# Computer Science Competition
# District 2023
## Programming Problem Set

### I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

### II. Names of Problems

| Number | Name |
|---|---|
| Problem 1 | Bogdan |
| Problem 2 | Christine |
| Problem 3 | George |
| Problem 4 | Hisoka |
| Problem 5 | Janice |
| Problem 6 | Krishna |
| Problem 7 | Liza |
| Problem 8 | Miguel |
| Problem 9 | Patrick |
| Problem 10 | Shreya |
| Problem 11 | Sunil |
| Problem 12 | Vanessa |

# 1. Bogdan

**Program Name: Bogdan.java** **Input File: none**

Bogdan has a big month of January to start the new year in 2023! The month of January starts off with no class on January 2nd as part of the New Year's Celebration. There is also no class on January 16th, in observance of Martin Luther King Day. Last, but certainly not least, Bogdan's 16th birthday is on January 31st.

Bogdan attends the very prestigious Ultimate Intellectual Leadership academy, which utilizes a block scheduling system for their class schedule. Instead of attending every single one of his classes every day for a short period of time, he attends his 1st, 3rd, 5th, and 7th periods on "A" days, and he attends his 2nd, 4th, 6th, and 8th periods on "B" days. Since he only goes to each class every-other-day, each class period is twice as long, as it would have been if all eight periods would be if attended every single day.

With so much going on, Bogdan decides to write a program that will output his calendar in a spreadsheet like format. Can you help him write a program that will output his January calendar so that Bogdan always knows whether he needs to attend his "A" day or "B" day classes?

**Input:** None

**Output:** The exact calendar as shown below.

**Sample input:** None

**Sample output:**

```
-------------------------------------------------------------
                1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
-------------------------------------------------------------
S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T
U O U E H R A U O U E H R A U O U E H R A U O U E H R A U O U
N N E D U I T N N E D U I T N N E D U I T N N E D U I T N N E
-------------------------------------------------------------
N N A B A B N N A B A B A N N N B A B A N N B A B A B N N A B
o o       o o         o o o       o o         o o   i
    D D D D    D D D D D D     D D D D    D D D D D    D r
C C a a a a C C a a a a a C C C a a a a C C a a a a a C C a t
l l y y y y l l y y y y y l l l y y y y l l y y y y y l l y h
a a       a a         a a a       a a         a a   d
s s       s s         s s s       s s         s s   a
s s       s s         s s s       s s         s s   y
-------------------------------------------------------------
```

# 2. Christine

**Program Name: Christine.java** **Input File: christine.dat**

Christine is learning how to use loops in Computer Science class. Her teacher has given her a challenge problem that will demonstrate her skill with using loops. Your job is to create that program so that she may use yours as a guide should she have problems.

Your program should read in exactly six positive integers, each no larger than 32. We will call those integers A, B, C, D, E, and F.

The program will print five lines of output. Each line will consist of from 1 to 32 integers as described below.

Line 1: Print all the integers from A to B.
Line 2: Print all the integers from B to C.
Line 3: Print all the integers from C to D.
Line 4: Print all the integers from D to E.
Line 5: Print all the integers from E to F.

Her teacher gave her three situations to consider:

(1) What if the first number is less than the next.
(2) What if the first number is greater than the next.
(3) What if the two numbers are equal.

Her teacher also told her to look very closely at the sample outputs below.

**Input:** Input will consist of six integers (A, B, C, D, E, F). Each integer will be in range [1,32].

**Output:** The five lines of output will consist of all integers with integers separated by one space:
1: from A to B
2: from B to C
3: from C to D
4: from D to E
5: from E to F

**Sample input:**

```
7
3
3
12
20
5
```

**Sample output:**

```
7 6 5 4 3
3
3 4 5 6 7 8 9 10 11 12
12 13 14 15 16 17 18 19 20
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
```

# 3. George

**Program Name: George.java** **Input File: george.dat**

Your and your brother George are driving down to see your grandparents for the weekend. Oh No! George forgot to fill up the gas tank! And there are no gas stations between you and your grandparents' house! Quickly find out if you have enough gas to get to their house, or not

**Input:** The first line will contain a single integer `n (0 < n < 50)` that indicates the number of data sets that follow. Each data set will start with three integers A, B, and C, separated by spaces, denoting the average miles per gallon that your car gets at the current (undisclosed, and unnecessary for the problem) speed, the number of gallons of gas left in the car, and how many miles left between you and your grandparents' house, respectively.

**Output:** If you have enough gas to get to their house, output the string "`Never tell me the odds.`", otherwise output the string "`I've got a bad feeling about this.`".

**Sample input:**

```
4
10 15 175
10 15 100
10 10 101
10 10 100
```

**Sample output:**

```
I've got a bad feeling about this.
Never tell me the odds.
I've got a bad feeling about this.
Never tell me the odds.
```

# 4. Hisoka

**Program Name: Hisoka.java** **Input File: hisoka.dat**

Hisoka overheard his parents talking about how they had built a retirement fund by making a regular monthly payment into a retirement annuity. He wants to be able to play with various numbers to see how large the retirement fund can get. Using the formula shown below, it starts with a steady monthly payment into the retirement account. That money earns interest based on a monthly rate that is simply 1/12 of a stated annual percentage rate or APR. A higher APR means more profit from the amount that has been deposited along with monthly interest that has also been added back into the account. The number of periods is just the number of months that payments have been deposited.

Formula:

$$v = p \times \frac{(1+r)^n - 1}{r}$$

Legend:
p = amount of payment
r = monthly interest rate
n = number of months
v = annuity value

Can you help Hisoka create a program to calculate values of savings annuities based on various financial factors?

**Input:** First line will contain a positive integer **T**, the number of test cases with $1 \leq \mathbf{T} \leq 25$. The following **T** lines will contain three items separated by spaces: amount of a monthly payment, annual percentage rate (APR), and number of years. The monthly payment will be standard dollars and cents in the range \$25.00 … \$1500.00. The APR is a percentage in the range 2.0% … 15.0% and must be converted into decimal form. The number of years will be a whole number in the range 10 … 50.

**Output:** For each test case, display three items on a single line: sum of all monthly payments into the account, total interest earned, and total value with all items rounded to 2 decimal places and separated by single spaces.

**Sample input:**

```
3
25.00 12.0 30
50.00 5.75 25
1234.56 7.99 20
```

**Sample output:**

```
9000.00 78374.10 87374.10
15000.00 18346.73 33346.73
296294.40 429983.64 726278.04
```

# 5. Janice

**Program Name: Janice.java** **Input File: janice.dat**

Your best friend Janice and her friend Ariel like to send secret messages to each other, and they've taken to using a Caesar cipher to encode their messages. You need to determine what key they are using, given the encoded and real messages, and use it to decode another message. A Caesar cipher works by shifting the alphabet to the right a given amount (let's call it s), and substituting all the letters in the original message with their corresponding letter in the newly shifted alphabet.

**Input:** The first line will contain a single integer `n (0 < n < 50)` that indicates the number of data sets that follow. Each data set will consist of three lines, each with their own string of uppercase letters and spaces. The first line will be the actual message that has been sent, the second line will be the encoded version of that message, and the third line is another encoded message to be decoded.

**Output:** Output the decoded third line of the input, using the first two lines to determine the new alphabet to be used to decode the third line.

**Sample input:**

```
4
HELLO ITS ME
DAHHK EPO IA
E SWO SKJZANEJC EB WBPAN WHH PDAOA UAWNO UKQZ HEGA PK IAAP
LUKE
SBRL
P HT FVBY MHAOLY
SOMEBODY ONCE TOLD ME
EAYQNAPK AZOQ FAXP YQ
FTQ IADXP IME SAZZM DAXX YQ
ABCDEF
STUVWX
YZABCDEFGHIJKLMNOPQR
```

**Sample output:**

```
I WAS WONDERING IF AFTER ALL THESE YEARS YOUD LIKE TO MEET
I AM YOUR FATHER
THE WORLD WAS GONNA ROLL ME
GHIJKLMNOPQRSTUVWXYZ
```

# 6. Krishna

**Program Name: Krishna.java** **Input File: krishna.dat**

Krishna is intrigued by a song she heard on the radio during December.

Taking the words to the song literally:

On the first day, 1 gift was given. (1)
On the second day, 3 gifts were given (1+2)
On the third day, 6 gifts were given. (1+2+3)
On the fourth day, 10 gifts were given. (1+2+3+4)
On the fifth day, 15 gifts were given. (1+2+3+4+5)

So, if Krishna wanted to know how many gifts were received from Day #3 to Day #5, the answer would be a total of 31 gifts (6+10+15).

Help Krishna write a program to allow her to input two integers A and B representing two different days in the song. The output should be the total number of gifts received from the Day #A to Day #B.

**Input:** Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will contain two integers A and B representing two different days, both in the range [1,100] where A <= B.

**Output:** Each line of output will consist of one integer representing the number of gifts received from Day #A to Day #B.

**Sample input:**

```
5
1 3
1 5
1 12
4 6
10 20
```

**Sample output:**

```
10
35
364
46
1375
```

# 7. Liza

**Program Name: Liza.java** **Input File: liza.dat**

You and Liza have been given detention! It is your job to sort through all recent student assignment grades and print out the top three scores, mean, and median scores for each assignment.

**Input:** The first line will contain a single integer `n (0 < n < 50)` that indicates the number of data sets that follow. Each data set will begin with a line containing the name of the assignment you are currently working on (this name may contain spaces). The following line will contain an integer `m` (`5 < m < 50`), denoting how many students have grades for this particular assignment. The next `m` lines will contain a students' first and last name, followed by an integer denoting their grade, with no spaces, all separated by commas.

**Output:** For each data set, output the following: The first line will contain the name of the assignment and nothing more. The next 3 lines will be the names and grades of the top 3 scoring students on the assignment (any ties will be broken alphabetically, so if two students have the same score, the one with the alphabetically first name will be listed first. Sort by last name, then if those are equal, sort by first name. There will never be two students with the same first and last name), in the following format: `<First Name> <Last Name>: <Grade for this assignment>`. After these three lines, you will output the mean score for the assignment, formatted to 2 decimal places, and on its own line, preceded by the string "`MEAN SCORE: `". The last line of output for each case will consist of the string "`MEDIAN SCORE: `" followed by the median score for the assignment, formatted to one decimal place. Each test case should be separated by an empty line, however there should be no extra empty line at the end of the last data set.

**Sample input:**

```
3
HOMEWORK 1
20
Washington,Bobbi,85
Petrovic,James,84
George,Rahul,64
Gates,Charles,79
Davis,Jack,65
Hatley,Aayush,92
Washington,Monica,89
Targaryen,Aayush,96
Bryant,Angela,70
Chan,Monica,98
James,Benjamin,84
Ketchum,Lebron,86
Jefferson,Angela,68
Jefferson,Cameron,82
Hatley,Samantha,64
Vu,Elsa,96
Parker,Benjamin,93
Petrovic,Phyllis,70
Adams,Henry,73
Lee,Monica,73
```

***~ Sample input continues on next page (with no blank line between sections) ~***

***Liza, continued (with no blank line between sections)***

```
EXAM - DIGITAL LOGIC
15
George,Bobbi,83
Anderson,Lebron,94
Franklin,Leia,87
James,Hunter,66
Parker,Tina,98
Adams,David,64
O'neal,Angela,74
Petrovic,Samantha,75
O'neal,David,96
O'neal,Monica,61
Vu,Samantha,99
Davis,Hunter,68
Lee,Samantha,86
Davis,Samantha,77
Gates,Henry,85
MACHINE LEARNING PROJECT PRELIMINARY EVALUATION
16
Galo,Cameron,88
O'reilly,Shaquille,88
Anderson,Benjamin,63
Gates,Elsa,76
Gates,Phyllis,63
Bryant,Hunter,70
Johnson,Tina,70
Vu,Tristan,62
Targaryen,Hunter,89
Vu,Leia,77
Stroud,Phyllis,77
America,Bobbi,100
America,Hunter,67
Hatley,Angela,82
Franklin,Jack,87
Davis,Tina,90
```

**Sample output:**

```
HOMEWORK 1
Monica Chan: 98
Aayush Targaryen: 96
Elsa Vu: 96
MEAN SCORE: 80.55
MEDIAN SCORE: 83.0

EXAM - DIGITAL LOGIC
Samantha Vu: 99
Tina Parker: 98
David O'neal: 96
MEAN SCORE: 80.87
MEDIAN SCORE: 83.0

MACHINE LEARNING PROJECT PRELIMINARY EVALUATION
Bobbi America: 100
Tina Davis: 90
Hunter Targaryen: 89
MEAN SCORE: 78.06
MEDIAN SCORE: 77.0
```

# 8. Miguel

**Program Name: Miguel.java** **Input File: miguel.dat**

Miguel has dreams of creating a very cool and very challenging game involving letters and a puzzle situation. He wants to encrypt a phrase and then have the user try to determine what the original was. Below is an example:

The question might be: What did the Oz girl say to her canine?

The clue would be: `Aaaa, E'ee e eefgiii kl'mn nnn no Ooorrs stttvwy.`

The answer is: `Toto, I've a feeling we're not in Kansas anymore.`

To start this game, Miguel needs a program that will take a quote, or any list of characters, and encrypt it using the following rules:

- If a character is any non-alphabetic character, it will remain in the same position.
- All of the letters will be sorted in alphabetical order not considering the case of the letter.
- Then, the sorted letters will be placed back in the list of characters in positions where letters originally were. If a position originally held an uppercase letter, the new letter in that position will now be uppercase. If a position originally held a lowercase letter, it will again be lowercase.

Look carefully at the example above as you plan your approach.

Miguel is counting on you to write this program so he can start making his millions of dollars as soon as possible.

**Input:** Input will consist of an integer N, the number of test cases. The number of test cases will be in range [1,20]. Each subsequent line will consist of a list of characters that includes both letters and non-letters. Any printed keyboard character could be used. The length of the string of characters is in the range [1,80]

**Output:** Each line of output will consist of a list of characters. All non-letters will be in their original positions. Every position that was an uppercase letter will still hold an uppercase letter, and every position that was a lowercase letter will still hold a lowercase letter. Now, however, the original letters have been sorted.

**Sample input:**

```
5
Dog
University of Texas Interscholastic League
Lmnop1234-5678Qrstabcdefghijk
z-C-N-e=I^q
Rock-Paper-Scissors
```

**Sample output:**

```
Dgo
Aaacceeeee fg Hiiii Llnnoorrssssttt Tuuvxy
Abcde1234-5678Fghijklmnopqrst
c-E-I-n=Q^z
Acce-Ikoop-Prrrssss
```

# 9. Patrick

**Program Name: Patrick.java** **Input File: patrick.dat**

Patrick has been researching mathematical sequences for a special project and stumbled upon one called the "look-and-say" sequence. It is rather unique because it is not generated from an infinite application of a mathematical formula. Instead, the sequence is extended as if a human reader was describing the current number as a count of occurrences of each subsequence containing one specific digit. The starting number, like many numerical sequences is 1. If you were asked to describe the current number it would be one count of the digit one or "one one" which becomes the next number in the sequence 11. The new number contains "two ones" producing 21. Continuing the pattern, "one two, one one" produces 1211 then "one one, one two, two ones" produces 111221. This sequence is shown below along with several more numbers in the sequence.

1, 11, 21, 1211, 111221 as described above and sequence continues as follows:
312211 is "one three, one one, two twos, two ones"
13112221 is "one one, one three, two ones, three twos, one one"
1113213211 is "three ones, one three, one two, one one, one three one two, two ones"
31131211131221 etc.

How about starting with an arbitrary sequence of digits?

7440888 is "one seven, two fours, one zero, three eights"
17241038 is "one one, one seven, one two, one four, one one, one zero, one three, one eight"
1117121411101318 etc.

Can you help Patrick create a program to generated such sequences given an initial starting number?

**Input:** An unknown number of lines, greater than 1 and no more than 20. Each line contains 2 integers separated by a single space: N, the first number of a sequence with $0 \leq N \leq 999999999999999$ and P, the position of the desired number in the sequence with $1 \leq P \leq 25$. N will not contain a sequence of any single digit that is longer than 9.

**Output:** For each test case, display one line containing the $P^{th}$ number of the sequence. Length of resulting number will not exceed 2000 digits.

**Sample input:**
```
1 9
7440888 3
2 13
5 1
```

**Sample output:**
```
31131211131221
1117121411101318
311311222113111231133211121312211231131112311211232221l2
5
```

# 10. Shreya

**Program Name: Shreya.java** **Input File: shreya.dat**

Your friend Shreya left you to pay for lunch, but you only have the coins in your pocket! Quickly, determine whether or not you can pay for the meal given the money you have in your pocket. However, you are a cheapskate, and will only be able to pay for the meal if you can make the target amount **exactly** with the coins in your pocket. For example, if you have 2 coins of value 5, and you need to make 8 to pay for the meal, due to your cheapskate ways, you will not be able to pay for the meal, and you must dine and dash.

**Input:** The first line will contain a single integer `n (0 < n < 50)` that indicates the number of data sets that follow. Each data set will consist of a line with an unknown number of integers, each denoting the value of one of the coins in your pocket. The following line will contain a single integer denoting the target value, or the value we are trying to make.

**Output:** If you can make the correct amount, output the string "`Business as usual.`", otherwise, output the string "`Dine and Dash.`"

**Sample input:**

```
3
2 3 5
6
2 3 3 5
6
4 5 6 7
15
```

**Sample output:**

```
Dine and Dash.
Business as usual.
Business as usual.
```

# 11. Sunil

**Program Name: Sunil.java** **Input File: sunil.dat**

Sunil has been working hard to prepare for this year's UIL programming contests. He feels fairly confident about the written test and most programming techniques but is still unsure of his skills for working with 2-dimension arrays. He wants to build a program that can perform basic processing of arrays of various sizes. Nothing complicated, just basic input and simple output to confirm proper handling of data.

Sunil has requested your assistance on this project, can you help?

**Input:** First line contains a single integer **T** the number of test cases that follow with $1 \le$ **T** $\le 10$. Each test case starts with a line containing 2 integers separated by whitespace: **R**, the number of rows, and **C**, the number of columns, with both $2 \le$ **R**, **C** $\le 15$. That line will be followed by **R** lines of data with each containing **C** integers separated by whitespace with integers in (-100,100).

**Output:** For each test case, output four lines. First line contains **R** numbers, one for each row of the array which is the average of data items in that row. Second line contains **C** numbers, one for each column of the array which is the average of data items in that column. Third line contains a single number, the average of all data items. All averages must be rounded to two decimal places and displayed right-aligned in fields that are 8 positions wide with two decimal places. The last line follows previous lines with 25 equal signs "==========" starting in column 1.

**Sample input:**

```
3
5      2
11     -81
26     86
71     -23
-68    6
-62    48
3      6
40     90     -42    21     97     31
21     -28    -84    67     -85    -67
-30    -55    -36    -99    35     -22
7      7
38     26     -87    76     -29    -34    14
95     35     31     0      -71    -96    -99
-37    -70    -97    -39    56     -36    -27
53     -5     36     4      -36    97     -28
-14    31     17     78     -86    21     -29
16     98     99     93     -19    22     -43
6      -63    -89    99     -81    -41    -41
```

**Sample output:**

```
  -35.00   56.00   24.00  -31.00   -7.00
   -4.40    7.20
    1.40
==========
   39.50  -29.33  -34.50
   10.33    2.33  -54.00   -3.67   15.67  -19.33
   -8.11
==========
    0.57  -15.00  -35.71   17.29    2.57   38.00  -30.00
   22.43    7.43  -12.86   44.43  -38.00   -9.57  -36.14
   -3.18
==========
```

# 12. Vanessa

**Program Name: Vanessa.java** **Input File: vanessa.dat**

Vanessa and her fellow UIL Computer Science teammates, are looking to raise funds to buy new laptops for the upcoming regional and state competitions. Fortunately, a very gracious donor in the community has agreed to make a sizable donation to their cause, but there's a catch… The donor wants Vanessa and her team to earn every dollar received. In order to receive the funding, the donor has devised a game that must be played.

The game the donor has come up with is as follows: The donor provides the team with a set of unique numbers as well as a desired sum to be reached. It is up to the team to determine every possible way to use the unique numbers, that when added up, equal the desired sum. For every valid combination, the donor will donate one dollar to the cause. For example, suppose the donor gave the team the numbers 1, 2, and 3 as well as a desired sum of 4. There are four unique ways this sum can be achieved given the unique number set:

- 1+1+1+1=4
- 1+3=4
- 1+1+2=4
- 2+2=4

In this instance, the team would receive four dollars from the donor and is allowed to play the game again with a different set of numbers and desired sum. Vanessa and her team don't have to list out all the combinations, but instead just have to give the number of possible combinations. If at any point Vanessa and her team are unable to determine the correct number of combinations, the game is over and the donor will cease to distribute any more funds to the team. With so much on the line, Vanessa knows it's in her team's best interest to write a program that will guarantee their submission for the number of combinations to the donor is correct every time, in order to maximize the funds to be received. Can you help Vanessa and her team write such a program?

**Input:** Input will consist of an integer N, the number of test cases. N will be in range [1,30]. Each test case is made up of three lines. Line one is the unique list of integers in range [1,100]. It will be guaranteed that the list will contain at least 1 integer and no more than 10 integers. The list is separated by commas with no spaces in between. The second line of the test case is the desired sum that is to be reached. The desired sum is guaranteed to be in range [0,2000]. The third line of each test case is 20 dashes meant purely to break up each test.

**Output:** For each test case, you are to output the exact number of unique combinations that add up to the desired sum. The number of possible combinations will be guaranteed to be in range [0,2147483647].

**Sample input:**

```
5
1,2,3
4
--------------------
1,2,3,5
5
--------------------
2,5,3,6
10
--------------------
1,2,3,4,5,10
12
--------------------
2,4,6,8
7
--------------------
```

**Sample output:**

```
4
6
5
49
0
```