19BCE0829
KANDRA KSHEERAJ

CSE2003- Data Structures and Algorithms

LAB ASSIGNMENT-3

Slot: L53 + L54

Faculty: GAYATHRI P Mam

Date: 21-8-2020

19BCE0829 KSHEERAJ KANDRA

QUESTION-1:

1. Menu-driven C program to implement stack using linked list.

PSEUDO CODE:

```
/* Node creation */
Struct StackNode
{ int data;
struct StackNode *next;
}*top = 0;

void main(){
 int choice=0;
 while(choice!=4){
 Read input choice;
 switch(choice){
     case 1:
     push();

     case 2:
     pop();

     case 3:
     display();

     case 4:
     break;
     };
  }}
```

```
Void Push(){
 define intval,val;
 struct node*ptr=(struct node*)malloc(sizeof(struct node));
 if(ptr==NULL)
     /* "not able to push the element" */
 else
     read input val;
 if(head==NULL)
 {   ptr->val=val;
     ptr->next=NULL;
     head=ptr;}
 else
 {   ptr->val=val;
     ptr->next=head;
     head=ptr;}
}

void Pop(){
define item;
 struct node*ptr;
 if(head==NULL)
     /* "Underflow" */
 else
 {   item=head->val;
     ptr=head;
     head=head->next;
     free(ptr);}
}

void display(){
 struct node*ptr;
 ptr=head;
 if(ptr==NULL)
     /* "Stack is empty" */
 else
     /* "Printing Stack elements"; */
     while(ptr!=NULL){
       print ptr->val;
       ptr=ptr->next; }}
```

PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();

struct node
{
int intval,val;
struct node*next;
};
struct node*head;

void main()
{
 int choice=0;
 printf("\nProgram for Stack Operations using linkedlist \n");
 printf("\n-----------------------------------------\n");
 while(choice!=4)
 {
 printf("\n\nChose one from the below options...\n");
 printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
     case 1:
     {
     push();
     break;
     }
     case 2:
     {
     pop();
     break;
     }
     case 3:
     {
```

```
    display();
    break;
    }
    case 4:
    {
    break;
    }
    default:
    {
    printf("Please Enter valid choice");
    }
    };
  }
}
void push()
{
 int intval,val;
 struct node*ptr=(struct node*)malloc(sizeof(struct node));
 if(ptr==NULL)
 {
 printf("not able to push the element");
 }
 else
 {
 printf("Enter the value: ");
 scanf("%d",&val);
 if(head==NULL)
 {
 ptr->val=val;
 ptr->next=NULL;
 head=ptr;
 }
 else
 {
 ptr->val=val;
 ptr->next=head;
 head=ptr;
 }
 printf("Item pushed");
```

```
 }
}
void pop()
{
 int item;
 struct node*ptr;
 if(head==NULL)
 {
 printf("Underflow");
 }
 else
 {
 item=head->val;
 ptr=head;
 head=head->next;
 free(ptr);
 printf("Item popped");
 }
}
void display()
{
 int i;
 struct node*ptr;
 ptr=head;
 if(ptr==NULL)
 {
 printf("Stack is empty\n");
 }
 else
 {
 printf("Printing Stack elements\n");
 while(ptr!=NULL)
 {
 printf("%d\n",ptr->val);
 ptr=ptr->next;
 }
 }
}
```

19BCE0829
KANDRA KSHEERAJ

PROGRAM OUTPUT:

```
Program for Stack Operations using linkedlist

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


Chose one from the below options...

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the value: 2
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the value: 8
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the value: 5
Item pushed
```

```
Chose one from the below options...

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Printing Stack elements
5
8
2


Chose one from the below options...

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
Item popped

Chose one from the below options...

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Printing Stack elements
8
2
```

QUESTION 2:

2. Menu-driven C program implement queue using linked list.

PSEUDO CODE:

```
/* Node Creation */
Struct QueueNode
{
int data;
struct QueueNode *next;
}*rear = 0, *front = 0;
```

```
/* Enqueue */
void enq(int data){
 if (rear == NULL){
     rear = (struct node *)malloc(1*sizeof(struct node));
     rear->ptr = NULL;
     rear->info = data;
     front = rear;}
 else{
     temp=(struct node *)malloc(1*sizeof(struct node));
     rear->ptr = temp;
     temp->info = data;
     temp->ptr = NULL;
     rear = temp;
     }}

/* Dequeue */
Dequeue(){
 front1 = front;
 if (front1 == NULL){
 /* "empty queue" */
 else
   if (front1->ptr != NULL){
     print front->info ;
     front1 = front1->ptr;
     free(front);
     front = front1;}
   else{
     print front->info ;
     free(front);
     front = NULL;
     rear = NULL;
 }}

void display(){
 front1 = front;
 if ((front1 == NULL) && (rear == NULL))
 /* Queue is empty */
```

```
while (front1 != rear){
     print front1->info ;
     front1 = front1->ptr;}
if (front1 == rear)
     print front1->info;}

int frontelement(){ /*display the first element in queue*/
 if ((front != NULL) && (rear != NULL))
     return(front->info);
 else
 return 0;}

void empty(){ /*check if queue is empty*/
 if ((front == NULL) && (rear == NULL))
     print "Queue empty";
 else
     print "Queue not empty";}

void queuesize(){ /*check size of queue*/
 print " Queue size", count;
}
```

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int info;
 struct node *ptr;
}*front,*rear,*temp,*front1;
int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();
int count = 0;
void main()
```

```
{
 int no, ch, e;
 printf("\n Program for Queue operations using linkedlist \n");
 printf("\n-----------------------------------------\n");
 create();
 while (1)
 {
 printf("\n\n Chose any option");
 printf("\n 1.Enque");
 printf("\n 2.Deque");
 printf("\n 3.Front element");
 printf("\n 4.Queue size");
 printf("\n 5.Empty");
 printf("\n 6.Display");
 printf("\n 7.Exit");
 printf("\n Enter choice : ");
 scanf("%d", &ch);
 switch (ch)
 {
     case 1:
     printf("Enter data : ");
     scanf("%d", &no);
     enq(no);
     break;
     case 2:
     deq();
     break;
     case 3:
     e = frontelement();
     if (e != 0)
     printf("Front element : %d", e);
     else
     printf("\n No front element in Queue as queue is empty");
     break;
     case 4:
     queuesize();
     break;
     case 5:
     empty();
```

```
    break;
    case 6:
    display();
    break;
    case 7:
    exit(0);
    break;
    default:
    printf("Wrong choice, Please enter correct choice ");
    break;
    }
  }
}
void create()
{
 front = rear = NULL;
}
void queuesize()
{
 printf("\n Queue size : %d", count);
}
void enq(int data)
{
 if (rear == NULL)
 {
 rear = (struct node *)malloc(1*sizeof(struct node));
 rear->ptr = NULL;
 rear->info = data;
 front = rear;
 }
 else
 {
 temp=(struct node *)malloc(1*sizeof(struct node));
 rear->ptr = temp;
 temp->info = data;
 temp->ptr = NULL;
 rear = temp;
 }
 count++;
```

```
}
void display()
{
 front1 = front;
 if ((front1 == NULL) && (rear == NULL))
 {
 printf("Queue is empty");
 return;
 }
 while (front1 != rear)
 {
 printf("%d ", front1->info);
 front1 = front1->ptr;
 }
 if (front1 == rear)
 printf("%d", front1->info);
}
void deq()
{
 front1 = front;
 if (front1 == NULL)
 {
 printf("\n Error: Trying to display elements from empty
queue");
 return;
 }
 else
 if (front1->ptr != NULL)
 {
 front1 = front1->ptr;
 printf("\n Dequed value : %d", front->info);
 free(front);
 front = front1;
 }
 else
 {
 printf("\n Dequed value : %d", front->info);
 free(front);
 front = NULL;
```

```
 rear = NULL;
 }
 count--;
}
int frontelement()
{
 if ((front != NULL) && (rear != NULL))
 return(front->info);
 else
 return 0;
}
void empty()
{
 if ((front == NULL) && (rear == NULL))
 printf("\n Queue empty");
 else
 printf("Queue not empty");
}
```

PROGRAM OUTPUT:

19BCE0829
KANDRA KSHEERAJ

```
 Program for Queue operations using linkedlist

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


 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 1
Enter data : 7


 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 1
Enter data : 5


 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 6
7 5
```

```
 Enter choice : 3
Front element : 7

 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 5
Queue not empty

 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 4

 Queue size : 2

 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 2

 Dequed value : 7
```

```
 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 6
5

 Chose any option
 1.Enque
 2.Deque
 3.Front element
 4.Queue size
 5.Empty
 6.Display
 7.Exit
 Enter choice : 4

 Queue size : 1
```

19BCE0829
KANDRA KSHEERAJ

QUESTION 3:

3. Menu-driven C program to implement polynomial addition and multiplication using LL.

PSEUDO CODE:

```
define MAX
typedef struct node{
    define coeff;
    struct node *next;
 }node;
node * init();
void read(node *h1);
void print(node *h1);
node * add(node *h1,node *h2);
node * multiply(node *h1,node *h2);

void main(){
  node *h1=NULL,*h2=NULL,*h3=NULL;
  define option;
  do{
  read input option;
  switch(option){
     case 1:
     h1=init();
     read(h1);

     case 2:
     h2=init();
     read(h2);

     case 3:
     h3=add(h1,h2);
     print " 1st polynomial -> " ;
     print(h1);
     print " 2nd polynomial -> ";
     print(h2);
     print " Sum = ";
```

```
     print(h3);
     case 4:
     h3=multiply(h1,h2);
     /* " 1st polynomial -> " */;
     print(h1);
     /* " 2nd polynomial -> " */;
     print(h2);
     /* " Product = " */;
     print(h3);
     }
}}
void read(node *h){
  define n,i,j,power,coeff;
  node *p;
  p=init();

/* "Enter number of terms :" */
read input n;
for (i=0;i<n;i++){

/* " term(power & coeff)" */
  read input power, coeff;
  for(p=h,j=0;j<power;j++)
  p=p->next;
  p->coeff=coeff;
}}
void print(node *p){
  define i;
  for(i=0;p!=NULL;i++,p=p->next)
  if(p->coeff!=0)
  print p->coeff;
}
node * add(node *h1, node *h2){
  node *h3,*p;
  h3=init();
  p=h3;
  while(h1!=NULL){
     h3->coeff=h1->coeff+h2->coeff;
     h1=h1->next;
```

```
    h2=h2->next;
    h3=h3->next;
  }
return(p);
}
node * multiply(node *h1, node *h2){
  node *h3,*p,*q,*r;
  define i,j,k,coeff,power;
  h3=init();
  for(p=h1,i=0;p!=NULL;p=p->next,i++)
  for(q=h2,j=0;q!=NULL;q=q->next,j++){
    coeff=p->coeff * q->coeff;
    power=i+j;
    for(r=h3,k=0;k<power;k++)
    r=r->next;
    r->coeff=r->coeff+coeff;
  }
return(h3);
}
node * init(){
  define i;
  node *h=NULL,*p;
  for(i=0;i<MAX;i++){
  p=(node*)malloc(sizeof(node));
  p->next=h;
  p->coeff=0;
  h=p;
}}
```

PROGRAM CODE:

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#define MAX 17
typedef struct node
 {
int coeff;
struct node *next;
```

```
 }node;
node * init();
void read(node *h1);
void print(node *h1);
node * add(node *h1,node *h2);
node * multiply(node *h1,node *h2);
void main()
{
node *h1=NULL,*h2=NULL,*h3=NULL;
int option;
 printf("\n Program to implement polynomial addition and
multiplication using LL \n");
 printf("\n-----------------------------------------------------
------------------\n");
do
{
printf("\n 1.create 1st polynomial");
printf("\n 2.create 2nd polynomial");
printf("\n 3.Add polynomials");
printf("\n 4.Multiply polynomials");
printf("\n 5.Quit");
printf("\n Enter your choice :");
scanf("%d",&option);
switch(option)
{
                case 1:
                h1=init();
                read(h1);
                break;
                case 2:
                h2=init();
                read(h2);
                break;
                case 3:
                h3=add(h1,h2);
                printf("\n 1st polynomial -> ");
                print(h1);
                printf("\n 2nd polynomial -> ");
                print(h2);
```

```
                printf("\n Sum = ");
                print(h3);
                break;
                case 4:
                h3=multiply(h1,h2);
                printf("\n 1st polynomial -> ");
                print(h1);
                printf("\n 2nd polynomial -> ");
                print(h2);
                printf("\n Product = ");
                print(h3);
                break;
                }
}while(option!=5);
}
void read(node *h)
{
int n,i,j,power,coeff;
node *p;
p=init();
printf("\nEnter number of terms :");
scanf("%d",&n);
for (i=0;i<n;i++)
{ printf("Enter a term(power & coeff)");
scanf("%d%d",&power,&coeff);
for(p=h,j=0;j<power;j++)
 p=p->next;
p->coeff=coeff;
}
}
void print(node *p)
{
 int i;
for(i=0;p!=NULL;i++,p=p->next)
if(p->coeff!=0)
printf("%dX^%d ",p->coeff,i);
}
node * add(node *h1, node *h2)
{
```

```
 node *h3,*p;
 h3=init();
 p=h3;
 while(h1!=NULL)
 {
h3->coeff=h1->coeff+h2->coeff;
h1=h1->next;
h2=h2->next;
h3=h3->next;
 }
return(p);
}
node * multiply(node *h1, node *h2)
{
node *h3,*p,*q,*r;
int i,j,k,coeff,power;
h3=init();
for(p=h1,i=0;p!=NULL;p=p->next,i++)
for(q=h2,j=0;q!=NULL;q=q->next,j++)
 {
coeff=p->coeff * q->coeff;
power=i+j;
for(r=h3,k=0;k<power;k++)
r=r->next;
r->coeff=r->coeff+coeff;
 }
 return(h3);
}
node * init()
{
 int i;
 node *h=NULL,*p;
 for(i=0;i<MAX;i++)
{
p=(node*)malloc(sizeof(node));
p->next=h;
p->coeff=0;
h=p;
}}
```

19BCE0829
KANDRA KSHEERAJ

PROGRAM OUTPUT:

```
 Program to implement polynomial addition and multiplication using LL

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

 1.create 1st polynomial
 2.create 2nd polynomial
 3.Add polynomials
 4.Multiply polynomials
 5.Quit
 Enter your choice :1

Enter number of terms :3
Enter a term(power & coeff)0
2
Enter a term(power & coeff)1
2
Enter a term(power & coeff)2
2

 1.create 1st polynomial
 2.create 2nd polynomial
 3.Add polynomials
 4.Multiply polynomials
 5.Quit
 Enter your choice :2

Enter number of terms :3
Enter a term(power & coeff)0
3
Enter a term(power & coeff)1
3
Enter a term(power & coeff)2
3
```

```
 1.create 1st polynomial
 2.create 2nd polynomial
 3.Add polynomials
 4.Multiply polynomials
 5.Quit
 Enter your choice :3

 1st polynomial -> 2X^0 2X^1 2X^2
 2nd polynomial -> 3X^0 3X^1 3X^2
 Sum = 5X^0 5X^1 5X^2
```

```
1.create 1st polynomial
2.create 2nd polynomial
3.Add polynomials
4.Multiply polynomials
5.Quit
Enter your choice :4

1st polynomial -> 2X^0 2X^1 2X^2
2nd polynomial -> 3X^0 3X^1 3X^2
product = 6X^0 12X^1 18X^2 12X^3 6X^4
```

19BCE0829
KANDRA KSHEERAJ

QUESTION 4:

4. Menu driven C program to create binary tree and to perform preorder, inorder and postorder traversal.

PSEUDO CODE:

```
struct node {
  int data;
  struct node* left;
  struct node* right;
};
void inorder(struct node* root){
  if(root == NULL) return;
  inorder(root->left);
  print root->data;
  inorder(root->right);
}
void preorder(struct node* root){
  if(root == NULL) return;
  print root->data;
  preorder(root->left);
  preorder(root->right);
}
void postorder(struct node* root) {
  if(root == NULL) return;
  postorder(root->left);
  postorder(root->right);
  print root->data;
}
struct node* createNode(value){
  struct node* newNode = malloc(sizeof(struct node));
  newNode->data = value;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
struct node* insertLeft(struct node *root, int value) {
  root->left = createNode(value);
  return root->left;
```

```
}
struct node* insertRight(struct node *root, int value){
  root->right = createNode(value);
  return root->right;
}
int main(){
     define n[10],i,j,choice;
     while (1){
     read input choice;
     switch(choice){

     case 1:
     /*number of nodes*/
     read input i;
     for( j = 1; j <=i ; j++) {
        read input n[j];
        }
     struct node* root = createNode(n[1]);
     insertLeft(root, n[2]);
     insertRight(root, n[3]);
     insertLeft(root->left, n[4]);
     insertRight(root->left, n[5]);

     case 2:
     /*inorder traversal */
     inorder(root);

     case 3:
     /* Preorder traversal */
     preorder(root);

     case 4:
     /* Postorder traversal */
     postorder(root);

     case 5:
        exit(0);
     default:
 }}}
```

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
 int data;
 struct node* left;
 struct node* right;
};
void inorder(struct node* root){
 if(root == NULL) return;
 inorder(root->left);
 printf("%d ->", root->data);
 inorder(root->right);
}
void preorder(struct node* root){
 if(root == NULL) return;
 printf("%d ->", root->data);
 preorder(root->left);
 preorder(root->right);
}
void postorder(struct node* root) {
 if(root == NULL) return;
 postorder(root->left);
 postorder(root->right);
 printf("%d ->", root->data);
}
struct node* createNode(value){
 struct node* newNode = malloc(sizeof(struct node));
 newNode->data = value;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
struct node* insertLeft(struct node *root, int value) {
 root->left = createNode(value);
 return root->left;
}
struct node* insertRight(struct node *root, int value){
 root->right = createNode(value);
```

```
 return root->right;
}
int main(){
                  printf("\nProgram to create Binary Tree and
to perform preorder inorder and postorder traversal \n");
     printf("\n----------------------------------------\n");
                int n[10],i,j,choice;
                while (1)
                {
                printf("\n\n1.Create\n");
                printf("2.Inorder Traversal\n");
                printf("3.Preorder Traversal\n");
    printf("4.Postorder Traversal\n");
                printf("5.Exit\n");
                printf("Enter choice : ");
                scanf("%d", &choice);
                switch(choice)
   {
            case 1:
            printf("Enter The Number of Nodes In The Tree: ");
            scanf("%d", &i);
                for( j = 1; j <=i ; j++)
                {
                printf("Enter The Node values: ");
     scanf("%d", &n[j]);
    }
                struct node* root = createNode(n[1]);
                insertLeft(root, n[2]);
                insertRight(root, n[3]);
                insertLeft(root->left, n[4]);
                insertRight(root->left, n[5]);
                break;
                case 2:
                printf("Inorder traversal \n");
                inorder(root);
                break;
                case 3:
                printf("\nPreorder traversal \n");
                preorder(root);
```

```
                    break;
                    case 4:
                    printf("\nPostorder traversal \n");
                    postorder(root);
                    break;
                    case 5:
        exit(0);
        break;
                    default:
                    printf("Wrong choice, Please enter correct
    choice ");
                    break;
     }
    }
    }
```

PROGRAM OUTPUT:

```
Program to create Binary Tree and to perform preorder inorder and postorder traversal

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


1.Create Binary Tree
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter choice : 1
Enter The Number of Nodes In The Tree: 5
Enter The Node values: 1
Enter The Node values: 12
Enter The Node values: 9
Enter The Node values: 5
Enter The Node values: 6


1.Create Binary Tree
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter choice : 2
Inorder traversal
5 ->12 ->6 ->1 ->9 ->
```

```
1.Create Binary Tree
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter choice : 3

Preorder traversal
1 ->12 ->5 ->6 ->9 ->

1.Create Binary Tree
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter choice : 4

Postorder traversal
5 ->6 ->12 ->9 ->1 ->
```

QUESTION 5:

5. Menu driven C program to create binary search tree. Perform insertion and deletion operations. Display the contents of BST using preorder, inorder and postorder traversal.

PSEUDO CODE:

```
struct node
{
     int info;
     struct node *lchild;
     struct node *rchild;
}*root;
void find(int item,struct node **par,struct node **loc)
{
     struct node *ptr,*ptrsave;
     if(root==NULL)  /*tree empty*/
     {
          *loc=NULL;
          *par=NULL;
          return;
     }
     if(item==root->info) /*item is at root*/
     {
```

```
        *loc=root;
        *par=NULL;
        return;
    }
    /*Initialize ptr and ptrsave*/
    if(item<root->info)
        ptr=root->lchild;
    else
        ptr=root->rchild;
    ptrsave=root;

    while(ptr!=NULL)
    {
        if(item==ptr->info)
        {       *loc=ptr;
            *par=ptrsave;
            return;
        }
        ptrsave=ptr;
        if(item<ptr->info)
            ptr=ptr->lchild;
        else
            ptr=ptr->rchild;
     }
     *loc=NULL;   /*item not found*/
     *par=ptrsave;
}

void insert(int item)
{       struct node *tmp,*parent,*location;
    find(item,&parent,&location);
    if(location!=NULL)
    {
        print "Item already present";
    }

    tmp=(struct node *)malloc(sizeof(struct node));
    tmp->info=item;
    tmp->lchild=NULL;
```

```
     tmp->rchild=NULL;

     if(parent==NULL)
          root=tmp;
     else
          if(item<parent->info)
               parent->lchild=tmp;
          else
               parent->rchild=tmp;
}


void case_a(struct node *par,struct node *loc )
{
     if(par==NULL) /*item to be deleted is root node*/
          root=NULL;
     else
          if(loc==par->lchild)
               par->lchild=NULL;
          else
               par->rchild=NULL;
}

void case_b(struct node *par,struct node *loc)
{
     struct node *child;

     /*Initialize child*/
     if(loc->lchild!=NULL) /*item to be deleted has lchild */
          child=loc->lchild;
     else                 /*item to be deleted has rchild */
          child=loc->rchild;

     if(par==NULL )   /*Item to be deleted is root node*/
          root=child;
     else
          if( loc==par->lchild)/*item is lchild of its parent*/
               par->lchild=child;
```

```
          else                    /*item is rchild of its parent*/
               par->rchild=child;
}

void case_c(struct node *par,struct node *loc)
{
     struct node *ptr,*ptrsave,*suc,*parsuc;

     /*Find inorder successor and its parent*/
     ptrsave=loc;
     ptr=loc->rchild;
     while(ptr->lchild!=NULL)
     {
          ptrsave=ptr;
          ptr=ptr->lchild;
     }
     suc=ptr;
     parsuc=ptrsave;

     if(suc->lchild==NULL && suc->rchild==NULL)
          case_a(parsuc,suc);
     else
          case_b(parsuc,suc);

     if(par==NULL) /*if item to be deleted is root node */
          root=suc;
     else
          if(loc==par->lchild)
               par->lchild=suc;
          else
               par->rchild=suc;

     suc->lchild=loc->lchild;
     suc->rchild=loc->rchild;
}/*End of case_c()*/
int del(int item)
{
     struct node *parent,*location;
     if(root==NULL)
```

```
          Print "Tree empty";


     find(item,&parent,&location);
     if(location==NULL)
          Print "Item not present in tree";

     if(location->lchild==NULL && location->rchild==NULL)
          case_a(parent,location);
     if(location->lchild!=NULL && location->rchild==NULL)
          case_b(parent,location);
     if(location->lchild==NULL && location->rchild!=NULL)
          case_b(parent,location);
     if(location->lchild!=NULL && location->rchild!=NULL)
          case_c(parent,location);
     free(location);
}

int preorder(struct node *ptr)
{
     if(root==NULL)
     {
          print "Tree is empty";
          return 0;
     }
     if(ptr!=NULL)
     {
          print ptr->info;
          preorder(ptr->lchild);
          preorder(ptr->rchild);
     }
}

void inorder(struct node *ptr)
{
     if(root==NULL)
     {
          print "Tree is empty";
          return;
```

```
     }
     if(ptr!=NULL)
     {
          inorder(ptr->lchild);
          print ptr->info;
          inorder(ptr->rchild);
     }
}

void postorder(struct node *ptr)
{
     if(root==NULL)
     {
          print "Tree is empty";
          return;
     }
     if(ptr!=NULL)
     {
          postorder(ptr->lchild);
          postorder(ptr->rchild);
          print ptr->info ;
     }
}

void display(struct node *ptr,int level)
{
     define i;
     if ( ptr!=NULL )
     {
          display(ptr->rchild, level+1);
          for (i = 0; i < level; i++)
          print ptr->info;
          display(ptr->lchild, level+1);
     }
}
main()
{
     define choice,num;
     root=NULL;
```

```
    while(1)
    {
        read input choice;

        switch(choice)
        {
         case 1:
            /*number to be inserted :*/
            read input num;
            insert(num);

         case 2:
            /*number to be deleted*/
            read input num;
            del(num);

         case 3:
            inorder(root);

         case 4:
            preorder(root);

         case 5:
            postorder(root);

         case 6:
            display(root,1);

         case 7:
         default:
            print "Wrong choice";
        }
    }
}
```

PROGRAM CODE:

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

```
struct node
{
                int info;
                struct node *lchild;
                struct node *rchild;
}*root;


void find(int item,struct node **par,struct node **loc)
{
                struct node *ptr,*ptrsave;

                if(root==NULL)
                {
                  *loc=NULL;
                  *par=NULL;
                  return;
                }
                if(item==root->info)
                {
                  *loc=root;
                  *par=NULL;
                  return;
                }
                if(item<root->info)
                  ptr=root->lchild;
                else
                  ptr=root->rchild;
                ptrsave=root;

                while(ptr!=NULL)
                {
                  if(item==ptr->info)
                  {       *loc=ptr;
                       *par=ptrsave;
                       return;
                  }
                  ptrsave=ptr;
                  if(item<ptr->info)
```

```
                    ptr=ptr->lchild;
                  else
                    ptr=ptr->rchild;
                  }
                  *loc=NULL;
                  *par=ptrsave;
}

void insert(int item)
{       struct node *tmp,*parent,*location;
                find(item,&parent,&location);
                if(location!=NULL)
                {
                  printf("Item already present");
                  return;
                }

                tmp=(struct node *)malloc(sizeof(struct
node));
                tmp->info=item;
                tmp->lchild=NULL;
                tmp->rchild=NULL;

                if(parent==NULL)
                  root=tmp;
                else
                  if(item<parent->info)
                    parent->lchild=tmp;
                  else
                    parent->rchild=tmp;
}


void case_a(struct node *par,struct node *loc )
{
                if(par==NULL)
                  root=NULL;
                else
                  if(loc==par->lchild)
```

```
                    par->lchild=NULL;
                else
                    par->rchild=NULL;
}

void case_b(struct node *par,struct node *loc)
{
                struct node *child;


                if(loc->lchild!=NULL)
                  child=loc->lchild;
                else
                  child=loc->rchild;

                if(par==NULL )
                  root=child;
                else
                  if( loc==par->lchild)
                      par->lchild=child;
                  else
                      par->rchild=child;
}
void case_c(struct node *par,struct node *loc)
{
                struct node *ptr,*ptrsave,*suc,*parsuc;

                ptrsave=loc;
                ptr=loc->rchild;
                while(ptr->lchild!=NULL)
                {
                  ptrsave=ptr;
                  ptr=ptr->lchild;
                }
                suc=ptr;
                parsuc=ptrsave;

                if(suc->lchild==NULL && suc->rchild==NULL)
                  case_a(parsuc,suc);
```

```
                else
                  case_b(parsuc,suc);

                if(par==NULL)
                  root=suc;
                else
                  if(loc==par->lchild)
                       par->lchild=suc;
                  else
                       par->rchild=suc;

                suc->lchild=loc->lchild;
                suc->rchild=loc->rchild;
}
int del(int item)
{
                struct node *parent,*location;
                if(root==NULL)
                {
                  printf("Tree empty");
                  return 0;
                }

                find(item,&parent,&location);
                if(location==NULL)
                {
                  printf("Item not present in tree");
                  return 0;
                }

if(location->lchild==NULL && location->rchild==NULL)
                  case_a(parent,location);
if(location->lchild!=NULL && location->rchild==NULL)
                  case_b(parent,location);
if(location->lchild==NULL && location->rchild!=NULL)
                  case_b(parent,location);
if(location->lchild!=NULL && location->rchild!=NULL)
                  case_c(parent,location);
                free(location);
```

```
}

int preorder(struct node *ptr)
{
                if(root==NULL)
                {
                  printf("Tree is empty");
                  return 0;
                }
                if(ptr!=NULL)
                {
                  printf("%d  ",ptr->info);
                  preorder(ptr->lchild);
                  preorder(ptr->rchild);
                }
}

void inorder(struct node *ptr)
{
                if(root==NULL)
                {
                  printf("Tree is empty");
                  return;
                }
                if(ptr!=NULL)
                {
                  inorder(ptr->lchild);
                  printf("%d  ",ptr->info);
                  inorder(ptr->rchild);
                }
}

void postorder(struct node *ptr)
{
                if(root==NULL)
                {
                  printf("Tree is empty");
                  return;
                }
```

```
                if(ptr!=NULL)
                {
                  postorder(ptr->lchild);
                  postorder(ptr->rchild);
                  printf("%d  ",ptr->info);
                }
}

void display(struct node *ptr,int level)
{
                int i;
                if ( ptr!=NULL )
                {
                  display(ptr->rchild, level+1);
                  printf("\n");
                  for (i = 0; i < level; i++)
                       printf("    ");
                  printf("%d", ptr->info);
                  display(ptr->lchild, level+1);
                }
}
main()
{
                int choice,num;
                root=NULL;
                printf("\n Program to create BST with
insertion & deletion operations, preorder inorder postorder
traversal \n");
    printf("\n-----------------------------------------\n");
while(1)
{
printf("\n\n1.Insert a Node \n");
printf("2.Delete a Node From The Tree\n");
printf("3.Inorder Traversal\n");
printf("4.Preorder Traversal\n");
printf("5.Postorder Traversal\n");
printf("6.Display\n");
printf("7.Exit\n");
printf("Enter your choice : ");
```

```
    scanf("%d",&choice);

    switch(choice)
    {
    case 1:
          printf("Enter the number to be inserted : ");
          scanf("%d",&num);
          insert(num);
          break;
    case 2:
          printf("Enter the number to be deleted : ");
          scanf("%d",&num);
          del(num);
          break;
    case 3:
          inorder(root);
          break;
    case 4:
          preorder(root);
          break;
    case 5:
          postorder(root);
          break;
    case 6:
          display(root,1);
          break;
    case 7:
          break;
    default:
          printf("Wrong choice\n");
           }
        }
    }
```

PROGRAM OUTPUT:

19BCE0829
KANDRA KSHEERAJ

```
 Program to create BST with insertion & deletion operations, preorder inorder postorder traversal

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


1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 1
Enter the number to be inserted : 70


1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 1
Enter the number to be inserted : 67


1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 1
Enter the number to be inserted : 85
```

19BCE0829
KANDRA KSHEERAJ

```
1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 3
67  70  85

1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 4
70  67  85

1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 5
67  85  70

1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 2
Enter the number to be deleted : 67
```

```
1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 3
70  85

1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 4
70  85

1.Insert a Node
2.Delete a Node From The Tree
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Exit
Enter your choice : 5
85  70
```
