**Binary Search tree Creation and traversal**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

typedef struct Node

{

    int data;

    struct Node\* left;

    struct Node\* right;

}Node;

Node\* root=NULL;

void create(){

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

    printf("Enter data:");

    scanf("%d",&new->data);

    new->left=new->right=NULL;

    root=new;

    while(1){

        printf("add more nodes?(y/n):");

        char ch;

        scanf(" %c",&ch);

        if(ch=='y' || ch=='Y'){

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

            printf("Enter data:");

            scanf("%d",&new->data);

            new->left=new->right=NULL;

            int item=new->data;

            Node\* curr=root;

            Node\* ptr;

            while(curr){

                ptr=curr;

                curr=(item>=curr->data)?curr->right:curr->left;

            }

            if(item <(ptr->data)) ptr->left=new;

            else ptr->right=new;

        }

        else return;

    }

}

void inorder(Node\* root){

    if(root==NULL) return;

    inorder(root->left);

    printf("%d\t",root->data);

    inorder(root->right);

}

void preorder(Node\* root){

    if(root==NULL) return;

    printf("%d\t",root->data);

    preorder(root->left);

    preorder(root->right);

}

void postorder(Node\* root){

    if(root==NULL) return;

    postorder(root->left);

    postorder(root->right);

    printf("%d\t",root->data);

}

typedef struct Queue {

    Node\* array[MAX]; // Array to hold nodes

    int front;                   // Index of the front element

    int rear;                    // Index of the rear element

} Queue;

// Function to initialize an empty queue

void initializeQueue(Queue\* q) {

    q->front = 0;

    q->rear = 0;

}

// Function to check if the queue is empty

int isEmpty(Queue\* q) {

    return q->front == q->rear;

}

// Function to add an item to the queue

void enqueue(Queue\* q, Node\* item) {

    if (q->rear < MAX) { // Check for space in the queue

        q->array[q->rear++] = item; // Add item and increment rear

    }

}

// Function to remove an item from the queue

Node\* dequeue(Queue\* q) {

    if (!isEmpty(q)) { // Check if the queue is not empty

        return q->array[q->front++]; // Return front item and increment front

    }

    return NULL; // Return NULL if the queue is empty

}

// Level order traversal function

void levelOrder(Node\* root) {

    if (root == NULL) {

        return;

    }

    Queue q; // Declare a queue

    initializeQueue(&q); // Initialize the queue

    enqueue(&q, root); // Start with the root node

    while (!isEmpty(&q)) {

        Node\* current = dequeue(&q); // Get the front node

        // Print the current node's data

        printf("%d ", current->data);

        // Enqueue left child if it exists

        if (current->left != NULL) {

            enqueue(&q, current->left);

        }

        // Enqueue right child if it exists

        if (current->right != NULL) {

            enqueue(&q, current->right);

        }

    }

}

int main(){

    int ch;

    printf("create-1 inorder-2 preorder-3 postorder-4 levelorder-5 exit-6\n");

    while(1){

        printf("\nEnter choice: ");

        scanf("%d", &ch);

        switch (ch){

            case 1: create();break;

            case 2: inorder(root);break;

            case 3: preorder(root);break;

            case 4: postorder(root);break;

            case 5: levelOrder(root);break;

            case 6:exit(0);

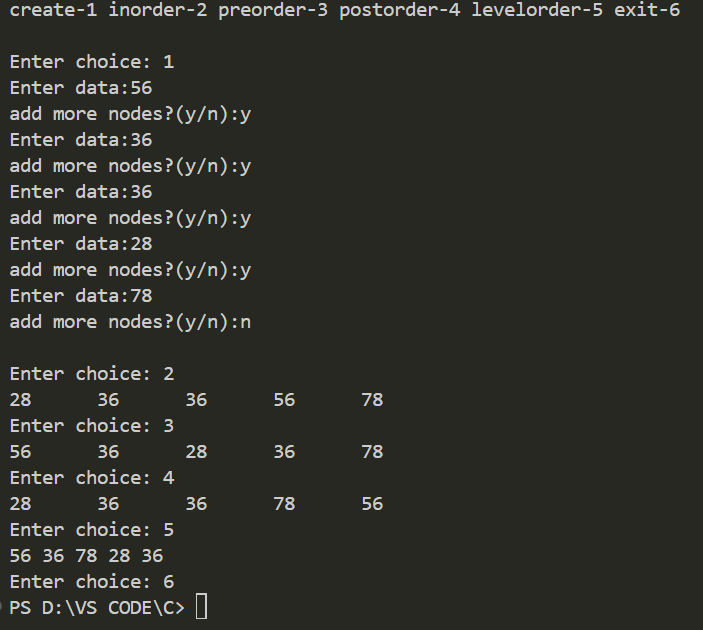
        }

    }

    return 0;

}

**Output:**

****