You will write a program to build a binary search tree using the node structure below:

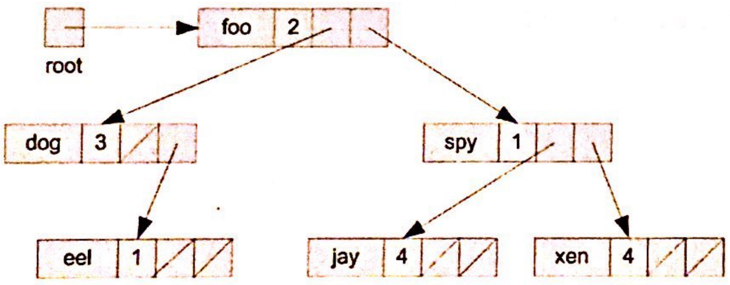
```
typedef struct node node;

struct node {
  char word[WORDLEN];
  int  freq;
  node *lchild, *rchild;
};
```

The program will accept the command-line arguments as a file name.

```
$ program lincoln.txt
```

It extracts English words from the file (use the given **getword()** function), and build a binary search tree, in which words less than a word of a node is on a left subtree and words greater than a word of a node is on a right subtree as shown in the figure. A root is pointing the root node, from which all nodes can be accessed.



Initially, the root has a null value indicating no tree. A new word is inserted to the rooted tree with **freq** is one. It is natural that the first word becomes the root node. The next word is inserted either to the left-subtree or the right subtree depending on the comparison result with the rooted node. If a word is already in the tree, its **freq** increments. You will print the following output for each text file given in the speech.tar.

- The height of the tree, where is the maximum of the heights of left subtree and right subtree incremented by a factor of 1.
- The number of distinct words,
- The number of words,
- The tree where each word indented as its depth.

You are recommended to implement the following functions:

```
void insert(node **rp, char *w);      // to insert a word to the tree rooted by rp
node *search(node *root, char *w);    // to return a pointer of a node containing string s
void delete_tree(node *root);         // to delete a subtree rooted by the root.
int height(node *root);               // to return the height of a subtree rooted by the root
int count(node *root);                // to return the number of distinct words of a subtree
int total(node *root);                // to return the number of words of a subtree
int print(node *root, int depth, FILE *fout);// to print a subtree whose root is indented by depth
```

Note that a recursive function makes it easy to design algorithms needed as given an example of **count()** function.

A sample output is as below

```
Thank (1)
  Mr (1)
    General (1)
      Executive (1)
        …
            But (2)
          Excellencies (1)
    Secretary (1)
      My (3)
        Nam (2)
          Myself (2)
          …
  you (6)
    UNICEF (3)
      There (1)
        Tomorrow (1)
          …
-- summary of ../data/bts-unicef.txt
height = 23
number of nodes = 335
number of words = 740
```