C++ Fundamentals: Exam 1

The following tasks should be submitted to the SoftUni Judge system, which will be open starting **Sunday, 15 July 2018, 09:00** (in the morning) and will close on **Sunday, 15 July 2018, 15:00**. Submit your solutions here: https://judge.softuni.bg/Contests/Compete/Index/1113.

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 3 – Code (Exam-1-Task-3-Code)

In the far future, the citizens of Europa (the Jupiter moon, not the continent) decide to ban the use of links to websites, without the express permission of the owners of said websites (*of course, this is a fictional story, and any correlations to actual people or events is purely accidental*). The Internet community on the planet, however, did not really like this new law, so they found a way around it – they started communicating through a special code that couldn't be read by the lawmakers.

You work for the government of Europa, and you are tasked with analyzing the coded messages. What the government knows about the code so far, is that each message it consists of a sequence of **non-negative** integer numbers. The sequence is separated into "parts" – some of the integer values in the sequence are consider separators, and every **value** that is **not a separator** is an element of a part. A part is simply a sequence of **values** (non-separators), starting after a separator (or at the start of the sequence), and ending before a separator (or at the end of the text).

To do your analysis, you need some software that, given a **message** as a sequence of integers, and given **search values**, counts **how many total parts contain** each of **those values**. If multiple identical parts (same length, with the same values, in the same order) contain a value, count **all** of them (i.e. we are not searching for unique parts containing a value, but for all parts containing it).

Luckily, you know a thing or two about programming, so you will write that software yourself… well, what are you waiting for?

Input

The first line of the standard input will contain a sequence of non-negative integer values - the **separators**.

The second line of the standard input will contain a sequence of non-negative integer values, separated by single spaces – the **message**.

The following lines of the standard input will contain the **search values** – positive integers, each on a single line.

The last line of the standard input will contain the value `0` – this is not a search value (and no search value will be `0`) – it is an indicator that the program should stop reading input.

Output

For each of the entered **search values**, print a single line containing the **number of parts** that contain that **search value**.

Restrictions

There will be no more than **`30000`** values in the message.

There will be no more than **`40000`** searches.

There will be no search for the value `0`, but there could be searches for values that are not contained in any part.

The total running time of your program should be no more than **`0.2s`**

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

Example I/O

Example Input (NOTE: copy this into a text file to view the lines better)

```
46 44 32

121 111 117 32 97 114 101 32 103 105 118 101 110 32 97 32 116 101 120 116
32 105 110 32 101 110 103 108 105 115 104 46 32 108 101 116 32 117 115 32
100 101 102 105 110 101 32 97 32 119 111 114 100 32 97 115 32 97 110 121
32 115 101 113 117 101 110 99 101 32 111 102 32 97 108 112 104 97 98 101
116 105 99 97 108 32 99 104 97 114 97 99 116 101 114 115 46 32 101 97 99
104 32 111 102 32 116 104 111 115 101 32 99 104 97 114 97 99 116 101 114
115 32 119 101 32 119 105 108 108 32 99 97 108 108 32 97 32 108 101 116
116 101 114 44 32 98 117 116 32 119 101 32 119 105 108 108 32 99 111 110
115 105 100 101 114 32 116 104 101 32 117 112 112 101 114 99 97 115 101 32
97 110 100 32 108 111 119 101 114 99 97 115 101 32 118 97 114 105 97 110
116 32 111 102 32 97 32 99 104 97 114 97 99 116 101 114 32 105 110 32 97
32 119 111 114 100 32 97 115 32 116 104 101 32 115 97 109 101 32 108 101
116 116 101 114 46

97

121

104

0
```

Example Output (NOTE: copy this into a text file to view the lines better)

```
20
2
9
```

NOTE: we are not counting unique parts, we are counting all parts that contain the value (i.e. if there are two identical parts containing a value, we will count them as 2, not 1)

Example Input	Example Output

63 47 46
 	
100 105 100 63 121 111 117 63 103
117 101 115 115 47 116 104 105 115
46 105 115 46 116 101 120 116 63
 	
105
 	
63
 	
0
	3
	
	0