# Exam Preparation

## Problem 1 – Crossroads

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/995/>

Our favorite super-spy action hero Sam is back from his mission in the previous exam, and he has finally found some time to go on a **holiday**. He is taking his wife somewhere nice and they’re going to have a really good time, but first, they have to get there. Even on his holiday trip, Sam is still going to run into some **problems** and the first one is, of course, getting to the airport. Right now, he is stuck in a traffic jam at a **very** **active** **crossroads** where a lot of **accidents** happen.

Your job is to keep track of traffic at the crossroads and report whether a **crash happened** or everyone **passed** the **crossroads** **safely** and our hero is one step closer to a much desired vacation.

The road Sam is on has a **single** **lane** where cars queue up until the **light** **goes** **green**. When it does, they start passing one by one during the **green** **light** and the **free window** before the **intersecting** **road’s** **light** goes **green**. During **one** **second** only **one** **part** of a **car** (a **single** **character**) passes the crossroads. If a car is still in the crossroads when the **free** **window** ends, it will get hit at the **first character** that is still in the crossroads.

### Input

* On the **first line**, you will receive the duration of the **green** **light** in seconds – an **integer** **in the range [1-100]**
* On the **second line**, you will receive the duration of the **free** **window** in seconds – an **integer** **in the range [0-100]**
* On the **following lines**, until you receive the "**END**" command, you will receive one of two things:
  + A **car** – a **string** containing any ASCII character, or
  + The command "**green**" which indicates the **start** of a **green** **light** **cycle**

A **green** **light** **cycle** goes as follows:

* During the **green** **light** cars will enter and exit the crossroads one by one
* During the **free window** cars will only exit the crossroads

### Output

* If a **crash** **happens**, **end the program** and print:  
  "A crash happened!"  
  "{car} was hit at {characterHit}."
* If everything **goes** **smoothly** and you receive an "**END**" command, print:  
  "Everyone is safe."  
  **"**{totalCarsPassed} total cars passed the crossroads.**"**

### Constraints

* The input will be **within the constaints** specified above and will **always be valid**. There is **no need** to check it explicitly.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10  5  Mercedes  green  Mercedes  BMW  Skoda  green  END | Everyone is safe.  3 total cars passed the crossroads. | During the first green light (10 seconds), the Mercedes (8) passes safely.  During the second green light, the Mercedes (8) passes safely and there are 2 seconds left.  The BMW enters the crossroads and when the green light ends, it still has 1 part inside ('W'), but has 5 seconds to leave and passes successfully.  The Skoda never enters the crossroads, so 3 cars passed successfully. |
| 9  3  Mercedes  Hummer  green  Hummer  Mercedes  green  END | A crash happened!  Hummer was hit at e. | Mercedes (8) passes successfully and Hummer (6) enters the crossroads but only the 'H' passes during the green light. There are 3 seconds of free window, so "umm" passes and the Hummer gets hit at 'e' and the program ends with a crash. |

## Problem 2 – Knight Game

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/670/>

Chess is the oldest game, but it is still popular these days. For this task we will use only one chess piece – the **Knight**.

The knight moves to the **nearest square but not on the same**[row](https://en.wikipedia.org/wiki/Glossary_of_chess#rank)**,**[column](https://en.wikipedia.org/wiki/Glossary_of_chess#file)**, or**[diagonal](https://en.wikipedia.org/wiki/Glossary_of_chess#diagonal). (This can be thought of as moving two squares horizontally, then one square vertically, or moving one square horizontally then two squares vertically— i.e. in an **"L" pattern**.)

The knight game is played on a board with dimensions **N x N** and a lot of chess knights **0 <= K <= N2**.

You will receive a board with **K** for knights and '**0'** for empty cells. Your task is to remove a minimum of the knights, so there will be no knights left that can attack another knight.

### Input

On the first line, you will receive the **N** size of the board

On the next **N** lines you will receive strings with **Ks** and **0s**.

### Output

Print a single integer with the minimum amount of knights that needs to be removed

### Constraints

* Size of the board will be 0 < N < 30
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  0K0K0  K000K  00K00  K000K  0K0K0 | 1 |
| 2  KK  KK | 0 |
| 8  0K0KKK00  0K00KKKK  00K0000K  KKKKKK0K  K0K0000K  KK00000K  00K0K000  000K00KK | 12 |

## Problem 3 – Ticket Trouble

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/995/>

Our hero finally managed to park successfully and now realized that he has forgotten to clear out his suitcase and has a bunch of documents and old plane tickets in there. He doesn’t have enough time to go through it all and needs your help to find the tickets for the trip. Sam can tell you the location they’re headed and give you all the contents of his suitcase. Your job is to filter through them all and find the correct pair of tickets.

First, you need to find all valid tickets in his suitcase. Each valid **ticket** has the following characteristics:

* It’s **enclosed** in either **brackets** {} or **square brackets** [].
  + If it contains **mixed opening/closing brackets** (such as {] or [}), **ignore that ticket entirely**
* It contains **any printable ASCII character** inside it
* It contains 2 blocks, enclosed in either **brackets** {} or **square brackets** [], different from the **opening/closing brackets** (examples below):
  + The first will contain the location of their trip, in the format **"CountryCode CityCode"**, with CountryCode being **always** **3 capital latin letters** and CityCode – always **2 capital latin letters**
  + The second will contain the seat number, which consists of **1 capital latin letter** and **1 or 2 digits**

An example valid ticket to Sofia, Bulgaria (BUL SF) looks like this:

|  |
| --- |
| 18dhskf5e{FT\*n\*uk\_[BUL SF]vqre31r/asd[A10]123asd}ad1rsfez |

In **green**, you have the **whole** **ticket** and in **yellow**, the inner blocks with the **location** and **seat** **number** of the **ticket**.

You need to **filter** **only** the **tickets** to the **same** **location** that **Sam** has **told** **you** (see **input**) and **if there are more than 2 valid tickets** for that **location**, take the **2 with the same row number** – the **numeric** **part** of the **seat** **number**.

### Input

* On the **first line** of input, you will receive the **location**, in **the same exact format as in the tickets**
* On the **second line**, you will receive the whole suitcase contents in a single string

### Output

* After **finding** the **correct** **tickets**, print them in the **following** **format**:  
  "You are traveling to {location} on seats {firstSeat} and {secondSeat}."
* The **two** **seats** should be printed in the **same** **row** **they** **were** **found** in the suitcase string

### Constraints

* There will **always** be at least **2 valid tickets**
* There will **never** be more than **2 valid tickets** with the same **row** **number**
* There will **never** be more than **1 valid pair of tickets**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| BUL SF  1d2ajsd/.{1d9823{BUL SF}10eu2{A11}12das}2fsdf[a2d{BUL SF}12e0dd1rrwg{A11}af/zc,s]d1d0429{d12dasd[LUB SF]123asdAS[A15]fsdf}21ijp3diasd{[BUL SF][B11]}112edasd | You are traveling to BUL SF on seats A11 and B11. | We have only 2 valid tickets here, so we just take their seats and print the results.  The first red ticket is invalid because it uses only one type of brackets.  The second red ticket is valid, but for a different location. |
| BHS PN  1d2ajsd/.{1d9823{BUL SF}10eu2{A11}12das}2fsdf[a2d{BHS PN}12e0dd1rrwg{S35}af/zc,s]d1d0429{d12dasd[BHS PN]123asdAS[A15]fsdf}21ijp3diasd{asda/gjkl.jlk[BHS PN]112easd[T35]pile1s1kartofi}112edasdtarator120asd{asda/gjkl.jlk[BHS PN]112easd[T45]hello??}qr3cf4 67 | You are traveling to BHS PN on seats S35 and T35. | This time we have an invalid ticket – different location and 4 valid tickets, so we take the ones with the same row – 35 (S35 and T35). |
| BHS PN  dsgbvcmv,[123asd{ BHS PN}asdasd{D48}zxcwqc]12dzsdaads2[asd23rf{BHS PN}ghv,/jkm,.n{F48}mbn,buhyj ergd][werg3 54atz {BHS PN}C 3ZSR VG{A1}dg 4a 34 zgdsgf]g45ag | You are traveling to BHS PN on seats F48 and A1. | Ticket [123asd{ BHS PN}asdasd{D48}zxcwqc] is invalid because of the extra whitespace in one of the inner blocks, so we take the other 2 valid tickets. |

## Problem 4 – Movie Time

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/995/>

Sam and his wife are finally on the plane and they’re about to watch a movie and relax during the flight. But first, Sam needs to overcome one last obstacle – picking the movie.

After all the trouble they’ve gone through so far, his wife has become very irritable and he has to tread carefully. He needs your help to **sort** **all** the **available** **movies** properly and pick **the** **best** **one** **according** to his **wife’s** **preferences**.

You will be given information about his wife’s favorite **movie** **genre** and whether she prefers **short** or **long** movies.

Then, you will receive information about **all** the **movies** Sam can **choose** **from**.

Due to her current mood, his wife still **might** **not** **like** the **best** **movie** according to her **preferences**, so you will **have** to **keep** **offering** **her** **movies** in her **favorite** **genre**, **ordered** **by** their **timespan**, until she agrees to one. If 2 movies have the same duration, order them **alphabetically**.

### Input

* On the **first line**, you will receive Sam’s wife’s favorite genre
* On the **second** **line**, you will receive either “**Short**” or “**Long**”, indicating her favourite movie duration preference
* On **each** of the **following** **lines**, until you receive “**POPCORN**!”, you will receive information about a **movie** in the format: “{name}|{genre}|{duration}”
* The duration will be in the format “**HH:mm:ss**”. Keep in mind that this format is different for the **TimeSpan** class in C#.

### Output

* After receiving the “**POPCORN**!” command, you need to keep printing information about the **next** **best** **movie** in the chosen favourite genre, until you receive “**Yes**” as an answer. Print the movies in the format: “{name}”
* After you receive the “**Yes**”, print on 2 lines:
  + Information about the **chosen** **movie** in the format:  
    “We're watching {chosenMovie} - {chosenMovieDuration}”  
    Duration should be in the same format as received.
  + Information about the **summed** **duration** of **all** **available** **movies** in the format:  
    “Total Playlist Duration: {totalPlaylistDuration}”  
    Duration should be in the same time format as above.

### Constraints

* There will **always** be at least **one** movie in the input
* Sam’s wife will **always** agree to a movie before you run out movies from her favorite genre
* The **total** **playlist** **duration** will **never** be more than **23:59:59**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Drama  Short  Pick This|Drama|00:12:25  Incorrect Genre|Not Drama|01:35:35  Don't Pick This|Drama|01:01:01  Skip This|Drama|00:00:01  POPCORN!  Nope  Yes | Skip This  Pick This  We're watching Pick This - 00:12:25  Total Playlist Duration: 02:49:02 |
| Horror  Long  IT|Horror|01:57:34  Friday the 13th|Horror|01:35:43  Big Sad Drama|Drama|02:04:06  Some Stupid Comedy|Comedy|00:35:35  Very Long Horror Movie|Horror|03:06:09  POPCORN!  Nah  Yes | Very Long Horror Movie  IT  We're watching IT - 01:57:34  Total Playlist Duration: 09:19:07 |
| Comedy  Short  IT|Horror|01:57:34  Funny Movie|Comedy|01:37:46  Friday the 13th|Horror|01:35:43  Big Sad Drama|Drama|02:04:06  Some Stupid Comedy|Comedy|00:35:35  Some Stupid Comedy 2|Comedy|00:49:49  Funny Movie 2|Comedy|01:38:12  Funny Movie: New LOL|Comedy|01:44:00  Funny Movie 3|Comedy|01:46:12  Funny Movie 4|Comedy|01:46:12  POPCORN!  No, thanks  Not that one  No  Uhh, maybe?  Nope  Yes | Some Stupid Comedy  Some Stupid Comedy 2  Funny Movie  Funny Movie 2  Funny Movie: New LOL  Funny Movie 3  We're watching Funny Movie 3 - 01:46:12  Total Playlist Duration: 15:35:09 |