University of Computer Studies, Yangon

Asia-Yangon National Programming Contest

Rehearsal

June 24, 2016

Problem Set

Please check that you have **7** problems and **18** sheets (excluding additional materials).

Problem A.	**The Castle**	(2 pages)
Problem B.	**Mean and Median of Measured Wave Heights**	(3 pages)
Problem C.	**Mickey in the Muddy Yard**	(4 pages)
Problem D.	**Evaluating Voltage Measurement Data**	(3 pages)
Problem E.	**Performing Arithmetic Operations in Postfix Expression**	(2 pages)
Problem F.	**Movement of a Knight**	(2 pages)
Problem G.	**The Buses**	(1 page)

Problem A: The Castle

Run Time Limit: 5 sec

```
#############################
#   |   #   |   #   |   |   #
#####---#####---#---#####---#
#   #   |   #   #   #   #   #
#---#####---#####---#####---#
#   |   |   #   #   #   #   #
#---#########---#####---#---#
# ->#   |   |   |   |   #   #
#############################
```

\# = Wall
| = No Wall
--- = No Wall
-> = It points to the wall to remove according to the example output

The above figure shows the map of a castle. Write a program that calculates

1. How many rooms the castle was
2. How big the largest room is
3. Which wall to remove from the castle to make as large a room as possible

The castle is divided into m*n (m≤50, n≤50) square modules. Each such module can have between zero and four walls.

***Input Data*:**

The map is stored in the form of numbers, one for each module.

- The input starts with the number of modules in the north-south direction and the number of modules in the east-west direction.
- In the following lines each module is described by a number (0≤p≤15). The number is the sum of: 1(= wall to the west), 2(= wall to the north), 4(= wall to the east), 8(= wall to the south). Inner walls are defined twice; a wall to the south in module 1,1 is also indicated as a wall to the north in module 2,1.
- The castle always has at least two rooms.

***Output Data*:**

As output, the following are written on three lines: First the number of rooms, then the area of the largest room (counted in modules) and a suggestion of which wall to remove (first the row and then the column of the module next to the wall and finally the compass direction that points to the wall).

Sample Input	*Sample Output*
4 7 11 6 11 6 3 10 6 7 9 6 13 5 15 5 1 10 12 7 13 7 5 13 11 10 8 10 12 13	5 9 4 1 E

=-=

Problem B: Mean and Median of Measured Wave Heights

Run Time Limit: 2 sec

As part of a study to evaluate the reasons why a particular beach is eroding quickly, a number of wave height measurements have been made. To calculate the movement of the sand, it is necessary to determine an average wave height from the measurements. Two different types of averages can be taken, the mean and the median. Write a program to calculate the mean and median values of measured wave heights. Read the input wave heights and print the results.

The definition of the mean:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

where x_i = i[th] value in a list of numbers,
n = number of x values in the list,
$\bar{x}$ = mean of the set of n values.

In other words, the mean is the sum of all the values in a list divided by the number of values.

The median of the set of n data points commonly is described as follows:

n_{lower} = number of values less than or equal to x_i
n_{higher} = number of values greater than or equal to x_i

If n_{lower} is greater than $\frac{n}{2}$ and n_{higher} is greater than $\frac{n}{2}$, then

$$\hat{x} = x_i$$

where $\hat{x}$ = median value.

Assume that only an odd number of values in a list, because the definition of median becomes less clear for an even number of values.

Algorithm

- Read the input data values of wave heights that are separated by only one space.
- For mean,
  1. Sum all of the values in the list.
  2. Divide the sum by the number of values in the list.
- For median,
  1. Compare a value to the other values, and
     (a) Count the number of values less than or equal to the compared value.
     (b) Count the number of values greater than or equal to the compared value.

2. If both (a) and (b) are greater than n/2, then found the median value, stop the repetition and print it.
3. If either (a) or (b) are not greater than n/2, then repeat step 1 with the next value on the list.

For the results, print the list the values of x_i, number of values <= x_i and number of values >= x_i before discovering the median value, and finally, print out the mean and median value.

As an example, once per day over a period of approximately one month (29 days), wave heights have been measured. The following values have been found for each day (measurements in centimeters):

67 87 56 34 85 98 56 67 87 90 45 42 31 97 58 78 12 16 22 42 83 95 53 27 49 85 58 79 79

In this example, the following is the list of the values in order, and the number of values above and below that value:

Value x_i	Number of values < = x_i	Number of values > = x_i
67	17	14
87	25	6
56	13	18
34	6	24
85	23	8
98	29	1
56	13	18
67	17	14
87	25	6
90	26	4
45	9	21
42	8	23
31	5	25
97	28	2
58	15	16

The mean wave height is **61.3103** and the median value is **58**.

***Input Data*:**

The first line is the number of data points to be calculated which contains integer T (1<=T<=31). The input for all data points is given in a single line by separating each point with a single space.

***Output Data*:**

Print the list of pairs for the value x_i, the number of values <= x_i and the number of values >= x_i by separating with a single underscore among them. After that, output the mean value in four decimal-place format and the median value.

Sample Input	Sample Output
29 67 87 56 34 85 98 56 67 87 90 45 42 31 97 58 78 12 16 22 42 83 95 53 27 49 85 58 79 79	67_17_14 87_25_6 56_13_18 34_6_24 85_23_8 98_29_1 56_13_18 67_17_14 87_25_6 90_26_4 45_9_21 42_8_23 31_5_25 97_28_2 58_15_16 61.3103 58

=-=

Problem C: Mickey in the Muddy Yard

Run Time Limit: 3 sec

A robot named "Mickey" has to pass a muddy yard to get his destination. You have to help Mickey for his ways to get the destination. Suppose you are required to create a yard with **M** rows and **N** columns that corresponds to the area of the muddy yard. Assumed that his starting point is row# 1 and column# 1, and his destination point is row# M and column# N. In a muddy yard with 9 rows and 9 columns, Mickey will be in one cell located at row#1 and column# 1 and he will pass through the muddy yard to get his destination cell located at row# 9 and column# 9.

Mickey's Back

Mickey's Right | ***Mickey's Left***

	1	2	3	4	5	6	7	8	9
1	☺	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	⊗

Mickey's Front

☺ ***Mickey's Start Position***

⊗ ***Mickey's Destination***

The very first task that you need to do is to design the grid and to set the position of swamps, but to ensure that position of all swamps should not overlap. At first, the swamps should be placed in the areas of the cell and there should only be one swamp at any one particular area of the cell. The number of swamps to be placed is not less than 20% of the total number of cells. Otherwise, the process will terminate immediately by displaying the message: "Not enough Swamp! Bye Bye!". If the area of swamps in the input is beyond the boundary of the yard, or it is at the starting point or destination point, the message "Invalid Swamp! Bye Bye!", will be displayed and then, the program will terminate immediately. If the number of and area of swamps are valid, you have to plot every cell by setting the number of swamps adjacent to the area.

In each turn, Mickey can go straight forward or turn left only. So, his legal positions to move can only be increased in both rows and columns. If an area of a cell containing a swamp is revealed in front of or left of his location, he cannot go through the swamp and he has to choose another way without revealing swamp. But if there are two cells with swamps in both his front and left cells, he can be blocked for this way and he has no choice to get another way. A digit is revealed in the area, indicating the number of swamps adjacent to the area. If this number is zero, then the area and its entire adjacent areas will be unveiled as blank.

As an example, in a `9x9` muddy yard, if there are `16` swamps in `16` cells located at the positions: `{2,1}`, `{2,2}`, `{2,6}`, `{3,1}`, `{4,5}`, `{4,7}`, `{4,1}`, `{5,2}`, `{2,5}`, `{5,5}`, `{5,7}`, `{6,1}`, `{1,6}`, `{5,6}`, `{7,2}` respectively, the following result will be printed. X means swamp located in that cell, 0 means no swamp in its adjacent area and other digit means the number of swamps adjacent to the area.

Starting Point

eg: {3,2} is surrounded by 4 swamps.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	2	2	1	1	3	X	2	0	0
2	**X**	**X**	1	1	X	X	2	0	0
3	**X**	**4**	1	2	3	4	2	1	0
4	**X**	3	1	2	X	5	X	2	0
5	3	X	1	2	X	X	X	2	0
6	X	3	2	1	2	3	2	1	0
7	3	X	1	0	0	0	0	0	0
8	X	2	1	0	0	0	0	0	0
9	1	1	0	0	0	0	0	0	0

eg: If Mickey steps into {1,5} or {4,6}, he will be blocked in these cells. So, there are 2 blocks in this example. But there is other way to pass through to get his destination.

End Point

Input Data:

The input begins with a number representing the number of test case: T (1<=T<=10). The second line is the dimension of the yard (**M** rows and **N** columns). The third line is the number of swamps and the other lines represent the positions of swamps (i,j).

Output Data:

Print the output as follows:

(1) the number of swamp in the adjacent area for each cell
(2) the position/s of the cell in which Mickey can be blocked
(3) the number of blocked position, and if there is no way to pass through, print a message like: "No other way to pass through!".

Sample (1)

- Three swaps is less than 20% of the total number of cells.

Sample Input	Sample Output
3 5 7 3 1 2 5 4 3 9	`Case_1:Not enough Swamp! Bye Bye!`

Sample (2)

- 6, 7 and 9 are beyond the border.

Sample Input	Sample Output
2 5 5 5 5 7 4 9 1 5 6 5 2 3	`Case_1:Invalid Swamp! Bye Bye!`

Output Format: Case<underscore><case#><colon><your output>

Sample (3)

- In both test case (1) and (2), Mickey cannot reach his destination.
- In test case (3), Mickey can be blocked in two position (1,5) and (4,6), but there is other way to pass through to his destination.

Sample Input	Sample Output
3	Case_1:
5 5	3_X_2_0_0
6	X_X_3_1_1
4 5	2_2_2_X_2
5 4	0_0_2_3_X
2 2	0_0_1_X_2
1 2	1,1
2 1	No_of_Block:1
3 4	No other way to pass through!
5 5	Case_2:
6	1_2_X_1_0
3 1	3_X_3_1_0
2 2	X_X_3_1_1
3 2	2_3_3_X_1
1 3	0_1_X_2_1
5 3	2,1
4 4	1,2
9 9	No_of_Block:2
16	No other way to pass through!
2 1	Case_3:
2 2	2_2_1_1_3_X_2_0_0
2 6	X_X_1_1_X_X_2_0_0
3 1	X_4_1_2_3_4_2_1_0
4 5	X_3_1_2_X_5_X_2_0
4 7	3_X_1_2_X_X_X_2_0
4 1	X_3_2_1_2_3_2_1_0
5 2	3_X_1_0_0_0_0_0_0
2 5	X_2_1_0_0_0_0_0_0
5 5	1_1_0_0_0_0_0_0_0
5 7	1,5
6 1	4,6
1 6	No_of_Block:2
5 6	
7 2	
8 1	

Output Format: Case<underscore><case#><colon><end line>
For all cells: <digit><underscore><digit><underscore><digit>
For all blocks: <digit><comma><digit>

=-=

Problem D: Evaluating Voltage Measurement Data

Run Time Limit: 2 sec

Voltage measurements have been made for an electronic circuit at various times. However, after collecting the data, it is desired to know the voltage at a time that is different from the times at which the data were collected. Write a program that will read the collected voltage data, the corresponding times, and the time at which the voltage is desired and compute the desired voltage based on a linear interpolation using the nearest measured values. Have the program read the time and voltage measurement data and the desired time from the keyboard, and print the interpolated voltage.

Interpolating linearly between two points, (x_1, y_1) and (x_2, y_2), essentially means drawing a straight line connecting the points and obtaining a y value on that line for a given x value that is between the x values of the two endpoints.

The slope of the line, m, is:

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

The **x** distance from x_1 to the given x_{value} is $(x_{value} - x_1)$. So, the calculated **y** value is:

$$y_{value} = \frac{(y_2 - y_1)}{(x_2 - x_1)} (x_{value} - x_1) + y_1$$

In interpolating between points $(i-1)$ and i, then the equation becomes:

$$y_{value} = \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})} (x_{value} - x_{i-1}) + y_{i-1} \quad \text{---------- Equation (a)}$$

Algorithm

1. Read all of the data point x and y coordinates.
2. Read the x value for which the y value is to be calculated.
3. Determine the data point numbers that are immediately to the left and right of the x value specified.
4. Interpolate using **Equation (a)** mentioned above.
5. Print the result in two decimal place format.

For example:
Number of data points =10

Data Point	Time (milliseconds)	Voltage(millivolts)
0	0	23
1	2	78
2	3	89
3	6	-12
4	8	0
5	19	90
6	29	18
7	34	-23
8	37	76
9	45	98

The estimate of the voltage at **25** milliseconds is **46.8** millivolts.

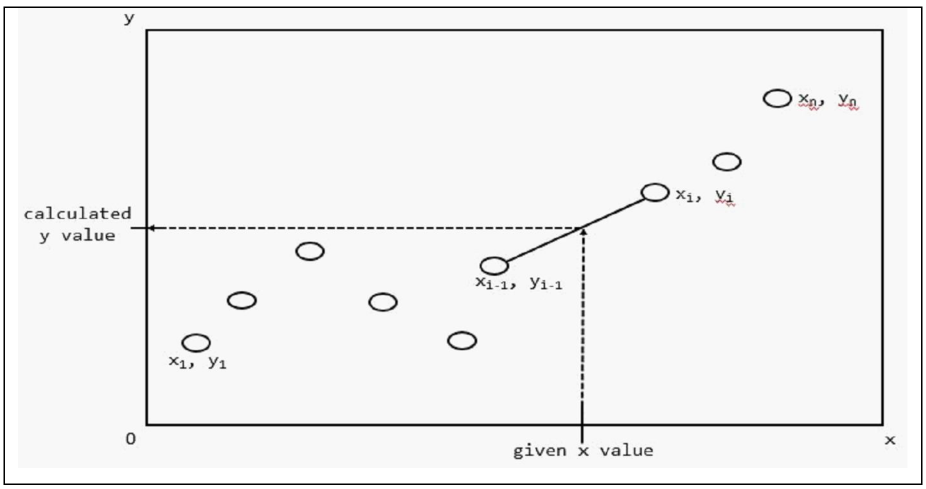


Figure: Sample location of n data points

Input Data:

First line of the input will be a positive integer T (1<=T<= 20) that represents the number of data points. Each point is entered line by line with the value of Time (milliseconds) and the value of Voltage (millivolts) by separating with a single space. The last line is the milliseconds to be applied to estimate the voltage.

Output Data:

Print out the value of the estimated voltage in two decimal place format.

Sample Input	*Sample Output*
10 0 23 2 78 3 89 6 -12 8 0 19 90 29 18 34 -23 37 76 45 98 25	46.80

=-=

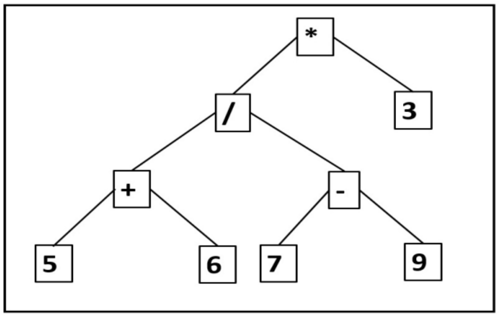
Problem E: Performing Arithmetic Operations in Postfix Expression

Run Time Limit: 3 sec

Any expression in the standard form like "`2*3-4/5`" is called an Infix (Inorder) expression. The Postfix Expression (Postorder) form of the above expression is "`2 3 * 4 5 / -`". In normal algebra, we use the infix notation like "`a+b*c`". The corresponding postfix notation is "`a b c * +`". For simplicity, you should consider only the four integer arithmetic operators: `+, -, *, /`. In a postfix expression, the operands are listed before the operators. Operators are just after the two operands. Thus, the expression "`3+4`" in postfix becomes: "`3 4 +`".

It is a convention which humans are familiar with and is easier to read in the infix notation. But for a computer, calculating the result from an expression in this form is difficult. Hence the need arises to find another suitable system for representing arithmetic expressions which can be easily processed by computing systems. The Prefix and Postfix Notations make the evaluation procedure really simple. But since we can't expect the user to type an expression in either prefix or postfix, we must convert the infix expression (specified by the user) into prefix or postfix before processing it. Therefore, your program would have to convert the infix expression to postfix and to compute the result from the converted expression.

For example, the expression `5*(3+4)` becomes: `5 3 4 + *` and the expression `(35-3*(3+2))/4` becomes: `35 3 3 2 + * - 4 /`. The result of the expression `((5+6)/(7–9))*3` is `-16.5`. The operation using postfix (reverse Polish) notation as: `5 6 + 7 9 - / 3 *`. The binary tree can be represented like as follows:



Input Data:

The first line of the input is the number of test cases and each test case contains the infix expression in a single line without any space.

Output Data:

Output will be the corresponding postfix expression separated by an underscore between each operand and operator per line, and in the next line, print its calculated result.

Sample Input	Sample Output
3 (84-2)/54+72-9*100 ((90-1)*11)/3+4*2 (1+27)-(3+36)/20*34	Case_1: 84_2_-_54_/_72_+_9_100_*_- -827 Case_2: 90_1_-_11_*_3_/_4_2_*_+ 334 Case_3: 1_27_+_3_36_+_20_/_34_*_- -6

Output format for each case:

Case<underscore><case#><colon><end line>
<operand><underscore><operand><underscore><operator>
<calculated result>

=-=

Problem F: Movement of a Knight

Run Time Limit: 3 sec

A standard chess board has eight rows and eight columns. If you are familiar with chess, then you know that a knight can move from its location (i,j) in the **i** direction one or two squares and in the **j** direction one or two squares. A knight in the center of the chessboard has eight possible moves as illustrated in the following figure. However, a knight located in the corner has fewer possible moves because of the edges of the chessboard.

For this application, write a program that can compute the number of possible moves for a knight located at any location on the chessboard. The output is to be to the screen in the form of a grid-like pattern, showing the chessboard with the number of possible moves as a numeral in each chessboard location.

With a grid problem, it is necessary first to number the grid locations. For the chessboard, you should use a two number sequence (i , j) to label the board. The values of i and j increase from left to right and down to up, respectively. If the initial knight location is (i , j), then you can call the new location (i+k , j+m). A move is legal if 1<= (i+k) <= 8 and 1 <= (j+m) <= 8. You must check each location (i , j) on the chessboard for all possible combinations of **k** and **m** to see if they are feasible.

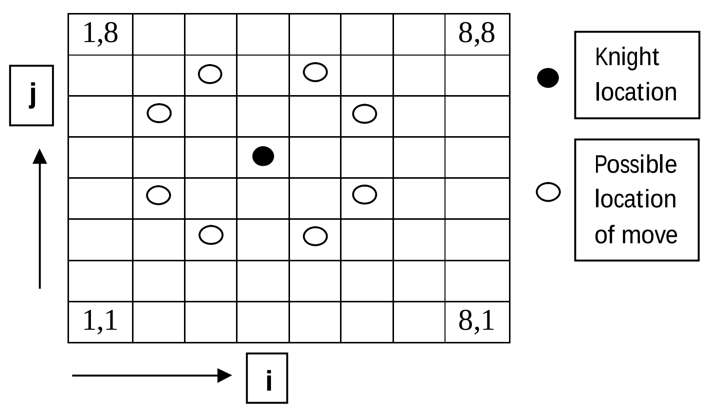


Figure : Possible movement locations of a knight in the center of a chessboard

Input Data:

Input the number of test case in the first line and the next line is to enter the number of rows and columns of the chess board you want to apply. You can use 10 by 10 or 15 by 23 chessboard.

Output Data:

Plot all possible movement locations of a knight in the chess board. Use an underscore between each locations. Please notice that the rightmost location has no underscore.

Sample Input	Sample Output
1 8 8	2_3_4_4_4_4_3_2 3_4_6_6_6_6_4_3 4_6_8_8_8_8_6_4 4_6_8_8_8_8_6_4 4_6_8_8_8_8_6_4 4_6_8_8_8_8_6_4 3_4_6_6_6_6_4_3 2_3_4_4_4_4_3_2

=-=

Problem G: The Buses

Run Time Limit: 4 sec

A man arrives at a bus stop at 12:00. He remains there during 12:00-12:59. The bus stop is used by a numbers of bus routes. The man notes the times of arriving buses. The times when buses arrive are given.

- Buses on the same route arrive at regular intervals from 12:00 to 12:59 throughout the entire hour.
- Times are given in whole minutes from 0 to 59.
- Each bus route stops at least 2 times.
- The number of bus routes in the test example will be ≤ 17.
- Buses from different routes may arrive at the same time.

Several bus routes can have the same time of first arrival and/or rime interval. If two bus routes have the same starting time and interval, they are distinct and are both to be presented.

Find the schedule with the fewest number of bus routes that must stop at the bus stop to satisfy the input data. For each bus route, output the starting time and the interval.

***Input Data*:**

The input contains a number n (n≤300) telling how many arriving buses have been noted, followed by the arrival times in ascending order.

***Output Data*:**

Write a table as an output with one line for each bus route. Each line in the file gives the time of arrival for the first bus and the time interval in minutes.

Sample Input	***Sample Output***
17 0 3 5 13 13 15 21 26 27 29 37 39 39 45 51 52 53	0 13 3 12 5 8

=-=