Q1. WAP Write the following menu driven program for the binary search tree

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

Binary Search Tree Menu

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

0. Quit

1. Create

2. In-Order Traversal

3. Pre-Order Traversal

4. Post-Order traversal

5. Search

6. Find Smallest Element

7. Find Largest Element

8. Deletion of Tree

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

Enter your choice:

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 50

struct node

{

    struct node \*left;

    int data;

    struct node \*right;

};

struct node \*search\_nrec(struct node \*root, int skey);

struct node \*min\_nrec(struct node \*root);

struct node \*max\_nrec(struct node \*root);

struct node \*insert\_nrec(struct node \*root, int ikey);

struct node \*del\_nrec(struct node \*root, int dkey);

struct node \*case\_c(struct node \*root, struct node \*par, struct node \*ptr);

struct node \*case\_b(struct node \*root, struct node \*par, struct node \*ptr);

struct node \*case\_a(struct node \*root, struct node \*par, struct node \*ptr);

struct node \*del\_nrec1(struct node \*root, int item);

void nrec\_pre(struct node \*root);

void nrec\_in(struct node \*root);

void nrec\_post(struct node \*root);

void level\_trav(struct node \*root);

void display(struct node \*ptr, int level);

struct node \*queue[MAX];

int front = -1, rear = -1;

void insert\_queue(struct node \*item);

struct node \*del\_queue();

int queue\_empty();

struct node \*stack[MAX];

int top = -1;

void push\_stack(struct node \*item);

struct node \*pop\_stack();

int stack\_empty();

int main()

{

    struct node \*root = NULL, \*ptr;

    int choice, k;

    while (1)

    {

        printf("\n");

        printf("0.Quit\n");

        printf("1.Search\n");

        printf("2.Insert\n");

        printf("3.Delete\n");

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

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

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

        printf("7.Level order traversal\n");

        printf("8.Find minimum and maximum\n");

        printf("9.Display\n");

        printf("\nEnter your choice : ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 0:

            exit(1);

        default:

            printf("\nWrong choice\n");

        case 1:

            printf("\nEnter the key to be searched : ");

            scanf("%d", &k);

            ptr = search\_nrec(root, k);

            if (ptr == NULL)

                printf("\nKey not present\n");

            else

                printf("\nKey present\n");

            break;

        case 2:

            printf("\nEnter the key to be inserted : ");

            scanf("%d", &k);

            root = insert\_nrec(root, k);

            break;

        case 3:

            printf("\nEnter the key to be deleted : ");

            scanf("%d", &k);

            root = del\_nrec(root, k);

            break;

        case 4:

            nrec\_pre(root);

            break;

        case 5:

            nrec\_in(root);

            break;

        case 6:

            nrec\_post(root);

            break;

        case 7:

            level\_trav(root);

            break;

        case 8:

            ptr = min\_nrec(root);

            if (ptr != NULL)

                printf("\nMinimum key is %d\n", ptr->data);

            ptr = max\_nrec(root);

            if (ptr != NULL)

                printf("\nMaximum key is %d\n", ptr->data);

            break;

        case 9:

            printf("\n");

            display(root, 0);

            printf("\n");

            break;

        }

    }

    return 0;

}

struct node \*search\_nrec(struct node \*ptr, int skey)

{

    while (ptr != NULL)

    {

        if (skey < ptr->data)

            ptr = ptr->left;

        else if (skey > ptr->data)

            ptr = ptr->right;

        else

            return ptr;

    }

    return NULL;

}

struct node \*insert\_nrec(struct node \*root, int ikey)

{

    struct node \*tmp, \*par, \*ptr;

    ptr = root;

    par = NULL;

    while (ptr != NULL)

    {

        par = ptr;

        if (ikey < ptr->data)

            ptr = ptr->left;

        else if (ikey > ptr->data)

            ptr = ptr->right;

        else

        {

            printf("\nDuplicate key");

            return root;

        }

    }

    tmp = (struct node \*)malloc(sizeof(struct node));

    tmp->data = ikey;

    tmp->left = NULL;

    tmp->right = NULL;

    if (par == NULL)

        root = tmp;

    else if (ikey < par->data)

        par->left = tmp;

    else

        par->right = tmp;

    return root;

}

struct node \*del\_nrec1(struct node \*root, int dkey)

{

    struct node \*par, \*ptr, \*child, \*succ, \*parsucc;

    ptr = root;

    par = NULL;

    while (ptr != NULL)

    {

        if (dkey == ptr->data)

            break;

        par = ptr;

        if (dkey < ptr->data)

            ptr = ptr->left;

        else

            ptr = ptr->right;

    }

    if (ptr == NULL)

    {

        printf("\ndkey not present in tree");

        return root;

    }

    if (ptr->left != NULL && ptr->right != NULL)

    {

        parsucc = ptr;

        succ = ptr->right;

        while (succ->left != NULL)

        {

            parsucc = succ;

            succ = succ->left;

        }

        ptr->data = succ->data;

        ptr = succ;

        par = parsucc;

    }

    if (ptr->left != NULL)

        child = ptr->left;

    else

        child = ptr->right;

    if (par == NULL)

        root = child;

    else if (ptr == par->left)

        par->left = child;

    else

        par->right = child;

    free(ptr);

    return root;

}

struct node \*del\_nrec(struct node \*root, int dkey)

{

    struct node \*par, \*ptr;

    ptr = root;

    par = NULL;

    while (ptr != NULL)

    {

        if (dkey == ptr->data)

            break;

        par = ptr;

        if (dkey < ptr->data)

            ptr = ptr->left;

        else

            ptr = ptr->right;

    }

    if (ptr == NULL)

        printf("dkey not present in tree\n");

    else if (ptr->left != NULL && ptr->right != NULL)

        root = case\_c(root, par, ptr);

    else if (ptr->left != NULL)

        root = case\_b(root, par, ptr);

    else if (ptr->right != NULL)

        root = case\_b(root, par, ptr);

    else

        root = case\_a(root, par, ptr);

    return root;

}

struct node \*case\_a(struct node \*root, struct node \*par, struct node \*ptr)

{

    if (par == NULL)

        root = NULL;

    else if (ptr == par->left)

        par->left = NULL;

    else

        par->right = NULL;

    free(ptr);

    return root;

}

struct node \*case\_b(struct node \*root, struct node \*par, struct node \*ptr)

{

    struct node \*child;

    if (ptr->left != NULL)

        child = ptr->left;

    else

        child = ptr->right;

    if (par == NULL)

        root = child;

    else if (ptr == par->left)

        par->left = child;

    else

        par->right = child;

    free(ptr);

    return root;

}

struct node \*case\_c(struct node \*root, struct node \*par, struct node \*ptr)

{

    struct node \*succ, \*parsucc;

    parsucc = ptr;

    succ = ptr->right;

    while (succ->left != NULL)

    {

        parsucc = succ;

        succ = succ->left;

    }

    ptr->data = succ->data;

    if (succ->left == NULL && succ->right == NULL)

        root = case\_a(root, parsucc, succ);

    else

        root = case\_b(root, parsucc, succ);

    return root;

}

struct node \*min\_nrec(struct node \*ptr)

{

    if (ptr != NULL)

        while (ptr->left != NULL)

            ptr = ptr->left;

    return ptr;

}

struct node \*max\_nrec(struct node \*ptr)

{

    if (ptr != NULL)

        while (ptr->right != NULL)

            ptr = ptr->right;

    return ptr;

}

void nrec\_pre(struct node \*root)

{

    struct node \*ptr = root;

    if (ptr == NULL)

    {

        printf("Tree is empty\n");

        return;

    }

    push\_stack(ptr);

    while (!stack\_empty())

    {

        ptr = pop\_stack();

        printf("%d  ", ptr->data);

        if (ptr->right != NULL)

            push\_stack(ptr->right);

        if (ptr->left != NULL)

            push\_stack(ptr->left);

    }

    printf("\n");

}

void nrec\_in(struct node \*root)

{

    struct node \*ptr = root;

    if (ptr == NULL)

    {

        printf("Tree is empty\n");

        return;

    }

    while (1)

    {

        while (ptr->left != NULL)

        {

            push\_stack(ptr);

            ptr = ptr->left;

        }

        while (ptr->right == NULL)

        {

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

            if (stack\_empty())

                return;

            ptr = pop\_stack();

        }

        printf("%d  ", ptr->data);

        ptr = ptr->right;

    }

    printf("\n");

}

void nrec\_post(struct node \*root)

{

    struct node \*ptr = root;

    struct node \*q;

    if (ptr == NULL)

    {

        printf("Tree is empty\n");

        return;

    }

    q = root;

    while (1)

    {

        while (ptr->left != NULL)

        {

            push\_stack(ptr);

            ptr = ptr->left;

        }

        while (ptr->right == NULL || ptr->right == q)

        {

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

            q = ptr;

            if (stack\_empty())

                return;

            ptr = pop\_stack();

        }

        push\_stack(ptr);

        ptr = ptr->right;

    }

    printf("\n");

}

void level\_trav(struct node \*root)

{

    struct node \*ptr = root;

    if (ptr == NULL)

    {

        printf("Tree is empty\n");

        return;

    }

    insert\_queue(ptr);

    while (!queue\_empty())

    {

        ptr = del\_queue();

        printf("%d ", ptr->data);

        if (ptr->left != NULL)

            insert\_queue(ptr->left);

        if (ptr->right != NULL)

            insert\_queue(ptr->right);

    }

    printf("\n");

}

void insert\_queue(struct node \*item)

{

    if (rear == MAX - 1)

    {

        printf("Queue Overflow\n");

        return;

    }

    if (front == -1)

        front = 0;

    rear = rear + 1;

    queue[rear] = item;

}

struct node \*del\_queue()

{

    struct node \*item;

    if (front == -1 || front == rear + 1)

    {

        printf("Queue Underflow\n");

        return 0;

    }

    item = queue[front];

    front = front + 1;

    return item;

}

int queue\_empty()

{

    if (front == -1 || front == rear + 1)

        return 1;

    else

        return 0;

}

void push\_stack(struct node \*item)

{

    if (top == (MAX - 1))

    {

        printf("Stack Overflow\n");

        return;

    }

    top = top + 1;

    stack[top] = item;

}

struct node \*pop\_stack()

{

    struct node \*item;

    if (top == -1)

    {

        printf("Stack Underflow....\n");

        exit(1);

    }

    item = stack[top];

    top = top - 1;

    return item;

}

int stack\_empty()

{

    if (top == -1)

        return 1;

    else

        return 0;

}

void display(struct node \*ptr, int level)

{

    int i;

    if (ptr == NULL) /\*Base Case\*/

        return;

    else

    {

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

        printf("\n");

        for (i = 0; i < level; i++)

            printf("    ");

        printf("%d", ptr->data);

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

    }

}

**OUTPUT**

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 5

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 6

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 4

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 7

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 3

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 8

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 2

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 9

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 2

Enter the key to be inserted : 1

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 9

9

8

7

6

5

4

3

2

1

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 4

5 4 3 2 1 6 7 8 9

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 5

1 2 3 4 5 6 7 8 9

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 6

1 2 3 4 9 8 7 6 5

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 7

5 4 6 3 7 2 8 1 9

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 8

Minimum key is 1

Maximum key is 9

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 1

Enter the key to be searched : 5

Key present

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice :

3

Enter the key to be deleted : 6

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 9

9

8

7

5

4

3

2

1

0.Quit

1.Search

2.Insert

3.Delete

4.Preorder Traversal

5.Inorder Traversal

6.Postorder Traversal

7.Level order traversal

8.Find minimum and maximum

9.Display

Enter your choice : 0