NAME- Aniket Kalbhor

PRN- 12210601

ROLL- CSB-48

CODE- **4.Write a menudriven program in C to perform following  operations  on BST**

**a.Create b. Insert c.Delete d.Traversals -Inorder with recursion and without recursion, Preoder  with recursion and without recursion ,Postorder  with recursion and without recursion**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct node {

int data;

struct node \*l;

struct node \*r;

};

struct node \*root = NULL;

struct node \*temp = NULL;

struct stack {

struct node \*T;

struct stack \*next;

};

void push(struct node \*root, struct stack \*\*top)

{

struct stack \*temp = (struct stack \*)malloc(sizeof(struct stack));

temp->T= root;

temp->next = (\*top);

(\*top) = temp;

}

struct node \*pop(struct stack \*\*top)

{

struct stack \*temp;

struct node \*n;

temp = (\*top);

(\*top) = (\*top)->next;

n = temp->T;

free(temp);

return n;

}

bool emptyStack(struct stack \*top)

{

if (top == NULL)

return 1;

else

return 0;

}

struct Node\* topStack(struct stack\*\* top)

{

return((\*top)->T);

};

struct node\*createBST(int data)

{

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

if (newNode == NULL)

printf("Memory Allocation failed\n");

newNode->data = data;

newNode->l = NULL;

newNode->r = NULL;

return newNode;

}

struct node \*insertNode(struct node\* root, int newdata)

{

if(root == NULL){

root = createBST(newdata);

}

else if (newdata>root->data)

root->r = insertNode(root->r, newdata);

else

root->l = insertNode(root->l, newdata);

return root;

}

void RpreOrder(struct node \*root)

{

if(root != NULL){

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

RpreOrder(root->l);

RpreOrder(root->r);

}

}

void RinOrder(struct node \*root)

{

if(root != NULL){

RinOrder(root->l);

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

RinOrder(root->r);

}

}

void RpostOrder(struct node \*root)

{

if(root != NULL){

RpostOrder(root->l);

RpostOrder(root->r);

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

}

}

void StackInOrder(struct node \*root)

{

struct stack \*top = NULL;

while(root != NULL){

push(root, &top);

root = root->l;

}

while(!emptyStack(top)){

root = pop(&top);

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

root = root->r;

while(root != NULL){

push(root, &top);

root = root->l;

}

}

}

void StackPreOrder(struct node \*root)

{

struct stack \*top = NULL;

while(root != NULL){

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

push(root, &top);

root = root->l;

}

while(!emptyStack(top)){

root = pop(&top);

root = root->r;

while(root != NULL){

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

push(root, &top);

root = root->l;

}

}

}

void StackPostOrder(struct node \*root)

{

struct stack \*top = NULL;

struct node \*prev = NULL;

struct node \*t = root;

while(t != NULL){

push(t, &top);

t = t->l;

}

while(!emptyStack(top)){

t = topStack(&top);

if(t->r == NULL || t->r == prev){

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

pop(&top);

prev = t;

}

else{

t = t->r;

while(t != NULL){

push(t, &top);

t = t->l;

}

}

}

}

struct node \*minValueNode(struct node \*node)

{

if (node == NULL)

return NULL;

while(node->l != NULL)

node = node->l;

return node;

}

struct node \*deleteNode(struct node \*root, int key)

{

struct node \*dtemp;

if(root==NULL)

return root;

else if (key<root->data)

root->l = deleteNode(root->l, key);

else if (key>root->data)

root->r = deleteNode(root->r, key);

else{

if(root->l == NULL){

dtemp = root->r;

free(root);

return dtemp;

}

else if(root->r == NULL){

dtemp = root->l;

free(root);

return dtemp;

}

else{

dtemp = minValueNode(root->r);

root->data = dtemp->data;

root->r = deleteNode(root->r, dtemp->data);

}

}

return root;

}

int main()

{

int choice;

int createdata;

int newdata;

int deletedata;

while(1){

printf("\n1. Create BST\n2. Insert\n3. Delete\n4. Recursive PreOrder\n5. Recursive InOrder\n6. Recursive PostOrder\n7. Exit\n8. Stack InOrder\n9. Stack PreOrder\n10. Stack PostOrder\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice){

case 1:

printf("Enter the data: ");

scanf("%d", &createdata);

root = createBST(createdata);

break;

case 2:

printf("Enter the data: ");

scanf("%d", &newdata);

root = insertNode(root, newdata);

break;

case 3:

printf("Enter the data: ");

scanf("%d", &deletedata);

root = deleteNode(root, deletedata);

break;

case 4:

RpreOrder(root);

break;

case 5:

RinOrder(root);

break;

case 6:

RpostOrder(root);

break;

case 7:

exit(0);

break;

case 8:

StackInOrder(root);

break;

case 9:

StackPreOrder(root);

break;

case 10:

StackPostOrder(root);

break;

}

}

}

OUTPUT-

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 1

Enter the data: 25

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 2

Enter the data: 20

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 2

Enter the data: 36

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 2

Enter the data: 27

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 4

25 20 36 27

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 5

20 25 27 36

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 6

20 27 36 25

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 3

Enter the data: 20

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 8

25 27 36

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 9

25 36 27

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder

Enter your choice: 10

27 36 25

1. Create BST

2. Insert

3. Delete

4. Recursive PreOrder

5. Recursive InOrder

6. Recursive PostOrder

7. Exit

8. Stack InOrder

9. Stack PreOrder

10. Stack PostOrder