main.c

Run

```c
#192210211
#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;
}
struct Node *rightRotate(struct Node *y)
```

Output

Clear

```
/tmp/dKzoRx1qjh.o
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11
```

main.c

Run

```
{
    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* insert(struct Node* node, int key)
{
    if (node == NULL)
        return(newNode(key));
```

Output

Clear

```
/tmp/dKzoRx1qjh.o
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11
```

main.c

Run

```
61      if (key < node->key)
62          node->left  = insert(node->left, key);
63      else if (key > node->key)
64          node->right = insert(node->right, key);
65      else
66          return node;
67      node->height = 1 + max(height(node->left),
68                             height(node->right));
69      int balance = getBalance(node);
70      if (balance > 1 && key < node->left->key)
71          return rightRotate(node);
72      if (balance < -1 && key > node->right->key)
73          return leftRotate(node);
74      if (balance > 1 && key > node->left->key)
75      {
76          node->left =  leftRotate(node->left);
77          return rightRotate(node);
78      }
79      if (balance < -1 && key < node->right->key)
80      {
81          node->right = rightRotate(node->right);
82          return leftRotate(node);
83      }
84      return node;
85  }
86  struct Node * minValueNode(struct Node* node)
87  {
88      struct Node* current = node;
89      while (current->left != NULL)
90          current = current->left;
```

Output

Clear

```
/tmp/dKzoRx1qjh.o
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11
```

main.c

Run

```
    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);
```

Output

Clear

```
/tmp/dKzoRx1qjh.o
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11
```

main.c

Run

```
120             root->right = deleteNode(root->right, temp->key);}}
121     if (root == NULL)
122       return root;
123     root->height = 1 + max(height(root->left),
124                            height(root->right));
125     int balance = getBalance(root);
126     if (balance > 1 && getBalance(root->left) >= 0)
127         return rightRotate(root);
128     if (balance > 1 && getBalance(root->left) < 0)
129     {
130         root->left =  leftRotate(root->left);
131         return rightRotate(root);}
132     if (balance < -1 && getBalance(root->right) <= 0)
133         return leftRotate(root);
134     if (balance < -1 && getBalance(root->right) > 0)
135     {
136         root->right = rightRotate(root->right);
137         return leftRotate(root);
138     }
139     return root;
140 }
141 void preOrder(struct Node *root)
142 {
143     if(root != NULL)
144     {
145         printf("%d ", root->key);
146         preOrder(root->left);
147         preOrder(root->right);
148     }
149 }
```

Output

Clear

```
/tmp/dKzoRx1qjh.o
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11
```

main.c

Run

```
}
void preOrder(struct Node *root)
{
    if(root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}
int main()
{
  struct Node *root = NULL;
    root = insert(root, 9);
    root = insert(root, 5);
    root = insert(root, 10);
    root = insert(root, 0);
    root = insert(root, 6);
    root = insert(root, 11);
    root = insert(root, -1);
    root = insert(root, 1);
    root = insert(root, 2);
    printf("Preorder traversal of the constructed AVL "
           "tree is \n");
    preOrder(root);
    root = deleteNode(root, 10);
    printf("\nPreorder traversal after deletion of 10 \n");
    preOrder(root);
    return 0;
}
```

Output

Clear

```
/tmp/dKzoRx1qjh.o
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11
```