DSA – ASSIGNMENT

QUE. – Write a program to construct an AVL tree for the given set of keys. Also write function for deleting a key from the given AVL tree.

ANS. -

```
// AVL tree implementation

#include <stdio.h>

#include <stdlib.h>

// Create Node

struct node {

  int key;

  struct node *left;

  struct node *right;

  int height;

};

//declaring all the functions of AVL tree

int max(int a, int b);

int height(struct node *N);

int max(int a, int b);

struct node *newNode(int key);

struct node *rightRotate(struct node *y);

struct node *leftRotate(struct node *x);

int getBalance(struct node *N);

struct node *insertNode(struct node *node, int key);

struct node *minValueNode(struct node *node);

struct node *maxValueNode(struct node *node);

struct node *search(struct node *root, int key);

struct node *deleteNode(struct node *root, int key);

void preOrder(struct node *root);

void inOrder(struct node *root);
```

```
void postOrder(struct node *root);


//main function
int main() {
   //declaring local variables
 int num = -1, ch = 0;
 struct node *root = NULL;
   while (ch != 9)
   {
 //choice check
   printf("\n1. Insert key");
   printf("\n2. Delete key");
   printf("\n3. Search key");
   printf("\n4. Inorder transversal ");
   printf("\n5. Preorder transversal ");
   printf("\n6. Postorder transversal ");
   printf("\n7. Minimum value key");
   printf("\n8. Minimum value key");
   printf("\n9. Exit\n");
   printf("Enter your choice : ");
   scanf("%d", &ch);
switch (ch)
{
case 1:
    printf("\nEntre data to insert in binary search tree(0 to end insertion) :\n");
     while (1){ //infinite loop to insert element in binary search tree
        scanf("%d", &num);
        if (num != 0){
          root = insertNode(root, num);
```

```
        }
        else{
          break;
        }
      }
    break;
case 2:
    printf("Entre data to be Deleted : ");
     scanf("%d", &num);
     deleteNode(root, num);      //delete function call
     printf("\n Key Deleted ! \n");
     break;
case 3:
    printf("Entre data to be Searched : ");
    scanf("%d", &num);
    struct node *s = search(root, num); //search function call
    if (s != NULL)
    {
       printf("\nElement %d found !\n", s->key);
    }
    else
    {
       printf("\n!!!Element not found!!!\n");
    }
    break;
case 4:
    printf("In order transversal : ");
    inOrder(root);     //inOrder traversal function call
    break;
```

```
case 5:
   printf("Pre order transversal : ");
   preOrder(root);    //preOrder traversal function call
   break;
case 6:
   printf("Post order transversal : ");
   postOrder(root);       //postOrder traversal function call
   break;
case 7:
      printf("\nMinimum value key : %d\n",minValueNode(root)->key);
      break;
   case 8:
      printf("\nMaximum value key : %d\n",maxValueNode(root)->key);
      break;
   case 9:
      break;
   default:
      printf("\n!!!Wrong Input!!!\n");
      break;
   }
}
  return 0;
}



// Calculate height
int height(struct node *N) {
  if (N == NULL)
    return 0;
```

```
  return N->height;
}
//to compare to integers
int max(int a, int b) {
  return (a > b) ? a : b;
}
// Create a node
struct node *newNode(int key) {
  struct node *node = (struct node *)
    malloc(sizeof(struct node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
// Right rotate
struct node *rightRotate(struct node *y) {
  struct node *x = y->left;
  struct node *T2 = x->right;

  x->right = y;
  y->left = T2;

  y->height = max(height(y->left), height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;

  return x;
}
```

```
// Left rotate
struct node *leftRotate(struct node *x) {
  struct node *y = x->right;
  struct node *T2 = y->left;

  y->left = x;
  x->right = T2;

  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;

  return y;
}
// To find balance factor
int getBalance(struct node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}
// Insert node
struct node *insertNode(struct node *node, int key) {
// Find the correct position to insertNode the node and insertNode it
if (node == NULL)
  return (newNode(key));

if (key < node->key)
  node->left = insertNode(node->left, key);
else if (key > node->key)
  node->right = insertNode(node->right, key);
```

```
else
  return node;

// Update the balance factor of each node and
// Balance the tree
node->height = 1 + max(height(node->left),
          height(node->right));

int balance = getBalance(node);
if (balance > 1 && key < node->left->key)
  return rightRotate(node);

if (balance < -1 && key > node->right->key)
  return leftRotate(node);

if (balance > 1 && key > node->left->key) {
  node->left = leftRotate(node->left);
  return rightRotate(node);
}

if (balance < -1 && key < node->right->key) {
   node->right = rightRotate(node->right);
   return leftRotate(node);
 }

  return node;
}
//Find minimum value key
struct node *minValueNode(struct node *node) {
```

```
  struct node *current = node;
  while (current->left != NULL)
    current = current->left;
  return current;
}
//Find maximum value key
struct node *maxValueNode(struct node *node) {
  struct node *current = node;
  while (current->right != NULL)
    current = current->right;
  return current;
}
// Delete a nodes
struct node *deleteNode(struct node *root, int key) {
  // Find the node and delete it
  if (root == NULL)
    return root;

  if (key < root->key)
    root->left = deleteNode(root->left, key);

else if (key > root->key)
  root->right = deleteNode(root->right, key);

else {
  if ((root->left == NULL) || (root->right == NULL)) {
    struct node *temp = root->left ? root->left : root->right;

    if (temp == NULL) {
```

```
    temp = root;
    root = NULL;
  } else
    *root = *temp;
  free(temp);
} else {
  struct node *temp = minValueNode(root->right);

  root->key = temp->key;

    root->right = deleteNode(root->right, temp->key);
  }
}

if (root == NULL)
  return root;

// Update the balance factor of each node and
// balance the tree
root->height = 1 + max(height(root->left),
            height(root->right));

int balance = getBalance(root);
if (balance > 1 && getBalance(root->left) >= 0)
  return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0) {
  root->left = leftRotate(root->left);
  return rightRotate(root);
```

```
}

if (balance < -1 && getBalance(root->right) <= 0)
  return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {
  root->right = rightRotate(root->right);
  return leftRotate(root);
}

  return root;
}
// Print the tree in preorder traversal
void preOrder(struct node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    preOrder(root->left);
    preOrder(root->right);
  }
}
// Print the tree in inorder traversal
void inOrder(struct node *root) {
  if (root != NULL) {
    inOrder(root->left);
    printf("%d ", root->key);
    inOrder(root->right);
  }
}
// Print the tree in postorder traversal
```

```
void postOrder(struct node *root) {
  if (root != NULL) {
    postOrder(root->left);
    postOrder(root->right);
    printf("%d ", root->key);
  }
}
//search function
struct node *search(struct node *root, int key)
{
 if (root == NULL)
 {
 return NULL;
 }
 if (key == root->key)
 {
 return root;
 }
 else if (key < root->key)
 {
 return search(root->left, key);
 }
 else
 {
 return search(root->right, key);
 }
}
```

```
1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 1

Entre data to insert in binary search tree(0 to end insertion) :
4
7
9
8
2
3
10
0

1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 2
Entre data to be Deleted : 2

 Key Deleted !
```

```
1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 3
Entre data to be Searched : 4

Element 4 found !

1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 4
In order transversal : 3 4 7 8 9 10
1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 5
Pre order transversal : 7 3 4 9 8 10
```

```
1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 6
Post order transversal : 4 3 8 10 9 7
1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 7

Minimum value key : 3

1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 8

Maximum value key : 10

1. Insert key
2. Delete key
3. Search key
4. Inorder transversal
5. Preorder transversal
6. Postorder transversal
7. Minimum value key
8. Minimum value key
9. Exit
Enter your choice : 9
PS C:\Users\Lenovo\Desktop\c_programms>
```