7-9（树） 目录树 （30 分）

在ZIP归档文件中，保留着所有压缩文件和目录的相对路径和名称。当使用WinZIP等GUI软件打开ZIP归档文件时，可以从这些信息中重建目录的树状结构。请编写程序实现目录的树状结构的重建工作。

输入格式:

输入首先给出正整数N（≤10^4），表示ZIP归档文件中的文件和目录的数量。随后N行，每行有如下格式的文件或目录的相对路径和名称（每行不超过260个字符）：

- 路径和名称中的字符仅包括英文字母（区分大小写）；
- 符号“\”仅作为路径分隔符出现；
- 目录以符号“\”结束；
- 不存在重复的输入项目；
- 整个输入大小不超过2MB。

输出格式:

假设所有的路径都相对于root目录。从root目录开始，在输出时每个目录首先输出自己的名字，然后以字典序输出所有子目录，然后以字典序输出所有文件。注意，在输出时，应根据目录的相对关系使用空格进行缩进，每级目录或文件比上一级多缩进2个空格。

输入样例:

```
7
b
c\
ab\cd
a\bc
ab\d
a\d\a
a\d\z\
```

输出样例:

```
root
  a
    d
      z
      a
    bc
  ab
    cd
    d
  c
  b
```

```
#include <bits/stdc++.h>
const int MAXSIZE = 10010;
using namespace std;
typedef struct Directory
{
    char key[270];
```

```
    int DirectoryNum;
    int FileNum;
    struct Directory * DirectoryChild[MAXSIZE];
    struct File * FileChild[MAXSIZE];

}Directory;
typedef struct File
{
    char key[270];
}File;
typedef Directory *DirectoryPtr;
typedef Directory *Tree;
typedef File *FilePtr;
int N;
bool cmp1(FilePtr p1, FilePtr p2)
{
    return strcmp(p1->key, p2->key) < 0;
}
bool cmp2(DirectoryPtr p1, DirectoryPtr p2)
{
    return strcmp(p1->key, p2->key) < 0;
}
bool Insert(Tree &t, char s[270])
{
    int len = strlen(s);
    if(len == 0)
    {
        return false;
    }
    else
    {
        char key[270], temp[270];
        int IsFile = 1;
        for(int i = 0; i < len; ++i)
        {
            if(s[i] == '\\')
            {
                IsFile = 0;
                strncpy(key, s, i);
                strncpy(temp,s + i + 1, len - i - 1);
                temp[len - i - 1] = '\0';
                key[i] = '\0';
//                printf("==%s+%s\n", key, temp);
                break;
            }
        }
        if(IsFile == 1)
```

```
        {
            FilePtr fp = (FilePtr)malloc(sizeof(File));
            strcpy(fp->key, s);
            t->FileChild[t->FileNum++] = fp;
//            sort(t->FileChild, t->FileChild + t->FileNum, cmp1);
            return true;
        }
        else
        {
            DirectoryPtr dp;
            int IsFind = 0;
            for(int j = 0; j < t->DirectoryNum; ++j)
                if(strcmp(t->DirectoryChild[j]->key, key) == 0)
                {
                    IsFind = 1;
                    dp = t->DirectoryChild[j];
                    break;
                }
            if(IsFind != 1)
            {
                dp = (DirectoryPtr)malloc(sizeof(Directory));
                strcpy(dp->key, key);
                dp->DirectoryNum = dp->FileNum = 0;
                t->DirectoryChild[t->DirectoryNum++] = dp;
//                sort(t->DirectoryChild, t->DirectoryChild + t->DirectoryNum, cmp2);
            }
            Insert(dp, temp);
        }
    }
}
Tree BuildTree()
{
    char s[270];
    Tree T;
    T = (Tree)malloc(sizeof(Directory));
    strcpy(T->key, "root");
    T->DirectoryNum = T->FileNum = 0;
    scanf("%d", &N);
    for(int i = 1; i <= N; ++i)
    {
        scanf("%s", s);
        Insert(T, s);
    }
    return T;
}
void Print(Tree t, int top)
{
```

```
    for(int j = 0; j < top; ++j)
        printf("  ");
    printf("%s\n", t->key);
    sort(t->DirectoryChild, t->DirectoryChild + t->DirectoryNum, cmp2);
    for(int i = 0; i < t->DirectoryNum; ++i)
    {
        Print(t->DirectoryChild[i], top + 1);
    }
    sort(t->FileChild, t->FileChild + t->FileNum, cmp1);
    for(int i = 0; i < t->FileNum; ++i)
    {
        for(int j = 0; j < top + 1; ++j)
            printf("  ");
        printf("%s\n", t->FileChild[i]->key);
    }
}
int main()
{
    Tree T;
    T = BuildTree();
    Print(T, 0);
}
```