



## ☆ Missing Words

Given two strings, one is a **subsequence** if all of the elements of the first string occur in the same order within the second string. They do not have to be contiguous in the second string, but order must be maintained. For example, given the string "I like cheese", the words "I cheese" are a subsequence of that string.

In this challenge, you will be given two strings,  $s$  and  $t$ . Create the longest common subsequence of  $s$  in  $t$  and report the words missing from  $s$  in the order they are missing. Revisiting the earlier example, if  $s = \text{"I like cheese"}$  and  $t = \text{"like, like"}$  is the longest subsequence, and  $\text{"I cheese"}$  is the list of missing words in order.

### Function Description

Complete the function `missingWords` in the editor below. It must return an array of strings containing any words in  $s$  that are missing from  $t$  in the order they occur within  $s$ .

`missingWords` has the following parameter(s):

- $s$ : a sentence of space-separated words
- $t$ : a sentence of space-separated words

### Constraints

- Strings  $s$  and  $t$  consist of English alphabetic letters (i.e.,  $a-z$  and  $A-Z$ ) and spaces only.
- $1 \leq |t| \leq |s| \leq 10^6$
- $1 \leq \text{length of any word in } s \text{ or } t \leq 15$
- It is guaranteed that string  $t$  is a subsequence of string  $s$ .

### ► Input Format for Custom Testing

#### ▼ Sample Case 0

##### Sample Input 0

```
I am using HackerRank to improve programming
am HackerRank to improve
```

##### Sample Output 0

```
I
using
programming
```

##### Explanation 0

The missing words are:

1. I
2. using
3. programming

We add these words *in order* to the array `["I", "using", "programming"]`, then return this array as our answer.

### YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour.

[Start tour](#)

Draft saved 09:54 pm

Original code

C



```
1 ▶ #include <stdio.h>
11
12 char* readline();
13
14
15 // Complete the missingWords function below.
16
17 /*
18  * To return the string array from the function, you should:
19  *   - Store the size of the array to be returned in the result_count variable
20  *   - Allocate the array statically or dynamically
21  *
22  * For example,
23  * char** return_string_array_using_static_allocation(int* result_count) {
24  *     *result_count = 5;
25  * }
```

```

26 *     static char* a[5] = {"static", "allocation", "of", "string", "array"};
27 *
28 *     return a;
29 * }
30 *
31 * char** return_string_array_using_dynamic_allocation(int* result_count) {
32 *     *result_count = 5;
33 *
34 *     char** a = malloc(5 * sizeof(char*));
35 *
36 *     for (int i = 0; i < 5; i++) {
37 *         *(a + i) = malloc(20 * sizeof(char));
38 *     }
39 *
40 *     *(a + 0) = "dynamic";
41 *     *(a + 1) = "allocation";
42 *     *(a + 2) = "of";
43 *     *(a + 3) = "string";
44 *     *(a + 4) = "array";
45 *
46 *     return a;
47 * }
48 *
49 */
50 char** missingWords(char* s, char* t, int* result_count) {
51
52
53 }
54
55

```

```

56 int main()
57 {
58     FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
59
60     char* s = readline();
61
62     char* t = readline();
63
64     int res_count;
65     char** res = missingWords(s, t, &res_count);
66
67     for (int i = 0; i < res_count; i++) {
68         fprintf(fptr, "%s", *(res + i));
69
70         if (i != res_count - 1) {
71             fprintf(fptr, "\n");
72         }
73     }
74
75     fprintf(fptr, "\n");
76
77     fclose(fptr);
78
79     return 0;
80 }
81
82 char* readline() {
83     size_t alloc_length = 1024;
84     size_t data_length = 0;
85     char* data = malloc(alloc_length);
86
87     while (true) {
88         char* cursor = data + data_length;
89         char* line = fgets(cursor, alloc_length - data_length, stdin);
90
91         if (!line) {
92             break;
93         }
94
95         data_length += strlen(cursor);
96
97         if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
98             break;
99         }
100
101         alloc_length <= 1;
102
103         data = realloc(data, alloc_length);
104
105         if (!data) {
106             data = '\0';
107

```

```
108         break;
109     }
110 }
111
112 if (data[data_length - 1] == '\n') {
113     data[data_length - 1] = '\0';
114
115     data = realloc(data, data_length);
116
117     if (!data) {
118         data = '\0';
119     }
120 } else {
121     data = realloc(data, data_length + 1);
122
123     if (!data) {
124         data = '\0';
125     } else {
126         data[data_length] = '\0';
127     }
128 }
129
130 return data;
131 }
132
```

Line: 43 Col: 1

☐ Test against custom input

Run Code

Submit code & Continue

(You can submit any number of times)

[Download sample test cases](#) The input/output files have Unix line endings. Do not use Notepad to edit them on windows.