**Assignment: 11**

**EXP.**Kindly use the following link for making submission related to binary tree.

1. WAP to create a tree with a given inputs from user.

2. WAP to write the inorder,pre-order and post order traversal.

3. WAP to insert element at desired locations.

4. WAP to delete the elements from tree, take input from the user for deleting the entries from trees.

5. WAP to check for the key elements in node, if the node has multiple of 3, replace it with 5 or if Key elements has any vowel element, replace those with 'Z'.

**Coding:**

#include <stdio.h>

#include <stdlib.h>

struct tree\_node

{

    int val;

    char v;

    struct tree\_node \*left;

    struct tree\_node \*right;

};

typedef struct tree\_node tnode;

struct tree

{

    tnode \*root;

};

typedef struct tree tree;

tnode \*create\_node(int ele, char e)

{

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

    temp->val = ele;

    temp->v = e;

    temp->left = NULL;

    temp->right = NULL;

    return temp;

}

tree \*create\_tree()

{

    tree \*bt = (tree \*)malloc(sizeof(tree \*));

    bt->root = NULL;

    printf("Tree created!\n");

    return bt;

}

void insert\_wherever(tree \*bt, tnode \*e)

{

    int choice;

    int flag = 0;

    if (bt->root == NULL)

    {

        bt->root = e;

        return;

    }

    else

    {

        tnode \*temp = bt->root;

        while (temp != NULL)

        {

            printf("Press 1 to move left\n");

            printf("Press 2 to move right\n");

            scanf("%d", &choice);

            switch (choice)

            {

            case 1:

                if (temp->left == NULL)

                {

                    temp->left = create\_node(e->val, e->v);

                    flag = 1;

                    break;

                }

                else

                {

                    temp = temp->left;

                }

                break;

            case 2:

                if (temp->right == NULL)

                {

                    temp->right = create\_node(e->val, e->v);

                    flag = 1;

                    break;

                }

                else

                {

                    temp = temp->right;

                }

                break;

            default:

                printf("Wrong choice!\n");

                break;

            }

            if (flag == 1)

            {

                break;

            }

        }

    }

    printf("Element Inserted!\n");

}

tnode \*find(tree \*bt, tnode \*root, int ele)

{

    tnode \*temp;

    if (root == NULL)

    {

        return NULL;

    }

    else if (root->val == ele)

    {

        return root;

    }

    else

    {

        temp = find(bt, root->left, ele);

        if (temp != NULL)

        {

            return temp;

        }

        temp = find(bt, root->right, ele);

        return temp;

    }

}

tnode \*rightmost\_node(tree \*bt)

{

    tnode \*temp = bt->root;

    tnode \*t;

    while (temp->right->right != NULL)

    {

        temp = temp->right;

    }

    t = temp->right;

    temp->right = NULL;

    return t;

}

void delete\_node(tree \*bt, int ele)

{

    tnode \*del\_node = find(bt, bt->root, ele);

    tnode \*deep\_node = rightmost\_node(bt);

    if (del\_node == deep\_node)

    {

        free(deep\_node);

        free(del\_node);

    }

    else if (del\_node == NULL)

    {

        printf("Cannot delete!\n");

        return;

    }

    else

    {

        del\_node->val = deep\_node->val;

        del\_node->v = deep\_node->v;

        free(deep\_node);

    }

    printf("Node deleted!\n");

}

void inorder\_traversal(tree \*bt, tnode \*root)

{

    if (root == NULL)

    {

        return;

    }

    else

    {

        inorder\_traversal(bt, root->left);

        printf("{%d, %c} ", root->val, root->v);

        inorder\_traversal(bt, root->right);

    }

}

void preorder\_traversal(tree \*bt, tnode \*root)

{

    if (root == NULL)

    {

        return;

    }

    else

    {

        printf("{%d, %c} ", root->val, root->v);

        inorder\_traversal(bt, root->left);

        inorder\_traversal(bt, root->right);

    }

}

void postorder\_traversal(tree \*bt, tnode \*root)

{

    if (root == NULL)

    {

        return;

    }

    else

    {

        inorder\_traversal(bt, root->left);

        inorder\_traversal(bt, root->right);

        printf("{%d, %c} ", root->val, root->v);

    }

}

void modify(tree \*bt, tnode \*root)

{

    if (root == NULL)

    {

        return;

    }

    else

    {

        modify(bt, root->left);

        if ((root->v == 'a') || (root->v == 'e') || (root->v == 'i') || (root->v == 'o') || (root->v == 'u') || (root->v == 'A') || (root->v == 'E') || (root->v == 'I') || (root->v == 'O') || (root->v == 'U'))

        {

            root->v = 'Z';

        }

        if (root->val % 3 == 0)

        {

            root->val = 5;

        }

        modify(bt, root->right);

    }

}

void display(tree \*bt, tnode \*ptr, int level)

{

    int i;

    if (ptr != NULL)

    {

        display(bt, ptr->right, level + 1);

        printf("\n");

        if (ptr == bt->root)

        {

            printf("Root->:  ");

        }

        else

        {

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

            {

                printf("       ");

            }

        }

        printf("{%d, %c}", ptr->val, ptr->v);

        display(bt, ptr->left, level + 1);

    }

    printf("\n");

}

void displaycall(tree \*bt)

{

    if (bt->root == NULL)

    {

        printf("No nodes present....\n");

        return;

    }

    display(bt, bt->root, 1);

}

int main()

{

    int choice = 0;

    tree \*bt;

    tnode \*n;

    int ele;

    char e;

    while (choice != -1)

    {

        printf("1. Create tree.\n");

        printf("2. Insert into tree.\n");

        printf("3. Delete from tree.\n");

        printf("4. Preorder traversal.\n");

        printf("5. Inorder traversal.\n");

        printf("6. Postorder traversal.\n");

        printf("7. Modify tree.\n");

        printf("8. Display tree.\n");

        printf("9. Exit.\n");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            bt = create\_tree();

            break;

        case 2:

            printf("Enter int value to be inserted:\n");

            scanf("%d", &ele);

            printf("Enter char value to be inserted:\n");

            scanf("\n%c", &e);

            n = create\_node(ele, e);

            insert\_wherever(bt, n);

            break;

        case 3:

            printf("Enter int value for node to be deleted:\n");

            scanf("%d", &ele);

            delete\_node(bt, ele);

            break;

        case 4:

            printf("Preorder traversal of tree:\n");

            preorder\_traversal(bt, bt->root);

            printf("\n");

            break;

        case 5:

            printf("Inorder traversal of tree:\n");

            inorder\_traversal(bt, bt->root);

            printf("\n");

            break;

        case 6:

            printf("Postorder traversal of tree:\n");

            postorder\_traversal(bt, bt->root);

            printf("\n");

            break;

        case 7:

            modify(bt, bt->root);

            printf("Tree modified!\n");

            break;

        case 8:

            displaycall(bt);

            break;

        case 9:

            printf("Exiting...\n");

            break;

        default:

            printf("Wrong choice!!\n");

            break;

        }

    }

    return 0;

}

**Output:**















