**Java program:** Prob05.java

**Input File:** Prob05.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

The invention of e-reading technology on computers, cell phones, tablets, and e-readers is revolutionizing the book industry. In years past, there were a handful of publishers capable of bringing a book to publication. Now, almost anyone with a computer can become an independent publishing house. As the number of publishers grow, so too will the number of books. This is beneficial to authors and consumers, but can be problematic for standards organizations that are charged with maintaining order among industries.

A book’s ISBN (International Standard Book Number) is a way to uniquely identify a work while simultaneously providing information about that work. ISBNs currently have 13 digits broken down into the following sections:

1. A 3 digit prefix assigned by the GS1 standards organization (currently 978 or 979, but they could be anything).
2. A registration group number that varies from 1 to 5 digits. This number identifies the region or language of the work.
3. A registrant element, which identifies a specific publisher.
4. A publication element, which identifies a work published by the publisher.
5. A single check digit (used to verify the legitimacy of the ISBN).

The check digit is assigned in such a way that taking the sum of all 13 numbers, each multiplied by a factor alternating between 1 and 3, is a factor of 10. For example, for ISBN 978-0-306-40615-7:

9\*1 + 7\*3 + 8\*1 + 0\*3 + 3\*1 + 0\*3 + 6\*1 + 4\*3 + 0\*1 + 6\*3 + 1\*1 + 5\*3 + 7\*1

= 9 + 21 + 8 + 0 + 3 + 0 + 6 + 12 + 0 + 18 + 1 + 15 + 7

= 100

Since 100 is a factor of 10, this is a valid ISBN number. There is no standard format for an ISBN. ISBNs can contain dashes or other separating characters (spaces, underscores, etc.) that can make them more readable by humans, or they can just be 13 digits in a row. Your job is to validate a list of ISBN candidate numbers.

**Program Input**

The file Prob05.in.txt will contain a list of ISBN numbers, one per line, which you must validate.

**Example Input:**

978-0-306-40615-7

9788175257665

1234567890123

**Program Output**

For each ISBN number your program should output one of the following:

* If the ISBN is valid, output the word VALID
* If the ISBN is invalid, output the value that the check bit should have been to make the ISBN valid

**Example Output:**

VALID

VALID

8

**Java program:** Prob06.java

**Input File:** Prob06.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Everyone played connect four as a kid, right? Well, you might call this problem “multiply four”.Given a square grid of non-negative integers, you must find the greatest product of any four adjacent numbers in any direction (up, down, diagonally, left, right). All numbers in the grid will be less than 100.

**Program Input**

The file Prob06.in.txt will contain a square grid of integers. Numbers on the same line will be separated by a single space.

**Example Input:**

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08

49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00

81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65

52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91

22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80

24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50

32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70

67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21

24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72

21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95

78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92

16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57

86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58

19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40

04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66

88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69

04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36

20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16

20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54

01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

**Program Output**

Your program should print out the greatest product of any four adjacent numbers in the grid using the format shown below. The solution is highlighted above for clarity.

**Example Output:**

Greatest product: 70600674

**Java program:** Prob07.java

**Input File:** Prob07.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Everything in a computer boils down to ones and zeroes – things we call bits. More accurately, we think of a bit represented by a 1 to be considered “on”, and that is measured by the presence of an electric charge. We think of a bit represented by a 0 to be considered “off”, and that is measured by the lack of an electric charge. Today we have computers that have billions of bits that we can put together to form useful information – information that can be readily consumed by untrained users (like text on a screen). But what if there were only a single bit?

Back in 1836, Samuel Morse needed a way to transmit messages using only electric current. The problem: how to represent text using only an “on” or an “off” signal at a single node. The answer: the length of time that the bit was on or off would have to mean something. The product: Morse code! Here is how Morse code works:

* The building blocks of Morse code are the “dot” and the “dash”. A dot is an electric signal with a length of one time unit. A dash is an electric signal with the length of three time units.
* The space between dots and dashes within a single letter has a length of one time unit.
* The space between letters in the same word has a length of three time units.
* The space between two words has a length of seven time units.

Your task has two parts: first, read in some text and turn it into Morse code. Second, read in some Morse code and decode it into text. The last page of this problem has a translation table for you to use when coding your solution.

**Program Input**

The file Prob07.in.txt will contain the following:

* Some number of lines of text that you are to read in and change to Morse code. Line breaks should be preserved, so your output should have the same number of lines that the input text has. There will be only text – no numbers, and no special characters or punctuation.
* A line reading END OF TRANSMISSION which will serve as the break point between the text and the Morse code that you are to interpret.
* Some number of lines of Morse code that you are to read in and change to plain text. Again, line breaks should be preserved.
* A line reading END OF TRANSMISSION which will denote the end of the file

**Example Input:**

It is I

Code Quest

END OF TRANSMISSION

===\_=\_===\_=\_\_\_===\_===\_===\_\_\_===\_=\_=\_\_\_=\_\_\_\_\_\_\_===\_===\_=\_===\_\_\_=\_=\_===\_\_\_=\_\_\_=\_=\_=\_\_\_===\_\_\_\_\_\_\_=\_===\_=\_\_\_=\_=\_===\_\_\_=\_===\_=\_=\_\_\_=\_\_\_=\_=\_=

=\_=\_\_\_\_\_\_\_=\_=\_=\_=\_\_\_===\_===\_===\_\_\_=\_===\_===\_=\_\_\_=\_\_\_\_\_\_\_===\_=\_===\_===\_\_\_===\_===\_===\_\_\_=\_=\_===\_\_\_\_\_\_\_===\_===\_=\_\_\_=\_=\_===\_\_\_===\_=\_===\_===\_\_\_=\_=\_=\_\_\_\_\_\_\_=\_===\_=\_=\_\_\_=\_=\_\_\_===\_=\_===\_\_\_=\_\_\_\_\_\_\_=\_===\_===\_=\_\_\_=\_=\_\_\_===\_===\_=\_=\_\_\_===\_===\_=\_=\_\_\_=\_===

END OF TRANSMISSION

**Program Output**

Because Morse code deals only with the presence or absence of an electrical current, you will use the following when translating between text and Morse code:

* The presence of an electrical current will be represented by an equal sign (=).
* The absence of an electrical current will be represented by an underscore (\_).

Your output should mimic the input file by having the following four elements:

* The Morse Code representation of the text from the input file
* A line containing the phrase END OF TRANSMISSION
* The text representation of the Morse Code from the input file (all in lowercase)
* A line containing the phrase END OF TRANSMISSION

Make sure you take a look at the electronic copy of this input and output – the Morse Code sections are too long to fit on one printed line, so they are wrapping here in the printed packet.

**Example Output:**

=\_=\_\_\_===\_\_\_\_\_\_\_=\_=\_\_\_=\_=\_=\_\_\_\_\_\_\_=\_=

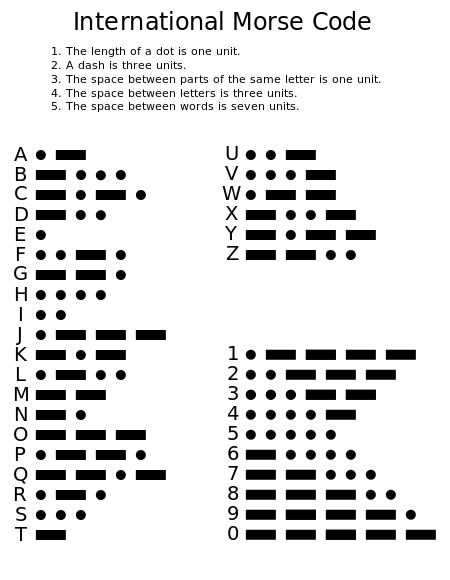
===\_=\_===\_=\_\_\_===\_===\_===\_\_\_===\_=\_=\_\_\_=\_\_\_\_\_\_\_===\_===\_=\_===\_\_\_=\_=\_===\_\_\_=\_\_\_=\_=\_=\_\_\_===

END OF TRANSMISSION

code quest rules

i hope you guys like pizza

END OF TRANSMISSION



**Java program:** Prob08.java

**Input File:** Prob08.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Have you ever seen a spray paint artist? Their art depends on layering – they spray lots of colors on top of each other and then scrape paint away to make beautiful scenes. In this problem, you will deal with layered rectangles in a workspace grid. Your art will involve these rectangles layered on top of each other.

Lockheed's stealth technology has influenced the world of rectangle art by providing rectangle artists with a mysterious material that turns clear when there are even numbers of layers of that material touching each other. So, any square in the grid that is covered by an odd number of rectangles appears covered, while squares in the grid with an even number of rectangles covering a space appear to be uncovered.

Your task is to write a program that will compute what a workspace will look like after placing these special rectangles down. All your inputs will be given in Cartesian format.

**Program Input**

The file Prob08.in.txt will contain the following:

* The first line of the file will be the workspace area in X,Y format – meaning that the workspace area is X units wide and Y units tall. The origin will be in the bottom left corner of the workspace, so all coordinates will be positive. The numbers will be separated by a single comma.
* The rest of the lines in the file will describe the location of the rectangles in the workspace. Each line will contain coordinates for two points that will be the diagonal corners of the rectangle. The x and y coordinates for each point will be separated by a single comma, and there will be a single space between the two coordinates.

**Example Input:**

20,20

0,10 10,20

20,9 10,20

0,0 20,9

4,13 6,15

14,15 16,13

3,6 4,8

4,5 5,7

5,4 6,6

6,4 7,5

6,4 14,3

13,4 15,5

14,5 16,6

15,6 17,7

17,8 16,7

0,9 10,10

0,20, 20,0

0,0 20,1

0,19, 20,20

0,1 1,19

19,1 20,19

**Program Output**

Your output should produce a grid where a space means a spot looks clear and an asterisk means a spot looks covered.

**Example Output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* \*

\* \*

\* \*

\* \*\* \*\* \*

\* \*\* \*\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \* \* \*

\* \*\* \*\* \*

\* \*\* \*\* \*

\* \*\* \*\* \*

\* \*\*\*\*\*\*\*\* \*

\* \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*