# 9. Implementation of AVL Tree

**Program :**

```c
#include <stdio.h>

#include <stdlib.h>

struct Node {
  int key;
  struct Node * left;
  struct Node * right;
  int height;
};

int max(int a, int b);

int height(struct Node * N) {
  if (N == NULL)
    return 0;
  return N -> height;
}

int max(int a, int b) {
  return (a > b) ? a : b;
}

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);
}

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;
}

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;
}

int getBalance(struct Node * N) {
  if (N == NULL)
    return 0;
  return height(N -> left) - height(N -> right);
}

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

  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;
}

struct Node * minValueNode(struct Node * node) {
  struct Node * current = node;

  while (current -> left != NULL)
    current = current -> left;

  return current;
}

struct Node * deleteNode(struct Node * root, int key) {
```

```
  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;

  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;
}

void printPreOrder(struct Node * root) {
  if (root != NULL) {
    printf("%d ", root -> key);
```

```
    printPreOrder(root -> left);
    printPreOrder(root -> right);
  }
}

int main() {
  int choice, e;
  struct Node * root = NULL;
  do {
    printf("\n1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\n");
    printf("Enter your choice: ");
    scanf("%d", & choice);
    switch (choice) {
    case 1:
      printf("Enter the element to be inserted : ");
      scanf("%d", & e);
      root = insertNode(root, e);
      break;
    case 2:
      printf("Enter the element to be deleted : ");
      scanf("%d", & e);
      root = deleteNode(root, e);
      break;
    case 3:
      printf("Preorder : ");
      printPreOrder(root);
      break;
    case 4:
      break;
    default:
      printf("\nInvalid choice.... Try again!!");
    }
  } while (choice != 4);
  return 0;
}
```

**Output :**

```
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 1
Enter the element to be inserted : 2

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 1
Enter the element to be inserted : 32

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
```

```
Enter your choice: 1
Enter the element to be inserted : 46

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 1
Enter the element to be inserted : 2

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 2
Enter the element to be deleted : 2

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 1
Enter the element to be inserted : 543

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 3
Preorder : 46 32 543
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 1
Enter the element to be inserted : 231

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 1
Enter the element to be inserted : 97

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 3
Preorder : 46 32 231 97 543
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter your choice: 4
```