#include <iostream>

#include <stack>

using namespace std;

class Node{

public:

int data;

Node\* left, \*right; // Left Pointer and Right Pointer

bool lthread, rthread; // Left Thread and Right Thread

int flag; // Flag

};

class TBT{

public:

Node\* root = NULL; // Root Node

Node\* head; // Dummy Node

TBT();

Node\* getNode(int val); // Function to create new node

void createRoot(int data); // Function to create Root node

void insertBST(int data, Node\* root); // Function to Insert Node as per BST implementation

void insert(int data); // Function to Insert Node as per user

void inorder(bool resetFlag); // Function to perform In-Order Traversal

void preorder(); // Function to perform Pre-Order Traversal

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONSTRUCTOR \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

TBT::TBT(){

root = NULL; // Initialize the root to NULL

head = new Node; // Creating the dummy node

head->right = head->left = head; // Dummy nodes pointing to itself

head->data = -1;

head->lthread = 0; // Left thread is initially set to 0

head->rthread = 1; // Right thread is initially set to 1

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CREATE NEW NODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Node\* TBT::getNode(int val){

Node\* nn = new Node; // New Node Created

nn->data = val; // Assigned Value

nn->left = nn->right = NULL; // Left and Right pointer initialized as NULL

nn->lthread = nn->rthread= true; // Left and Right thread initialized as true

return nn;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CREATE ROOT NODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void TBT::createRoot(int data){

root = getNode(data);

head->left = root; // Left pointer of Dummy points to root node

head->lthread = 1; // Left thread initialized as 1

root->left = root->right = head; // Root points to head

root->lthread = root->rthread= 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INSERT AS PER BST \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void TBT::insertBST(int data, Node\* root){

if(head == NULL){

cout<<"Tree is empty. Create Root first."<<endl;

return;

}

Node\* current = root;

Node\* parent = NULL;

while(current != NULL){

parent = current;

if(data < current->data){

if(current->lthread){

Node\* newNode = getNode(data);

newNode->left = current->left;

newNode->right = current;

newNode->lthread = true;

newNode->rthread = true;

current->left = newNode;

current->lthread = false;

break;

}

else{

current = current->left;

}

}

else if(data > current->data){

if(current->rthread){

Node\* newNode = getNode(data);

newNode->right = current->right;

newNode->left = current;

newNode->lthread = true;

newNode->rthread = true;

current->right = newNode;

current->rthread = false;

break;

}

else{

current = current->right;

}

}

else{

cout<<"Duplicate value, not inserted"<<endl;

return;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INSERT AS PER USER CHOICE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void TBT::insert(int data){

if(head->left == head){ // Checking if the tree is empty,and creating the root node

cout<<"Creating root "<<endl;

this->createRoot(data);

cout<<"Root with data "<<data<<" has been created"<<endl;

return;

}

Node\* newNode = new Node;

newNode->data = data;

newNode->lthread = newNode->rthread = 0;

Node\* temp = root;

bool flag = true;

while(flag){

int ch;

cout<<"\nRoot is : "<<temp->data<<endl;

cout<<"1. Insert at left\n2. Insert at right"<<endl;

cout<<"Enter Choice code"<<endl;

cin>>ch;

if(ch == 1){ // If the choice is to insert at the left

if(temp->lthread == 1){ // If left thread is active, move to the left child

temp = temp->left;

}

else{

newNode->left = temp->left; // Insert the new node to the left of the current node

temp->left = newNode;

newNode->right = temp;

temp->lthread = 1; // Activate left thread of the current node

flag = false; // Exit the loop

}

}

else if(ch == 2){ // If the choice is to insert at the right

if(temp->rthread == 1){ // If right thread is active, move to the right child

temp = temp->right;

}

else{

newNode->right = temp->right; // Insert the new node to the right of the current node

temp->right = newNode;

newNode->left = temp;

temp->rthread = 1; // Activate right thread of the current node

flag = false; // Exit the loop

}

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IN-ORDER TRAVERSAL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void TBT::inorder(bool resetFlag){

Node\* temp = head->left;

if(temp == head){ // Check if the tree is empty

cout<<"\nTree Empty!"<<endl;

return;

}

while(temp != head){

while(temp->lthread != 0){ // Move to the leftmost node of the current subtree

temp = temp->left;

}

while(temp->rthread != 1){ // Process nodes along the threaded right links

if(!resetFlag){

cout<<temp->data<<" "; // If not resetting flags, print the data; otherwise, reset the flag

}

else{

temp->flag = 0;

}

temp = temp->right; // Move to the next node along the threaded right link

}

if(temp == head){ // If the current subtree is exhausted, exit the loop

break;

}

if(!resetFlag){ // If not resetting flags, print the data; otherwise, reset the flag

cout<<temp->data<<" ";

}

else{

temp->flag = 0;

}

temp = temp->right; // Move to the next node along the threaded right link

}

if(!resetFlag){

cout<<endl;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PRE-ORDER TRAVERSAL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void TBT::preorder(){

Node\* temp = head->left;

if(temp == head){ // Check if the tree is empty

cout<<"\nTree Empty!"<<endl;

return;

}

while(temp != head){ // Process nodes along the threaded left links until a leaf is reached

while(true){

cout<<temp->data<<" ";

if(temp->lthread == 0){ // If the current node has a left child, move to the left child

break;

}

else{

temp = temp->left;

}

}

while(temp->rthread != 1){ // Move to the next node along the threaded right link

temp = temp->right;

}

temp = temp->right; // Move to the next node along the threaded right link

}

cout<<endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN FUNCTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main(){

TBT tbt;

int rootData;

cout<<"Enter root data: ";

cin>>rootData;

tbt.createRoot(rootData);

int data;

while(true){

cout<<"Enter data(enter -1 to exit): ";

cin>>data;

if(data == -1)

break;

tbt.insert(data);

cout<<"Inorder: ";

tbt.inorder(false);

cout<<"PreOrder: ";

tbt.preorder();

cout<<endl;

}

return 0;

}