**Problem 1 – Melons and Watermelons**

|  |  |
| --- | --- |
| **Week day** | **Amount of melons and watermelons** |
| Monday | 1 Watermelon |
| Tuesday | 2 Melons |
| Wednesday | 1 Watermelon and 1 Melon |
| Thursday | 2 Watermelons |
| Friday | 2 Watermelons and 2 Melons |
| Saturday | 1 Watermelon and 2 Melons |
| Sunday | Didko takes a break |

Didko likes to eat melons and watermelons **almost** every day of the week. Some days he eats only melons, some days only watermelons and some days he eats both. Every day of the week he eats different amount of melons and watermelons. The table on the right shows how many watermelons and melons he eats in each day of the week. You will be given a starting day of the week (1-7) when Didko starts eating, and the amount of days he is eating. Didko wants to know whether he has **eaten more melons**, **more watermelons** or **equal amount of watermelons and melons** for the given amount of days. He is not good at counting so he needs your help.

**Input**

The input data should be read from the console.

* At the **first line** comes aninteger number **s,** specifying the **starting day of the week**.
* At the **second line** comes an integer number **d** specifying the amount of sequential days Didko is eating melons and watermelons.

The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

The output should be printed on the console. It should consist of **exactly 1** line:

* Print “**{0} more watermelons**” if the eaten watermelons are more than the melons.
* Print “**{0} more melons**” if the eaten melons are more than the watermelons.
* Print “**Equal amount: {0}**” if the eaten melons and watermelons are the same amount.

**Constraints**

* The starting day **s** will be an integer in the range [1…7].
* The amount of days **d** will be an integer in the range [0…100 000].
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  3 | 2 more watermelons | The first input shows that Didko starts on the **third** day of the week: Wednesday. He eats in 3 consecutive days. Wednesday: **1w** + **1m**;Thursday: **2w**;Friday: **2w** + **2m**. In those 3 days he has eaten **5 watermelons** and **3 melons.** The output shows that he has eaten **5 - 3 = 2** **more watermelons** than melons. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 7  7 | Equal amount: 7 | 5  6 | 2 more melons | 7  23560 | 1 more watermelons |

**Problem 2 – Letters, Symbols, Numbers**

You are given **N strings.** Every string consists of letters, symbols, numbers and whitespace. All letters (**a-z**) and (**A**-**Z**) have values corresponding to their position in the English alphabet \* 10 (**a = 10, A = 10, b = 20, B = 20, …, z = 260, Z = 260**). All symbols (like **`~!@#$%^&\*()\_+{}:"|<>?-=[];'\,./**) have a fixed value of **200** and all numbers have the value of their digit \* 20 (**0 = 0**, **1 = 20**, **2 = 40**, **3 = 60**, …, **9 = 180**). The whitespace is ignored. Your task is to calculate the **sum** of all **letters**, all **symbols** and all **numbers** from the input and print them, each at a separate line.

**Input**

The input data should be read from the console.

* At the **first line** an integer number **N** is given, specifying the count of the input **strings**.
* Each of the next **N lines** holds an **input string**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

The output should be printed on the console. It should consist of **exactly 3** lines:

* On the **first** **line** print the **sum** of all **letters**.
* On the **second** **line** print the **sum** of all **numbers**.
* On the **third** **line** print the **sum** of all **symbols**.

**Constraints**

* **N** is an integer between in range [0… 1000].
* All of the strings combined have a maximal length of 255 000 characters.
* All characters, which are not Latin letters or whitespace, are considered **symbols**.
* The whitespace are the symbols '', '**\t**', '**\r**' and '**\n**'.
* Allowed working time for your program: 0.2 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 aaa.zzz.123 | 810  120  400 | The string has 6 letters: 3\*a + 3\*z = 3\*10 + 3\*260 = 30+780 = 810; 3 numbers: (1+2+3) \* 20 = 120 and 2 symbols (dots): 2 \* 200 = 400. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  SoftUni – practical  software development  training + job! | 5440  0  600 | 4  374687697641  %^^&%&^&#^$&%&\*  9997557.,.,.,  %^6^44^&\*$\_+\_ | 0  2660  6200 |  | 10 The Software  University (SoftUni)  provides quality  practical education,  profession and a job  for thousands young  people who become  skillful software  engineers.  -- Nakov, July 2014 | 18130  140  1400 |

**Problem 3 – Panda Scotland Flag**

Stoyan, a little Panda from Pleven's zoo has a great dream: to visit **Scotland** someday. Because of the budget limitations, it is not likely that Stoyan will visit Scotland soon, and he knows this, so Stoyan is playing every day by trying to draw the flag of Scotland in the sand, and of course, incorrectly. Once at the zoo a visitor left a tablet in the Stoyan's cage but without Internet. The only things found in the tablet were Nakov's C# lessons and a carefully installed C# development environment. Stoyan started learning C# - he has nothing else to do with a tablet without Internet. Now Stoyan is trying to write a program to print a special version of the Scotland's flag at the console.

Help the little Panda Stoyan to write a program that prints at the console the Scotland's flag of size **N**, following the examples below.

**Input**

The input data should be read from the console. It consists of a single line holding an **integer number N** – the size of the flag. The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

Print at the console the Scotland's flag (Stoyan's version) like at the examples below.

**Constraints**

* The input number **N** always will be **odd** positive integer number [1…501].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3 | A#B  -C-  D#E |  | 5 | A###B  ~C#D~  --E--  ~F#G~  H###I |  | 9 | A#######B  ~C#####D~  ~~E###F~~  ~~~G#H~~~  ----I----  ~~~J#K~~~  ~~L###M~~  ~N#####O~  P#######Q | 21 | A###################B  ~C#################D~  ~~E###############F~~  ~~~G#############H~~~  ~~~~I###########J~~~~  ~~~~~K#########L~~~~~  ~~~~~~M#######N~~~~~~  ~~~~~~~O#####P~~~~~~~  ~~~~~~~~Q###R~~~~~~~~  ~~~~~~~~~S#T~~~~~~~~~  ----------U----------  ~~~~~~~~~V#W~~~~~~~~~  ~~~~~~~~X###Y~~~~~~~~  ~~~~~~~Z#####A~~~~~~~  ~~~~~~B#######C~~~~~~  ~~~~~D#########E~~~~~  ~~~~F###########G~~~~  ~~~H#############I~~~  ~~J###############K~~  ~L#################M~  N###################O |

**Problem 4 – Morse Code Numbers**

"**Morse code**" is a method of transmitting text information as a series of on-off tones / lights / clicks / etc. All symbols are represented by “**.**” (**dots**) and “**-**“ (**dashes**).

You are given a 4-digit number **n** (1000 ≤ **n** ≤ 9999). First, you have to calculate the sum of all digits of the number **n** called **nSum.** Write a program to **generate all sequences of 6 numbers** in the range 0…5**,** represented by their Morse code encodings **(0 = “-----”, 1 = “.----”, 2 = “..---”, 3 = “...--”, 4 = “....-”, 5 = “.....”),** such that **the product of these 6** numbers is **equal** to **nSum.** This product is called **morseProduct**. Put “**|**” (**pipe**) as a separator after each Morse code digit. These sequences of strings are called **“Morse code numbers”**. See the examples below for better understanding.

**Input**

* The input data should be read from the console.
* The number **n** stays at the first line.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

The output should be printed on the console as a sequence of strings (Morse code numbers), each at a separate line. The **order** of the output lines is not important. In case no **Morse code numbers** exist, print “**No**”.

**Constraints**

* The number **n** will be an **integer** number in the range [1000…9999].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1000 | .----|.----|.----|.----|.----|.----| | nSum = 1+0+0+0 = 1  morseProduct = 1\*1\*1\*1\*1 = 1 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1001 | .----|.----|.----|.----|.----|..---|  .----|.----|.----|.----|..---|.----|  .----|.----|.----|..---|.----|.----|  .----|.----|..---|.----|.----|.----|  .----|..---|.----|.----|.----|.----|  ..---|.----|.----|.----|.----|.----| | nSum = 1+0+0+1 = 2  morseProduct = 1\*1\*1\*1\*1\*2 = 2 morseProduct = 1\*1\*1\*1\*2\*1 = 2 morseProduct = 1\*1\*1\*2\*1\*1 = 2 morseProduct = 1\*1\*2\*1\*1\*1 = 2 morseProduct = 1\*2\*1\*1\*1\*1 = 2 morseProduct = 2\*1\*1\*1\*1\*1 = 2 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1231 | No | nSum = 1+2+3+1 = 7  No Morse code numbers match the condition |

**Problem 5 – Bit Paths**

You are given a **board** of size **8 rows \* 4 columns**. It is initially empty (holds zeroes). A **bit path** starts at certain column at the first row and falls 7 times either **down left** (direction = -1), **down** (direction = 0) or **down right** (direction = +1). Bit paths **invert all visited cells** in the board (turns 0 to 1 and 1 to 0). Bit paths are represented as strings of 8 items, separated by a comma: a **start column** + **7 directions**. For example, the bit path "2,-1,-1,+1,-1,+1,+1,-1" starts from column 2 at the first row, moves to column 1 at the second row, then to column 0 at the third row, etc. The bit paths will never go out of the board. See the below examples to catch the idea.

After the bit paths are processed, the **bits of each board row** are taken as binary number, converted to **decimal**, and then **summed**. Finally, the obtained sum is printed in **binary** and **hexadecimal** form. The below example illustrates the entire process:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input Data** | **Initial Board** | **Process the 1st Path** | **Process the 2nd Path** | **Process the 3rd Path** | **Output** |
| 3  2,-1,-1,+1,-1,+1,+1,-1  0,0,+1,+1,0,-1,0,-1  3,0,0,-1,-1,-1,0,+1 | 0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0 | 0 0 **1** 0  0 **1** 0 0  **1** 0 0 0  0 **1** 0 0  **1** 0 0 0  0 **1** 0 0  0 0 **1** 0  0 **1** 0 0 | **1** 0 1 0  **1** 1 0 0  1 **1** 0 0  0 1 **1** 0  1 0 **1** 0  0 **0** 0 0  0 1 **1** 0  1 **1** 0 0 | 1 0 1 **1**  1 1 0 **1**  1 1 0 **1**  0 1 **0** 0  1 **1** 1 0  **1** 0 0 0  **1** 1 1 0  1 **0** 0 0 | Decimal(1011) = 11, Sum = 11  Decimal(1101) = 13, Sum = 24  Decimal(1101) = 13, Sum = 37  Decimal(0100) = 4, Sum = 41  Decimal(1110) = 14, Sum = 55  Decimal(1000) = 8, Sum = 63  Decimal(1110) = 14, Sum = 77  Decimal(1000) = 8, Sum = 85  Binary(85) = **1010101**; Hex(85) = **55** |

**Input**

The input data should be read from the console. The first line holds the number of paths **n**. The next **n** lines hold **n paths** (in the above-described format). The input data will always be valid and in the format described.

**Output**

Print at the console two lines: the sum of board rows in binary and hexadecimal form.

**Constraints**

* The count of paths **n** is an integer in the range [0…100].
* The **start positions** are a positive numbers in the range [0…3].
* The **directions** are one of the following values: **-1**, **0**, **+1**.
* Allowed working time: 0.25 seconds. Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  2,-1,-1,+1,-1,+1,+1,-1  0,0,+1,+1,0,-1,0,-1  3,0,0,-1,-1,-1,0,+1 | 1010101  55 |  | 2  2,-1,-1,+1,-1,+1,+1,-1  0,0,+1,+1,0,-1,0,-1 | 1000100  44 |  | 1  2,0,+1,0,0,0,-1,+1 | 1011  B |