WEEK-8: Test : @ 22.04.2024 / 27.04.2024

Q: Write a C function to implement Linear QUEUE using Linked List .

**TURBO C++ Practice Programs**

//1. Binary Tree Using Arrays

#include<stdio.h>

#include<conio.h>

struct tree

{

struct tree \*left;

char data;

struct tree \*right;

};

typedef struct tree node;

node \*root=NULL;

node \*insert(char a[],int index)

{

node \*temp=NULL;

if(a[index]!='\0')

{

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

temp->left=insert(a,2\*index+1);

temp->data=a[index];

temp->right=insert(a,2\*index+2);

}

return temp;

}

void buildtree(char a[])

{

root=insert(a,0);

}

void inorder(node \*root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%c\t",root->data);

inorder(root->right);

}

}

void preorder(node \*root)

{

if(root!=NULL)

{

printf("%c\t",root->data);

preorder(root->left);

preorder(root->right);

}

}

void postorder(node \*root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%c\t",root->data);

}

}

void main()

{

char a[]={'a','b','c','d','e','f','g','\0','\0','\0','\0','\0','\0','\0','\0','\0'};

clrscr();

buildtree(a);

printf("\n preorder traversal is\n");

preorder(root);

printf("\n inorder traversal is \n");

inorder(root);

printf("\n postorder traversal is \n");

postorder(root);

getch();

}

// 2. Binary Tree using LL with and without recursion traversals.

#include<stdio.h>

#include<alloc.h>

struct tree

{

char data;

struct tree \*left;

struct tree \*right;

};

typedef struct tree btree;

btree \*root,\*stack[100];

int top=0;

void rec();

void traversal();

void recpostorder();

void recinorder();

void recpreorder();

void push(btree \*x)

{

stack[++top]=x;

}

btree \*pop()

{

return(stack[top--]);

}

void create()

{

btree \*p,\*temp;

char ans;

stack[top]=NULL;

root=(btree \*)malloc(sizeof(btree));

printf("enter data for root node");

scanf(" %c",&root->data);

p=root;

while(p)

{

printf("do u want to create a right child for node %c(y/n):",p->data);

scanf(" %c",&ans);

if(toupper(ans)=='N')

p->right=NULL;

else

{

temp=(btree \*)malloc(sizeof(btree));

printf("enter data for the node");

scanf(" %c",&temp->data);

p->right=temp;

printf("data pushing in to stk %c",temp->data);

push(temp);

}

printf("do u want to create a left child for the node %c(y/n):",p->data);

scanf(" %c",&ans);

if(toupper(ans)=='N')

{

p->left=NULL;

p=pop();

printf("data popped from stk %c",p->data);

}

else

{

temp=(btree \*)malloc(sizeof(btree));

printf("enter data for the node");

scanf(" %c",&temp->data);

p->left=temp;

p=p->left;

}

}

}

int menu()

{

int c;

printf("MAIN MENU\n");

printf("1:create btree\n 2: traversal \n 3:exit\n enter ur choice\n");

scanf("%d",&c);

return( c);

}

void recinorder(btree \*r)

{

if(r!=NULL)

{

recinorder(r->left);

printf(" %c",r->data);

recinorder(r->right);

}

}

void recpreorder(btree \*r)

{

if(r!=NULL)

{

printf("%c",r->data);

recpreorder(r->left);

recpreorder(r->right);

}

}

void recpostorder(btree \*r)

{

if(r!=NULL)

{

recpostorder(r->left);

recpostorder(r->right);

printf("%c",r->data);

}

}

void main()

{

int ch,n;

top=0;

clrscr();

while(1)

{

ch=menu();

switch(ch)

{

case 1: create(); break;

case 2: traversal(); break;

case 3: exit(0);

}

}

}

void traversal()

{

int x;

printf("1: with recursion 2. Exit \n );

scanf("%d",&x);

switch(x)

{

case 1: rec(); break;

case 2: return;

}

}

void rec()

{

while(1)

{

int x;

printf("1: inorder 2: preorder 3: postorder\n enter ur choice");

scanf("%d",&x);

switch(x)

{

case 1: printf("rec inorder is\n");recinorder(root); break;

case 2: printf("rec preorder is\n:");recpreorder(root); break;

case 3: printf("rec postorder is\n");recpostorder(root); break;

case 4: return;

}

}

}

// 3. Binary Search Tree Operations

#include<stdio.h>

#include<conio.h>

struct tree

{

struct tree \*left;

int data;

struct tree \*right;

};

typedef struct tree node;

node \*root=NULL;

void insert()

{

node \*p,\*q,\*r;

q=root;

if(q==NULL)

{

printf("empty tree\n");

return;

}

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

printf("enter the node data\n");

scanf("%d",&p->data);

p->left=NULL;

p->right=NULL;

while(q!=NULL)

{

r=q;

if(q->data>p->data)

q=q->left;

else

q=q->right;

}

if(r->data>p->data)

r->left=p;

else

r->right=p;

}

void create()

{

node \*p;

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

printf("enter the root node data\n");

scanf("%d",&p->data);

p->left=NULL;

p->right=NULL;

root=p;

}

void inorder(node \*p)

{

if(p!=NULL)

{

inorder(p->left);

printf("%d\t",p->data);

inorder(p->right);

}

}

void preorder(node \*p)

{

if(p!=NULL)

{

printf("%d\t",p->data);

preorder(p->left);

preorder(p->right);

}

}

void postorder(node \*p)

{

if(p!=NULL)

{

postorder(p->left);

postorder(p->right);

printf("%d\t",p->data);

}

}

void main()

{

char ch='y';

int choice;

clrscr();

do

{

printf("\n1.create\n2.insert\n3.preorder\n4.inorder\n5.postorder\n");

printf("enter ur choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:create();

break;

case 2:insert();

break;

case 3:printf("\n preorder traversal is\n");

preorder(root);

break;

case 4:printf("\n inorder traversal is \n");

inorder(root);

break;

case 5:printf("\n postorder traversal is \n");

postorder(root);

break;

default:printf("\nenter proper choice\n");

}

printf("\ndo u want to continue(y/n)\n");

fflush(stdin);

ch=getchar();

}while(ch=='y');

getch();

}

// 4. C program to insert a node in AVL tree

#include<stdio.h>

#include<stdlib.h>

// An AVL tree node

struct Node

{

int key;

struct Node \*left;

struct Node \*right;

int height;

};

void preOrder(struct Node\*);

int maxi(int,int);

struct Node\* insert(struct Node \*,int);

/\* Driver program to test above function\*/

int main()

{

struct Node \*root = NULL;

clrscr();

/\* Constructing tree given in the above figure \*/

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 25);

/\* The constructed AVL Tree would be

30

/ \

20 40

/ \ \

10 25 50

\*/

printf("Preorder traversal of the constructed AVL"

" tree is \n");

preOrder(root);

return 0;

}

// A utility function to get the height of the tree

int height(struct Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

// A utility function to get maximum of two integers

int maxi(int a, int b)

{

return (a > b)? a : b;

}

/\* Helper function that allocates a new node with the given key and

NULL left and right pointers. \*/

struct Node\* newNode(int key)

{

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

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1; // new node is initially added at leaf

return(node);

}

// A utility functiond to right rotate subtree rooted with y

// See the diagram given above.

struct Node \*rightRotate(struct Node \*y)

{

struct Node \*x = y->left;

struct Node \*T2 = x->right;

// Perform rotation

x->right = y;

y->left = T2;

// Update heights

y->height = maxi(height(y->left), height(y->right))+1;

x->height = maxi(height(x->left), height(x->right))+1;

// Return new root

return x;

}

// A utility function to left rotate subtree rooted with x

// See the diagram given above.

struct Node \*leftRotate(struct Node \*x)

{

struct Node \*y = x->right;

struct Node \*T2 = y->left;

// Perform rotation

y->left = x;

x->right = T2;

// Update heights

x->height = max(height(x->left), height(x->right))+1;

y->height = max(height(y->left), height(y->right))+1;

// Return new root

return y;

}

// Get Balance factor of node N

int getBalance(struct Node \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

// Recursive function to insert a key in the subtree rooted

// with node and returns the new root of the subtree.

struct Node\* insert(struct Node\* node, int key)

{

int balance;

/\* 1. Perform the normal BST insertion \*/

if (node == NULL)

return(newNode(key));

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

else // Equal keys are not allowed in BST

return node;

/\* 2. Update height of this ancestor node \*/

node->height = 1 + max(height(node->left),height(node->right));

/\* 3. Get the balance factor of this ancestor

node to check whether this node became

unbalanced \*/

balance = getBalance(node);

// If this node becomes unbalanced, then

// there are 4 cases

// Left Left Case

if (balance > 1 && key < node->left->key)

return rightRotate(node);

// Right Right Case

if (balance < -1 && key > node->right->key)

return leftRotate(node);

// Left Right Case

if (balance > 1 && key > node->left->key)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

// Right Left Case

if (balance < -1 && key < node->right->key)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

/\* return the (unchanged) node pointer \*/

return node;

}

// A utility function to print preorder traversal of the tree.

// The function also prints height of every node

void preOrder(struct Node \*root)

{

if(root != NULL)

{

printf("%d ", root->key);

preOrder(root->left);

preOrder(root->right);

}

}

// 5.Searching a key on a B-tree in C

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

#define MIN 2

struct BTreeNode {

int val[MAX + 1], count;

struct BTreeNode \*link[MAX + 1];

};

struct BTreeNode \*root;

// Create a node

struct BTreeNode \*createNode(int val, struct BTreeNode \*child) {

struct BTreeNode \*newNode;

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

newNode->val[1] = val;

newNode->count = 1;

newNode->link[0] = root;

newNode->link[1] = child;

return newNode;

}

// Insert node

void insertNode(int val, int pos, struct BTreeNode \*node,

struct BTreeNode \*child) {

int j = node->count;

while (j > pos) {

node->val[j + 1] = node->val[j];

node->link[j + 1] = node->link[j];

j--;

}

node->val[j + 1] = val;

node->link[j + 1] = child;

node->count++;

}

// Split node

void splitNode(int val, int \*pval, int pos, struct BTreeNode \*node,

struct BTreeNode \*child, struct BTreeNode \*\*newNode) {

int median, j;

if (pos > MIN)

median = MIN + 1;

else

median = MIN;

\*ndewNode = (struct BTreeNode \*)malloc(sizeof(struct BTreeNode));

j = median + 1;

while (j <= MAX) {

(\*newNode)->val[j - median] = node->val[j];

(\*newNode)->link[j - median] = node->link[j];

j++;

}

node->count = median;

(\*newNode)->count = MAX - median;

if (pos <= MIN) {

insertNode(val, pos, node, child);

} else {

insertNode(val, pos - median, \*newNode, child);

}

\*pval = node->val[node->count];

(\*newNode)->link[0] = node->link[node->count];

node->count--;

}

// Set the value

int setValue(int val, int \*pval,

struct BTreeNode \*node, struct BTreeNode \*\*child) {

int pos;

if (!node) {

\*pval = val;

\*child = NULL;

return 1;

}

if (val < node->val[1]) {

pos = 0;

} else {

for (pos = node->count;

(val < node->val[pos] && pos > 1); pos--)

;

if (val == node->val[pos]) {

printf("Duplicates are not permitted\n");

return 0;

}

}

if (setValue(val, pval, node->link[pos], child)) {

if (node->count < MAX) {

insertNode(\*pval, pos, node, \*child);

} else {

splitNode(\*pval, pval, pos, node, \*child, child);

return 1;

}

}

return 0;

}

// Insert the value

void insert(int val) {

int flag, i;

struct BTreeNode \*child;

flag = setValue(val, &i, root, &child);

if (flag)

root = createNode(i, child);

}

// Search node

void search(int val, int \*pos, struct BTreeNode \*myNode) {

if (!myNode) {

return;

}

if (val < myNode->val[1]) {

\*pos = 0;

} else {

for (\*pos = myNode->count;

(val < myNode->val[\*pos] && \*pos > 1); (\*pos)--)

;

if (val == myNode->val[\*pos]) {

printf("%d is found", val);

return;

}

}

search(val, pos, myNode->link[\*pos]);

return;

}

// Traverse then nodes

void traversal(struct BTreeNode \*myNode) {

int i;

if (myNode) {

for (i = 0; i < myNode->count; i++) {

traversal(myNode->link[i]);

printf("%d ", myNode->val[i + 1]);

}

traversal(myNode->link[i]);

}

}

void main() {

int val, ch;

insert(8);

insert(9);

insert(10);

insert(11);

insert(15);

insert(16);

insert(17);

insert(18);

insert(20);

insert(23);

traversal(root);

printf("\n");

search(11, &ch, root);

}

**EXAMLY Practice Programs:**

**DATA STRUCTURES\_CODING\_WEEK 8**

No. of Questions: 3 Total Duration: 180 min

Q1. **Problem Statement**

﻿Write a program to create a Binary Search Tree (BST) and perform the following traversals.

inorder

preorder

postorder

**Input Format**

The first input consists of the choice.

If the choice is 1, enter the number of elements N and the elements to be inserted into the tree.

If the choice is 2, print the in-order traversal.

If the choice is 3, print the pre-order traversal.

If the choice is 4, print the post-order traversal.

If the choice is 5, exit.

If the choice is greater than 5, print "Wrong choice".

**Output Format**

The output prints the results based on the choice.

For choice 1, print "BST with N nodes is ready to use" where N is the number of nodes inserted.

For choice 2, print the in-order traversal of the BST.

For choice 3, print the pre-order traversal of the BST.

For choice 4, print the post-order traversal of the BST.

For choice 5, the program exits.

If the choice is greater than 5, print "Wrong choice".

**Refer to the sample output for the formatting specifications.**

**Constraints**

In this scenario, the given test cases will fall under the following constraints:

1 ≤ N ≤ 15.

**Sample Input Sample Output**

1

5

12 78 96 34 55

2

3

4

5

BST with 5 nodes is ready to use

BST Traversal in INORDER

12 34 55 78 96

BST Traversal in PREORDER

12 78 34 55 96

BST Traversal in POSTORDER

55 34 96 78 12

Q1**Test Solution**

#include <stdio.h>

#include<stdlib.h>

struct tnode

{

int data;

struct tnode \*right;

struct tnode \*left;

};

struct tnode \*CreateBST(struct tnode \*, int);

void Inorder(struct tnode \*);

void Preorder(struct tnode \*);

void Postorder(struct tnode \*);

int main()

{

struct tnode \*root = NULL;

int choice, item, n, i;

do

{

scanf("%d",&choice);

switch(choice)

{

case 1:

root = NULL;

scanf("%d",&n);

for(i = 1; i <= n; i++)

{

scanf("%d",&item);

root = CreateBST(root,item);

}

printf("BST with %d nodes is ready to use\n", n);

break;

case 2:

printf("BST Traversal in INORDER\n");

Inorder(root);

printf("\n");

break;

case 3:

printf("BST Traversal in PREORDER\n");

Preorder(root);

printf("\n");

break;

case 4:

printf("BST Traversal in POSTORDER\n");

Postorder(root);

printf("\n");

break;

case 5:

exit(0);

break;

default:

printf("Wrong choice\n");

break;

}

} while(1);

return 0;

}

struct tnode \*CreateBST(struct tnode \*root, int item)

{

if(root == NULL)

{

root = (struct tnode \*)malloc(sizeof(struct tnode));

root->left = root->right = NULL;

root->data = item;

return root;

}

else

{

if(item < root->data )

root->left = CreateBST(root->left,item);

else if(item > root->data )

root->right = CreateBST(root->right,item);

return(root);

}

}

void Inorder(struct tnode \*root)

{

if( root != NULL)

{

Inorder(root->left);

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

Inorder(root->right);

}

}

void Preorder(struct tnode \*root)

{

if( root != NULL)

{

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

Preorder(root->left);

Preorder(root->right);

}

}

void Postorder(struct tnode \*root)

{

if( root != NULL)

{

Postorder(root->left);

Postorder(root->right);

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

}

}

Q2.**Problem Statement**

Write a program that constructs a binary search tree (BST) using a given set of integer values, deletes a specified key value, and prints the in-order traversal of the BST both before and after the deletion.

**Input Format**

The first line consists of the input number of nodes, n.

The second line consists of the space-separated input node values.

The third line consists of the node value to be deleted.

**Output Format**

The output prints the node values in the in-order traversal before and after the deletion of the key value.

If the key value is not present in the tree, the output prints the in-order traversal of the original tree as it is.

**Refer to the sample output for the formatting specifications.**

**Constraints**

The given test cases will fall under the following constraints.

1 ≤ n ≤ 50.

0≤key values≤102.

1≤delKey≤102.

**Sample Input Sample Output**

5

10 30 5 6 7

30

Before deletion of key value: 5 6 7 10 30

After deleting key value 30: 5 6 7 10

Q2**Test Solution**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void inorder(struct Node\* root) {

if (root == NULL) {

return;

}

inorder(root->left);

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

inorder(root->right);

}

struct Node\* getMinimumKey(struct Node\* curr) {

while (curr->left != NULL) {

curr = curr->left;

}

return curr;

}

struct Node\* insert(struct Node\* root, int key) {

if (root == NULL) {

return createNode(key);

}

if (key < root->data) {

root->left = insert(root->left, key);

} else {

root->right = insert(root->right, key);

}

return root;

}

struct Node\* deleteNode(struct Node\* root, int key) {

struct Node\* parent = NULL;

struct Node\* curr = root;

while (curr != NULL && curr->data != key) {

parent = curr;

if (key < curr->data) {

curr = curr->left;

} else {

curr = curr->right;

}

}

if (curr == NULL) {

return root; // Key not found, return the tree as it is

}

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

if (curr != root) {

if (parent->left == curr) {

parent->left = NULL;

} else {

parent->right = NULL;

}

free(curr);

} else {

root = NULL;

}

} else if (curr->left && curr->right) {

struct Node\* successor = getMinimumKey(curr->right);

int val = successor->data;

root = deleteNode(root, successor->data);

curr->data = val;

} else {

struct Node\* child = (curr->left != NULL) ? curr->left : curr->right;

if (curr != root) {

if (parent->left == curr) {

parent->left = child;

} else {

parent->right = child;

}

free(curr);

} else {

root = child;

}

}

return root;

}

int main() {

int size;

scanf("%d", &size);

struct Node\* root = NULL;

int key;

for (int i = 0; i < size; i++) {

scanf("%d", &key);

root = insert(root, key);

}

printf("Before deletion of key value: ");

inorder(root);

int delKey;

scanf("%d", &delKey);

root = deleteNode(root, delKey);

printf("\nAfter deleting key value %d: ", delKey);

inorder(root);

return 0;

}

Q3.**Problem Statement**

You are tasked with implementing a program that creates a binary tree using an array of integers as input. The program should then print the tree nodes in three different traversals: preorder, inorder, and postorder.

**Input Format**

The input consists of a single line containing 5 integers separated by spaces.

**Output Format**

The program outputs the results of three tree traversal methods: preorder, inorder, and postorder.

Each traversal result is displayed on a separate line.

**Refer to the sample output for the formatting specifications.**

**Constraints**

In this scenario, the given test cases will fall under the following constraints:

1≤integer≤100.

**Sample Input Sample Output**

1 2 3 4 5

Preorder traversal:

1 2 4 5 3

Inorder traversal:

4 2 5 1 3

Postorder traversal:

4 5 2 3 1

Q3**Test Solution**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

/\* Given a binary tree, print its nodes according to the

"bottom-up" postorder traversal. \*/

void printPostorder(struct node \*node)

{

if (node == NULL)

return;

// first recur on left subtree

printPostorder(node->left);

// then recur on right subtree

printPostorder(node->right);

// now deal with the node

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

}

/\* Given a binary tree, print its nodes in inorder\*/

void printInorder(struct node \*node)

{

if (node == NULL)

return;

/\* first recur on left child \*/

printInorder(node->left);

/\* then print the data of node \*/

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

/\* now recur on right child \*/

printInorder(node->right);

}

/\* Given a binary tree, print its nodes in preorder\*/

void printPreorder(struct node \*node)

{

if (node == NULL)

return;

/\* first print data of node \*/

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

/\* then recur on left subtree \*/

printPreorder(node->left);

/\* now recur on right subtree \*/

printPreorder(node->right);

}

int main()

{

int arr[10];

for (int i = 0; i < 10; i++)

scanf("%d", &arr[i]);

struct node \*root = newNode(arr[0]);

root->left = newNode(arr[1]);

root->right = newNode(arr[2]);

root->left->left = newNode(arr[3]);

root->left->right = newNode(arr[4]);

printf("Preorder traversal:\n");

printPreorder(root);

printf("\nInorder traversal:\n");

printInorder(root);

printf("\nPostorder traversal:\n");

printPostorder(root);

return 0;

}

Top of Form

Bottom of Form

Top of Form

Bottom of Form

**DATA STRUCTURES\_ASSESSMENT\_WEEK 8**

**Test Summary**

* No. of Questions: 2 Total Duration: 180 min

Q1. **Problem Statement**

Given a sorted array, write a function that creates a Balanced Binary Search Tree using array elements.

**Example**

**Input:**

3

1 2 3

**Output:**

2 1 3

**Explanation**:

The given input is formed like the below tree.

2

/ \

1 3

A BST is considered balanced when all elements less than 2 are on the left side of 2, and all the elements greater than 2 are on the right side.

**Input Format**

The first line of input is the integer value representing the size of the array.

The second line of input is the list of integer values, separated by spaces.

**Output Format**

The output print the pre-order traversal of the constructed Balanced BST.

The pre-order traversal should be space-separated integers.

**Refer to the sample output for the formatting specifications.**

**Constraints**

In this scenario, the given test cases will fall under the following constraints:

1 <= n <= 20.

1<= arr[i] <= 100.

**Sample Input Sample Output**

7

1 2 3 4 5 6 7

4 2 1 3 6 5 7

Q1**Test Solution**

#include<stdio.h>

#include<stdlib.h>

/\* A Binary Tree node \*/

struct TNode

{

int data;

struct TNode\* left;

struct TNode\* right;

};

struct TNode\* newNode(int data);

/\* A function that constructs Balanced Binary Search Tree from a sorted array \*/

struct TNode\* sortedArrayToBST(int arr[], int start, int end)

{

/\* Base Case \*/

if (start > end)

return NULL;

/\* Get the middle element and make it root \*/

int mid = (start + end)/2;

struct TNode \*root = newNode(arr[mid]);

/\* Recursively construct the left subtree and make it

left child of root \*/

root->left = sortedArrayToBST(arr, start, mid-1);

/\* Recursively construct the right subtree and make it

right child of root \*/

root->right = sortedArrayToBST(arr, mid+1, end);

return root;

}

struct TNode\* newNode(int data)

{

struct TNode\* node = (struct TNode\*)

malloc(sizeof(struct TNode));

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

void preOrder(struct TNode\* node)

{

if (node == NULL)

return;

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

preOrder(node->left);

preOrder(node->right);

}

/\* Driver program to test above functions \*/

int main()

{

int n;

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

/\* Convert List to BST \*/

struct TNode \*root = sortedArrayToBST(arr, 0, n-1);

// printf("PreOrder Traversal of constructed BST\n");

preOrder(root);

return 0;

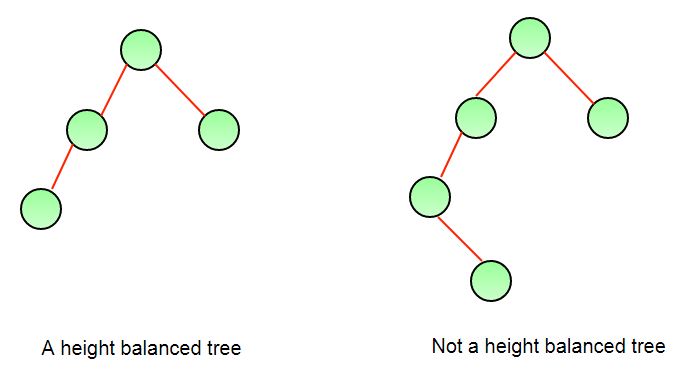
}

Q2.**Problem Statement**

Determine if a binary tree is height-balanced.

**Note:** A height-balanced binary tree is a binary tree in which the heights of the left subtree and right subtree of any node do not differ by more than 1 and both the left and right subtree are also height-balanced.

**Examples:** The tree on the left is a height-balanced binary tree, whereas the tree on the right is not a height-balanced tree. Because the left subtree of the root has a height that is 2 more than the height of the right subtree.



**Input Format**

The first line of input is an integer n, representing the size.

The second line of input is the elements x, separated by space.

**Output Format**

The output displays whether the tree is height-balanced or not.

**Refer to the sample output for the exact text.**

**Constraints**

In this scenario, the given test cases will fall under the following constraints:

1 <= n <= 20.

1<= x <= 100.

**Sample Input Sample Output**

6

1 2 3 4 5 8

The binary tree is not height-balanced

Q2**Test Solution**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

};

int max(int a, int b)

{

return (a >= b) ? a : b;

}

int height(struct Node\* node)

{

if (node == NULL)

return 0;

return 1 + max(height(node->left), height(node->right));

}

int isBalanced(struct Node\* node)

{

int lh, rh;

if (node == NULL)

return 1;

lh = height(node->left);

rh = height(node->right);

if (abs(lh - rh) <= 1 && isBalanced(node->left) && isBalanced(node->right))

return 1;

return 0;

}

struct Node\* newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

struct Node\* insert(struct Node\* node, int data)

{

if (node == NULL)

return newNode(data);

if (data < node->data)

node->left = insert(node->left, data);

else if (data > node->data)

node->right = insert(node->right, data);

return node;

}

int main()

{

int n, i, x;

struct Node \*root = NULL;

//printf("Enter the number of nodes in the binary tree: ");

scanf("%d", &n);

//printf("Enter the elements of the binary tree: ");

for(i = 0; i < n; i++)

{

scanf("%d", &x);

root = insert(root, x);

}

if(isBalanced(root))

printf("The binary tree is height-balanced\n");

else

printf("The binary tree is not height-balanced\n");

return 0;

}

Top of Form

Bottom of Form

Top of Form

Bottom of Form