21st Annual
Rowan University
Programming Contest

hosted by the

Computer Science Department

Friday, 27 April 2007

Contest Problem

Rowan University

1 Background

Many computer systems that handle text, such as editors, web browsers, and word processors, do automatic word-wrapping. Typesetters may also hyphenate words and adjust the inter-word space width to make a neat right margin. Early computer systems used letters which were all the same width, but most modern systems have to deal with letters of different widths. Wrapping correctly with variable-width characters requires knowing how wide each letter is. The word ‘momentum’ is 67% wider than ‘trillion’, even though both are eight letters long. But knowing the widths of the letters is not enough, however. Other considerations include *kerning* and *ligatures*.

Kerning is adjusting the space between letters to improve the appearance of a grouping. For example, ‘VA’ should be set closer together than either ‘VV’ or ‘AA’, because the sides of the letters slope in the same direction. In some fonts, pairs such as ‘rn’ have to be spaced out so they won’t look like an ‘m’. *Ligatures* are situations in which two letters are joined; in ‘Cæsar’, the letters ‘a’ and ‘e’ are connected. Some ligatures are nearly invisible because they are so familiar; ‘f’ is often connected to ‘i’, such that the dot over the ‘i’ is replaced by the ball on the end of the ‘f’. Here are some examples in larger print; the first line has kerning and ligatures on, and the second has them off:

Venus	AWACS	aﬃx	ﬂuid
Venus	AWACS	affix	fluid

In the word ‘Venus’, the large gap which results between the ‘V’ and the ‘e’ on the second line, and the smaller but still noticeable gap between the ‘n’ and the ‘u’, may be distracting. Similarly, the acronym ‘AWACS’ appears strangely spaced out. When magnified as above, it is easy to see the ligatures in the two rightmost words; but most readers of this document probably didn’t notice anything about the word ‘diﬀerent’ in the ﬁrst paragraph of this page, or the word ‘ﬁrst’ in the second.

In order to do word wrapping properly, then, the system must also take into account kerning and ligatures. Therefore, it not only has to know how wide each letter is, it must also know what special groupings exist and how wide those groupings are.

Your challenge for this contest is to write a program which reads in a list of characters and their widths, and then a list of groupings which (due to either kerning or ligature) have a different width than the letters would have by themselves. Then the program will read in a number of lines of text. The output will be the length of each line of text, if typeset using the characters and kerning/ligature adjustments as defined.

All measurements will be in ‘points’, as that is the standard unit used in typesetting. To keep matters simple, the problem requires precision only to 0.1 points.

2 Input

2.1 Input Specification

For text input, your program should accept input in the following format:

1. An integer, $\mathcal{N}$, where $\mathcal{N} \geq 1$, the number of data sets in this file.
2. $\mathcal{N}$ data sets, each of which is in this format:
   (a) An integer, $\mathcal{S}$, where $1 \leq \mathcal{S} \leq 256$, the number of symbols in this data set.
   (b) $\mathcal{S}$ lines, each with one character, a space, and the character's width in points.
   (c) An integer, $\mathcal{G}$, where $1 \leq \mathcal{G} \leq 128$, the number of special character groupings.
   (d) $\mathcal{G}$ lines (one per grouping), each with the characters in the grouping, a space, and the number of points that grouping's width **is to be adjusted**.
   (e) An integer $\mathcal{L}$, the number of text lines whose printed with must be found.
   (f) $\mathcal{L}$ lines of text, whose typeset width your program will need to print. Each of these lines will have less than 80 characters. Except for a space character, no line will have any character not specified in the data set's alphabet.

2.2 Sample Input #1

Data	Item #	Meaning
in file	above:	in plain English
1	1	*this file has 1 data set*
5	2a	*data set 1 has 5 characters*
V 8.8	2b	*the 5 characters in this data set and their widths*
e 5.2		
n 6.5		
u 6.5		
s 4.6		
2	2c	*data set 1 has 2 special groupings*
Ve -1.0	2d	*the 2 special groups in this data set and their width adjustments*
nu -0.3		
3	2e	*data set 1 has 3 lines of text*
Ve	2f	*the 3 lines whose width is to be found*
nu		
Venus		

(This input will be available on the website as **sample1.txt**.)

Unless they are part of a special grouping, the spacing between letters is 0.0 points; that is, the width of the letter includes a little bit of space on each side. Between two words, you may assume a space of 4.0 points.

3 Output

3.1 Output Specification

For each data set configuration, your program must generate output as follows:

1. The text '`Data Set N:`', where **N** is the number of the data set being reported on.
2. The text '`Characters: C:`', where **C** is the number of symbols in this set's alphabet.
3. The text '`Groupings: G:`', where **G** is the number of special groupings in this set.
4. The text '`Lines: L:`', where **L** is the number of lines whose width is to be calculated.
5. For each line of text, you should print '`Line #: W points`', where # is what line number this is, and **W** is the width in points of that line when it has been typeset.

3.2 Sample Output #1

Output Generated by Program	Item # above:	Meaning in plain English
`Data Set 1:`	1	*this is for the first data set*
`Characters: 5`	2	*data set 1 had 5 characters*
`Groupings: 2`	3	*data set 1 had 2 special-width groupings*
`Lines: 3`	4	*data set 1 had 3 lines whose width was to be found*

`Line 1:  13.0 points`
`Line 2:  12.7 points`
`Line 3:  30.3 points`
	5	*the lines and their widths when typeset*

NOTES:

You need not do error-checking on the input. Each line described as having a single integer in a given range will have exactly one integer on it. The lines giving character widths will have exactly one character, one space, and one floating-point number. The lines giving group widths will have the characters in the grouping, one space, and one floating-point number. No width specification will include a character not in that data set's alphabet.

You may choose to have your program read the input from the keyboard, or ask the user for a filename and then read the file. Users of GUI-based programming environments may prefer to use text boxes into which the values can be entered, and buttons to begin their calculation. Any reasonable variation in the spirit of the problem is acceptable.

Your output **does not** have to match the sample output exactly as regards use of upper/lower case or precise spacing. It should be neat and readable, but does *not* have to match exactly.

4 Sample Data

4.1 Sample Input #2

```
3            this file has 3 data sets
4            data set 1 has 4 characters
A 8.8
W 12.1       the 4 characters and their
C 8.5        widths
S 6.5
3            data set 1 has 3 special groupings
AW -1.3
WA -1.3      grouping width adjustments
AC -0.3
3            data set 1 has 3 lines of text
AWACS        the lines whose width is to
SAW          be found
A W A
4            data set 2 has 4 characters
a 5.9
f 3.6        the 4 characters and their
i 3.3        widths
x 6.2
3            data set 2 has 3 special groupings
ff -0.3
fi -0.3      grouping width adjustments
ffi -0.5
3            data set 2 has 3 lines of text
affix        the lines whose width is to
fix          be found
if
5            data set 3 has 5 characters
d 6.5
f 3.6
i 3.3        the 5 characters and their
l 3.3        widths
u 6.5
3            data set 3 has 3 special groupings
fl -0.3
fi -0.3      grouping width adjustments
ff -0.3
5            data set 3 has 5 lines of text
fluid
fulfill
fluff        the lines whose width is to
lid          be found
dull
```

4.2 Sample Output #2

```
Data Set 1:
  Characters: 4
  Groupings: 3
  Lines: 3
    Line 1: 41.8 points
    Line 2: 26.1 points
    Line 3: 37.7 points
Data Set 2:
  Characters: 4
  Groupings: 3
  Lines: 3
    Line 1: 22.1 points
    Line 2: 12.8 points
    Line 3: 6.9 points
Data Set 3:
  Characters: 5
  Groupings: 3
  Lines: 5
    Line 1: 22.9 points
    Line 2: 26.6 points
    Line 3: 20.0 points
    Line 4: 13.1 points
    Line 5: 19.6 points
```

NOTES:

Data Set 1, Line 1: kerning adjustments have been applied on both sides of the 'W' and the second 'A'. If adjustments exist for both sides of a character, both must be applied.

Data Set 2, Line 1: the width for 'affix' uses the grouping for 'ffi', **not** the groupings for 'ff' and 'fi'. If a set of characters matches multiple groupings, use the longer one.

4.3 Sample Input #3

```
2                this file has 2 data sets
9                data set 1 has 9 characters
A 8.8
B 8.3
C 8.5
D 9.0
E 8.0            the widths of the characters
F 7.7            A-I
G 9.2
H 8.8
I 4.2
1                data set 1 has 1 special grouping
AC -0.3          adjustment for 'AC'
5                data set 1 has 5 lines of text
AAAAAAAA
BBBBBBBB
CCCCCCCC         the lines whose width is to
HI ABC DIE       be found
II BB BED
4                data set 2 has 4 characters
A 8.8
V 8.8            the widths of the 4
T 8.5            characters
L 7.3
8                data set 2 has 8 special groupings
AT -1.0
TA -1.0
AV -1.3
VA -1.3          adjustments for the
LT -1.0          6 groupings
LV -1.3
LTA -1.9
TAT -1.8
6                data set 2 has 6 lines of text
TTAT AAVV
TV LA
AVA TAV          the lines whose width is to
LVAT             be found
LTAVAVAT
ALTAT
```

4.4 Sample Output #3

```
Data Set 1:
  Characters: 9
  Groupings: 1
  Lines: 5
    Line 1 : 70.4 points
    Line 2 : 66.4 points
    Line 3 : 68.0 points
    Line 4 : 67.8 points
    Line 5 : 58.3 points
Data Set 2:
  Characters: 4
  Groupings: 8
  Lines: 6
    Line 1: 70.4 points
    Line 2: 37.4 points
    Line 3: 51.6 points
    Line 4: 29.8 points
    Line 5: 60.2 points
    Line 6: 39.0 points
```

NOTES:

Data Set 2, Line 6: This line could be grouped as A-LT-TAT, or as A-LTA-AT. Note that two different lengths will result depending on which version you choose. In case of such a conflict, apply the one which has the longest applicable correction in the first position. For this example, the program did A-LTA-AT, because LTA is longer than LT.

These data sets are available at <http://elvis.rowan.edu/hspc/2007> as **sample2.txt** and **sample3.txt**. (The file **sample1.txt** is the data from §2.2 on page 3.) The test data on the facing page is at the same URL, under the names **test1.txt** and **test2.txt**.

5 Test Data

Run your program on this input and print the results. **You must submit printed output for this input to earn full points.** Your program will also be run on secret data known only to the judges.

5.1 Test Input #1

```
1
13
R 8.8
o 6.4
w 8.4
a 5.4
n 6.6
U 9.4
i 3.4
v 5.5
e 5.6
r 4.4
s 4.6
t 4.4
y 5.5
13
Ro -0.5
Un -0.7
rw 0.5
Ur -0.7
as 0.1
ov 0.1
rn 0.1
rt 0.1
ry 0.5
te -0.2
to -0.2
we -0.1
wo -0.1
16
Rowan University
airway anniversary aorta
assertive eastern entertainer
eyestrain inerrant innovative
insane intense irony nanny
overstate raisin yesteryear
reiteration resistant roast
sanitation sanity sensitivity
serenity sonneteer sorority
stairway stationary stationery
stony tannin tasty variation
tertiary titration torrent Ursa
transverse tyrant Unitarian
vanity waitress warrior weariness
venostasis veterinarian visionary
witness woven yarn yeast yore
```

5.2 Test Input #2

```
2
9
H 10.4
o 6.4
e 5.6
f 4.0
l 3.1
r 4.5
T 8.5
x 5.8
t 4.4
8
Ho -0.4
fl -0.1
Te -1.5
ff -0.6
ll 0.1
rt 0.1
te -0.2
to -0.2
4
Hoefler Text
effort extort ferret fetter flexor offer
forteller letter referrer reflex retort
text troll
5
V 9.2
e 5.6
n 6.6
u 6.3
s 4.6
4
Ve -1.9
nu -0.0
su 0.0
ue -0.0
6
Ve
nu
Venus
ensue
unseen
sense
```

6 Notes (not required to solve the problem)

This problem just touches on the complexity of typesetting systems. Beyond what is described here, words may be hyphenated, and once the optimum set of words and word portions is chosen, the spaces may be compressed or expanded slightly to get the line length exactly right. Doing this may involve moving words between lines. Consider:

Here's a sentence that needs to be word wrapped in order to fit in this box.	Here's a sentence that needs to be word wrapped in order to fit in this box.

Both boxes are the same size, but adjusting the inter-word spaces changes which line has the word 'be'. In some cases, such as when there are many long words which the system doesn't know how to hyphenate, the results may look strange. This can often be improved by setting the text to 'ragged-right', or by adding a hyphenation rule which tells the system how to hyphenate a word it doesn't know.

Hyphenation is itself a complex problem: in some cases, adding a suffix changes where a hyphen may appear in a word's stem; compare 'rec-ord' and 're-cord-able'. One solution, discovered by Frank Liang, is explained in Appendix H of *The TEXbook*, by Donald E. Knuth. Still, there can be problematic cases, such as certain German words which change their spelling when broken across lines. For example, the word 'bachen' becomes 'bak-ken'.

This problem description was typeset using LATEX, which was built as an extension to Don Knuth's TEX system. Knuth is the author of *The Art of Computer Programming*, having completed four of an intended seven volumes. He created TEX partly to solve his own problem, which was typesetting his book. TEX was designed to make possible (though not always easy) certain difficult elements of typesetting, especially mathematical formulæ, while maintaining a visually pleasing result. The formula for a normal distribution is:

$$y = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}$$

Obviously, typesetting such formulæ is far more complex than tasks such as word-wrapping.

Most widths and kerning/ligature adjustments used in the sample input are for 12-point Computer Modern (in which this problem is set), and most values for the test data are for 12-point Hoefler Text, all rounded to the nearest tenth. (Some were just made up.)

A *point*, the standard measure of printing, is not actually standardized. The American Typefounders Association defined it as 0.013837 inches in 1886; that works out to $72.27\overline{00007227}$ points per inch. PostScript (a programming language used by many computer printers) defines a point as 1/72 of an inch, which has become standard for desktop publishing. TEX sets its point size at 72.27 per inch, closer to the 1886 definition than PostScript, but still relatively easy for a computer to manage (divided into 65,536 subunits for internal calculations). A 12-point upper-case Computer Modern A is 8.80824 points wide, somewhat more precision than made sense for this problem, hence rounding to nearest tenths.