Problem Set 5
Linked lists, trees

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,**

<u>ANSWER</u>

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

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.

```
struct node* node_creation (int data)
{
struct node* p =(struct node*) malloc (sizeof(struct node));
if (p!=NULL)
    {
    p->next=NULL;
    p->data=data ;
    }
return p ;
}
```

```c
struct node* add_back(struct node* head,int val)
{
    struct node* ptr;
    ptr = node_creation(val);
    ptr->data = val;
    if(head == NULL)
    {
        ptr -> next = NULL;
        head = ptr;
        return head;

    }
    else
    {
        struct node *temp = head;
        while (temp -> next != NULL)
        {
            temp = temp -> next;
        }
        temp->next = ptr;
        ptr->next = NULL;
        return head;

    }
}
```

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.**

```c
struct node* find(struct node* head , int data)
{
        struct node* temp = head;
        for ( ; temp -> next !=NULL; temp = temp->next )
        {
            if ( temp->data == data )
            {
                printf("element found\n");
                return temp;
            }
        }
    return NULL;
}
```

(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.

```
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 ;
  if (p==NULL)
     return head;

  if ( q==NULL)
      {
          head=head->next ;
          free (p );
      }
  else
      {
          q->next = p->next ;
          free(p);
      }
  return head ;
}
```

(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.

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

(f) Write test code to illustrate the working of each of the above functions.
All the code and sample outputs should be submitted

```
int main()
{
   printf("enter 5 elements :\n");
    for(int i = 0; i < 5; i++)
    {
        int n ;
        scanf("%d", &n);
        head = add_back(head,n);
    }

    display(head);
    printf("enter the search element :\n");
    int ele;
    scanf("%d",&ele);
    if(find(head,ele) == NULL)
     {
        printf("element not found.\n\n");
     }
      printf("\n DELETE THE 3rd ELEMENT :\n");
      head = delnode(head,head->next->next);
      display(head);
      printf("\n ADD ELEMENT 10 AT FRONT :\n");
      head = add_front(head,10);
      display(head);
      freelist(head);
}
```

<u>OUTPUT</u>:

```
PS D:\VS C++ PROJECTS> gcc Linkedlist.c -o Linkedlist.exe
PS D:\VS C++ PROJECTS> ./Linkedlist.exe
enter 5 elements :
100 50 30 400 30
100 -> 50 -> 30 -> 400 -> 30 ->  NULL
enter the search element :
50
element found

 DELETE THE 3rd ELEMENT :
100 -> 50 -> 400 -> 30 ->  NULL

 ADD ELEMENT 10 AT FRONT :
10 -> 100 -> 50 -> 400 -> 30 ->  NULL
PS D:\VS C++ PROJECTS>
```

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 * l e f t ;
struct tnode * r i g h t ;
};
```

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.

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

(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.

```
struct tnode* addnode ( struct tnode* root , int data )
    {
        if ( root == NULL)
            {
                struct tnode* node = talloc(data);
                root = node;
                return ( root = node);
            }
        else if ( data < root->data )
            {
                root->left = addnode (root->left ,data);
            }
        else
            {
                root->right=addnode (root->right ,data);
            }
        return root ;
    }
```

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

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

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

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

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).

```
int deltree ( struct tnode* root )
{

    if ( root==NULL) return 0;
    int l = deltree(root->left );
    int r = deltree (root->right );
    free (root);
    return l+r+1 ;
}
```

(f) Write test code to illustrate the working of each of the above functions.
All the code and sample outputs should be submitted.

<u>OUTPUT</u>

```
int main()
{
printf("enter 8 elements :\n");
for(int i = 0; i < 8; i++)
{
    int n ;
    scanf("%d", &n);
    root = addnode(root,n);
}
printf("\n PREORDER TRAVERSAL \n");
preorder(root);
printf("\n INORDER TRAVERSAL \n");
inorder(root);
printf("\n the number node are - %d:",deltree(root));
}
```

```
enter 8 elements :
3
1
0
2
8
6
5

 PREORDER TRAVERSAL
3 1 0 2 8 6 5 9
 INORDER TRAVERSAL
1 0 2 3 8 6 5 9
```