Shrihari K Mangle
CSB2-74 12010517

LAB-07

Write a C program to create BST and perform following operations on it.:

1. Insertion
2. Deletion.
3. Display Tree Level-wise.
4. Find Min and Max in given tree.
5. Display only Leaf nodes.
6. Find Height of the Tree.
7. Find Mirror Image.

CODE :

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_Q_SIZE 500
// THIS TREE WILL LOOK LIKE THIS

    //    8
    //  / \
    // 3   10
  //  / \    \
  // 1   6    14
    //  / \   /
    // 4   7 13

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

typedef struct queue
{
    struct node *DATA;
    struct queue *next;
} queue;

struct node *top(queue *top)
{
```

```
    return top->DATA;
}
void push(queue **top, struct node *root)
{
    queue *new_top = (queue *)malloc(sizeof(queue));
    new_top->DATA = root;
    new_top->next = (*top);
    (*top) = new_top;
}

struct node *pop(queue **top)
{
    queue *new_top;
    struct node *root;

    root = (*top)->DATA;
    new_top = (*top);

    (*top) = (*top)->next;
    free(new_top);
    return (root);
}

struct node **createQueue(int *front, int *rear)
{
    struct node **queue = (struct node **)malloc(sizeof(struct node *) *
MAX_Q_SIZE);

    *front = *rear = 0;
    return queue;
}
void enQueue(struct node **queue, int *rear,
             struct node *new_node)
{
    queue[*rear] = new_node;
    (*rear)++;
}

struct node *deQueue(struct node **queue, int *front)
{
    (*front)++;
    return queue[*front - 1];
}

// PRINTING LEVEL ORDER TRAVERSAL
void printLevelOrder(struct node *root)
{
    int rear, front;
    struct node **queue = createQueue(&front, &rear);
    struct node *temp_node = root;
```

```
    while (temp_node)
    {
        printf("%d ", temp_node->data);

        if (temp_node->left)
            enQueue(queue, &rear, temp_node->left);

        if (temp_node->right)
            enQueue(queue, &rear, temp_node->right);

        temp_node = deQueue(queue, &front);
    }
}

struct node *createNode(int data)
{
    struct node *p;
    p = (struct node *)malloc(sizeof(struct node));
    p->data = data;
    p->left = NULL;
    p->right = NULL;

    return p;
}

// INSERTING INTO BST
void Insert(struct node *root, int data)
{
    struct node *prev = NULL;
    if (root == NULL)
    {
        printf("Cannot Insert into a tree\n");
        return;
    }
    while (root!=NULL)
    {
        prev = root;
        if (data == root->data)
        {
            printf("Binary Tree cannot have same entry\n");
            return;
        }

        else if(data<root->data){
            root = root->left;
        }
        else{
            root = root->right;
        }
    }
```

```
    struct node *new = createNode(data);
    if (data < prev->data)
    {
        prev->left = new;

    }
    else
    {
        prev->right = new;
    }

}

// MINIMUM VALUE IN BST
struct node * min(struct node *root)
{
    int val;
    while (root->left!=NULL)
    {
        root = root->left;
    }
    //val = root->data;
     return root;

}

// MAXIMUM VALUE IN BST
struct node *max(struct node *root){
    int val;
    while (root->right!=NULL)
    {
        root = root->right;
    }
    val = root->data;
     return root;

}

// DELETING A NODE IN BST
struct node * delete(struct node *root, int value){

    if (root==NULL)
    {
        return NULL;
    }
    if (root->left == NULL && root->right == NULL)
    {
        free(root);
        return NULL;
    }
```

```
    if (value<root->data)
    {
        root->left = delete(root->left, value);
    }
    else if (value>root->data)
    {
        root->right = delete(root->right, value);
    }
    else
    {
        if (root->right && root->left)
        {
            struct node *iPre = max(root->left);
            root->data = iPre->data;
            root->left = delete(root->left,iPre->data);
        }

        else
        {
            struct node *temp;
            //struct node *prev = root;
            if (root->left==NULL)
            {
                temp = root->right;
                free(root);
                return temp;

            }
            if(root->right==NULL){
                temp = root->left;
                free(root);
                return temp;
            }


        }

        return root;

    }


}

void swap(struct node * root){
    struct node *temp = root->left;
    root->left = root->right;
    root->right = temp;
}
```

```c
// MIRROR IMAGE OF BST
void mirrorTree(struct node *root){
    if (root!=NULL)
    {
        swap(root);
        mirrorTree(root->right);
        mirrorTree(root->left);
    }


}

// HEIGHT OF BST
int height(struct node *root){
    if (root==NULL)
    {
        return -1;
    }
    else
    {
        int l = height(root->left);
        int r = height(root->right);

         if (l > r)
            return (l + 1);
        else
            return (r + 1);
    }


}

// PRINTING ONLY LEAF NODES OF BST
void printleafNodes(struct node *root){
    if (root==NULL)
    {
        return;
    }

    if (!root->left && !root->right)
    {
        printf("%d ", root->data);
    }

    if (root->left)
    {
        printleafNodes(root->left);
    }

    if (root->right)
    {
```

```
        printleafNodes(root->right);
    }

}

// INORDER TRAVERSAL OF BST
void inOrder(struct node *root){
    if (root!=NULL)
    {
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }

}

int main()
{
    struct node *p1 = createNode(8);
    struct node *p2 = createNode(3);
    struct node *p3 = createNode(10);
    struct node *p4 = createNode(1);
    struct node *p5 = createNode(6);
    struct node *p6 = createNode(14);
    struct node *p7 = createNode(4);
    struct node *p8 = createNode(7);
    struct node *p9 = createNode(13);

    p1->left = p2;
    p1->right = p3;

    p2->left = p4;
    p2->right = p5;

    p5->left = p7;
    p5->right = p8;


    p3->right = p6;

    p6->left = p9;


    //    8
    //  / \
    // 3   10
  //  / \   \
  // 1   6   14
    //  / \  /
    // 4   7 13
```

```
    int choice, no, x;

    printf("\n");
    printf("LEVEL ORDER TRAVERSAL OF A GIVEN TREE : ");
    inOrder(p1);
    printf("\n\n");

    while (1)
    {
        printf("\n");
        printf("ENTER THE CHOICE : \n 1.INSERT NODE\n 2.DELETE NODE\n
3.DISPLAY TREE LEVEL WISE\n 4.MAXIMUM VALUE\n 5.MINIMUM VALUE\n 6.DISPLAY
LEAF NODES\n 7.HEIGHT OF TREE\n 8.MIRROR IMAGE OF TREE\n 9.INORDER
TRAVERSAL\n 10.EXIT\n");
        printf("\nPut here : ");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {

        case 1:
            printf("Enter the value to push : ");
            scanf("%d", &no);
            Insert(p1, no);
            printf("\n");
            break;

        case 2:
            printf("Enter the no to delete : ");
            scanf("%d", &x);
            printf("The deleted item is : %d", x);
            delete(p1, x);
            printf("\n");
            break;

        case 3:
            printf("Level-order traversal of a tree is : ");
            printLevelOrder(p1);
            printf("\n");
            break;

        case 4:
            printf("Maximum value is : %d", max(p1)->data);
            printf("\n");
            break;

        case 5:
             printf("Minimum value is : %d", min(p1)->data);
             printf("\n");
            break;
```

```
        case 6:
            printf("Leaf nodes are : ");
            printleafNodes(p1);
            printf("\n");
            break;

        case 7:
            printf("Height of a tree is : %d", height(p1));
            printf("\n");
            break;

        case 8:
            //printf("The mirror image of a given tree : ");
            printf("Given tree : ");
            inOrder(p1);
            printf("\n");
            mirrorTree(p1);
            printf("Mirrored tree : ");
            inOrder(p1);
            printf("\n");
            break;

        case 9:
            printf("Inorder traversal : ");
            inOrder(p1);
            printf("\n");
            break;

        case 10:
            exit(0);
            break;

        default:
            break;
        }
    }

    return 0;
}
```

Shrihari K Mangle
CSB2-74 12010517

OUTPUT :

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\> cd "e:\DS_ALGO\" ; if ($?) { gcc 27_Insertion_in_tree.c -o 27_Insertion_in_tree } ;

LEVEL ORDER TRAVERSAL OF A GIVEN TREE : 1 3 4 6 7 8 10 13 14


ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 1

Enter the value to push : 5


ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 9

Inorder traversal : 1 3 4 5 6 7 8 10 13 14
```

```
ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 2

Enter the no to delete : 10
The deleted item is : 10

ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 9

Inorder traversal : 1 3 4 5 6 7 8 13 14
```

```
ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 3

Level-order traversal of a tree is : 8 3 14 1 6 13 4 7 5

ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 4

Maximum value is : 14
```

```
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 5

Minimum value is : 1

ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 6

Leaf nodes are : 1 5 7 13

ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 7

Height of a tree is : 4
```

```
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 8

Given tree : 1 3 4 5 6 7 8 13 14
Mirrored tree : 14 13 8 7 6 5 4 3 1

ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 9

Inorder traversal : 14 13 8 7 6 5 4 3 1

ENTER THE CHOICE :
 1.INSERT NODE
 2.DELETE NODE
 3.DISPLAY TREE LEVEL WISE
 4.MAXIMUM VALUE
 5.MINIMUM VALUE
 6.DISPLAY LEAF NODES
 7.HEIGHT OF TREE
 8.MIRROR IMAGE OF TREE
 9.INORDER TRAVERSAL
 10.EXIT

Put here : 10

PS E:\DS_ALGO>
```