void BinarySearchTree::remove(int d)

{

//Locate the element

bool found = false;

if(isEmpty())

{

cout<<" This Tree is empty! "<<endl;

return;

}

tree\_node\* curr;

tree\_node\* parent;

curr = root;

while(curr != NULL)

{

if(curr->data == d)

{

found = true;

break;

}

else

{

parent = curr;

if(d>curr->data) curr = curr->right;

else curr = curr->left;

}

}

if(!found)

{

cout<<" Data not found! "<<endl;

return;

}

// 3 cases :

// 1. We're removing a leaf node

// 2. We're removing a node with a single child

// 3. we're removing a node with 2 children

// Node with single child

if((curr->left == NULL && curr->right != NULL)|| (curr->left != NULL

&& curr->right == NULL))

{

if(curr->left == NULL && curr->right != NULL)

{

if(parent->left == curr)

{

parent->left = curr->right;

delete curr;

}

else

{

parent->right = curr->right;

delete curr;

}

}

else // left child present, no right child

{

if(parent->left == curr)

{

parent->left = curr->left;

delete curr;

}

else

{

parent->right = curr->left;

delete curr;

}

}

return;

}

//We're looking at a leaf node

if( curr->left == NULL && curr->right == NULL)

{

if(parent->left == curr) parent->left = NULL;

else parent->right = NULL;

delete curr;

return;

}

//Node with 2 children

// replace node with smallest value in right subtree

if (curr->left != NULL && curr->right != NULL)

{

tree\_node\* chkr;

chkr = curr->right;

if((chkr->left == NULL) && (chkr->right == NULL))

{

curr = chkr;

delete chkr;

curr->right = NULL;

}

else // right child has children

{

//if the node's right child has a left child

// Move all the way down left to locate smallest element

if((curr->right)->left != NULL)

{

tree\_node\* lcurr;

tree\_node\* lcurrp;

lcurrp = curr->right;

lcurr = (curr->right)->left;

while(lcurr->left != NULL)

{

lcurrp = lcurr;

lcurr = lcurr->left;

}

curr->data = lcurr->data;

delete lcurr;

lcurrp->left = NULL;

}

else

{

tree\_node\* tmp;

tmp = curr->right;

curr->data = tmp->data;

curr->right = tmp->right;

delete tmp;

}

}

return;

}

}

//////

void BinarySearchTree::insert(int d)

{

tree\_node\* t = new tree\_node;

tree\_node\* parent;

t->data = d;

t->left = NULL;

t->right = NULL;

parent = NULL;

// is this a new tree?

if(isEmpty()) root = t;

else

{

//Note: ALL insertions are as leaf nodes

tree\_node\* curr;

curr = root;

// Find the Node's parent

while(curr)

{

parent = curr;

if(t->data > curr->data) curr = curr->right;

else curr = curr->left;

}

if(t->data < parent->data)

parent->left = t;

else

parent->right = t;

}

}

x

/ \

A Y

/\

B C