Sample Programming Contest Problems
P. N. Hilfinger

Balancing Chemical Equations

Write a program that, given an equation such as

_H2O+_CO2 = _O2+_C6H12O6.

will fill in the blanks to produce a balanced equation, such as

6H2O + 6CO2 = 6O2 + C6H12O6,

More specifically, an equation is input in the form of a sequence of lines, one for each molecule (e.g., H2O or CO2) in the formula. The k^{th} line has the following form.

$sign_k\ n_k\ atom_{k,1}\ count_{k,1}\ \ldots\ atom_{k,n}\ count_{k,n}$

where $sign_k$, n_k, $atom_{k,j}$, and $count_{k,j}$ are two-character fields separated by single blanks; $sign_k$ is ±1; each $atom_{k,j}$ consists of two characters, the first of which must be alphabetic, and the second of which may be alphabetic or blank; and each $count_{k,j}$ is a positive integer. After all the lines describing molecules, there is a delimiting line of the form

00 00

If there are m input lines, the problem is find m positive integer values, $C_1, \ldots, C_m$, such that for every distinct two-character atom name, a, appearing in the input,

$$\sum sign_i \cdot C_i \cdot count_{i,j} = 0,$$

where the sum is over all i and j for which $atom_{i,j} = a$.

For example, if part of the input is ‘‘N 2’’, this represents two atoms of N (nitrogen). If it occurs on a line (molecule) with C_i equal to 3, it represents 6 atoms. A *sign* of +1 indicates that this molecule appears on the left of an equation, and -1 indicates that it appears on the right. The problem, then, is to get the result of adding up the number of atoms of N with a +1 *sign* and subtracting the number of atoms with -1 *sign* so as to get a total of 0. Further, the C_i must be chosen so that this happens for all atoms simultaneously. Note that an atom may be repeated in a given line of input, as in

+1 6 C 1 H 5 C 1 O 1 O 1 H 1,

which stands for CH_5COOH.

The input for the illustrative equation at the beginning of this problem is

```
+1 2 H 2 O 1
+1 2 C 1 O 2
-1 1 O 2
-1 3 C 6 H 12 O 6
00 00
```

and the output is to be as shown in that example.

HINT: There are ways to do this right, but for the purposes of this contest, you may assume that exhaustive search is an acceptable way of finding the C_i. Assume that the C_i are bounded above by 10.

Macro Processor

Write a program to expand calls on textual macros in a body of text. The input will consist of a sequence of lines of text, each 80 characters long. Your program must find instances of

macro-name

or

macro-name (*argument*$_1$,...,*argument*$_n$)

outside of quoted strings in the text, where *macro-name* is the name of a macro that has been declared. The program must replace each such instance with the appropriate expansion declared for *macro-name* and then treat the resulting text as if it had been the original input (in particular, any macro calls in the resulting text must themselves be expanded.)

A *macro-name* is any string of letters and digits beginning with a letter. Macro names are recognized only when surrounded by characters other than letters and digits, so that, for example, if Scan is a macro, it does *not* get replaced in the identifier ScanFirst. Case is significant in macro names; X is different from x. Each of the *argument*$_i$ is a sequence of characters that does not include commas or right parentheses. The left parenthesis on an argument list must follow the macro name immediately; if there is a parenthesis, there are assumed to be arguments, and otherwise, there are assumed to be none.

A declaration of a macro consists of a call on a pre-defined pseudo-macro, DefineMacro. An appearance of

DefineMacro(*macro-name*,"*text*")

defines a macro. The *text* between the quotation marks can contain any characters other than quotation mark. When the macro *macro-name* is called, this text replaces the call. Each instance of *n*\ in the text of the macro, where *n* is a positive integer, is replaced by the n^{th} argument to the macro call (or with nothing if there is no n^{th} argument.)

Ends of lines in macro calls (including DefineMacro) are replaced by blanks. The lines of text are output in 80 character lines. If macro expansion makes a line longer than 80 characters, it is to be continued on a new line. If macro expansion makes a line shorter than 80 characters, it is to be padded with blanks. Other line breaks in the input text are to be repeated in the output.

Text Formatter

Write a program to fill and justify lines of text. The input will be in the form of 80 characters records. Ordinary text on these lines is to be transferred to the output in such a way that as many words as possible are put on a line as will fit in the current line length, moving words to the next line if a line is too long and from a previous line if it is too short. For the purposes of this problem, a *word* is a sequence of non-blank characters (including letters, digits, and punctuation.) Extra blanks are to be inserted between words on all lines of a paragraph except the last so that the text of each line is aligned at the right margin.

In the input, ends of paragraphs are marked by blank lines or by any of the commands described below. The first line of each paragraph is indented by an amount called the *paragraph indentation*. Otherwise, both blanks and blank lines in the input are ignored (for example, putting five blank lines after a paragraph produces the same output as putting one.) In the output, paragraphs are separated from each other by some number of blank lines (0 or more) called the *paragraph separation.*

The input may contain *commands*, which are not printed, but which affect the formating of subsequent text. If any of these immediately follow text (as opposed to another command or a blank line), then they mark the end of a paragraph. Each command must begin in the first column of an input line and nothing must follow the command on that line. Commands begin with a period. Any line that begins with a period but does not contain a valid command is an error and an error message should be inserted into the output text at that point.

The valid commands are

.lm *n*

Set the left margin (the position of the first character output on a line of output) to the n^{th} column, $n>0$.

.rm *n*

Set the right margin (the position of the last character output on a line of output) to the n^{th} column, $n>1$.

.pm *n*

Set the paragraph margin to column n, $n>0$.

Pattern Recognition

Write a program to find and report all instances of a given set of patterns in some sample data. The data (sample and patterns) are in the form of rectangular arrays of single-digit numbers. Each rectangle has the format

m n
$d_{11}...d_{1n}$
. . .
$d_{m1}...d_{mn}$

where *n* and *m* are integers between 1 and 100, inclusive. The entire input to your program will have the following form.

pattern 1
. . .
pattern k
0 0
sample data

There may be up to 100 patterns.

An *m* by *n* pattern is said to *match* a certain subrectangle in the sample data if there is *some orientation* of the pattern (i.e., top edge up, left edge up, right edge up, or bottom edge up) that can be overlaid on the subrectangle such that

> If a non-zero digit, d_1, in the pattern overlays a given digit, d'_1, in the sample data and another non-zero digit, d_2, in the pattern overlays d'_2 in the sample data (d_1 may, but need not, equal d_2), then $d_1-d_2=d'_1-d'_2$.

Zero digits in the pattern are ignored.

Consider the following input, consisting of two patterns and sample data.

2 2
01
20
2 2
22
02
0 0
4 4
1050
0200
3341
1331

The output might be as follows (comments in italics.)

Pattern 1 matches at (1,1) (Turned on left side)
Pattern 2 matches at (1,3) (Turned on right side)
Pattern 1 matches at (2,1)

Pattern 1 matches at (2,3) (Turned on left side)
Pattern 2 matches at (3,1)
Pattern 1 matches at (3,2) (Turned upside down)
Pattern 2 matches at (3,2) (Turned upside down)

Digital Signal Processing

Write a program that accepts a description of a network of adders, multipliers, and delays and a sequence of values resulting from sampling a signal, and produces the result of putting the signal through the network.

The description of the network consists of a sequence of *nodes*, each described by one line of input in the form

label operation operand operand

where *label* is a unique positive integer label; *operation* is one of the characters '+' (add), '-' (subtract), '*' (multiply), and 'D' (delay); and the *operands* are either real numbers or references to nodes of the form

N*non-negative-integer*

Each of the fields (label, operation, and each operand) is separated from the others by a blank. The input to the program has the form

node
node
. . .
node
0
sample data: a sequence of real numbers on one or more lines separated by blanks.

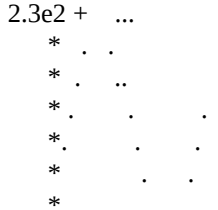
To perform the computation, for each sample value, compute the value of each node. The value of a node is obtained by performing the indicated operation on the two operands. An operand of the form N*p* refers to the value of the node labeled *p*. A delay node (with a 'D' operator) takes only one argument, which must be a node reference. It represents the value of the referenced node on the *previous* sample value. Operand references to node 0 refer to the current sample input value. The value computed for node 1 for each sample value is the output of the network for that sample.

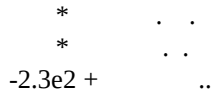
The nodes need not be written in order of increasing labels. You may assume an upper limit of 100 on the value of a node label. The nodes *need not be written* in the order their values must be computed; for example, in

1 * N2 3.5
2 + N3 N4

The value of node 2 must be computed before computing that of node 1. Your program must compute the values of the nodes in an order that assures that each operand's value has been computed before it is used. It should also detect circular node structures (a cycle of nodes unbroken by a delay.)

The output from your program should be in the form of rough graphs of the input and output. For example,





Allow ten divisions vertically, choosing the scale so that all points fit on it, and indicating the maximum and minimum values, as shown.