

☆ Missing Words

















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.

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 contiquous

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

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.

missing. Revisiting the earlier example, if s = I like cheese and t = like, like is the longest subsequence, and I cheese is the list of missing words in order.

(\) 4d 19h

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.
- $1 \le |t| \le |s| \le 10^6$
- $1 \le \text{length of any word in } s \text{ or } t \le 15$
- 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

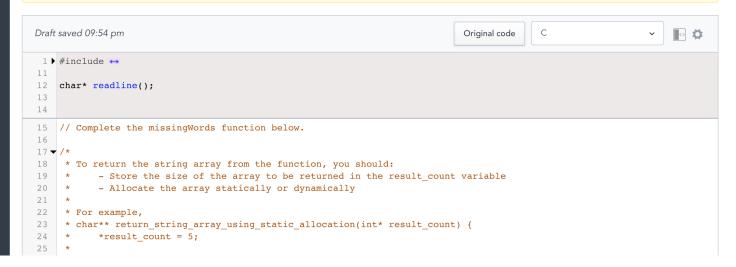
The missing words are:

- 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.



```
27
28
           return a;
    * }
29
30
    * char** return_string_array_using_dynamic_allocation(int* result_count) {
31
32
           *result_count = 5;
33
34
          char** a = malloc(5 * sizeof(char*));
35
          for (int i = 0; i < 5; i++) {
36
37
               *(a + i) = malloc(20 * sizeof(char));
38
39
           *(a + 0) = "dynamic";
40
           *(a + 1) = "allocation";
41
42
           *(a + 2) = "of";
           *(a + 3) = "string";
43
44
           *(a + 4) = "array";
45
46
           return a;
     * }
47
48
    */
50 ▼ char** missingWords(char* s, char* t, int* result_count) {
51
52
53
   }
54
55
56 int main()
57 ▼ {
         FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
58
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++) {</pre>
             fprintf(fptr, "%s", *(res + i));
68
69
70 ▼
             if (i != res_count - 1) {
                 fprintf(fptr, "\n");
 72
73
 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) {
             char* cursor = data + data length;
88
             char* line = fgets(cursor, alloc_length - data_length, stdin);
89
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
101
             alloc_length <<= 1;</pre>
102
             data = realloc(data, alloc_length);
104
             if (!data) {
105 ▼
106
                 data = '\0';
107
```

static char* a[5] = {"static", "allocation", "of", "string", "array"};

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

ab Download sample test cases The input/output files have Unix line endings. Do not use Notepad to edit them on windows.