**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PROGRAMMING LABORATORY (CSE 351)**

**ASSIGNMENT 5**

**Asmit De**

**10/CSE/53**

**Date: 22.09.2011**

**Program 1: Binary Search Tree**

*Source Code –*

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

typedef struct node

{

struct node \*left, \*right;

int data;

}NODE;

NODE \*getNode(int data)

{

NODE \*node = (NODE\*)malloc(sizeof(NODE));

node->data = data;

node->left = node->right = NULL;

return node;

}

void preOrder(NODE \*tree)

{

if(tree == NULL)

return;

printf("%d ", tree->data);

preOrder(tree->left);

preOrder(tree->right);

}

void inOrder(NODE \*tree)

{

if(tree == NULL)

return;

inOrder(tree->left);

printf("%d ", tree->data);

inOrder(tree->right);

}

void postOrder(NODE \*tree)

{

if(tree == NULL)

return;

postOrder(tree->left);

postOrder(tree->right);

printf("%d ", tree->data);

}

NODE \*insertNode(NODE \*tree, int data)

{

if(tree == NULL)

{

tree = getNode(data);

return tree;

}

if(data < tree->data)

tree->left = insertNode(tree->left, data);

else

tree->right = insertNode(tree->right, data);

return tree;

}

int sizeOfTree(NODE \*tree)

{

if(tree == NULL)

return 0;

return sizeOfTree(tree->left) + sizeOfTree(tree->right) + 1;

}

int maxPathWt(NODE \*tree)

{

int lWt, rWt;

if(tree == NULL)

return 0;

lWt = maxPathWt(tree->left);

rWt = maxPathWt(tree->right);

return (lWt > rWt ? lWt : rWt) + tree->data;

}

int heightOfTree(NODE \*tree)

{

int lSz, rSz;

if(tree == NULL)

return 0;

lSz = heightOfTree(tree->left);

rSz = heightOfTree(tree->right);

return (lSz > rSz ? lSz : rSz) + 1;

}

void generateMirror(NODE \*tree)

{

NODE \*temp = NULL;

if(tree != NULL)

{

generateMirror(tree->left);

generateMirror(tree->right);

temp = tree->left;

tree->left = tree->right;

tree->right = temp;

}

}

int compareTree(NODE \*tree1, NODE \*tree2)

{

if(tree1 == NULL && tree2 == NULL)

return 0;

else if((tree1 == NULL && tree2 != NULL) || (tree1 != NULL && tree2 == NULL))

return 1;

else if(tree1->data != tree2->data)

return 1;

else

{

if(compareTree(tree1->left, tree2->left))

return 1;

else if(compareTree(tree1->right, tree2->right))

return 1;

else

return 0;

}

}

int main()

{

char c, ch;

NODE \*tree1 = NULL, \*tree2 = NULL;

int data;

while(1)

{

system("cls");

puts("MENU");

puts("1\tDisplay by pre-order traversal");

puts("2\tDisplay by in-order traversal");

puts("3\tDisplay by post-order traversal");

puts("4\tInsert node");

puts("5\tFind size of tree");

puts("6\tFind height of tree");

puts("7\tFind maximum path value of tree");

puts("8\tCreate a mirror of tree");

puts("9\tCompare two trees");

puts("x\tExit");

printf("\nEnter your choice...");

c = getch();

fflush(stdin);

switch(c)

{

case '1':

system("cls");

printf("Pre-order Traversal:\n\n");

preOrder(tree1);

printf("\nPress any key to return to menu...");

getch();

break;

case '2':

system("cls");

printf("In-order Traversal:\n\n");

inOrder(tree1);

printf("\nPress any key to return to menu...");

getch();

break;

case '3':

system("cls");

printf("Post-order Traversal:\n\n");

postOrder(tree1);

printf("\nPress any key to return to menu...");

getch();

break;

case '4':

system("cls");

printf("Insert Node:\n\n");

printf("Enter data: ");

scanf("%d", &data);

tree1 = insertNode(tree1, data);

printf("\nData entered successfully...\nPress any key to return to menu...");

getch();

break;

case '5':

system("cls");

printf("Size of Tree:\n\n");

printf("Size: %d", sizeOfTree(tree1));

printf("\nPress any key to return to menu...");

getch();

break;

case '6':

system("cls");

printf("Height of Tree:\n\n");

printf("Height: %d", heightOfTree(tree1));

printf("\nPress any key to return to menu...");

getch();

break;

case '7':

system("cls");

printf("Maximum Path Value of Tree:\n\n");

printf("Maximum path weight: %d", maxPathWt(tree1));

printf("\nPress any key to return to menu...");

getch();

break;

case '8':

system("cls");

printf("Mirror of Tree:\n\n");

printf("Tree:\t");

inOrder(tree1);

generateMirror(tree1);

printf("\nMirror:\t");

inOrder(tree1);

printf("\nPress any key to return to menu...");

getch();

break;

case '9':

system("cls");

printf("Tree Comparison:\n\n");

printf("Tree1:\t");

inOrder(tree1);

printf("\nCreate Tree2:\n");

tree2 = NULL;

do

{

printf("Enter data: ");

scanf("%d", &data);

tree2 = insertNode(tree2, data);

puts("Press c to continue entering data, any other key to stop: ");

ch = getch();

fflush(stdin);

}while(ch == 'c' || ch == 'C');

printf("\nTree2:\t");

inOrder(tree2);

if(!compareTree(tree1, tree2))

printf("\nTrees are same");

else

printf("\nTrees are different");

printf("\nPress any key to return to menu...");

getch();

break;

case 'x':

case 'X':

exit(0);

default:

system("cls");

printf("\aError: Invalid input...\nPress any key to return to menu...");

getch();

}

}

}