# Problem 1 – Four Factors

Little Spiridon is a big NBA fan. However, since he's very good with math and not very athletic, he doesn't dream of becoming an NBA superstar, but an NBA analyst instead. Spiro just finished reading the book “Basketball on Paper” by Dean Oliver and is now looking to understand how the Four Factors of wining basketball games work. Will you help little Spiridon - the NBA fan, become big Spiridon – the NBA analyst?

According to Dean Oliver, the four factors of wining basketball games, are Shooting, Turnovers, Rebounding and Free Throws. Each factor can be applied to both the team’s offense, and defense, which basically gives us eight factors, but we will be focusing on the Offense part only, for now.

The Shooting Factor is measured using **Effective Field Goal Percentage (eFG%)**. The formula, by which the eFG% is calculated, is **(Field Goals (FG) + 0.5 \* 3-Point Field Goals (3P)) / Field Goals Attempted (FGA)**.

The Turnover Factor is measured using **Turnover Percentage (TOV%)**. The formula for the offensive TOV% is **Turnovers (TOV) / (FGA + 0.44 \* Free Throw Attempts (FTA) + TOV)**.

The Rebounding Factor for offense is measured in **Offensive Rebounding Percentage (ORB%)** and the formula for it is **Offensive Rebounds (ORB) / (ORB + Opponent Defensive Rebounds (Opp DRB))**.

The Free Throw factor, or the **Free Throw Percentage (FT%)**, is by far the easiest to calculate and the formula for it is **Free Throws (FT) / FGA**.

Your job is to write a program, which takes certain input parameters, calculates the four offensive factors, **rounded to the third digit after the decimal sign**, and then prints them on the console. Each factor should **always be a positive number**.

**Input**

The input data should be read from the console. It consists of **eight** input values, each at a separate line:

1. **FG** – Field Goals
2. **FGA** – Field Goal Attempts
3. **3P** – 3-Point Field Goals
4. **TOV** – Turnovers
5. **ORB** – Offensive Rebounds
6. **Opp DRB** – Opponent Defensive Rebounds
7. **FT** – Free Throws
8. **FTA** – Free Throw Attempts

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 and it should consist of **four** strings, each printed on a separate line in the following format:

* **eFG%** {factor result}
* **TOV%** {factor result}
* **ORB%** {factor result}
* **FT%** {factor result}

Look at the examples below, to get a better understanding of how the output should be structured.

**Constraints**

* All input parameters will be valid integers, in the range [1 ... 3,000,000,000 ].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3351  7018  796  1125  967  2550  1556  2080 | eFG% 0.534  TOV% 0.124  ORB% 0.275  FT% 0.222 | eFG% = (3351 + 0.5 \* 796) / 7018 = 0.534  TOV% = 1125 / (7018 + 0.44 \* 2080 + 1125) = 0.124  ORB% = 967 / (967 + 2550) = 0.275  FT% = 1556 / 7018 = 0.222 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3643  7168  138  1088  1173  2171  1587  2132 | eFG% 0.518  TOV% 0.118  ORB% 0.351  FT% 0.221 |  | 2193  5293  263  957  677  2193  1090  1462 | eFG% 0.439  TOV% 0.139  ORB% 0.236  FT% 0.206 |

# Problem 2 – Detective Boev

Detective Boev is very good with decrypting messages. One day he got a message sent by general Ivanov. Knowing the decrypting algorithm he asked his secret agent, who is learning C# in SoftUni, to write a console application for decrypting general’s messages. Help him.

You will be given an **encrypted message** and a **secret word.**

You need to take the secret word and sum all of it’s symbols ASCII code. After that you must sum all digits until the result is only one digit.  
For example if the word is **code** (c + o + d + e => 99 + 111 + 100 + 101 => 411 => 4 + 1 +1 => **6** ). The **mask** is **6**.

You must decrypt all symbols in the message. Loop over the message and check if the ASCII code of the symbol can be **devided by the mask** without reminder. If so, **add the mask** to the simbol’s ASCII code. Otherwise, **subtract** it. Finally you must **reverse** the message and print the result on the console.

**Input**

The input data should be read from the console. The input will contain exactly two lines.

* The first line will hold the **secret word.**
* The second line will hold the **encrypted message.**

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

**Output**

The output data must be printed on the console.

On the only output line you must print the decrypted message.

**Constraints**

* The **message** and the **secret word** will contain only characters from the ASCII table.
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Mask** |
| C#  |wfvohyfqX#hodzwclV | Software University | C# => 67 + 35 => 102 => 1 + 0 + 2 = 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| abc  ;7\*8&yioyg<&mtossglmulV | Programming Basics 2015 | 0.5  dofe | bira |

# Problem 3 – Summertime for Programmers

Dimitrichko has a dream – to become the best programmer ever. He is studying at SoftUni, investing all of his free time doing homework and watching videos. He enjoys drinking beer, but he doesn’t have time for this. Your task is to draw him one and make him happy. Beware that he may get tipsy very easy. Fill exactly half of the bottle with beer.

You are given an integer number **N** (always **odd**). The **width** of the **bottom** of the bottle should be 2 \* **N.** The **width** of the **top** of the bottle should be **N + 1.** The **height** of the bottle should be **3 \* N + 1.** Check the examples below to understand your task better.

### Input

The input number **N** should be read from the console. It will be **odd** number in the range **[3..59]**.  
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. Use the “**\***” to draw the bottle and fill it with “.” and “@”. Follow the examples below.

### Constraints

* **N** will always be a positive **odd** number in the range [**3** … **59].**
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7 | \*\*\*\*\*\*\*\*  \* \*  \* \*  \* \*  \* \*  \* \*  \* \*  \*............\*  \*............\*  \*............\*  \*............\*  \*............\*  \*............\*  \*............\*  \*@@@@@@@@@@@@\*  \*@@@@@@@@@@@@\*  \*@@@@@@@@@@@@\*  \*@@@@@@@@@@@@\*  \*@@@@@@@@@@@@\*  \*@@@@@@@@@@@@\*  \*@@@@@@@@@@@@\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3 | \*\*\*\*  \* \*  \* \*  \*....\*  \*....\*  \*....\*  \*@@@@\*  \*@@@@\*  \*@@@@\*  \*\*\*\*\*\* |  | 5 | \*\*\*\*\*\*  \* \*  \* \*  \* \*  \* \*  \*........\*  \*........\*  \*........\*  \*........\*  \*........\*  \*@@@@@@@@\*  \*@@@@@@@@\*  \*@@@@@@@@\*  \*@@@@@@@@\*  \*@@@@@@@@\*  \*\*\*\*\*\*\*\*\*\* |

# Problem 4 – Light the Torches

Father Haralambi is an Orthodox priest, who is serving in the small Elin Pelinian monastery. During the day, Father Haralambi mostly prays and, since he's a casual fisherman, on Sundays, he goes to the river near the monastery and tries to catch his own dinner. In the evenings, it's Father Haralambi's duty, to go down to the unusually large monastery basement and light the torches in each room. He is a good Orthodox priest and as such he is not fully sober in the evenings, thus he sometimes is not able to light all the torches. Every morning, after the morning prayer, the father counts the rooms that are dark and using a certain formula that he created, calculates the number of prays that he has to say during the day, in order to make up for the rooms that he wasn't able to reach. Your job is to create a program, which will monitor Father Haralambi's movements in the basement and automate the calculation process, thus allowing him to focus on praying, rather than lose precious time doing math.

The monastery's basement **is linear** and there is **only one single door** connecting two adjacent rooms. The only entrance to the basement, is in the room that's **in the middle of the basement** – the rooms on the left from the entrance will always be equal to the number of rooms on the right from the entrance. **The total number of rooms will be an odd number N**.

You will receive a **string of characters**, which will tell you which rooms are currently lit and which rooms are dark. The string can have **a length less than or equal** to the **number** of rooms. It will contain only **two types of characters, 'L' for light** and **'D' for dark** and each character will represent the state of **a single room**.  
Your first task is to go through all the rooms, starting from the **leftmost room** (room 0), and **update** the state of each room, using the **characters** in the string **from left to right**. If you reach the end of the string before you reach the end of the basement, **go back** at the beginning of the string **and continue**.

After the rooms’ state has been updated, Father Haralambi will enter the basement **from the only entrance** and you are to begin monitoring his movements. The movements will be represented as a **series of commands**, each provided **on a separate line**. The possible commands are **RIGHT X, LEFT X and END**, with **X** being **the number of rooms** that the priest passes through and **LEFT/RIGHT** being the **direction** that he's walking in.

For example, if the priest is currently in **room 2** and the command **RIGHT 3** is received that will mean that the priest will end up in **room 6** – he passes through 3 rooms and ends up in the next one. Respectfully, if he is currently in **room 5** and you receive the command **LEFT 3**, the priest will wind up in room 1. When the priest **stops walking**, he will **update** the **state** of the current room. If the room is dark, then he will light it up and if the room is already lit, he will stumble upon the torch and the room will become dark. The priest **updates only** the rooms that he **ends up in** and **not** the ones that **he passes through,** while trying to get there. Also, don't forget that the priest is not quite sober, so he may try to go to **rooms that do not exist**. For example, if he is in **room 0**, he does a **RIGHT 6** movement, but the basement has **only 5** total rooms, then he will **end up** in the last room on the right, which will be **room 4**. If the priest **does not change** rooms, after you receive a command, then the **state** of the current room **is not changed.**

When you receive the **END command,** that means the priest has fallen asleep and you should start calculating his prays for the next day.

Once the priest has fallen asleep, go through the entire basement and **count the rooms that are still dark**. The total number of prays that Father Haralambi has to say, when he wakes up, is equal to the **ASCII code** of the character **'D' multiplied** bythe **number** of **dark** rooms.

**Input**

The input data should be read from the console.

* On the **first line** you will receive an **ODD number N**, specifying the **total number of rooms in the basement**.
* On the **second line**, you will receive a **string**, containing a **series of 'L' and 'D' characters**.
* On the **next lines**, you will receive a **series of commands** in the described formats. **The input ends when the END command is received.**

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 and should consist of a **single number**. The number is the **total number of prays**.

**Constraints**

* The number N will be a valid ODD integer in the range [1 ... 99].
* The string has length less than or equal to N and will NOT contain any white spaces.
* Each command will be provided on a separate line in the format: RIGHT {X} / LEFT {X} / END
* The number X will always be a valid integer in the range [0 ... 100].
* The number of commands will be in the range [1 ... 202].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Example**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  LD  LEFT 0  LEFT 20  RIGHT 2  LEFT 2  END | 0 | 5 total rooms and string “LD” (The grey cell marks the current position of the priest):   |  |  |  |  |  | | --- | --- | --- | --- | --- | | L | D | L | D | L |   **LEFT 0** – the priest moves to the **Left** passing through **0 rooms**:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | L | L | L | D | L |   **LEFT 20** – the priest goes **as far Left** **as he can**:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | D | L | L | D | L |   **RIGHT 2** – the priest goes to the **Right** passing through **2 rooms**:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | D | L | L | L | L |   **LEFT 2** – priest goes to the **Left** passing through **2 rooms:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | L | L | L | D | L |   **END** – we **stop** receiving commands, we calculate the number of prays and we print the result on the console:  0 Dark Rooms \* (ASCII code of 'D') = 0 |

# Problem 5 – Wiggle Wiggle

Nakov likes Hip-hop music. He got inspired by the music clips he saw on YouTube and now he really wants to learn how to twerk. You got to help him to accomplish his goal. Since he is a lot into bits, you need to use bitwise operations.

You are given a **sequence of 64 bit integer** numbers on **one line,** separated by a space. You have to take each **consecutive pair** of numbers (first with second, third with fourth and so on). Then you have to pass through the bits from **right to left** and switch **every bit at even positions** from these two numbers. Assume the **position ‘0’ is even.**

For example, the first number pair is 36854775807 and 2285477580.

|  |  |
| --- | --- |
| 36854775807 | 000000000000000000000000000100010010100101101110011111111111111 |
| 2285477580 | 000000000000000000000000000000010001000001110011001111011001100 |

After switching, the numbers are:

|  |  |
| --- | --- |
| 36519493358 | 000000000000000000000000000100010000000101100110011111011101110 |
| 2554175453 | 000000000000000000000000000000010011100001111011001111111011101 |

In the end, you need to **reverse** the whole two numbers **bitwise**. The final result is:

|  |  |
| --- | --- |
| 9223372000335806737 | 111111111111111111111111111011101111111010011001100000100010001 |
| 9223372034233491490 | 111111111111111111111111111111101100011110000100110000000100010 |

### Input

The input data should be read from the console. It consists of a sequence of 64-bit integer numbers, separated by a space. The sequence **count** will always be an **even** number.

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

### Output

Print each of the obtained numbers on a single line in the following format:  
**{decimal} {binary}**

### Constraints

* **Every** **input number** will be a 64-bit integer in the range [0 … 9,223,372,036,854,775,807].
* The sequence **count** will always be an **even** number.
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.
* Hint: You should print all **63** bits in the final result for each number (without the sign bit).

### Examples

|  |
| --- |
| **Input** |
| 36854775807 2285477580 |
| **Output** |
| 9223372000335806737 111111111111111111111111111011101111111010011001100000100010001  9223372034233491490 111111111111111111111111111111101100011110000100110000000100010 |

|  |
| --- |
| **Input** |
| 4558 488755 25 555 |
| **Output** |
| 9223372036854426212 111111111111111111111111111111111111111111110101010101001100100  9223372036854632089 111111111111111111111111111111111111111111111011100111010011001  9223372036854775798 111111111111111111111111111111111111111111111111111111111110110  9223372036854775236 111111111111111111111111111111111111111111111111111110111000100 |

|  |
| --- |
| **Input** |
| 5436 4266 64367 234256 423456 2 |
| **Output** |
| 9223372036854771671 111111111111111111111111111111111111111111111111110111111010111  9223372036854770241 111111111111111111111111111111111111111111111111110101001000001  9223372036854662341 111111111111111111111111111111111111111111111100100010011000101  9223372036854590650 111111111111111111111111111111111111111111111010010110010111010  9223372036854635999 111111111111111111111111111111111111111111111011101110111011111  9223372036854492157 111111111111111111111111111111111111111111110111010101111111101 |