#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define WORD_SIZE 40
#define DICTIONARY_SIZE 1000
#define TRUE 1
#define FALSE 0

void lower_case_word(char *
int find_word(char *, char [DICTIONARY_SIZE][WORD_SIZE], int *, int);
void insert_word(char *, char [DICTIONARY_SIZE][WORD_SIZE], int *, int *);
void dump_dictionary(char [DICTIONARY_SIZE][WORD_SIZE], int *, int);

int main(void)
{
    char c, word[WORD_SIZE], dictionary[DICTIONARY_SIZE][WORD_SIZE];
    int frequency[DICTIONARY_SIZE];
    int char_index = 0, dictionary_size = 0, num_words = 0, i;

    printf("Parsing input ...
    /* Read in characters until end is reached */
    while ((c = getchar()) != EOF) {
        if ((c == ' ') || (c == ',') || (c == '.') || (c == '!') || (c == '"') || (c == ':') || (c == '\n')) {
            /* End of a word */
            if (char_index) {
                /* Word is not empty */
                word[char_index] = '\0';
                lower_case_word(word);
            }
            char_index = 0;
        } else { /* Word is empty */
            if (c == ' ')
                word[char_index] = ' ';
            else
                word[char_index] = c;
            char_index++;
        }
    }

    /* Dump dictionary */
    dump_dictionary(dictionary, frequency, &dictionary_size);
    /* Insert word */
    insert_word(word, dictionary, frequency, &dictionary_size);
    /* Find word */
    find_word(word, dictionary, frequency, &dictionary_size);
    /* Print dictionary */
    dump_dictionary(dictionary, frequency, &dictionary_size);
    printf("Number of unique words: %d
", dictionary_size);
    printf("Number of total words: %d
", num_words);
    printf("Number of unique words in dictionary: %d
", dictionary_size);

    return 0;
}
if (!find_word(word, dictionary, frequency, dictionary_size)) {
    insert_word(word, dictionary, frequency, &dictionary_size);
}
char_index = 0;
num_words++;
}
} else {
    /* Continue assembling word */
    word[char_index++] = c;
}
}
printf("There were %d words; %d unique words.\n", num_words,
dictionary_size);
dump_dictionary(dictionary, frequency, dictionary_size);
}

void lower_case_word(char *w)
{
    int i = 0;

    while (w[i] != '\0') {
        w[i] = tolower(w[i]);
        i++;
    }
}

int find_word(char *w, char d[DICTIONARY_SIZE][WORD_SIZE], int *f, int d_size)
{
    int i;

    for (i = 0; i < d_size; i++) {
        if (!strcmp(d[i], w)) {
            f[i]++;
            return TRUE;
        }
    }

    return FALSE;
}

void insert_word(char *w, char d[DICTIONARY_SIZE][WORD_SIZE],
    int f[DICTIONARY_SIZE], int *d_size)
```c
{  
    strcpy(d[*d_size], w);
    f[*d_size] = 1;
    (*d_size)++;
}

void dump_dictionary(char d[DICTIONARY_SIZE][WORD_SIZE], int *f, int d_size)  
{  
    int i;
    
    for (i = 0; i < d_size; i++)
        if (f[i] > 1)  
            printf("%s: %d\n", d[i], f[i]);
}

Computation structures / concepts:

- Learn about “parsing” textual data
- cat gettysburg.txt | parse
- getchar() until EOF
- word: string or array of chars
- dictionary: 2-D array of chars, or array of strings
- Textual DELIMITERS
- NUL character
- tolower(), strcmp(), strcpy()
- arrays are passed in via call by reference
- pass “d_size” in via call by reference to enable side effects
- forward declarations: char * vs char []
- forward declarations for 2-D char arrays: char [][]
- “%s” formatting placeholder for printing strings