Name – Shrinivas Hatyalikar
Div – CS-B
Roll no – 24

WAP to create a Binary tree and perform non-recursive Preorder, Inorder and Postorder traversal on it.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct node{
    struct node *left;
    int data;
    struct node *right;
};
struct Queue{
    int size;
    int front;
    int rear;
    struct node **Q;
};
void create(struct Queue *q,int size){
    q->size=size;
    q->front=q->rear=0;
    q->Q=(struct node **)malloc(q->size*sizeof(struct node *));
}
void enqueue(struct Queue *q,struct node *x){
    if((q->rear+1)%q->size==q->front)
    printf("Queue is Full");
    else
    {
    q->rear=(q->rear+1)%q->size;
    q->Q[q->rear]=x;
    }
}
struct node * dequeue(struct Queue *q){
    struct node* x=NULL;
```

```
    if(q->front==q->rear){
        printf("Queue is Empty\n");
    }
    else{
    q->front=(q->front+1)%q->size;
    x=q->Q[q->front];
    }

    return x;
}
int isEmpty(struct Queue q){
    return q.front==q.rear;
}

struct Stack{
    int size;
    int top;
    struct node **S;
};

void createstack(struct Stack *s,int size){
    s->size=size;
    s->top=-1;
    s->S=(struct node*)malloc(s->size*sizeof(struct node));
}



void push(struct Stack *st,struct node*t){
    if(st->top==st->size-1){
        printf("Stack OverFlow\n");
    }
    else{
        st->top++;
        st->S[st->top]=t;
    }
}

struct node* pop(struct Stack *st){
    struct node*x=NULL;
```

```
    if(st->top==-1){
        printf("Stack Underflow\n");
    }
    else{
        x=st->S[st->top];
        st->top--;
    }
    return x;
}


int isEmptystack(struct Stack st){
    if(st.top==-1){
        return 1;
    }
    else{
        return 0;
    }
}

struct node *root=NULL;

void createTree(){
    struct node *p,*t;
    int x=0;
    struct Queue q;
    printf("Enter Root Value ");     //Creating a root
    scanf("%d",&x);
    root=(struct node *)malloc(sizeof(struct node));
    root->data=x;
    root->right=root->left=NULL;
    create(&q,100);
    enqueue(&q,root);

    while(!isEmpty(q)){
        p=dequeue(&q);

        printf("Enter Value of Left Child %d ",p->data);
        scanf("%d",&x);
```

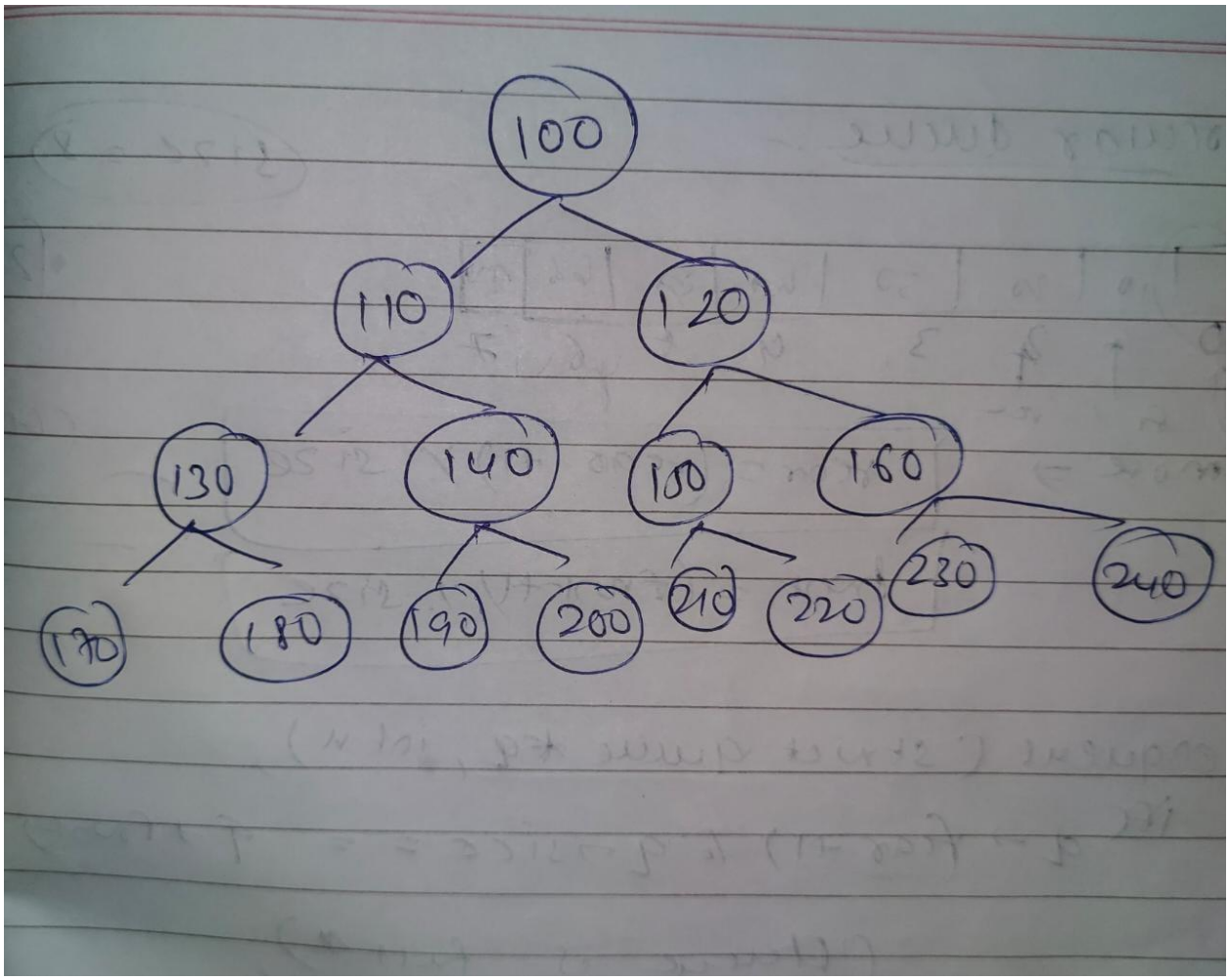
```
        if(x!=-1){
            t=(struct node *)malloc(sizeof(struct node));
            t->data=x;
            t->left=t->right=NULL;
            p->left=t;
            enqueue(&q,t);
        }

        printf("Enter Value of Right Child %d ",p->data);
        scanf("%d",&x);

        if(x!=-1){
            t=(struct node *)malloc(sizeof(struct node));
            t->data=x;
            t->left=t->right=NULL;
            p->right=t;
            enqueue(&q,t);
        }

    }
}

void preorder(struct node *p){
    if(p){
        printf("%d ",p->data);
        preorder(p->left);
        preorder(p->right);
    }
}


void postorder(struct node *p)
{
    if(p){
        postorder(p->left);
        postorder(p->right);
        printf("%d ",p->data);
    }
}
```

```
void inorder(struct node *p){
   if(p){
      inorder(p->left);
      printf("%d ",p->data);
      inorder(p->right);
   }
}

void preorderiteration(struct node*root){
        struct node *t=root;
        struct Stack st;
        createstack(&st,100);
        while(t!=NULL || !isEmptystack(st)){
                if(t!=NULL){
                        printf("%d ",t->data);
                        push(&st,t);
                        t=t->left;
                }
                else{
                        t=pop(&st);
                        t=t->right;

                }
        }
}
void postorderiteration(struct node*root){
        struct node *t=root;
        struct Stack st;
        createstack(&st,100);
        long int temp;
        while(t!=NULL || !isEmptystack(st)){
                if(t!=NULL){
                        push(&st,t);
                        t=t->left;
                }
                else{
                        temp=pop(&st);
                        if(temp>0){
                                push(&st,-temp);
                                t=((struct node*)temp)->right;
```

```
                        }
                        else{
                                printf("%d ",((struct node *)(-1*temp))->data);
                                t=NULL;
                        }
                }
        }
}
void Inorderiteration(struct node *root){
        struct Stack st;
        struct node *t=root;
        int x;
        createstack(&st,100);
        while(t!=NULL || !isEmptystack(st)){
                if(t!=NULL){
                        push(&st,t);
                        t=t->left;
                }
                else{
                        t=pop(&st);
                        printf(" %d ",t->data);
                        t=t->right;
                }
        }

}

int main(){
   createTree();
   printf("\nPREORDER with Recurrsion\n");
   preorder(root);
   printf("\n\nPOSTORDER with Recurrsion\n");
   postorder(root);
   printf("\n\nINORDER with Recurrsion\n");
   inorder(root);
   printf("\n\nPREORDER with Iteration\n");
   preorderiteration(root);
   printf("\n\nPOSTORDER with Iteration\n");
   postorderiteration(root);
   printf("\n\nPREORDER with Iteration\n");
```

```
    Inorderiteration(root);

}
```

INPUT TREE:



```
PS C:\Users\sheeh\OneDrive\Desktop\C\output> & .\'trees.exe'
Enter Root Value 100
Enter Value of Left Child 100 110
Enter Value of Right Child 100 120
Enter Value of Left Child 110 130
Enter Value of Right Child 110 140
Enter Value of Left Child 120 150
Enter Value of Right Child 120 160
Enter Value of Left Child 130 170
Enter Value of Right Child 130 180
Enter Value of Left Child 140 190
Enter Value of Right Child 140 200
Enter Value of Left Child 150 210
Enter Value of Right Child 150 220
Enter Value of Left Child 160 230
Enter Value of Right Child 160 240
Enter Value of Left Child 170 -1
Enter Value of Right Child 170 -1
Enter Value of Left Child 180 -1
Enter Value of Right Child 180 -1
Enter Value of Left Child 190 -1
Enter Value of Right Child 190 -1
Enter Value of Left Child 200 -1
Enter Value of Right Child 200 -1
Enter Value of Left Child 210 -1
Enter Value of Right Child 210 -1
Enter Value of Left Child 220 -1
Enter Value of Right Child 220 -1
Enter Value of Left Child 230 -1
Enter Value of Right Child 230 -1
Enter Value of Left Child 240 -1
Enter Value of Right Child 240 -1
```

```
PREORDER with Recurrsion
100 110 130 170 180 140 190 200 120 150 210 220 160 230 240

POSTORDER with Recurrsion
170 180 130 190 200 140 110 210 220 150 230 240 160 120 100

INORDER with Recurrsion
170 130 180 110 190 140 200 100 210 150 220 120 230 160 240

PREORDER with Iteration
100 110 130 170 180 140 190 200 120 150 210 220 160 230 240

POSTORDER with Iteration
170 180 130 190 200 140 110 210 220 150 230 240 160 120 100

PREORDER with Iteration
 170  130  180  110  190  140  200  100  210  150  220  120  230  160  240
```