Binary Tree

This reads words from the standard input, and sorts them in increasing ASCII order. It then prints the words.

/*
 * A Word Listing Program
 *
 * Problem:
 * Given input, print all the words, one per line, and put
 * line number in front of the word. A "word" is a maximal
 * sequence of alphanumerical characters.
 *
 * Author:
 * Matt Bishop (bishop@cs.ucdavis.edu)
 */
#include <stdio.h>
#include <ctype.h>
#include <malloc.h>

/*
 * macros
 *
#define SZBUFFER 1024 /* max length of an input line */
*/

/*
 * the tree node structure
 */
struct node {
    char *word; /* pointer to word being stored */
    int count; /* number of times word found */
    struct node *left; /* points to left subtree */
    struct node *right; /* points to right subtree */
};
#define N_NULL((struct node *) NULL)/* NULL node pointer */

/*
 * forward declarations (prototypes)
 */
struct node *ninsert(struct node *, char *); /* insert word into tree */
void nprint(struct node *); /* print the tree */
struct node *nalloc(char *); /* create a new node */

/*
 * this is the main routine
 *
 * arguments: ignored
 * return: exits with 0
 * function: read from stdin, break each line into words,
 *           add words to binary tree, and when input ends
 *           print the words, one per line, prefixed by count
 *
 * exceptions: none
 */
void main(void)
{ char line[SZBUFFER]; /* input line */
char word[SZBUFFER]; /* word */
register int lineno = 0; /* current line number */
register char *p; /* current line position */
register char *w; /* used to load line */
struct node *root; /* root of the tree */

/* * read the file a line at a time */
while(fgets(line, 101, stdin) != NULL){
    /* * begin at the beginning of a new line */
    lineno++;
    p = line;
    /* * loop until end of line */
    while(*p){
        /* * skip leading non-word stuff */
        while (*p && !isalnum(*p))
            p++;
        /* * stopped at word; put it into word[] */
        w = word;
        while(isalnum(*p))
            *w++ = *p++;
        *w = '\0';
        /* * insert it into tree */
        if (word[0])
            root = ninsert(root, word);
    }
}

/* * print the tree */
nprint(root);

/* * say goodbye nicely */
exit(0);

/******************** the tree stuff ********************/
/*
* nalloc: create a node
* arguments: char *word  word to be inserted into tree
* return: pointer to newly-created node
* output: none
* exceptions: no memory for node at word %s (returns N_NULL)
  * no memory for word at word %s (returns N_NULL)
*/

struct node *nalloc(char *word)
{
    register struct node *n; /* tmp ptr for new node */

    /* allocate the node and space for the word */
    if ((n = malloc(sizeof(struct node))) == NULL){
        fprintf(stderr, "no memory for node at word %s
", word);
        return(N_NULL);
    }
    if ((n->word = malloc(strlen(word) + 1) * sizeof(char))) == NULL){
        fprintf(stderr, "no memory for word at word %s
", word);
        return(N_NULL);
    }
    /* now set the components of the node and return success */
    (void) strcpy(n->word, word);
    n->count = 1;
    n->left = n->right = N_NULL;
    return(n);
}

/* ninsert: insert a word into the binary tree */
/* arguments:  struct node *base  root of tree
* char *word  word to be inserted into tree
* return: pointer to root of tree with word in it
* output: none
* exceptions: none */

struct node *ninsert(struct node *base, char *word)
{
    register int cmp; /* result of comparison */

    /* see if we have no tree,
     * in which case the new node IS the tree */
    if (base == N_NULL)
        return(nalloc(word));
    /* we have one -- where do we go */
if ((cmp = strcmp(word, base->word)) == 0){
    /* stay here */
    base->count++;
} else if (cmp < 0){
    /* insert in left subtree */
    base->left = ninsert(base->left, word);
} else{
    /* insert in right subtree */
    base->right = ninsert(base->right, word);
}
/*
 * return pointer to root of tree
 */
return(base);

/*
 * nprint: print the tree inorder
 *
 * arguments:   struct node *base root of tree
 * return:      nothing
 * output:      count and word, one per line, in order
 * exceptions:  none
 */
void nprint(struct node *base)
{
    /*
     * no tree means nothing to print
     */
    if (base == N_NULL)
        return;
    /*
     * print away!
     */
    /* print left subtree */
    nprint(base->left);
    /* print node contents */
    printf("%d\t%s\n", base->count, base->word);
    /* print right subtree */
    nprint(base->right);
}