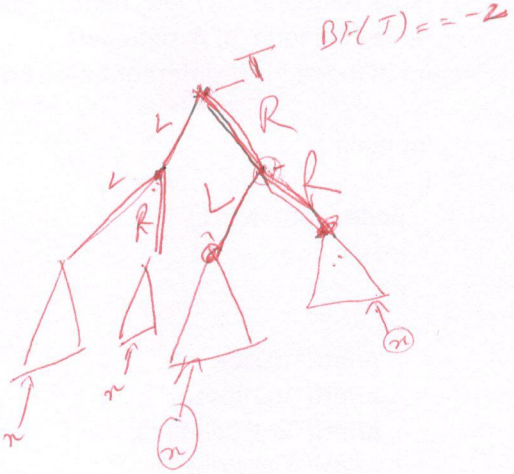
AVL

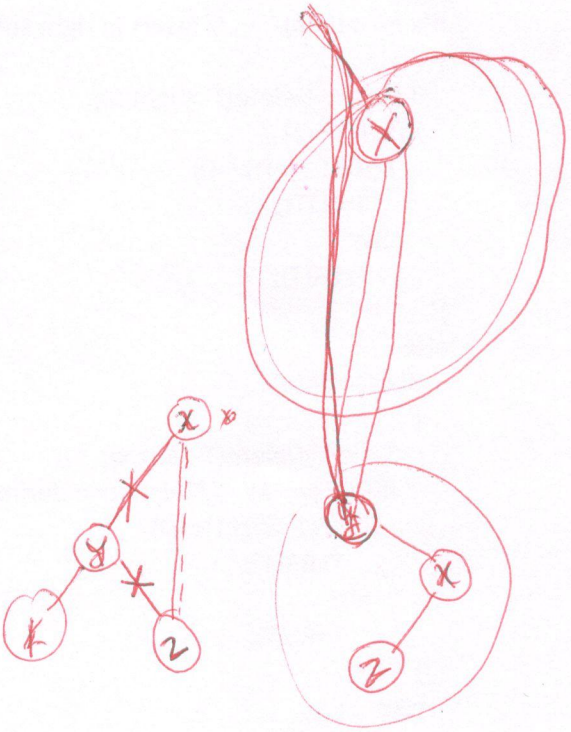
```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
   int data;
   struct node *left,*right;
   int ht;
}node;

node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *); // right_right
node *LL(node *); // left_left
node *LR(node *);// left_right
node *RL(node *); // right_left
int BF(node *); // different between height of left and right child // Balanced Factor

int main()
{
   node *root=NULL;
   int x,n,i,op;
   do
   {
      printf("\n1)Create:");
      printf("\n2)Insert:");
      printf("\n3)Delete:");
      printf("\n4)Print:");
      printf("\n5)Quit:");
      printf("\n\nEnter Your Choice:");
      scanf("%d",&op);

      switch(op)
      {
         case 1: printf("\nEnter no. of elements:");
               scanf("%d",&n);
               printf("\nEnter tree data:");
               root=NULL;
               for(i=0;i<n;i++)
               {
                  scanf("%d",&x);
                  root=insert(root,x);
               }
               break;

         case 2: printf("\nEnter a data:");
               scanf("%d",&x);
               root=insert(root,x);
               break;
```

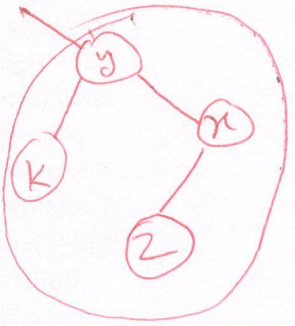
```
        case 3: printf("\nEnter a data:");
                scanf("%d",&x);
                root=Delete(root,x);
                break;

        case 4: printf("\nPreorder sequence:\n");
                preorder(root);
                printf("\n\nInorder sequence:\n");
                inorder(root);
                printf("\n");
                break;
      }
   }while(op!=5);

   return 0;
}

node * insert(node *T,int x)
{
   if(T==NULL)
   {
      T=(node *)malloc(sizeof(node));
      T->data=x;
      T->left=NULL;
      T->right=NULL;
   }
   else
      if(x > T->data)       // insert in right subtree
      {
         T->right=insert(T->right,x);
         if(BF(T)==-2)
            if(x>T->right->data)
               T=RR(T);
            else
               T=RL(T);
      }
      else
         if(x<T->data)
         {
            T->left=insert(T->left,x);
            if(BF(T)==2)
               if(x < T->left->data)
                  T=LL(T);
               else
                  T=LR(T);
         }

      T->ht=height(T);

      return(T);
}
```



```
node * Delete(node *T,int x)
{
   node *p;
   if(T==NULL)
   {
      return NULL;
   }
   else
      if(x > T->data)       // insert in right subtree
      {
         T->right=Delete(T->right,x);
         if(BF(T)==2)
            if(BF(T->left)>=0)
               T=LL(T);
            else
               T=LR(T);
      }
      else
         if(x<T->data)
         {
            T->left=Delete(T->left,x);
            if(BF(T)==-2)   //Rebalance during windup
               if(BF(T->right)<=0)
                  T=RR(T);
               else
                  T=RL(T);
         }
         else
         {
            //data to be deleted is found
            if(T->right!=NULL)
            {   //delete its inorder succesor
               p=T->right;
               while(p->left!= NULL)
                  p=p->left;
               T->data=p->data;
               T->right=Delete(T->right,p->data);
              if(BF(T)==2)//Rebalance during windup
                  if(BF(T->left)>=0)
                     T=LL(T);
                  else
                     T=LR(T);\
            }
            else
               return(T->left);
         }
   T->ht=height(T);
   return(T);
}
```

```
int height(node *T)
{
  int lh,rh;
  if(T==NULL)
    return(0);
  if(T->left==NULL)
    lh=0;
  else
    lh=1+T->left->ht;
  if(T->right==NULL)
    rh=0;
  else
    rh=1+T->right->ht;
  if(lh>rh)
    return(lh);
  return(rh);
}

node * rotateright(node *x)
{
  node *y;
  y=x->left;
  x->left=y->right;
  y->right=x;
  x->ht=height(x);
  y->ht=height(y);
  return(y);
}

node * rotateleft(node *x)
{
  node *y;
  y=x->right;
  x->right=y->left;
  y->left=x;
  x->ht=height(x);
  y->ht=height(y);
  return(y);
}

node * RR(node *T)
{
  T=rotateleft(T);
  return(T);
}

node * LL(node *T)
{
  T=rotateright(T);
  return(T);
}
```





```
node * LR(node *T)
{
   T->left=rotateleft(T->left);
   T=rotateright(T);
   return(T);
}

node * RL(node *T)
{
   T->right=rotateright(T->right);
   T=rotateleft(T);
   return(T);
}

int BF(node *T)
{
   int lh,rh;
   if(T==NULL)
      return(0);
   if(T->left==NULL)
      lh=0;
   else
      lh=1+T->left->ht;
   if(T->right==NULL)
      rh=0;
   else
      rh=1+T->right->ht;
   return(lh-rh);
}

void preorder(node *T)
{
   if(T!=NULL)
   {
      printf("%d(Bf=%d)",T->data,BF(T));
      preorder(T->left);
      preorder(T->right);
   }
}

void inorder(node *T)
{
   if(T!=NULL)
   {
      inorder(T->left);
      printf("%d(Bf=%d)",T->data,BF(T));
      inorder(T->right);
   }
}
```