【C】Day2

Course	Advanced C
Study Date	@March 16, 2022

【Ch6】Struct

6.5 Self-referential Structures

Suppose we want to handle the more general problem of counting the occurrences of all the words in some input.

We can keep the set of words seen so far sorted at all times, by placing each word into its proper position in the order as it arrives. This shouldn't be done by shifting words in a linear array, though-that also takes too long.

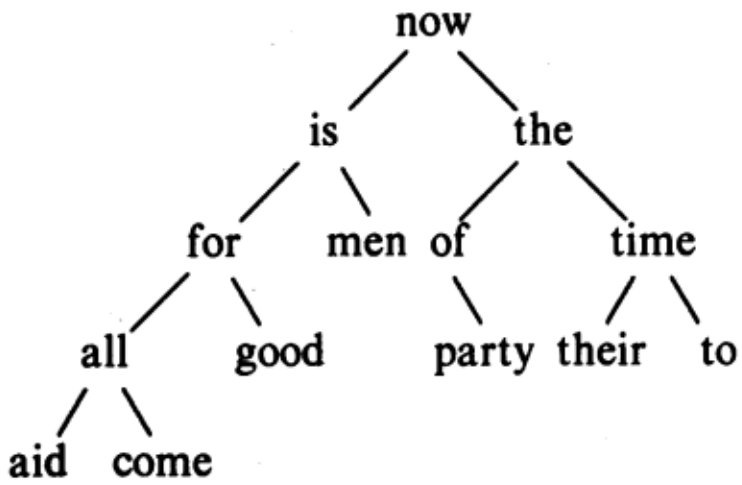
Instead we will use a data structure called a binary tree.

The tree contains one "node" per distinct word; each node contains

- A pointer to the text of the word
- A count of the number of occurrences
- A pointer to the left child node
- A pointer to the right child node

The nodes are maintained so that at any node the left subtree contains only words that are lexicographically less than the word at the node, and the right subtree contains only words that are greater.

For example, this is the tree for the sentence "now is the time for all good men to come to the aid of their party.



The structure definition of a node:

```
struct tnode {
  char *word;
  int count;
  struct tnode *left;
  struct tnode*right;
};
```

A structure is illegal to contain an instance of itself, but `struct tnode *left` declares `left` to be a pointer to a `tnode`, not a `tnode` itself.

The code for the whole program is surprisingly small.

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define MAXWORD 100
struct tnode *addtree(struct tnode *, char *);
void treeprint(struct tnode *);
int getword(char *, int);

/* word frequency count */
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;
}
```