>AVL<

```
#include<iostream>
#include<cstdio>
#include<sstream>
#include<algorithm>
#define pow2(n) (1 << (n))
using namespace std;
class avl {
  public:
  int d;
  avl *l;
  avl *r;
}*r;

class avl_tree {
  public:
    int height(avl *);
    int difference(avl *);
    avl *rr_rotat(avl *);
    avl *ll_rotat(avl *);
    avl *lr_rotat(avl*);
    avl *rl_rotat(avl *);
    avl * balance(avl *);
    avl * insert(avl*, int);
    void show(avl*, int);
    void inorder(avl *);
    void preorder(avl *);
    void postorder(avl*);
    avl_tree() {
      r = NULL;
    }
```

```
};

// Calculates the height o current Node
int avl_tree::height(avl *t) {
  int h = 0;
  if (t != NULL) {
    int l_height = height(t->l);
    int r_height = height(t->r);
    int max_height = max(l_height, r_height);
    h = max_height + 1;
  }
  return h;
}

// Calculates the Balance factor at Current Node.
int avl_tree::difference(avl *t) {
  int l_height = height(t->l);
  int r_height = height(t->r);
  int b_factor = l_height - r_height;
  return b_factor;
}

// Left Rotation
avl *avl_tree::rr_rotat(avl *parent) {
  avl *t;
  t = parent->r;
  parent->r = t->l;
  t->l = parent;
  cout<< endl << "\t\t Left Rotation ==> RR case";
  return t;
}
```

```
// Right Rotation
avl *avl_tree::ll_rotat(avl *parent) {
  avl *t;
  t = parent->l;
  parent->l = t->r;
  t->r = parent;
  cout<< endl << "\t\t Right Rotation ==> LL Case";
  return t;
}

// Left Right Rotation (LR)
avl *avl_tree::lr_rotat(avl *parent) {
  avl *t;
  t = parent->l;
  parent->l = rr_rotat(t);
  cout<< endl << "\t\t LR Rotation";
  return ll_rotat(parent);
}

// Right Left Rotation (RL)
avl *avl_tree::rl_rotat(avl *parent) {
  avl *t;
  t = parent->r;
  parent->r = ll_rotat(t);
  cout<< endl << "\t\t RL Rotation";
  return rr_rotat(parent);
}

// Apply Rotations to Balance the Node / Tree
avl *avl_tree::balance(avl *t) {
```

```
  int bal_factor = difference(t);

  if (bal_factor > 1) {   // Height of Left Subtree is more than right subtree.

    if (difference(t->l) > 0)
      t = ll_rotat(t);   // Balance factor of left child of t is +ve == Right Rotation
    else
      t = lr_rotat(t);   // Balance factor of left child of t is -ve == LR Rotation

  } else if (bal_factor < -1) { // Height of Right Subtree more than left subtree

    if (difference(t->r) > 0)
      t = rl_rotat(t);   // Balance factor of right child of t is +ve == RL Rotation
    else
      t = rr_rotat(t);   // Balance actor of right child of t is -ve == Left Rotation
  }
  return t;
}



// Insertation in AVL Tree
avl *avl_tree::insert(avl *r, int v) {
  if (r == NULL) {
    r = new avl;
    r->d = v;
    r->l = NULL;
    r->r = NULL;
    return r;
  } else if (v < r->d) {
    r->l = insert(r->l, v);
```

```
    r = balance(r);
  } else if (v >= r->d) {
    r->r = insert(r->r, v);
    r = balance(r);
  } return r;
}

// Display of AVL Tree
void avl_tree::show(avl *p, int l) {
  int i;
  if (p != NULL) {
    show(p->r, l+ 1);
    cout<<" ";
    if (p == r)
      cout << "Root -> ";
    for (i = 0; i < l&& p != r; i++)
      cout << " ";
      cout << p->d;
      show(p->l, l + 1);
  }
}

// Inorder Traversal
void avl_tree::inorder(avl *t) {
  if (t == NULL)
    return;
    inorder(t->l);
    cout << t->d << " ";
    inorder(t->r);
}
```

```
// Preorder Traversal
void avl_tree::preorder(avl *t) {
  if (t == NULL)
     return;
     cout << t->d << " ";
     preorder(t->l);
     preorder(t->r);
}

// Postorder Traversal
void avl_tree::postorder(avl *t) {
  if (t == NULL)
       return;
   postorder(t ->l);
   postorder(t ->r);
   cout << t->d << " ";
}

// Main
int main() {
  int c, i;
  avl_tree avl;
  while (1) {
     cout << endl << "\t You can perform following operations on AVL Tree :-" << endl;
     cout << endl << "\t\t 1. Construct AVL" << endl;
     cout << "\t\t 2. Insert Element into the tree" << endl;
     cout << "\t\t 3. Show Balanced AVL Tree" << endl;
     cout << "\t\t 4. InOrder Traversal" << endl;
     cout << "\t\t 5. PreOrder Traversal" << endl;
     cout << "\t\t 6. PostOrder Traversal" << endl;
     cout << "\t\t 7. Exit" << endl << endl;
```

```
cout << "\t Enter your Choice: ";
cin >> c;
switch (c) {
  case 1 :
    //Number of nodes to be inserted
    int t;
    cout<<endl<<"\t Enter number of nodes to insert in AVL Tree :- ";
    cin>>t;
    while(t--){
       int d;
       cout<<endl<<"\t\t Enter "<<t<<" Element.....:- ";
       cin>>d;
       r = avl.insert(r,d);
    }
    cout<<endl<<"\t .....TBT Constructed....."<<endl;
    break;
  case 2:
    cout << "\t Enter value to be inserted: ";
    cin >> i;
    r = avl.insert(r, i);
  break;
  case 3:
    if (r == NULL) {
      cout << endl << "\t\t Tree is Empty....." << endl;
      continue;
    }
    cout << "\t Balanced AVL Tree:" << endl;
    avl.show(r, 1);
    cout<<endl;
  break;
  case 4:
```

```
  if (r == NULL) {
    cout << endl << "\t\t Tree is Empty....." << endl;
    continue;
  }
  cout << endl << "\t\t Inorder Traversal:";
  avl.inorder(r);
  cout << endl;
break;
case 5:
  if (r == NULL) {
    cout << endl << "\t\t Tree is Empty....." << endl;
    continue;
  }
  cout << endl << "\t\t Preorder Traversal:";
  avl.preorder(r);
  cout << endl;
break;
case 6:
  if (r == NULL) {
    cout << endl << "\t\t Tree is Empty....." << endl;
    continue;
  }
  cout << endl << "\t\t Postorder Traversal:";
  avl.postorder(r);
  cout << endl;
break;
case 7:
  exit(1);
break;
default:
  cout << endl << "\t\t Wrong Choice....." << endl;
```

```
    }
  }
  return 0;
}
```