runner% **cat treealt.c**

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAXWORD 100
#define MAXS 100

struct tnode {
   char *word;
   int count;
   struct tnode *left;
   struct tnode *right;
};
struct tnode *addtree(struct tnode *, char *);
struct tnode *newnode(char *);
void treeprint(struct tnode *);
struct tnode *talloc(void);
int getword(char *, int);
char *strdupl(char *);
/* stuff below is for a stack of pointers to nodes */
void push(struct tnode *);
struct tnode *pop(void);
int size();
struct tnode *s[MAXS];
int sp = 0;

/* word frequency count -- non-recursive version */
main()
{
   struct tnode *root;
   char word[MAXWORD];
   root = NULL;
   while (getword(word, MAXWORD) != EOF)
      if (isalpha(word[0]))
         root = addtree(root, word);
   treeprint(root);
   return 0;
}

/* addtree: add a node, non-recursive */
struct tnode *addtree(struct tnode *p, char *w)
{
   int cond;
   struct tnode *root = p;
   if (p == NULL) return newnode(w);
   for (;;) {
      if ((cond = strcmp(w, p -> word)) == 0) {
         (p -> count)++;
         return root;
      }
      if (cond < 0) {
         if ((p -> left) == NULL) {
            p -> left = newnode(w);
            return root;
         }
         else p = p -> left;
      }
      else if (cond > 0) {
         if ((p -> right) == NULL) {
            p -> right = newnode(w);
            return root;
         }
         else p = p -> right;
      }
   }
}

/* newnode: fix up a new node */
struct tnode *newnode(char *w)
{
   struct tnode *p = talloc();
   p -> word = strdupl(w);
   p -> count = 1;
   p -> left = p -> right = NULL;
   return p;
}

/* treeprint: in-order print of tree p, non-recursive
version */
void treeprint(struct tnode *p)
{
   for (;;) {
      while (p != NULL) {
         push(p);
         p = p -> left;
      }
      for (;;) {
         if (size() == 0) return;
         p = pop();
         printf("%4d %s\n", p -> count, p -> word);
         if ((p -> right) != NULL) {
            p = p -> right;
            break;
         }
      }
   }
}
```

```
/* talloc: make a tnode */
struct tnode *talloc(void)
{
   return (struct tnode *)
      malloc(sizeof(struct tnode));
}

/* getword: get next word or charcter from input */
int getword(char *word, int lim)
{
   int c;
   char *w = word;
   while (isspace(c = getchar()))
      ;
   if (c != EOF)
      *w++ = c;
   if (!isalpha(c)) {
      *w = '\0';
      return c;
   }
   for ( ; --lim > 0; w++)
      if (!isalpha(*w = getchar())) {
         ungetc(*w, stdin);
         break;
      }
   *w = '\0';
   return word[0];
}

/* strdupl: make a duplicate of s. (builtin) */
char *strdupl(char *s)
{
   char *p;
   p = (char *) malloc(strlen(s)+1);
   if (p != NULL)
      strcpy(p, s);
   return p;
}

/* push, pop, size: stack routines */
void push(struct tnode *p)
{
   s[sp++] = p;
}

struct tnode *pop(void)
{
   return s[--sp];
}

int size(void)
{
   return sp;
}
```

```
runner% lint -m -u treealt.c

function returns value which is always ignored
    strcpy         printf         ungetc
runner% cc -o treealt treealt.c
runner% treealt <treealt.c
   2 EOF
   2 MAXS
   3 MAXWORD
   8 NULL
   5 a
   1 add
   4 addtree
   1 all
   1 below
   2 break
   1 builtin
   6 c
  15 char
   1 charcter
   4 cond
   5 count
   1 ctype
   1 d
   2 define
   1 duplicate
   3 else
   1 fix
   5 for
   1 frequency
   1 from
   1 get
   2 getchar
   4 getword
   4 h
  13 if
   1 in
   4 include
   1 input
  10 int
   2 is
   3 isalpha
   1 isspace
(rest omitted)
```