```c
/*
 ============================================================================
 Name        : wl.c
 Author      : Nathaniel Churchill
 Professor   : Dr. David Smith
 Description :  C, Ansi-style progam to read text file and prints on what lines the word occurs
 ============================================================================
 */

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

typedef struct node {
    int lines[200];
    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   ", head->word);
        int i;
        for (i = 0; i < head->count; i++){
            printf("%d, ", head->lines[i]);
        }
        printf("\n");
```

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

```
114
115 /*
116  *   addWord handles the adding and incrementing of a word
117  *
118  *   head: a pointer to a Node
119  *   word: a pointer to a string
120  *   lineNumber: a point to an int for the line number
121  */
122 static void addWord(Node *head, char *word, int lineNumber){
123     Node *nodeForWord = findNodeForWord(head, word);
124     int *previous = &nodeForWord->lines[nodeForWord->count - 1];
125     //test to remove duplicate numbers for words that appear multiple times on the same line
126     if (*previous != lineNumber){
127         nodeForWord->lines[nodeForWord->count] = lineNumber;
128         nodeForWord->count++;
129     }
130
131 }
132
133
134 int main ( int argc, char **argv) {
135     char c;
136     int i, j = 0;
137     char buffer[100];
138     //initialize the list with appropriate values
139     Node *list = malloc(sizeof(Node));
140     list->next = NULL;
141     list->count = 0;
142
143
144     int fileOutput = 0;
145     int fileInput = 0;
146     int lineNumber = 1;
147
148     char *fileName = NULL;
149     FILE *src = NULL;
150     FILE *output = NULL;
151
152     int opt;
153     //get the command line options
154     while ((opt = getopt (argc, argv, "o:")) != -1){
155         switch (opt){
156         case 'o':
157             fileName = optarg;
158             fileOutput = 1;
159             if (argv[3] != NULL) {
160                 fileInput = 1;
161                 src = fopen(argv[3], "r");
162             }
163             output = fopen (fileName, "w" );
164             break;
165
166         }
167     }
168
169     //check what parameter combination we have
170     if ((argv[1] != NULL) && (src == NULL)) {
```

```
171         fileInput = 1;
172         src = fopen(argv[1], "r");
173     }
174 
175     if (fileInput == 1) {
176         for (i = 0; (c = fgetc(src)) != EOF; ++i) {
177             if (c == '\n') { // if line number is encountered increment it
178                 lineNumber++;
179             }
180 
181             if (isalpha(c))
182                 buffer[j++] = tolower(c);
183             else {
184                 buffer[j++] = '\0';
185                 addWord(list, buffer, lineNumber);
186                 j = 0;
187             }
188         }
189     } else {
190         while ((c = getchar()) != EOF) {
191             if (c == '\n'){ // if line number is encountered increment it
192                 lineNumber++;
193             }
194 
195             if (isalpha(c))
196                 buffer[j++] = tolower(c);
197             else {
198                 buffer[j++] = '\0';
199                 addWord(list, buffer, lineNumber);
200                 j = 0;
201             }
202         }
203     }
204 
205 
206 
207 
208     /*
209      * print the list, skip the two nodes as the first is null and the second
210      * contains "" which is allowed by isalpha()
211      */
212     if(fileOutput == 1) {
213         printListFile(list->next->next, output);
214     }else {
215         printList(list->next->next);
216     }
217     fclose (src); // close the file
218     fclose (output);
219     return 0;
220 }
221 
```