1. Text Highlight

1.1. Statement

Even if writing code can sometimes seem like an easy action, it involves a lot of theoretical knowledge as well as a lot of practice, especially in writing complex programs.

Thus, to help users, text highlighting was created, which means that there will be keywords such as"**for", "while" , "int" , "float**"which will be highlighted for an easier understanding of the code. In this problem we aim to simulate the TextHighlight facility.

In order to simulate the highlighting on the text, we will "underline" the keywords encountered on the next line. Underlining a keyword will be done using the character '_' (underline) for each character of that word (including spaces in the case of keywords consisting of several words), and for the rest of the characters that are not part of the keywords, ' will be used ' (space).

Careful! Since many programs allow the writing of key sequences consisting of several words with a variable number of spaces between them, for our problem this case will also be taken into account!

Avoid, however, writing code in this way, because it becomes difficult to understand and a coding style must be respected.

For this topic we will consider as keywords only the following words and sequences of two words:**first of, for, for each, from, in, is, is a, list of, unit, or, while, int, float, double, string.**

1.2. Input data

- N - the number of lines entered
- N lines of 100 characters maximum, representing the text on which the highlighting will be applied

1.3. Output data

- Text with appropriate highlighting

1.4. Restrictions and clarifications

- N will be a valid natural number between 0 and 100
- The text will be read line by line
- The vector with the N lines will be dynamically allocated
- For each line, exactly as many characters will be allocated as are needed to store that line
- Solutions without dynamic allocation will be scored accordingly
- The given keywords will be stored in one or more vectors, not necessarily dynamically allocated.
- **Important**! Define and implement **functions** auxiliary. Solutions written completely or almost completely in **main** will not be considered! Global variables are not used!

1.5. Testing and Scoring

- The maximum score is of**30**of points.

1.6. Examples

Input	Explanations
4 for each number from 3 to 100 divide to first of list of divisors of number that is a number different than 1 or 2	The keywords that will be highlighted are: • 'for each' on the first line (you can see it was underlined, even though there are more spaces between the two words) and 'from' • 'list of' from line three • 'is a' from line three • 'or' from line four **Observation!**–Although "first of" is a keyword, this one**NOT**was underlined, because "first" and "of" are on different lines.
Output for each number from 3 to 100 divide to first of list of divisors of number that is a number	

2. Autocomplete

2.1. Statement

Autocomplete is a frequently used functionality. Although this helps in theory to save time, it can also generate errors, depending on how it is used.

Over time, most of the problems have been solved and the autocomplete helps in almost every case, among the most useful and used is the autocomplete of the Google search. This autocomplete is quite complex though, using both the previous searches of the current user, other users, keywords of existing pages on Google, and the most popular searches.

Thus, in this problem we propose to implement an autocomplete that starts from a certain number of words (a mini-dictionary) and that updates with the introduction of new words or the use of recent ones.

Each word in the dictionary will initially have priority 0 and will be represented in this form:

```
struct dictionary_entry{
        char* word; int
        priority;
};
```

Each word written by the user belongs to one of the following 3 categories:

- **word**--which does not match any existing word in the dictionary, so it will be written in the output as in the input and will be added to the dictionary with priority 1, being its first occurrence in the text

- **word***--which matches a higher priority word, but the user wants the word he wrote, even though it has a lower priority (this word may or may not exist in the dictionary, in which case it should be added) -> will be displayed **word** and its priority will increase

- **St.**--that matches one or more words in the dictionary (starting with **vat)**and the one with the highest priority is chosen. Also, the priority of the chosen word in the dictionary will increase for the selected word.

  Observation: If we have words in the dictionary with the same priority and we are looking for a match for a given word, the word considered to be the closest in lexicographic order will be chosen.

  For example, we are looking for a match for "abc" and we have two possibilities in the dictionary: "abcde" with priority 2 and "abce" also with priority 2. We will choose the word "abcde" because it is smaller than "abce".

2.2. Input data

- N – the initial number of words in the dictionary, which will be used for the initial dynamic allocation of the dictionary
- N words – the words from the initial dictionary
- M – the number of words entered by the user for autocomplete
- M words – the words entered by the user for autocomplete

2.3. Output data

- The text resulting from the autocomplete (the resulting words, with a space between them)

2.4. Restrictions and clarifications

- $0 < N \leq 5000$
- $0 < M \leq 5000$
- **Use dynamic allocation and reallocation for the word dictionary so there is no unused memory!**
- **Each word in the dictionary will be allocated exactly as much memory as it needs.**
- Solutions that do not use dynamic allocation will be de-pointed.
- For words entered for autocomplete **NOT** you need to use dynamic allocation!
- The characters (, . : ! ?) will be treated as stand-alone words and will be displayed as they are read
- All words contain lowercase letters of the English alphabet.
- Each word entered has a maximum of 20 characters
- **Important** !Define and implement **functions** at least for **searches** in the dictionary **adding** new word in the dictionary, etc. Solutions written completely or almost completely in **main** will not be considered!
- No global variables are used!

2.5. Testing and Scoring

- The maximum score is of**30**of points.
- The source containing the function**tomorrow**must be called:**problem 2.c**

2.6. Examples

Input	Explanation

8 arrow discussed hi interesting was divorced interview investment 13 his inv w di in* an interv , it was rather interest . **Output** his investment was discussed in an interview, it was rather interesting.	In the dictionary are the words **arrow discussed hi interesting was divorced interview investment**. All have priority 0 at this time. **his**–new word that does not match any word in the dictionary is displayed as input and added to the dictionary with priority 1 **inv**–the only match is **investment**. Investment is displayed, the dictionary does not change, and the priority of the word "investment" increases to 1. **w**–the only match is **was**. Was is displayed, the dictionary does not change, and the priority of the word "was" increases to 1. **di**–has two matches "discussed", respectively "divorced". Both have the same priority, so the smallest lexicographic word will be chosen. It will be displayed **discussed** and its priority will be incremented to 1. **in the***-the higher priority word that matches is "investment". However, the user does not want the recommended word, but the one he entered. It will be displayed **in the**(* will be removed from the output) **year**–new word is added to the dictionary with priority 1 **interval**–the only match is **interview**, which will have priority 1 from now on **it**–new word is added to the dictionary with priority 1 **wow**–the only match is **was**, which will increase its priority to 2 **rather**–new word is added to the dictionary with priority 1 **interest**–the only match is **interesting**, which will increase its priority to 1