DEPARTMENT OF INFORMATION TECHNOLOGY

II B Tech – I Semester

DATA STRUCTURES LAB

ANURAG UNIVERSITY

B.Tech. IT – II Year I Sem.	L	T / P / D	C
	0	**4**	**2**

DATA STRUCTURES LAB

Prerequisites: Any programming language and a parallel course on data structures.

Course Outcomes:

At the end of this Data Structures Lab course, students will be able to:

1. Develop the programs on stacks and its applications.
2. Demonstrate the operations on Trees.
3. Code the implementation of various advanced trees.
4. Design and implementation of programs on BST and Graph Traversals.
5. Develop the programs on Hashing and Dictionaries

List of Experiments:

Week 1:

1.Review of Stack and Queue Operations using arrays and Linked Lists

Week 2:

2. Program to convert infix to postfix notation

3. Program to evaluate postfix notations

Week 3:

4.Program to implement towers of Hanoi

5.Program to implement parenthesis checker

Week 4:

6.Program to illustrate tree traversals
a) In order b) Preorder c) Post order

Week 5:

7.Program to illustrate insertion, deletion and searching in Binary Search Tree.

Week 6:

8.Program to implement Heaps

a) Min Heap b) Max Heap

Week 7:

9.Program to illustrate Insertion on AVL Trees.

10.Program to illustrate deletion and Rotation on AVL Trees.

Week 8:

11.Program to implement B-Trees

a) Insertion b) Search c) Display

Week 9:

12.Program to illustrate Graph traversals

a. Breadth First Search
b. Depth First Search

Week 10:

13.Program to implement

a) Prim's algorithm b) Kruskal's algorithm

Week 11:

14.Program to Implement Dijkstra algorithm.

Week 12 & 13:

15.Program to implement Hashing and collision resolution techniques

Week 14:

16.Program to implement Dictionaries.

Week 15:

Review

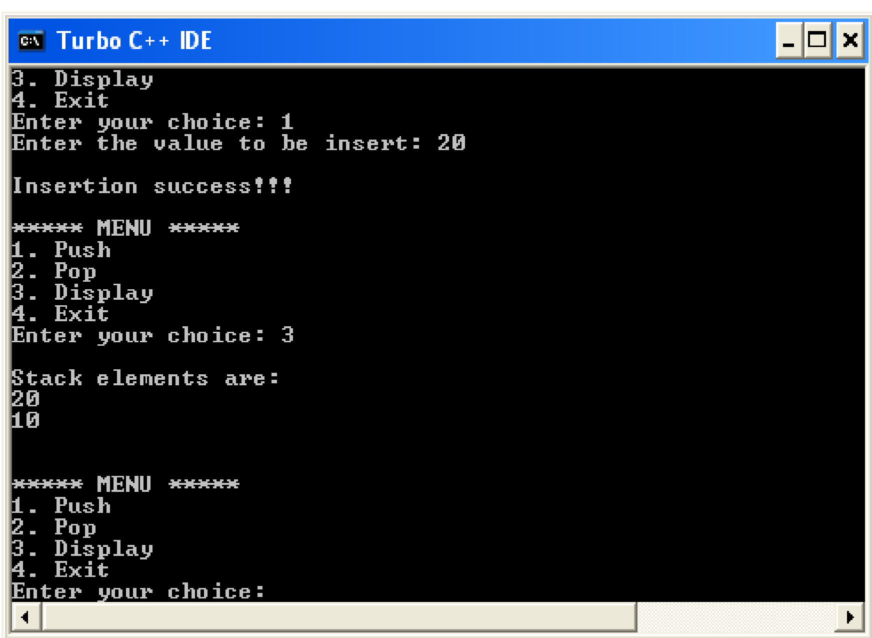
INDEX

1.**Review of Stack and Queue Operations using arrays and Linked Lists**

Stacks Using Arrays:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10

void push(int);
void pop();
void display();

int stack[SIZE], top = -1;

void main()
{
int value, choice;
clrscr();
while(1){
	printf("\n\n***** MENU *****\n");
	printf("1. Push\n2. Pop\n3. Display\n4. Exit");
	printf("\nEnter your choice: ");
	scanf("%d",&choice);
	switch(choice){
	case 1: printf("Enter the value to be insert: ");
		scanf("%d",&value);
		push(value);
		break;
	case 2: pop();
		break;
	case 3: display();
		break;
	case 4: exit(0);
	default: printf("\nWrong selection!!! Try again!!!");
	}
}
}
void push(int value){
if(top == SIZE-1)
	printf("\nStack is Full!!! Insertion is not possible!!!");
```

```
else{
        top++;
        stack[top] = value;
        printf("\nInsertion success!!!");
}
}
void pop(){
if(top == -1)
        printf("\nStack is Empty!!! Deletion is not possible!!!");
else{
        printf("\nDeleted : %d", stack[top]);
        top--;
}
}
void display(){
if(top == -1)
        printf("\nStack is Empty!!!");
else{
        int i;
        printf("\nStack elements are:\n");
        for(i=top; i>=0; i--)
        printf("%d\n",stack[i]);
}
}
```



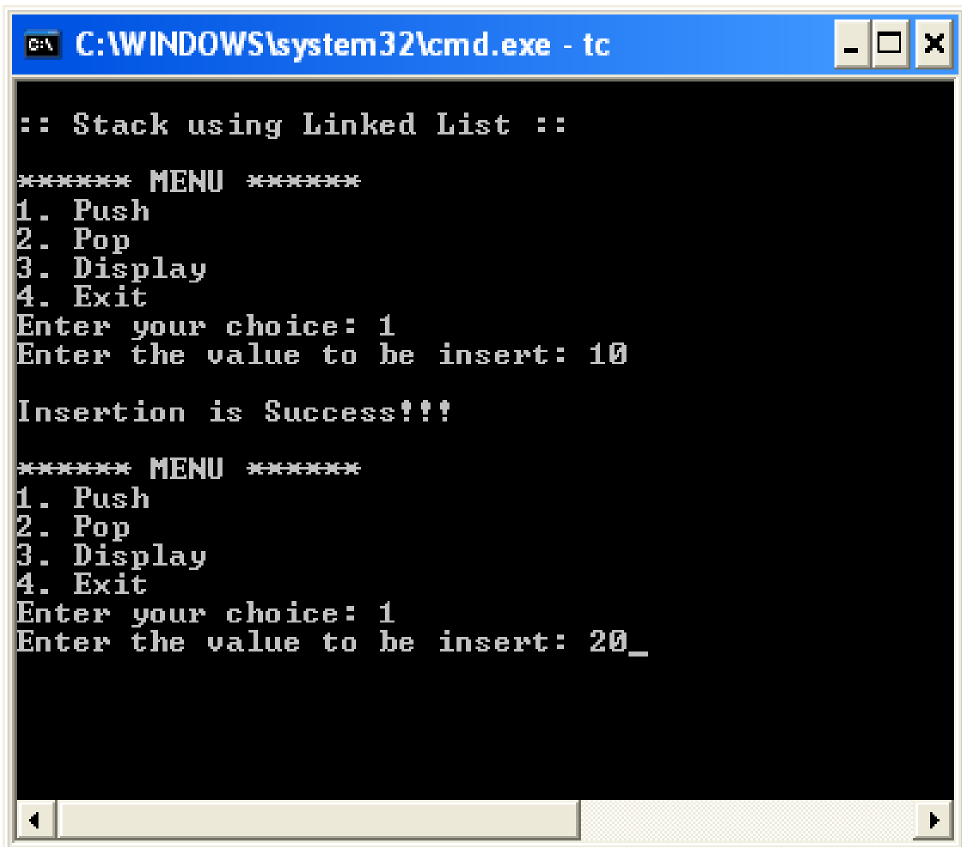
Stacks Using Linked Lists:

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

struct Node
{
  int data;
  struct Node *next;
}
*top = NULL;

void push(int);
void pop();
void display();

void main()
{
  int choice, value;
  clrscr();
  printf("\n:: Stack using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
          case 1: printf("Enter the value to be insert: ");
                    scanf("%d", &value);
                    push(value);
                    break;
          case 2: pop(); break;
          case 3: display(); break;
```

```
        case 4: exit(0);
        default: printf("\nWrong selection!!! Please try again!!!\n");
     }
   }
}
void push(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  if(top == NULL)
    newNode->next = NULL;
  else
    newNode->next = top;
  top = newNode;
  printf("\nInsertion is Success!!!\n");
}
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element: %d", temp->data);
    top = temp->next;
    free(temp);
  }
}
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
```

```
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
          printf("%d--->",temp->data);
          temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }
}
```

Output



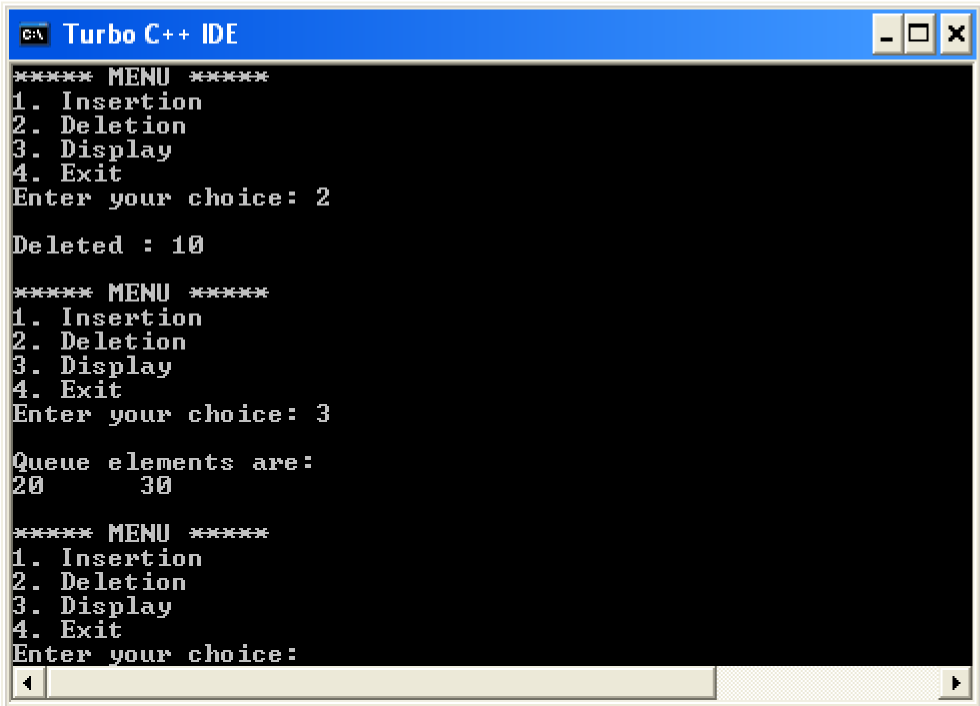
Queues Using Arrays:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10

void enQueue(int);
void deQueue();
void display();

int queue[SIZE], front = -1, rear = -1;

void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
{
          case 1: printf("Enter the value to be insert: ");
                    scanf("%d",&value);
                    enQueue(value);
                    break;
          case 2: deQueue();
                    break;
          case 3: display();
                    break;
          case 4: exit(0);
          default: printf("\nWrong selection!!! Try again!!!");
```

```
      }
    }
}
void enQueue(int value){
   if(rear == SIZE-1)
      printf("\nQueue is Full!!! Insertion is not possible!!!");
   else{
      if(front == -1)
            front = 0;
      rear++;
      queue[rear] = value;
      printf("\nInsertion success!!!");
   }
}
void deQueue(){
   if(front == rear)
      printf("\nQueue is Empty!!! Deletion is not possible!!!");
   else{
      printf("\nDeleted : %d", queue[front]);
      front++;
      if(front == rear)
            front = rear = -1;
   }
}
void display(){
   if(rear == -1)
      printf("\nQueue is Empty!!!");
   else{
      int i;
      printf("\nQueue elements are:\n");
      for(i=front; i<=rear; i++)
            printf("%d\t",queue[i]);
```

```
  }
}
```

Output



```
Turbo C++ IDE
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 10

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
20      30

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice:
```

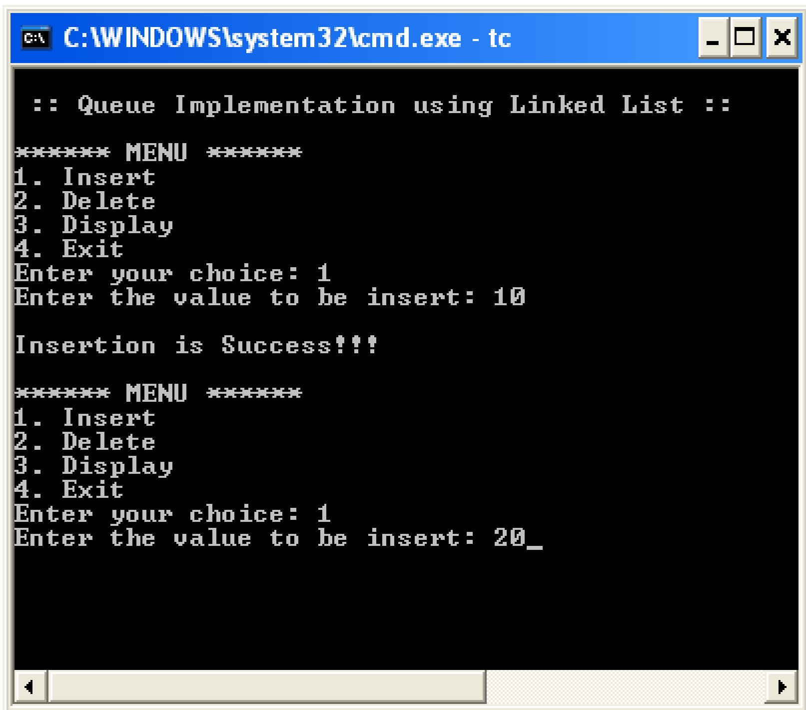
Queues Using Linked Lists:

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

struct Node
{
   int data;
   struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();

void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Queue Implementation using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
            case 1: printf("Enter the value to be insert: ");
                     scanf("%d", &value);
                     insert(value);
                     break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: exit(0);
```

```
            default: printf("\nWrong selection!!! Please try again!!!\n");
        }
    }
}
void insert(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode -> next = NULL;
    if(front == NULL)
        front = rear = newNode;
    else{
        rear -> next = newNode;
        rear = newNode;
    }
    printf("\nInsertion is Success!!!\n");
}
void delete()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        front = front -> next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
}
void display()
{
    if(front == NULL)
```

```
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      while(temp->next != NULL){
            printf("%d--->",temp->data);
            temp = temp -> next;
      }
      printf("%d--->NULL\n",temp->data);
   }
}
```

Output



2. Program to convert infix to postfix notation

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#define size 50
char s[size];
int top=-1;
push(char elem)
{
return  s[++top]=elem;
}
char pop()
{
return(s[top--]);
}
int pr(char elem)
{
switch(elem)
{
case '#':
return 0;
case '(':
return 1;
case '+':
case '-':
return 2;
case '*':
case '/':

return 3;

}

return ;

}
```

```
void main()
{
char infx[50],pofx[50],ch,elem;
int i=0,k=0;
printf("\n\n read the infix expression?");
scanf("%s", infx);
push('#');
while ((ch=infx[i++])!='\0')
{
if(ch=='(')
push(ch);
else if(isalnum(ch))
pofx[k++]=ch;
else if (ch==')')
{
while(s[top]!='(')
pofx[k++]=pop();
elem=pop();
}
else
{
while(pr(s[top])>=pr(ch))
pofx[k++]=pop();
elem=pop();
}
}
while(s[top]!='#')
pofx[k++]=pop();
pofx[k]='\0';
printf("\n\n Given infix expn: %s \npostfix expn:%s\n", infx,pofx);

}
```

Output:

```
read the infix expression?((a*b)+c)


Given infix expn: ((a*b)+c)
postfix expn:ab*c+
```

3. Program to evaluate postfix notation

```
#include<conio.h>
#include<string.h>
#define MAX 50
int stack[MAX];
char post[MAX];
int top=-1;
void pushstack(int tmp);
void calculator(char c);
void main()
{
int i;
clrscr();
printf("insert a postfix notation::");
gets(post);
for(i=0;i<strlen(post);i++)
{
if(post[i]>='0' && post[i]<='9')
{
pushstack(i);
}
if(post[i]== '+' || post[i]== '-' || post[i]== '*' || post[i]== '/' || post[i]== '^')
{
calculator(post[i]);
}
printf("\n\nResult::%d",stack[top]);
getch();
}
}
void pushstack(int tmp)
{
top++;
```

```
stack[top]=(int) (post[tmp]-48);
}
void calculator(char c)
{
int a,b,ans;
a=stack[top];
stack[top]= '\0';
top--;
b=stack[top];
stack[top]= '\0';
top--;
switch(c)
{
case '+':
ans=b+a;
break;
case '-':
ans=b-a;
break;
case  '*':
ans=b*a;
break;
case  '/':
ans= b/a;
break;
case  '^':
ans=b^a;
break;
default:
ans=0;
}
top++;
stack[top]=ans;
}
```

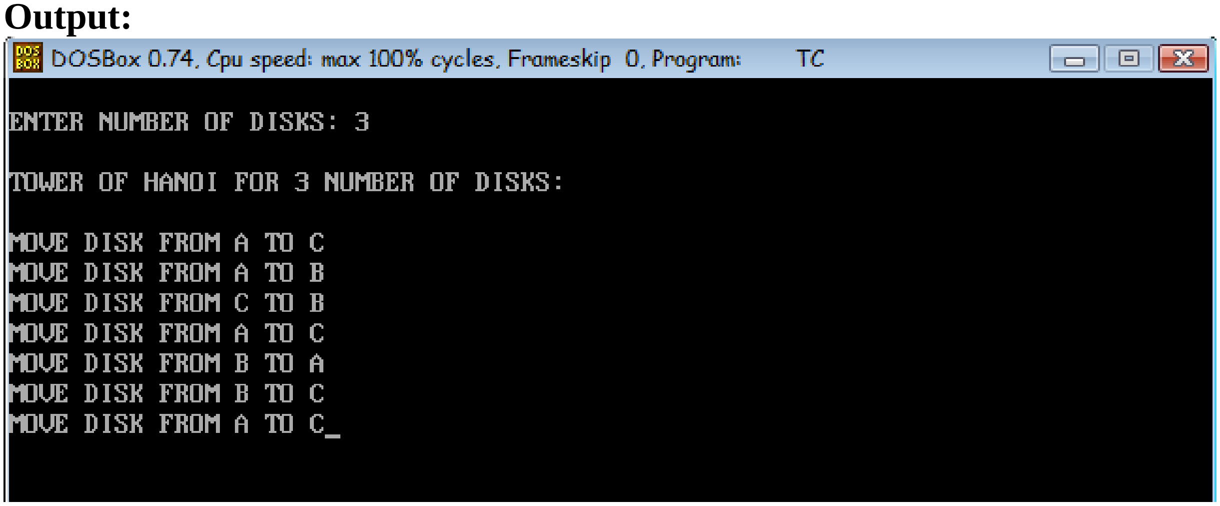
Output:

```
inser a postfix notation::257*14-6+

Result::3_
```

4. Program to implement Towers of Hanoi

```
#include <stdio.h>
#include <conio.h>
void hanoi(char,char,char,int);
void main()
{
	int num;
	clrscr();
	printf("\nENTER NUMBER OF DISKS: ");
	scanf("%d",&num);
	printf("\nTOWER OF HANOI FOR %d NUMBER OF DISKS:\n", num);
	hanoi('A','B','C',num);
	getch();
}
void hanoi(char from,char to,char other,int n)
{
	if(n<=0)
		printf("\nILLEGAL NUMBER OF DISKS");
	if(n==1)
		printf("\nMOVE DISK FROM %c TO %c",from,other);
	if(n>1)
	{
		hanoi(from,other,to,n-1);
		hanoi(from,to,other,1);
		hanoi(to,from,other,n-1);
	}
}
```

Output:

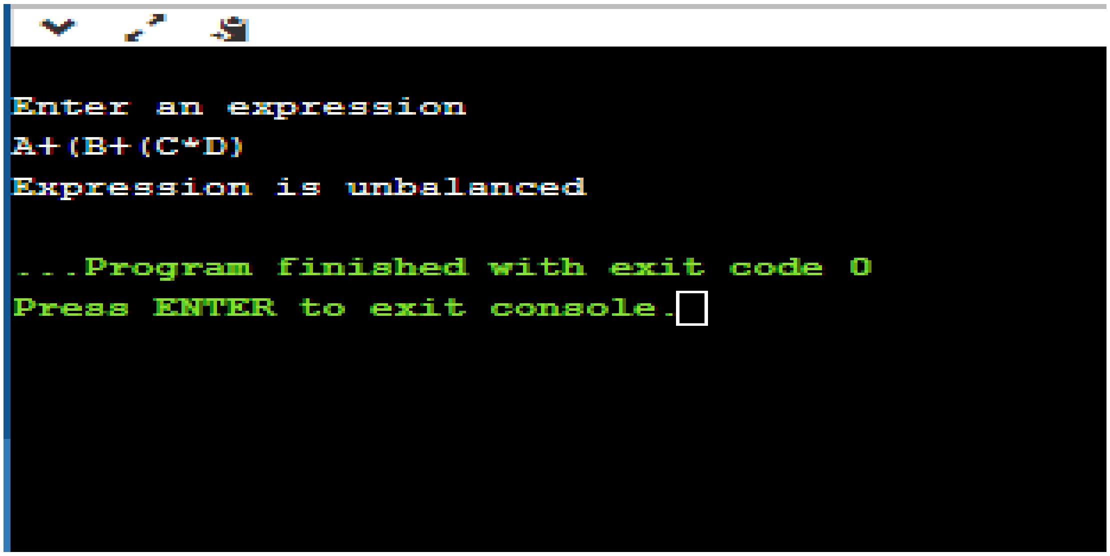


5. Program to implement Parenthesis Checker

```
#include<stdio.h>
int main ()
{
    char expression [50];
    int x=0, i=0;
    printf("\nEnter an expression");
    scanf("%s", expression);
    while(expression[i]!= '\0')
    {
     if(expression[i]=='(')
        {
            x++;
        }

     else if(expression[i]==')')
        {
            x--;
            if(x<0)
            break;
        }
    i++;
    }

    if(x==0)
    {
        printf("Expression is balanced");
    }

    else
    {
        printf("Expression is unbalanced");
    }
```

```
    return 0;
}
```

Output:



6.Program to illustrate tree traversals

a) **In order**
b) **Pre order**
c) **post order**
d) **insertion**
e) **deletion**
f) **search Tree**

```
#include<stdio.h>
#include<conio.h>
struct bstnode
{
   int data;
  struct bstnode *lc,*rc;
}*root;
void insert(int);
void deletele(int);
int search(int);
int menu();
void inorder(struct bstnode *);
void preorder(struct bstnode *);
void postorder(struct bstnode *);
void main()
{
int choice,ele;
root=NULL;
clrscr();
do{
   choice=menu();
   switch(choice)
   {
     case 1:printf("\n enter element to be inserted");
              scanf("%d",&ele);
              insert(ele);
```

```
                break;
     case 2:printf("enter element to be deleted");
              deletele(ele);
                   break;
     case 3:printf("\n enter search element");
              scanf("%d",&ele);
              if(search(ele)==1)
                printf("found");
              else
                 printf("not found");
                 break;
     case 4:inorder(root);
                 break;
     case 5:preorder(root);
                 break;
     case 6:postorder(root);
                 break;
   }
   } while(choice!=0);
}

int menu()
{
int opt;
printf("\n option purpose");
printf("\n 1.insert \n 2.delete \n 4.inorder \n 5,preorder \n 6.postorder \n0.exit");
printf("\n enter your option");
scanf("%d",&opt);
return(opt);
}
void insert(int ele)
{
struct bstnode *temp,*t,*par;
temp=(struct bstnode *)malloc(sizeof(struct bstnode));
temp->data=ele;
```

```
temp->lc=NULL;
temp->rc=NULL;
if(root==NULL)
{ root=temp;
  return;
}
par=NULL;
t=root;
 while(t!=NULL)
 {
  par=t;
   if(ele<t->data)
      t=t->lc;
   else
      t=t->rc;
 }
 if(ele<par->data)
    par->lc=temp;
 else
    par->rc=temp;
}
void deletele(int ele)
{
 struct bstnode *t,*par,*insucpar,*insuc;
 if(search(ele)==1)
 { if((t->rc=NULL)&&(t->lc==NULL))/*leaf node*/
    { if(par->lc==t)
           par->lc=NULL;
      else
           par->rc=NULL;
    }
 else
      if(((t->rc!=NULL)&&(t->lc==NULL))||((t->rc==NULL)&&(t->lc!=NULL)))
      {  /* only one child*/
          if(t->rc!=NULL)
           { if(par->lc==t)
                   par->lc=t->rc;
             else
                   par->rc=t->rc;
           }
```

```
          else
           { if(par->lc==t)
                    par->lc=t->lc;
             else
                    par->rc=t->lc;
           }
      }
 else
     if((t->rc!=NULL)&&(t->lc!=NULL))
     { /* 2 children*/
       insucpar=t;
       insuc=t->rc;
       while(insuc->lc!=NULL)
              {
                insucpar=insuc;
                insuc=insuc->lc;
              }
          if(insuc->rc!=NULL) /*right child exists*/
                insucpar->lc=insuc->rc;
          else
                insucpar->lc=NULL;
          insuc->lc=t->lc;
          insuc->rc=t->rc;
          if(par!=NULL) /*not root*/
           {  if(par->lc==t)
                     par->lc=insuc;
              else
                     par->rc=insuc;
           }
          else /*node being root itself*/
            root=insuc;
      }
     }
     else
       printf("elements not found and no children");

    }
int search(int ele)
{
  struct bstnode *t;
```

```
   t=root;
   while(t!=NULL)
    { if(t->data==ele)
           return(1);
      else
           {if(ele>t->data)
             t=t->rc;
           else
             t=t->lc; }
     }
    return(0);
  }
void inorder(struct bstnode *t)
{
  if(t!=NULL)
   { inorder(t->lc);
     printf("\n %d",t->data);
     inorder(t->rc);
   }
}

void preorder(struct bstnode *t)
{
  if(t!=NULL)
  { printf("\n %d",t->data);
    preorder(t->lc);
    preorder(t->rc);
  }
}
void postorder(struct bstnode *t)
{
  if(t!=NULL)
  {  postorder(t->lc);
     postorder(t->rc);
     printf("\n %d",t->data);
  }
}
```

Output:

```
 enter your option1

 enter element to be inserted7

 option purpose
 1.insert
 2.delete
 4.inorder
 5,preorder
 6.postorder
0.exit
 enter your option4

 6
 7
 8
 9
 option purpose
 1.insert
 2.delete
 4.inorder
 5,preorder
 6.postorder
0.exit
 enter your option_
```

```
 6
 7
 8
 9
 option purpose
 1.insert
 2.delete
 4.inorder
 5,preorder
 6.postorder
0.exit
 enter your option5

 9
 6
 8
 7
 option purpose
 1.insert
 2.delete
 4.inorder
 5,preorder
 6.postorder
0.exit
 enter your option_
```

```
 9
 6
 8
 7
 option purpose
 1.insert
 2.delete
 4.inorder
 5,preorder
 6.postorder
0.exit
 enter your option6

 7
 8
 6
 9
 option purpose
 1.insert
 2.delete
 4.inorder
 5,preorder
 6.postorder
0.exit
 enter your option
```

7.Program to illustrate insertion deletion searching in Binary search tree

```
#include <stdio.h>
#include <stdlib.h>
 struct treeNode {
       int data;
       struct treeNode *left, *right;
 };
 struct treeNode *root = NULL;
 /* create a new node with the given data */
 struct treeNode* createNode(int data) {
       struct treeNode *newNode;
       newNode = (struct treeNode *) malloc(sizeof (struct treeNode));
       newNode->data = data;
       newNode->left = NULL;
       newNode->right = NULL;
       return(newNode);
 }
 /* insertion in binary search tree */
 void insertion(struct treeNode **node, int data) {
       if (*node == NULL) {
              *node = createNode(data);
       } else if (data < (*node)->data) {
              insertion(&(*node)->left, data);
       } else if (data > (*node)->data) {
              insertion(&(*node)->right, data);
       }
 }
 /* deletion in binary search tree */
 void deletion(struct treeNode **node, struct treeNode **parent, int data) {
       struct treeNode *tmpNode, *tmpParent;
       if (*node == NULL)
              return;
       if ((*node)->data == data) {
```

```
                /* deleting the leaf node */
                if (!(*node)->left && !(*node)->right) {
                        if (parent) {
                                /* delete leaf node */
                                if ((*parent)->left == *node)
                                        (*parent)->left = NULL;
                                else
        (*parent)->right = NULL;
                                free(*node);
                        } else {
                                /* delete root node with no children */
                                free(*node);
                        }
                /* deleting node with one child */
                } else if (!(*node)->right && (*node)->left) {
                        /* deleting node with left child alone */
                        tmpNode = *node;
                        (*parent)->right = (*node)->left;
                        free(tmpNode);
                        *node = (*parent)->right;
                } else if ((*node)->right && !(*node)->left) {
                        /* deleting node with right child alone */
                        tmpNode = *node;
                        (*parent)->left = (*node)->right;
                        free(tmpNode);
                        (*node) = (*parent)->left;
                } else if (!(*node)->right->left) {
                        /*
                         * deleting a node whose right child
                         * is the smallest node in the right
                         * subtree for the node to be deleted.
                         */
                        tmpNode = *node;

                        (*node)->right->left = (*node)->left;
```

```
                        (*parent)->left = (*node)->right;
                        free(tmpNode);
                        *node = (*parent)->left;
                } else {
                        /*
                         * Deleting a node with two children.
                         * First, find the smallest node in
                         * the right subtree.  Replace the
                         * smallest node with the node to be
                         * deleted. Then, do proper connections
                         * for the children of replaced node.
                         */
                        tmpNode = (*node)->right;
                        while (tmpNode->left) {
                                tmpParent = tmpNode;
                                tmpNode = tmpNode->left;
        }
                        tmpParent->left = tmpNode->right;
                        tmpNode->left = (*node)->left;
                        tmpNode->right =(*node)->right;
                        free(*node);
                        *node = tmpNode;
                }
        } else if (data < (*node)->data) {
                /* traverse towards left subtree */
                deletion(&(*node)->left, node, data);
        } else if (data > (*node)->data) {
                /* traversing towards right subtree */
                deletion(&(*node)->right, node, data);
        }
 }
 /* search the given element in binary search tree */
 void findElement(struct treeNode *node, int data) {
        if (!node)
```

```
            return;
        else if (data < node->data) {
            findElement(node->left, data);
        } else if (data > node->data) {
            findElement(node->right, data);
        } else
            printf("data found: %d\n", node->data);
        return;
    }
    void traverse(struct treeNode *node) {
        if (node != NULL) {
            traverse(node->left);
            printf("%3d", node->data);
            traverse(node->right);
        }
        return;  }
    int main() {
        int data, ch;
        while (1) {
            printf("1. Insertion in Binary Search Tree\n");
            printf("2. Deletion in Binary Search Tree\n");
            printf("3. Search Element in Binary Search Tree\n");
            printf("4. Inorder traversal\n5. Exit\n");
            printf("Enter your choice:");
            scanf("%d", &ch);
            switch (ch) {
                case 1:
                    while (1) {
                    printf("Enter your data:");
                    scanf("%d", &data);
                    insertion(&root, data);
                    printf("Continue Insertion(0/1):");
                    scanf("%d", &ch);
                    if (!ch)
                        break;
```

```
                }
                break;
            case 2:
                printf("Enter your data:");
                scanf("%d", &data);
                deletion(&root, NULL, data);
                break;
            case 3:
                printf("Enter value for data:");
                scanf("%d", &data);
                findElement(root, data);
                break;
            case 4:
                printf("Inorder Traversal:\n");
                traverse(root);
                printf("\n");
                break;
            case 5:
                exit(0);
            default:
                printf("u've entered wrong option\n");
                break;
        }}
    return 0; }
```

Output: (C Program For Insertion, Deletion & Traversal In BST Tree)

```
jp@jp-VirtualBox:$ ./a.out
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:1
Enter your data:20
Continue Insertion(0/1):1
Enter your data:14
Continue Insertion(0/1):1
Enter your data:9
Continue Insertion(0/1):1
```

```
Enter your data:19
Continue Insertion(0/1):1
Enter your data:25
Continue Insertion(0/1):1
Enter your data:21
Continue Insertion(0/1):1
Enter your data:23
Continue Insertion(0/1):1
Enter your data:30
Continue Insertion(0/1):1
Enter your data:26
Continue Insertion(0/1):0
Resultant Binary Search Tree after insertion operation:
                    20
                   /  \
                 14    25
                / \   / \
               9   19 21  30
                       \  /
                       23 26

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
9 14 19 20 21 23 25 26 30
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:2
Enter your data:9

Delete node 9
                    20
                   /  \
                 14    25
                   \   / \
                   19 21  30
                       \  /
                       23 26




1. Insertion in Binary Search Tree
```

```
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:4
 Inorder Traversal:
 19 21 23 25 26
 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:1
 Enter your data:15
 Continue Insertion(0/1):1
 Enter your data:14
 Continue Insertion(0/1):1
 Enter your data:16
 Continue Insertion(0/1):1
 Enter your data:17
 Continue Insertion(0/1):0
 Binary Search Tree After Insertion Operation:

                21
               /  \
             19    25
            /      / \
          15      23  26
         /  \
        14   16
               \
                17



1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:4
 Inorder Traversal:
 14 15 16 17 19 21 23 25 26
 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit

 1. Insertion in Binary Search Tree
```

```
2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:4
 Inorder Traversal:
 14 16 17 19 21 23 25 26
 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:3
 Enter value for data:21
 data found: 21
 1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:5
```

8. Program to Implement Heaps:

a) Min Heap

```
#include<stdio.h>
#include<limits.h>
int heap[1000000], heapSize;
void init()
{
   heapSize = 0;
   heap[0] = -INT_MAX;
}

void Insert(int element)
{
   heapSize++;
   heap[heapSize] = element;
   int now = heapSize;
   while (heap[now / 2] > element) {
      heap[now] = heap[now / 2];
      now /= 2;
   }
   heap[now] = element;
}

int DeleteMin()
{
   int minElement, lastElement, child, now;
   minElement = heap[1];
   lastElement = heap[heapSize--];
   for (now = 1; now * 2 <= heapSize; now = child) {
      child = now * 2;
      if (child != heapSize && heap[child + 1] < heap[child]) {
         child++;
  }
```

```
        if (lastElement > heap[child]) {
            heap[now] = heap[child];
        } else
        {
            break;
        }
    }
    heap[now] = lastElement;
    return minElement;
}

int main() {
    int number_of_elements;
    printf("Program to demonstrate Heap:\nEnter the number of elements: ");
    scanf("%d", &number_of_elements);
    int iter, element;
    Init();
    printf("Enter the elements: ");
    for (iter = 0; iter < number_of_elements; iter++) {
        scanf("%d", &element);
        Insert(element);
    }
    for (iter = 0; iter < number_of_elements; iter++) {
        printf("%d ", DeleteMin());
    }
    printf("\n");
    return 0;
}
```

Output:

Program to demonstrate Heap
Enter the number of elements: 5
Enter the elements: 645 897 612 849 643
Elements deleted in a sequence: 612 643 645 849 897

b) **Max Heap**

```
#include <stdio.h>
 int main()
 {
 int arr[10], no,  i, j, c, heap_root, temp;
 printf("Input number of elements: ");
 scanf("%d", &no);
 printf("\nInput array values one by one : ");
 for (i = 0; i <  no; i++)
 scanf("%d", &arr[i]);
 for (i = 1; i <  no; i++)
 {
 c = i;
 do
 {
 heap_root  = (c - 1) / 2;
 /* to create MAX arr  array */
 if  (arr[heap_root] < arr[c])
 {
 temp =  arr[heap_root];
 arr[heap_root] = arr[c];
 arr[c]  = temp;
 }
 c =  heap_root;
 } while (c !=  0);
 }

 printf("Heap  array : ");
 for (i = 0; i <  no; i++)
 printf("%d\t ", arr[i]);
 for (j = no - 1; j  >= 0; j--)
 {
 temp = arr[0];
 arr[0] = arr[j];
 arr[j] = temp;
 heap_root = 0;
 do
 {
 c = 2 *  heap_root + 1;
```

```
if  ((arr[c] < arr[c + 1]) && c < j-1)
c++;
if  (arr[heap_root]<arr[c] && c<j)
{
temp =  arr[heap_root];
arr[heap_root] = arr[c];
arr[c]  = temp;
}
heap_root  = c;
} while (c  < j);
}
printf("\nSorted  array : ");
for (i = 0; i <  no; i++)
printf("\t%d", arr[i]);
printf("\n");
}
```

Sample Input:

3
12
15
56

Sample Output:

Input number of elements:
Input array values one by one : Heap array : 56   12   15
Sorted array :   12   15   56

9&10. Program to illustrate insertion and deletion and Rotation on AVL Trees.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
typedef struct node
{
int data;
struct node *left,*right;
int ht;
}
node;
node *insert(node*,int);
node *Delete(node*,int);
void preorder(node *);
void inorder(node *);
int  height(node *);
node *rotateright(node*);
node *rotateleft(node*);
node *RR(node*);
node *LL(node*);
node *LR(node*);
node *RL(node*);
int BF(node*);
int main()
{
node *root=NULL;
int x,n,i,op;

do
{
printf("\n1)create");
printf("\n2)insert");
printf("\n3)delete");
printf("\n4)print");
printf("\n5)quit");
```

```
printf("\n\nenter your choice:");
scanf("%d",&op);
switch(op)
{
case 1:
printf("\n enter no of elememt:");
scanf("%d",&n);
printf("\n enter tree data:");
root=NULL;
for(i=0;i<n;i++)
{
scanf("%d",&x);
root=insert(root,x);
}
break;
case 2:
printf("\n enter the data:");
scanf("%d",&x);
root=insert(root,x);
break;
case 3:
printf("\n enter a data");
scanf("%d",&x);
root=Delete(root,x);
break;
case 4:
printf("\n preorder sequence:\n");
preorder(root);
printf("\n\n inorder sequence:\n");
inorder(root);
printf("\n");
break;
}
}
while(op!=5);
return 0;
}
node *insert(node *T,int x)
{
if(T==NULL)
```

```
{
T=(node*)malloc(sizeof(node));
T->data=x;
//T->data=NULL;
T->left=NULL;
T->right=NULL;
}
else
if(x>T->data)   //insert in right subtree
{
T->right=insert(T->right,x);
if(BF(T)==-2)
if(x>T->right->data)
T=RR(T);
else
T=RL(T);
}
else
if(x<T->data)
{
T->left=insert(T->left,x);
if(BF(T)==2)
if(x<T->left->data)
T=LL(T);
else
T=LR(T);
}
T->ht=height(T);
return(T);
}
node *Delete(node *T,int x)
{
node *p;
if(T==NULL)
{
return NULL;
}
else
if(x>T->data)
{
```

```
T->right=Delete(T->right,x);
if(BF(T)==2)
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);
}
else
if(x<T->data)
{
T->left=Delete(T->left,x);
if(BF(T)==-2)
if(BF(T->right)<=0)
T=RR(T);
else

T=RL(T);
}
else
{
if(T->right!=NULL)
{
p=T->right;
while(p->left!=NULL)
p=p->left;
T->data=p->data;
T->right=Delete(T->right,p->data);
if(BF(T)==2)
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);
}
else
return(T->left);
}
T->ht=height(T);
return(T);
}
int height(node *T)
```

```
{
int lh,rh;
if(T==NULL)
 return(0);
 if(T->left==NULL)
 lh=0;
 else
 lh=0;
 //else
 lh=1+T->left->ht;
 if(T->right==NULL)
   rh=0;
   else
   rh=1+T->right->ht;
   if(lh>rh)
   return(lh);

   return(rh);

  }


 node *rotateright(node *x)
 {
 node *y;
 y=x->left;
 x->left=y->right;
 y->right=x;
 x->ht=height(x);
 y->ht=height(y);
 return(y);
 }
 node *rotateleft(node *x)
 {
 node *y;
 y=x->right;
 x->right=y->left;
 y->left=x;
 x->ht=height(y);
 y->ht=height(y);
 return(y);
```

```
}
node *RR(node *T)
{
T=rotateleft(T);
return(T);
}
node *LL(node *T)
{
T=rotateright(T);
return(T);
}
node *LR(node *T)

{

T->left=rotateleft(T->left);

T=rotateleft(T);

return(T);

}

node *RL(node *T)
{
T->right=rotateright(T->right);
T=rotateleft(T);
return(T);
}
int BF(node *T)
{
int lh, rh;
if(T==NULL)
return(0);
if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;
if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;
```

```
return(lh-rh);
}
void preorder(node *T)
{
if(T!=NULL)
{
printf("%d(BF=%d)",T->data,BF(T));
preorder(T->left);
preorder(T->right);
}
}
void inorder(node *T)
{

if(T!=NULL)
{
inorder(T->left);
printf("%d(BF=%d)",T->data,BF(T));
inorder(T->right);
}
}
```

Output:

```
enter your choice:2

 enter the data:9

1)create
2)insert
3)delete
4)print
5)quit

enter your choice:4

 preorder sequence:
6(BF=-1)5(BF=0)8(BF=0)7(BF=0)9(BF=0)

 inorder sequence:
5(BF=0)6(BF=-1)7(BF=0)8(BF=0)9(BF=0)

1)create
2)insert
3)delete
4)print
5)quit

enter your choice:_
```

```
enter your choice:3

 enter a data7

1)create
2)insert
3)delete
4)print
5)quit

enter your choice:4

 preorder sequence:
6(BF=-1)5(BF=0)8(BF=-1)9(BF=0)

 inorder sequence:
5(BF=0)6(BF=-1)8(BF=-1)9(BF=0)

1)create
2)insert
3)delete
4)print
5)quit

enter your choice:_
```

11. Program to implement B-Tree

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
#define M 3
typedef struct _node {
int n; /*n < M No. of keys in node will always less than order of B tree*/
int keys[M - 1];
struct _node *p[M]; /* (n+1 pointers will be in use) */
} node;
node *root = NULL;
typedef enum KeyStatus { Duplicate, SearchFailure, Success, InsertIt, LessKeys,
} KeyStatus;
void insert(int key);
void display(node *root, int);
void DelNode(int x);
void search(int x);
KeyStatus ins(node *r, int x, int* y, node** u);
int searchPos(int x, int *key_arr, int n);
KeyStatus del(node *r, int x);
void eatline(void);
void inorder(node *ptr);
int main() {
clrscr();
int key;
int choice;
printf("Creation of B tree for M=%d\n", M);
while (1) {
printf("1.Insert\n2.Delete\n3.Search\n4.Display\n5.Quit\nEnter your choice: ");
scanf("%d", &choice); eatline();
switch (choice) {
case 1:
printf("Enter the key: ");
```

```
scanf("%d", &key); eatline();
insert(key);
break;
case 2:
printf("Enter the key : ");
scanf("%d", &key); eatline();
DelNode(key);
break;
case 3:
printf("Enter the key : ");
scanf("%d", &key); eatline();
search(key);
break;
case 4:
printf("Btree is :\n");
display(root, 0);
break;
case 5:
exit(1);
default:
printf("Wrong choice\n");
break;
}}
return 0;
}
void insert(int key) {
node *newnode;
int upKey;
KeyStatus value;
value = ins(root, key, &upKey, &newnode);
if (value == Duplicate)
printf("Key already available\n");
if (value == InsertIt) {
node *uproot = root;
root = (node*)malloc(sizeof(node));
```

```
root->n = 1;
root->keys[0] = upKey;
root->p[0] = uproot;
root->p[1] = newnode;
}}
KeyStatus ins(node *ptr, int key, int *upKey, node **newnode) {
node *newPtr, *lastPtr;
int pos, i, n, splitPos;
int newKey, lastKey;
KeyStatus value;
if (ptr == NULL) {
*newnode = NULL;
*upKey = key;
return InsertIt;
}
n = ptr->n;
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
return Duplicate;
value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)
return value;
/*If keys in node is less than M-1 where M is order of B tree*/
if (n < M - 1) {
pos = searchPos(newKey, ptr->keys, n);
/*Shifting the key and pointer right for inserting the new key*/
for (i = n; i>pos; i--) {
ptr->keys[i] = ptr->keys[i - 1];
ptr->p[i + 1] = ptr->p[i];
}
/*Key is inserted at exact location*/
ptr->keys[pos] = newKey;
ptr->p[pos + 1] = newPtr;
++ptr->n; /*incrementing the number of keys in node*/
return Success;
```

```
}
/*If keys in nodes are maximum and position of node to be inserted is last*/
if (pos == M - 1) {
lastKey = newKey;
lastPtr = newPtr;
}
else { /*If keys in node are maximum and position of node to be inserted is not
last*/
lastKey = ptr->keys[M - 2];
lastPtr = ptr->p[M - 1];
for (i = M - 2; i>pos; i--) {
ptr->keys[i] = ptr->keys[i - 1];
ptr->p[i + 1] = ptr->p[i];
}
ptr->keys[pos] = newKey;
ptr->p[pos + 1] = newPtr;
}
splitPos = (M - 1) / 2;
(*upKey) = ptr->keys[splitPos];
(*newnode) = (node*)malloc(sizeof(node));/*Right node after split*/
ptr->n = splitPos; /*No. of keys for left splitted node*/
(*newnode)->n = M - 1 - splitPos;/*No. of keys for right splitted node*/
for (i = 0; i < (*newnode)->n; i++) {
(*newnode)->p[i] = ptr->p[i + splitPos + 1];
if (i < (*newnode)->n - 1)
(*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
else
(*newnode)->keys[i] = lastKey;
}
(*newnode)->p[(*newnode)->n] = lastPtr;
return InsertIt;
}
void display(node *ptr, int blanks) {
if (ptr) {
int i;
```

```
for (i = 1; i <= blanks; i++)
printf(" ");
for (i = 0; i < ptr->n; i++)
printf("%d ", ptr->keys[i]);
printf("\n");
for (i = 0; i <= ptr->n; i++)
display(ptr->p[i], blanks + 10);
}}
void search(int key) {
int pos, i, n;
node *ptr = root;
printf("Search path:\n");
while (ptr) {
n = ptr->n;
for (i = 0; i < ptr->n; i++)
printf(" %d", ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos]) {
printf("Key %d found in position %d of last displayed node\n", key, i);
return;
}
ptr = ptr->p[pos];
}
printf("Key %d is not available\n", key);
}
int searchPos(int key, int *key_arr, int n) {
int pos = 0;
while (pos < n && key > key_arr[pos])
pos++;
return pos;
}
```

12. Program to illustrate Graph traversals

a)Breadth first Search.

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
for(i=1;i<=n;i++)
if(a[v][i]&& !visited[i])
q[++r]=i;
if(f<=r)
{
 visited[q[f]]=1;
 bfs(q[f++]);
}
}
void main()
{
int v;
clrscr();
printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
q[i]=0;
visited[i]=0;
}
printf("\n enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d", &a[i][j]);
printf("\n enter the starting vertex:");
scanf("%d",&v);
```

```
bfs(v);
printf("\n the node which are reachable are:\n");
for(i=1;i<=n;i++)
 if(visited[i]!=0)
  printf("%d\t",i);
  else
   printf("\n bfs is not possible");
getch();
}
```

Output:

```
 Enter thr number of vertices:3

 enter graph data in matrix form:
0
1
0
1
0
0
0
0
0

 enter the starting vertex:1

 the node which are reachable are:
1       2
 bfs is not possible_
```

```
 Enter thr number of vertices:4

 enter graph data in matrix form:
0
1
1
1
1
0
1
0
1
1
0
1
1
0
1
0

 enter the starting vertex:3

 the node which are reachable are:
1       2       3       4
```

b) Depth first Search

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
int i;
reach[v]=1;
for(i=1; i<=n; i++)
if(a[v][i]&& !reach[i])
{
printf("\n%d->%d", v,i);
dfs(i);
}
}
void main()
{
int i,j,count=0;
clrscr();
printf("\n enter number of vertice:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
reach[i]=0;
for(j=1;j<=n;j++)
a[i][j]=0;
}
printf("\nEnter adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d", &a[i][j]);
dfs(1);
printf("\n");
for(i=1;i<=n;i++)
{
if(reach[i])
 count++;
 }
if(count==n)
```

```
printf("\n Graph is connectde");
else
printf("\n Graph is not connected");
}
```

Output:

```
 enter number of vertice:4

Enter adjacency matrix:
0
1
1
1
1
0
1
0
1
1
0
1
1
0
1
0

1->2
2->3
3->4

 Graph is connectde_
```

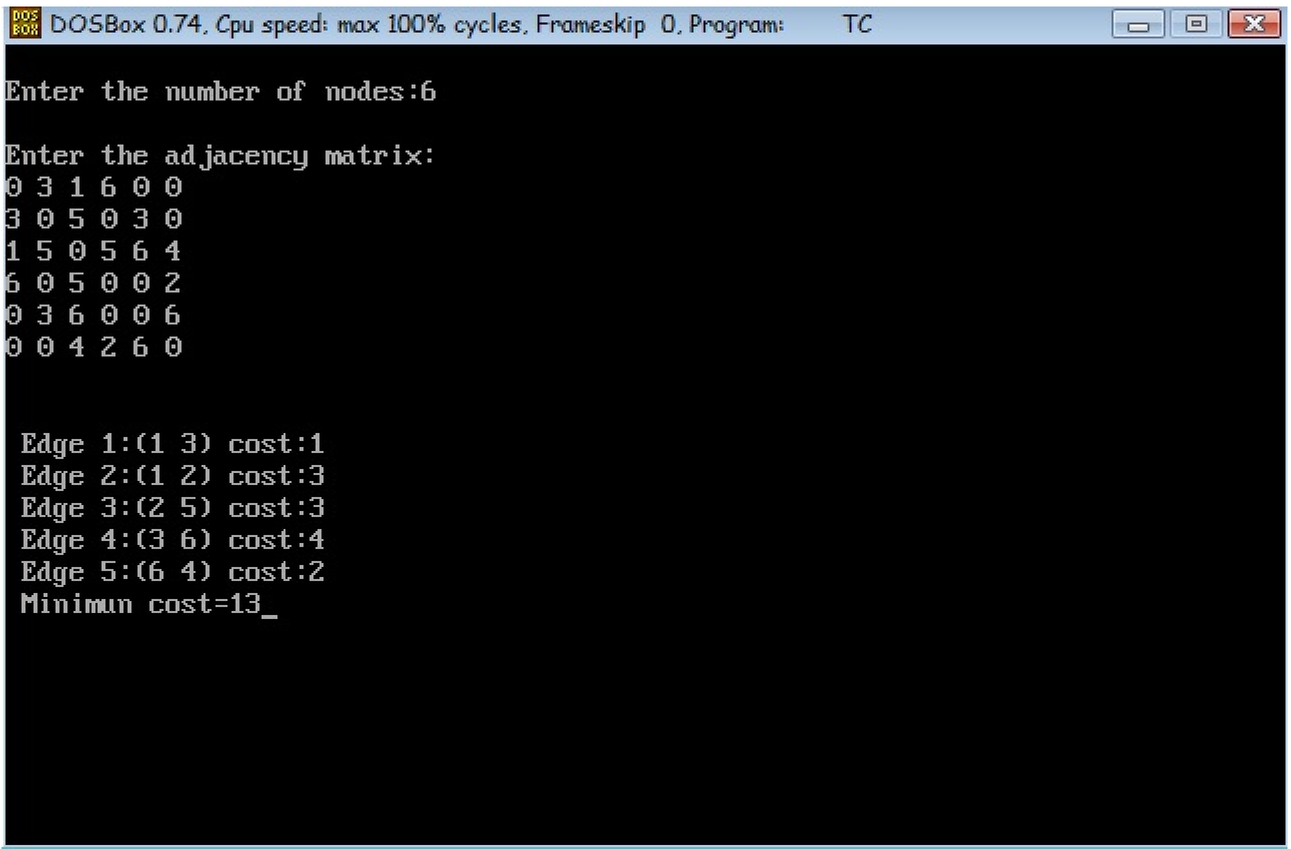
13. Program to implement

a) Prim's algorithm

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

int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{

clrscr();
printf("\nEnter the number of nodes:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;

}
visited[1]=1;
printf("\n");
while(ne < n)
{
        for(i=1,min=999;i<=n;i++)
        for(j=1;j<=n;j++)
        if(cost[i][j]< min)
        if(visited[i]!=0)
        {
                min=cost[i][j];
                a=u=i;
                b=v=j;
        }

if(visited[u]==0 || visited[v]==0)
{
        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
```

```
        mincost+=min;
        visited[b]=1;
 }
        cost[a][b]=cost[b][a]=999;
 }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

Output:



b) Kruskal's algorithm

```
#include<stdio.h>
#define MAX 30

typedef struct edge
{
int u,v,w;
}edge;

typedef struct edgelist
{
edge data[MAX];
int n;
}edgelist;

edgelist elist;

int G[MAX][MAX],n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();

void main()
{
int i,j,total_cost;
printf("\nEnter number of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
kruskal();
print();
}

void kruskal()
```

```
{
int belongs[MAX],i,j,cno1,cno2;
elist.n=0;

for(i=1;i<n;i++)
for(j=0;j<i;j++)
{
if(G[i][j]!=0)
{
elist.data[elist.n].u=i;
elist.data[elist.n].v=j;
elist.data[elist.n].w=G[i][j];
elist.n++;
}
}

sort();
for(i=0;i<n;i++)
belongs[i]=i;
spanlist.n=0;
for(i=0;i<elist.n;i++)
{
cno1=find(belongs,elist.data[i].u);
cno2=find(belongs,elist.data[i].v);
if(cno1!=cno2)
{
spanlist.data[spanlist.n]=elist.data[i];
spanlist.n=spanlist.n+1;
union1(belongs,cno1,cno2);
}
}
}

int find(int belongs[],int vertexno)
{
return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2)
{
```

```
int i;
for(i=0;i<n;i++)
if(belongs[i]==c2)
belongs[i]=c1;
}

void sort()
{
int i,j;
edge temp;
for(i=1;i<elist.n;i++)
for(j=0;j<elist.n-1;j++)
if(elist.data[j].w>elist.data[j+1].w)
{
temp=elist.data[j];
elist.data[j]=elist.data[j+1];
elist.data[j+1]=temp;
}
}

void print()
{
int i,cost=0;
for(i=0;i<spanlist.n;i++)
{
printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
cost=cost+spanlist.data[i].w;
}

printf("\n\nCost of the spanning tree=%d",cost);
}
```

Output:

C:\Users\Student\Documents\program.exe

```
Enter number of vertices:6

Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

2       0       1
5       3       2
1       0       3
4       1       3
5       2       4

Cost of the spanning tree=13
Process returned 30 (0x1E)   execution time : 41.362 s
Press any key to continue.
```

14. Program to Implement Dijkstra algorithm.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijikstra(int G[MAX][MAX], int n, int startnode);

void main(){
        int G[MAX][MAX], i, j, n, u;
        clrscr();
        printf("\nEnter the no. of vertices:: ");
        scanf("%d", &n);
        printf("\nEnter the adjacency matrix::\n");
        for(i=0;i < n;i++)
                for(j=0;j < n;j++)
                        scanf("%d", &G[i][j]);
        printf("\nEnter the starting node:: ");
        scanf("%d", &u);
        dijikstra(G,n,u);
        getch();
}

void dijikstra(int G[MAX][MAX], int n, int startnode)
{
        int cost[MAX][MAX], distance[MAX], pred[MAX];
        int visited[MAX], count, mindistance, nextnode, i,j;
        for(i=0;i < n;i++)
                for(j=0;j < n;j++)
                        if(G[i][j]==0)
                                cost[i][j]=INFINITY;
                        else
                                cost[i][j]=G[i][j];
```

```
for(i=0;i< n;i++)
{
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count < n-1){
        mindistance=INFINITY;
        for(i=0;i < n;i++)
                if(distance[i] < mindistance&&!visited[i])
                {
                        mindistance=distance[i];
                        nextnode=i;
                }
        visited[nextnode]=1;
        for(i=0;i < n;i++)
                if(!visited[i])
                        if(mindistance+cost[nextnode][i] < distance[i])
                        {
                                distance[i]=mindistance+cost[nextnode][i];
                                pred[i]=nextnode;
                        }
                count++;
}

for(i=0;i < n;i++)
        if(i!=startnode)
        {
                printf("\nDistance of %d = %d", i, distance[i]);
                printf("\nPath = %d", i);
                j=i;
```

```
                do
                {
                        j=pred[j];
                        printf(" <-%d", j);
                }
                while(j!=startnode);
        }
}
```

output

```
Enter the no. of vertices:: 4

Enter the adjacency matrix::
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0

Enter the starting node:: 1

Distance of 0 = 1
Path = 0<-1
Distance of 2 = 1
Path = 2<-1
Distance of 3 = 2
Path = 3<-0<-1
```

15. Program to implement Hashing and collision resolution techniques.

```
#include<stdio.h>
#include<stdlib.h>
#define size 7

struct node
{
   int data;
   struct node *next;
};

struct node *chain[size];
void init()
{
   int i;
   for(i = 0; i < size; i++)
      chain[i] = NULL;
}

void insert(int value)
{
   //create a newnode with value
   struct node *newNode = malloc(sizeof(struct node));
   newNode->data = value;
   newNode->next = NULL;

   //calculate hash key
   int key = value % size;

   //check if chain[key] is empty
   if(chain[key] == NULL)
      chain[key] = newNode;
   //collision
   else
   {
      //add the node at the end of chain[key].
      struct node *temp = chain[key];
```

```
        while(temp->next)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void print()
{
    int i;

    for(i = 0; i < size; i++)
    {
        struct node *temp = chain[i];
        printf("chain[%d]-->",i);
        while(temp)
        {
            printf("%d -->",temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main()
{
    //init array of list to NULL
    init();
    insert(7);
    insert(0);
    insert(3);
    insert(10);
    insert(4);
    insert(5);
    print();

    return 0;
}
```

Output:

```
chain[0]-->7 -->0 -->NULL
chain[1]-->NULL
chain[2]-->NULL
chain[3]-->3 -->10 -->NULL
chain[4]-->4 -->NULL
chain[5]-->5 -->NULL
chain[6]-->NULL
```

16. Program to implement Dictionaries.

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

static char *companies[] = { "Intel", "AMD", "ARM", "Apple",
                    "Marvell", "Qualcomm", "IBM", "Nvidia" };

static char *uarch[] = { "Willow Cove", "Zen 3", "A78", "A14",
                    "ThunderX2", "Kryo", "z15", "Ampere" };

int main(void) {
  ENTRY e;
  ENTRY *ep;

  const size_t capacity = sizeof companies / sizeof companies[0];
  hcreate(capacity);

  for (size_t i = 0; i < capacity - 2; i++) {
     e.key = companies[i];
     e.data = (void *) uarch[i];

     ep = hsearch(e, ENTER);

     if (ep == NULL) {
        fprintf(stderr, "entry failed\n");
        exit(EXIT_FAILURE);
     }
  }

  for (size_t i = 0; i < capacity; i++) {
     e.key = companies[i];
     ep = hsearch(e, FIND);
```

```
        ep ? printf("%s -> %s\n", e.key, (char*)ep->data) :
            printf("%s -> %s\n", e.key, "Entry not found");
    }

    hdestroy();
    exit(EXIT_SUCCESS);
}
```

Output:

Intel -> Willow Cove
AMD -> Zen 3
ARM -> A78
Apple -> A14
Marvell -> ThunderX2
Qualcomm -> Kryo
IBM -> Entry not found
Nvidia -> Entry not found