TeamsCode Fall 2019 MIHS Programming Contest Problem Set

Rules:

1. Each question is worth 50 points. With each incorrect submission, the value of that question decreases by 5 points. You do not need to solve the problems in order.
2. The internet may not be accessed during the contest coding phase. Only one computer per team can be out during this time, meaning phones also need to be put away.
3. The team with the highest score at the end of the contest wins. In the case of a tie, the team that made their final successful submission first will be the winner.

Notes:

1. Assume all input and output is encoded in UTF-8 using UNIX (LF) line endings.
2. Your computer must be able to read from USB flash drives formatted to FAT32.

Problems:

1. Logo
2. Missing Prices
3. Spanish Conjugations
4. Fancy Border
5. Unit Conversion
6. Itoa
7. FizzBuzzBloop
8. Note Sorting
9. Coded Message
10. Mixing
11. Stitching
12. Esoteric
13. Markdown
14. Algebra
15. A Hike

1. Logo
Input File: logo.txt

For *Logo*, print out the TeamsCode shown below. There is no input in this problem, but your logo must match **exactly** with the one shown, down to the number of spaces.

Input:

None.

Output:

Output the logo exactly as shown below.

Example Output:

```
    /|
  _| |  _____
 \_  | |  __/
   | | | |__
   |_| |____\
  'TeamsCode'
```

2. Missing Prices
Input File: missingPrices.txt

There has been data corruption in the price database at Conosco Corp. Management wants you to extrapolate the missing values from the known ones. Each data entry has three values: the base price of the item, the sales tax rate (in percent form), and the resulting price. The resulting price will be the base price of the item plus tax, which can be found by multiplying the tax rate and the base price.

Input:

The first line contains an integer, N. The following N lines contain two integers and an 'X'. These may be in any order, but they signify base price of the item, the sales tax rate, and the resulting price respectively.

Output:

For each line, output the value of the missing value which would satisfy the rules described above. Round decimals to the nearest hundredth.

Example Input:

```
3
X 7 150
10 10 X
10 X 20
```

Example Output:

```
140.19
11.00
100.00
```

3. Spanish Conjugations
Input File: spanishConjugations.txt

Your Spanish teacher wants you to write a program that conjugates present tense, regular verbs. Regular verbs in Spanish can be conjugated into five forms: yo, tú, el, nosotros, and ellos. Additionally, verbs are conjugated by the last two letters of the verb, which will either be -ar, -er, or -ir. To conjugate a verb replace the last two letters with the correct ending. Consult the Spanish verb conjugation guide below to find the correct ending:

	-ar	**-er**	**-ir**
yo	-o	-o	-o
tú	-as	-es	-es
el	-a	-e	-e
nosotros	-amos	-emos	-imos
ellos	-an	-en	-en

Input:

The first line of the file contains the integer, N. The following N lines are each an uncapitalized, unconjugated verb. Verbs can have -ar, -er or -ir endings.

Output:

Output the regular conjugation, according to the table above, in the yo, tú, el, nosotros, and ellos forms, in that order, with each conjugation on a new line. Every conjugation should start with a capital letter.

Example Input:

```
3
caminar
comer
vivir
```

Example Output:

```
Camino
Caminas
Camina
Caminamos
```

Caminan
Como
Comes
Come
Comemos
Comen
Vivo
Vives
Vive
Vivimos
Viven

4. Fancy Border
Input File: fancyBorder.txt

You must put a fancy border on some text for a poster. You are given several lines of text, each up to 9 characters long, and must put a fancy border around those lines.

Input:

The first line contains an integer, N. The following N lines will each contain a string that is less than 9 characters long.

Output:

First, output the top of the fancy border ("`\--******--/`"). Then, output each line of text so that there is border ("`|`") before and after each inputted string and the borders lines up, with exactly 9 spaces in between the borders. For example, If a line is 3 characters long it must be printed before 6 spaces. If a line is 5 characters long, it must be printed before 4 spaces. Finally, output the bottom of the fancy border ("`/--******--\`").

Example Input:

```
5
Hello
Test1
Testing
TestThree
Goodbye
```

Example Output:

```
\--******--/
|Hello    |
|Test1    |
|Testing  |
|TestThree|
|Goodbye  |
/--******--\
```

5. Unit Conversion
Input File: unitConversion.txt

Your program must convert a given number of centimeters or inches into kilometers, meters, and centimeters. A meter is 100 centimeters, and a kilometer is 1000 meters. 1 inch is 2.54 centimeters.

Input:

The first line contains an integer, N. The following N lines each contain a number of centimeters or inches. A line containing centimeters will start with "`c`" (i.e. "`c 90`"), and a line containing inches will start with "`i`" (i.e. "`i 90`"). The number following the "`c`" or "`i`" is the number of centimeters or inches.

Output:

Output the given measurements in kilometers, meters, and centimeters respectively, each separated by a space. The number of centimeters must be rounded to the nearest centimeter.

Example Input:

```
4
c 126
c 225335
i 50
i 93
```

Example Output:

```
0 1 26
2 253 35
0 1 27
0 2 36
```

6. Itoa
Input FIle: itoa.txt

You are tasked with converting numbers from one base to another base.

Input:

The first line contains an integer, N.. The following N lines in the file contain three integers separated by spaces. The first integer is the base which the number must be outputted in. The second integer is the base which the number is currently in. The third column is the number which must be converted.

Output:

Output the number in the specified base. The numerals that must be used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, where A through F represent the values 11 through 16 in base 10. For example, the number 10 in base 10 is represented by the letter A in base 16 (hexadecimal), and number 11 in base 10 is represented by the letter B in base 16.

Example Input:

```
4
2 8 5
16 10 464367618
2 16 24
15 12 1B
```

Example Output:

```
101
1BADB002
100100
18
```

7. FizzBuzzBloop
Input File: fizzBuzzBloop.txt

Your program must output a series of numbers in a certain range according to a set of rules given to you by the input. Each rule will consist of an integer followed by a word. Every number that is a multiple of the integer in a rule must be replaced by the word in that rule. If it is a multiple of several numbers, it must be replaced with all applicable words in the order in which the rules are given. With the rules given in the example, the number 3 would be "`Fizz`" because 3 is a multiple of 3, and the number 15 would be "`FizzBuzz`" because 15 is a multiple of both 3 and 5, but the rule describing the replacement of multiples of 3 comes first.

Input:

The first line contains the starting integer. The second line contains the ending integer. The third line contains an integer, N. The following N lines each contain one rule, following the format described above.

Output:

Output the numbers between the starting and ending integer, both inclusive, after they have been changed to fit the rules.

Example Input:

```
2
21
3
3 Fizz
5 Buzz
7 Bloop
```

Example Output:

```
2
Fizz
4
Buzz
Fizz
Bloop
8
Fizz
Buzz
11
Fizz
```

```
13
Bloop
FizzBuzz
16
17
Fizz
19
Buzz
FizzBloop
```

8. Note Sorting
Input File: noteSorting.txt

You are asked to convert music notes in their letter form to their frequency and sort them. The notes in an octave from lowest to highest are given in the chart below, and each note is separated from the next by a half-step. After the note B, the next octave starts, and below the note C, is the note B one octave lower.

C	C#	D	D#	E	F	F#	G	G#	A	A#	B

Frequency is calculated using this formula: $f_n = 440 * (1.05946)^n$ where n is equal to the half-steps above A_4. If the note is below A_4, n should be negative.

Input:

The first line contains an integer, N. The following N lines each will describe a note and have an integer and a string without spaces after. The integer defines the octave of the note, and string defines which note it is.

Output:

Output the frequencies (rounded to integers) of the inputted notes sorted from lowest to highest.

Example Input:

```
7
4 A#
2 C#
8 D
3 A#
7 A#
0 C#
2 D
```

Example Output:

```
17
69
73
233
466
3729
4698
```

9. Coded Message
Input File: codedMessage.txt

You are given a code for each letter of the alphabet in a file. Given an integer which shifts each letter's designated coded value line by a set amount, you must take a line of input and decode it from the code to a hidden message.

Input:

The first 28 lines of a file represent the encryption. The first 26 of the lines will each contain a number that represents a letter, in ascending order from "a" to "z"; the first number will represent "a", and the last represents "z". The next 2 lines will contain numbers that represent a period (".") and a space respectively. The 29th line will contain a message written in the code contained in the first 28 lines. Each number that represents something on the 29th line will be separated by a space.

Output:

Output the decoded message, where each number in the message is replaced by the letter it represents.

Example Input:

```
1
2
4
5
3
6
7
9
8
12
11
10
13
14
15
16
17
18
19
20
21
```

```
22
23
24
25
26
27
28
9 3 10 10 15 28 23 15 18 10 5 27
```

Example Output:

```
hello world.
```

10. Mixing
Input FIle: mixing.txt

You are testing how several chemicals mix together. The volume of the mixing container is limited, and you must decide which liquids to mix together. The mixing container is a perfect rectangular prism. Given the volume, as well as a "value score" (which will be a number), you must figure out the highest combined value of liquids that can fit into the mixing container when poured in. As the chemicals are very volatile, only entire containers of liquid can be poured in.

Input:

The first line contains 3 integers (separated by a space), representing the length, width, and height of the mixing container hold respectively. The second line contains an integer X and an integer Y, separated by a space. The following X sets of Y lines are each their own set of liquids which the highest combined score must be figured out for. There will be no blank line between sets. Every liquid in each set will be formatted as such: "volume value" (ex "`6 10`")

Output:

Output X lines, each giving the highest possible score for the corresponding item set in the form of a single integer, where the total volume of liquids poured in must not exceed the volume of the mixing container and only entire containers of liquid are poured in.

Example Input:

```
2 5 3
3 3
8 10
15 15
20 13
8 7
12 10
5 5
18 6
6 10
4 8
```

Example Output:

```
25
22
24
```

11. Stitching
Input File: stitching.txt

Your bed sheet has ripped, and you have decided to stitch various brightly-colored pieces of fabric you have found onto the material for your new bed sheet. The new bed sheet is a perfect rectangle, as is each material. You want your new bed sheet to be as colorful as possible, and so must find the greatest amount of fabric you can fit onto it.

Input:

The first line contains an integer, W and an integer, H, separated by a space. These integers represent the width and height of the bed sheet you want to cover. The second line contains an integer X and an integer Y, separated by a space. The following X sets of Y lines are the sets of fabric. Every fabric in each set is formatted “height width” (separated by a space). The sets will not be separated by a blank line.

Output:

Output the maximum area which can be covered by the scraps for each case. The scraps may be rotated and placed in any way. But fabrics cannot overlap, and all of the fabric must be on the bed sheet.

Example Input:

```
5 10
3 3
2 2
2 5
5 7
1 20
4 10
1 10
4 8
2 10
4 6
```

Example Output:

```
45
50
32
```

12. Esoteric
Input File: esoteric.txt

You are tasked with writing an interpreter for an esoteric programming language, which follows the syntax below:

>	increment the data pointer (to point to the next cell to the right).
<	decrement the data pointer (to point to the next cell to the left).
+	increment (increase by one) the byte at the data pointer.
-	decrement (decrease by one) the byte at the data pointer.
.	output the ascii character of the value stored in the byte at the data pointer.
[	if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump forward to the command after the matching] command.
]	if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump back to the command after the matching [command.

Memory is made out of cells, each one byte (unsigned 8-bit data type) in size; so when the value of a memory cell is greater than 255 (2^8), it wraps back to zero. There are an infinite amount of cells in memory. The data pointer is the selected memory cell. All memory cells begin at 0, and moving the data pointer does not change the value of the memory cells.

Input:

There will be on line of input, which is a program in the esoteric programming language Each character in the line is a command in this programming language as defined above, and any character which is not in the language definition should be ignored. The language starts from the left most character and moves to the right most character.

Output:

Output whatever would be produced by the “.” (output) statements in the program.

Example Input:

```
+[-[<<[+[--->]-[<<<]]]>>>-]>-.---.>..>.<<<<-.<+.>>>>>.>.<<.<-.end
```

Example Output:

```
hello world
```

13. Markdown
Input File: markdown.txt

Markdown is a markup language which is made up of simple equivalents to the more complicated HTML tags. For example, "`*word1*`" represents "`<b>word1</b>`". You are tasked with writing a converter from Markdown to HTML. Below is a conversion table from Markdown to HTML.

Markdown	**Description**	**HTML**
`*bold*`	Bold Text	`<b>bold</b>`
`_italic_`	Italic Text	`<i>italic</i>`
`[Link Text](http://linkurl.test)`	Link	`<a href="http://linkurl.test">Link Text</a>`
`![Image Alt Text](image.png)`	Image	`<img src="image.png" alt="Image Alt Text">`
`=Header=`	Header	`<h1>Header</h1>`
`==Subheader==`	Sub-header	`<h2>Subheader</h2>`

`* List Item 1`
`* List Item 2`
`* List Item 3`
	Unordered List	`<ul>`
		`<li>List Item 1</li>`
		`<li>List Item 2</li>`
		`<li>List Item 3</li>`
		`</ul>`

`::Annoying Blinking Text::`	Blinking Text	`<blink>Annoying Blinking Text</blink>`

For all the tags besides the list tags, there will not be a space between the tag and the content, which means the bold tag and the list tag can be distinguished by whether there is a space after the tag.

Input:

The input is a non-standard markup document, which is composed of the tags above. Markdown tags can be combined. For example, "`*_bold'n'italic_*`" produces both bold and italic text; the resulting HTML would be "`<b><i>bold'n'italic</i></b>`".

Output:

Output the inputted document converted from markdown to HTML. All line breaks should be preserved. No extra padding or indentation should be added. For the img element (<img>), the src attribute will always come before the alt attribute. This means that `<img src="x.png" alt="x">` is correct while `<img alt="x" src="x.png">`.

Example Input:

```
![Banner](banner.gif)
=My Epic Webpage=
Welcome to my website!
It is very cool and good.
==About My Webpage==My webpage is so *_Epic_*
it doesn’t need an explanation.
==My Favorite Things==
* My webpage
* My webpage
* ::My webpage but it’s blinking::
==Things I Do Not Like==
* Green eggs and ham
* I do not like them [Sam I Am](http://samiam.test).
```

Example Output:

```
<img src="banner.gif" alt="Banner">
<h1>My Epic Webpage</h1>
Welcome to my website!
It is very cool and good.
<h2>About My Webpage</h2>My webpage is so <b><i>Epic</i></b>
it doesn’t need an explanation.
<h2>My Favorite Things</h2>
<ul>
<li>My webpage</li>
<li>My webpage</li>
<li><blink>My webpage but it’s blinking</blink></li>
</ul>
<h2>Things I Do Not Like</h2>
<ul>
<li>Green eggs and ham</li>
<li>I do not like them <a href="http://samiam.test">Sam I Am</a>.</li>
</ul>
```

14. Algebra
Input File: algebra.txt

You are tasked with solving for "x" in each equation.

Input:

The first line contains an integer, N. The following N lines are each an algebraic equation. The possible operations in the equation are addition ("+"), subtraction ("-"), multiplication ("*") and division ("/"). All equations follow this order of operations: multiplication and division, then addition and subtraction. "x" appears one time in each equation; an equation like "`x * 2 + x = 15`" will not appear. All numbers will be positive and whole, and "x" will be a whole number.

Output:

Output the value of "x" in each equation.

Example Input:

```
4
X = 2 * 2
90 * 4 / 2 = 100 - x
4 = x / 2 + 1
10 * 4 + 5 + 10 = x * 5
```

Example Output:

```
4
-80
6
11
```

15. A Hike
Input File: ahike.txt

Mike likes to hike, but he hates to walk the same path or hike twice. So every single day, Mike tries to find a new path in his neighborhood. In *A Hike*, given a network, you must determine how many distinct hikes, which can be taken without repeating a path, there are.

Input:

The first line will contain integer N. The N following lines will each contain two different characters, which defines a path. So "`a b`" means there is a path from point `a` to point `b`. The paths together form the network which Mike can walk.

Output:

Output the number of distinct hikes, which is a collection of paths, through the network that Mike can take, given that: he can start from any point and end at any point, he doesn't have to cross every path between two points or visit every point, he cannot walk the same path twice, and the hike from `a` to `b` to `c` to `d` to `b` is the same as `a` to `b` to `d` to `c` to `b` and `b` to `c` to `d` to `b` to `a`.

Example Input:

```
4
a b
b c
c d
b d
```

Example Output:

```
13
```