```c
/*
 ============================================================================
 Name        : wcExtraCredit1.c
 Author      : Nathaniel Churchill
 Professor   : Dr. David Smith
 Description : Program to read a text file and print how many times
               all the words occur in the given file.  Has the ability to process
               command line arguments as well.
 ============================================================================
 */

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

typedef struct node {
    int count;
    char *word;
    struct node *next;
} Node;

/*
 *   This function accepts a pointer to a node and pointer to a character.
 *   The created node gets placed after the passed node.
 *
 *   head: a pointer to a node
 *   word: a pointer to a string
 *
 *   returns: a pointer to the created node
 */
Node *makeNode ( Node *head, char *word ) {
    Node *current = NULL;
    current = malloc(sizeof(Node));
    current->word = malloc(strlen(word) + 1);
    strcpy(current->word, word);
    current->next = NULL;
    current->count = 0; // initialize the count to be null
    head->next = current;
    return current;
}

/*
 *   This function accepts a pointer to a node and recursively prints the nodes
 *   their counts
 *
 *   head: a pointer to a node
 */
static void printList (Node *head){
    if (head != NULL){
        printf ( "%-10s   %d\n", head->word,  head->count);
        head = head->next;
        printList(head);
    }else {
        printf("List has ended\n");
    }
```

```
58
59 }
60
61 /*
62  *   This function accepts a pointer to a node and recursively prints the nodes
63  *   their counts to a file
64  *
65  *   outpu: a pointer to a file
66  *   head: a pointer to a node
67  */
68 static void printListFile (Node *head, FILE *output){
69     if (head != NULL){
70         fprintf (output, "%-10s   %d\n", head->word,  head->count);
71         head = head->next;
72         printListFile(head, output);
73     }else {
74         fprintf(output, "List has ended\n");
75     }
76
77 }
78
79 /*
80  *   This function accepts a pointer to a node and pointer to a character.
81  *   The function then finds the node or creates a new node recursively in
82  *   ascending order
83  *
84  *   head: a pointer to a node
85  *   word: a pointer to a string
86  *
87  *   returns: a pointer to the found or created node
88  */
89 Node *findNodeForWord(Node *head, char *word){
90     if (head->next == NULL){
91         Node *insertNode = makeNode(head, word); //insert after the head
92         return insertNode;
93     }else if (strcmp(head->next->word, word) == 0){//stuff in the list
94         return head->next;
95     }else if (strcmp(head->next->word, word) < 0){ // list word is less than given word
96         head = head->next;
97         findNodeForWord(head, word);
98     }else if (strcmp(head->next->word, word) > 0){
99         Node *linkNode = head->next;
100        Node *insertedNode = makeNode(head, word);
101        insertedNode->next = linkNode;
102        return insertedNode;
103    }
104
105 }
106
107 /*
108  *   addWord handles the adding and incrementing of a word
109  *
110  *   head: a pointer to a Node
111  *   word: a pointer to a string
112  */
113 static void addWord(Node *head, char *word){
114     Node *nodeForWord = findNodeForWord(head, word);
```

```
115     nodeForWord->count++;
116 }
117
118
119 int main ( int argc, char **argv) {
120     char c;
121     int i, j = 0;
122     char buffer[100];
123     //initialize the list with appropriate values
124     Node *list = malloc(sizeof(Node));
125     list->next = NULL;
126     list->count = 0;
127
128     int fileOutput = 0;
129     int fileInput = 0;
130
131     char *fileName = NULL;
132     FILE *src = NULL;
133     FILE *output = NULL;
134
135     int opt;
136     //get the command line options
137     while ((opt = getopt (argc, argv, "o:")) != -1){
138         switch (opt){
139         case 'o':
140             fileName = optarg;
141             fileOutput = 1;
142             if (argv[3] != NULL) {
143                 fileInput = 1;
144                 src = fopen(argv[3], "r");
145             }
146             output = fopen (fileName, "w" );
147             break;
148
149         }
150     }
151
152     //check what parameter combination we have
153     if ((argv[1] != NULL) && (src == NULL)) {
154         fileInput = 1;
155         src = fopen(argv[1], "r");
156     }
157
158     if (fileInput == 1) {
159         for (i = 0; (c = fgetc(src)) != EOF; ++i) {
160             if (isalpha(c))
161                 buffer[j++] = tolower(c);
162             else {
163                 buffer[j++] = '\0';
164                 addWord(list, buffer);
165                 j = 0;
166             }
167         }
168     } else {
169         while ((c = getchar()) != EOF) {
170             if (isalpha(c))
171                 buffer[j++] = tolower(c);
```

```
172             else {
173                 buffer[j++] = '\0';
174                 addWord(list, buffer);
175                 j = 0;
176             }
177         }
178     }
179 
180 
181 
182 
183     /*
184      * print the list, skip the two nodes as the first is null and the second
185      * contains "" which is allowed by isalpha()
186      */
187     if(fileOutput == 1) {
188         printListFile(list->next->next, output);
189     }else {
190         printList(list->next->next);
191     }
192     fclose (src); // close the file
193     fclose (output);
194     return 0;
195 }
196 
```