UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Hacking Contest **P. N. Hilfinger**
Fall 1991

Programming Problems

You have 5 hours in which to solve as many of the following problems as possible. Put each complete solution into a single `.c` file. To submit a solution, use the command

```
    submit N foo.c
```

where *foo*`.c` is the name of your program and N is the problem number (1–6). `Submit` will first compile your program in a scratch subdirectory using the command

```
        /usr/custom/gcc -O foo.c -lm
```

then run it on some test data, compare the results against a standard, and actually record your program only if the results compare equal and the program runs in less than two minutes of execution time. If your program is rejected for producing incorrect output, `submit` will display your output and the desired output. WARNING: The test data for a given problem may vary from one running of `submit` to the next. File $MASTER/examples/$N$.in contains a sample set of test data for problem N, with output in $MASTER/examples/$N$.out

All input will be placed in `stdin`, and all output must be produced on `stdout`. Anything written to `stderr` will be ignored. Your program must exit with a status code of 0. Because the testing of programs is automatic, you must be careful to observe appropriate use of whitespace. Do not use tabs in place of blanks. Make sure that the last line of output ends with a newline.

Scoring. Scoring will be according to the ACM Contest Rules. You will be ranked by the number of problems solved. Where two or more contestants complete the same number of problems, they will be ranked by the *total time* required for the problems solved. The total time is defined as the sum of the *time consumed* for each of the problems solved. The time consumed on a problem is the time elapsed between the start of the contest and successful submission, plus 20 minutes for each unsuccessful submission. Unsuccessful submissions of problems that are not solved do not count. As a matter of strategy, you can derive from these rules that it is best to work on the problems in order of increasing expected completion time.

1. Given a non-negative number N as input, print the position of N, $P(N) \geq 0$, in the numerically-ordered sequence of all non-negative numbers with the same number of 1-bits as N in binary representation. Inputs will be non-negative decimal numbers less than 2^{16}, entered free-form on the standard input. For each input number, your program must output one line of the form

$\dashv N$ `is number` $P(N)$ `in the sequence of numbers with` k `1-bits.`

where N is replaced by the input number, $P(N)$ by the appropriate position number, and k by the number of 1-bits in N. The character '$\dashv$' marks the left margin. All numbers must be in decimal notation with no leading 0's.

For example, if $N = 9$, then the output should be

```
⊣9 is number 3 in the sequence of numbers with 2 1-bits.
```

This is because the sequence of numbers with exactly two 1-bits is 3 (11_2), 5 (101_2), 6 (110_2), and 9 (1001_2); counting the first as number 0, 9 is number 3 in this sequence. ($\dashv$ marks the left margin.)

2. Given a sequence of words of input (in free-form, separated from each other by whitespace), print all words found in the file `/usr/dict/words` that can be formed using some or all of the letters in each of the words in turn. That is, each of the output words produced for a given input word must contain only characters that appear in the input word, and may not contain any given character more times than it appears in the input word. For each input word, the output has the following format.

$\dashv$`Word:` W
$\dashv$`Forms:` X_1`,` X_2`, ...,` X_n

where the Ws and Xs are the strings ($\dashv$ marks the left margin). If no anagrams are found, the second line should read

```
Forms: *none*
```

Assume that case of letters is *not* significant. Output words should be printed in the order they appear in `/usr/dict/words`.

For example, if the input is

```
tab
```

the output will be

```
Word: tab
Forms: a, at, b, bat, t, TA, tab
```

3. Given a sequence of one or more input lines, each of which contains N strings separated by ampersand (&) characters, output the strings in tabular form, centered in columns. That is, the input has the form

$$\begin{array}{l} S_{11}\&S_{12}\&\cdots\&S_{1m} \\ S_{21}\&S_{22}\&\cdots\&S_{2m} \\ \qquad\quad\cdots \\ S_{n1}\&S_{n2}\&\cdots\&S_{nm} \end{array}$$

where the S_{ij} are strings (possibly empty) containing no ampersands or newlines. There is either a null line or the end-of-file after the newline character terminating the last input line. Any whitespace at the beginning or end of an S_{ij} is to be removed. The output is to have the following form ('⊣' marks the left margin).

$$\begin{array}{l} \dashv S_{11} \;|\; S_{12} \;|\cdots|\; S_{1m} \\ \dashv S_{21} \;|\; S_{22} \;|\cdots|\; S_{2m} \\ \qquad\quad\cdots \\ \dashv S_{n1} \;|\; S_{n2} \;|\cdots|\; S_{nm} \end{array}$$

Each of the m columns is to be two characters wider than the widest string in that column of the input, and each S_{ij} is to appear centered in its column. If perfect centering would require adding half a space before and after S_{ij}, add a whole space before it instead. It is a consequence of these rules that the first column of output is blank and that there is at least one blank between each string and any vertical bar delimiting its column. There should be no trailing blanks. You may assume that $0 < m < 28$ and $0 < n < 60$.

4. Given a positive integer, $K \leq 250,000$, compute the K^{th} prime (for $K = 1$, this is 2; for $K = 2$, it is 3, etc.). The input is a single free-form integer. The output has the format

```
⊣Prime K is P.
```

where K is replaced by the input and P by the prime. Note: the $250,000^{\text{th}}$ prime is less than 3,500,000.

As usual, your program must complete in less than two minutes of execution time. In addition, it must use less than 275 Kbytes of data area (i.e., the program must be runnable with

```
limit datasize 275k
```

in effect).

5. An instance of the Bounded Post's Correspondence Problem consists of a positive integer K and two sequences of non-empty strings: $x_1, x_2, \ldots, x_m$ and $w_1, w_2, \ldots, w_m$, with $m > 0$. The problem is to find a sequence of integers in the range 1 to m—$i_1, i_2, \ldots, i_k$—with $1 \leq k \leq K$ (and possibly with repetitions) such that

$$x_{i_1} \cdot x_{i_2} \cdots x_{i_k} = w_{i_1} \cdot w_{i_2} \cdots w_{i_k}.$$

Here, '$\cdot$' denotes string concatenation.

For example, if $K \geq 5$, the x's are (a, bbc, ba, b), and the w's are (ab, b, ac, cb), then a solution is $i_1 = 1$, $i_2 = 2$, and $i_3 = 4$, because

$$a \cdot bbc \cdot b = ab \cdot b \cdot cb.$$

If $K = 2$, with the same x and w, then there is no solution.

There will be one or more sets of input to your program. Each set will consist of a positive integer (for K) followed by a sequence of strings separated by blanks (and themselves containing no whitespace), a newline character (which may but need not be preceded by blanks and tabs), and another sequence of the same number of strings. Within these limitations, input is free-form—K will be separated from the first string by arbitrary whitespace, and the sets of input are separated from each other by arbitrary whitespace. You may assume that $m < 20$ and that no string is more than 10 characters long.

For the N^{th} input set (the first is number 1), your output must consist of an echo of the input, followed by a blank line and then a line containing either the list of i_j, if it exists, separated by single blanks, and otherwise the message "No solution." The format should be as follows (italic quantities to be suitably replaced).

⊣`Problem` N`:`
⊣`K=`K
⊣x_1 `...` x_m
⊣w_1 `...` w_m

⊣`Solution:` i_1 `...` i_k

In the absence of a solution, the last line reads simply

⊣`No solution.`

In case there are multiple solutions, choose the lexicographically least (i.e., smallest i_1, and for equal i_1's, smallest i_2, etc.). Finally, put a blank line before each `Problem` line.

6. Given a two-dimensional maze, find the *length* of the shortest path out of it. The maze is represented as a sequence of lines containing the characters ‘W’ (representing a wall), blank (representing a traversable square), ‘S’, indicating the starting position, and ‘E’, indicating the exit square. The exit square will be on the edge of the maze; all other squares on the edge will be ‘W’. Each line of the maze will be of equal length (the width of the maze). Moves must be to a blank square or to ‘E’, and must be either one step horizontally or vertically. You may assume that the maze has a solution, that its width and height are no more than 80 (including the bounding walls). The last line of the maze is followed by zero or more null lines and the end of file. The output should have the form

```
There are N steps to the exit.
```

For example, if the input is

```
WWWWWWWWWWWW
W   W S  W E
W W  WWW W W
W W      W W
W WWWWWWWW W
W          W
WWWWWWWWWWWW
```

your output should be

```
⊣There are 31 steps to the exit.
```