```c
// Implements a dictionary's functionality

#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <stdlib.h>

#include "dictionary.h"

// Represents a node in a hash table
typedef struct node
{
    char word[LENGTH + 1];
    struct node *next;
}
node;

// Function prototype
void word_count(node *pnode);

// TODO: Choose number of buckets in hash table
const unsigned int N = 26 * 26 * 26 * 26 ;
int count = 0;
// Hash table
node *table[N];

// Returns true if word is in dictionary, else false
bool check(const char *word)
{
    // TODO
    int hash_value = hash(word);
    node *cursor = table[hash_value];
    while (cursor != NULL)
    {
        if (strcasecmp(word, cursor->word) == 0)
        {
            return true;
        }
        cursor = cursor->next;
    }
    return false;
}
```

```c
// Hashes word to a number
unsigned int hash(const char *word)
{
    // TODO: Improve this hash function
    if ((strlen(word) == 1) | (word[1] == '\''))
    {
        return toupper(word[0]) - 'A';
    }

    if ((strlen(word) == 2) | (word[2] == '\''))
    {
        return (toupper(word[0]) - 'A' + 1) * (toupper(word[1]) - 'A' + 1) -1;
    }

    if ((strlen(word) == 3) | (word[3] == '\''))
    {
        return (toupper(word[0]) - 'A' + 1) * (toupper(word[1]) - 'A' + 1)
        * (toupper(word[2]) - 'A' + 1) - 1;
    }

    return (toupper(word[0]) - 'A' + 1) * (toupper(word[1]) - 'A' + 1)
    * (toupper(word[2]) - 'A' + 1) * (toupper(word[3]) - 'A' + 1)- 1;

/*     if ((strlen(word) == 4) | (word[4] == '\''))
    {
        return (toupper(word[0]) - 'A' + 1) * (toupper(word[1]) - 'A' + 1)
        * (toupper(word[2]) - 'A' + 1) * (toupper(word[3]) - 'A' + 1)- 1;
    }
 */
/*     return (toupper(word[0]) - 'A' + 1) * (toupper(word[1]) - 'A' + 1)
    * (toupper(word[2]) - 'A' + 1) * (toupper(word[3]) - 'A' + 1)
    * (toupper(word[4]) - 'A' + 1)- 1;
 */}

// Loads dictionary into memory, returning true if successful, else false
bool load(const char *dictionary)
{
    // TODO: Open file
    FILE *file = fopen(dictionary,"r");
    // Check
    if (file == NULL)
    {
        printf("Can't open %s\n ", dictionary);
```

```c
    }

    // Read the dictionary
    char word[LENGTH + 1];
    while (fscanf(file, "%s", word) != EOF)
    {
        // Get a memory location
        node *tmp = malloc(sizeof(node));
        if (tmp == NULL)
        {
            printf("Memory allocation error!");
            return false;
        }

        int hash_value = hash(word);
        // Add word to the linked list
        strcpy(tmp->word, word);
        tmp->next = table[hash_value];
        table[hash_value] = tmp;
    }

    // Check whether there was an error
    if (ferror(file))
    {
        fclose(file);
        printf("Error reading %s.\n", dictionary);
        unload();
        return 1;
    }

    // Close text
    fclose(file);
    return true;
}

// Returns number of words in dictionary if loaded, else 0 if not yet loaded
unsigned int size(void)
{
    // TODO :

    for (int i = 0; i < N; i++)
    {
        word_count(table[i]);
    }
```

```c
    return count;
}

// Unloads dictionary from memory, returning true if successful, else false
bool unload(void)
{
    // TODO
    for (int i = 0; i < N; i++)
    {
        while (table[i] != NULL)
        {
            node *tmp = table[i]->next;
            free(table[i]);
            table[i] = tmp;
        }
    }
    return true;
}

// Counting word up
void word_count(node *pnode)
{
    // Check
    if (pnode == NULL)
    {
        return;
    }

    // Recursion
    word_count(pnode->next);
    count++;
    return;
}
```