DS/tinery-tree-using-linked-list.c

```
//tinery tree implementation using linnked list.
#include <stdio.h>
#include <stdlib.h>

typedef struct TTree{
    int data;
    struct TTree *l;
    struct TTree *m;
    struct TTree *r;
}ttree;


void add(ttree **t, int new_data){
    ttree *new_ttree = (ttree*)malloc(sizeof(ttree));
    new_ttree->data = new_data;
    new_ttree->l = NULL;
    new_ttree->m = NULL;
    new_ttree->r = NULL;
    *t = new_ttree;
}

void delete(ttree **t){
    (*t)->l = NULL;
    (*t)->m = NULL;
    (*t)->r = NULL;
    (*t) = NULL;
}

void display(ttree *t){
    printf("%d (",t->data);
    if(t->l){
        printf("%d,",t->l->data);
    }else{
        printf("0,");
    }
    if(t->m){
        printf("%d,",t->m->data);
    }else{
        printf("0,");
    }
    if(t->r){
        printf("%d)",t->r->data);
    }else{
        printf("0)");
    }
    printf("\n");
}

void preorderTraversal(ttree *root){
    if(root){
        display(root);
        preorderTraversal(root->l);
        preorderTraversal(root->m);
        preorderTraversal(root->r);
    }
}

int size(ttree* root) {
    if (root == NULL)
```

```
        return 0;
    else
        return size(root->l) + 1 + size(root->r) + size(root->m);
}

void main(){
    ttree *t1 = NULL;
    add(&t1,1);
    add(&t1->l,2);
    add(&t1->m,3);
    add(&t1->r,4);
    add(&t1->l->l,5);
    add(&t1->l->m,6);
    add(&t1->l->r,7);
    add(&t1->m->l,8);
    add(&t1->m->m,9);
    add(&t1->m->r,10);
    add(&t1->r->l,11);
    add(&t1->r->m,12);
    add(&t1->r->r,13);
    preorderTraversal(t1);
    printf("The size of the tree is %d",size(t1));
}
```

OUTPUT

```
PS S:\WorkSpace\CollegeWork\DataStructure> gcc .\tinary-tree-using-linked-list.c
PS S:\WorkSpace\CollegeWork\DataStructure> ./a
1 (2,3,4)
2 (5,6,7)
5 (0,0,0)
6 (0,0,0)
7 (0,0,0)
3 (8,9,10)
8 (0,0,0)
9 (0,0,0)
10 (0,0,0)
4 (11,12,13)
11 (0,0,0)
12 (0,0,0)
13 (0,0,0)
The size of the tree is 13
PS S:\WorkSpace\CollegeWork\DataStructure>
```