**Binary Tree**

**Done By:** Rohit Karunakaran **Roll No:** 58

**Aim:** Implement a Binary Tree

**Data Structures used:** Linked List, Binary Tree

**Algorithm for Insertion**

**Input:** The root node (root) and the key after which the element is to be inserted

**Output :** The binary tree with the node inserted

**Data Structure :** Binary Tree

**Steps**

* 1. Step 1: Start
  2. Step 2: ptr = Srearch(root,key)
  3. Step 3: if(ptr == NULL) then
  4. Step 1: print(“No element found”)
  5. Step 2: exit
  6. Step 4: endif
  7. Step 5: If(ptr→lc ==NULL or ptr→rc==NULL) then
  8. Step 1: read option to insert the node left or right
  9. Step 2: if(option == l) then
  10. Step1: if(ptr→lc == NULL)
  11. Step1: new=GetNode(node)
  12. Step 2: new→data = item
  13. Step 3: new→lc=new→rc =NULL
  14. Step 4: ptr→lc = new
  15. Step 2: else
  16. Step 1: print(“Insertion not possible”)
  17. Step 2: exit
  18. Step 3: endif
  19. Step 3: else if(option == r)then
  20. Step 1: if(ptr→rc= NULL) then
  21. Step 1: new = getNode(node)
  22. Step 2: new→data = item
  23. Step 3: new→lc=new→rc= NULL
  24. Step 4: ptr→rc = new
  25. Step 2 : else
  26. Step 1: print(“Insertion not posiible”)
  27. Step 2: exit
  28. Step 3:endif
  29. Step 3: endif
  30. Step 6: endif
  31. Step 7: Stop

**Algorithm for Deleting a node**

**Input:** Root node of the binary tree, the element to be deleted

**Output:** Binary tree with the element deleted

**Data Structure used:** Binary tree

Steps

Step 1: Start

Step 2: getParent(root,elem)

Step 3: if(parent → rc == elem) then

Step 1: ptr = parent → rc

Step 4: else

Step 1: ptr = parent →lc

Step 5: endif

Step 6: if(ptr→rc!=NULL || ptr→lc!=NULL) then

Step 1: print(“ptr is a leaf node it cant be deleted”)

Step 7: else if(ptr==parent →rc) then

Step 1:parent →rc=NULL

Step 8:else

Step 1: parent → lc =NULL

Step 9: endif

Step 10: returnNode(ptr)

**Algorithm for Inorder Traversal**

**Input:** Root node of the binary tree

**Output :** All the nodes of the binary tree visited in an inorder fashion

**Data Structure used:** Binary trees

Steps

1. Step 1: Start
2. Step 2: if(root!=NULL) then
3. Step 1: inorder\_traversal(root→lc)
4. Step 2: visit(root)
5. Step 3: inorder\_traversal(root→ rc)
6. Step 3: else
7. Step 1: return
8. Step 4: endif
9. Step 5: Stop

**Algorithm for Postorder Traversal**

**Input:** Root node of the binary tree

**Output :** All the nodes of the binary tree visited in an postorder fashion

**Data Structure used:** Binary trees

Steps

1. Step 1: Start
2. Step 2: if(root!=NULL) then
3. Step 1: postorder\_traversal(root→lc)
4. Step 2: postorder\_traversal(root→ rc)
5. Step 3: visit(root)
6. Step 3: else
7. Step 1: return
8. Step 4: endif
9. Step 5: Stop

**Algorithm for Preorder Traversal**

**Input:** Root node of the binary tree

**Output :** All the nodes of the binary tree visited in an preorder fashion

**Data Structure used:** Binary trees

Steps

1. Step 1: Start
2. Step 2: if(root!=NULL) then
3. Step 1: visit(root)
4. Step 2: preorder\_traversal(root→lc)
5. Step 3: preorder\_traversal(root→ rc)
6. Step 3: else
7. Step 1: return
8. Step 4: endif
9. Step 5: Stop

**Algorithm for Searching**

**Input:** Root node (root) and the value to be searched(key)

**Output:** A pointer to the corresponding node, if the key is present in the binary tree else null

**Data Structure:** Linked List, Binary Tree

**Steps**

1. Step 1: Start
2. Step 2: ptr=root
3. Step 3: if(ptr→data!=key) then
4. Step 1: if(ptr→lc!=NULL) then
5. Step 1: Search(root→lc,key)
6. Step2: endif
7. Step3: if(ptr→rc!=NULL) then
8. Step 1: Search(root→rc,key)
9. Step4: endif
10. Step 5: return (NULL)
11. Step 4: else
12. Step 1 : return ptr //base case
13. Step 5: endif

**Program Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Binary tree

\* Done By: Rohit Karunakaran

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct binary\_tree\_node{

struct binary\_tree\_node\* lc;

struct binary\_tree\_node\* rc;

int value;

}node;

/\*

node\* init\_tree(){

root\_node = (node\*) malloc(sizeof(node));

}

\*/

node\* search\_node(node\* root, int value){

node\* ptr=NULL;

if(root->value != value){

if(root->lc==NULL && root->rc==NULL){

return NULL;

}

else{

if(root->lc!=NULL){

ptr = search\_node(root->lc, value);

if(ptr!=NULL){

return ptr;

}

}

if(root->rc!=NULL){

ptr = search\_node(root->rc,value) ;

if(ptr !=NULL){

return ptr;

}

}

return ptr;

}

}

else{

return root;

}

}

node\* search\_parent(node\* root, int value){

node\* ptr = NULL;

if(root!=NULL){

if(root->lc !=NULL && root->rc!=NULL){

if(root->lc ->value == value||root->rc->value==value){

return root;

}else{

ptr = search\_parent(root->lc, value);

if(ptr == NULL){

ptr = search\_parent(root->rc, value);

}

return ptr;

}

}

else if(root -> lc ==NULL && root ->rc ==NULL){

return NULL;

}

else{

if(root->lc == NULL){

if(root->rc->value==value){

return root;

}

else{

ptr = search\_parent(root->rc,value);

return ptr;

}

}

else{

if(root->lc->value==value){

return root;

}

else{

ptr = search\_parent(root->lc,value);

return ptr;

}

}

}

}

else{

return NULL;

}

}

void insert\_node(node\* root,int value){

node\* ptr = search\_node(root,value);

char c;

if(ptr!=NULL){

fflush(stdin);

printf("Insert Node as Left child or as a right child: ");

scanf("\n%c",&c);

if(c == 'l'){

if(ptr->lc == NULL){

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

printf("Enter the value to be inserted: ");

scanf("%d",&(tmp->value));

tmp->rc = NULL;

tmp->lc = NULL;

ptr->lc = tmp;

}

else{

printf("Insertion at the left node of %d is not possible\n",ptr->value);

}

}

else if(c =='r'){

if(ptr->rc == NULL){

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

printf("Enter the value to be inserted: ");

scanf("%d",&(tmp->value));

tmp->rc = NULL;

tmp->lc = NULL;

ptr->rc = tmp;

}

else{

printf("Insertion at the right node of %d is not possible\n",ptr->value);

}

}

else{

printf("Proper option was not chosen\n");

}

}

else{

printf("Value %d not found!!!!\nInsertion not possible\n",value);

}

}

void inorder\_traversal(node\* root){

if(root!=NULL){

inorder\_traversal(root->lc);

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

inorder\_traversal(root->rc);

}

else{

return;

}

}

void postorder\_traversal(node\* root){

if(root!=NULL){

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

postorder\_traversal(root->lc);

postorder\_traversal(root->rc);

}

else{

return;

}

}

void preorder\_traversal(node\* root){

if(root!=NULL){

preorder\_traversal(root->lc);

preorder\_traversal(root->rc);

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

}

else{

return;

}

}

void delete\_node(node\*\* root, int value)

{

node\* parent = search\_parent(\*root, value);

if(parent == NULL){

if((\*root)->value == value&&(\*root)->rc==NULL&&(\*root)->lc==NULL){

free(\*root);

\*root = NULL;

}

else if((\*root)->value == value){

printf("Deletion not possible\n");

}

else{

printf("The value %d not found in the tree\n\n",value);

}

}

else{

if(parent->rc !=NULL&&parent->rc->value==value){

if(parent->rc->rc==NULL && parent->rc->lc==NULL){

free(parent->rc);

parent->rc =NULL;

}

else{

printf("Deletion not possible\n");

}

}

else{

if(parent->lc->lc==NULL && parent->lc->rc==NULL){

free(parent->lc);

parent->lc =NULL;

}

else{

printf("Deletion not possible\n");

}

}

}

}

int menu(node\* root){

printf("Binary Tree implementation\n");

int RUN=1;

int choice;

int elem;

while(RUN){

printf("\nMenu\n");

printf("1.Insert\n");

printf("2.Inorder traversal\n");

printf("3.Preorder traversal\n");

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

printf("5.Delete Node\n");

printf("6. Exit\n");

printf("Enter Choice: ");

scanf("%d",&choice);

switch(choice){

case 1: if(root==NULL){

root = (node\*)malloc(sizeof(node));

printf("Enter the value to be inserted: ");

scanf("%d",&elem);

root->value = elem;root->lc = NULL;root->rc = NULL;

}

else{

printf("Enter the value to be searched for : ");

scanf("%d",&elem);

insert\_node(root,elem);

}

break;

case 2: if(root!=NULL){

printf("\nInorder Traversal : ");

inorder\_traversal(root);

}

else

printf("The tree is Empty!!!!\n");

break;

case 3: if(root!=NULL){

printf("\nProerder Traversal : ");

preorder\_traversal(root);

}

else

printf("The tree is Empty!!!!\n");

break;

case 4: if(root!=NULL){

printf("\nPostorder Traversal : ");

postorder\_traversal(root);

}

else

printf("The tree is Empty!!!!\n");

break;

case 5: printf("Enter the value to be deleted: ");

scanf("%d",&elem);

delete\_node(&root,elem);

break;

case 6: RUN=0;

break;

}

}

return RUN;

}

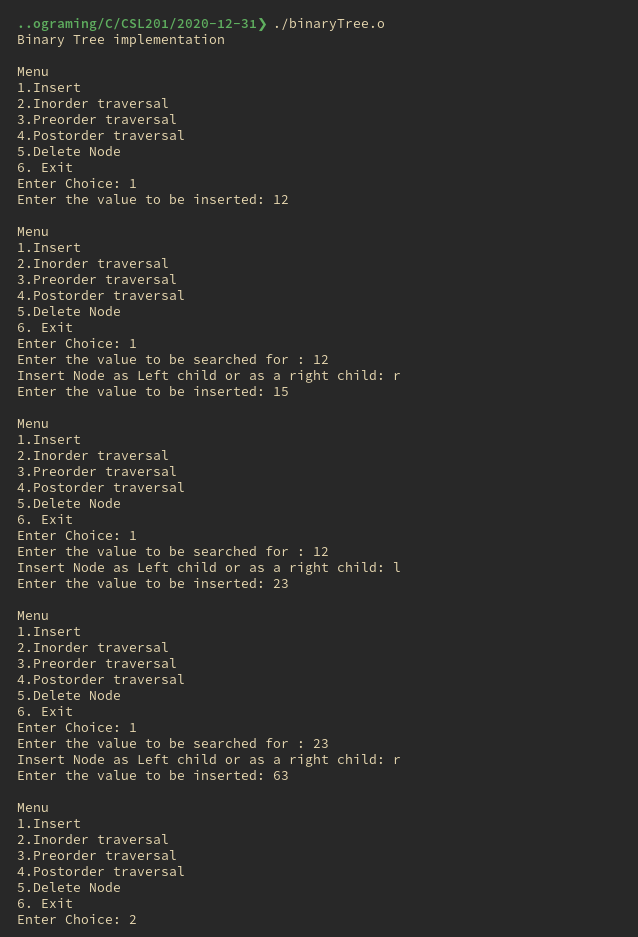
int main(){

node\* root = NULL;

return menu(root);

}

**Sample Input and Output**

****

