Pandemic

Travis is germaphobic. He doesn't like the beginning or end of the semester, because many students try to shake his hands. The ails of his class consume his thoughts, and his fear of getting sick has crippled his ability to make good homework problems. Travis needs a program to ease his mind and help him determine if the class is actually infected with the bubonic plague.

Problem

Travis has a documented list of people and their symptoms (to prevent a FERPA violation Travis has replaced the student's names). Travis can modify the symptom list by adding and removing symptoms for students. To determine the infected students Travis will ask either what symptoms a student has or ask for a list of all of the students with a particular symptom.

Input Specification

Input will start with a single positive integer, n (n < 100,000), representing the number of updates/queries Travis will give. The next n lines contain either an update or a query operation.

An update operation will start with the letter "u" (quotes for clarity) followed by a space. Following the space a name and symptom will be given separated by a single space character. The name will appear first and contain at most 100 lower case Latin letters (no whitespace). The symptom will immediately follow and will contain at most 100 lower case Latin letters (no whitespace). This update will imply that the student with the given name has begun to show the given symptom.

A query operation will start with the letter "q" (quotes for clarity) followed by a space. Following the space either the word "student" or "symptom" (quotes for clarity) will appear following this and ending the line will be a single word denoting either a given student or symptom (depending on the query type). If the word "student" appears first, you will print out all the symptoms the student has. The student's name will be the word that follows the word "student". If the word "symptom" appears first, you will print out all the students that has the given symptom. The symptom will be the word that follows the word "symptom".

You can assume no student will be given the same symptom twice.

Output Specification

For each query you will either output all the students that have a given symptom or all the symptoms a given student has.

When printing all the students with a given symptom begin the output with the number ST, where ST denotes the number of students with the given symptom. After this should follow ST lines each containing the name of the afflicted student.

When printing all the symptoms a given student has begin the output with the number $\boldsymbol{SY}$, where $\boldsymbol{SY}$ denotes the number of symptoms the student has. After this should follow $\boldsymbol{SY}$ lines each containing the name of the symptom.

The symptoms/students can be printed in any order, but each one should be printed at most once per query.

Sample Input	Sample Output

`5`
`u john sneezed`
`u john coughed`
`u tyler cried`
`q student john`
`q student tyler`
	`2`
	`sneezed`
	`coughed`
	`1`
	`cried`

`1`
`q student evan`
	`0`

`8`
`u eric laughed`
`u tim laughed`
`u alex blinked`
`u tim blinked`
`q symptom laughed`
`u alex laughed`
`u mike sneezed`
`q student alex`
	`2`
	`eric`
	`tim`
	`2`
	`blinked`
	`laughed`

Grading Details

Read/Write from/to standard input/output – 10 points

Good comments, whitespace, and variable names – 15 points

Read in all the input – 5 points

No extra input output (e.g. input prompts, "Please enter the number of words") – 10 points

Your program will be tested on 10 test cases – 5 points each

No points will be awarded to programs that do not compile using gcc -std=gnu11 (gnu "eleven").

Sometimes a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem you must use array lists to store the information of students. ***Without this your program will earn at most 50 points!***

Any case that causes your program to return a non-zero error return code will be treated as completely wrong. Additionally any case that takes longer than the maximum allowed time (the max of {5 times my solution, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.

Base Structure

I recommend using a 2D array list of strings. Consider using the following structure organization below. You will need to make the functions to modify and utilize the structure

```
typedef struct name {
   char * str; // Just a string
} name;
typedef struct array_list {
   name * arr; // array of names
   int size, cap;
} array_list;
typedef struct full_list{
   array_list * aol; // array of (array)lists
   int size, cap;
} full_list;
```

In your main method you will want two of these full_lists one for the people that can help you find symptoms and one for the symptoms that can help you find people.

<u>Advice:</u> Consider making the first stored name in your array_list the name of the corresponding student/symptom.

Imagine you create a full_list pointer named `symp`

```
full_list * symp = calloc(1, sizeof(full_list)); // maybe initialize
                                                  // this
```

that stores the a list of students for each symptom, then you could expect the following memory layout for some cases might look like the following (pictorially speaking)

▆	symp->aol[i].arr[0]	symp→aol[i].arr[1]	symp→aol[i].arr[2]	...
symp->aol[0]	“coughed”	“john”	“eric”	...
symp->aol[1]	“sneezed”	“eric”		...
symp->aol[2]	“sighed”	“adam”	“mike”	...
symp->aol[3]	“wheezed”	“mike”	“john”	...
⋮	⋮	⋮	⋮	⋱

As an example symp→aol[2].list[1] would be “adam”, and symp->aol[1].list[2] would not have been created yet