```c
/*
 ============================================================================
 Name        : wc.c
 Author      : Nathaniel Churchill
 Professor   : Dr. David Smith
 Description :  C, Ansi-style progam to read text file and print how many times
                all the words occur in the given file
 ============================================================================
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.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 ) { //inserts node after a given node
         Node *current = NULL;
         current = malloc(sizeof(Node));
         current->word = malloc(strlen(word) + 1);
         strcpy(current->word, word);
         current->next = NULL;
         current->count = 0;
         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 *   This function accepts a pointer to a node and pointer to a character.
 61 *   The function then finds the node or creates a new node recursively in
 62 *   ascending order
 63 *
 64 *   head: a pointer to a node
 65 *   word: a pointer to a string
 66 *
 67 *   returns: a pointer to the found or created node
 68 */
 69 Node *findNodeForWord(Node *head, char *word){
 70     if (head->next == NULL){
 71         Node *insertNode = makeNode(head, word); //insert after the head
 72         return insertNode;
 73     }else if (strcmp(head->next->word, word) == 0){//stuff in the list
 74         return head->next;
 75     }else if (strcmp(head->next->word, word) < 0){ // list word is less than given word
 76         head = head->next;
 77         findNodeForWord(head, word);
 78     }else if (strcmp(head->next->word, word) > 0){
 79         Node *linkNode = head->next;
 80         Node *insertedNode = makeNode(head, word);
 81         insertedNode->next = linkNode;
 82         return insertedNode;
 83     }
 84
 85 }
 86
 87 /*
 88 *   addWord handles the adding and incrementing of a word
 89 *
 90 *   head: a pointer to a Node
 91 *   word: a pointer to a string
 92 */
 93 static void addWord(Node *head, char *word){
 94     Node *nodeForWord = findNodeForWord(head, word);
 95     nodeForWord->count++;
 96 }
 97
 98
 99 int main (void){
100   char c;
101   int charCount = 0;
102   char buffer[100];
103   //initialize the list with appropriate values
104   Node *list = malloc(sizeof(Node));
105   list->next = NULL;
106   list->count = 0;
107
108   // read words and add them to the list
109   while ( (c = getchar()) != EOF ) {
110     if (isalpha(c))
111       buffer[charCount++] = tolower(c);
112     else {
113       buffer[charCount++] = '\0';
114       addWord(list, buffer);
```

```
115       charCount = 0;
116     }
117   }
118
119   /*
120   * print the list, skip the two nodes as the first is null and the second
121   * contains "" which is allowed by isalpha()
122   */
123   printList(list->next->next);
124   return 0;
125 }
126
```