Lab Assignment-12

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 09/11/21

QUES 1: [1] Write a menu driven program to perform the following operations on a

* Binary Search Tree (BST).
* Insert a node (process of creation)
* Find the height of the tree
* Check whether the tree is a fully complete binary tree or not.
* Count the number of nodes with degree 0, 1 and 2.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

typedef *struct* Node

{

*int* data;

*struct* Node \*right;

*struct* Node \*left;

} Node;

*void* insert(Node \*\**root*, *int* *val*)

{

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

    temp->data = *val*;

    temp->left = NULL;

    temp->right = NULL;

    if (!\**root*)

    {

        \**root* = temp;

        return;

    }

    Node \*ptrR = \**root*;

    Node \*ptr\_prev;

    while (ptrR)

    {

        ptr\_prev = ptrR;

        if (ptrR->data >= *val*)

            ptrR = ptrR->left;

        else

            ptrR = ptrR->right;

    }

    if (ptr\_prev->data > *val*)

        ptr\_prev->left = temp;

    else

        ptr\_prev->right = temp;

}

*void* preorder(Node \**root*)

{

    if (!*root*)

        return;

    printf("%d->", *root*->data);

    preorder(*root*->left);

    preorder(*root*->right);

}

*void* inorder(Node \**root*)

{

    if (!*root*)

        return;

    inorder(*root*->left);

    printf("%d->", *root*->data);

    inorder(*root*->right);

}

*void* postorder(Node \**root*)

{

    if (!*root*)

        return;

    postorder(*root*->left);

    postorder(*root*->right);

    printf("%d->", *root*->data);

}

*int* treeHeight(Node \**root*)

{

    if (!*root*)

        return -1;

    return ((treeHeight(*root*->left) > treeHeight(*root*->right)) ? treeHeight(*root*->left) + 1 : treeHeight(*root*->right) + 1);

}

*int* isFullyComplete(Node \**root*)

{

    if (!*root*)

        return 1;

    else if (!*root*->left && !*root*->right)

        return 1;

    else if (*root*->left && *root*->right)

        return isFullyComplete(*root*->left) &&

               isFullyComplete(*root*->right);

    return 0;

}

*void* order\_Nodes(Node \**root*, *int* \**zero*, *int* \**first*, *int* \**second*)

{

    if (!*root*)

        return;

    if (!*root*->left && !*root*->right)

        (\**zero*)++;

    else if (!*root*->left || !*root*->right)

        (\**first*)++;

    else

        (\**second*)++;

    order\_Nodes(*root*->left, *zero*, *first*, *second*);

    order\_Nodes(*root*->right, *zero*, *first*, *second*);

}

*void* count\_degree(Node \**root*)

{

*int* zero = 0;

*int* first = 0;

*int* second = 0;

    order\_Nodes(*root*, &zero, &first, &second);

    printf("Degree zero: %d\n", zero);

    printf("Degree first: %d\n", first);

    printf("Degree second: %d\n", second);

}

*int* main()

{

    Node \*root = NULL;

*int* choice, val, t\_hold;

    do

    {

        printf("1) Insert\n2) Preorder\n3) postorder\n4) Inorder\n5) Tree Height\n");

        printf("6) Fully Complete BST\n7) Number of nodes of each degree\n");

        printf("8) Exit\n->: ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

        case 1:

            printf("Enter value: ");

            scanf("%d", &val);

            insert(&root, val);

            break;

        case 2:

            preorder(root);

            printf("\b\b \n");

            break;

        case 3:

            postorder(root);

            printf("\b\b \n");

            break;

        case 4:

            inorder(root);

            printf("\b\b \n");

            break;

        case 5:

            printf("Height of the tree: %d\n", treeHeight(root));

            break;

        case 6:

            if (isFullyComplete(root))

                printf("True\n");

            else

                printf("False\n");

            break;

        case 7:

            count\_degree(root);

            break;

        default:

            printf("Exiting...\n");

        }

        printf("-------------------------------------------------\n");

    } while (choice >= 1 && choice <= 7);

    return 0;

}

OUTPUT:

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 1

Enter value: 80

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 1

Enter value: 85

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 1

Enter value: 70

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 1

Enter value: 75

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 1

Enter value: 60

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 2

80->70->60->75->85 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 3

60->75->70->85->80 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 4

60->70->75->80->85 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 5

Height of the tree: 2

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 6

True

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 7

Degree zero: 3

Degree first: 0

Degree second: 2

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Tree Height

6) Fully Complete BST

7) Number of nodes of each degree

8) Exit

->: 8

Exiting...

-------------------------------------------------

QUES 2: [2] Write a program to construct an expression tree for a given postfix

expression.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

typedef *struct* Node

{

*char* data;

*struct* Node \*left;

*struct* Node \*right;

} Node;

typedef *struct* Stack

{

    Node \*data;

*struct* Stack \*link;

} Stack;

*int* isEmpty\_stack(Stack \**stack*)

{

    if (!*stack*)

        return 1;

    return 0;

}

*void* push(Stack \*\**stack*, Node \**data*)

{

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

    temp->data = *data*;

    temp->link = \**stack*;

    \**stack* = temp;

}

Node \*pop(Stack \*\**stack*)

{

    if (isEmpty\_stack(\**stack*))

    {

        printf("\nUnderflow!");

        return NULL;

    }

    Stack \*temp = (\**stack*);

    \**stack* = (\**stack*)->link;

    Node \*val = temp->data;

    free(temp);

    return val;

}

*void* preorder(Node \**root*)

{

    if (!*root*)

        return;

    printf("%c", *root*->data);

    preorder(*root*->left);

    preorder(*root*->right);

}

*void* inorder(Node \**root*)

{

    if (!*root*)

        return;

    inorder(*root*->left);

    printf("%c", *root*->data);

    inorder(*root*->right);

}

Node \*scanExpression(*char* \**expression*)

{

    if (!*expression*)

        return NULL;

*int* i = 0;

*char* operations[6] = {'+', '-', '\*', '/', '^', '%'};

    Stack \*stack = NULL;

    while (*expression*[i] != '\0')

    {

        if ((*expression*[i] >= 'A' && *expression*[i] <= 'Z') ||

            (*expression*[i] >= 'a' && *expression*[i] <= 'z'))

        {

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

            temp->right = temp->left = NULL;

            temp->data = *expression*[i];

            push(&stack, temp);

        }

        else

        {

            for (*int* j = 0; j < 6; j++)

            {

                if (operations[j] == *expression*[i])

                {

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

                    temp->right = temp->left = NULL;

                    temp->data = *expression*[i];

                    temp->right = pop(&stack);

                    temp->left = pop(&stack);

                    push(&stack, temp);

                    break;

                }

            }

        }

        i++;

    }

    return pop(&stack);

}

*int* main()

{

*char* \*input;

    printf("Input Expression: ");

    scanf(" %s", input);

    Node \*root = scanExpression(input);

    printf("\nInorder: ");

    inorder(root);

    printf("\nPreorder: ");

    preorder(root);

    printf("\n");

    return 0;

}

OUTPUT:

Input Expression: AB+CDE+\*\*

Inorder: A+B\*C\*D+E

Preorder: \*+AB\*C+DE