***Tree traversal with recursion in C***

#include <stdio.h>

#include <stdlib.h>

/\* A binary tree node has data, pointer to left child

and a pointer to right child \*/

struct node

{

int data;

struct node\* left;

struct node\* right;

};

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

given data and NULL left and right pointers. \*/

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 sutree \*/

printPreorder(node->left);

/\* now recur on right subtree \*/

printPreorder(node->right);

}

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

int main()

{

struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("\nPreorder traversal of binary tree is \n");

printPreorder(root);

printf("\nInorder traversal of binary tree is \n");

printInorder(root);

printf("\nPostorder traversal of binary tree is \n");

printPostorder(root);

getchar();

return 0;

}

***Inorder Tree traversal without recursion C***

#include<stdio.h>

#include<stdlib.h>

#define bool int

/\* A binary tree tNode has data, pointer to left child

and a pointer to right child \*/

struct tNode

{

int data;

struct tNode\* left;

struct tNode\* right;

};

/\* Structure of a stack node. Linked List implementation is used for

stack. A stack node contains a pointer to tree node and a pointer to

next stack node \*/

struct sNode

{

struct tNode \*t;

struct sNode \*next;

};

/\* Stack related functions \*/

void push(struct sNode\*\* top\_ref, struct tNode \*t);

struct tNode \*pop(struct sNode\*\* top\_ref);

bool isEmpty(struct sNode \*top);

/\* Iterative function for inorder tree traversal \*/

void inOrder(struct tNode \*root)

{

/\* set current to root of binary tree \*/

struct tNode \*current = root;

struct sNode \*s = NULL; /\* Initialize stack s \*/

bool done = 0;

while (!done)

{

/\* Reach the left most tNode of the current tNode \*/

if(current != NULL)

{

/\* place pointer to a tree node on the stack before traversing

the node's left subtree \*/

push(&s, current);

current = current->left;

}

/\* backtrack from the empty subtree and visit the tNode

at the top of the stack; however, if the stack is empty,

you are done \*/

else

{

if (!isEmpty(s))

{

current = pop(&s);

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

/\* we have visited the node and its left subtree.

Now, it's right subtree's turn \*/

current = current->right;

}

else

done = 1;

}

} /\* end of while \*/

}

/\* UTILITY FUNCTIONS \*/

/\* Function to push an item to sNode\*/

void push(struct sNode\*\* top\_ref, struct tNode \*t)

{

/\* allocate tNode \*/

struct sNode\* new\_tNode =

(struct sNode\*) malloc(sizeof(struct sNode));

if(new\_tNode == NULL)

{

printf("Stack Overflow \n");

getchar();

exit(0);

}

/\* put in the data \*/

new\_tNode->t = t;

/\* link the old list off the new tNode \*/

new\_tNode->next = (\*top\_ref);

/\* move the head to point to the new tNode \*/

(\*top\_ref) = new\_tNode;

}

/\* The function returns true if stack is empty, otherwise false \*/

bool isEmpty(struct sNode \*top)

{

return (top == NULL)? 1 : 0;

}

/\* Function to pop an item from stack\*/

struct tNode \*pop(struct sNode\*\* top\_ref)

{

struct tNode \*res;

struct sNode \*top;

/\*If sNode is empty then error \*/

if(isEmpty(\*top\_ref))

{

printf("Stack Underflow \n");

getchar();

exit(0);

}

else

{

top = \*top\_ref;

res = top->t;

\*top\_ref = top->next;

free(top);

return res;

}

}

/\* Helper function that allocates a new tNode with the

given data and NULL left and right pointers. \*/

struct tNode\* newtNode(int data)

{

struct tNode\* tNode = (struct tNode\*)

malloc(sizeof(struct tNode));

tNode->data = data;

tNode->left = NULL;

tNode->right = NULL;

return(tNode);

}

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

int main()

{

struct tNode \*root = newtNode(1);

root->left = newtNode(2);

root->right = newtNode(3);

root->left->left = newtNode(4);

root->left->right = newtNode(5);

inOrder(root);

getchar();

return 0;

}

***Inorder Tree traversal without recursion C++***

// C++ program to print inorder traversal

// using stack.

#include<bits/stdc++.h>

using namespace std;

/\* A binary tree Node has data, pointer to left child

and a pointer to right child \*/

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node (int data)

{

this->data = data;

left = right = NULL;

}

};

/\* Iterative function for inorder tree

traversal \*/

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

while (curr != NULL || s.empty() == false)

{

/\* Reach the left most Node of the

curr Node \*/

while (curr != NULL)

{

/\* place pointer to a tree node on

the stack before traversing

the node's left subtree \*/

s.push(curr);

curr = curr->left;

}

/\* Current must be NULL at this point \*/

curr = s.top();

s.pop();

cout << curr->data << " ";

/\* we have visited the node and its

left subtree. Now, it's right

subtree's turn \*/

curr = curr->right;

} /\* end of while \*/

}

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

int main()

{

struct Node \*root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

inOrder(root);

return 0;

}

1. Write a program to compute the following operations on binary tree
2. Height of a binary tree
3. List of leaf nodes
4. Number of nodes
5. Degree of each node
6. Ancestors of a given node
7. A full node is a node with two children. Write a program to compute all full nodes in the given binary tree.