REPORT ON THE ANSI X3J6 MEETING

Lynne A. Price

Supported by TUG, I spent January 25–29 in Lancaster, Pennsylvania attending a meeting of the ANSI X3J6 Text Processing Language Standards Committee. The committee is defining a standard language with facilities for text editing, text formatting, and generalized markup. For text editing, the object is to identify basic editing functions and a macro facility so that an individual user can take a personal macro file from system to system and not need to learn a new editor for each computer. For text formatting, the object is to be able to produce readable output on different systems from a single source file. It is understood that line breaks, hyphenation, page breaks, and so on cannot be preserved across different facilities. The output devices considered include daisy-wheel printers, word-processing equipment, and high-resolution typesetters. Text markup refers to labelling elements of a document—titles, chapters, footnotes, etc. The goal for generalized markup is to itemize the elements needed for common types of documents, so that input for various document formatters could be automatically prepared from a source file containing the text to be formatted interspersed with markup codes. Thus, preprocessors might exist to translate source files from the standard markup language to TEX input form, to SCRIBE input form, to APS-5 input form, etc.

Of the fifteen individuals in Lancaster, approximately half were committee members (to join, an individual must attend two meetings and pay $100). This attendance is fairly typical, although the mailing list has about sixty names. The committee has been meeting four times a year, for week-long sessions. The next meetings are scheduled for Phoenix in April, Edmonton in August, New Hampshire in October, and the Bay Area in January or February. If the current schedule, which calls for completion of the standard in 1983, can be met, only three meetings will be required next year. Once the standard is approved, the committee will continue to have short meetings once or twice a year; activity will then increase as the five-year review approaches.

I can forward a copy of the not-yet-completed draft standard (dated just before the Lancaster meeting) to any interested TUG member. The X3J6 formatting language has been greatly influenced by the concepts of boxes and glue as used by TEX. It is currently assumed that it will be easy to translate, in both directions, between the eventual standard language and TEX. Several committee members also belong to TUG. However, none of the X3J6 members in Lancaster yet has access to TEX. As a TEX user, I was repeatedly able to contribute to the discussion. During the week, topics pertinent to formatting ranged over paragraph justification, word spacing, letter spacing, line spacing, leaders, rules, and page layout. I learned quite a bit about typesetting. Subtopics I found most interesting involved generalizations of structures and algorithms used by TEX.

It is very clear that X3J6 can benefit from involvement by TUG. There are advantages to the TEX community as well. X3J6 is formed of individuals knowledgeable in both typesetting and automatic text processing. Until the TEX language stabilizes, X3J6 can comment on its applicability to general, non-mathematical typesetting. There has always been interest within TUG in a possible "Son of TEX"; X3J6 may be an outlet for future generalizations. Finally, X3J6 and TUG have a common interest in separating font sales from sales of typesetting equipment. For the above reasons, I recommend that TUG continue to finance a representative at X3J6 meetings. Although we granted the Finance Committee authority to make this decision in Cincinnati, we can all provide input to the process through TUGboat, mail, and telephone.

* * * * * * * * * * *

Software

* * * * * * * * * * *

FIXED-POINT GLUE SETTING AN EXAMPLE OF WEB

Donald E. Knuth
Stanford University

The "definitive" version of TEX is being written in a new language called WEB, which is a mixture of TEX and PASCAL. I will soon be publishing a complete manual about WEB, but in the meantime I think it will be useful to have an example of a fairly short piece of code written in "web" form. Therefore I have prepared the accompanying program, which also serves another function: It illustrates how to remove the last vestiges of floating-point arithmetic from the new TEX.

The eleven pages that follow this introduction contain the example program in its "woven" form, including the table of contents and the two indices that are generated automatically. I hope the reader can guess how WEB works just by looking at this particular example. The PASCAL version of the TEX process or will eventually appear in the same format, only it will be somewhat longer.

The twelfth page, which is page 23 of this issue of TUGboat, is an example of the output generated by the fixed-point routines. And the page after that is the actual PASCAL program that was produced from the "web". (This PASCAL code isn't very readable, but it is intended to be read only by the PASCAL compiler, except in rare emergencies. It does contain cross-references that show where each numbered part of the web has been inserted.)

Following the PASCAL code I have attached an example page of the WEB file, which is what I actually typed into the computer. This file, GLUE.WEB, was the source of everything else. A program called TANGLE took GLUE.WEB as input and produced the PASCAL code GLUE.PAS as output; I never looked at that output, I just let PASCAL compile it. Another program called WEAVE took GLUE.WEB as input and produced GLUE.TEX as output. (A sample page of GLUE.TEX appears after the sample page of GLUE.WEB, so that you can see what WEAVE does.) When TEX processed GLUE.TEX, the result was the eleven pages that I mentioned first; you should read these eleven pages first.

How much computer time did this all take? I didn't gather exact data, which is not easy to obtain on our time-shared DEC-10 computer, but the following approximate times are fairly accurate: TANGLE took two seconds to convert the WEB file to the PAS file, PASCAL took two seconds to convert that to a REL file, the system loader took two seconds to get the program in memory, and the program produced its output in a small fraction of a second. Furthermore WEAVE took four seconds to convert the WEB file to the TEX file, TEX took 40 seconds to convert that to an output file (in this case a PRESS file for the Dover printer), and the hardcopy output was printed by the time I walked down one flight of stairs to the printer room. You have to multiply the TANGLE-PASCAL-load-run time by about 5, since I went through five passes while debugging; and you have to multiply the WEAVE-TEX-print time by 2, since this is my second draft.

How much human time did it take? I spent a full day considering various ways to do the necessary fixed-point computations, until deciding that this scheme was preferable to another that was based on two 16-bit integers instead of powers of 2. I spent about three hours writing the WEB code, about two hours typing it into the computer and editing it as I went, and about two hours proofreading and debugging.

The bugs turned out to be mostly typographical or related to fussy details, since the web structure made my program so clear (to me at least) that I was pretty sure it was correct as I wrote it. Here are the bugs I remember making:

1) I forgot that WEB doesn't allow me to use its special notation for octal constants in a comment, unless the constant appears in "PASCAL mode".
2) In one place I typed 'global' instead of 'Global', so WEB could not match the two names.
3) I left a dollar sign off at the end of a formula. (This later caused TEX to give an error message that I had an extra right brace; then it said I couldn't do something-or-other in restricted horizontal mode.)
4) I forgot that PASCAL doesn't allow a function to return a structured type.
5) I forgot to declare the variables a, b, and c in one procedure.
6) I used '*write*' instead of '*writeln*' in one place.
7) I left off the **begin** and **end** that now surround the module called ⟨Compute c by long division⟩.
8) I used s instead of ss in the so-called "easy case".

Note that there are bugs in my use of WEB, in my use of PASCAL, in my use of TEX, and in my algorithm. But I believe the total number of bugs would have been a lot more if I had programmed separately in PASCAL and written a separate description in TEX. And the final documentation is not only better than I know how to make by any other method, it also is guaranteed to be a documentation of exactly the program as it describes, since the documentation and the program were generated by the same WEB source file.

As I gain more experience with WEB, I am finding that it significantly improves my ability to write reliable programs quickly. This is a pleasant surprise, since I had designed WEB mainly as a documentation tool.

Fixed-point Glue Setting

This research was supported in part by the National Science Foundation under grants IST-7921977 and MCS-7723738; by Office of Naval Research grant N00014-81-K-0330; and by the IBM Corporation.

1. Introduction. If TeX is being implemented on a microcomputer that does 32-bit addition and subtraction, but with multiplication and division restricted to 16-bit multipliers and divisors, it can still do the computations associated with the setting of glue in a suitable way. This program illustrates one solution to the problem.

Another purpose of this program is to provide the first "short" example of the use of WEB.

2. The program itself is written in standard PASCAL. It begins with a normal program header, most of which will be filled in with other parts of this "web" as we are ready to introduce them.

```
program GLUE(input, output);
  type ⟨Types in the outer block 6⟩
  var ⟨Globals in the outer block 8⟩
  procedure initialize; {this procedure gets things started}
    var ⟨Local variables for initialization 9⟩
    begin ⟨Set initial values 10⟩
    end;
```

3. Here are two macros for common programming idioms.

```
  define incr(#) ≡ # ← # + 1   {increase a variable by unity}
  define decr(#) ≡ # ← # − 1   {decrease a variable by unity}
```

4. The problem and a solution. We are concerned here with the "setting of glue" that occurs when a TEX box is being packaged. Let $x_1, \ldots, x_n$ be integers whose sum $s = x_1 + \cdots + x_n$ is positive, and let t be another positive integer. These x_i represent scaled amounts of glue in units of spt (scaled points), where one spt is 2^{-16} of a printer's point. The other quantity t represents the total by which the glue should stretch or shrink. Following the conventions of TEX82, we will assume that the integers we deal with are less than 2^{31} in absolute value.

After the glue has been set, the actual amounts of incremental glue space (in spt) will be the integers $f(x_1), \ldots, f(x_n)$, where f is a function that we wish to compute. We want $f(x)$ to be nearly proportional to x, and we also want the sum $f(x_1) + \cdots + f(x_n)$ to be nearly equal to t. If we were using floating-point arithmetic, we would simply compute $f(x) = (t/s) \cdot x$ and hope for the best; but the goal here is to compute a suitable f using only the fixed-point arithmetic operations of a typical "16-bit microcomputer."

The solution adopted here is to determine integers a, b, c such that

$$f(x) = \lfloor 2^{-b} c \lfloor 2^{-a} x \rfloor \rfloor$$

if x is positive. Thus, we take x and shift it right by a bits, then multiply by c (which is 2^{15} or less), and shift the product right by b bits. The quantities a, b, and c are to be chosen so that this calculation doesn't cause overflow and so that $f(x_1) + \cdots + f(x_n)$ is reasonably close to t.

The following method is used to calculate a and b: Suppose

$$y = \max_{1 \le i \le n} |x_i|.$$

Let d and e be the smallest integers such that $t < 2^d s$ and $y < 2^e$. Since s and t are less than 2^{31}, we hve $-30 \le d \le 31$ and $1 \le e \le 31$. An error message is given if $d + e > 31$; in such a case some x_m has $|x_m| \ge 2^{e-1}$ and we are trying to change $|x_m|$ to $|(t/s)x_m| \ge 2^{d+e-2} \ge 2^{30}$ spt, which TEX does not permit. (Consider, for example, the "worst case" situation $x_1 = 2^{30} + 1$, $x_2 = -2^{30}$, $t = 2^{31} - 1$; surely we need not bother trying to accommodate such anomalous combinations of values.) On the other hand if $d + e \le 31$, we set $a = e - 16$ and $b = 31 - d - e$. Notice that this choice of a guarantees that $\lfloor 2^{-a} x \rfloor < 2^{16}$. We will choose c to be at most 2^{15}, so that the product will be less than 2^{31}.

The computation of c is the tricky part. The "ideal" value for c would be $\rho = 2^{a+b} t/s$, since $f(x)$ should be approximately $(t/s) \cdot x$. Furthermore it is better to have c slightly larger than ρ, instead of slightly smaller, since the other operations in $f(x)$ have a downward bias. Therefore we shall compute $c = \lceil \rho \rceil$. Since $2^{a+b} t/s < 2^{a+b+d} = 2^{15}$, we have $c \le 2^{15}$ as desired.

We want to compute $c = \lceil \rho \rceil$ exactly in all cases. There is no difficulty if $s < 2^{15}$, since c can be computed directly using the formula $c = \lfloor (2^{a+b} t + s - 1)/s \rfloor$; we have $2^{a+b} t < 2^{15} s < 2^{30}$.

Otherwise let $s = s_1 2^l + s_0$, where $2^{14} \le s_1 < 2^{15}$ and $0 \le s_0 < 2^l$. We will essentially carry out a long division. Let t be "normalized" so that $2^{30} \le 2^h t < 2^{31}$ for some h. Then we form the quotient and remainder of $2^h t$ divided by s_1,

$$2^h t = q s_1 + r.$$

It follows that $2^{h+l} t - qs = 2^l r - s_0 q = R$, say. If $0 \ge R > -s$ we have $q = \lceil 2^{h+l} t/s \rceil$; otherwise we can replace (q, R) by $(q \pm 1, R \mp s)$ until R is in the correct range. It is not difficult to prove that q needs to be increased at most once and decreased at most seven times, since $2^l r - s_0 q < 2^l s_1 \le s$ and $s_0 q/s \le (2^h t/s_1)(s_0/2^l s_1) < 2^{31}/s_1^2 \le 8$. Finally $c = \lceil 2^{a+b-h-l} q \rceil$; and we have $a + b - h - l = -1$ or -2, since $2^{28+l} \le 2^{14} s = 2^{a+b+d-1} s \le 2^{a+b} t < 2^{a+b+d} s = 2^{15} s < 2^{30+l}$ and $2^{30} \le 2^h t < 2^{31}$.

An error analysis shows that these values of a, b, and c work satisfactorily, except in unusual cases where we wouldn't expect them to. We have

$$\begin{aligned} f(x) &= 2^{-b}(2^{a+b} t/s + \theta_0)(2^{-a} x - \theta_1) - \theta_2 \\ &= (t/s)x + \theta_0 2^{-a-b} x - \theta_1 2^a t/s - 2^{-b} \theta_0 \theta_1 - \theta_2 \end{aligned}$$

where $0 \le \theta_0, \theta_1, \theta_2 < 1$. Now $0 \le \theta_0 2^{-a-b} x < 2^{e-a-b} = 2^{d+e-15}$ and $0 \le \theta_1 2^a t/s < 2^{a+d} = 2^{d+e-16}$, and the other two terms are negligible. Therefore $f(x_1) + \cdots + f(x_n)$ differs from t by at most about $2^{d+e-15} n$. Since 2^{d+e} spt is larger than the largest stretching or shrinking of glue after expansion, the error is at worst about $n/32000$ times as much as this, so it is quite reasonable. For example, even if fill glue is being used to stretch 20 inches, the error will still be less than $\frac{1}{1600}$ of an inch.

5. To sum up: Given the positive integers s, t, and y as above, we set $a \leftarrow \lfloor \lg y \rfloor - 15$, $b \leftarrow 29 - \lfloor \lg y \rfloor - \lfloor \lg t/s \rfloor$, and $c \leftarrow \lceil 2^{a+b} t/s \rceil$. The implementation below shows how to do the job in PASCAL without using large numbers.

6. TEX wants to have the glue-setting information in a 32-bit data type called *glue_ratio*. The PASCAL implementation of TEX82 has *glue_ratio* = *real*, but alternative definitions of *glue_ratio* are explicitly allowed.

For our purposes we shall let *glue_ratio* be a record that is packed with three fields: The *a_part* will hold the positive integer $a + 16$, the *b_part* will hold the nonnegative integer b, and the *c_part* will hold the nonnegative integer c. Note that we have only about 25 bits of information in all, so it should fit in 32 bits with ease.

```
⟨Types in the outer block 6⟩ ≡
  glue_ratio = packed record a_part: 0 .. 31;  {the quantity a + 16 in our derivation}
    b_part: 0 .. 31;  {the quantity b in our derivation}
    c_part: 0 .. '100000;  {the quantity c in our derivation}
    end;
  scaled = integer;  {this data type is used for quantities in spt units}
```
This code is used in section 2.

7. The real problem is to define the procedures that TEX needs to deal with such *glue_ratio* values: (a) Given scaled numbers s, t, and y as above, to compute the corresponding *glue_ratio*. (b) Given a scaled number x and a *glue_ratio* g, to compute the scaled number $f(x)$. (c) Given a *glue_ratio* g, to print out a decimal equivalent of g for diagnostic purposes.

8. Glue multiplication. The easiest procedure of the three just mentioned is the one that is needed most often, namely, the computation of $f(x)$.

PASCAL doesn't have built-in binary shift commands or built-in exponentiation, although many computers do have this capability. Therefore our arithmetic routines use an array called '*two_to_the*', containing powers of two. Divisions by powers of two are never done in the programs below when the dividend is negative, so the operations can safely be replaced by right shifts on machines for which this is most appropriate. (Contrary to popular opinion, the PASCAL operation 'x **div** 2' is not the same as shifting x right one binary place, when x is a negative odd integer, if the computer uses two's complement arithmetic. But division is equivalent to shifting when x is nonnegative.)

```
⟨Globals in the outer block 8⟩ ≡
two_to_the: array [0 .. 30] of integer;   { two_to_the[k] = $2^k$ }
```
See also sections 15 and 20.
This code is used in section 2.

9. ⟨Local variables for initialization 9⟩ ≡
```
k: 1 .. 30;   { an index for initializing two_to_the }
```
This code is used in section 2.

10. ⟨Set initial values 10⟩ ≡
```
  two_to_the[0] ← 1;
  for k ← 1 to 30 do two_to_the[k] ← two_to_the[k − 1] + two_to_the[k − 1];
```
This code is used in section 2.

11. The glue-multiplication function f can now be written:
```
  define ga ≡ g.a_part   { convenient abbreviations }
  define gb ≡ g.b_part   { as alternatives to }
  define gc ≡ g.c_part   { PASCAL's with statement }
function glue_mult(x : scaled; g : glue_ratio): integer;   { returns f(x) as above, assuming that x ≥ 0 }
  begin if ga > 16 then x ← x div two_to_the[ga − 16]   { right shift by a places }
  else x ← x * two_to_the[16 − ga];   { left shift by −a places }
  glue_mult ← (x * gc) div two_to_the[gb];   { right shift cx by b places }
  end;
```

12. Glue setting. The *glue_fix* procedure computes a, b, and c by the method explained above. TeX does not normally compute the quantity y, but it would not be difficult to make it do so.

This procedure would be a function that returns a *glue_ratio*, if PASCAL would allow functions to produce records as values.

```
procedure glue_fix(s, t, y : scaled; var g : glue_ratio);
  var a, b, c: integer;   {components of the desired ratio}
    k, h: integer;   {30 − ⌊lg s⌋, 30 − ⌊lg t⌋}
    ss: integer;   {original (unnormalized) value of s}
    q, r, v: integer;   {quotient, remainder, divisor}
    w: integer;   {2^l}
  begin ⟨Normalize s, t, and y, computing a, k, and h 13⟩;
  if t < s then b ← 15 − a − k + h else b ← 14 − a − k + h;
  if b < 0 then
    begin write_ln('!␣Excessive␣glue.');   {error message}
    b ← 0; c ← 1;   {make f(x) = ⌊2^(−a) x⌋}
    end
  else begin if k ≥ 16 then   {easy case, s < 2^15}
      c ← (t div two_to_the[h − a − b] + ss − 1) div ss
    else ⟨Compute c by long division 14⟩;
    end;
  ga ← a + 16; gb ← b; gc ← c;
  end;
```

13. ⟨Normalize s, t, and y, computing a, k, and h 13⟩ ≡

```
  begin a ← 15; k ← 0; h ← 0; ss ← s;
  while y < '10000000000 do   {y is known to be positive}
    begin decr(a); y ← y + y;
    end;
  while s < '10000000000 do   {s is known to be positive}
    begin incr(k); s ← s + s;
    end;
  while t < '10000000000 do   {t is known to be positive}
    begin incr(h); t ← t + t;
    end;
  end
```

This code is used in section 12.

14. ⟨Compute c by long division 14⟩ ≡

```
  begin w ← two_to_the[16 − k]; v ← ss div w; q ← t div v; r ← ((t mod v) * w) − ((ss mod w) * q);
  if r > 0 then
    begin incr(q); r ← r − ss;
    end
  else while r ≤ −ss do
      begin decr(q); r ← r + ss;
      end;
  if a + b + k − h = −17 then c ← (q + 1) div 2   {l = 16 + k − h}
  else c ← (q + 3) div 4;
  end
```

This code is used in section 12.

15. Glue-set printing. The last of the three procedures we need is *print_glue*, which displays a *glue_ratio* in symbolic decimal form. Before constructing such a procedure, we shall consider some simpler routines, copying them from TEX.

```
  define unity ≡ ´200000   { $2^{16}$, represents 1.0000 }
⟨Globals in the outer block 8⟩ +≡
dig: array [0 .. 15] of 0 .. 9;   { for storing digits }
```

16. An array of digits is printed out by *print_digs*.

```
procedure print_digs(k : integer);   { prints dig[k − 1] ... dig[0] }
  begin while k > 0 do
    begin decr(k); write(chr(ord(´0´) + dig[k]));
    end;
  end;
```

17. A nonnegative integer is printed out by *print_int*.

```
procedure print_int(n : integer);   { prints an integer in decimal form }
  var k: 0 .. 12;   { index to current digit; we assume that $0 \le n < 10^{12}$ }
  begin k ← 0;
  repeat dig[k] ← n mod 10; n ← n div 10; incr(k);
  until n = 0;
  print_digs(k);
  end;
```

18. And here is a procedure to print a nonnegative *scaled* number.

```
procedure print_scaled(s : scaled);   { prints a scaled real, truncated to four digits }
  var k: 0 .. 3;   { index to current digit of the fraction part }
  begin print_int(s div unity);   { print the integer part }
  s ← ((s mod unity) * 10000) div unity;
  for k ← 0 to 3 do
    begin dig[k] ← s mod 10; s ← s div 10;
    end;
  write(´.´); print_digs(4);
  end;
```

19. Now we're ready to print a *glue_ratio*. Since the effective multiplier is $2^{-a-b}c$, we will display the scaled integer $2^{16-a-b}c$, taking care to print something special if this quantity is terribly large.

```
procedure print_glue(g : glue_ratio);   { prints a glue multiplier }
  var d: −32 .. 31;   { the quantity 16 − a − b }
  begin d ← 32 − ga − gb;   { the amount to shift c }
  while d > 15 do
    begin write(´2x´); decr(d);   { indicate multiples of 2 for BIG cases }
    end;
  if d < 0 then print_scaled(gc div two_to_the[−d])   { shift right }
  else print_scaled(gc * two_to_the[d])   { shift left }
  end;
```

20. The driver program. In order to test these routines, we will assume that the *input* file contains a sequence of test cases, where each test case consists of the integer numbers t, x_1, …, x_n, 0; the final test case should be followed by an additional zero.

```
⟨Globals in the outer block 8⟩ +≡
x: array [1 .. 1000] of scaled;   {the $x_i$}
t: scaled;   {the desired total}
m: integer;   {the test case number}
```

21. Each case will be processed by the following routine, which assumes that t has already been read.

```
procedure test;   {processes the next data set, given $t$ and $m$}
  var n: 0 .. 1000;   {the number of items}
    k: 0 .. 1000;   {runs through the items}
    y: scaled;   {$\max_{1\le i\le n}|x_1|$}
    g: glue_ratio;   {the computed glue multiplier}
    s: scaled;   {the sum $x_1 + \cdots + x_n$}
    ts: scaled;   {the sum $f(x_1) + \cdots + f(x_n)$}
  begin write_ln('Test␣data␣set␣number␣', m : 0, ':');
  ⟨Read $x_1, \ldots, x_n$ 22⟩;
  ⟨Compute s and y 23⟩;
  if $s \le 0$ then write_ln('Invalid␣data␣(nonpositive␣sum);␣this␣set␣rejected.')
  else begin ⟨Compute g and print it 24⟩;
    ⟨Print the values of $x_i$, $f(x_i)$, and the totals 25⟩;
    end;
  end;
```

22. ⟨Read $x_1, \ldots, x_n$ 22⟩ ≡

```
  begin n ← 0;
  repeat incr(n); read(x[n]);
  until x[n] = 0;
  decr(n);
  end
```

This code is used in section 21.

23. ⟨Compute s and y 23⟩ ≡

```
  begin s ← 0; y ← 0;
  for k ← 1 to n do
    begin s ← s + x[k];
    if y < abs(x[k]) then y ← abs(x[k]);
    end;
  end
```

This code is used in section 21.

24. ⟨Compute g and print it 24⟩ ≡

```
  begin glue_fix(s, t, y, g);   {set g, perhaps print an error message}
  write('␣␣Glue␣ratio␣is␣'); print_glue(g); write_ln('␣(', ga − 16 : 0, ',', gb : 0, ',', gc : 0, ')');
  end
```

This code is used in section 21.

25. ⟨Print the values of x_i, $f(x_i)$, and the totals 25⟩ ≡

```
begin ts ← 0;
for k ← 1 to n do
  begin write(x[k] : 20);
  if x[k] ≥ 0 then  y ← glue_mult(x[k], g) else y ← −glue_mult(−x[k], g);
  write_ln(y : 15); ts ← ts + y;
  end;
write_ln('␣Totals', s : 13, ts : 15, '␣(versus␣', t : 0, ')');
end
```

This code is used in section 21.

26. Here is the main program.

```
begin initialize; m ← 1; read(t);
while t > 0 do
  begin test; incr(m); read(t);
  end;
end.
```

27. Index. Here are the section numbers where various identifiers are used in the program, and where various topics are discussed.

⟨Compute c by long division 14⟩
⟨Compute g and print it 24⟩
⟨Compute s and y 23⟩
⟨Globals in the outer block 8⟩
⟨Local variables for initialization 9⟩
⟨Normalize s, t, and y, computing a, k, and h 13⟩
⟨Print the values of x_i, $f(x_i)$, and the totals 25⟩
⟨Read x_1, ..., x_n 22⟩
⟨Set initial values 10⟩
⟨Types in the outer block 6⟩

```
17 Dec 1981   18:25        GLUE.OUT[PAS,DEK]           Page 1

Test data set number 1:
  Glue ratio is 1.1111 (0,14,18205)
                30000          33334
                40000          44445
                50000          55557
                60000          66668
 Totals        180000         200004 (versus 200000)
Test data set number 2:
  Glue ratio is 0.0111 (0,21,23302)
                30000            333
                40000            444
                50000            555
                60000            666
 Totals        180000           1998 (versus 2000)
Test data set number 3:
  Glue ratio is 71.4101 (8,0,18281)
              8000000      571281250
             -9000000     -642686836
              8000000      571281250
                 4000         274215
              7000000      499857383
 Totals      14004000     1000007262 (versus 1000000000)
Test data set number 4:
  Glue ratio is 0.0000 (8,24,15335)
              8000000             28
             -9000000            -32
              8000000             28
                 4000              0
              7000000             24
 Totals      14004000             48 (versus 100)
Test data set number 5:
  Glue ratio is 2x2x2x2x2x2x8681.0000 (-6,1,17362)
                  800      444467200
                 -900     -500025600
                  800      444467200
                  400      222233600
                  700      388908800
 Totals          1800     1000051200 (versus 1000000000)
Test data set number 6:
! Excessive glue.
  Glue ratio is 2x2x2x2x2x2x0.5000 (-6,0,1)
                  800          51200
                 -900         -57600
                  800          51200
                  400          25600
                 -700         -44800
 Totals           400          25600 (versus 1000000000)
Test data set number 7:
Invalid data (nonpositive sum); this set rejected.
Test data set number 8:
  Glue ratio is 0.0000 (1,30,11931)
                60000              0
               -59999              0
                90000              0
 Totals         90001              0 (versus 1)
```

```
17 Dec 1981   18:26        GLUE.PAS[PAS,DEK]           Page 1

{2}PROGRAM GLUE(INPUT,OUTPUT);
TYPE{6}GLUERATIO=PACKED RECORD APART:0..31;BPART:0..31;CPART:0..32768;
END;SCALED=INTEGER;VAR{8}TWOTOTHE:ARRAY[0..30]OF INTEGER;
{15}DIG:ARRAY[0..15]OF 0..9;{20}X:ARRAY[1..1000]OF SCALED;T:SCALED;
M:INTEGER;PROCEDURE INITIALIZE;VAR{9}K:1..30;BEGIN{10}TWOTOTHE[0]:=1;
FOR K:=1 TO 30 DO TWOTOTHE[K]:=TWOTOTHE[K-1]+TWOTOTHE[K-1];END;
{11}FUNCTION GLUEMULT(X:SCALED;G:GLUERATIO):INTEGER;
BEGIN IF G.APART>16 THEN X:=X DIV TWOTOTHE[G.APART-16]ELSE X:=X*TWOTOTHE
[16-G.APART];GLUEMULT:=(X*G.CPART)DIV TWOTOTHE[G.BPART];END;
{12}PROCEDURE GLUEFIX(S,T,Y:SCALED;VAR G:GLUERATIO);VAR A,B,C:INTEGER;
K,H:INTEGER;SS:INTEGER;Q,R,V:INTEGER;W:INTEGER;BEGIN{13}BEGIN A:=15;
K:=0;H:=0;SS:=S;WHILE Y<1073741824 DO BEGIN A:=A-1;Y:=Y+Y;END;
WHILE S<1073741824 DO BEGIN K:=K+1;S:=S+S;END;
WHILE T<1073741824 DO BEGIN H:=H+1;T:=T+T;END;END;
IF T<S THEN B:=15-A-K+H ELSE B:=14-A-K+H;
IF B<0 THEN BEGIN WRITELN('! Excessive glue.');B:=0;C:=1;
END ELSE BEGIN IF K>=16 THEN C:=(T DIV TWOTOTHE[H-A-B]+SS-1)DIV SS ELSE{
14}BEGIN W:=TWOTOTHE[16-K];V:=SS DIV W;Q:=T DIV V;
R:=((T MOD V)*W)-((SS MOD W)*Q);IF R>0 THEN BEGIN Q:=Q+1;R:=R-SS;
END ELSE WHILE R<=-SS DO BEGIN Q:=Q-1;R:=R+SS;END;
IF A+B+K-H=-17 THEN C:=(Q+1)DIV 2 ELSE C:=(Q+3)DIV 4;END;END;
G.APART:=A+16;G.BPART:=B;G.CPART:=C;END;
{16}PROCEDURE PRINTDIGS(K:INTEGER);BEGIN WHILE K>0 DO BEGIN K:=K-1;
WRITE(CHR(ORD('0')+DIG[K]));END;END;{17}PROCEDURE PRINTINT(N:INTEGER);
VAR K:0..12;BEGIN K:=0;REPEAT DIG[K]:=N MOD 10;N:=N DIV 10;K:=K+1;
UNTIL N=0;PRINTDIGS(K);END;{18}PROCEDURE PRINTSCALED(S:SCALED);
VAR K:0..3;BEGIN PRINTINT(S DIV 65536);
S:=((S MOD 65536)*10000)DIV 65536;
FOR K:=0 TO 3 DO BEGIN DIG[K]:=S MOD 10;S:=S DIV 10;END;WRITE('.');
PRINTDIGS(4);END;{19}PROCEDURE PRINTGLUE(G:GLUERATIO);VAR D:-32..31;
BEGIN D:=32-G.APART-G.BPART;WHILE D>15 DO BEGIN WRITE('2x');D:=D-1;END;
IF D<0 THEN PRINTSCALED(G.CPART DIV TWOTOTHE[-D])ELSE PRINTSCALED(G.
CPART*TWOTOTHE[D])END;{21}PROCEDURE TEST;VAR N:0..1000;K:0..1000;
Y:SCALED;G:GLUERATIO;S:SCALED;TS:SCALED;
BEGIN WRITELN('Test data set number ',M:0,':');{22}BEGIN N:=0;
REPEAT N:=N+1;READ(X[N]);UNTIL X[N]=0;N:=N-1;END;{23}BEGIN S:=0;Y:=0;
FOR K:=1 TO N DO BEGIN S:=S+X[K];IF Y<ABS(X[K])THEN Y:=ABS(X[K]);END;
END;IF S<=0 THEN WRITELN(
'Invalid data (nonpositive sum); this set rejected.')ELSE BEGIN{24}BEGIN
GLUEFIX(S,T,Y,G);WRITE('  Glue ratio is ');PRINTGLUE(G);
WRITELN(' (',G.APART-16:0,',',G.BPART:0,',',G.CPART:0,')');END;
{25}BEGIN TS:=0;FOR K:=1 TO N DO BEGIN WRITE(X[K]:20);
IF X[K]>=0 THEN Y:=GLUEMULT(X[K],G)ELSE Y:=-GLUEMULT(-X[K],G);
WRITELN(Y:15);TS:=TS+Y;END;
WRITELN(' Totals',S:13,TS:15,' (versus ',T:0,')');END;END;END;
{26}BEGIN INITIALIZE;M:=1;READ(T);WHILE T>0 DO BEGIN TEST;M:=M+1;
READ(T);END;END.
```

```
17 Dec 1981   18:28        GLUE.TEX[PAS,DEK]           Page 6

\M12.  Glue setting.
The \\{glue\_fix} procedure computes $a$, $b$, and $c$ by the method explained
above. \TEX\ does not normally compute the quantity $y$, but
it would not be difficult to make it do so.

This procedure would be a function that returns a \\{glue\_ratio}, if \PASCAL\
would allow functions to produce records as values.

\Y\P\4\&{procedure}\1\  \37$\\{glue\_fix}(\\s,\39\\t,\39\\y:\\{scaled};\,\3
5\mathop{\&{var}}\\g:\\{glue\_ratio})$;\6
\4\&{var} \37$\\a,\39\\b,\39\\c$: \37\\{integer};\C{components of the desired
ratio}\6
$\\k,\39\\h$: \37\\{integer};\C{$30-\lfloor\lg s\rfloor$, $30-\lfloor\lg
t\rfloor$}\6
\\{ss}: \37\\{integer};\C{original (unnormalized) value of $s$}\6
$\\q,\39\\r,\39\\v$: \37\\{integer};\C{quotient, remainder, divisor}\6
\\w: \37\\{integer};\C{$2↑1$}\2\6
\&{begin} \37\X13:Normalize $s$, $t$, and $y$, computing $a$, $k$, and $h$\X;\6
\&{if} $\\t<\\s$ \1\&{then}\5
$\\b\K15-\\a-\\k+\\h$\ \&{else} $\\b\K14-\\a-\\k+\\h$;\2\6
\&{if} $\\b<0$ \1\&{then}\6
\&{begin} \37$\\{write\_ln}(\.{\'!\ Excessive\ glue.\'})$;\C{error message}\6
$\\b\K0$;\5
$\\c\K1$;\C{make $f(x)=\lfloor2↑{-a}x\rfloor$}\6
\&{end}\6
\4\&{else} \&{begin} \37\&{if} $\\k\G16$ \1\&{then}\C{easy case, $s<2↑{16}$}\6
$\\c\K(\\t\mathbin{\&{div}}\\{two\_to\_the}[\\h-\\a-\\b]+\\{ss}-1)\mathbin{\&
{div}}\\{ss}$\6
\4\&{else} \X14:Compute \\c by long division\X;\2\6
\&{end};\2\6
$\\{ga}\K\\a+16$;\5
$\\{gb}\K\\b$;\5
$\\{gc}\K\\c$;\6
\&{end};\par

\M13. \P$\X13:Normalize $s$, $t$, and $y$, computing $a$, $k$, and $h$\X\S$\6
\&{begin} \37$\\a\K15$;\5
$\\k\K0$;\5
$\\h\K0$;\5
$\\{ss}\K\\s$;\6
\&{while} $\\y<\01000000000$ \1\&{do}\C{\\y is known to be positive}\6
\&{begin} \37$\\{decr}(\\a)$;\5
$\\y\K\\y+\\y$;\6
\&{end};\2\6
\&{while} $\\s<\01000000000$ \1\&{do}\C{\\s is known to be positive}\6
\&{begin} \37$\\{incr}(\\k)$;\5
$\\s\K\\s+\\s$;\6
\&{end};\2\6
\&{while} $\\t<\01000000000$ \1\&{do}\C{\\t is known to be positive}\6
\&{begin} \37$\\{incr}(\\h)$;\5
$\\t\K\\t+\\t$;\6
\&{end};\2\6
\&{end}\par
\U section 12.

\M14. \P$\X14:Compute \\c by long division\X\S$\6
\&{begin} \37$\\w\K\\{two\_to\_the}[16-\\k]$;\6
$\\v\K\\{ss}\mathbin{\&{div}}\\w$;\5
$\\q\K\\t\mathbin{\&{div}}\\v$;\5
$\\r\K((\\t\mathbin{\&{mod}}\\v)\ast\\w)-((\\{ss}\mathbin{\&{mod}}\\w)\ast\\
q)$;\6
\&{if} $\\r>0$ \1\&{then}\6
\&{begin} \37$\\{incr}(\\q)$;\5
$\\r\K\\r-\\{ss}$;\6
\&{end}\6
\4\&{else} \&{while} $\\r\L-\\{ss}$ \1\&{do}\6
\&{begin} \37$\\{decr}(\\q)$;\5
```

```
17 Dec 1981   18:28        GLUE.TEX[PAS,DEK]            Page 8-2

$\\r\K\\r+\\{ss}$;\6
\&{end};\2\2\6
\&{if} $\\a+\\b+\\k-\\h=-17$ \1\&{then}\6
$\\c\K(\\q+1)\mathbin{\&{div}}2$\C{$1=16+k-h$}\6
\4\&{else} $\\c\K(\\q+3)\mathbin{\&{div}}4$;\2\6
\&{end}\par
\U section 12.
```

```
17 Dec 1981   18:26        GLUE.WEB[PAS,DEK]          Page 6

@* Glue setting.
The |glue_fix| procedure computes $a$, $b$, and $c$ by the method explained
above. \TEX\ does not normally compute the quantity $y$, but
it would not be difficult to make it do so.

This procedure would be a function that returns a |glue_ratio|, if \PASCAL\
would allow functions to produce records as values.

@p procedure glue_fix(@!s,@!t,@!y:scaled;var@!g:glue_ratio);
var@!a,@!b,@!c:integer; {components of the desired ratio}
@!k,@!h:integer; {$30-\lfloor\lg s\rfloor$, $30-\lfloor\lg t\rfloor$}
@!ss:integer; {original (unnormalized) value of $s$}
@!q,@!r,@!v:integer; {quotient, remainder, divisor}
@!w:integer; {$2↑1$}
begin @<Normalize $s$, $t$, and $y$, computing $a$, $k$, and $h$@>;
if t<s then b←15-a-k+h@+else b←14-a-k+h;
if b<0 then
        begin write_ln('! Excessive glue.'); {error message}
        b←0; c←1; {make $f(x)=\lfloor2↑{-a}x\rfloor$}
        end
else    begin if k≥16 then {easy case, $s<2↑{15}$}
        c←(t div two_to_the[h-a-b]+ss-1) div ss
        else @<Compute |c| by long division@>;
        end;
ga←a+16; gb←b; gc←c;
end;

@ @<Normalize $s$...@>=
begin a←15; k←0; h←0; ss←s;
while y<@'10000000000 do {|y| is known to be positive}
        begin decr(a); y←y+y;
        end;
while s<@'10000000000 do {|s| is known to be positive}
        begin incr(k); s←s+s;
        end;
while t<@'10000000000 do {|t| is known to be positive}
        begin incr(h); t←t+t;
        end;
end

@ @<Compute |c|...@>=
begin w←two_to_the[16-k]; v←ss div w; q←t div v;
r←((t mod v)*w)-((ss mod w)*q);
if r>0 then
        begin incr(q); r←r-ss;
        end
else while r≤-ss do
        begin decr(q); r←r+ss;
        end;
if a+b+k-h=-17 then c←(q+1) div 2 {$1=16+k-h$}
else c←(q+3) div 4;
end
```