C++ Fundamentals: Second Exam

The following tasks should be submitted to the SoftUni Judge system, which will be open starting Sunday, 28 January 2018, 09:00 (in the morning) and will close the same day at 15:00. Submit your solutions here: https://judge.softuni.bg/Contests/Compete/Index/923.

For this exam, the code for each task should be a single C++ file, the contents of which you copy-paste into the Judge system.

Please be mindful of the strict input and output requirements for each task, as well as any additional requirements on running time, used memory, etc., as the tasks are evaluated automatically and not following the requirements strictly may result in your program's output being evaluated as incorrect, even if the program's logic is mostly correct.

You can use C++03 and C++11 features in your code.

Unless explicitly stated, any integer input fits into **`int`** and any floating-point input can be stored in **`double`**. On the Judge system, a C++ **`int`** is a **`32-bit`** signed integer and a C++ **`double`** is a **`64-bit`** IEEE754 floating point number.

NOTE: the tasks here are NOT ordered by difficulty level.

Task 4 – Decryption (Second-Exam-Task-4-Decryption)

Supreme Leader Stroke *(I am NOT telling you where I got the idea for that name from)* has introduced a new encryption system for all Imperial transmissions, so that the Rabble Alliance can't directly read the Imperial message traffic. It's a dumb type of encryption, but hey – we are talking about a universe with FTL travel and handheld laser weapons where the elite warriors, capable of manipulating the universe with their thoughts, fight with shiny sticks… and where the supposedly powerful Empire's troops can't hit the broad side of a barn – from INSIDE the barn… so yeah, dumb encryption is totally believable. Anyway, the Alliance needs to be able to listen in to Imperial traffic, so it needs to find a way to decrypt the messages.

Here's how the encryption works:

- Each original message contains only lowercase English letters (**`a-z`**) and the space symbol (ASCII code 32)
- The encryption algorithm uses an encryption table. The encryption table simply maps one English letter to another English letter.
- For each character in the original message, the algorithm replaces it with the matching character from the encryption table (spaces remain spaces)
- Decrypting an encrypted message is simple – just use the same algorithm, but with the "reversed" mapping

An encryption table is just a representation of what each English character is changed to during the encryption. Here's an example encryption table:

Original	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
Encrypted	b	c	d	h	a	w	g	e	i	j	k	l	n	m	p	q	r	o	t	s	u	v	f	x	y	z

Note that some (even all) letters can remain the same in an encryption, but no two "original" characters will map to the same "encrypted" character and there will always be a single "encrypted" character for each "original" character in the English alphabet.

If we use the above table to encrypt the text "**`hello`**", we will get the text "**`eallp`**" (original **h** becomes **e**, original **e** becomes **a**, original **l** stays the same, original **o** becomes **p**). Also note that to decrypt the text, we just find the symbol in the "encrypted" row and change it with the one above it in the "original" row – **e** becomes **h**, **a** becomes **e**, the two **l** remain the same, and **p** becomes **o** – giving us "**`hello`**".

Also note that to define an encryption table we only need the second ("encrypted") row – we already know how the English alphabet is ordered, so we just need info about what the first letter translates to, what the second translates to, and so on. We can define the table from the example with the following string:

`bcdhawgeijklnmpqrotsuvfxyz`

The string above has the same values as the encrypted row. We know the "original" letters, because they are just the English alphabet, i.e. the "original" row is the same for each encryption table – **`abcdefghijklmnopqrstuvwxyz`**. Only the "encrypted" row changes, so we only need the encrypted row to represent an encryption.

So, we know how the Empire encrypts messages, but how can we decrypt them? Well, if you know the encryption table, it's easy, as we saw above. However, the Empire uses multiple different encryption tables for their transmissions and they try to keep them a secret.

The good news is that a group of Bothans *(many of which died)* have discovered some of the Empire's encryption tables. The bad news is that we don't know which encryption table was used for which message. An example of this situation is the following:

- We have the encrypted messages:
  uvtqmg vjgug ctgpv vjg ftqkfu yg ctg nqqmkpi hqt
  wxvsoz xcdw qzwwkbz gsrx lz dr xcz vzwypx
  wxvsoz vciqzw gdxc sko krn lpsoz krn mcsoz
  uvtqmg vjg tgdgnu ctg vjgtg
  uvtqmg aqw jcxg uqogvjkpi qp aqwt hceg
  uvtqmg aqw jcxg c hceg qp aqwt uqogvjkpi
- The Bothans have discovered the following single encryption table:
  cdefghijklmnopqrstuvwxyzab
- How do we check which encryption table is used for which message (if they are used for these messages at all)? We could try to decrypt each message with each encryption table, but how would we know if we used the correct table? Even if we get reasonably sounding text, we still can't be sure that's the original text, because it's completely possible that decrypting with two different tables would give us two messages with reasonably sounding text – we can't know which text is correct.

If you look carefully at the messages, you might notice something – the Empire has committed a huge mistake in security. The Bothans have found out that **every original Imperial message starts with the same text**! If we know what text each message starts with, we can then verify which encryption was used for which text – we just decrypt with that table and check if the decrypted text starts as we expect *(by the way, this is completely realistic – in World War 2, a similar mistake by the Axis helped in breaking the Enigma code)*!

- For the above example the text the Empire's message start with is "**stroke**" (a reference to the Supreme Leader)
- We can notice that there are **two groups of messages** – the ones starting with **uvtqmg** and the ones starting with **wxvsoz**. Those strings match the length of "**stroke**". We can now check which table works for which messages. It turns our that the first (and only) table **cdefghijklmnopqrstuvwxyzab** decrypts **uvtqmg** to **stroke** – so that is the table for the messages starting with **uvtqmg**. We can't know what the **wxvsoz** messages actually contain (unless we try every possible encryption table that decrypts **wxvsoz** to **stroke** – note that many tables can do that, so we can't ever be completely sure), but at least we have decrypted some of the messages.
- Decrypted with the **cdefghijklmnopqrstuvwxyzab** table, the messages starting with **uvtqmg** from above are the following (in order of appearance):
  stroke these arent the droids we are looking for
  stroke the rebels are there
  stroke you have something on your face
  stroke you have a face on your something

The Alliance realizes that there will be **many decrypted messages**, so it only wants to read **some of them**. Specifically, the Alliance wants to read the **biggest group of messages decrypted by one of the tables** that have been discovered.

Your task is to write a program, which, given the starting text of each original message, along with a list of encrypted messages and a list of encryption tables, prints the decrypted versions of the largest "group" of messages decrypted with one of the tables (in the order of appearance of the messages).

Input

The input will be separated into 3 parts – the first part containing the text that each original message starts with, the second part containing the encrypted messages, and the third part containing encryption tables.

The first part will be defined on the first line of the standard input – a single line will contain the symbols with which each original message starts.

The second part of the input will contain encrypted messages, each on a single line. This part of the input will end when the string **`[encryptions]`** is entered.

The third part of the input will contain encryption tables (sequences of letters of the English alphabet, in which each lowercase English letter appears exactly once). This part of the input will end when the string **`[end]`** is entered.

Output

The largest group of messages decrypted by a single table – each decrypted (original) message should be printed on a single line.

Restrictions

No encrypted messages in the input will be decryptable by more than 1 encryption.

No two groups of messages (messages encrypted with the same table) will be of equal size (i.e. there will always be a single max group).

There will be no more than 100 groups and no more than 100 messages per group.

There will be no more than 100 encryptions in the output. There can be less encryptions than groups, but there can't be more encryptions than groups.

Each encryption will "match" at least one message.

There is no specific order of the messages or the encryptions in the input – they are ordered completely randomly.

The total running time of your program should be no more than **`0.1s`**

The total memory allowed for use by your program is **16MB**

Additional Files

You are given a C++ implementation of the **encryption** algorithm. It reads an "original" message on the first line of the standard input and an "encryption" on the second line of the standard input and prints the encrypted message on a single line in the standard output. You are strongly advised to use it for testing – you should also probably write yourself a similar tool for decryption, so you can more easily understand the input.

Example I/O

Example Input *(Note: copy into a text file for easier reading)*	**Expected Output**
stroke uvtqmg vjgug ctgpv vjg ftqkfu yg ctg nqqmkpi hqt wxvsoz xcdw qzwwkbz gsrx lz dr xcz vzwypx wxvsoz vciqzw gdxc sko krn lpsoz krn mcsoz stroke holy smoke holy smoke plenty bad preachers for the devil to stoke uvtqmg vjg tgdgnu ctg vjgtg stroke why do you dislike all those rebel folk stroke im out of ideas for these testswxvsoz vciqzw gdxc sko krn lpsoz krn mcsoz uvtqmg aqw jcxg uqogvjkpi qp aqwt hceg uvtqmg aqw jcxg c hceg qp aqwt uqogvjkpi [encryptions] abcdefghijklmnopqrstuvwxyz cdefghijklmnopqrstuvwxyzab [end]	stroke these arent the droids we are looking for stroke the rebels are there stroke you have something on your face stroke you have a face on your something
master vader	master vader this will be in the output

emghub jmtub scib fmeu gciftg kiffs emghub jmtubnkcbqchhchsdugdmpug emghub jmtubg bumz fmeu ng mfmynf nzhgvi ezwvi hzb rg jfrxp zmw blfoo ivzorav dsb gsvb xzoo srn oliw mlg nzhgvi seukat redat khiu niww fa il kha mqkoqk seukat redat khiu niww ewum fa il kha mqkoqk [encryptions] zyxwvutsrqponmlkjihgfedcba efcdabghijvwslmoptukqrnxyz [end]	master vader this will also be in the output
abc cdeeee cdeeef cdeeeg abcaaa bcdccc bcdccd [encryptions] cdefghijklmnopqrstuvwxyzba abcdefghijklmnopqrstuvwxyz bcdefghijklmnopqrstuvwxyza [end]	abcccc abcccd abccce