C program to implement a binary tree using linked list and develop functions to perform traversal, searching, insertion and deletion operations.

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

//Represent a node of binary tree

struct node{

int data;

struct node \*left;

struct node \*right;

};

//Represent the root of binary tree

struct node \*root = NULL;

//createNode() will create a new node

struct node\* createNode(int data){

//Create a new node

struct node \*newNode = (struct node\*)malloc(sizeof(struct node));

//Assign data to newNode, set left and right child to NULL

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

//Represent a queue

struct queue

{

int front, rear, size;

struct node\* \*arr;

};

//createQueue() will create a queue

struct queue\* createQueue()

{

struct queue\* newQueue = (struct queue\*) malloc(sizeof( struct queue ));

newQueue->front = -1;

newQueue->rear = 0;

newQueue->size = 0;

newQueue->arr = (struct node\*\*) malloc(100 \* sizeof( struct node\* ));

return newQueue;

}

//Adds a node to queue

void enqueue(struct queue\* queue, struct node \*temp){

queue->arr[queue->rear++] = temp;

queue->size++;

}

//Deletes a node from queue

struct node \*dequeue(struct queue\* queue){

queue->size--;

return queue->arr[++queue->front];

}

//insertNode() will add new node to the binary tree

void insertNode(int data) {

//Create a new node

struct node \*newNode = createNode(data);

//Check whether tree is empty

if(root == NULL){

root = newNode;

return;

}

else {

//Queue will be used to keep track of nodes of tree level-wise

struct queue\* queue = createQueue();

//Add root to the queue

enqueue(queue, root);

while(true) {

struct node \*node = dequeue(queue);

//If node has both left and right child, add both the child to queue

if(node->left != NULL && node->right != NULL) {

enqueue(queue, node->left);

enqueue(queue, node->right);

}

else {

//If node has no left child, make newNode as left child

if(node->left == NULL) {

node->left = newNode;

enqueue(queue, node->left);

}

//If node has left child but no right child, make newNode as right child

else {

node->right = newNode;

enqueue(queue, node->right);

}

break;

}

}

}

}

//inorder() will perform inorder traversal on binary search tree

void inorderTraversal(struct node \*node) {

//Check whether tree is empty

if(root == NULL){

printf("Tree is empty\n");

return;

}

else {

if(node->left != NULL)

inorderTraversal(node->left);

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

if(node->right != NULL)

inorderTraversal(node->right);

}

}

int main(){

//Add nodes to the binary tree

insertNode(1);

//1 will become root node of the tree

printf("Binary tree after insertion: \n");

//Binary after inserting nodes

inorderTraversal(root);

insertNode(2);

insertNode(3);

//2 will become left child and 3 will become right child of root node 1

printf("\nBinary tree after insertion: \n");

//Binary after inserting nodes

inorderTraversal(root);

insertNode(4);

insertNode(5);

//4 will become left child and 5 will become right child of node 2

printf("\nBinary tree after insertion: \n");

//Binary after inserting nodes

inorderTraversal(root);

insertNode(6);

insertNode(7);

//6 will become left child and 7 will become right child of node 3

printf("\nBinary tree after insertion: \n");

//Binary after inserting nodes

inorderTraversal(root);

return 0;

}