Experiment 15
Date 22-12-2022

Binary Tree

Aim

Write a menu driven C Program to create a binary tree using linked list with the following operations:

a) Insert a new node
b) Inorder traversal
c) Preorder traversal
d) Postorder traversal
e) Delete a node

1 Binary Tree

1.1 Algorithm

```
Step 1:Start
Step 2: Create structure having members data and self referential LC and RC
Step 3: Declare structure variable root and dynamically allocate memory to the node
Step 4: Set default values of data, LC and RC of root as NULL
Step 5: Print ''Menu''
        Print ''1. Insert''
        Print ''2. Preorder Traversal''
        Print ''3. Inorder''
        Print ''4. Postorder Traversal''
        Print ''5. Delete''
        Print ''6. Exit''
Step 6: Input option
Step 7: If option=1, then input key, insert(key), print tree elements using
        inorder(Header→RC)
Step 8: If option=2, then preorder(Header→RC)
Step 9: If option=3, then inorder(Header→RC)
Step 10: If option=4, then postorder(Header→RC)
Step 11: If option=5, then input key, delete(key), print tree elements
         using inorder(Header→RC)
Step 12: If option=6, then goto STEP 15
Step 13: Go to Step 5
Step 14: Stop
```

```
Start of the function insert(key)
Step 1: Let ptr ← ptr1 ← Header → RC
Step 2: If ptr = NULL, goto STEP 7
Step 3: Let ptr1 ← ptr
Step 4: Check if key is less than or greater than ptr → data,
        accordingly set ptr to ptr→LC or ptr to ptr → RC, goto step 3
Step 5: If key = ptr → data, then print ''Item already exists in tree'',
        return
Step 6: Allocate memory for node, new using get node()
Step 7: If ptr1=NULL, then let Header → RC ← new
Step 8: Check if key is less than or greater than ptr1 → data, ac-
        cordingly set ptr1 → LC or ptr1 → RC to new
Step 9: Set new → to key, new → RC and new → LC to NULL

Start of the function preorder(ptr)
Step 1: If ptr = NULL, return
Step 2: Print ptr → data
Step 3: preorder(ptr → LC)
Step 4: preorder(ptr → RC)

Start of the function inorder(ptr)
Step 1: If ptr = NULL, return
Step 2: inorder(ptr → LC)
Step 3: Print ptr → data
Step 4: inorder(ptr → RC)

Start of the function postorder(ptr)
Step 1: If ptr=NULL, return
Step 2: postorder(ptr→LC)
Step 3: postorder(ptr→RC)
Step 4: Print ptr→data
Start of the function delete(key)
Step 1: Let ptr ← Header → RC
Step 2: If ptr=NULL, print ''Item does not exist in tree'', return
Step 3: Check if key is less than or greater than ptr → data,
        accordingly set prnt to ptr, ptr to ptr → LC or ptr to
        ptr → RC, goto STEP 43
Step 4: If key=ptr → data, check whether the node is a leaf node,
         or has one or both children
Step 5: For case 1, set the LC or RC of prnt node, which points to
        the key, to NULL
Step 6: For case 2, set the LC or RC of prnt node, which points to
         the key, to the child node of ptr
Step 7: For case 3, let ptr1 be the inorder successor of ptr, let
        value ← ptr1 → data, delete(value), set ptr → data to value
```

1.2 Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*rchild;
struct node*lchild;
}*temp,*head,*new,*ptr,*LC,*RC,*k,*ptrr,*ptrl,*parent;
void create(struct node*,int);
void insert();
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);
struct node* search_parent(struct node*,struct node*,int);
void delete(struct node*,int);
void main()
{
int c,l,d;
char ch='y';
while(ch=='y' || ch=='Y')
{
printf("1.Insert node\n2.Inorder\n3.Preorder\n4.Postorder\n5.Delete\n6.Exit\n");
printf("Input choice\n");
scanf("%d",&c);
switch(c)
{
case 1:
{
head=(struct node*)malloc(sizeof(struct node));
printf("Enter data of root\n");
scanf("%d",&d);
create(head,d);
break;
}
case 2:
{
temp=head;
inorder(temp);
printf("\n");
break;
```

```
}
case 3:
{
temp=head;
preorder(temp);
printf("\n");
break;
}
case 4:
{
temp=head;
postorder(temp);
printf("\n");
break;
}
case 5:
{
int e;
printf("Element to be deleted\n");
scanf("%d",&e);
delete(head,e);
break;
}
case 6:
{
exit(0);
break;
}
default:
{
printf("Invalid choice\n");
}
}
printf("Do you wish to continue\n");
scanf(" %c",&ch);
}
}

void create(struct node*ptr,int item)
{
int x;
char ch;
if(ptr!=NULL)
{
ptr->data=item;
printf("If %d has left subtree\n",item);
```

```
scanf(" %c",&ch);
if(ch=='y'||ch=='Y')
{
LC=(struct node*)malloc(sizeof(struct node));
ptr->lchild=LC;
printf("Enter data of node\n");
scanf("%d",&x);
create(LC,x);
}
else
{
LC=(struct node*)malloc(sizeof(struct node));
LC=NULL;
ptr->lchild=NULL;
}
printf("If %d has right subtree\n",item);
scanf(" %c",&ch);
if(ch=='y'||ch=='Y')
{
RC=(struct node*)malloc(sizeof(struct node));
ptr->rchild=RC;
printf("Enter data of node\n");
scanf("%d",&x);
create(RC,x);
}
else
{
RC=(struct node*)malloc(sizeof(struct node));
RC=NULL;
ptr->rchild=NULL;
}
}
}
void inorder(struct node*temp)
{

if(temp!=NULL)
{
inorder(temp->lchild);
printf("%d  ",temp->data);
inorder(temp->rchild);
}
}


void preorder(struct node*temp)
```

```
{

if(temp!=NULL)
{
printf("%d  ",temp->data);
preorder(temp->lchild);
preorder(temp->rchild);
}
}
void postorder(struct node*temp)
{

if(temp!=NULL)
{
postorder(temp->lchild);
postorder(temp->rchild);
printf("%d  ",temp->data);
}
}
void delete(struct node* head,int key)
{
if (head == NULL)
{
printf("\n\nTree is Empty\n\n");
return ;
}
parent = search_parent(head,NULL,key);
if (parent == NULL)
printf("\nKey not Found!!\n");
else
{
ptrl = parent->lchild;
ptrr = parent->rchild;
if (ptrl != NULL)
{
if (ptrl->data == key)
if(ptrl->lchild == NULL && ptrl->rchild==NULL)
parent->lchild = NULL;
else
{
printf("\nElement cannot be deleted!!!\n");
return ;
}
}
if (ptrr!=NULL)
{
```

```
if (ptrr->data == key)
if(ptrr->lchild == NULL && ptrr->rchild==NULL)
parent->rchild = NULL;
else
{
printf("\nElement cannot be deleted!!!\n");
return ;
}
}
}
}
struct node* search_parent(struct node* ptr,struct node* prev,int key)
{
if (ptr == NULL)
return NULL;
if (ptr->data == key)
{
k = prev;
return prev;
}
else
{
search_parent(ptr->lchild,ptr,key);
search_parent(ptr->rchild,ptr,key);
}
return k;
}
```

1.3 Sample Output

```
1.Insert node
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Exit
Input choice
1
Enter data of root
1
If 1 has left subtree
y
Enter data of node
2
If 2 has left subtree
y
Enter data of node
3
If 3 has left subtree
n
If 3 has right subtree
n
If 2 has right subtree
y
Enter data of node
4
If 4 has left subtree
n
If 4 has right subtree
n
If 1 has right subtree
n
Do you wish to continue
y
1.Insert node
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Exit
Input choice
2
3  2  4  1
Do you wish to continue
y
1.Insert node
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Exit
```

```
Input choice
3
1  2  3  4
Do you wish to continue
y
1.Insert node
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Exit
Input choice
4
3  4  2  1
Do you wish to continue
n
```

1.4 Result

Implemented binary tree using a self referential structure which has a the fields rchild,lchild and data. Functions create(),insert(),inorder(), preorder(),postorder() and delete() are used to create a binary tree, insert an element into the binary tree, inorder,preorder and postorder traversal and deletion of the binary tree respectively.