Problem Set 5

Problem 5.1

In this problem, we continue our study of linked lists. Let the nodes in the list have the following structure

```
struct node
{
    int data;
    struct node* next;
};
```

Use the template in Lec06 to add elements to the list.

(a) Write the function `void display(struct node* head)` that displays all the elements of the list.

(b) Write the function `struct node* addback(struct node* head, int data)` that adds an element to the end of the list. The function should return the new head node to the list.

(c) Write the function `struct node* find(struct node* head, int data)` that returns a pointer to the element in the list having the given data. The function should return NULL if the item does not exist.

(d) Write the function `struct node* delnode(struct node* head, struct node* pelement)` that deletes the element pointed to by **pelement** (obtained using find). The function should return the updated head node. Make sure you consider the case when `pelement` pointers to the head node.

(e) Write the function `void freelist(struct node* head)` that deletes all the element of the list. Make sure you do not use any pointer after it is freed.

(f) Write test code to illustrate the working of each of the above functions.

All the code and sample outputs should be submitted.

Answer:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node
{
    int data;
    struct node* next;
};

struct node* nalloc(int data)
{
    struct node* p = (struct node*)malloc(sizeof(struct node));
```

```c
    if (p != NULL)
    {
        p->data = data;
        p->next = NULL;
    }
    return p;
}

struct node* addfront(struct node* head, int data)
{
    struct node* p = nalloc(data);
    if (p == NULL) return head;
    p->next = head;
    return p;
}

void display(struct node* head)
{
    struct node* p = head;
    while (p != NULL)
    {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n");
}

struct node* addback(struct node* head, int data)
{
    struct node* p = nalloc(data);
    struct node* curr = NULL;
    /* consider empty list */
    if (head == NULL)
        head = p;
    else
    {
        curr = head;
        while (curr->next != NULL)
            curr = curr->next;
        curr->next = p;
        p->next = NULL;
    }
    return head;
}

void freelist(struct node* head)
{
    struct node* curr = NULL;
    while (head != NULL)
    {
        curr = head;
        head = head->next;
        free(curr);
    }
}

struct node* find(struct node* head, int data)
{
    struct node* curr = NULL;
    curr = head;
    while (curr != NULL)
    {
        if (curr->data == data)
            return curr;
        curr = curr->next;
    }
    return NULL; /* if not found, return NULL */
}

struct node* delnode(struct node* head, struct node* pelement)
{
    struct node* curr = NULL;
    /* if pelement is head */
    if (pelement == head)
    {
        head = head->next;
        free(pelement);
        pelement = NULL;
    }
    else
    {
        curr = head;
        while (curr != NULL)
        {
            if (curr->next == pelement)
            {
                curr->next = pelement->next;
                pelement->next = NULL;
                free(pelement);
            }
            curr = curr->next;
        }
    }
    return head;
}

int main(void)
{
    struct node* head = NULL;

    /* test addfront and display */
    puts("test addfront and display");
    head = addfront(head, 1);
    head = addfront(head, 2);
    puts("should display: 2 1");
    display(head);
    /* test freelist */
    puts("test freelist");
    freelist(head);
    head = NULL;
    puts("should display: ");
    display(head);

    /* test addback */
    puts("test addback");
    head = addback(head, 1);
    head = addback(head, 2);
    puts("should display: 1 2");
    display(head);

    /* test find */
    puts("test find");
    struct node* n1 = NULL;
    /* test find whether value is found or not */
    // n1 = find(head, 0); /* test when value not found */
    n1 = find(head, 2);
    if (n1 == NULL)
        puts("value not found");
    else
        puts("value found");

    /* test delnode */
    puts("test delnode");
    head = delnode(head, n1);
    puts("should display: 1 ");
    display(head);

    /* test freelist again */
    puts("test freelist again");
    freelist(head);
    head = NULL;
    puts("should display: ");
    display(head);

    return 0;
}
```

Notice: My code for problem 5.1 is a little different from code the official website offered. The code needs further improving when it comes to `struct node* delnode(struct node* head, struct node* pelement)`, the situation when `pelement` is NULL is not considered in the code. However, the code satisfies all the requirement for the questions.

Problem 5.2

In this problem, we continue our study of binary trees. Let the nodes in the tree have the following structure

```
struct tnode
{
    int data;
    struct tnode* left;
    struct tnode* right;
};
```

Use the template in Lec06 to add elements to the list.

(a) Write the function `struct tnode* talloc(int data)` that allocates a new node with the given data.

(b) Complete the function `addnode()` by filling in the missing section. Insert elements 3, 1, 0, 2, 8, 6, 5, 9 in the same order.

(c) Write function `void preorder(struct tnode* root)` to display the elements using pre-order traversal.

(d) Write function `void inorder(struct tnode* root)` to display the element using in-order traversal. Note that the elements are sorted.

(e) Write function `int deltree(struct tnode* root)` to delete all the elements of the tree. The function must return the number of nodes deleted. Make sure not to use any pointer after it has been freed. (Hint: use post-order traversal).

(f) Write test code to illustrate the working of each of the above functions.

All the code and sample outputs should be submitted.

Answer:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct tnode
{
    int data;
    struct tnode* left;
    struct tnode* right;
};

struct tnode* talloc(int data)
{
    struct tnode* p = (struct tnode*)malloc(sizeof(struct tnode));
    if (p != NULL)
    {
        p->data = data;
        p->left = NULL;
        p->right = NULL;
    }
    return p;
}
```

```c
struct tnode* addnode(struct tnode* root, int data)
{
    struct tnode* p = NULL;
    /* termination condition */
    if (root == NULL)
    {
        /* allocate node */
        p = talloc(data);
        root = p;
        /* return new root */
        return root;
    }
    /* recursive call */
    else if (data < root->data)
        root->left = addnode(root->left, data);
    else
        root->right = addnode(root->right, data);
}

void preorder(struct tnode* root)
{
    struct tnode* curr = root;
    if (curr != NULL)
    {
        printf("%d ", curr->data);
        preorder(curr->left);
        preorder(curr->right);
    }
}

void inorder(struct tnode* root)
{
    struct tnode* curr = root;
    if (curr != NULL)
    {
        inorder(curr->left);
        printf("%d ", curr->data);
        inorder(curr->right);
    }
}

void postorder(struct tnode* root)
{
    struct tnode* curr = root;
    if (curr != NULL)
    {
        postorder(curr->left);
        postorder(curr->right);
        printf("%d ", curr->data);
    }
}

int deltree(struct tnode* root)
```

```
{
    int count = 0;
    struct tnode* curr = root;
    if (curr != NULL)
    {
        count += deltree(curr->left);
        count += deltree(curr->right);
        curr->left = NULL;
        curr->right = NULL;
        free(curr);
        curr = NULL;
        count++;
    }
    return count;
}

int main(void)
{
    /* test addnode and talloc */
    struct tnode* root = NULL;
    int num[] = {3, 1, 0, 2, 8, 6, 5, 9};
    for (int i = 0; i < 8; i++)
        root = addnode(root, num[i]);

    /* test preorder */
    puts("test preorder...");
    puts("should display: 3 1 0 2 8 6 5 9");
    preorder(root);
    printf("\n");

    /* test inorder */
    puts("test inorder...");
    puts("should display: 0 1 2 3 5 6 8 9");
    inorder(root);
    printf("\n");

    /* test postorder */
    puts("test postorder...");
    puts("should display: 0 2 1 5 6 9 8 3");
    postorder(root);
    printf("\n");

    /* test deltree */
    int number = 0;
    puts("test deltree...");
    puts("should display: 8");
    number = deltree(root);
    printf("%d\n", number);

    return 0;
}
```