LAB 7

Aniket sambher
Reg no-190905466
Roll no-58
Section -A

1.Modify the solved exercise to find the balance factor for every node in the binary search tree.

Code

```
#include<stdio.h>
#include <stdlib.h>
typedef struct node
{
   int info;
   struct node *left,*right;
}  NODE;
int max(int a, int b)
{
   return (a>b)?a:b;
}
NODE* create(NODE *bnode,int x)
{
   NODE *getnode;
   if(bnode==NULL)
   {
      bnode=(NODE*) malloc(sizeof(NODE));
      bnode->info=x;
      bnode->left=bnode->right=NULL;
   }
   else if(x>bnode->info)
      bnode->right=create(bnode->right,x);
```

```
   else if(x<bnode->info)
      bnode->left=create(bnode->left,x);
   else
   {
   printf("Duplicate node\n");
   exit(0);
   }
return(bnode);
}
int height(NODE *root){
   if(root == NULL)
      return 0;
   else
      return (1+ max(height(root->left),height(root->right)));

}
void balance(NODE *root){
   if(root==NULL)
      return ;
   balance(root->left);
   printf("balance factor of node %d is %d \n",root->info,(height(root->left)-height(root->right)));
   balance(root->right);

}
void inorder(NODE *ptr)
{
if(ptr!=NULL)
{
   inorder(ptr->left);
   printf("%5d",ptr->info);
   inorder(ptr->right);
```

```
}
}
void postorder(NODE *ptr)
{
if(ptr!=NULL)
{
postorder(ptr->left);
postorder(ptr->right);
printf("%5d",ptr->info);
}
}
void preorder(NODE *ptr)
{
if(ptr!=NULL)
{
printf("%5d",ptr->info);
preorder(ptr->left);
preorder(ptr->right);
}
}
void main()
{
int n,x,ch,i;
NODE *root;
root=NULL;
while(1)
{
   printf("********************Output********************\n\n");
   printf("-----------Menu-----------\n");
   printf("1. Insert\n2. All traversals\n3. Exit\n4.find balance factor\n");
   printf("Enter your choice:");
```

```
   scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter node (do not enter duplicate nodes):\n");
scanf("%d",&x);
root=create(root,x);
break;
case 2: printf("\nInorder traversal:\n");
inorder(root);
printf("\nPreorder traversal:\n");
preorder(root);
printf("\nPostorder traversal:\n");
postorder(root);
printf("\n\n*****************************************");
break;
case 3: exit(0);
case 4:printf("\nbalance factor:\n");
   balance(root);
   break;

}
}
}
```

OUTPUT

```
----------Menu----------
1. Insert
2. All traversals
3. Exit
4.find balance factor
Enter your choice:1
Enter node (do not enter duplicate nodes):
5
********************Output********************

----------Menu----------
1. Insert
2. All traversals
3. Exit
4.find balance factor
Enter your choice:1
Enter node (do not enter duplicate nodes):
3
********************Output********************

----------Menu----------
1. Insert
2. All traversals
3. Exit
4.find balance factor
Enter your choice:1
Enter node (do not enter duplicate nodes):
7
********************Output********************
```

```
********************Output********************

-----------Menu-----------
1. Insert
2. All traversals
3. Exit
4.find balance factor
Enter your choice:1
Enter node (do not enter duplicate nodes):
1
********************Output********************

-----------Menu-----------
1. Insert
2. All traversals
3. Exit
4.find balance factor
Enter your choice:4

balance factor:
balance factor of node 1 is 0
balance factor of node 3 is 1
balance factor of node 5 is 1
balance factor of node 7 is 0
********************Output********************
```

PTO

2. Write a program to create the AVL tree by iterative insertion.

CODE

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

typedef struct node
{
   int info;
   struct node *left, *right;
} NODE;

struct Stack
{
   int top;
   unsigned capacity;
   NODE **array;
};

struct Stack *createStack(unsigned capacity)
{
   struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
   stack->capacity = capacity;
   stack->top = -1;
   stack->array = (NODE **)malloc(stack->capacity * sizeof(NODE *));
   return stack;
}

int isFull(struct Stack *stack)
```

```
{
   return stack->top == stack->capacity - 1;
}

int isEmpty(struct Stack *stack)
{
   return stack->top == -1;
}

void push(struct Stack *stack, NODE *item)
{
   if (isFull(stack))
      return;
   stack->array[++stack->top] = item;
   // printf("%d pushed to stack\n", item);
}

NODE *pop(struct Stack *stack)
{
   if (isEmpty(stack))
      return NULL;
   return stack->array[stack->top--];
}

NODE *peek(struct Stack *stack)
{
   if (isEmpty(stack))
      return NULL;
   return stack->array[stack->top];
```

```
}

int max(int x, int y)
{
   return x > y ? x : y;
}

int height(NODE *root)
{
   if (root == NULL)
      return 0;
   return 1 + max(height(root->left), height(root->right));
}

int getBalFactor(NODE *root)
{
   return height(root->left) - height(root->right);
}

NODE *rightRotate(NODE *y)
{
   NODE *x = y->left;
   NODE *T2 = x->right;
   x->right = y;
   y->left = T2;
   return x;
}

NODE *leftRotate(NODE *x)
```

```
{
    NODE *y = x->right;
    NODE *T2 = y->left;
    y->left = x;
    x->right = T2;
    return y;
}

NODE *create(NODE *root, int x)
{
    struct Stack *stack = createStack(100);
    NODE *newnode = (NODE *)malloc(sizeof(NODE));
    newnode->info = x;
    newnode->right = NULL;
    newnode->left = NULL;
    NODE *curr = root;
    NODE *trail = NULL;
    while (curr != NULL)
    {
        trail = curr;
        push(stack, trail);
        if (x < curr->info)
            curr = curr->left;
        else if (x > curr->info)
            curr = curr->right;
        else
        {
            printf("Duplicate element\n");
            exit(0);
```

```
    }
}
if (trail == NULL)
{
    trail = newnode;
    return trail;
}
else if (x < trail->info)
    trail->left = newnode;
else
    trail->right = newnode;
NODE *newRoot = root;
while (!isEmpty(stack))
{
    NODE *toBalance = pop(stack);
    NODE *prev = peek(stack);
    int balance = getBalFactor(toBalance);
    if (balance > 1 && x < toBalance->left->info)
    {
        toBalance = rightRotate(toBalance);
    }
    else if (balance < -1 && x > toBalance->right->info)
    {
        toBalance = leftRotate(toBalance);
    }
    else if (balance > 1 && x > toBalance->left->info)
    {
        toBalance->left = leftRotate(toBalance->left);
        toBalance = rightRotate(toBalance);
```

```
        }
        else if (balance < -1 && x < toBalance->right->info)
        {
            toBalance->right = rightRotate(toBalance->right);
            toBalance = leftRotate(toBalance);
        }
        if (prev != NULL && prev->info > toBalance->info)
        {
            prev->left = toBalance;
        }
        else if (prev != NULL)
        {
            prev->right = toBalance;
        }
        newRoot = toBalance;
    }
    return newRoot;
}

void inorder(NODE *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%5d", root->info);
        inorder(root->right);
    }
}
```

```
void postorder(NODE *root)
{
   if (root != NULL)
   {
      postorder(root->left);
      postorder(root->right);
      printf("%5d", root->info);
   }
}

void preorder(NODE *root)
{
   if (root != NULL)
   {
      printf("%5d", root->info);
      preorder(root->left);
      preorder(root->right);
   }
}

int printBalanceFactor(NODE *root)
{
   if (root != NULL)
   {
      printf("\nBalance factor of node with value %d : %d", root->info, getBalFactor(root));
      printBalanceFactor(root->left);
      printBalanceFactor(root->right);
   }
}
```

```
void main()
{
   int n, x, ch, i;
   NODE *root;
   root = NULL;
   printf("-----------Menu----------\n");
   printf(" 1. Insert\n 2. All traversals\n 3. Get Balance Factor\n 4. Exit\n");
   while (1)
   {
      printf("Enter your choice : ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         printf("Enter node (do not enter duplicate nodes) : ");
         scanf("%d", &x);
         root = create(root, x);
         break;
      case 2:
         printf("\n*****************************************");
         printf("\nInorder traversal   : ");
         inorder(root);
         printf("\nPreorder traversal  : ");
         preorder(root);
         printf("\nPostorder traversal : ");
         postorder(root);
         printf("\n\n*****************************************\n");
         break;
```

```
        case 3:
            printf("\n*****************************************");
            printBalanceFactor(root);
            printf("\n\n*****************************************\n");
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid Choice\n");
        }
    }
}
```

```
Enter the Number of Nodes of The AVL Tree :
4
Enter the 1 Node in the AVL Tree
5
Enter the 2 Node in the AVL Tree
6
Enter the 3 Node in the AVL Tree
7
Enter the 4 Node in the AVL Tree
8
The AVL Tree Inserted has the Preorder Traversal given by :
6 5 7 8
```