19/2/24

1) Delete the middle node of linked list. (leet code)

→
```
struct node * deleteMid (struct node *head){

    struct node ptr, *fast = &ptr,
        *slow = &ptr;
    ~~slow~~ ptr.next = head;
    while ( fast->next && fast->next->next ){
        slow = slow->next;
        fast = fast->next->next;
    }
    slow->next = slow->next->next;
    return ptr.next ;
}
```

Test Result

head = [1,3,4,7,1,2,6]

Output

[1,3,4,1,2,6]

head = [1,2,3,4]

Output

[1,2,4]

head = [2,1]

Output

[2]

2) Odd Even linked list (leet code)

```
-> int len ( struct node *head) {
        int l =0
       struct node *p = head;
        while( p != NULL) {
            p = p -> next ;
             l = l +1;
        }
        return l;
   }


   struct node *oddEvenList( struct node *head) {
         if ( head = = NULL) {
               return head;
            }
          if ( len (head) = = 1) {
               return head;
           }


       struct node *p = head ;
       struct node * dum = NULL;
       struct node *u;
       while ( p != NULL) {

           struct node *n = (struct node *) malloc
                                   ( sizeof ( struct node));
            n -> data = p -> data;
             n -> next = NULL;
```

```
if ( dumm== NULL) {
        dumm = n
        u = dumm
    }
    else
      {
        ~~u =~~ u->next = n
          u = n
        }
     P = P->next;
  }
struct node *t = head;
struct node *O = dumm;
struct node *e = dumm->next;
while( o -> next != NULL){

        t -> ~~val~~ data = o->data;
        O = O -> next -> next;
        t = t -> next;
        if ( O == NULL){
              break;
          }
      }
    if ( len(head) %2 != 0) {
        t->data = o->data;
          t = t->next;
      }
    while (e!= NULL) {
          t -> ~~val~~ data = e->data ;
          ~~t=~~ if ( e->next == NULL){
            break; }
            e = e->next;
            t = t->next;
        }
        return head;
    }
```

Test Result

head = [1, 2, 3, 4, 5]

Output

[1, 3, 5, 2, 4]

head = [2, 1, 3, 5, 6, 4, 7]

Output

[2, 3, 6, 7, 1, 5, 4]

3) Write a program

ⓐ To construct Binary Search Tree

ⓑ Transverse the tree using inorder, postorder &, preorder.

ⓒ Display the elements in the tree.

→

```
#include <stdio.h>
~~#include <conio.h>~~
#include <stdlib.h>

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

 struct node *tree = NULL;
 struct node * insertElement (struct node *, int);
 void preorderTraversal (struct node *);
 void inorderTraversal (struct node *);
 void postorderTraversal (struct node *);
 void disp (struct node *);
 void main () {

        int option, val ;
        while (1) {
    printf ("\n **Main Menu** \n");
    printf ("\n 1. Insert Element ");
    printf ("\n 2. Preorder Traversal ");
    printf ("\n 3. Inorder Traversal ");
    printf ("\n 4. Postorder Traversal ");
    printf ("\n 5. Display ");
```

```
printf("\n 6. Exit ");
scanf("%d", & option);
switch(option) {

Case 1 : printf("\n Enter value of the
                   newnode: ");
         scanf("%d", &val);
         tree = insertElement(tree, val);
         break;

Case 2 : printf("\n The elements of
         the tree in preorder traversal
              are : \n");
         preorderTraversal(tree);
         break;

Case 3 :
      printf("\n The elements of the tree in
              inorder traversal are : \n");
      inorderTraversal(tree);
      break;

Case 4 :
      printf(" \n The elements of the tree in
               postorder traversal are : \n");
      postorder~~(tree)~~Traversal(tree);
      break;

Case 5 : printf(" \n The elements of the tree
                 are: \n");
         disp(tree);
case 6 : exit(0) ;
default : printf(" Invalid input ");
      }
    }
}

Struct node *insertElement ( struct node *tree,
        int val){

    struct node *ptr, *nodeptr, *parentptr;
ptr = (struct node *) malloc ( sizeof (struct node));
    ptr->data = val;
    ptr->left = NULL;
    ptr->right = NULL;
    if (tree == NULL){
            tree = ptr;
            tree->left = NULL;
            tree->right = NULL;
    }
    else {
            parentptr = NULL;
            nodeptr = tree;
        while (nodeptr != NULL){
            parentptr = nodeptr;
            if( val <nodeptr->data)
                nodeptr = nodeptr->left ;
            else
                nodeptr - nodeptr->right;
        }
        if (val < parentptr->data)
            parentptr->left = ptr;
        else
            parentptr -> right = ptr;
      }
    return tree
  }

void   preorderTraversal( struct node *tree ){
           if ( tree != NULL ) {
                printf ("%d \t", tree -> data);
                preorderTraversal ( tree -> left );
        }       preorderTraversal ( tree -> right );
      }
  }

void  inorderTraversal ( struct node *tree ){

          if ( tree != NULL ) {
               inorder Traversal ( tree -> left );
             printf ( "%d \t", tree -> data );
               inorder Traversal ( tree -> right );
        }
     }


void  postorder Traversal ( struct node *tree ){

        if ( tree != NULL ) {

            postorder Traversal ( tree -> left );
            postorder Traversal ( tree -> right );
         printf ( " %d \t" , tree -> data );
      }
  }
```

```
void disp (struct node *tree){

        if (tree != NULL) {
            disp(tree->left);
            printf("%d \t", tree->data);
            disp(tree->right);
        }

    }
```

Output

```
        ** Main Menu **
      1. Insert Element
      2. Preorder ~~Element~~ Traversal
      3. Inorder Traversal
      4. Postorder Traversal
      5. Display Elements
      6. Exit.
      1          (inserting)
Enter the value of the newnode: ~~3~~5
  1        (inserting)
Enter the value of the new node: 3
  1        (inserting)
~~8~~ Enter the value of the newnode: 8
  1        (inserting)
 Enter the value of the newnode: 2

  5      (displaying)
The elements of the tree are:
      2  3  5  8
```

2 (preorder)

The elements of the tree in preorder traversal are:

5 3 2 8

3 (inorder)

The elements of the tree in inorder traversal are:

2 3 5 8

~~The~~ Tho elements of the tree in postorder traversal are:

~~202~~ 2 3 8 5

6

.

19.02.24