Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.087: Practical Programming in C

IAP 2010

Problem Set 5 – Solutions

Pointers. Arrays. Strings. Searching and sorting algorithms.

Out: January 19, 2010. **Due:** January 20, 2010.

Problem 5.1

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

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

Use the template in Lec06 (slides 35,36) 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` points 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: Here's one possible implementation:

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

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

/*
    @function nalloc
    @desc     allocates a new node elements
    @returns  pointer to the new element on success, NULL on failure
    @param data [IN] payload of the new element
*/
struct node* nalloc(int data)
{
    struct node* p=(struct node*)malloc(sizeof(struct node));
    if(p!=NULL)
    {
        p->next=NULL;
        p->data=data;
    }
    return p;
}

/*
    @function addfront
    @desc     adds node to the front of the list
    @param    head [IN] current head of the list
    @param    data [IN] data to be inserted
    @return   updated head of the list
*/
struct node* addfront(struct node* head,int data)
{
    struct node* p=nalloc(data);
    if(p==NULL) return head; /*no change*/
    p->next=head;
    return p;
}

/*
    @function   display
    @desc       displays the nodes in the list
    @param      head [IN] pointer to the head node of the list
*/
void display(struct node* head)
{
    struct node* p=NULL;
    printf("list:");
    for(p=head;p!=NULL;p=p->next)
            printf("%d ",p->data);
    printf("\n");
}

/*
    @function addback
```

```c
    @desc      adds node to the back of the list
    @param     head [IN] current head of the list
    @param     data [IN] data to be inserted
    @return    updated head node
*/
struct node* addback(struct node* head,int data)
{
    struct node* p=nalloc(data);
    struct node* curr=NULL;
    if(p==NULL) return head;
    /*special case: empty list*/
    if(head==NULL)
    {
        head=p;
        return p;
    }
    else
    {
        /*find last element*/
        for(curr=head;curr->next!=NULL;curr=curr->next)
            ;
        curr->next=p;
        return head;
    }
}

/*
    @function freelist
    @desc      frees the element of the list
    @param     head [IN] pointer to the head node
*/
void freelist(struct node* head)
{
    struct node* p=NULL;
    while(head)
    {
        p=head;
        head=head->next;
        free(p);
    }
}

/*
    @function find
    @desc      finds the elements that contains the given data
    @param     head [IN] pointer to the head node
    @param     data [IN] payload to match
    @return    NULL if not found, pointer to the element if found
*/
struct node* find(struct node* head,int data)
{
    struct node* curr=NULL;
    for(curr=head;curr->next!=NULL;curr=curr->next)
    {
        if(curr->data==data) return curr;
    }
    return NULL;
}
```

```c
/*
    @function delnode
    @desc     deletes a node
    @param    head [IN] pointer to the head node
    @param    pnode [IN] pointer to the element to be removed
    @return   updated head node
*/
struct node* delnode(struct node* head,struct node* pnode)
{
    struct node* p=NULL;
    struct node* q=NULL;
    for(p=head;p!=NULL && p!= pnode; p=p->next)
        q=p; /*follows p*/
    if(p==NULL) /*not found*/
        return head;
    if(q==NULL) /*head element*/
    {
        head=head->next;
        free(p);
    }
    else
    {
        q->next=p->next; /*skip p*/
        free(p);
    }
    return head;
}
/* @function main
   @desc     tests linked-list implementation
*/

int main()
{
    /*test addfront*/
    struct node* head=NULL;/*head node*/
    struct node* np=NULL; /*node pointer*/
    puts("should display empty");
    display(head); /*should print empty*/

    /*test add front*/
    head=addfront(head,10);
    head=addfront(head,20);
    puts("should display 20,10");
    display(head);

    /*test free list*/
    freelist(head);head=NULL;
    puts("should display empty");
    display(head);

    /*test add back*/
    head=addback(head,10);
    head=addback(head,20);
    head=addback(head,30);
    puts("should display 10,20,30");
    display(head);

    /*test find*/
    np=find(head,-20);
```

```c
    puts("should display empty");
    display(np);

    np=find(head,20);
    puts("should display 20,30");
    display(np);

    /*test delnode*/
    head=delnode(head,np);
    puts("should display 10,30");
    display(head);

    np=find(head,10);
    head=delnode(head,np);
    puts("should display 30");
    display(head);

    /*clean up*/
    freelist(head);
    return 0;
}
```

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 (slides 41) 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 elements 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: Here's one possible implementation:

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

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

/*
    @function talloc
    @desc     allocates a new node
    @param    data [IN] payload
    @return   pointer to the new node or NULL on failure
*/
struct tnode* talloc(int data)
{
    struct tnode* p=(struct tnode*)malloc(sizeof(struct tnode));
    if(p!=NULL)
    {
        p->data=data;
        p->left=p->right=NULL;
    }
    return p;
}

/*
    @function addnode
    @desc     inserts node into the tree
    @param    data [IN] data to be inserted
    @returns  updated root to the tree
*/
struct tnode* addnode(struct tnode* root,int data)
{
   if(root==NULL)
   {
        struct tnode* node=talloc(data);
        return (root=node);
   }
   else if(data<root->data)
   {
       root->left=addnode(root->left,data);
   }
   else
   {
       root->right=addnode(root->right,data);
   }
   return root;
}

/*
    @function preorder
    @desc     prints elements in pre-order
    @param    root [IN] pointer to the root of the tree
    @returns  nothing
*/
```

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

/*
    @function inorder
    @desc     prints elements in in-order
    @param    root [IN] pointer to the root of the tree
    @returns  nothing
*/
void inorder(struct tnode* root)
{
    if(root==NULL) return;
    inorder(root->left);
    printf("%d ",root->data);
    inorder(root->right);
}

/*
    @function deltree
    @desc     delete nodes of the tree
    @param    root [IN] pointer to the root of the tree
*/
int deltree(struct tnode* root)
{
    int count=0;
    if(root==NULL) return;
    count+=deltree(root->left);
    count+=deltree(root->right);
    free(root);
    return ++count;
}
/*
    @function main
    @desc     tests binary tree functions
*/
int main()
{
    struct tnode* root=NULL;
    int count=0;
    /*adding elements*/
    root=addnode(root,3);
    root=addnode(root,1);
    root=addnode(root,0);
    root=addnode(root,2);
    root=addnode(root,8);
    root=addnode(root,6);
    root=addnode(root,5);
    root=addnode(root,9);

    /*test preorder*/
    puts("should print 3,1,0,2,8,6,5,9");
    preorder(root);puts("");
    /*test inorder*/
    puts("should print 0,1,2,3,5,6,8,9");
```

```c
    inorder(root);puts("");
    /*test deltree*/
    count=deltree(root);root=NULL;
    puts("should expect 8 nodes deleted");
    printf("%d nodes deleted\n",count);
    return 0;

}
```

MIT OpenCourseWare
http://ocw.mit.edu

6.087 Practical Programming in C
January (IAP) 2010

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.