Computer Science Competition
District 2018
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	Alice
Problem 2	Bayani
Problem 3	Candela
Problem 4	Carla
Problem 5	Diya
Problem 6	Gleb
Problem 7	Jeremy
Problem 8	Kinga
Problem 9	Layla
Problem 10	Max
Problem 11	Nandita
Problem 12	Raymond

1. Alice

Program Name: Alice.java Input File: none

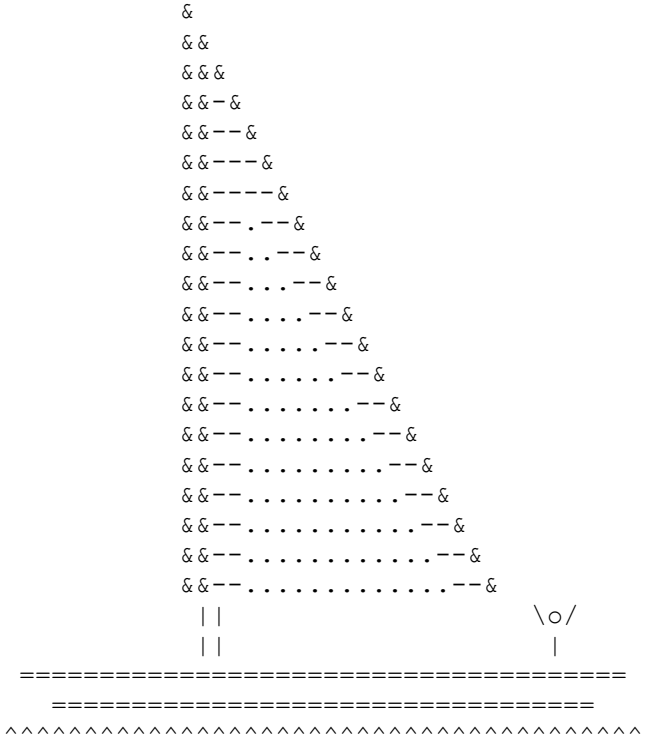
Alice is waving to you from her sailboat! Write a program to output this image, exactly as you see it.

Input: None.

Output: A picture of Alice in her sailboat, having fun out on the water.

Sample input:
None

Sample output:



2. Bayani

Program Name: Bayani.java Input File: bayani.dat

Bayani wants to printout a report of his bill expenses for the month so far and needs a simple program to do that.

Input: A list of bill expenses, each on one line of a data file, each value greater than zero and less than $1,000.00.

Output: Given a list of values, generate a listing of each value aligned and formatted exactly as shown below, and a final total at the end, with **exactly four spaces** to be allocated for the whole number portion after the $ sign for each value (no commas):

$####.##

Assumption: The final total is guaranteed to fit within the format shown above.

Sample input:

```
6.99
12.87
5.44
99.99
115.87
```

Sample output:

```
        $   6.99
        $  12.87
        $   5.44
        $  99.99
        $ 115.87
Total = $ 241.16
```

3. Candela

Program Name: Candela.java **Input File: candela.dat**

Candela's teacher, who gives very difficult tests, has announced one for next week, and has provided her class some preview information about the questions that will be on the test so that she and her classmates can strategize about how best to approach it. The information provided by the teacher only includes the question number, how difficult the question will be on a scale of 1 to 10, and how many points it will be worth if it is answered correctly, on a scale of 1 to 20. The worth of each question does not necessarily match its difficulty, so it is possible that an easy question will be worth as much if not more than a difficult question. The total number of points available will exceed the maximum score of 100 that will be awarded, so students do not have to answer every question, and will be awarded a score of 100 if they meet it exactly, or even exceed it.

What Candela and her classmates have done in the past is figured out the level of difficulty they think they can handle, based on previous efforts, and then strategize to only answer the questions that approach or meet that effort, hoping for a good score because of their work.

For example, on the last test there were 10 questions with the following point and difficulty levels:

- Q#1: 12 points, difficulty level 8
- Q#2: 10 points, difficulty level 5
- Q#3: 8 points, difficulty level 3
- Q#4: 12 points, difficulty level 4
- Q#5: 7 points, difficulty level 5
- Q#6: 13 points, difficulty level 3
- Q#7: 16 points, difficulty level 2
- Q#8: 2 points, difficulty level 8
- Q#9: 14 points, difficulty level 4
- Q#10: 4 points, difficulty level 5

Being very conscientious about their grades, Candela and her friends decide to figure out the best approach to maximize their efforts in getting a decent grade without exceeding their target difficulty level. The teacher will allow students to bring with them the information provided, along with any strategies for which questions to attempt to maximize their test score.

For the last test, the result for Candela was a score of 96, which she was able to achieve by answering all but question #8, for a total of 96 points and an accumulated difficulty level of 39. She will shoot for a 40 level of difficulty for the text next week. Carla, on the other hand had earned a test score of 85 with a combined difficulty level of 29, and therefore thinks 30 is a reasonable difficulty number for her to attempt.

Input and output descriptions and samples on next page.

(Candela – cont.)

Input: The data file will contain an initial integer Q (10 <= Q <= 30), indicating Q questions to follow. Each question data set consists of two values, an integer P representing the number of points that question is worth, an integer D indicating the difficulty level of the question. The first question is Question #1, the second is Question #2, continuing in that sequence, with the last one as Question #Q. Following the list of questions, several integers T will follow, each on one line and each representing a target difficulty level indicated by a student. There will be only one set of questions to process, but several target values after the question listing.

Output: For each data set, list the target difficulty designated, the calculated difficulty expected, the total score of the questions the student has selected to attempt and hopes to achieve, and each question on the calculated list, shown with the points and difficulty level, exactly as displayed, formatted and aligned in the examples below. Print a final "**=====**" line below each complete output.

Sample input:

```
10
12 8
10 5
8 3
12 4
7 5
13 3
16 2
2 8
14 4
4 5
40
30
10
```

Sample output:

```
Target diff     = 40
Calculated diff = 39
Expected points = 96
Q# 1, 12 pts, diff 8
Q# 2, 10 pts, diff 5
Q# 3,  8 pts, diff 3
Q# 4, 12 pts, diff 4
Q# 5,  7 pts, diff 5
Q# 6, 13 pts, diff 3
Q# 7, 16 pts, diff 2
Q# 9, 14 pts, diff 4
Q#10,  4 pts, diff 5
=====
Target diff     = 30
Calculated diff = 29
Expected points = 85
Q# 1, 12 pts, diff 8
Q# 2, 10 pts, diff 5
Q# 3,  8 pts, diff 3
Q# 4, 12 pts, diff 4
Q# 6, 13 pts, diff 3
Q# 7, 16 pts, diff 2
Q# 9, 14 pts, diff 4
=====
Target diff     = 10
Calculated diff = 9
Expected points = 43
Q# 6, 13 pts, diff 3
Q# 7, 16 pts, diff 2
Q# 9, 14 pts, diff 4
=====
```

4. Carla

Program Name: Carla.java Input File: carla.dat

In the UNIX operating system, Carla has recently learned, each file, directory, or link is "owned" by a "user", who is a member of a "group", and has certain "permissions" assigned to it, represented by a 10-character string, such as "drwxrwxrwx". The first character is 'd', '-', or 'l' (directory, file, or link), followed by three sets of "rwx" values, indicating "read, write, execute" permissions. The first set is the user's rights, the middle set the group's rights, and the third everyone else's rights to that object.

Permission denied for any of these rights is represented by a '-'in place of the 'r', 'w', or 'x'. For example, a sample directory permission string would be "drwxr--r--", indicating full directory rights for the user, but "read-only" rights for the group member and all others.

Each "rwx" combination can also be represented by an octal value (0-7), as shown below:

Octal value	r w x combination	Interpretation
0	- - -	No permission
1	- - 1	Execute permission only
2	- 1 -	Write permission only
3	- 1 1	Write and execute permission
4	1 - -	Read-only permission
5	1 - 1	Read and execute only
6	1 1 -	Read and write only
7	1 1 1	Full permission

Given a four-character code string made up of a character 'D, 'F' or 'L', followed by a 3-digit octal integer value, like 664, output the resulting 10 character string that represents the permission value indicated.

Input: Several four-character codes as described above, each on one line.

Output: The resulting 10-character string based on the criteria described above.

Sample input:

```
F664
D775
L334
F530
D127
```

Sample output:

```
-rw-rw-r--
drwxrwxr-x
l-wx-wxr--
-r-x-wx---
d--x-w-rwx
```

5. Diya

Program Name: Diya.java Input File: diya.dat

Diya has decided to take on the challenge of producing the classic spiral matrix, a series of consecutive integers starting with 1 at the top left of the square, going across the top and down the right side, around and around until the square of the side length of the square is in the very center. He needs your help to write this program. You up to the challenge?

Input: Several integers N, all on one line, with single space separation, 2 <= N < 20.

Output: For each integer, output a spiral matrix as indicated above, and shown below. All column values must be left justified, with exactly one space following the largest value in the center of the output and all other columns consistently spaced with the column containing this value. Print a final "**=====**" line below each complete output.

Sample input:

```
3
6
10
```

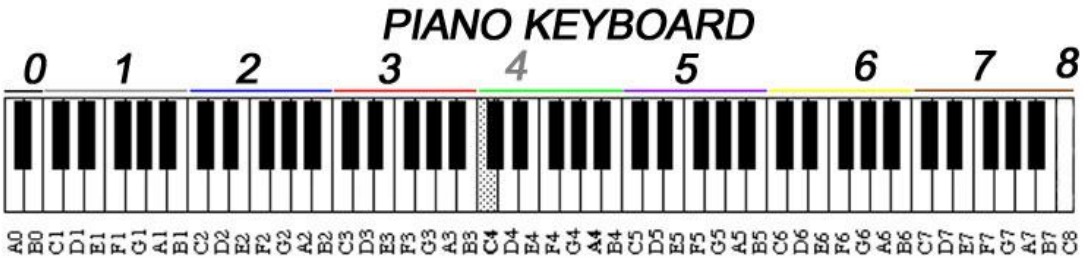
Sample output:

```
1 2 3
8 9 4
7 6 5
=====
1  2  3  4  5  6
20 21 22 23 24 7
19 32 33 34 25 8
18 31 36 35 26 9
17 30 29 28 27 10
16 15 14 13 12 11
=====
1   2   3   4   5   6   7   8   9   10
36  37  38  39  40  41  42  43  44  11
35  64  65  66  67  68  69  70  45  12
34  63  84  85  86  87  88  71  46  13
33  62  83  96  97  98  89  72  47  14
32  61  82  95  100 99  90  73  48  15
31  60  81  94  93  92  91  74  49  16
30  59  80  79  78  77  76  75  50  17
29  58  57  56  55  54  53  52  51  18
28  27  26  25  24  23  22  21  20  19
=====
```

6. Gleb

Program Name: Gleb.java **Input File: gleb.dat**

Having studied piano for several years, Gleb knows that the white keys on the standard 88-key piano are arranged in diatonic scales from C to B (C,D,E,F,G,A,B), each group of seven letters numbered from 0 to 8, starting with group zero with only two notes, A0 as the very lowest key, then B0, followed by group one with C1 through B1, group two, C2 to B2, C3 to B3, all the way to C8 as the highest key. A "C" scale starting at C3 would be C3, D3, E3, F3, G3, A3, B3, and C4 (which is middle C on the piano).



For data in a programming project he is planning, he decides to represent a melody with the starting note, followed by several positive or negative integers representing the intervals that follow that note, like 2 for the interval of "up a second", 3 for "up a third", -4 for "down a fourth", etc. The interval of "up a second" simply goes from one note to the next, like C to D, D to E, or A to B. "Up a third" would skip a letter, like going from C to E, or F to A. The rest of the interval jumps progress in the same way. *(There are further interval designations like "major", "minor", "perfect", "augmented" and "diminished", but for now we'll just stick to the basic white key intervals. Also, rhythms will not be included at this point in the project.)*

Gleb's goal is to create an alphanumeric text stream that could be interpreted by a melody function that would sound the actual notes. For example, the data for the melody, "The Eyes of Texas", would begin like this, with the actual notes produced shown below the words:

```
C4  4    -4 4   -4 4   2  2  -3
The eyes of Tex-as are up-on you
C4  F4   C4 F4  C4 F4  G4 A4 F4
```

C4 is the starting note, followed by 4 which indicates an interval of "up a fourth" to F4, -4 going back down to C4, and so on. The output for the above input would be:

```
C4 F4 C4 F4 C4 F4 G4 A4 F4
```

Input: An integer N, followed by N lines of data, each line containing a beginning "note" (letter, integer), followed by several positive or negative integers, no more than 25 notes in the entire melody.

Output: A stream of letter/number combinations representing the actual melody represented by the data. All letters must be uppercase, and at least one space must separate each "note".

Sample input:

```
C4 4 -4 4 -4 4 2 2 -3  (Melody for "The Eyes of Texas")
F5 4 2 -5 4 2 2 -3 2 2 -3 2  (Melody for "Maria", West Side Story)
C6 1 1 -4 2 1 -2 6 1 -2 1 -2  (Melody for "Old MacDonald Had A Farm")
```

Sample output:

```
C4 F4 C4 F4 C4 F4 G4 A4 F4
F5 B5 C6 F5 B5 C6 D6 B5 C6 D6 B5 C6
C6 C6 C6 G5 A5 A5 G5 E6 E6 D6 D6 C6
```

7. Jeremy

Program Name: Jeremy.java **Input File: jeremy.dat**

Jeremy knows that bitmap images can be represented by a matrix of integers, with each integer in the matrix representing the color of a "pixel". Various editing operations can be performed on a bitmap, but one of the most common ones is the flood fill. He knows that a flood fill is a change process that starts at a single pixel, changing every adjacent pixel that is the same color as the starting pixel *(not including adjacent cells in diagonal directions)*, to another color. The process continues for all the pixels that were changed, until an entire block of color in the picture has been changed. Jeremy needs your help in creating a program that, given the length and width of a bitmap, a matrix of integers from 0-9 that represents the bitmap, the starting pixel, and a "color" from 0-9 to change to, will perform a flood fill operation on that bitmap.

For example, in this 4 x 7 matrix, the color at position (2,2) is a 4, as shown in bold. If this location was changed to the color 6, and then a flood fill is performed for all neighboring values of 4, resulting in the second matrix. The remaining 4 in the top right corner stays unchanged since it is not reachable.

```
0 3 4 4 2 9 4
4 2 4 3 2 1 8
4 4 4 4 3 5 6
2 0 4 4 4 4 5

0 3 6 6 2 9 4
6 2 6 3 2 1 8
6 6 6 6 3 5 6
2 0 6 6 6 6 5
```

Input: An initial integer N, representing N data sets to follow. Each data set consists of two integers R and C, indicating an R x C matrix of integers, which follows on the next R rows. The matrix consists of single integers in the range 0-9, with single space separation. Following the matrix are two integers A and B representing the target location in the matrix, followed by an integer D as the flood fill color.

Output: The resulting matrix after the flood fill operation, which changes all instances of the color integer at location (A,B) to the color integer D, as described above and shown in the examples below. Print a final "**=====**" line below each complete output.

Sample input:

```
2
4 7
0 3 4 4 2 9 4
4 2 4 3 2 1 8
4 4 4 4 3 5 6
2 0 4 4 4 4 5
2 2
6
5 8
0 0 0 0 0 0 1 1
0 0 1 1 1 1 2 2
0 1 1 2 2 2 2 3
0 1 2 3 2 3 4 5
1 2 3 4 2 6 2 8
1 7
9
```

Sample output:

```
0 3 6 6 2 9 4
6 2 6 3 2 1 8
6 6 6 6 3 5 6
2 0 6 6 6 6 5
=====
0 0 0 0 0 0 1 1
0 0 1 1 1 1 9 9
0 1 1 9 9 9 9 3
0 1 2 3 9 3 4 5
1 2 3 4 9 6 2 8
=====
```

8. Kinga

Program Name: Kinga.java Input File: kinga.dat

Kinga knows that boolean variables are those that are either true or false, often represented as 1 or 0. In combination, two variables can have four possibilities, as shown in the table below.

```
A|B
0|0
0|1
1|0
1|1
```

She decides to write a program to output all possible combinations for up to nine variables, in table format, starting with all zeroes on the top row, and all 1s on the bottom row. She decides on the output format shown below, with A, B, C, and so on representing the variables, 0 for false, 1 for true, and the correct sequence of combinations in logical order. Columns are separated by the "|" symbol. Her input data will consist of one integer, between 1 and 9, inclusive, representing the number of boolean variables.

Input: Several integers N, 1<=N<=9.

Output: The corresponding Boolean truth table representing all possible true/false combinations, as described above and shown in the examples below. Print a final "**=====**" line below each complete output.

Sample input:

```
3
4
```

Sample output:

```
  A|B|C
1 0|0|0
2 0|0|1
3 0|1|0
4 0|1|1
5 1|0|0
6 1|0|1
7 1|1|0
8 1|1|1
=====
```

```
  A|B|C|D
1 0|0|0|0
2 0|0|0|1
3 0|0|1|0
4 0|0|1|1
5 0|1|0|0
6 0|1|0|1
7 0|1|1|0
8 0|1|1|1
9 1|0|0|0
10 1|0|0|1
11 1|0|1|0
12 1|0|1|1
13 1|1|0|0
14 1|1|0|1
15 1|1|1|0
16 1|1|1|1
=====
```

9. Layla

Program Name: Layla.java Input File: layla.dat

Layla is considering a thought experiment in measurement systems, different than metric or the traditional English system, perhaps ones that remote civilizations, or even aliens on different worlds might develop. She wants to allow for three levels of measure, like meters, kilometers, and centimeters, all members of the metric system. Just as it takes 100 centimeters to make a meter, and 1000 meters to make a kilometer, she wants to consider other systems with other conversion values.

For purposes of consistency, she decides to label the three units of measure as A, B and C, and then express the conversions as follows: B = xA and C = yB, where x and y are real values greater than 1.

In the metric system, A, B and C would be centimeters, meters and kilometers, and x and y would be 100 and 1000. To express 5 meters in terms of centimeters and kilometers, she would mathematically convert them using the values of 100 and 1000, resulting in 500 A units (centimeters), and 0.005 C units (kilometers). In her experiment, A will always be the smallest unit and C the largest.

In one possible random system, the values of x and y might be 16 and 7, which would mean that B = 16A, and C = 7B. She then would take a value, express it in one measure, and then convert it into equivalent values in the other two measures. 17 units of B would convert into 272 A units and 2.429 C units. Three C units in the same system would be equivalent to 336 A units and 21 B units.

Input: Several sets of data, each on one line, consisting of two integer values **x** and **y**, a value **d**, and a character **c**, all with single space separation.

Output: The equivalent values in all three units, A, B and C, for the given data set, rounded to three places of precision, and output as shown below. Print a final "**=====**" line below each complete output.

Sample input:

```
100 1000 5 B
16 7 17 B
16 7 3 C
10 6 4.25 A
```

Sample output:

```
A = 500.000
B = 5.000
C = 0.005
=====
A = 272.000
B = 17.000
C = 2.429
=====
A = 336.000
B = 21.000
C = 3.000
=====
A = 4.250
B = 0.425
C = 0.071
=====
```

10. Max

Program Name: Max.java **Input File: max.dat**

In ROTC class, Max has been learning how the military and other organizations use special words to represent letters of the English alphabet and the digits of the base ten number system. A special word corresponds to each symbol, each unique in its sound, created to better ensure reliable radio communication, especially when crucial information is being transmitted and received over systems that encounter noise and interference, like pilots talking to control towers, military personnel calling in air strike or rescue locations, police communicating a license plate number over the radio, ship captains communicating with other vessels while traversing the local waterways, or someone giving a credit card number over the phone for an important purchase. Over the years, many different systems have been developed, but the system shown here is the latest one adopted by NATO and used worldwide by many.

NATO Phonetic Alphabet

Phonetic Alphabet			
Alpha	Kilo	Uniform	0 Zero
Bravo	Lima	Victor	1 Wun
Charlie	Mike	Whiskey	2 Too
Delta	November	Xray	3 Tree
Echo	Oscar	Yankee	4 Fower
Foxtrot	Papa	Zulu	5 Fife
Golf	Quebec		6 Six
Hotel	Romeo	. Decimal	7 Seven
India	Sierra	. Stop	8 Ait
Juliet	Tango		9 Niner

Max has a verbal test coming up and needs to practice communicating various information using these words. He wants to write a program to input an alphanumeric string and produce the correct series of NATO phonetic words to communicate the message. Can you help?

Input: Several single alphanumeric strings, each on one line.

Output: The corresponding series of phonetic words that represent the string, all in caps, with single space separation.

Sample input:

```
ABC
DBD7555
54331234
TX1041HU
```

Sample output:

```
Alpha Bravo Charlie
Delta Bravo Delta Seven Fife Fife Fife
Fife Fower Tree Tree Wun Too Tree Fower
Tango Xray Wun Zero Fower Wun Hotel Uniform
```

11. Nandita

Program Name: Nandita.java **Input File: nandita.dat**

Nandita has learned that in some areas of the world the standard format for abbreviating a date differs from others, like the traditional month/day/year abbreviation method used often in the US. For example, in her research she has discovered that some may express **APRIL 15, 2018** as **04/15/18** (called "middle endian" format), others may use **15.04.2018** ("little endian" format), and still others **2018-04-15** ("big endian").

middle-endian (month, day, year), *e.g. 04/15/18*
little-endian (day, month, year), *e.g. 15.04.2018*
big-endian (year, month, day), *e.g. 2018-04-15*

Given a day of the year expressed fully, such as **`APRIL 15, 2018`**, show it in each of the abbreviated formats described above, in the order middle endian, little endian, big endian.

Input: Several dates fully expressed, as described above and shown in the examples below. All month names will be uppercased, fully spelled out, followed by one space, the day number, a comma and space, then the four-digit year number. Each input data set is all on one line.

Output: The given date abbreviated in three different formats: middle endian, little endian and big endian. Print a final "**`=====`**" line below each complete output.

Sample input:

```
APRIL 15, 2018
DECEMBER 7, 1941
SEPTEMBER 11, 2001
```

Sample output:

```
04/15/18
15.04.2018
2018-04-15
=====
12/07/41
07.12.1941
1941-12-07
=====
09/11/01
11.09.2001
2001-09-11
=====
```

12. Raymond

Program Name: Raymond.java **Input File: raymond.dat**

Raymond has just learned about complement values, with 12 and -13 being complements of each other, -46 the complement of 45, 8 the complement of -9, and so on. He wants to write program to output an integer value and its complement, but needs your help. Please?

Input: Several integers N, all on one line, with single space separation.

Output: The input value N, followed by a single space, followed by its complement value.

Sample input:
```
12 45 -9
```

Sample output:
```
12 -13
45 -46
-9 8
```