted version

type: Associated Press state and regional news
priority: urgent
date: 12-06-85 2116EST
category: Racing results and entries
subject: Results Fairgrounds-11-Add, fg1lvssvzbvztzvt
text:

11th_6,800, cl, 3YO up, 11-16mi.
Mickey Lynne (Ardoin) 9.00 6.90 4.20
Tyro Red (Broussard) 8.10 4.80
Iva's Choice (Romero) 5.10

Off 7:45. Time 1:48. Scratched_Lady Clontarf, Change The Power, Saber Shin, I Merrily, Popsalot, Chatham High. $3 Exacta (2-10) paid $45.90. $5 Quinella (2-10 & 10-2) paid $54.50. $3 Trifecta (2-10-11) paid $245.70. Attendance 7,174. Handle $1,091,167.
This is supposed to be the longest line of text in this file of test data.

This is a quad right test line.

This is a quad center test line.

This is a quad left test line.

This is supposed to be the longest line of text in this file of test data.

This is a quad right test line.

This is a quad center test line.

This is a quad left test line.
Earcap: Partly sunny.

Monday, Monday night and Tuesday, partly cloudy, high both days around 40, low Monday night 20 to 25. Wind northwest five to 10 mph Monday. <

Weather Story

High pressure will follow a weak cold front that passed through southern New england Sunday night.

Weather summary for Sunday:

Temperature: High of 45 at 2:00 p.m., low of 29 at 1:00 a.m.
Precipitation: None.
Relative humidity: 59 percent at 7 p.m.
Heavy snow covered parts of Wyoming, Utah and Colorado early today, with more than 3 feet of fresh snow reported in at least one town. A winter storm warning for heavy snow was issued across the northern mountains of Utah, where Snowbird already had 37 inches by Sunday evening. Alta had 20 inches and downtown Salt Lake City had 12 to 16 inches.

A heavy snow warning also was posted over central and southern Wyoming, with 16 inches of new snow already heaped on Lander, and 10 inches near Riverton and Shosone. The southwest mountains of Colorado were expecting heavy snow tonight and the the Nebraska Panhandle and northwest Kansas were under winter storm watches for possibly heavy snow.

Travel advisories for snow and icy roads stretched across southeast Idaho, northern Wyoming, the Nebraska Panhandle, Nevada's Lake Tahoe area and California's Sierra Nevada mountains and Kings Canyon.

Snow also fell from northwest Iowa across Nebraska, and over western South Dakota, southeast Montana and eastern Idaho.

Advisories for gusty winds were issued over the southern mountains and deserts of California.

Snowfall during the six-hour period ending at 1 a.m. EST included 6 inches at Lander and 5 inches at Casper Wyo., 4 inches at Fort Collins, Colo., and 3 inches at Alliance, Ogallala and Scottsbluff, Neb.

Farther east, snow reached from northern lower Michigan over upper Michigan and northeast Minnesota. And freezing rain was scattered over eastern and central New York.

And in Florida, a late-season tropical depression in the Gulf of Mexico was called more of an oddity than a threat by forecasters at the National Hurricane Centers in Miami.

Sunday night, the depression was about 100 miles northwest of Balboa, Panama, with maximum winds of about 35 mph. If the depression strengthens to maximum sustained winds of 39 mph, it will be named Tropical Storm Larry.

Today's forecast called for snow from western South Dakota and Nebraska across northwest Kansas, Wyoming, Colorado, Utah, northern Arizona, eastern Nevada and eastern Idaho, turning to rain over Oklahoma.

High temperatures should be in the teens and 20s over northern New England, upper Michigan, Minnesota, the northern half of the Plains and the northern and central Rockies. Highs in the 30s and
40s should stretch from the rest of New England down through the Mid-Atlantic States, much of the Great Lakes, the Ohio Valley, Kansas, the southern Rockies and the northern Pacific Coast. Highs should range in the 70s over central and southern Florida the central Gulf Coast, reaching into the 80s over southern Texas.

Temperatures around the nation at 2 a.m. EST ranged from 2 degrees at Havre, Mont., to 74 at West Palm Beach, Fla.

Other reports:

East: Atlanta 49 cloudy; Boston 36 fair; Charleston, S.C. 39 fair; Cincinnati 26 foggy; Cleveland 34 cloudy; Detroit 34 foggy; Miami 65 fair; New York 39 fair; Philadelphia 39 fair; Pittsburgh 29 fair; Portland 30 fair; Washington 35 fair.

Central: Bismarck 27 cloudy; Chicago 26 foggy; Dallas-Fort Worth 61 fair; Denver 28 snow; Des Moines 16 foggy; Indianapolis 29 foggy; Kansas City 28 foggy; Minneapolis-St. Paul 23 foggy; Nashville 44 partly cloudy; New Orleans 53 fair; St. Louis 33 foggy.

West: Albuquerque 39 fair; Anchorage 31 cloudy; Las Vegas 41 fair; Los Angeles 54 fair; Phoenix 51 fair; Salt Lake City 33 snow; San Diego 54 fair; San Francisco 47 fair; Seattle 38 fair.

Canada: Montreal 27 cloudy; Toronto 28 cloudy.
Heavy snow covered parts of Wyoming, Utah and Colorado early today, with more than 3 feet of fresh snow reported in at least one town.

A winter storm warning for heavy snow was issued across the northern mountains of Utah, where Snowbird already had 37 inches by Sunday evening. Alta had 20 inches and downtown Salt Lake City had 12 to 16 inches.

A heavy snow warning also was posted over central and southern Wyoming, with 16 inches of new snow already heaped on Lander, and 10 inches near Riverton and Shoshone. The southwest mountains of Colorado were expecting heavy snow tonight and the the Nebraska Panhandle and northwest Kansas were under winter storm watches for possibly heavy snow.

Travel advisories for snow and icy roads stretched across southeast Idaho, northern Wyoming, the Nebraska Panhandle, Nevada's Lake Tahoe area and California's Sierra Nevada mountains and Kings Canyon.

Snow also fell from northwest Iowa across Nebraska, and over western South Dakota, southeast Montana and eastern Idaho.

Advisories for gusty winds were issued over the southern mountains and deserts of California.

Snowfall during the six-hour period ending at 1 a.m. EST included 6 inches at Lander and 5 inches at Casper Wyo., 4 inches at Fort Collins, Colo., and 3 inches at Alliance, Ogallala and Scottsbluff, Neb.

Farther east, snow reached from northern lower Michigan over upper Michigan and northeast Minnesota. And freezing rain was scattered over eastern and central New York.

And in Florida, a late-season tropical depression in the Gulf of Mexico was called more of an oddity than a threat by forecasters at the National Hurricane Centers in Miami.

Sunday night, the depression was about 100 miles northwest of Balboa, Panama, with maximum winds of about 35 mph. If the depression strengthens to maximum sustained winds of 39 mph, it will be named Tropical Storm Larry.

Today's forecast called for snow from western South Dakota and Nebraska across northwest Kansas, Wyoming, Colorado, Utah, northern Arizona, eastern Nevada and eastern Idaho, turning to rain over Oklahoma.

High temperatures should be in the teens and 20s over northern New England, upper Michigan, Minnesota, the northern half of the Plains and the northern and central Rockies. Highs in the 30s and
40s should stretch from the rest of New England down through the Mid-Atlantic States, much of the Great Lakes, the Ohio Valley, Kansas, the southern Rockies and the northern Pacific Coast. Highs should range in the 70s over central and southern Florida the central Gulf Coast, reaching into the 80s over southern Texas. Temperatures around the nation at 2 a.m. EST ranged from 2 degrees at Havre, Mont., to 74 at West Palm Beach, Fla.

Other reports:

_East: Atlanta 49 cloudy; Boston 36 fair; Charleston, S.C. 39 fair; Cincinnati 26 foggy; Cleveland 34 cloudy; Detroit 34 foggy; Miami 65 fair; New York 39 fair; Philadelphia 39 fair; Pittsburgh 29 fair; Portland 30 fair; Washington 35 fair.

_Central: Bismarck 27 cloudy; Chicago 26 foggy; Dallas-Fort Worth 61 fair; Denver 28 snow; Des Moines 16 foggy; Indianapolis 29 foggy; Kansas City 28 foggy; Minneapolis-St. Paul 23 foggy; Nashville 44 partly cloudy; New Orleans 53 fair; St. Louis 33 foggy.

_West: Albuquerque 39 fair; Anchorage 31 cloudy; Las Vegas 41 fair; Los Angeles 54 fair; Phoenix 51 fair; Salt Lake City 33 snow; San Diego 54 fair; San Francisco 47 fair; Seattle 38 fair.

_Canada: Montreal 27 cloudy; Toronto 28 cloudy.
### Article 9: unformatted version

type: The New York Times  
priority: Unknown  
date: nyt-01-28-86 0059est  
category: Unknown  
version: Undated  

text:

```
player Alvarez     34 222 2  .111
player Caruso     22 321 21 .234
player Keane      99 98 34 .222
player LeClair    43 200 12 .333
player Turpin     32 213 1  .444
player Remondini  27 200 12 .302

Soul Mate        22 44 55 666 Cat in the Hat 22 44 55 77  
this test       12 23 34 45 for double tabs 21 23 32 45  
Water bug fun   12 234 2 27 sorry boss 25 54 23 11  
Water bug test  12 34 56 12 Power forcaster 21 23 45 345

This is a test data file of the NYT tabbing system by VJD. I Hope it works.
```
Black Americans are much more likely to approve of how President Reagan is handling his job than they were three and four years ago, but they are far more critical than are whites, the latest New York Times-CBS News Poll shows.

The latest survey, conducted from Jan. 19 through 23, shows that 37 percent of blacks approve and 49 percent disapprove. In 1982, in contrast, 10 percent of blacks approved and 76 percent disapproved. Among whites in the latest poll, 68 percent approved and 21 percent disapproved.

The January finding on blacks is roughly equivalent to the figures in Times-CBS News Polls last September and November. But in a Times Poll in December, a much different picture -- 56 percent approval against 24 percent disapproval -- was found among blacks.

The polls have a margin of sampling error of plus or minus three percentage points for the national figures and a margin of error of plus or minus nine percentage points for the figures on black views; the margin of error becomes larger as the size of the sample is reduced, and the black respondents make up only about 12 percent of the population. But the difference between the two findings exceeded the margin of sampling error of the surveys.

Adam Clymer, assistant to the executive editor of The Times, who heads the newspaper's polling operations, was asked to explain the inconsistency. He said:

'It now appears that our December poll had a very unrepresentative black sample, especially of black men, and the findings plainly exceeded normal sampling error. This month's sample appears, on matters from education to household size, much more representative of the black population as a whole.'
Black Americans are much more likely to approve of how President Reagan is handling his job than they were three and four years ago, but they are far more critical than are whites, the latest New York Times-CBS News Poll shows.

The latest survey, conducted from Jan. 19 through 23, shows that 37 percent of blacks approve and 49 percent disapprove. In 1982, in contrast, 10 percent of blacks approved and 76 percent disapproved. Among whites in the latest poll, 68 percent approved and 21 percent disapproved.

The January finding on blacks is roughly equivalent to the figures in Times-CBS News Polls last September and November. But in a Times Poll in December, a much different picture -- 56 percent approval against 24 percent disapproval -- was found among blacks.

The polls have a margin of sampling error of plus or minus three percentage points for the national figures and a margin of error of plus or minus nine percentage points for the figures on black views; the margin of error becomes larger as the size of the sample is reduced, and the black respondents make up only about 12 percent of the population. But the difference between the two findings exceeded the margin of sampling error of the surveys.

Adam Clymer, assistant to the executive editor of The Times, who heads the newspaper's polling operations, was asked to explain the inconsistency. He said:

'It now appears that our December poll had a very unrepresentative black sample, especially of black men, and the findings plainly exceeded normal sampling error. This month's sample appears, on matters from education to household size, much more representative of the black population as a whole.'
APPENDIX E

Complete Listing of the Code

```c
#define TF '\001' /* tape feed symbol */
#define CW 4 /* column width on regular tabular data */
#define SCW 4 /* box score type tabular data column width */
#define TFI '\037' /* "tab field indicator" symbol */
#define LF '\012' /* "line feed (carriage return)" symbol */
#define MAXW 79 /* maximum line width */
#define SP '\040' /* "space" symbol */
#define ENL '\043' /* "EN leader" symbol */
#define EML '\052' /* "EM leader" symbol */
#define UP_RAL '\136' /* "Upper Rail" symbol */
#define TLI '\010' /* "tab line indicator" symbol */
#define MAXWG 38 /* maximum width for each side of side by side tab */
#define CFS '\034' /* "center field separator" symbol */
#define QC '\075' /* quad center character */
#define QL '\074' /* quad left character */
#define QR '\076' /* quad right character */
#include <stdio.h>
tabbera(inbuf,outbuf)
char inbuf[], outbuf[];
{
    int h;
    char 1[310], t[85];
    int const[7];
    int tp=78;
    int tg=38;

    const[5]=0;
    const[6]=0;
    fconst(inbuf,const);
    const[5]=0;
    const[6]=0;

    tp=78-const[3];
    tg=38-const[4];

    if (tg <= 0)
        tg= 38;

    while (getaln(inbuf,l,const) != NULL)
    {
        if (l[0] != TLI)
            {
                h=lffndr(l) -1;
                if (l[h] == QC)
                    {
                        h=h-1;
                    }
            }
    }
```
quadc(l,t,h);
a_stuff(outbuf,t,const);
}
else if (l[h] == QL)
{
    quadl(l,t);
a_stuff(outbuf,t,const);
}
else if (l[h] == QR)
{
    if (h>=const[1])
    {
        if (h>=(MAXW-6))
        {
            quadr(l,t,h,(h+1));
        }
        else
        {
            quadr(l,t,h,(h+5));
        }
    }
    else
    {
        quadr(l,t,h,const[1]);
a_stuff(outbuf,t,const);
    }
else
{
    a_stuff(outbuf,l,const);
}
}
else if (l[l] != TLI)
{
    tabber(l,t,tp);
a_stuff(outbuf,t,const);
}
else
{
    staber(l,t,tg);
a_stuff(outbuf,t,const);
}
}
e1:  h=const[5];
    outbuf[h]='\0';
    return;
}

fconst(name,const)
int const[];
char name[];
{
    int h;
    char l[310],t[85];
    int ki=1;
int ks=78;
int kd=40;
int kt=0;
int ka=0;
int k=0;

f2: while (getaln(name,1,const) != NULL)
{
if (1[0] != TLI)
{
    h=1ffndr(l)-1;
    if (l[h]==QC || l[h]==QL || l[h]==QR);
    else
    {
        if (h>kt)
            kt=h;
        ki=ki+1;
        ka=ka+h;
    }
}
else if (1[1] != TLI)
{
    k=tabber(1,t,78);
    if (k <= ks)
        ks=k;
}
else
{
    k=staber(1,t,38);
    if (k<=kd)
        kd=k;
}
}

f1: if (ki==1)
    ;
else
    ki=ki-1;

ka=ka/ki;
const[1]=kt;
const[2]=ka;
const[3]=ks;
const[4]=kd;
return;
}

tabber(l,t,tg)
char l[], t[];
int tg;
{
    int tp;
    int i=0;
    int a=1;
    int b;
    int j;
    int c;
    int tga=0;

    j=tg;
    /* clear the target array */
    aclear(t);

    tp=tg;
    t[tg+1]=LF;
    t[tg+2]='\0';

    /* fill the array from the front with printable material until reach TFI */
    while (l[a] != TFI)
    {
        if (i>MAXW)
        {
            return (MAXW-1);
        } else
        {
            if (l[a] == LF)
            {
                return (MAXW-1);
            } else
            {
                if ((decode(l[a])) == 0)
                {
                    a=a+1;
                } else
                {
                    t[i]=l[a];
                    a=a+1;
                    i=i+1;
                    tga=tga+1;
                }
            }
        }
    }

    /* l[a] = TFI now find the end of the line */
    b=a;
    while (l[b] != LF)
    {
        b=b+1;
    }

    /* l[b] = end of the line */
    /* fill array in from back to front until reach TFI */
    j=tg;
    tp=tg;
    c=b-1;

    /* find the first printable character of a column */
    start: if (l[c] == TFI)
return (j-tga);
if (prntbl(1[c]) == 0)
{
   c=c-1;
   goto start;
}

t[tp]=l[c];
j=tp-1;
c=c-1;

if (j <= tga)
   return 0;

/* keep filling in until find the end of the column */
middle: if (1[c] == TFI)
   return (j-tga);
if (((prntbl(l[c])) == 0)
{
   while (tp >= j)
      tp=tp-CW;
   if (tp <= tga)
      return (j-tga);
   c=c-1;
   goto start;
}
else
{
   t[j]=1[c];
   c=c-1;
   j=j-1;
   if (j <= tga)
      return 0;
   goto middle;
}
}

clear(t)
char t[];
{
   int i;

   for (i=0;i<=MAXW-1;i=i+1)
      t[i]=SP;

   t[MAXW]=\n;    
   t[MAXW+1]=\0;

} decode(a)
char a;
{
    if (a >= ' ')
    {
        if (a == ENL)
            return 0;
        else if (a == EML)
            return 0;
        else if (a == UP_RAL)
            return 0;
        else
            return 1;
    }
    else
        return 0;
}

prntbl(a)
char a;
{
    if (a > SP)
    {
        if (a == ENL)
            return 0;
        else if (a == EML)
            return 0;
        else if (a == UP_RAL)
            return 0;
        else
            return 1;
    }
    else
        return 0;
}

quadc(l,t,h)
char l[],t[];
int h;
{
    int tmps;
    int d;
    int e;
    int off=9;
    e=h;
    tmps=((MAXW-off)-e)/2;
    aclear(t);
d=(MAXW-off)-tmps;

t[d+1]=LF;
t[d+2]=\'\0\';

while (e >= 0)
{
    if (d<0)
        return;
    t[d]=l[e];
    d=d-1;
    e=e-1;
}

quadr(1,t,h,p)
char 1[]
int h,p;
{
    int e;
    int i;

    aclear(t);
    i=h-1;
    e=p;
    while (e>=0)
    {
        t[e]=l[i];
        if (i<=0)
            return;
        e=e-1;
        i=i-1;
    }
    return;
}

quad1(1,t)
char 1[]
{
    int i=0;
    int e=0;

    aclear(t);
    while (l[e]==SP)
    {
        e=e+1;
    }
    while (l[e]!=QL)
    {
        t[i]=l[e];
    }
i=i+1;
e=e+1;
}
t[i]=LF;
t[i+1]='\0';
return;
}

ffndr(s)
char s[];
{

int i;
for (i=0;i<= 300;i=i+1)
{
    if (s[i] == LF)
    {
        return i;
    }
}
return 0;
}

staber(l,t,tg)
char l[], t[];
int tg;
{
    int tp, tl;
    int i=0, ri=0;
    int a=3, ra;
    int b, rb;
    int j, rj;
    int c, rc;
    int cl, dl, bl;
    int kgsa, kgsb, ksa;

    j=MAXW;
    rj=MAXW;

    kgsa=MAXW-1;
    kgsb=MAXW-1;
    ksa=MAXW-1;

    bl=tg;
    cl=rg+3;
    dl=cl+tg;
/* clear the target array */
aclear(t);

    t[d1+1]=LF;
t[d1+2]='\0';

    tp=bl;
/* fill the array from the front with printable material until reach TFI */

while (l[a] != TFI)
{
    if (i>MAXW)
        return ksa;
    if (l[a] == LF)
        return ksa;
    if ((decode(l[a])) == 0)
        a=a+1;
    else
    {
        t[i]=l[a];
a=a+1;
        i=i+1;
    }
}

/* l[a] = TFI now find the CFS */

b=a;
while (l[b] != CFS)
{
    if (l[b]==LF)
    {
        return ksa;
        b=b+l;
    }

/* l[b] = CFS */

/* fill array in from center (bl) to front until reach TFI */

c=b;
j=tp;

start: if (l[c] == TFI)
{
    kgsa=j-i;
goto rside;
}
    if (prntbl(l[c]) == 0)
    {
        c=c-1;
goto start;
    }
t[tp]=l[c];
j=tp-1;
c=c-1;

if (j <= i)
{
    kgsa=1;
goto rside;
}

middle: if (l[c] == TFI)
{
    kgsa=j-i;
goto rside;
}
if ((prntbl(l[c])) == 0) {
    while (tp >= j)
    
    tp=tp-SCW;
c=c-1;
if (tp <= i)
{
    kgsa=1;
goto rside;
}
goto start;
}
else
{
    t[j]=l[c];
c=c-1;
j=j-1;
if (j <= i)
{
    kgsa=1;
goto rside;
}
goto middle;
}

/* format the right side */

rside: ra=b;
ri=c1;
rj=MAXW-1;
tl=d1;

/* fill the array from CFS with printable material until reach TFI */

while (l[ra] != TFI)
if (ri>MAXW)
    return ksa;
if (1[ra] == LF)
    return ksa;
if ((decode(1[ra])) == 0)
    ra=ra+1;
else
    { 
        t[ri]=1[ra];
        ra=ra+1;
        ri=ri+1;
    }
}

/* 1[ra] = TFI now find the end of the line */
rb=ra;
while (1[rb] != LF)
    rb=rb+1;

/* 1[rb] = end of the line */
/* fill array in from back to front until reach TFI */
rc=rb;
restart: if (1[rc] == TFI)
{ 
    kgsb=rj-ri;
    ksa=min(kgsa,kgsb);
    return ksa;
}
if (prntbl(1[rc]) == 0)
{ 
    rc=rc-1;
    goto restart;
}
else
{ 
    t[t1]=1[rc];
    rj=t1-1;
    rc=rc-1;
}
if (rj <=ri)
    return ksa;
rmidle: if (1[rc] == TFI)
{ 
    kgsb=rj-ri;
    ksa=min(kgsa,kgsb);
    return ksa;
if ((prntbl(l[rc])) == 0) {
    while (tl >= rj)
        tl=tl-SCW;
    rc=rc-1;
    if (tl <= ri)
        return ksa;
    goto rstart;
} else {
    t[rj]=1[rc];
    rc=rc-1;
    rj=rj-1;
    if (rj <= ri)
        return ksa;
    goto rmiddle;
}

min(a,b)
int a,b;
{
    if (a <= b)
        return a;
    else
        return b;
}

a_stuff(targ,srce,strt)
char targ[],srce[];
int strt[];
{
    int i;
    int j;
    j=strt[5];
    i=0;
    while (srce[i]!='$0')
    {
        targ[j]=srce[i];
        i=i+1;
        j=j+1;
    }
    targ[j]='$0';
    strt[5]=j;
getln(src, targ, const)
char src[], targ[];
int const[];
{
    int i=0;
    int j;
    j=const[6];
    while (src[j] != LF)
    {
        if (src[j]=='\0')
            return NULL;
            targ[i]=' '@;
        else
            targ[i]=src[j];
        i=i+1;
        j=j+1;
        if (i>300)
            return NULL;
    }
    i=i-1;
    while (targ[i] == SP)
    {
        i=i-1;
    }
    targ[i+1]=LF;
    targ[i+2]=\0';
    j=j+1;
    const[6]=j;
    return 1;
}

tabbern(inbuf, outbuf)
char inbuf[], outbuf[];
{
    int h;
    char l[310], t[85];
    int const[8];
    int tp=78;
    int tg=38;

    const[5]=0;
    const[6]=0;
    const[7]=0;
    nconst(inbuf, const);
    const[6]=0;
    const[7]=0;

    tp=78-const[3];
tg = 38 - const[4];
if (tg <= 0)
tg = 38;

e3: while (ngetaln(inbuf, l, const) != NULL)
{
    if (l[0] != TLI)
    {
        a_stuff(outbuf, l, const);
    }
    else if (l[1] != TLI)
    {
        tabber(l, t, tp);
        a_stuff(outbuf, t, const);
    }
    else
    {
        staber(l, t, tg);
        a_stuff(outbuf, t, const);
    }
}
el: h = const[5];
outbuf[h] = '\0';
return;

ngetaln(src, targ, const)
char src[], targ[];
int const[];
{
    int i = 0;
    int j;
    j = const[6];
    while (src[j] != LF)
    {
        if (src[j] == '\0')
            return NULL;
            targ[i] = ' ';
        else
            targ[i] = src[j];
        i = i + 1;
        j = j + 1;
        if (i > 300)
            return NULL;
    }
    targ[i] = LF;
    targ[i + 1] = '\0';
j=j+1;
const[7]=const[6];
const[6]=j;
return 1;
}

nconst(name,const)
int const[];
char name[];
{
    int h;
    int i;
    char l[310],t[85];
    int ks=78;
    int kd=40;
    int k=0;

f2:  while (ngeta1n(name,l,const) != NULL)
    {
if (l[0] != TF)
    ;
else if (l[1] != TF)
    {
        k=ntabs(1,78,name,const);
        if (k <= ks)
            ks=k;
    }
else
    {
        k=nstab(1,38,name,const);
        if (k<=kd)
            kd=k;
    }
}

f1:  const[1]=78;
const[2]=78;
const[3]=ks;
const[4]=kd;
return;
}

ntabs(l,tg,name,const)
char l[],name[]; 
int const[];
int tg;
{
    int tp;
    int i=0;
    int a=1;
int b;
int j;
int c;
int tga=0;

j=tg;
name[const[7]]=TLI;

tp=tg;

/* find where TFI should be placed and insert it */

al: if (1[a]!=SP)
{
    if (i>MAXW)
        return (MAXW-1);
    if (1[a] == LF)
        return (MAXW-1);
    if ((decode(1[a])) == 0)
        a=a+1;
    else
    {
        a=a+1;
        i=i+1;
        tga=tga+1;
    }
    goto al;
}
else
{
    if (tfer(1[a+1])==1)
    {
        a=a+1;
        name[const[7]+a]=TFI;
        1[a]=TFI;
        goto bl;
    }
    else
    {
        a=a+1;
        i=i+1;
        tga=tga+1;
        goto al;
    }
}

/* 1[a] = TFI  now find the end of the line */

bl:  b=a;
    while (1[b] != LF)
        b=b+1;
/* 1[b] = end of the line */
/* fill array in from back to front until reach TFI */
    j=tb;
    tp=tb;
    c=b-1;

/* find the first printable character of a column */

start: if (1[c] == TFI)
    return (j-tb);
if (prntbl(l[c]) == 0)
{
    c=c-1;
    goto start;
}

j=tp-1;
    c=c-1;

if (j <= tba)
    return 0;

/* keep filling in until find the end of the column */

middle: if (1[c] == TFI)
    return (j-tb);
if ((prntbl(l[c]) == 0)
{
    while (tp >= j)
        tp=tp-CW;
    if (tp <= tba)
        return (j-tb);
    c=c-1;
    goto start;
}
else
{
    c=c-1;
    j=j-1;
    if (j <= tba)
        return 0;
    goto middle;
}

nstab(1,tb,name,const)
char 1[],name[];
int const[];
int tg;
{
    int tp, tl;
    int i=0, ri=0;
    int a=3, ra;
    int b, rb;
    int j, rj;
    int c, rc;
    int cl, dl, bl;
    int kgsa, kgsb, ksa;

    j=MAXW;
    rj=MAXW;

    kgsa=MAXW-1;
    kgsb=MAXW-1;
    ksa=MAXW-1;

    bl=tg;
    cl=tg+3;
    dl=cl+tg;

    /* clear the target array */
    name[const[7]]=TLI;

    tp=bi;
    /* fill the array from the front with printable material until reach TFI */
    al:    if (l[a] != SP)
    {
        if (i>MAXW)
            return ksa;
        if (l[a] == LF)
            return ksa;
        if ((decode(l[a])) == 0)
            a=a+1;
        else
        {
            a=a+1;
            i=i+1;
        }
        goto al;
    }
    else
    {
        if (tfer(l[a+1])==1)
        {
            a=a+1;
            name[const[7]+a]=TFI;
        }
    }
}
1[a]=TFI;
goto bl;
}
else
{
    a=a+1;
i=i+1;
goto a1;
}

/* 1[a] = TFI now find the CFS */

bl:   b=a;
    while (1[b] != TF)
    {
        if (1[b]==LF)
            return ksa;
        b=b+1;
    }
    name[const[7]+b]=CFS;
    1[b]=CFS;

/* 1[b] = CFS */
/* fill array in from center (bl) to front until reach TFI */

c=b;
    j=tp;

start: if (1[c] == TFI)
    {
        kgsa=j-i;
goto rside;
    }
    if (prntbl(1[c]) == 0)
    {
        c=c-1;
goto start;
    }

j=tp-1;
c=c-1;

if (j <= i)
{
    kgsa=1;
goto rside;
}

middle: if (1[c] == TFI)
kgsa=j-i;
goto rside;
}
if ((prntbl(1[c])) == 0) {

while (tp >= j)
    tp=tp-SCW;
c=c-1;
if (tp <= i) {
    kgsa=1;
goto rside;
}
goto start;
} else {

    c=c-1;
    j=j-1;
    if (j <= i) {
        kgsa=1;
goto rside;
    }
goto middle;
}

/* format the right side */

rside: ra=b;
ri=cl;
rj=MAXW-1;
tl=dl;

/* fill the array from CFS with printable material until reach TFI */
a2: if (1[ra] != SP) {

    if (ri>MAXW)
        return ksa;
    if (1[ra] == LF)
        return ksa;
    if (((decode(1[ra])) == 0)
        ra=ra+1;
    else {
        ra=ra+1;
        ri=ri+1;
    }
goto a2;
}
else
{
    if (tfrr(l[ra+1]==1)
    {  
        ra=ra+1;
        l[ra]=TFI;
        goto b2;
    }
    else
    {
        ra=ra+1;
        ri=ri+1;
        goto a2;
    }
}

/* l[ra] = TFI  now find the end of the line */

b2:
    rb=ra;
    while (l[rb] != LF)
        rb=rb+1;

/* l[rb] = end of the line */
/* fill array in from back to front until reach TFI */

rc=rb;

rstart: if (l[rc] == TFI)
{  
    kgsb=rj-ri;
    ksa=min(kgsa,kgsb);
    return ksa;
}
if (prntbl(l[rc]) == 0)
{  
    rc=rc-1;
    goto rstart;
}
else
{
    rj=tl-1;
    rc=rc-1;
}
if (rj <=ri)
    return ksa;

rmiddle: if (l[rc] == TFI)
{  
    kgsb=rj-ri;
ksa=min(kgsa,kgsb);
return ksa;

if ((prntbl(1[rc])) == 0)
{
    while (t1 >= rj)
    {
        tl=tl-SCW;
        rc=rc-1;
        if (tl <= ri)
            return ksa;
        goto rstart;
    }
    else
    {
        rc=rc-1;
        rj=rj-1;
        if (rj <= ri)
            return ksa;
        goto rmidle;
    }
}

tfer(a)
char a;
{
    if (a==' ')
        return 1;
    else
        return 0;
}
## APPENDIX H

### High Speed Wire Service ASCII Code Table

<table>
<thead>
<tr>
<th>code</th>
<th>character</th>
<th>code</th>
<th>character</th>
<th>code</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>43</td>
<td>+</td>
<td>86</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>44</td>
<td></td>
<td>87</td>
<td>W</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>45</td>
<td>hyphen</td>
<td>88</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>46</td>
<td></td>
<td>89</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>47</td>
<td>/</td>
<td>90</td>
<td>Z</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>48</td>
<td>0</td>
<td>91</td>
<td>1/8</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>49</td>
<td>1</td>
<td>92</td>
<td>1/4</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>50</td>
<td>2</td>
<td>93</td>
<td>3/8</td>
</tr>
<tr>
<td>8</td>
<td>TLI</td>
<td>51</td>
<td>3</td>
<td>94</td>
<td>upper rail</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>52</td>
<td>4</td>
<td>95</td>
<td>em dash</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>53</td>
<td>5</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>54</td>
<td>6</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>55</td>
<td>7</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>56</td>
<td>8</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>57</td>
<td>9</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>58</td>
<td>:</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>16</td>
<td>TTS Space</td>
<td>59</td>
<td>;</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>17</td>
<td>DC1</td>
<td>60</td>
<td>QL</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>18</td>
<td>DC2</td>
<td>61</td>
<td>QC</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>19</td>
<td>DC3</td>
<td>62</td>
<td>QR</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
<td>63</td>
<td>?</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>21</td>
<td>NAK</td>
<td>64</td>
<td>lower rail</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>22</td>
<td>SYN</td>
<td>65</td>
<td>A</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>23</td>
<td>ETB</td>
<td>66</td>
<td>B</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
<td>67</td>
<td>C</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>25</td>
<td>EM space</td>
<td>68</td>
<td>D</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>26</td>
<td>SUB</td>
<td>69</td>
<td>E</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>27</td>
<td>ESC</td>
<td>70</td>
<td>F</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>28</td>
<td>CPS</td>
<td>71</td>
<td>G</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>29</td>
<td>THIN space</td>
<td>72</td>
<td>H</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>30</td>
<td>EN space</td>
<td>73</td>
<td>I</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>31</td>
<td>TFI</td>
<td>74</td>
<td>J</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>32</td>
<td>SP</td>
<td>75</td>
<td>K</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>76</td>
<td>L</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>77</td>
<td>M</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>35</td>
<td>EN leader</td>
<td>78</td>
<td>N</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>79</td>
<td>O</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>80</td>
<td>P</td>
<td>123</td>
<td>1/2</td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>81</td>
<td>Q</td>
<td>124</td>
<td>5/8</td>
</tr>
<tr>
<td>39</td>
<td>'</td>
<td>82</td>
<td>R</td>
<td>125</td>
<td>3/4</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>83</td>
<td>S</td>
<td>126</td>
<td>7/8</td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>84</td>
<td>T</td>
<td>127</td>
<td>DEL</td>
</tr>
<tr>
<td>42</td>
<td>EM leader</td>
<td>85</td>
<td>U</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


