

# **1998 ACM Rocky Programming Contest**

## **Table of Contents**

**General Instructions**

**Problem 1: The Year 2000 Problem**

**Problem 2: Defective Microwave**

**Problem 3: Rank the Teams**

**Problem 4: Disk Sectors**

**Problem 5: Grid Decryption**

**Problem 6: Family Tree**

**Problem 7: Japanese Abacus**

**Problem 8: TinyBasic Interpreter**

## General Instructions

- For a solution written in C++, a source file named `PROGn.CPP`, for a solution written in C, a source file named `PROGn.C`, for a solution written in Java, an application (not an applet) named `PROGn.JAVA` must be submitted for judging. Here `n` is the assigned number for the problem.
- The judges will load your program into the indicated system (IBM Visual Age C++ or IBM Visual Age Java) and attempt to compile and execute it.
- The judges will not correct any error, even if it is only a misspelling of the program name, or if the wrong language is specified.
- Unless otherwise specified in the problem statement, each program will get its input from a file named `PROGn.DAT` where again `n` is the assigned number for the problem.
- The output from each program must go to a file named `PROGn.OUT`.
- Your program must read from and write to the current directory – do not specify the disk drive or directory in your program.

The first line of output from each program must be:

Problem `n` by Team `<team number>`

where the `<team number>` is replaced by your team number, and `n` is replaced by the problem number.

The last line of output from every program must be:

End of Problem `n` by Team `<team number>`

- There will be no blank lines between the first and last lines of your output, except as specified by the problem statement.
- Failure to follow these guidelines in a judged submission will result in “WRONG OUTPUT FORMAT” and your submission will be returned for correction.

### Input data compatibility:

Looking at the sample data in the instructions will not reveal everything about the input data file. If you write a C or C++ program to display the ASCII code of every character in an input file, you should find that

- there are no trailing blanks at the end of any line
- successive lines are separated by one new line character (ASCII 10)
- there are exactly two new line characters (ASCII 10) after the last printable character (because, when the data file was created using a text editor, the last line was terminated by one final carriage return).

### Scoring the contest:

1. The team that has solved the most problems in the time allotted is the winner.
2. If two or more teams solve the same number of problems, they are ranked by least total time. The total time is the sum of time consumed for each problem solved. The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submittal of the accepted run plus 20 minutes for each rejected run. There is no time consumed for a problem not solved.
3. Only one correct solution will be accepted for any one problem from a single team.

## Error Messages

There are five error categories for which a submission may be rejected. They are:

1. **Syntax Error** – submitted program will not compile.
2. **Run-Time Error** – program failed while running or program or data file name was wrong.
3. **Time-Limit Exceeded** – program took too long to run. Every problem should complete in a reasonable length of time. This means "effectively immediately." If a program takes more than a few seconds, it is too slow.
4. **Wrong Answer** – the answer may be just plain wrong, or it may fit into one of the subcategories:
  - **Inaccurate Answer** – numerical inaccuracy.
  - **Failed Test Case** – failed at least one of the test cases, passed the others.
  - **Too Little/Much Output** – reread the problem.
  - **Wrong Output Format** – reread the problem (possibly you forgot to bracket the output with the required team information).
5. **Presentation Error** – you failed to follow the instructions for submitting your program; for example, you submitted it under a wrong file name, or on a diskette intended for another problem.
  - The format of your output is important and must be exactly as shown in the samples, including blank lines and spacing on the line. Failure to follow the required output format will result in a "**Wrong Output Format**" error, and your program will be returned for correction. In many cases, the placement and number of blank spaces in the output are as significant as printable characters.
  - Any attempt by your program to write anything to the judge's disk except the required output files will result in immediate disqualification of your team from the contest.
  - Your team also may be disqualified for any activity that jeopardizes the integrity of the contest such as interfering with other teams or disruptive behavior. Contestants are not to converse with anyone except other members of their team and personnel designated by the contest directors.
  - The contest judges are solely responsible for determining the correctness of submitted solutions. Their decisions are final.

## Problem 1: The Year 2000 Problem (2 pages)

The infamous *Year 2000 Problem* stems from the fact that, in many computer systems and databases, the *year* component of a date is represented by two decimal digits. So, if a person's date of birth (in month/day/year format) is 04/10/95, was the person born in 1895 or 1995?

Here is an example of the input to your program, contained in a file PROG1.DAT:

```
Dijkstra, Edsger          603849562 ---- 0 9 1 99
Hopper, Grace Murray     234258030 ---- 97 2 12 0
Hollerith, Herman       602348748 FGSH 97 5 20 77
Shannon, Claude         571839480 FGJH 96 10 28 79
Lovelace, Ada           183099831 BAMS 0 6 30 92
Babbage, Charles        419309823 ---- 0 8 31 92
McCarthy, John          458489022 ---- 0 8 31 93
Noether, Emmy           522309877 ---- 97 12 10 94
Curie, Marie           543322761 ---- 0 1 15 94
Shugart, Alan           580798236 ---- 95 11 1 93
Knuth, Donald           037887624 ---- 0 9 28 93
Atanasoff, John        798723424 ---- 97 11 14 98
Von Neumann, John      526387309 ---- 0 11 14 98
```

Each line of the input consists of seven fields. Adjacent fields are separated by one blank space. In order to understand the meaning of each field, let us consider a typical line of the input, preceded by a template which shows the numbering of the columns:

```

      1           2           3           4           5           6
123456789012345678901234567890123456789012345678901234567890
Hopper, Grace Murray      234258030 ---- 97 2 12 0
```

The meaning of each of the seven fields of input follows:

- Columns 1-25: a person's name.
- Columns 27-35: the person's social security number.
- Columns 37-40: a 4-character code which designates the last known school attended by the person during the past ten years. The code "----" means the person did not attend school during that period.
- Columns 42-43: the last two digits of the most recent year in which the person filed a tax return, if a tax return is known to have been filed since 1950. Otherwise, there is no tax record for this person, column 42 will be blank and column 43 will contain the single digit 0.
- Columns 45-46: the person's month of birth.
- Columns 48-49: the day of the month of the person's date of birth.
- Columns 51-52: the last two digits of the person's year of birth (or just the last digit, if the person was born in the period 1900 - 1909).

If the data in columns 45-46, or columns 48-49, or columns 51-52 requires only one digit, it will be right-justified in the stated columns.

Your program will write its output to the file PROG1.OUT. Each line in the input will give rise to an almost identical line of output. The differences between a line in the input and the corresponding line in the output are:

- a) The two-digit year of birth in columns 51-52 will be replaced by the four-digit year of birth in columns 51-54. The first two digits of the year of birth will be 18 or 19, determined by rules which will be stated on the next page.
- b) A letter "K" in column 56 of the output will indicate a child who is expected to attend kindergarten starting September 1999. The rule for identifying such children will be stated on the next page.

For the particular example of the input file listed above, your program will write the following output to PROG1.OUT:

```
Problem 1 by team x
Dijkstra, Edsger          603849562 ---- 0 9 1 1899
Hopper, Grace Murray     234258030 ---- 97 2 12 1900
Hollerith, Herman       602348748 FGSH 97 5 20 1977
Shannon, Claude         571839480 FGJH 96 10 28 1979
Lovelace, Ada           183099831 BAMS 0 6 30 1992
Babbage, Charles        419309823 ---- 0 8 31 1892
McCarthy, John          458489022 ---- 0 8 31 1993
Noether, Emmy           522309877 ---- 97 12 10 1894
Curie, Marie           543322761 ---- 0 1 15 1994 K
Shugart, Alan           580798236 ---- 95 11 1 1893
Knuth, Donald           037887624 ---- 0 9 28 1993 K
Atanasoff, John        798723424 ---- 97 11 14 1898
Von Neumann, John      526387309 ---- 0 11 14 1998
```

Here are the rules which determine a person's age:

**Rule 1:** All persons in the input data will have been born on or before the date of this contest (November 14, 1998). This means:

- If the last two digits of the year of birth are 99 then the year of birth is 1899.
- If the last two digits of the year of birth are 98 and the date within that year is in the range from November 15 through December 31, then the year of birth is 1898.

Rule 1 is illustrated by Line 1 of the above input.

Before stating the remaining rules, let's define a person's apparent age to be the age on August 31, 1999, calculated from the input data, assuming that the first two digits of the year of birth are 19. (This definition will apply only to cases not covered under Rule 1.)

**Rule 2:** If the apparent age is 20 or more, the first two digits of the year of birth are 19. Rule 2 is illustrated by Lines 2-3 of the above input. (In other words, it is assumed that no person's age will exceed 119 years.)

**Rule 3:** If the apparent age is in the range 7..19, then the existence of school records will determine the first two digits of the year of birth: if the person attended school during the past ten years, then the first two digits of the year of birth are 19, otherwise they are 18. Rule 3 is illustrated by Lines 4-6 of the above input.

**Rule 4:** If the apparent age is 6 or less, then the existence of tax records will determine the first two digits of the year of birth: if there is a tax record for the person, then the first two digits of the year of birth are 18, otherwise they are 19. Rule 4 is illustrated by Lines 7-13 of the above input.

The **Kindergarten Rule** states that children who are five years old on August 31, 1999 will be expected to attend kindergarten starting September 1999.

You may assume that

- all cases which arise in the input data will be covered by the rules just stated.

## Problem 2: Defective Microwave (1 page)

You find a microwave oven for a bargain price at a garage sale, with a note attached to it with the cryptic message:

0=6.

When you take it home, you discover the meaning of the message: the keypad is broken, and pressing 0 results in a "6" appearing in the display. However, pressing 6 also results in a "6", and all other keys function properly. After using the microwave a little while, you realize most times can be well approximated avoiding the digit 0 in the set time. For example, heating a cup of coffee for 59 seconds rather than 60 didn't make a difference. Being a stickler for precision, you decide to write a program to pick out the closest matching times when a zero digit is not available.

Here is an example of an input file PROG2.DAT:

```
3205
 220
  90
 100
9099
1170
1210
3100
```

- Each line of the input file contains a positive integer (in the range 10 .. 9999), right-justified in columns 1-4, representing a desired cook time.
- The minutes (if any) and seconds have been concatenated into a single positive integer of the form MMSS, or MSS, or SS.
- In a four-digit integer, the leftmost two digits represent minutes, the rightmost two digits represent seconds.
- In a three-digit integer, the leftmost digit represents minutes, the rightmost two digits represent seconds.
- A two-digit integer represent pure seconds.
- Thus, the first three lines of the above input represent the desired cook times 32 minutes 5 seconds, 2 minutes 20 seconds, and 90 seconds, respectively.
- Each integer in the input will contain at least one zero (but the leftmost digit will always be non-zero).

For the above example of PROG2.DAT, your program will write the following output to PROG2.OUT:

```
Problem 2 by team x
Desired cook time = 3205; Closest approximation(s): 3165
Desired cook time =  220; Closest approximation(s):  179  181  219  221
Desired cook time =   90; Closest approximation(s):   89   91  129  131
Desired cook time =  100; Closest approximation(s):   59   61
Desired cook time = 9099; Closest approximation(s): 9139
Desired cook time = 1170; Closest approximation(s): 1169 1171 1211
Desired cook time = 1210; Closest approximation(s): 1169 1171 1211
Desired cook time = 3100; Closest approximation(s): 3111
End of problem 2 by team x
```

- For each of the desired cook times in the input, the output will print the desired cook time, followed by the closest approximation(s) which do not use the digit 0.
- Two approximations (to the same desired cook time) may actually mean the same length of time, for example, 179 and 219.
- When there are two or more closest approximations which approximate the desired cook time equally well, all of them must be listed; in fact they are to be listed so that the integers representing the cook times are in increasing order. For example, 181 is listed before 219, even though 1 minute 81 seconds (which equals 2 minutes 21 seconds) is a longer cook time than 2 minutes 19 seconds.
- Note that 1170 and 1210 represent the same cook time, hence they yield the same output.
- Observe the labeling and formatting details of the output. In particular, all numbers in the output are right justified in fields of width 5.
- You may assume that, for each desired cook time, the output will fit on one line having 71 or fewer characters.

Here is a formatting template, with two lines of the above output:

```

      1           2           3           4           5           6           7
12345678901234567890123456789012345678901234567890123456789012345678901
Desired cook time = 3205; Closest approximation(s): 3165
Desired cook time =  220; Closest approximation(s):  179  181  219  221
```

### Problem 3: Rank the Teams (2 pages)

You are given a number of teams and the results of games played by pairs of these teams. The teams could be football teams, bowling teams, basketball teams, depending on your favorite sport. Each team will be identified by an upper case letter. The maximum number of teams is 26. Here is an example of an input file PROG3.DAT:

```
A B
C D
A D
C B
D B
C A
```

Each line contains two upper case letters, separated by one blank space.

- The two letters on a line indicate the winner and loser (in that order) of a game played by the two teams. For example, the first line of the input listed above means that team A defeated team B.
- Once two teams have played a game, the winner of that game is ranked better than the loser of the game.

Your mission is to rank the teams, i.e., to produce a linearly ordered list starting with the best team, and progressing down to the worst one.

- The output will list all teams on one line, labeled and formatted shown below. Each team will appear in the list only once, and there will be one blank space between adjacent letters representing the teams.
- If “X Y” occurs in the input then X will occur somewhere before Y in the output. There may, however, be additional teams between X and Y in the output.

In the above example, team C was the only undefeated team, team A was defeated only by team C, team D was defeated by team A (and possibly some team(s) which defeated team A), and team B was defeated by team D (and possibly some team(s) which defeated team D). Therefore, the output written by your program to the file PROG3.OUT is

```
Problem 3 by team x
Ranking of the teams:
C A D B
```

End of problem 3 by team x

- You may assume that, for any given input, there will be one unique linear ordering (with no ties) of the teams.
- In the above example, every team played every other team. That may not, however, be the case, as shown by the examples below.

Here are two additional examples of input/output:

#### Example 2:

PROG3.DAT:

```
B A
A D
B C
C D
B E
E D
A C
C E
```

PROG3.OUT:

```
Problem 3 by team x
Ranking of the teams:
B A C E D
End of problem 3 by team x
```

**Example 3:**

PROG3.DAT:

X L  
Y K  
X K  
K L  
U W  
W X  
B X  
U Y  
B Y  
X Y  
B U

PROG3.OUT:

Problem 3 by team x  
Ranking of the teams:  
B U W X Y K L  
End of problem 3 by team x

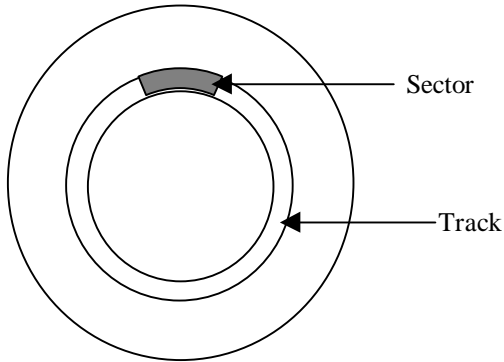
## Problem 4: Disk Sectors (3 pages)

Let's begin with some of the basic terminology of computer disks:

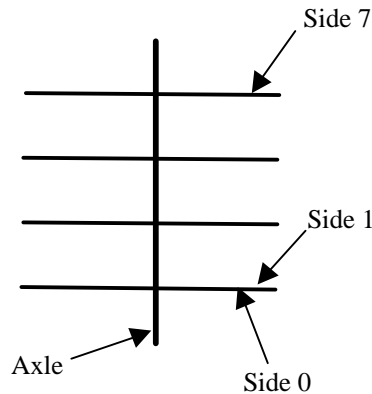
- A computer disk consists of a number of circular *platters* mounted on a common axle. A floppy disk consists of just one platter, whereas a hard disk consists of numerous platters.
- Each platter has data stored on both of its sides.
- Each side of each platter is divided into concentric rings called *tracks*.
- Each track consists of a number of *sectors*. All tracks, whether near the disk's edge or center, contain the same number of sectors.

The disk's total number of sectors is given by

$$(\text{number of sides}) * (\text{number of tracks per side}) * (\text{number of sectors per track}).$$



**Figure 1:** One side of one platter



**Figure 2:** Disk with four platters

There are two ways to identify (address) a particular sector on a disk: by its *physical coordinates* or by its *logical sector number*.

The *physical coordinates* of a sector are: the side number, the track number, and the physical sector number.

- Sides are numbered starting at zero. The side numbers 0, 1, 2, ... refer to the first side of the first platter, the second side of the first platter, the first side of the second platter, etc.
- Tracks are numbered starting at 0, the outermost track having number 0.
- Physical sectors are numbered starting at 1 and they go around a particular track. (Whether clockwise or counterclockwise, depends on the direction from which the disk is viewed.)

By using *logical sector numbers*, the disk's sectors can be viewed as a contiguous sequence (or array) of sectors.

Here are the rules which establish the relationship between physical coordinates and logical sector numbers:

- The logical sector numbers start at 0. The sector having logical sector number zero is physical sector 1 of track 0 of side 0 of the disk.
- Logical sector numbers increase along with physical sector numbers around each track.
- After the last sector of track  $t$  of side  $s$  (except when side  $s$  is the last side of the disk), the next sector in the sequence is physical sector 1 of track  $t$  of side  $s+1$ . In the sample output given below, this fact is illustrated by the results for logical sectors 8 and 9, as well as by the results for logical sectors 399 and 400.
- After the last sector of track  $t$  of side  $s$  (when side  $s$  is the last side of the disk), the next sector in the sequence is physical sector 1 of track  $t+1$  of side 0. In the sample output given below, this fact is illustrated by the results for logical sectors 17 and 18, as well as by the results for logical sectors 799 and 800.

The input to your program will come from the file PROG4.DAT. Each line of the file will contain four numbers. The first three of these are characteristics of the disk: the number of sides, the number of tracks per side, and the number of sectors per track. Each of these three numbers will be a positive integer not exceeding 512. The fourth one is the logical sector number of a particular sector on the disk; it will be a non-negative integer having at most nine digits. Here is an example of an input file:

```

2      80      9          8
2      80      9          9
2      80      9         17
2      80      9         18
2      80      9         49
2      80      9         40
2      80      9         21
2      80      9        1440
4      60     200         399
4      60     200         400
4      60     200         799
4      60     200         800
4      60     200        2300
512 512     512 134217727
512 512     512 134217728
    
```

Each line of valid input will generate one line of numerical output, consisting of seven numbers, written to PROG4.OUT. The first four of these will be the four numbers read from the input file. The remaining three will be the physical coordinates (side number, track number, and physical sector number) of the particular sector whose logical sector number was given.

A line of input will be invalid if the logical sector number is greater than or equal to the disk's total number of sectors. In this case, after the four input values, your program will write a message (shown in the example which follows) to the output file.

Here is the output resulting from the input listed above:

```

Problem 4 by team x
  Sides   Tracks   Sectors   Logical   Side   Track   Physical
    Per Side Per Track   Sector
=====
      2      80      9          8          0          0          9
      2      80      9          9          1          0          1
      2      80      9         17          1          0          9
      2      80      9         18          0          1          1
      2      80      9         49          1          2          5
      2      80      9         40          0          2          5
      2      80      9         21          0          1          4
      2      80      9        1440 Logical sector out of range!
      4      60     200         399          1          0          200
      4      60     200         400          2          0          1
      4      60     200         799          3          0          200
      4      60     200         800          0          1          1
      4      60     200        2300          3          2          101
     512     512     512 134217727          511          511          512
     512     512     512 134217728 Logical sector out of range!
End of problem 4 by team x
    
```

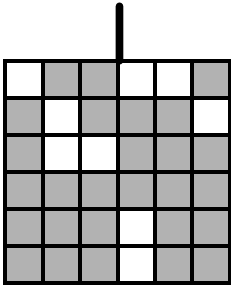
Finally, here is a formatting template shown with selected lines of the output:

1	2	3	4	5	6	7	
1234567890123456789012345678901234567890123456789012345678901234567890	Sides	Tracks	Sectors	Logical	Side	Track	Physical
	Per Side	Per Track	Sector				Sector
=====							
2	80	9	8	0	0	9	
2	80	9	9	1	0	1	
2	80	9	17	1	0	9	
2	80	9	1440	Logical sector out of range!			
4	60	200	399	1	0	200	
512	512	512	134217727	511	511	512	
512	512	512	134217728	Logical sector out of range!			

### Problem 5: Grid Decryption (3 pages)

N	t	i	o	v	l
e	e	s	4	,	m
.	b	e	_	_	_
1	9	_	_	t	9
h	_	_	r	e	8
_	_	d	_	_	a

Here is an encrypted message to be decrypted. The encrypted message comes as an  $N \times N$  (in this case,  $6 \times 6$ ) array of characters.



The person who decrypts the message (the "decryptor") uses a special grid to perform the task. (The "encryptor" used an identical grid to encrypt the message.) The grid is an  $N \times N$  array of squares. Some of the squares are solid, some are hollow. A small handle identifies the top of the grid and the decryptor also has a way of distinguishing the front side from the back. The squares of the grid have the same size as the squares holding individual characters in the message.

The decryption process will consist of four simple steps.

N				o	v	
	e					m
	b	e				
				r		
				_		

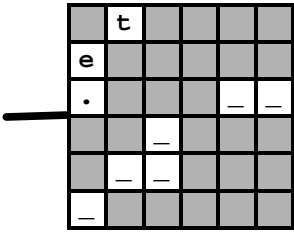
**Step 1.** The decryptor lays the grid on top of the message in its upright orientation and observes that some of the characters of the message are seen through the hollow squares. Reading these characters left-to-right across each row, starting at the top of the grid yields "November\_".

						l
				4	,	
				_		
1	9					9
						8
				_		

**Step 2.** The decryptor rotates the grid by ninety degrees clockwise, which exposes the next portion of the message, namely "14, \_1998\_".

		i				
		s				
				_	t	
h					e	
_		d				a

**Step 3.** The decryptor again rotates the grid by ninety degrees clockwise, which exposes the next portion of the message, namely "is\_the\_da".



**Step 4.** The decryptor again rotates the grid by ninety degrees clockwise, which exposes the next (and last) portion of the message, namely "te.\_\_\_\_\_".

Concatenating the results of the above four steps, the complete decrypted message reads "November\_14,\_1998\_is\_the\_date.\_\_\_\_\_" (The period here is followed by six underscores.)

Here is an example of an input file PROG5.DAT for your program:

```
6
011001
101110
100111
111111
111011
111011
Ntiov1
ees4,m
.be__
19__t9
h__re8
__d__a
```

- The first line contains the integer N, which is an even integer in the range 4 . . 16.
- The next N lines specify the grid in its upright orientation. Each of these lines specifies one row of the grid. A hollow grid square is represented by a 0, a solid one by a 1.
- The final N lines contain the encrypted message, as an N×N array of characters. Each of these lines will contain exactly N non-blank, printable characters.
- In the example shown here, the number of adjacent underscores can be determined by comparing neighboring lines. For example, the last line contains two underscores before the “d” and two underscores between the “d” and the “a”.

You may assume the following fact about the grid:

- Every character will be exposed exactly once as the grid is laid on the encrypted message and rotated.

Given the above input, your program will write the decrypted message to PROG5.OUT:

```
Problem 5 by team x
November_14,_1998_is_the_date._____
End of problem 5 by team x
```

- Note that the decrypted message will consist of exactly as many printable characters as the encrypted message, namely, N\*N printable characters.
- When the decrypted message contains more than 40 characters, all lines (except possibly the last line) of the decrypted message in PROG5.OUT will contain exactly 40 characters. This aspect of the output format is illustrated by the input/output example on the next page.

PROG5.DAT:

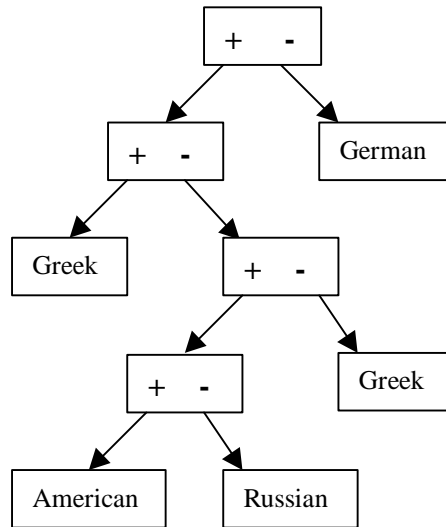
```
10
1111011001
1110111110
1000110100
1011000110
1011011111
1101111011
1111111110
1111111111
1111101101
1111011110
TterNo_ovP
rmoe_datgm
rberea__14
M,mm_19of9
_8oi_thune
g_i_Crlseo
ynt_Me99m_
orei8sal__
CAt._toChM
/mpIeuBM__
```

PROG5.OUT:

```
Problem 5 by team x
November_14,_1998_is_the_date_of_the_199
8_ACM/IBM_Tom_Mourey_Memorial_Computer_P
rogramming_Contest._
End of problem 5 by team x
```

## Problem 6: Family Tree (4 pages)

A person's family tree can be represented by a tree diagram, such as



The node at the tree's topmost level (known as the *root* of the tree) represents our *target person*. Every node of the tree represents a person. The tree has two types of nodes: *interior nodes* and *leaves*.

- An interior node is designated by a "+" (left) and a "-" (right) and pointers to its *left subtree* and *right subtree* which in turn represent the mother and father of the person represented by the given interior node. The topmost node of a left- or right subtree is also known as its *root*.
- A leaf has neither a left subtree nor a right subtree; it represents a person of known nationality.
- Every path which starts at the root of the tree and follows pointers to left- and right subtrees will eventually end at a leaf.

In the diagram shown above

- the target person's mother's mother was Greek,
- the target person's mother's father's mother's mother was American,
- the target person's mother's father's mother's father was Russian,
- the target person's mother's father's father was Greek, and
- the target person's father was German.

The input to your program, contained in the file PROG6.DAT, will contain a target person's family tree. The tree of the example illustrated above would be represented in a file as follows:

```

+
+
  Greek
-
+
+
  American
-
  Russian
-
  Greek
-
German
    
```

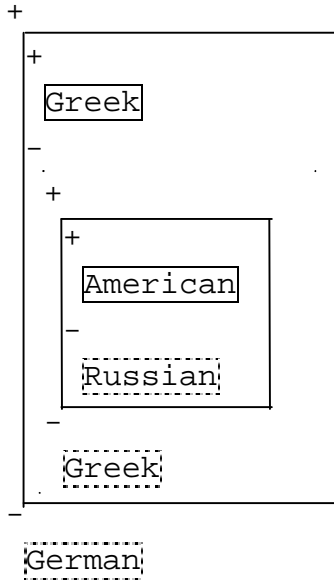
More generally, PROG6.DAT will have the following structure:

```

+
[One or more lines of input representing the left subtree]
-
[One or more lines of input representing the right subtree]
    
```

Each of the subtrees is either a leaf (represented by one line of input), or a tree having the above structure.

The successive nesting of subtrees in the above example is illustrated as follows; here each left (right) subtree is enclosed in a box with solid (dashed) border:



In general,

- Column 1 of line 1 of the input file contains the entire tree's leading "+".
- Each "+" is followed by the representation of a left subtree, starting one line below and one column to the right of the "+".
- The line following the representation of the left subtree contains a "-", in the same column as its matching "+".
- Each "-" is followed by the representation of a right subtree, starting one line below and one column to the right of the "-".

The output from your program, written to the file PROG6.OUT, will describe the target person's ethnic background. For the example given above, the output will be:

```
Problem 6 by team x
American      6.2500%
German        50.0000%
Greek         37.5000%
Russian       6.2500%
End of problem 6 by team x
```

- The nationalities which appear in the input file will be listed in the output, one per line, in lexicographically increasing order.
- Each nationality will be printed exactly as it appears in the input.
- The percentage of each nationality in the target person's ethnic background will be printed (on the same line) along with each nationality.
- Each percentage will be printed with a precision of four digits following the decimal point.
- Each percentage will be followed immediately by a percent sign.
- The decimal point of each percentage will be printed in column 21 of the output.

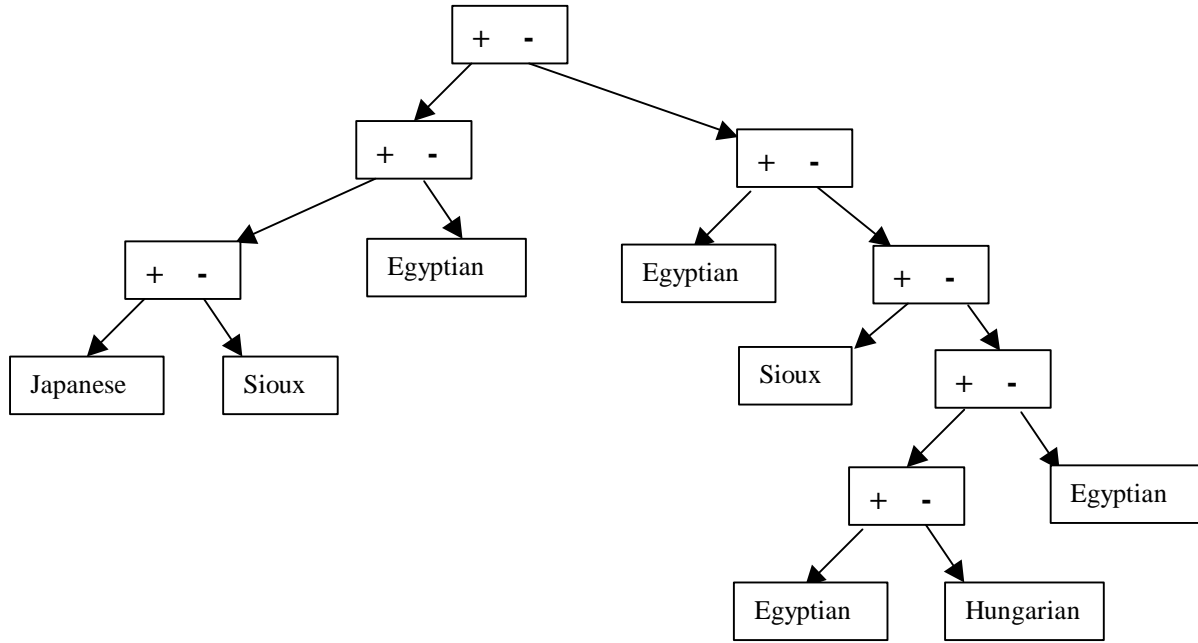
Here is a formatting template, shown with two lines of output:

```
123456789012345678901234567890
American      6.2500%
German        50.0000%
```

You may make the following assumptions:

- The maximum number of different nationalities in the input will be 20.
- Each nationality will contain only alphabetic characters.
- The maximum word length of each nationality is 16 characters.
- Each percentage in the output will be greater than 1.

As a **second example**, the tree diagram



is represented in the input file PROG6.DAT as

```

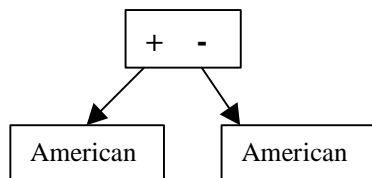
+
+
+
  Japanese
-
  Sioux
-
  Egyptian
-
+
  Egyptian
-
+
  Sioux
-
+
+
  Egyptian
-
  Hungarian
-
  Egyptian
    
```

and the corresponding output, written to PROG6.OUT is

```

Problem 6 by team x
Egyptian      59.3750%
Hungarian     3.1250%
Japanese      12.5000%
Sioux         25.0000%
End of problem 6 by team x
    
```

As a **third example** (illustrating the smallest family tree which may be input to your program), the tree diagram



is represented in the input file PROG6.DAT as

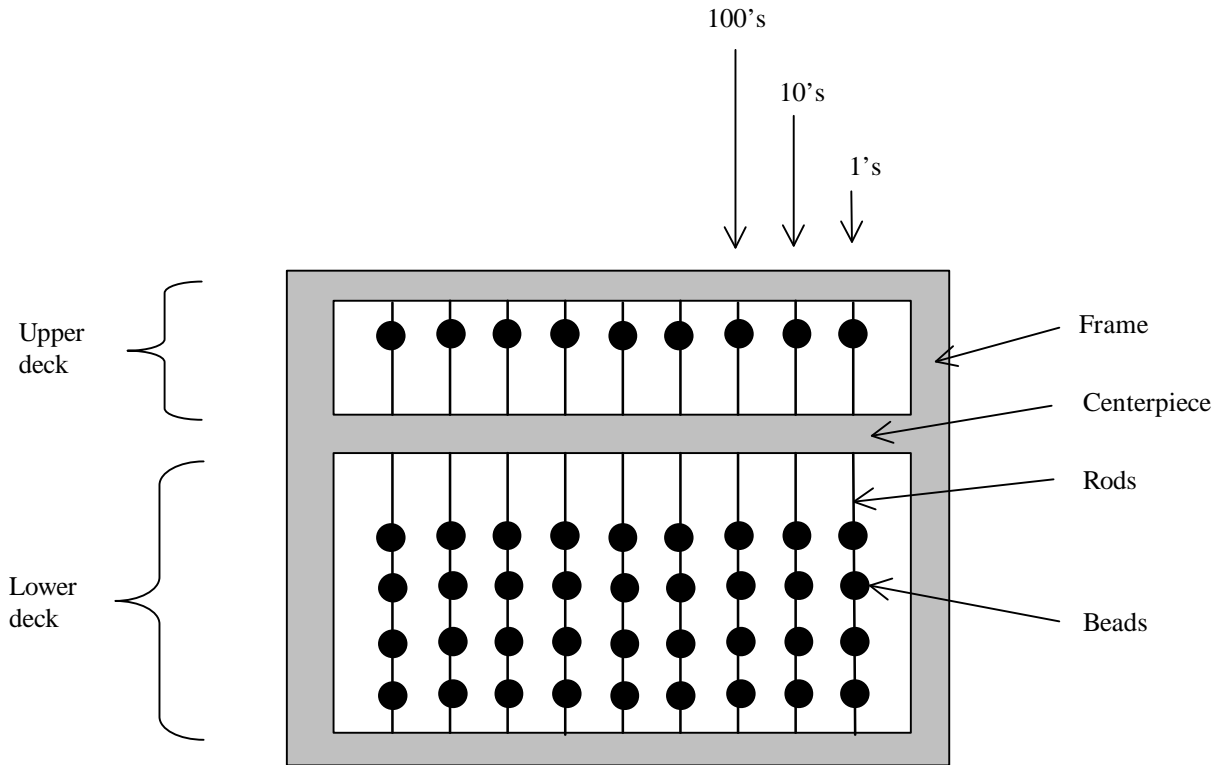
```
+  
  American  
-  
  American
```

and the corresponding output, written to PROG6.OUT is

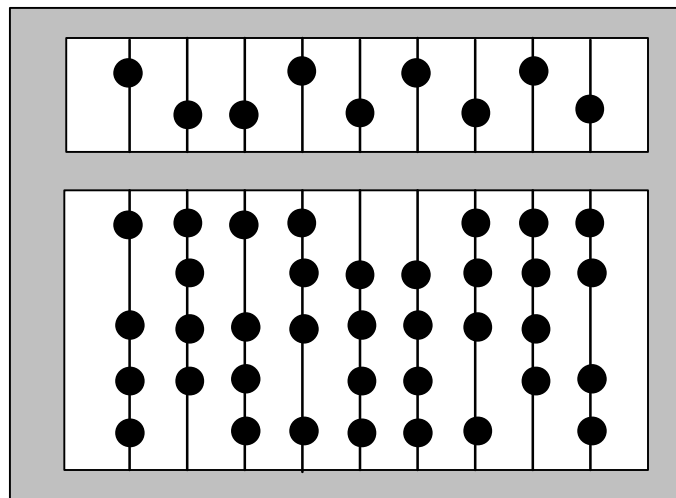
```
Problem 6 by team x  
American          100.0000%  
End of problem 6 by team x
```

### Problem 7: Japanese Abacus (2 pages)

A Japanese abacus and its components are shown in the figure:



- The model shown here can be used to represent non-negative integers having up to 9 digits in base 10.
- There are five beads on each of the 9 vertical rods, one in the upper deck and four in the lower deck.
- The beads can be moved up or down on the rods. Each rod will always have one vacant position in the upper deck and one in the lower deck.
- The beads in the rightmost position represent the unit's digit, in the next position, the ten's digit, and so on.
- Each bead in the upper deck can count as the digit 5 in the corresponding position.
- Each bead in the lower deck can count as the digit 1 in the corresponding position.
- To make a particular bead actually count, it has to be moved toward the centerpiece. In the configuration shown above, all of the beads are positioned away from the centerpiece, therefore it represents the value 0.
- In the configuration shown below, in the unit's (rightmost) place, for example, the bead in the upper deck (= 5) and two of the beads in the lower deck (each having a value of 1) have been moved toward the centerpiece. Therefore, the unit's digit is  $5 + 2 = 7$ . The value represented by this configuration is 196350847.



An abacus configuration will be represented in a text file according to the following rules:

- 10 lines are needed to represent an abacus configuration.
- Each of the 10 lines contains 11 characters (not counting the end-of-line character).
- Equal signs are used to represent the frame and the centerpiece.
- Each bead is represented by the upper case letter "O".
- Each rod position not occupied by a bead is represented by the vertical bar character "|".
- Each of the first, fourth, and tenth lines contains 11 equal signs.
- Each of the remaining lines has an equal sign in columns 1 and 11; each of the characters in columns 2..10 can be either a "O" or a "|", depending on the particular configuration.

The input to your program will be in a file PROG7.DAT. It will contain a particular abacus configuration. With the configuration shown in the above example, PROG7.DAT would be as follows:

```
=====
=O||O|O|O|=
=|OO|O|O|O=
=====
=0000||000=
=|O|000000=
=00000000|=
=000|00|00=
=O|00000|O=
=====
```

You may assume that the input file PROG7.DAT contains a valid abacus configuration. The output from your program, written to PROG7.OUT, will consist of the non-negative integer represented by the configuration. The configuration shown above will yield the following output:

```
Problem 7 by team x
Value of the abacus configuration = 196350847
End of problem 7 by team x
```

Here is a line of output shown with a formatting template:

```
1234567890123456789012345678901234567890123456789012345
Value of the abacus configuration = 196350847
```

- The unit's digit of the integer will be in column 45.
- If the result is zero, it will be represented by the single digit 0 in column 45.
- If the result is non-zero and contains fewer than 9 digits, there will not be any leading zeros displayed.

## Problem 8: TinyBasic Interpreter (2 pages)

The input to your program, contained in a file PROG8.DAT, will contain a computer program written in the TinyBasic language, to be specified below. The TinyBasic program will be a correct one, meaning:

- It will conform to the syntax rules of the TinyBasic language.
- When it is executed, it will produce one or more lines of output, and terminate. In particular, it will not go into an infinite loop.

Your program will be an interpreter for the TinyBasic language, meaning that the output produced by your program, written to PROG8.OUT, will be the output produced by the TinyBasic program.

### Rules of syntax of the TinyBasic language:

- Each line of a TinyBasic program has the form

[<spaces>]<line-number><spaces><statement>

More specifically, from left to right,

1. there are optional blank spaces.
2. there is a line number, which is required, starts with 1, and increases by 1 on each line.
3. there are one or more blank spaces.
4. there is one of the following statements:

LET<spaces><variable>=<expression>

PRINT<spaces><variable>

GOTO<spaces><expression>

IF<spaces><expression>

STOP

- **Rules which define <variable> and <expression>:**

<variable> is any single upper case alphabetic character, representing an integer value

<expression> is one of

<constant> integer constant in the range [ -10000 . . 10000 ], for example, 0, -5, or 34

<variable>+<variable> signed integer addition of two variables

<variable>><variable> the “greater than” operator between two variables evaluates to 1 if true, 0 if false.

- There are no blank spaces surrounding the “=” sign in a LET statement or within an expression.
- The maximum number of lines in a TinyBasic program is 100.

### Rules of execution of the TinyBasic language:

- Execution of the program begins on line 1.
- All variables used in the program have the initial value 0.
- Statements are executed consecutively, with the exception of the IF and GOTO statements.
- **Rules which define each of the five types of statements:**

LET is for variable assignment.

The result of adding two variables will be in the range [ -10000 . . 10000 ]

PRINT prints out the value of the variable in the form

<name>=<value>

which is left-justified and is followed by a new line.

There are no blank spaces surrounding the “=” sign.

GOTO jumps to a new location in the program given by <expression>.

<expression> is not necessarily a constant.

<expression> will evaluate to a valid line number within the program.

IF skips the next line of code if the value of the expression is zero.

The program will have at least two statements that follow any IF statement.

STOP halts execution.

The program will eventually execute a STOP statement.

The program may contain more than one STOP statement.

The last statement of the program does not need to be a STOP statement.

**Example 1.** Here is an example of a TinyBasic program, which prints out the first A odd integers, where A is set on line 1:

```
1 LET A=10
2 LET I=0
3 LET X=I+I
4 LET T=1
5 LET X=X+T
6 PRINT X
7 LET T=1
8 LET I=I+T
9 IF A>I
10 GOTO 3
11 STOP
```

Running the TinyBasic interpreter with this input should produce the following output in PROG8.OUT:

Problem 8 by team n

```
X=1
X=3
X=5
X=7
X=9
X=11
X=13
X=15
X=17
X=19
```

End of problem 8 by team n

**Example 2.** This example computes the negatives of three numbers, which are set on line 1, 5, and 9

```
1 LET X=-12
2 LET R=4
3 GOTO 14
4 PRINT V
5 LET X=3
6 LET R=8
7 GOTO 14
8 PRINT V
9 LET X=0
10 LET R=12
11 GOTO 14
12 PRINT V
13 STOP
14 IF Z>X
15 GOTO 23
16 LET V=1
17 LET T=-1
18 LET V=V+T
19 LET T=V+X
20 IF T>Z
21 GOTO 17
22 GOTO R+Z
23 LET V=-1
24 LET T=1
25 LET V=V+T
26 LET T=V+X
27 IF Z>T
28 GOTO 24
29 GOTO R+Z
```

Running the TinyBasic interpreter with this input should produce the following output in PROG8.OUT:

Problem 8 by team 2

```
V=12
V=-3
V=0
```

End of problem 8 by team 2

