Printing Each Word of a Sentence in a New Line

Problem, Solution, and Extensions

Problem Statement

Given a sentence, print each word of the sentence on a new line.

Input Format

The first and only line contains a sentence, S.

Constraints

- The sentence length will not exceed 1000 characters.
- The sentence consists of words separated by spaces.

Output Format

Print each word of the sentence in a new line.

Sample Input 0

```
This is C
```

Sample Output 0

```
This
is
C
```

Explanation 0

In the given string, there are three words `"This"`, `"is"`, and `"C"`. We print each word on its own line.

Solution in C

```c
#include <stdio.h>
#include <string.h>
```

```c

int main() {
    char sentence[1000];

    // Read the entire line including spaces and newline
       character
    fgets(sentence, sizeof(sentence), stdin);

    // Tokenize the sentence using space and newline as
       delimiters
    char *word = strtok(sentence, " \n");
    while (word != NULL) {
        printf("%s\n", word);
        word = strtok(NULL, " \n");
    }

    return 0;
}
```

Listing 1: Print each word of the sentence on a new line

How the code works:

- We use `fgets` instead of `scanf` to read the whole line including spaces.
- We use `strtok` with delimiters space and newline to split the input string into tokens (words).
- We print each token on a separate line until no more tokens are left.

Similar Problems for Practice

- **Count the number of words:** Instead of printing, count and print the total number of words.
- **Reverse words order:** Print the words in reverse order from last to first.
- **Longest word:** Find and print the longest word in the sentence.
- **Remove duplicates:** Print each word once even if it appears multiple times.
- **Word frequency:** Count and print how many times each word appears.

How This Problem Can Be Made Harder

- **Handle punctuation:** The sentence may include punctuation (e.g., commas, periods). You need to split correctly and/or clean punctuation from words.
- **Multiple spaces:** Handle multiple spaces or tabs between words.
- **Unicode support:** The input could contain Unicode characters (e.g., accented letters or non-English text).

- **Stream processing:** Process very large text input line by line without storing the entire sentence.
- **Case-insensitive operations:** For problems like counting or removing duplicates, treat words case-insensitively.
- **Memory optimization:** Implement the solution without using large fixed buffers or dynamic allocation.
- **Different delimiters:** Allow for other word separators like commas, semicolons, or newlines.

This document can be used as a study or practice guide to understand string tokenization in C and how to expand basic string parsing problems.