

The 2000 ACM Northeast North America Regional Contest sponsored by IBM

Program 1: Mini-Spreadsheet (2 pages)

Given cells filled with functional formulas and data, display the calculated results of a mini-spreadsheet. Spreadsheet cells are labeled by columns(A-J) and rows (1-10). The cell in the 6th column of the 3rd row would be labeled with address F3. All cell entries will be integers. All function names and cell references are case insensitive. Any function not adhering to proper syntax should output #SYN# in its cell. The functions the spreadsheet should handle are as follows:

Average(Range) Calculate the average of the non-blank cells in a specified Range. Truncate result.

Large(Range, k) Calculate the kth largest distinct value of non-blank cells in a specified Range.

k may be a cell address. Ex.. Large(2,2,5,5,5,8,8,8,3) is 2.

Output #ERR# if impossible. Ex.. Large(2,2,5,5,5,8,8,4)

CountIF(Range, condition) Count the number of cells satisfying a condition. Conditions are ">n","<n",">=n","<=n", or "=n". n represents any integer. Positive numbers may have a leading "+". Zero is a value; an empty cell, denoted by "B", is not. Ex. CountIF(A4, B1:B5, ">0")

Median(Range) Find the median of the cells in a specified Range

The median is the middle number of the cells if they are in numerical order. If the Range contains an odd number of cells, the median is the average of the two middle cells. Truncate the result if there is an even number of cells.

Mode(Range) Find the mode of the cells in a specified Range

The mode is the most frequently occurring value in the range. In case of a tie, use the first modal value occurring in a list of the Range values. Rectangular sections of the Range should be listed by rows, from left to right.

SumIf(Evaluation Range, condition, Sum Range) Sum certain cells.

If a cell in the Evaluation Range meets the specified condition, the corresponding cell in the Sum Range should be included in the sum. Output #ERR# if there is not a 1-1 correspondence between Evaluation Range cells and Sum Range cells. Condition specifications are the same as those of the CountIf function. Rectangular sections of the Range are in row major form. That is, they should be listed by rows, from left to right. Any blank cell in the Evaluation Range counts for purposes of the required 1-1 correspondence, but the condition should automatically evaluate to False.

Ex. If A1=1, A2=2, A4=4, B1=10, C1=20,B3=30, B4=40

SumIf(A1:A2, 3 ,A4, ">2" , B1 , C1 ,B3:B4) is 70.

A range may be a <Cell Address>, <Cell Address:Cell Address>, any integer, or a combination of these separated by commas.

Examples are "C5", or "A3:B4"(meaning A3,B3,A4, and B4), "55",or the combination "D2, 55, A3:B4, J3", respectively. There may be spaces within the parentheses. There will not be spaces separating a column and a row in a cell address.

Input beginning with "R" indicates a particular row's data. Any row not accounted for should be filled with blanks. If a row is listed, all entries in the row are listed, from left to right, separated by blank(s). A row's entries may extend over more than one line.

There are no circular references in the data; however, one cell's function may use another cell's results as input. If there is an #ERR# or #SYN# in an input cell the function should display #INP#. If there is an #INP# in any input cell the function should display #INP#. #INP# is the error with most precedence.

SAMPLE INPUT:

```
R1 8 4 5 7 B 4 3 6 90 B
R3 7 8 11 14 Average(A3:D3,A3) SumF(A1 A2) B B B B
R5 B 3 4 B B B B B B Large( B1:B10 , 3)
R6 B 4 3 B B B B B B Average(4, J7)
R7 B CountIf(B1:B6, ">0") LARGE(C1:C6,D1:E6,1) B B
B B B
B Large(B1:B10, 5)
R10 B Mode(A1:I1) Median(A1:I1,B1:B3) B B B B B B SUMIF(B5:C6, ">3" ,A1:D1)
```

SAMPLE OUTPUT:

| | A | B | C | D | E | F | G | H | I | J |
|----|---|---|----|----|--------|---|---|---|-------|---|
| 1 | 8 | 4 | 5 | 7 | | 4 | 3 | 6 | 90 | |
| 2 | | | | | | | | | | |
| 3 | 7 | 8 | 11 | 14 | 9#SYN# | | | | | |
| 4 | | | | | | | | | | |
| 5 | | 3 | 4 | | | | | | | 3 |
| 6 | | 4 | 3 | | | | | | #INP# | |
| 7 | | 4 | 14 | | | | | | #ERR# | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |
| 10 | | 4 | 5 | | | | | | | 9 |

Output should be **exactly as above**. Column and row heading labels should be right justified in a field width of 5. All cells should be of width 5. Integers should be right-aligned within cells. Any cells without data should contain 5 spaces. There should be no extraneous spaces in the output.

Program 2: Reserve Bookshelf (2 pages)

A very small rural library has only one shelf dedicated to reserve books. (Reserve books are deemed important to a large number of library users and can therefore be checked out only for very short periods of time.)

- ❖ When the library's staff places a book on reserve, the book is put on the reserve shelf's left end. All books that were already on the reserve shelf must therefore be moved to the right, just enough to make room for the new book on the shelf.
- ❖ When a reserve book is checked out and subsequently returned, it is also put on the reserve shelf's left end, as if it were just being placed on reserve.
- ❖ What happens when the shelf is full? When a book is placed on reserve, or when a previously checked out reserve book is returned, and there is not enough room on the reserve bookshelf, then, starting at the right end, as many books (but no more) are removed from the reserve shelf (they are "taken off reserve") as needed in order to make room for the book to be put on the reserve bookshelf.

Here is an example of an input file `prog2.dat` and the resulting output file `prog2.out`:

```

250
PRINT
ADD      101 Uses for Wax Paper      38
ADD      Thin Thighs in 30 Days      63
ADD      The Republic                 44
ADD      Mein Kampf                   101
PRINT
ADD      Principia Mathematica       79
PRINT
CHECKOUT The Republic
ADD      On the Origin of Species     55
ADD      The C Programming Language   15
PRINT
RETURN   The Republic
ADD      101 Uses for Wax Paper      38
CHECKOUT The C Programming Language
PRINT
  
```

```

Program 2 by team X
AVAILABLE SHELF SPACE:      250

Mein Kampf                  101
The Republic                 44
Thin Thighs in 30 Days      63
101 Uses for Wax Paper      38
AVAILABLE SHELF SPACE:      4

Principia Mathematica       79
Mein Kampf                  101
The Republic                 44
AVAILABLE SHELF SPACE:      26

The C Programming Language   15
On the Origin of Species     55
Principia Mathematica       79
Mein Kampf                  101
AVAILABLE SHELF SPACE:      0

101 Uses for Wax Paper      38
The Republic                 44
On the Origin of Species     55
Principia Mathematica       79
AVAILABLE SHELF SPACE:      34

End of program 2 by team X
  
```

- ❖ The first line of the input file contains a single integer, the total available space on the reserve book shelf (when it is empty), measured in millimeters (1 inch equals about 25 millimeters). Its value will be at least 250 and at most 1500.
 - Your program will start with an empty reserve book shelf.
- ❖ Each subsequent line of input will start with one of the four commands ADD, CHECKOUT, RETURN, PRINT in upper case letters, starting in column 1.
 - There will not be any trailing blank spaces in the input file.
 - The ADD command:
 - The command is followed by a title (starting in column 10), consisting of at most 29 printable characters.
 - The title is followed (starting in column 40) by the thickness of the book (the amount of book shelf space it will require), a positive integer, not greater than 150.
 - The effect of the ADD command will be to place a new title on reserve, as described above.
 - Duplicate titles:
 - Once a particular title has been ADDED, it will not be ADDED again as long as that book is on the reserve book shelf or has been checked out. However, if a title has been forced off the reserve book shelf (by another

ADD command or by a RETURN command) then that title may appear in a subsequent ADD command (with the same thickness as before).

- The CHECKOUT command:
 - The command is followed by a title (starting in column 10) which is currently on the reserve shelf.
 - The effect of the CHECKOUT command is to remove the specified title from the reserve shelf (until it is placed back on the reserve shelf by a subsequent RETURN command).
 - A CHECKOUT may create a gap between two books (for example, when “The C Programming Language” is checked out in the example shown). Such a gap will count as part of the free space.
 - The RETURN command:
 - The command is followed by a title (starting in column 10) which is currently checked out.
 - The effect of the RETURN command is to put the specified book to the left end of the reserve shelf, as described in the introductory paragraphs.
 - The PRINT command:
 - The effect of the command will be to display in the output file:
 - the title and thickness of each book currently on the reserve shelf, one book per line, in the (left-to-right) order in which the books are currently on the reserve shelf,
 - the currently available space on the reserve shelf, and
 - one blank line.
- ❖ Pay attention to all formatting details, such as punctuation, upper/lower case characters, and the presence or absence of blank spaces and blank lines.
- All titles, as well as the sentence “AVAILABLE SHELF SPACE:” are left-justified, starting in column 1.
 - All numerical quantities are displayed right-justified, with the unit’s digit in column 34.

Program 3: Clock (1 page)

You are given a standard 12-hour clock with analog display, an hour hand and a minute hand. How many times does the minute hand pass the hour hand in a given time interval? Here is an example of an input file prog3.dat and the resulting output file prog3.out:

```
12 50 1 2
3 8 3 20
2 45 11 0
11 0 3 20
1 2 12 50
3 20 3 8
```

```
Program 3 by team X
Initial time  Final time  Passes
      12:50      01:02      0
      03:08      03:20      1
      02:45      11:00      8
      11:00      03:20      4
      01:02      12:50     11
      03:20      03:08     10
End of program 3 by team X
```

- ❖ The input file contains an indefinite number of lines; each line contains four numbers.
 - The first pair of numbers represents an “initial time”; the second pair represents a “final time.”
 - In each such number pair, the first number represents hours, second number represents minutes.
 - The hours will be in the range 1 . . 12, the minutes in the range 0 . . 59.
 - No initial time and no final time will be an instant at which the minute hand just passes the hour hand. (In particular, 12 00 will not occur as an initial or final time.)
 - No initial time will be the same as the corresponding final time.
 - Between each initial time and corresponding final time, the hour hand will have turned less than one full revolution (360 degrees).
 - As the hour hand turns from its initial position to its final position, it may or may not sweep past the number 12 on the clock’s dial.
 - If it does, then either the initial time is an “A.M.” time and the final time a “P.M.” time, or vice versa.
 - If it does not, then either both times (initial and final) are “A.M.” or both are “P.M.”
- ❖ Each line of input gives rise to one line of output, containing
 - the initial and final times, and
 - the number of times the minute hand passes the hour hand between the initial time and the final time.

Observe all details of formatting, such as upper/lower case letters, punctuation, blank spaces, and the absence of blank lines.

- ❖ In each time display, the hours and minutes are displayed in fields of width 2, separated by a colon.
 - The ten’s digit (of hours or minutes) is displayed as a zero if it is zero.

Here is a formatting template shown between two lines of the above output:

```
Initial time  Final time  Passes
12345678901234567890123456789012
      12:50      01:02      0
```

Program 4: Triangle Encapsulation (2 pages)

- ❖ Each line of the input file `prog4.dat` to your program will specify the x- and y-coordinates of the three vertices of a triangle in the xy-plane, starting at some vertex and proceeding clockwise. For each vertex, the x-coordinate will be specified before the y-coordinate. Each x- and y-coordinate will be an integer in the range $-9 \dots 9$. The input file will consist of one or more lines.
- ❖ Your program's output will be written to `prog4.out`. For each triangle specified in the input file, the output will list all points in the xy-plane which have integer coordinates and lie in the triangle's interior. (Such points are said to be encapsulated by the triangle.) Points that lie on one of the triangle's borders (in particular, the vertices of the triangle) will not be listed in the output.
- ❖ Each triangle specified in the input will encapsulate at least one point with integer coordinates.
- ❖ Within each line of input, the difference between the largest and smallest x-coordinate will not exceed 9.

Here is an example of an input file `prog4.dat` and the resulting output file `prog4.out`:

```
4 4 2 -1 -2 2
2 4 4 -3 0 -1
```

```
Program 4 by team X
( 2, 3) ( 3, 3)
(-1, 2) ( 0, 2) ( 1, 2) ( 2, 2) ( 3, 2)
( 0, 1) ( 1, 1) ( 2, 1)
( 1, 0) ( 2, 0)
( 2, 3)
( 2, 2)
( 1, 1) ( 2, 1)
( 1, 0) ( 2, 0) ( 3, 0)
( 1, -1) ( 2, -1) ( 3, -1)
( 3, -2)
End of program 4 by team X
```

Figure 1 illustrates the first triangle of the above example, and the points with integer coordinates in its interior. Note that $(1, 3)$ lies on this triangle's border and is therefore not encapsulated by the triangle.

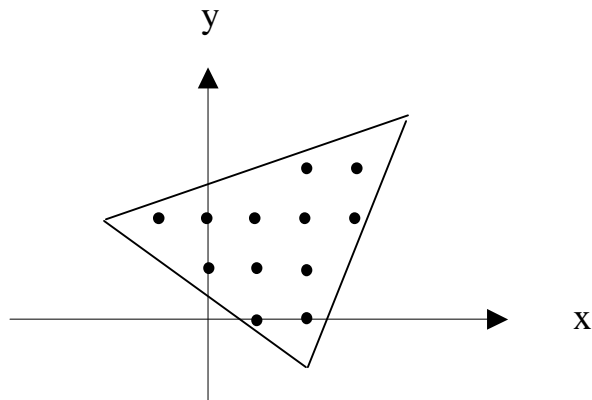


Figure 1

- ❖ Points (x, y) which are listed on one line of output have the same y-coordinate; the x-coordinates of these points are in increasing order from left to right.
- ❖ The y-coordinates of points in the interior of a triangle will be in decreasing order from one line of output to the next line.
- ❖ Each (x,y) point will be printed in a nine-column output field. Points that have the same x-coordinate (and pertain to the same triangle) will be listed in the same nine-column output field. Here is a formatting template between two lines of the above output:

```
( 2, 3) ( 3, 3)
12345678901234567890123456789012345678901234567890
```

$(-1, 2)$ $(0, 2)$ $(1, 2)$ $(2, 2)$ $(3, 2)$

- ❖ Each nine-column output field has the following layout (unless it is blank):
 - Column 1: opening parenthesis
 - Columns 2-3 : x coordinate (right justified)
 - Column 4: comma
 - Columns 5-7 : y-coordinate (right justified)
 - Column 8 : closing parenthesis.
 - Column 9: blank space to separate current nine-column field from the next one, if there is a next one.
- ❖ The line(s) containing the smallest x-value pertaining to a particular triangle (such as -1 for the first triangle in the above example) will not start with a blank output field.
- ❖ There will not be a blank output field between two non-blank ones on a given line.
- ❖ After listing the points encapsulated by each triangle, your program will produce one blank line of output. There will not be any additional blank lines in the output.

Program 5: Tribal Legislation (2 pages)

The Zhombi tribe populates an island in the South Pacific. The tribe consists of a number of clans (which are collections of related families). The tribe has a chief, and so does each clan.

- ❖ When a tribal law is proposed, the voting members of each clan convene and vote on the proposal. In case of a tie, the clan chief (who is otherwise not a voting member) will break the tie.
- ❖ After the clans have voted, the clan chiefs convene at the tribal headquarters to conduct the tribal level vote, which will determine whether the proposal passes. There each clan chief will cast a number of votes equal to the number of voting members of his/her tribe. Furthermore, all votes cast by one clan chief will be either in favor of the proposal or against the proposal, depending on how his/her clan voted. For example, if clan A has 28 voting members and voted in favor of the proposal, then clan A's chief will cast all 28 votes in favor of the proposal at the tribal level, even if the vote at the clan level was "15 for 13 against", or if the vote at the clan level was a 14:14 tie that the clan chief chose to break in favor of the proposal.
 - In case of a tie at the tribal level vote, the tribal chief (who is otherwise not a voting member) will break the tie.

You probably suspect now that a proposal could pass at the tribal level even if fewer than 50% of all votes were cast in its favor at the clan level. The main goal of the program will be to determine the smallest number of clan-level votes needed to pass a law at the tribal level, without having to break a tie either at the clan level or at the tribal level.

The input to your program will come from the file `prog5.dat`. Each **example** line will contain at least two, but no more than 20 integers, all on one line, separated by blank spaces, without any trailing blanks. Each of these numbers will be in the range 2 . . 999. The first line of the data file will indicate how many examples you are to calculate.

1

28 12 15 7

Each number in the input file represents the number of voting members of a clan (not counting the chief). We will refer to the clans of the tribe by upper case letters, in alphabetical order. Thus, the data in this example specify that the tribe consists of four clans, A, B, C, and D, having 28, 12, 15, and 7 voting members, respectively.

Given the above input, here is a valid instance of output written by your program to `prog5.out`:

Program 5 Example 1 by team X

```
Clan:                A    D
Clan level votes:    15    4
Tribal level votes:  28    7
```

```
Clan level summary:    19 out of    62, 30.6%
Tribal level summary:  35 out of    62, 56.5%
End of program 5 Example 1 by team X
```

The output presents a scenario in which a proposal passes at the tribal level with the smallest possible number of supporting votes at the clan level, without having to break a tie either at the clan level or at the tribal level.

- ❖ The line labeled "Clan" lists (in increasing alphabetical order) those clans that voted in favor of the proposal.
- ❖ The line labeled "Clan level votes" lists the (smallest) number of clan-level votes needed from each clan that voted in favor of the proposal, in order to pass the proposal at the tribal level. In those clans that are not listed, all votes were cast against the proposal.
- ❖ The line labeled "Tribal level votes" lists the votes cast in favor of the proposal at the tribal level.
- ❖ The two "summary" lines:
 - The number 19 represents the crux of the results, it is the smallest number of clan-level votes which can result in the stated goal.
 - The total (62 in this case) will always be the same number on the two "summary" lines; it is simply the sum of the numbers in the input.
 - The percentage of tribal-level votes (56.5% in this case) will always be greater than 50%. (In some cases the output, rounded to one decimal place, may be 50.0% even though the actual percentage is greater than 50%.)

- ❖ Non-uniqueness of output:
 - For the specific input shown above, the output shown above is not the only correct output. Here is another instance of correct output for the input shown above:

Program 5 Example 1 by team X

```
Clan:                B   C   D
Clan level votes:    7   8   4
Tribal level votes: 12  15   7
```

```
Clan level summary:    19 out of    62, 30.6%
Tribal level summary:  34 out of    62, 54.8%
End of program 5 Example 1 by team X
```

- ❖ In particular, for a given input, all correct instances of output will be identical on the line “Clan level summary”. Other lines of the output may be different, as long as the conditions of the problem are met.
- ❖ Pay attention to all formatting details, such as the presence or absence of blank lines, blank spaces, punctuation, upper/lower case letters.
 - The letters and numbers displayed on the “Clan”, “Clan level votes”, and “Tribal level votes” lines are right-justified in fields of width 4.
 - Integer values that appear in the “summary” lines are right-justified in fields of width 6.
 - You may assume that no more than 14 clans voted in favor of the proposal.
 - Here is a formatting template shown between two lines of the second instance of output from above:

```
Tribal level votes:  12  15   7
12345678901234567890123456789012345678901234567890
Clan level summary:    19 out of    62, 30.6%
```

- Note the column numbers of the ‘9’ in 19, the comma, the decimal point, and the percent sign.
- The percentages are displayed to a precision of one digit after the decimal point.
- Separate examples by a blank line.

Program 6: Floating Point Numbers (2 pages)

Floating point numbers are represented at the hardware level of most computer systems in normalized binary scientific notation. We will consider a hypothetical computer in which floating point numbers are represented in a 16-bit word. An example is shown in Figure 1. In the common Intel chips, the exponent of a **float** has 8 bits and the mantissa has 23 bits, otherwise the representation is the same as in Figure 1.

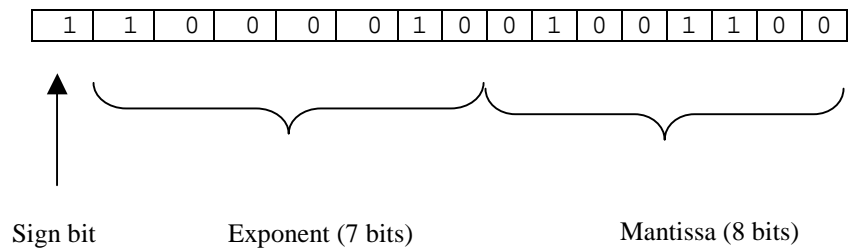
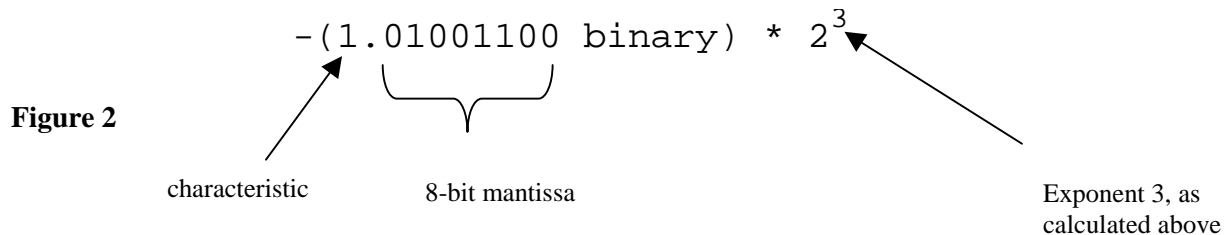


Figure 1

We will show that this particular example represents the number whose usual decimal representation is -10.375 .

- ❖ The leftmost bit (the sign bit) is 1 if the floating point number is negative, 0 otherwise.
- ❖ The next seven bits (**1000010** in this example) represent the Exponent component of the normalized binary scientific notation. The following two operations will reveal the exponent's actual value:
 - Evaluate the binary number **1000010**: it is $1*64+0*32+0*16+0*8+0*4+1*2+0*1 = 66$.
 - Subtract the bias correction 63 from 66, which yields 3 as the actual value of the exponent that we will use below. (The purpose of the bias correction is to allow negative exponents. As long as the exponent has seven bits, which will always be the case in this program, the bias correction is $63 = 2^{7-1} - 1$.)
- ❖ The remaining 8 bits (**01001100** in this example) constitute the mantissa, which is the fractional part of the normalized binary scientific notation. You will see in the next step how it is used.
- ❖ The sign bit, the mantissa, and the exponent are used to calculate the value of the floating point number:



- The period preceding the mantissa is the “binary point” which has the same meaning as the decimal point in a decimal (base ten) representation.
- In the normalized binary scientific notation there is only one binary digit preceding the binary point, and it is always 1 (except if the floating point number is the number zero). It is called the characteristic. Since it is always 1, it does not explicitly appear in the 16-bit representation of Figure 1.
- ❖ Since the number 1.01001100 is in binary, the digits to the right of the binary point represent successive negative powers of 2. Therefore, the number shown in Figure 2 can be re-written as

$$-(1+2^{-2}+2^{-5}+2^{-6}) * 2^3 = -10.375 = -1.0375 * 10^1$$

- Here the result has been expressed using scientific notation (base ten). Most programming languages would output this result as $-1.0375e+001$. The use of a lower case “e” or upper case “E”, and the number of leading zeros in the exponent can vary from one programming language to another.
- ❖ An important exception: if all bits except the sign bit are zero, then the floating point number is zero, regardless of the sign bit.

Here is an example of an input file prog6.dat and the resulting output file prog6.out:

```
1100001001001100
0011111100000000
1011111110000000
0000000010101010
0011011111100000
1001111011100000
0101011001010101
0100011011101101
0111111111111111
1100001000101100
0000000000000000
1000000000000000
```

```
Program 6 by team X
-1.037500e+001
 1.000000e+000
-1.500000e+000
 1.804180e-019
 7.324219e-003
-2.182787e-010
 1.117389e+007
 2.465000e+002
 3.682143e+019
-9.375000e+000
 0.000000e+000
 0.000000e+000
End of program 6 by team X
```

- ❖ The input file contains an undetermined number of lines. Each line contains, starting in column 1, exactly 16 printable characters which can be any permutation of 0s and 1s. The first character represents the sign bit, the next seven characters, the exponent, and the last eight characters, the mantissa of a floating point number in normalized binary scientific notation according to the conventions discussed on the preceding page.
- ❖ Each floating point number read from the input will be written by your program to the output file, one number per line. The following formatting conventions are required in your output:
 - The floating point numbers will be displayed in scientific notation (base ten).
 - Column 1 will either be blank (indicating a positive number or zero) or contain a minus sign (indicating a negative number).
 - Column 2 will contain a digit.
 - Column 3 will contain the decimal point.
 - Columns 4 – 9 will contain six additional significant digits of the floating point number.
 - The remaining columns will specify the exponent of scientific notation. The sign and magnitude of the exponent must of course be correct, but the exact format in which the exponent is displayed will be determined by the programming language you use.

Program 7: Bio - Informatics (2 pages)

Bio-informatics is an exciting new field of science, in which computer science techniques are applied to solving biological problems. The search for genetic drugs is one of the central problems of bio-informatics. In tackling this problem, genes from various organisms are compared.

- ❖ A gene is characterized by the sequence of amino acids that can be derived from it.
- ❖ There are altogether 20 amino acids. Each amino acid is identified by a one-letter abbreviation of its full chemical name. (The upper case letters of the alphabet, except B, J, O, U, X, and Z, are used to identify amino acids.)

Here is an example of an input file `prog7.dat` for your program:

| human | fruitfly | nematode | yeast | bacteria |
|-------|----------|----------|-------|----------|
| M | M | M | M | M |
| E | E | E | E | E |
| C | S | S | S | S |
| L | L | L | L | W |
| D | D | D | D | D |
| A | A | A | A | A |
| K | Q | G | N | G |
| C | A | A | C | K |
| T | T | T | T | T |
| S | H | E | M | R |

- ❖ Each logical column of the input file specifies a particular gene from a different organism. The number of organisms is at least three but not greater than eight.
 - The first line specifies the names of the organisms. Each name consists of at least one but not more than eight lower case characters and is right-justified in a field of width 9 characters.
 - Each of the remaining lines specifies an amino acid for each of the organisms listed on the first line. Each amino acid is represented by its one-letter abbreviation, right-justified in a field of width 9 characters, under the name of the organism with which it is associated. Thus, in the example shown, the amino acid sequence for the particular yeast gene is M, E, S, L, D, A, N, C, T, M.
 - The amino acid sequences of all organisms represented in a given input file will have the same length (in this example, 10).
 - The minimum length of the amino acid sequences in the input file is 10, the maximum length is 9999.
 - Each amino acid in the amino acid sequence of a particular gene occupies a certain position. The positions are numbered starting at 1 and they increase sequentially. Thus, the yeast sequence in the example shown has M in position 1, E in position 2, A in position 6, C in position 8 etc.

After re-displaying the names of the organisms (in the same order as in the input), your program will look for discrepancies among the amino acid sequences of the given organisms. Here is the output file `prog7.out` resulting from the above input:

```
Program 7 by team X
      human fruitfly nematode      yeast bacteria
  3          *
  4                      *
  7          *          *          *
  8                      *          *          *
 10          *          *          *          *
End of program 7 by team X
```

- ❖ For those positions in which all organisms have the same amino acid (positions 1, 2, 5, 6, and 9 in the example shown) , no output will be produced.
- ❖ In those positions in which not all organisms have the same amino acid (positions 3, 4, 7, 8 and 10 in the example shown) your program will:
 - Print the position number.

- Identify by an asterisk those organisms that deviate (in that particular position) from the most frequently occurring amino acid (in that particular position).
 - In position 3 of the given example, S is certainly the most frequently occurring amino acid, and human is the only organism that does not have S in position 3.
 - In case of a tie for the most frequently occurring amino acid in a particular position, the amino acid that has the rightmost occurrence (among those involved in the tie) will be chosen as the most frequent one.
 - For example, in position 8 in the given example, both C and A occur twice. We choose C as the most frequent amino acid in position 8, because its rightmost occurrence is under yeast, which is further right than the rightmost occurrence of A (nematode) in this position.
 - In the given example, there is an extreme case of a tie in position 10: all five organisms have different amino acids. Therefore, the amino having the rightmost occurrence, namely R, will be designated as the most frequent one.

Two lines of the above output are reproduced here with a formatting template:

```

          1           2           3           4           5           6           7
1234567890123456789012345678901234567890123456789012345678901234567890
      human fruitfly nematode   yeast bacteria
3
      *
```

- ❖ Note in particular:
 - The position numbers are right justified in columns 1-4.
 - Column 5 is blank.
 - Starting in column 6, the output (name of an organism or an asterisk) will be right-justified in fields of width 9.
- ❖ Also pay attention to formatting details, such as upper/lower case variations, blank spaces, and the absence of blank lines.

Program 8: Roman Forts (2 pages)

This program will acquaint you with the judicious method used by rulers of the ancient Roman empire to choose the locations of their military forts.

You are given a collection of cities.

- ❖ The cities will be labeled by upper case letters in alphabetical order, starting with 'A'.
- ❖ Given any pair of cities, there is one and only one path between them (see Figure 1).
 - This path may be a direct connection (as between J and C in Figure 1) or it may pass through a number of intermediate cities (as between J and B in Figure 1).
 - In the example of Figure 1, the path from J to B passes through C, G, and F.

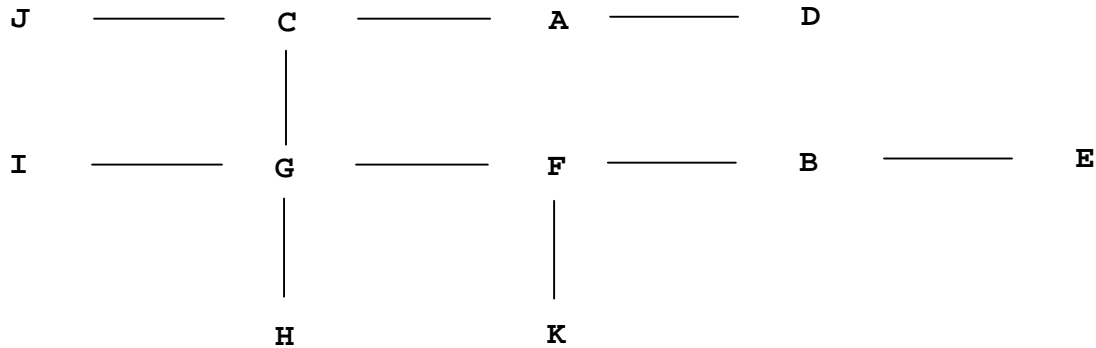


Figure 1

- ❖ The actual number of miles between cities is irrelevant. The distance between a pair of cities is defined as the number of hops (each “hop” being taken over a over a direct connection) between them.
 - In Figure 1, the number of hops between J and C is 1, and the number of hops between J and B is 4.
- ❖ Initially, only one of the cities is fortified (by a military fort). As your program executes, additional cities will be fortified.
- ❖ If a city is not fortified, we say it is vulnerable.
 - Furthermore, if a city is vulnerable, its degree of vulnerability is defined as the shortest distance between it and a fortified city.
- ❖ When one or more cities have already been fortified, the location of an additional fort is chosen by a simple rule: it is the most vulnerable city (the one having the highest degree of vulnerability).
 - In case of a tie, choose the alphabetically first city among those tied as the most vulnerable city.

An example of an input file prog8.dat is shown on the right.

- ❖ The first line of the input file contains three items of information, separated by one or more blank spaces:
 - The total number of cities in the given collection. It will be in the range 3 . . 26. In this example, it is 11, which means that the cities are labeled A . . K.
 - An upper case letter in the valid range of cities, denoting the city which is initially fortified.
 - The total number of cities that are to be fortified, including the initially fortified one. It will be at least three and not greater than the total number of cities in the collection
- ❖ Each of the remaining lines of the input file will contain two upper case letters, separated by one or more blank spaces.
 - Each letter will be in the range of labels of cities in the given collection.
 - Each pair of letters on one line of input determines a direct connection between two cities.
 - The direct connections listed in the input file shown on the right is identical to the direct connections of Figure 1.
 - The set of direct connections listed in the input file will satisfy all previously stated conditions.
- ❖ All lines of input will be free of leading or trailing blank spaces.

```

11 J 5
A D
B E
F B
C G
F G
C A
G H
G I
J C
K F
  
```

The output, written by your program to prog8.out, will list the cities to be fortified, as upper case letters, in the correct order. Given the above input, the following output will be produced:

```
Program 8 by team X
J E D H K
End of program 8 by team X
```

- ❖ Explanation of the output:
 - When there is only one fortified city, the degree of vulnerability of each vulnerable city is its distance from the fortified city.
 - Initially, with J being the only fortified city, E is the most vulnerable city with degree of vulnerability 5.
 - With J and E being fortified, D, H, I, and K contend for being the most vulnerable city, all having degree of vulnerability 3.
 - For example, the distance from D to E is 6, the distance from D to J is 3; the smaller of these two numbers is 3. Therefore, the degree of vulnerability of D is 3.
 - The alphabetically first city among the four contenders (D, H, I, and K) is D, which will be the next city to be fortified.
 - Similarly, H, and then K, will be selected as the next city to be fortified. At this point, the number of fortified cities is 5 (as specified in the input), and the program terminates
 - Note that city I is not fortified, because its degree of vulnerability drops from 3 to 2 once H is fortified, making K the most vulnerable city.
- ❖ Formatting details:
 - The upper case letters denoting the cities in the output will be separated by one blank space.
 - There will not be any blank lines or leading blank spaces in the output.