J.B.INSTITUTE OF ENGINEERING & TECHNOLOGY

YENKAPALLY(V), MOINABAD(M), HYDERABAD-500075

DEPARTMENT OF MCA

DATA STRUCTURES AND ALGORITHMS

S.NO	DESCRIPTION OF THE PROGRAM
1	Write C++ programs to implement the Stack ADT using an array
2	Write C++ programs to implement the Queue ADT using an array
3	Write C++ programs to implement the Stack ADT using a singly linkedlist
4	Write C++ programs to implement the Queue ADT using a singly linked list
5	Write C++ program to implement the deque (double ended queue) ADT using a doubly linked list
6	Write a C++ program to perform the following operations:a) Insert an element into a binary search tree. b) Delete an element from a binary search tree .c) Search for a key element in a binary search tree
7	Write a C++ program to implement circular queue ADT using an array
8	Write a C++ program to implement all the functions of a dictionary (ADT) using hashing
9	Write a C++ program to perform the following operations on AVL-trees: a) Insertion. b) Deletion.
10	Write C++ programs for the implementation of BFS for a given graph
11	Write C++ programs for the implementation of DFS for given graph
12	Write C++ programs to implement the Prim's algorithm to generate a minimum cost spanning tree
13	Write C++ programs to implement the Kruskal's algorithm to generate a minimum cost spanning tree
14	Write a C++ program to solve the single source shortest path problem. (Note: Use Dijkstra's algorithm
15	Write C++ program that uses recursive functions to traverse a binary tree in a)pre-order,b)in-order,c)post order
16	Write C++ program that uses non-recursive functions to traverse a binary tree in a)pre-order,b)in-order,c)post order
17	Write C++ programs for sorting a given list of elements in ascending order using Quick sort sorting methods
18	Write C++ programs for sorting a given list of elements in ascending order using merge sorting methods
19	Write a C++ program that uses dynamic programming algorithm to solve the optimal binary search tree problem
20	Write a C++ program for AVL TREE.
21	Write a C++ program for B+TREE.

/* Write C++ programs to implement the Stack ADT using an array */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class stack
{
        int stk[5];
        int top;
    public:
        stack()
         {
           top=-1;
         }
        void push(int x)
         {
           if(top >  4)
                {
                   cout <<"stack over flow";
                   return;
                }
```

```
    stk[++top]=x;
    cout <<"inserted" <<x;
  }
 void pop()
  {
    if(top <0)
     {
        cout <<"stack under flow";
        return;
     }
     cout <<"deleted" <<stk[top--];
   }
 void display()
  {
    if(top<0)
     {
        cout <<" stack empty";
        return;
     }
     for(int i=top;i>=0;i--)
     cout <<stk[i] <<" ";
  }};
main()
```

```
{
    int ch;
    stack st;
    while(1)
      {
          cout <<"\n1.push  2.pop  3.display  4.exit\nEnter ur choice";
          cin >> ch;
          switch(ch)
           {
            case 1:  cout <<"enter the element";
                     cin >> ch;
                     st.push(ch);
                     break;
            case 2:  st.pop();  break;
            case 3:  st.display();break;
            case 4:  exit(0);
            }
       }
return (0);
}
```

OUTPUTS

1.push 2.pop 3.display 4.exit

Enter ur choice2

stack under flow

1.push 2.pop 3.display 4.exit

Enter ur choice1

enter the element2

inserted2

1.push 2.pop 3.display 4.exit

Enter ur choice1

enter the element3

inserted3

1.push 2.pop 3.display 4.exit

Enter ur choice2

deleted3

1.push 2.pop 3.display 4.exit

Enter ur choice1

enter the element5

inserted5

1.push 2.pop 3.display 4.exit

Enter ur choice3

5 2

1.push 2.pop 3.display 4.exit

Enter ur choice4

/* Write C++ programs to implement the Queue ADT using an array */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;

class queue
{
        int queue1[5];
        int rear,front;
    public:
        queue()
          {
             rear=-1;
             front=-1;
          }
        void insert(int x)
         {
            if(rear >  4)
              {
                cout <<"queue over flow";
                front=rear=-1;
```

```
        return;
      }
      queue1[++rear]=x;
      cout <<"inserted" <<x;
 }
void delet()
 {
    if(front==rear)
      {
         cout <<"queue under flow";
         return;
      }
      cout <<"deleted" <<queue1[++front];
  }
void display()
 {
    if(rear==front)
      {
          cout <<" queue empty";
          return;
      }
    for(int i=front+1;i<=rear;i++)
    cout <<queue1[i]<<" ";
```

```
        }
};
main()
{
    int ch;
    queue qu;
    while(1)
     {
        cout <<"\n1.insert  2.delet  3.display  4.exit\nEnter ur choice";
        cin >> ch;
        switch(ch)
          {
           case 1:    cout <<"enter the element";
                      cin >> ch;
                    qu.insert(ch);
                    break;
           case 2:  qu.delet();  break;
           case 3:  qu.display();break;
           case 4: exit(0);
           }
      }
return (0);
}
```

OUTPUT

1.insert 2.delet 3.display 4.exit

Enter ur choice1

enter the element21

inserted21

1.insert 2.delet 3.display 4.exit

Enter ur choice1

enter the element22

inserted22

1.insert 2.delet 3.display 4.exit

Enter ur choice1

enter the element16

inserted16

1.insert 2.delet 3.display 4.exit

Enter ur choice3

21 22 16 1.insert 2.delet 3.display 4.exit

Enter ur choice2

deleted21

1.insert 2.delet 3.display 4.exit

Enter ur choice3

22 16

1.insert 2.delet 3.display 4.exit

Enter ur choice

/* Write C++ programs to implement the Stack ADT using a singly linkedlist*/

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class node
{
    public:
        class node *next;
        int data;
};

class stack : public node
{
        node *head;
        int tos;
    public:
        stack()
          {
              tos=-1;
```

```
 }
void push(int x)
 {
      if (tos < 0 )
    {
      head =new node;
      head->next=NULL;
      head->data=x;
      tos ++;
     }
else
    {
       node *temp,*temp1;
      temp=head;
      if(tos >= 4)
              {
                      cout <<"stack over flow";
             return;
          }
      tos++;
      while(temp->next != NULL)
          temp=temp->next;
      temp1=new node;
```

```
        temp->next=temp1;
        temp1->next=NULL;
        temp1->data=x;
      }
   }
void display()
 {
    node *temp;
    temp=head;
    if (tos < 0)
     {
        cout <<" stack under flow";
        return;
      }
    while(temp != NULL)
      {
        cout <<temp->data<< " ";
        temp=temp->next;
      }
 }
void pop()
 {
    node *temp;
```

```
            temp=head;
            if( tos < 0 )
             {
               cout <<"stack under flow";
               return;
             }
             tos--;
             while(temp->next->next!=NULL)
               {
                     temp=temp->next;
                }
             temp->next=NULL;
           }
};
main()
{
        stack s1;
        int ch;
        while(1)
        {
        cout <<"\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n enter ur choice:";
        cin >> ch;
        switch(ch)
```

```
        {
            case 1:   cout <<"\n enter a element";
                      cin >> ch;
                      s1.push(ch);
                      break;
            case 2:   s1.pop();break;




            case 3:   s1.display();
                       break;
            case 4:   exit(0);
            }
    }
return (0);
}
```

OUTPUT

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:1

enter a element23

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:1

enter a element67

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:3

23 67

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:2

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:3

23

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:2

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:2

stack under flow

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:4

/* Write C++ programs to implement the Queue ADT using a singly linked list */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class node
{
    public:
        class node *next;
        int data;
};

class queue : public node
{
        node *head;
        int front,rare;
       public:
        queue()
                {
                front=-1;
                rare=-1;
```

```
 }
void push(int x)
        {
        if (rare < 0 )
                {
                        head =new node;
                head->next=NULL;
        head->data=x;
                rare ++;
         }
  else
        {
        node *temp,*temp1;
                temp=head;
                if(rare >= 4)
             {
                cout <<"queue over flow";
                return;
              }
           rare++;
           while(temp->next != NULL)
                temp=temp->next;
           temp1=new node;
```

```
            temp->next=temp1;
            temp1->next=NULL;
            temp1->data=x;
          }  }

void display()
        {
   node *temp;
   temp=head;
   if (rare < 0)
    {
        cout <<" queue under flow";
        return;
     }
   while(temp != NULL)
    {
          cout <<temp->data<< " ";
       temp=temp->next;
    }
}
   void pop()
    {
         node *temp;
```

```
            temp=head;
        if( rare < 0)
          {
            cout <<"queue under flow";
            return;
           }
        if(front == rare)
          {
            front = rare =-1;
            head=NULL;
           return;
           }
        front++;
        head=head->next;
        }
};
main()
{
      queue s1;
      int ch;
      while(1)
    {
            cout<<"\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n enter ru choice:";
```

```
            cin >> ch;
            switch(ch)
            {
        case 1:
            cout <<"\n enter a element";
            cin >> ch;
            s1.push(ch); break;

        case 2: s1.pop();break;
        case 3: s1.display();break;
       case 4: exit(0);
              }
    }
return (0);
}
```

OUTPUT

```
1.PUSH 2.POP 3.DISPLAY 4.EXIT
 enter ru choice:1
 enter a element23

1.PUSH 2.POP 3.DISPLAY 4.EXIT
```

enter ru choice:1

enter a element54

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:3

23 54

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:2

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:2

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:2

queue under flow

1.PUSH 2.POP 3.DISPLAY 4.EXIT

enter ru choice:4

/* Write C++ program to implement the deque (double ended queue) ADT using a doubly linked list */

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

```
#include<stdlib.h>
using namespace std;


class node
{
public:
int data;
class  node *next;
class node *prev;
};


class dqueue: public node
{
  node *head,*tail;
  int top1,top2;
  public:
   dqueue()
   {
   top1=0;
   top2=0;
   head=NULL;
   tail=NULL;
   }
```

```
void push(int x){
        node *temp;
        int ch;
        if(top1+top2 >=5)
        {
          cout <<"dqueue overflow";
          return ;
        }
        if( top1+top2 == 0)
          {
            head = new node;
            head->data=x;
            head->next=NULL;
            head->prev=NULL;
            tail=head;
            top1++;
          }
         else
         {
           cout <<" Add element 1.FIRST 2.LAST\n enter ur choice:";
           cin >> ch;
```

```
        if(ch==1)
        {
          top1++;
          temp=new node;
          temp->data=x;
          temp->next=head;
          temp->prev=NULL;
          head->prev=temp;
          head=temp;
        }
        else
        {
          top2++;
          temp=new node;
          temp->data=x;
          temp->next=NULL;
          temp->prev=tail;
          tail->next=temp;
          tail=temp;
        }

      }
}
```

```
void pop()
{
 int ch;
 cout <<"Delete 1.First Node 2.Last Node\n Enter ur choice:";
 cin >>ch;
 if(top1 + top2 <=0)
 {
   cout <<"\nDqueue under flow";
   return;
 }
 if(ch==1)
 {
   head=head->next;
   head->prev=NULL;
   top1--;
 }
 else
 {
   top2--;
   tail=tail->prev;
   tail->next=NULL;
 }
}
```

```
void display()
{
 int ch;
 node *temp;
 cout <<"display from 1.Staring 2.Ending\n Enter ur choice";
 cin >>ch;
 if(top1+top2 <=0)
 {
   cout <<"under flow";
   return ;
 }
 if (ch==1)
 {
  temp=head;
  while(temp!=NULL)
  {
    cout << temp->data <<" ";
    temp=temp->next;
  }
 }
```

```
 else
 {
  temp=tail;
  while( temp!=NULL)
  {
    cout <<temp->data << " ";
    temp=temp->prev;
  }
 }
  }
};

main()
{
  dqueue d1;
  int ch;
  while (1){
       cout <<"1.INSERT  2.DELETE  3.DISPLAU  4.EXIT\n Enter ur choice:";
  cin >>ch;
  switch(ch)
  {
  case 1:     cout <<"enter element";
                cin >> ch;
```

```
                d1.push(ch); break;
    case 2: d1.pop(); break;
    case 3: d1.display(); break;
    case 4: exit(1);
    }
  }}
```

OUTPUT

```
1.INSERT  2.DELETE  3.DISPLAU  4.EXIT
 Enter ur choice:1
enter element4
1.INSERT  2.DELETE  3.DISPLAU  4.EXIT
 Enter ur choice:1
enter element5
 Add element 1.FIRST 2.LAST
 enter ur choice:1
1.INSERT  2.DELETE  3.DISPLAU  4.EXIT
 Enter ur choice:1
enter element6
 Add element 1.FIRST 2.LAST
 enter ur choice:2
1.INSERT  2.DELETE  3.DISPLAU  4.EXIT
 Enter ur choice:3
```

display from 1.Staring 2.Ending

Enter ur choice1

5 4 6 1.INSERT 2.DELETE 3.DISPLAU 4.EXIT

Enter ur choice:2

Delete 1.First Node 2.Last Node

Enter ur choice:1

1.INSERT 2.DELETE 3.DISPLAU 4.EXIT

Enter ur choice:3

display from 1.Staring 2.Ending

Enter ur choice1

4 6 1.INSERT 2.DELETE 3.DISPLAU 4.EXIT

Enter ur choice:4

*** Write a C++ program to perform the following operations:**

a) Insert an element into a binary search tree.

b) Delete an element from a binary search tree.

c) Search for a key element in a binary search tree. */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
```

```
void insert(int,int );
void delte(int);
void display(int);
int search(int);
int search1(int,int);
int tree[40],t=1,s,x,i;

main()
{
        int ch,y;
        for(i=1;i<40;i++)
        tree[i]=-1;
        while(1)
        {
cout <<"1.INSERT\n2.DELETE\n3.DISPLAY\n4.SEARCH\n5.EXIT\nEnter your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:
                        cout <<"enter the element to insert";
                        cin >> ch;
                        insert(1,ch);
```

```
        break;
case 2:
        cout <<"enter the element to delete";
        cin >>x;
        y=search(1);
        if(y!=-1) delte(y);
        else cout<<"no such element in tree";
        break;
case 3:
        display(1);
        cout<<"\n";
        for(int i=0;i<=32;i++)
        cout <<i;
        cout <<"\n";
          break;
case 4:
        cout <<"enter the element to search:";
        cin >> x;
        y=search(1);
        if(y == -1) cout <<"no such element in tree";
        else cout <<x << "is in" <<y <<"position";
        break;
case 5:
```

```
                exit(0);
            }
        }
}

void insert(int s,int ch )
{
        int x;
        if(t==1)
        {
            tree[t++]=ch;
            return;
        }
        x=search1(s,ch);
        if(tree[x]>ch)
            tree[2*x]=ch;
        else
            tree[2*x+1]=ch;
        t++;
}
void delte(int x)
{
        if( tree[2*x]==-1 && tree[2*x+1]==-1)
```

```
            tree[x]=-1;
        else if(tree[2*x]==-1)
            {   tree[x]=tree[2*x+1];
                tree[2*x+1]=-1;
            }
        else if(tree[2*x+1]==-1)
            {   tree[x]=tree[2*x];
                tree[2*x]=-1;
            }
        else
        {
          tree[x]=tree[2*x];
          delte(2*x);
        }
        t--;
}



int search(int s)
{
if(t==1)
{
cout <<"no element in tree";
```

```
return -1;
}
if(tree[s]==-1)
return tree[s];
if(tree[s]>x)
search(2*s);
else if(tree[s]<x)
search(2*s+1);
else
return s;
}

void display(int s)
{
if(t==1)
{cout <<"no element in tree:";
return;}
for(int i=1;i<40;i++)
if(tree[i]==-1)
cout <<" ";
else cout <<tree[i];
return ;
}
```

```
int search1(int s,int ch)
{
if(t==1)
{
cout <<"no element in tree";
return -1;
}
if(tree[s]==-1)
return s/2;
if(tree[s] > ch)
search1(2*s,ch);
else search1(2*s+1,ch);
}
```

OUTPUT

```
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:3
no element in tree:
0123456789011121314151617181920212223242526272829303132
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:1
```

```
Enter the element to insert 10
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:4
Enter the element to search: 10
10 is in 1 position
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:5
```

/* Write a C++ program to implement circular queue ADT using an array */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class cqueue
{
  int q[5],front,rare;
  public:
    cqueue()
    {
```

```
        front=-1;
        rare=-1;
    }
    void push(int x)
    {
        if(front ==-1 && rare == -1)
        {
          q[++rare]=x;
          front=rare;
          return;
        }
        else if(front == (rare+1)%5 )
        {
           cout <<" Circular Queue over flow";
           return;
        }
        rare= (rare+1)%5;
        q[rare]=x;
    }

    void pop()
    {
        if(front==-1 && rare==   -1)
```

```
        {
          cout <<"under flow";
          return;
        }
        else if( front== rare  )
        {
          front=rare=-1;
          return;
        }
        front= (front+1)%5;
    }



void display()
   {
     int i;
     if( front <= rare)
     {
        for(i=front; i<=rare;i++)
        cout << q[i]<<" ";
     }
```

```
    else
    {
        for(i=front;i<=4;i++)
        {
          cout <<q[i] << " ";
        }
        for(i=0;i<=rare;i++)
        {
           cout << q[i]<< " ";
        }
    }
   }
};

main()
{

int ch;
cqueue q1;
while( 1)
{
cout<<"\n1.INSERT   2.DELETE   3.DISPLAY    4.EXIT\nEnter ur choice";
cin >> ch;
```

```
switch(ch)
{
case 1: cout<<"enter element";
        cin >> ch;
        q1.push(ch); break;

case 2: q1.pop(); break;
case 3: q1.display(); break;
case 4: exit(0);
}}}
```

OUTPUT

```
1.INSERT   2.DELETE   3.DISPLAY    4.EXIT
Enter ur choice1
enter element4
1.INSERT   2.DELETE   3.DISPLAY    4.EXIT
Enter ur choice1
enter element5
1.INSERT   2.DELETE   3.DISPLAY    4.EXIT
Enter ur choice1
enter element3
1.INSERT   2.DELETE   3.DISPLAY    4.EXIT
Enter ur choice3
4 5 3
```

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT

Enter ur choice2

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT

Enter ur choice3

5 3

1.INSERT 2.DELETE 3.DISPLAY 4.EXIT

Enter ur choice4

/* Write a C++ program to implement all the functions of a dictionary (ADT) using hashing */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
# define max 10

typedef struct list
{
int data;
struct list  *next;
}node_type;
```

```
node_type *ptr[max],*root[max],*temp[max];


class Dictionary
{
public:
  int index;

  Dictionary();
  void insert(int);
  void search(int);
  void delete_ele(int);
};

Dictionary::Dictionary()
{
index=-1;
for(int i=0;i<max;i++)
{
root[i]=NULL;
ptr[i]=NULL;
temp[i]=NULL;
}
}
```

```
void Dictionary::insert(int key)
{
index=int(key%max);
ptr[index]=(node_type*)malloc(sizeof(node_type));
ptr[index]->data=key;
if(root[index]==NULL)
{
root[index]=ptr[index];
root[index]->next=NULL;
temp[index]=ptr[index];
}

else
{
temp[index]=root[index];
while(temp[index]->next!=NULL)
temp[index]=temp[index]->next;
temp[index]->next=ptr[index];
}
}

void Dictionary::search(int key)
```

```
{
int flag=0;
index=int(key%max);
temp[index]=root[index];
while(temp[index]!=NULL)
{
if(temp[index]->data==key)
{
cout<<"\nSearch key is found!!";
flag=1;
break;
}
else temp[index]=temp[index]->next;
}
if (flag==0)
cout<<"\nsearch key not found.....";
}

void Dictionary::delete_ele(int key)
{
index=int(key%max);
temp[index]=root[index];
while(temp[index]->data!=key && temp[index]!=NULL)
```

```
{
ptr[index]=temp[index];
temp[index]=temp[index]->next;
}
ptr[index]->next=temp[index]->next;
cout<<"\n"<<temp[index]->data<<" has been deleted.";
temp[index]->data=-1;
temp[index]=NULL;
free(temp[index]);
}
main()
{
int val,ch,n,num;
char c;
Dictionary d;

do
{
cout<<"\nMENU:\n1.Create";
cout<<"\n2.Search for a value\n3.Delete an value";
cout<<"\nEnter your choice:";
cin>>ch;
switch(ch)
```

```
{
case 1:cout<<"\nEnter the number of elements to be inserted:";
      cin>>n;
      cout<<"\nEnter the elements to be inserted:";
      for(int i=0;i<n;i++)
      {
      cin>>num;
      d.insert(num);
      }
      break;
case 2:cout<<"\nEnter the element to be searched:";
      cin>>n;
      d.search(n);
case 3:cout<<"\nEnter the element to be deleted:";
      cin>>n;
      d.delete_ele(n);
      break;
default:cout<<"\nInvalid choice....";
}
cout<<"\nEnter y to continue.....";
cin>>c;
}while(c=='y');
getch();
```

}

OUTPUT

MENU:
1.Create
2.Search for a value
3.Delete an value
Enter your choice:1

Enter the number of elements to be inserted:8

Enter the elements to be inserted:10 4 5 8 7 12 6 1

Enter y to continue.....y

MENU:
1.Create
2.Search for a value
3.Delete an value
Enter your choice:2

Enter the element to be searched:12

Search key is found!!
Enter the element to be deleted:1

1 has been deleted.
Enter y to continue.....y

/* Write a C++ program to perform the following operations on AVL-trees:

a) Insertion.

b) Deletion. */

```
#include<iostream>
#include<stdlib.h>
using namespace std;
#define TRUE 1
#define FALSE 0
#define NULL 0
class AVL;
class AVLNODE
{
        friend class AVL;
        private:
                int data;
                AVLNODE *left,*right;
                int bf;
};
class AVL
{
```

```
    private:
        AVLNODE *root;
    public:
        AVLNODE *loc,*par;
        AVL()
        {
            root=NULL;
        }
        int insert(int);
        void displayitem();
        void display(AVLNODE *);
        void removeitem(int);
        void remove1(AVLNODE *,AVLNODE *,int);
        void remove2(AVLNODE *,AVLNODE *,int);
        void search(int x);
        void search1(AVLNODE *,int);
};
int AVL::insert(int x)
{
    AVLNODE *a,*b,*c,*f,*p,*q,*y,*clchild,*crchild;
    int found,unbalanced;
    int d;
    if(!root)   //special case empty tree
```

```
{        y=new AVLNODE;
        y->data=x;
        root=y;
        root->bf=0;
        root->left=root->right=NULL;
        return TRUE;       }
//phase 1:locate insertion point for x.a keeps track of the most
// recent node with balance factor +/-1,and f is the parent of a
// q follows p through the tree.
f=NULL;
a=p=root;
q=NULL;
found=FALSE;
while(p&&!found)
{               //search for insertion point for x
        if(p->bf)
        {
                a=p;
                f=q;
        }
        if(x<p->data)    //take left branch
        {
                q=p;
```

```
            p=p->left;
        }
        else if(x>p->data)
        {
            q=p;
            p=p->right;
        }
        else
        {
            y=p;
            found=TRUE;
        }
    }           //end while
    //phase 2:insert and rebalance.x is not in the tree and
    // may be inserted as the appropriate child of q.
    if(!found)
    {
        y = new AVLNODE;
        y->data=x;
        y->left=y->right=NULL;
        y->bf=0;
        if(x<q->data)    //insert as left child
        q->left=y;
```

```
else
q->right=y;    //insert as right child
//adjust balance factors of nodes on path from a to q
//note that by the definition of a,all nodes on this
//path must have balance factors of 0 and so will change
//to +/- d=+1 implies that x is inserted in the left
// subtree of a d=-1 implies
//to that x inserted in the right subtree of a.



if(x>a->data)
{
        p=a->right;
        b=p;
        d=-1;
}
else
{
        p=a->left;
        b=p;
        d=1;
}
while(p!=y)
```

```
if(x>p->data)         //height of  right increases by 1
{
        p->bf=-1;
        p=p->right;
}
else                  //height of left increases by 1
{
        p->bf=1;
        p=p->left;
}
//is tree unbalanced
unbalanced=TRUE;
if(!(a->bf)||!(a->bf+d))
{               //tree still balanced
        a->bf+=d;
        unbalanced=FALSE;
}
if(unbalanced)   //tree unbalanced,determine rotation type
{
        if(d==1)
        {       //left imbalance
                if(b->bf==1)      //rotation type LL
                {
```

```
            a->left=b->right;
            b->right=a;
            a->bf=0;
            b->bf=0;
        }
        else    //rotation type LR
        {
            c=b->right;
            b->right=c->left;
            a->left=c->right;
            c->left=b;
            c->right=a;


            switch(c->bf)
            {
                case 1: a->bf=-1;  //LR(b)
                        b->bf=0;
                        break;
                case -1:b->bf=1;  //LR(c)
                        a->bf=0;
                        break;
                case 0: b->bf=0;  //LR(a)
```

```
                    a->bf=0;
                    break;
                }
                c->bf=0;
                b=c; //b is the new root
            } //end of LR
        }       //end of left imbalance
    else    //right imbalance
    {
            if(b->bf==-1)     //rotation type RR
            {
                a->right=b->left;
                b->left=a;
                a->bf=0;
                b->bf=0;
            }
            else    //rotation type LR
            {
                c=b->right;
                b->right=c->left;
                a->right=c->left;
                c->right=b;
                c->left=a;
```

```
                switch(c->bf)
                {
                        case 1: a->bf=-1;  //LR(b)
                                b->bf=0;
                                break;
                        case -1:b->bf=1;  //LR(c)
                                a->bf=0;
                                break;
                        case 0: b->bf=0;  //LR(a)
                                a->bf=0;
                                break;
                }
                c->bf=0;
                b=c; //b is the new root
            } //end of LR
        }
//subtree with root b has been rebalanced and is the new subtree

        if(!f)
        root=b;
        else if(a==f->left)
        f->left=b;
        else if(a==f->right)
```

```
                    f->right=b;
                }   //end of if unbalanced
                return TRUE;
        }       //end of if(!found)
        return FALSE;
}    //end of AVL INSERTION

void AVL::displayitem()
{
        display(root);
}
void AVL::display(AVLNODE *temp)
{
        if(temp==NULL)
        return;
        cout<<temp->data<<" ";
        display(temp->left);
        display(temp->right);
}
void AVL::removeitem(int x)
{
        search(x);
        if(loc==NULL)
```

```
        {
                cout<<"\nitem is not in tree";
                return;
        }
        if(loc->right!=NULL&&loc->left!=NULL)
        remove1(loc,par,x);
        else
        remove2(loc,par,x);
}
void AVL::remove1(AVLNODE *l,AVLNODE *p,int x)
{
        AVLNODE *ptr,*save,*suc,*psuc;
        ptr=l->right;
        save=l;
        while(ptr->left!=NULL)
        {
                save=ptr;
                ptr=ptr->left;
        }
        suc=ptr;
        psuc=save;
        remove2(suc,psuc,x);
        if(p!=NULL)
```

```
                if(l==p->left)
                        p->left=suc;
                else
                        p->right=suc;
        else
                root=l;
         suc->left=l->left;
         suc->right=l->right;
          return;
}
void AVL::remove2(AVLNODE *s,AVLNODE *p,int x)
{
        AVLNODE *child;
        if(s->left==NULL && s->right==NULL)
                child=NULL;
        else if(s->left!=NULL)
                child=s->left;
        else
                child=s->right;
        if(p!=NULL)
                if(s==p->left)
                        p->left=child;
                else
```

```
                p->right=child;
        else
                root=child;

}
void AVL::search(int x)
{
        search1(root,x);
}
void AVL::search1(AVLNODE *temp,int x)
{
    AVLNODE *ptr,*save;
    int flag;
    if(temp==NULL)
    {
            cout<<"\nthe tree is empty";
            return;
    }
    if(temp->data==x)
    {
            cout<<"\nthe item is root and is found";
            par=NULL;
            loc=temp;
```

```
            par->left=NULL;
            par->right=NULL;
            return;      }
    if( x < temp->data)
    {
            ptr=temp->left;
            save=temp;
    }
    else
    {
            ptr=temp->right;
            save=temp;
    }
    while(ptr!=NULL)
    {
            if(x==ptr->data)
            {       flag=1;
                    cout<<"\nitemfound";
                    loc=ptr;
                    par=save;

            }
            if(x<ptr->data)
```

```
            ptr=ptr->left;
            else
            ptr=ptr->right;
      }
      if(flag!=1)
      {
            cout<<"item is not there in tree";
            loc=NULL;
            par=NULL;
            cout<<loc;
            cout<<par;
      }
}

main()
{
        AVL a;
        int x,y,c;
       char ch;
        do
        {
            cout<<"\n1.insert";
            cout<<"\n2.display";
```

```
cout<<"\n3.delete";
cout<<"\n4.search";
cout<<"\n5.exit";
cout<<"\nEnter u r choice to perform on AVL tree";
cin>>c;


switch(c)
{
        case 1:cout<<"\nEnter an element to insert into tree";
                cin>>x;
                a.insert(x);
                break;
        case 2:a.displayitem(); break;
        case 3:cout<<"\nEnter an item to deletion";
              cin>>y;
              a.removeitem(y);
              break;
        case 4:cout<<"\nEnter an element to search";
                cin>>c;
                a.search(c);
                break;
        case 5:exit(0);        break;
```

```
                default :cout<<"\nInvalid option try again";
            }
            cout<<"\ndo u want to continue";
            cin>>ch;
        }
        while(ch=='y'||ch=='Y');
}
```

OUTPUT

```
1.insert 2.display 3.delete 4.search 5.exit
Enter u r choice to perform on AVL tree1
Enter an element to insert into tree4

do u want to continuey

1.insert 2.display 3.delete 4.search 5.exit
Enter u r choice to perform on AVL tree1
Enter an element to insert into tree5

do u want to continuey

1.insert 2.display 3.delete 4.search 5.exit
Enter u r choice to perform on AVL tree3

Enter an item to deletion5

itemfound
do u want to continuey

1.insert 2.display 3.delete 4.search 5.exit
Enter u r choice to perform on AVL tree2
4
do u want to continue4
```

/* Write C++ programs for the implementation of BFS for a given graph */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];

main()
{
int m;
cout <<"enterno of vertices";
cin >> n;
cout <<"ente no of edges";
cin >> m;
cout <<"\nEDGES \n";
for(k=1;k<=m;k++)
{
cin >>i>>j;
cost[i][j]=1;
}
```

```
cout <<"enter initial vertex";
cin >>v;
cout <<"Visitied vertices\n";
cout << v;
visited[v]=1;
k=1;
while(k<n)
{
for(j=1;j<=n;j++)
if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
{
visit[j]=1;
qu[rare++]=j;
}
v=qu[front++];
cout<<v << " ";
k++;
visit[v]=0; visited[v]=1;
}
}
```

OUTPUT

```
enterno of vertices9
ente no of edges9
```

EDGES

1 2

2 3

1 5

1 4

4 7

7 8

8 9

2 6

5 7

enter initial vertex1

Visited vertices

12 4 5 3 6 7 8 9

/* Write C++ programs for the implementation of DFS for a given graph */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10];
main()
{
int m;
```

```
cout <<"enterno of vertices";
cin >> n;
cout <<"ente no of edges";
cin >> m;
cout <<"\nEDGES \n";
for(k=1;k<=m;k++)
{
cin >>i>>j;
cost[i][j]=1;
}
cout <<"enter initial vertex";
cin >>v;
cout <<"ORDER OF VISITED VERTICES";
cout << v <<" ";
visited[v]=1;
k=1;
while(k<n)
{
for(j=n;j>=1;j--)
if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
{
visit[j]=1;
stk[top]=j;
```

```
top++;
}
v=stk[--top];
cout<<v << " ";
k++;
visit[v]=0; visited[v]=1;
}
}
```

OUTPUT

```
enterno of vertices9
ente no of edges9

EDGES
1 2
2 3
2 6
1 5
1 4
4 7
5 7
7 8
8 9
enter initial vertex1
```

ORDER OF VISITED VERTICES1 2 3 6 4 7 8 9 5

/* Write C++ programs to implement the Prim's algorithm to generate a minimum cost spanning tree */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10],u;

main()
{
        int m,c;
        cout <<"enterno of vertices";
        cin >> n;
        cout <<"ente no of edges";
        cin >> m;
        cout <<"\nEDGES Cost\n";
        for(k=1;k<=m;k++)
        {
                cin >>i>>j>>c;
                cost[i][j]=c;
```

```
}
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
        if(cost[i][j]==0)
        cost[i][j]=31999;

cout <<"ORDER OF VISITED VERTICES";
k=1;
while(k<n)
{
        m=31999;
        if(k==1)
        {
                for(i=1;i<=n;i++)
                        for(j=1;j<=m;j++)
                        if(cost[i][j]<m)
                        {
                                m=cost[i][j];
                                u=i;
                        }
        }
        else
        {
```

```
                for(j=n;j>=1;j--)
                if(cost[v][j]<m && visited[j]!=1 && visit[j]!=1)
                {
                        visit[j]=1;
                        stk[top]=j;
                        top++;
                        m=cost[v][j];
                        u=j;
                }
        }
        cost[v][u]=31999;
        v=u;
        cout<<v << " ";
        k++;
        visit[v]=0; visited[v]=1;
    }
}
```

OUTPUT

```
enterno of vertices7
ente no of edges9

EDGES Cost
1 6 10
6 5 25
5 4 22
```

4 3 12
3 2 16
2 7 14
5 7 24
4 7 18
1 2 28
ORDER OF VISITED VERTICES1 6 5 4 3 2

/* Write C++ programs to implement the Kruskal's algorithm to generate a minimum cost spanning tree */

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,m,c,visit,visited[10],l,v,count,count1,vst,p;

main()
{
int dup1,dup2;
cout<<"enter no of vertices";
cin >> n;
cout <<"enter no of edges";
cin >>m;
cout <<"EDGE Cost";
for(k=1;k<=m;k++)
```

```
{
cin >>i >>j >>c;
cost[i][j]=c;
cost[j][i]=c;
}
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]==0)
cost[i][j]=31999;
visit=1;
while(visit<n)
{
v=31999;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]!=31999 && cost[i][j]<v  && cost[i][j]!=-1 )
{
int count =0;
for(p=1;p<=n;p++)
{
if(visited[p]==i || visited[p]==j)
count++;
}
```

```
if(count >= 2)
{
for(p=1;p<=n;p++)
if(cost[i][p]!=31999 && p!=j)
dup1=p;
for(p=1;p<=n;p++)
if(cost[j][p]!=31999 && p!=i)
dup2=p;

if(cost[dup1][dup2]==-1)
continue;
}
l=i;
k=j;
v=cost[i][j];
}
cout <<"edge from " <<l <<"-->"<<k;
cost[l][k]=-1;
cost[k][l]=-1;
visit++;
int count=0;
count1=0;
for(i=1;i<=n;i++)
```

```
{
if(visited[i]==l)
count++;
if(visited[i]==k)
count1++;
}
if(count==0)
visited[++vst]=l;
if(count1==0)
visited[++vst]=k;
}
}
```

OUTPUT

```
enter no of vertices4
enter no of edges4
EDGE Cost
1 2 1
2 3 2
3 4 3
1 3 3
edge from 1-->2edge from 2-->3edge from 1-->3
```

/* Write a C++ program to solve the single source shortest path problem. (Note: Use Dijkstra's algorithm) */

```
#include<iostream>
```

```
#include<conio.h>
#include<stdio.h>
using namespace std;
int shortest(int ,int);
int cost[10][10],dist[20],i,j,n,k,m,S[20],v,totcost,path[20],p;
main()
{
int c;
cout <<"enter no of vertices";
cin >> n;
cout <<"enter no of edges";
cin >>m;
cout <<"\nenter\nEDGE Cost\n";
for(k=1;k<=m;k++)
{
cin >> i >> j >>c;
cost[i][j]=c;
}
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]==0)
cost[i][j]=31999;
cout <<"enter initial vertex";
```

```
cin >>v;
cout << v<<"\n";
shortest(v,n);
 }

int shortest(int v,int n)
{
int min;
for(i=1;i<=n;i++)
{
S[i]=0;
dist[i]=cost[v][i];
}
path[++p]=v;
S[v]=1;
dist[v]=0;
for(i=2;i<=n-1;i++)
{
k=-1;
min=31999;
for(j=1;j<=n;j++)
{
if(dist[j]<min && S[j]!=1)
```

```
{
min=dist[j];
k=j;
}
}
if(cost[v][k]<=dist[k])
p=1;
path[++p]=k;
for(j=1;j<=p;j++)
cout<<path[j];
cout <<"\n";
//cout <<k;
S[k]=1;
for(j=1;j<=n;j++)
if(cost[k][j]!=31999 && dist[j]>=dist[k]+cost[k][j] && S[j]!=1)
 dist[j]=dist[k]+cost[k][j];
}
}
```

OUTPUT

```
enter no of vertices6
enter no of edges11

enter
EDGE Cost
1 2 50
1 3 45
```

```
1 4 10
2 3 10
2 4 15
3 5 30
4 1 10
4 5 15
5 2 20
5 3 35
6 5 3
enter initial vertex1
1
14
145
1452
13
```

/* Write C++ program that uses non-recursive functions to traverse a binary tree in Pre-order */

```
#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class node

{
```

```
public:
class node *left;
class node *right;
int data;
};

class tree: public node
{
public:
int stk[50],top;
node *root;
tree()
{
root=NULL;
top=0;
}
void insert(int ch)
{
        node *temp,*temp1;
        if(root== NULL)
        {
                root=new node;
                root->data=ch;
```

```
            root->left=NULL;
            root->right=NULL;
            return;
        }
        temp1=new node;
        temp1->data=ch;
        temp1->right=temp1->left=NULL;
        temp=search(root,ch);
        if(temp->data>ch)
            temp->left=temp1;
        else
            temp->right=temp1;

}
node *search(node *temp,int ch)
{
        if(root== NULL)
        {
            cout <<"no node present";
            return NULL;
        }
        if(temp->left==NULL && temp->right== NULL)
            return temp;
```

```
        if(temp->data>ch)
            {  if(temp->left==NULL) return temp;
                search(temp->left,ch);}
        else
            { if(temp->right==NULL) return temp;
            search(temp->right,ch);

}           }

void display(node *temp)
{
        if(temp==NULL)
         return ;
        display(temp->left);
      cout<<temp->data <<" ";
        display(temp->right);
}
void preorder( node *root)
{
        node *p,*q;
        p=root;
        q=NULL;
```

```
        top=0;

        while(p!=NULL)
        {
                cout <<p->data  << " ";
                if(p->right!=NULL)
                {
                        stk[top]=p->right->data;
                        top++;
                }
                p=p->left;
                if(p==NULL && top>0)
        {
                p=pop(root);
        }
        }
}

node * pop(node *p)
{
int ch;
ch=stk[top-1];
if(p->data==ch)
```

```
{
top--;
return p;
}
if(p->data>ch)
pop(p->left);
else
pop(p->right);
}
};
main()
{
        tree t1;
        int ch,n,i;
        while(1)
        {
                cout <<"\n1.INSERT\n2.DISPLAY 3.PREORDER TRAVERSE\n4.EXIT\nEnter your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:   cout <<"enter no of elements to insert:";
                        cout<<"\n enter the elements";
```

```
                    cin >> n;
                    for(i=1;i<=n;i++)
                    {  cin >> ch;
                      t1.insert(ch);
                    }
                     break;
            case 2:   t1.display(t1.root);break;
            case 3:   t1.preorder(t1.root); break;
            case 4:   exit(1);
            }
        }
}
```

OUTPUT

```
1.INSERT
2.DISPLAY 3.PREORDER TRAVERSE
4.EXIT
Enter your choice:1
enter no of elements to insert
 enter the elements7
5 24 36 11 44 2 21

1.INSERT
2.DISPLAY 3.PREORDER TRAVERSE
4.EXIT
Enter your choice:2
2 5 11 21 24 36 44
1.INSERT
2.DISPLAY 3.PREORDER TRAVERSE
```

4.EXIT
Enter your choice:3
5 2 24 11 21 36 44
1.INSERT
2.DISPLAY 3.PREORDER TRAVERSE
4.EXIT
Enter your choice:4

/* Write C++ program that uses non-recursive functions to traverse a binary tree in In-order */

```
#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class node

{

public:

class node *left;

class node *right;

int data;

};


class tree: public node
```

```
{
public:
int stk[50],top;
node *root;
tree()
{
root=NULL;
top=0;
}
void insert(int ch)
{
        node *temp,*temp1;
        if(root== NULL)
        {
                root=new node;
                root->data=ch;
                root->left=NULL;
                root->right=NULL;
                return;
        }
        temp1=new node;
        temp1->data=ch;
        temp1->right=temp1->left=NULL;
```

```
        temp=search(root,ch);
        if(temp->data>ch)
                temp->left=temp1;
        else
                temp->right=temp1;

}
node *search(node *temp,int ch)
{
        if(root== NULL)
        {
                cout <<"no node present";
                return NULL;
        }
        if(temp->left==NULL && temp->right== NULL)
                return temp;

        if(temp->data>ch)
            {  if(temp->left==NULL) return temp;
                search(temp->left,ch);}
        else
             { if(temp->right==NULL) return temp;
             search(temp->right,ch);
```

```
}           }

void display(node *temp)
{
        if(temp==NULL)
           return ;
        display(temp->left);
        cout<<temp->data;
        display(temp->right);
}
void inorder( node *root)
{
        node *p;
        p=root;
        top=0;
        do
        {
                while(p!=NULL)
                {
                        stk[top]=p->data;
                        top++;
                        p=p->left;
```

```
            }
            if(top>0)
            {
                  p=pop(root);
                  cout << p->data;
                  p=p->right;
            }
      }while(top!=0 || p!=NULL);
}



node * pop(node *p)
{
      int ch;
      ch=stk[top-1];
      if(p->data==ch)
      {
            top--;
            return p;
      }
      if(p->data>ch)
            pop(p->left);
      else
```

```
            pop(p->right);
}
};


main()
{
        tree t1;
        int ch,n,i;
        while(1)
        {
            cout <<"\n1.INSERT\n2.DISPLAY 3.INORDER
TRAVERSE\n4.EXIT\nEnter your choice:";
            cin >> ch;
            switch(ch)
            {
            case 1:   cout <<"enter no of elements to insert:";
                      cin >> n;
                      for(i=1;i<=n;i++)
                      {  cin >> ch;
                        t1.insert(ch);
                      }
                       break;
            case 2:   t1.display(t1.root);break;
```

```
                case 3:   t1.inorder(t1.root); break;

                case 4:   exit(1);

                }

        }

}
```

OUTPUT

```
1.INSERT
2.DISPLAY 3.INORDER TRAVERSE
4.EXIT
Enter your choice:1
enter no of elements to inser
5 24 36 11 44 2 21

1.INSERT
2.DISPLAY 3.INORDER TRAVERSE
4.EXIT
Enter your choice:3
251121243644
1.INSERT
2.DISPLAY 3.INORDER TRAVERSE
4.EXIT
Enter your choice:3
251121243644
1.INSERT
2.DISPLAY 3.INORDER TRAVERSE
4.EXIT
Enter your choice:4
```

/* Write C++ program that uses non-recursive functions to traverse a binary tree in Post-order */

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>

class node
{
public:
class node *left;
class node *right;
int data;
};

class tree: public node
{
public:
int stk[50],top;
node *root;
tree()
{
root=NULL;
top=0;
}
void insert(int ch)
{
        node *temp,*temp1;
        if(root== NULL)
        {
                root=new node;
                root->data=ch;
                root->left=NULL;
                root->right=NULL;
                return;
        }
```

```
        temp1=new node;
        temp1->data=ch;
        temp1->right=temp1->left=NULL;
        temp=search(root,ch);
        if(temp->data>ch)
                temp->left=temp1;
        else
                temp->right=temp1;

}


node *search(node *temp,int ch)
{
        if(root== NULL)
        {
                cout <<"no node present";
                return NULL;
        }
        if(temp->left==NULL && temp->right== NULL)
                return temp;

        if(temp->data>ch)
            {  if(temp->left==NULL) return temp;
                search(temp->left,ch);}
        else
             { if(temp->right==NULL) return temp;
             search(temp->right,ch);

}            }

void display(node *temp)
{
        if(temp==NULL)
         return ;
        display(temp->left);
      cout<<temp->data << " ";
        display(temp->right);
}
void postorder( node *root)
```

```
{
        node *p;
        p=root;
        top=0;

        while(1)
        {
        while(p!=NULL)
        {
                stk[top]=p->data;
                top++;
                if(p->right!=NULL)
                        stk[top++]=-p->right->data;
                p=p->left;

        }
        while(stk[top-1] > 0 || top==0)
        {
           if(top==0) return;
           cout << stk[top-1] <<" ";
           p=pop(root);
        }
        if(stk[top-1]<0)
        {
          stk[top-1]=-stk[top-1];
          p=pop(root);
     }  }

}
node * pop(node *p)
{
int ch;
ch=stk[top-1];
if(p->data==ch)
{
top--;
return p;
}
if(p->data>ch)
pop(p->left);
```

```
else
pop(p->right);
}
};
void main()
{
        tree t1;
        int ch,n,i;
        clrscr();
        while(1)
        {
                cout <<"\n1.INSERT\n2.DISPLAY 3.POSTORDER
TRAVERSE\n4.EXIT\nEnter your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:   cout <<"enter no of elements to insert:";
                          cout<<"\n enter the elements";
                          cin >> n;
                          for(i=1;i<=n;i++)
                          {  cin >> ch;
                            t1.insert(ch);
                          }
                           break;
                case 2:   t1.display(t1.root);break;
                case 3:   t1.postorder(t1.root); break;
                case 4:   exit(1);
                }
        }
}
```

OUTPUT

1.INSERT

2.DISPLAY 3.POSTORDER TRAVERSE
4.EXIT
Enter your choice:1
enter no of elements to insert:
enter the elements7
5 24 36 11 44 2 21

1.INSERT
2.DISPLAY 3.POSTORDER TRAVERSE
4.EXIT
Enter your choice:2
2 5 11 21 24 36 44
1.INSERT
2.DISPLAY 3.POSTORDER TRAVERSE
4.EXIT
Enter your choice:3
2 21 11 44 36 24 5
1.INSERT
2.DISPLAY 3.POSTORDER TRAVERSE
4.EXIT
Enter your choice:4

/* Write C++ programs for sorting a given list of elements in ascending order using Quick sort sorting methods */

```
#include<iostream.h>
#include<conio.h>
int a[10],l,u,i,j;
void quick(int *,int,int);
void main()
{
clrscr();
cout <<"enter 10 elements";
for(i=0;i<10;i++)
cin >> a[i];
l=0;
u=9;
quick(a,l,u);
cout <<"sorted elements";
for(i=0;i<10;i++)
cout << a[i] << " ";
getch();
}

void quick(int a[],int l,int u)
{
  int p,temp;
  if(l<u)
  {
  p=a[l];
  i=l;
  j=u;
   while(i<j)
  {
    while(a[i] <= p && i<j )
        i++;
    while(a[j]>p && i<=j )
         j--;
    if(i<=j)
    {
    temp=a[i];
    a[i]=a[j];
    a[j]=temp;}
 }
 temp=a[j];
```

```
a[j]=a[l];
a[l]=temp;
cout <<"\n";
for(i=0;i<10;i++)
cout <<a[i]<<" ";
quick(a,l,j-1);
quick(a,j+1,u); }}
```

OUTPUT

enter 10 elements5 2 3 16 25 1 20 7 8 61 14

1 2 3 5 25 16 20 7 8 61
1 2 3 5 25 16 20 7 8 61
1 2 3 5 25 16 20 7 8 61
1 2 3 5 25 16 20 7 8 61
1 2 3 5 25 16 20 7 8 61
1 2 3 5 8 16 20 7 25 61
1 2 3 5 7 8 20 16 25 61
1 2 3 5 7 8 16 20 25 61
1 2 3 5 7 8 16 20 25 61 sorted elements1 2 3 5 7 8 16 20 25 61

/* Write C++ programs for sorting a given list of elements in ascending order using Merge sort the following sorting methods */

```
#include<iostream>
```

```
#include<conio.h>
using namespace std;
void mergesort(int *,int,int);
void merge(int *,int,int,int);
int a[20],i,n,b[20];

main()
{
cout <<"\n enter no of elements";
cin >> n;
cout <<"enter the elements";
for(i=0;i<n;i++)
cin >> a[i];
mergesort(a,0,n-1);
cout <<" numbers after sort";
for(i=0;i<n;i++)
cout << a[i] << " ";
getch();
}

void mergesort(int a[],int i,int j)
{
  int mid;
```

```
  if(i<j)
  {
    mid=(i+j)/2;
    mergesort(a,i,mid);
    mergesort(a,mid+1,j);
    merge(a,i,mid,j);
  }
}
void merge(int a[],int low,int mid ,int high)
{
  int h,i,j,k;
  h=low;
  i=low;
   j=mid+1;
  while(h<=mid && j<=high)
  {
    if(a[h]<=a[j])
        b[i]=a[h++];
    else
        b[i]=a[j++];
    i++;
  }
```

```
  if( h > mid)
     for(k=j;k<=high;k++)
         b[i++]=a[k];
  else
     for(k=h;k<=mid;k++)
          b[i++]=a[k];

  cout <<"\n";
  for(k=low;k<=high;k++)
  {  a[k]=b[k];
    cout << a[k] <<" ";
  }

}
```

OUTPUT

```
N enter no of elements8 12 5 61 60 50 1 70 81
enter the elements
5 12
60 61
5 12 60 61
1 50
70 81
1 50 70 81
1 5 12 50 60 61 70 81  numbers after sort1 5 12 50 60 61 70 81
```

/* Write a C++ program to implement dynamic programming algorithm to solve the all pairs shortest path problem */

```
#include<iostream>
#include<conio.h>
using namespace std;
int min(int a,int b);
int cost[10][10],a[10][10],i,j,k,c;
```

```
main()
{
  int n,m;
  cout <<"enter no of vertices";
  cin >> n;
  cout <<"enter no od edges";
  cin >> m;
  cout<<"enter the\nEDGE Cost\n";
for(k=1;k<=m;k++)
{
cin>>i>>j>>c;
a[i][j]=cost[i][j]=c;
}
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
if(a[i][j]== 0 && i !=j)
a[i][j]=31999;
}
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
```

```
a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
cout <<"Resultant adj matrix\n";
for(i=1;i<=n;i++)
{
for( j=1;j<=n;j++)
  {
  if(a[i][j] !=31999)
  cout << a[i][j] <<" ";
  }
cout <<"\n";
}
getch();
}
int min(int a,int b)
{
if(a<b)
return a;
else
return b; }
```

OUTPUT

```
enter no of vertices3
enter no od edges5
enter the
```

EDGE Cost
1 2 4
2 1 6
1 3 11
3 1 3
2 3 2
Resultant adj matrix
0 4 6
5 0 2
3 7 0

/* Write a C++ program that uses dynamic programming algorithm to solve the optimal binary search tree problem */

```
#include<iostream>

#include<conio.h>

#include<stdio.h>

using namespace std;

#define MAX 10

int find(int i,int j);

void print(int,int);
```

```
int p[MAX],q[MAX],w[10][10],c[10][10],r[10][10],i,j,k,n,m;
char idnt[7][10];

main()
{
        cout << "enter the no, of identifiers";
        cin >>n;
        cout <<"enter identifiers";
        for(i=1;i<=n;i++)
        gets(idnt[i]);
        cout <<"enter success propability for identifiers";
        for(i=1;i<=n;i++)
                cin >>p[i];
        cout << "enter failure propability for identifiers";
        for(i=0;i<=n;i++)
                cin >> q[i];
        for(i=0;i<=n;i++)
        {
                w[i][i]=q[i];
                c[i][i]=r[i][i]=0;
                w[i][i+1]=q[i]+q[i+1]+p[i+1];
                r[i][i+1]=i+1;
                c[i][i+1]=q[i]+q[i+1]+p[i+1];
```

```
        }
        w[n][n]=q[n];
        r[n][n]=c[n][n]=0;
        for(m=2;m<=n;m++)
        {
                for(i=0;i<=n-m;i++)
                {
                    j=i+m;
                    w[i][j]=w[i][j-1]+p[j]+q[j];
                    k=find(i,j);
                    r[i][j]=k;
                    c[i][j]=w[i][j]+c[i][k-1]+c[k][j];
                }
        }
      cout <<"\n";
      print(0,n); }

int find(int i,int j)
{
int min=2000,m,l;
for(m=i+1;m<=j;m++)
if(c[i][m-1]+c[m][j]<min)
{
```

```
min=c[i][m-1]+c[m][j];
l=m;
}
return l;
}
void print(int i,int j)
{
if(i<j)
puts(idnt[r[i][j]]);
else
return;
print(i,r[i][j]-1);
print(r[i][j],j);
}
```

OUTPUT

```
enter the no, of identifiers4
enter identifiersdo
if
int
while
enter success propability for identifiers3 3 1 1
enter failure propability for identifiers2 3 1 1 1

tree in preorder form
if
do
int
```

while

AVL TREE

```
#include <iostream.h>
#include <stdlib.h>
#include<constream.h>
#define FALSE 0
#define TRUE 1
struct AVLNode
{
        int data ;
        int balfact ;
        AVLNode *left ;
        AVLNode *right ;
} ;

class avltree
{
        private :
                AVLNode *root ;
        public :
                avltree( ) ;
                AVLNode*  insert ( int data, int *h ) ;
```

```
                static AVLNode* buildtree ( AVLNode *root, int data, int *h ) ;
                void display( AVLNode *root ) ;
                AVLNode* deldata ( AVLNode* root, int data, int *h ) ;
                static AVLNode* del ( AVLNode *node, AVLNode* root, int *h )
;
                static AVLNode* balright ( AVLNode *root, int *h ) ;
                static AVLNode* balleft ( AVLNode* root, int *h ) ;
                void setroot ( AVLNode *avl ) ;
                ~avltree( ) ;
                static void deltree ( AVLNode *root ) ;
} ;
avltree :: avltree( )
{
        root = NULL ;
}
AVLNode* avltree :: insert ( int data, int *h )
{
        root = buildtree ( root, data, h ) ;
        return root ;
}
AVLNode* avltree :: buildtree ( AVLNode *root, int data, int *h )
{
        AVLNode *node1, *node2 ;

        if ( root == NULL )
        {
                root = new AVLNode ;
                root -> data = data ;
                root -> left = NULL ;
                root -> right = NULL ;
                root -> balfact = 0 ;
                *h = TRUE ;
                return ( root ) ;
        }
        if ( data < root -> data )
        {
                root -> left = buildtree ( root -> left, data, h ) ;

                // If left subtree is higher
                if ( *h )
```

```
                {
                        switch ( root -> balfact )
                        {
                                case 1 :
                                        node1 = root -> left ;
                                        if ( node1 -> balfact == 1 )
                                        {
                                                cout << "\nRight rotation." ;
                                                root -> left = node1 -> right ;
                                                node1 -> right = root ;
                                                root -> balfact = 0 ;
                                                root = node1 ;
                                        }
                                        else
                                        {
                                                cout << "\nDouble rotation, left
then right." ;
                                                node2 = node1 -> right ;
                                                node1 -> right = node2 -> left ;
                                                node2 -> left = node1 ;
                                                root -> left = node2 -> right ;
                                                node2 -> right = root ;
                                                if ( node2 -> balfact == 1 )
                                                        root -> balfact = -1 ;
                                                else
                                                        root -> balfact = 0 ;
                                                if ( node2 -> balfact == -1 )
                                                        node1 -> balfact = 1 ;
                                                else
                                                        node1 -> balfact = 0 ;
                                                root = node2 ;
                                        }
                                        root -> balfact = 0 ;
                                        *h = FALSE ;
                                        break ;

                                case 0 :
                                        root -> balfact = 1 ;
                                        break ;
                                case -1 :
```

```
                                        root -> balfact = 0 ;
                                        *h = FALSE ;
                                }
                        }
                }

                if ( data > root -> data )
                {
                        root -> right = buildtree ( root -> right, data, h ) ;

                        if ( *h )
                        {
                                switch ( root -> balfact )
                                {
                                        case 1 :
                                                root -> balfact = 0 ;
                                                *h = FALSE ;
                                                break ;
                                        case 0 :
                                                root -> balfact = -1 ;
                                                break ;
                                        case -1 :
                                                node1 = root -> right ;
                                                if ( node1 -> balfact == -1 )
                                                {
                                                        cout << "\nLeft rotation." ;
                                                        root -> right = node1 -> left ;
                                                        node1 -> left = root ;
                                                        root -> balfact = 0 ;
                                                        root = node1 ;
                                                }
                                                else
                                                {
                                                        cout << "\nDouble rotation, right
then left." ;
                                                        node2 = node1 -> left ;
                                                        node1 -> left = node2 -> right ;
                                                        node2 -> right = node1 ;
                                                        root -> right = node2 -> left ;
                                                        node2 -> left = root ;
```

```
                                        if ( node2 -> balfact == -1 )
                                                root -> balfact = 1 ;
                                        else
                                                root -> balfact = 0 ;
                                        if ( node2 -> balfact == 1 )
                                                node1 -> balfact = -1 ;
                                        else
                                                node1 -> balfact = 0 ;
                                        root = node2 ;
                                }
                                root -> balfact = 0 ;
                                *h = FALSE ;
                        }
                }
        }
        return ( root ) ;
}
void avltree :: display ( AVLNode* root )
{
        if ( root != NULL )
        {
                display ( root -> left ) ;
                cout << root -> data << "\t" ;
                display ( root -> right ) ;
        }
}
AVLNode* avltree :: deldata ( AVLNode *root, int data, int *h )
{
        AVLNode *node ;
        if ( root -> data == 13 )
                cout << root -> data ;
        if ( root == NULL )
        {
                cout << "\nNo such data." ;
                return ( root ) ;
        }
        else
        {
                if ( data < root -> data )
                {
```

```
                        root -> left = deldata ( root -> left, data, h ) ;
                        if ( *h )
                                root = balright ( root, h ) ;
                }
                else
                {
                        if ( data > root -> data )
                        {
                                root -> right = deldata ( root -> right, data, h ) ;
                                if ( *h )
                                        root = balleft ( root, h ) ;
                        }
                        else
                        {
                                node = root ;
                                if ( node -> right == NULL )
                                {
                                        root = node -> left ;
                                        *h = TRUE ;
                                        delete ( node ) ;
                                }
                                else
                                {
                                        if ( node -> left == NULL )
                                        {
                                                root = node -> right ;
                                                *h = TRUE ;
                                                delete ( node ) ;
                                        }
                                        else
                                        {
                                                node -> right = del ( node ->
right, node, h ) ;
                                                if ( *h )
                                                        root = balleft ( root, h ) ;
                                        }
                                }
                        }
                }
        }
```

```
        return ( root ) ;
}
AVLNode* avltree :: del ( AVLNode *succ, AVLNode *node, int *h )
{
        AVLNode *temp = succ ;

        if ( succ -> left != NULL )
        {
                succ -> left = del ( succ -> left, node, h ) ;
                if ( *h )
                        succ = balright ( succ, h ) ;
        }
        else
        {
                temp = succ ;
                node -> data = succ -> data ;
                succ = succ -> right ;
                delete ( temp ) ;
                *h = TRUE ;
        }
        return ( succ ) ;
}
AVLNode* avltree :: balright ( AVLNode *root, int *h )
{
        AVLNode *temp1, *temp2 ;
        switch ( root -> balfact )
        {
                case 1 :
                        root -> balfact = 0 ;
                        break ;
                case 0 :
                        root -> balfact = -1 ;
                        *h  = FALSE ;
                        break ;
                case -1 :
                        temp1 = root -> right ;
                        if ( temp1 -> balfact <= 0 )
                        {
                                cout << "\nLeft rotation." ;
                                root -> right = temp1 -> left ;
```

```
                        temp1 -> left = root ;
                        if ( temp1 -> balfact == 0 )
                        {
                                root -> balfact = -1 ;
                                temp1 -> balfact = 1 ;
                                *h = FALSE ;
                        }
                        else
                        {
                                root -> balfact = temp1 -> balfact = 0 ;
                        }
                        root = temp1 ;
                }
                else
                {
                        cout << "\nDouble rotation, right then left." ;
                        temp2 = temp1 -> left ;
                        temp1 -> left = temp2 -> right ;
                        temp2 -> right = temp1 ;
                        root -> right = temp2 -> left ;
                        temp2 -> left = root ;
                        if ( temp2 -> balfact == -1 )
                                root -> balfact = 1 ;
                        else
                                root -> balfact = 0 ;
                        if ( temp2 -> balfact == 1 )
                                temp1 -> balfact = -1 ;
                        else
                                temp1 -> balfact = 0 ;
                        root = temp2 ;
                        temp2 -> balfact = 0 ;
                }
        }
        return ( root ) ;
}
AVLNode* avltree :: balleft ( AVLNode *root, int *h )
{
        AVLNode *temp1, *temp2 ;
        switch ( root -> balfact )
        {
```

```
case -1 :
        root -> balfact = 0 ;
        break ;

case 0 :
        root -> balfact = 1 ;
        *h = FALSE ;
        break ;

case 1 :
        temp1 = root -> left ;
        if ( temp1 -> balfact >= 0 )
        {
                cout << "\nRight rotation." ;
                root -> left = temp1 -> right ;
                temp1 -> right = root ;

                if ( temp1 -> balfact == 0 )
                {
                        root -> balfact = 1 ;
                        temp1 -> balfact = -1 ;
                        *h = FALSE ;
                }
                else
                {
                        root -> balfact = temp1 -> balfact = 0 ;
                }
                root = temp1 ;
        }
        else
        {
                cout << "\nDouble rotation, left then right." ;
                temp2 = temp1 -> right ;
                temp1 -> right = temp2 -> left ;
                temp2 -> left = temp1 ;
                root -> left = temp2 -> right ;
                temp2 -> right = root ;
                if ( temp2 -> balfact == 1 )
                        root -> balfact = -1 ;
                else
```

```
                                        root -> balfact = 0 ;
                                    if ( temp2-> balfact == -1 )
                                        temp1 -> balfact = 1 ;
                                    else
                                        temp1 -> balfact = 0 ;
                                    root = temp2 ;
                                    temp2 -> balfact = 0 ;
                                }
        }
        return ( root ) ;
}
void avltree :: setroot ( AVLNode *avl )
{
        root = avl ;
}
avltree :: ~avltree( )
{
        deltree ( root ) ;
}


void avltree :: deltree ( AVLNode *root )
{
        if ( root != NULL )
        {
                deltree ( root -> left ) ;
                deltree ( root -> right ) ;
        }
        delete ( root ) ;
}
void main( )
{
        avltree at ;
        AVLNode *avl = NULL ;
        int h ;
        clrscr();
        avl = at.insert ( 20, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 6, &h ) ;
        at.setroot ( avl ) ;
```

```
        avl = at.insert ( 29, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 5, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 12, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 25, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 32, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 10, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 15, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 27, &h ) ;
        at.setroot ( avl ) ;
        avl = at.insert ( 13, &h ) ;
        at.setroot ( avl ) ;
        cout << endl << "AVL tree:\n" ;
        at.display ( avl ) ;
        avl = at.deldata ( avl, 20, &h ) ;
        at.setroot ( avl ) ;
        avl = at.deldata ( avl, 12, &h ) ;
        at.setroot ( avl ) ;
        cout << endl << "AVL tree after deletion of a node:\n" ;
        at.display ( avl ) ;
        getch();
}
```

B+TREE

```
#include <iostream.h>
#include <stdlib.h>
#include <alloc.h>
```

```
const int MAX = 4 ;
const int MIN = 2 ;
struct btnode
{
        int count ;
        int value[MAX + 1] ;
        btnode *child[MAX + 1] ;
} ;
class btree
{
        private :
                btnode *root ;
        public :
                btree( ) ;
                void insert ( int val ) ;
                int setval ( int val, btnode *n, int *p, btnode **c ) ;
                static btnode * search ( int val, btnode *root, int *pos ) ;
                static int searchnode ( int val, btnode *n, int *pos ) ;
                void fillnode ( int val, btnode *c, btnode *n, int k ) ;
                void split ( int val, btnode *c, btnode *n,
                                int k, int *y, btnode **newnode ) ;
                void del ( int val ) ;
                int delhelp ( int val, btnode *root ) ;
                void clear ( btnode *root, int k ) ;
                void copysucc ( btnode *root, int i ) ;
                void restore ( btnode *root, int i ) ;
                void rightshift ( int k ) ;
                void leftshift ( int k ) ;
                void merge ( int k ) ;
                void show( ) ;
                static void display ( btnode *root ) ;
                static void deltree ( btnode *root ) ;
                ~btree( ) ;
} ;

btree :: btree( )
{
        root = NULL ;
}
void btree :: insert ( int val )
```

```
{
        int i ;
        btnode *c, *n ;
        int flag ;
        flag = setval ( val, root, &i, &c ) ;
        if ( flag )
        {
                n = new btnode ;
                n -> count = 1 ;
                n -> value[1] = i ;
                n -> child[0] = root ;
                n -> child[1] = c ;
                root = n ;
        }
}
int btree :: setval ( int val, btnode *n, int *p, btnode **c )
{
        int k ;
        if ( n == NULL )
        {
                *p = val ;
                *c = NULL ;
                return 1 ;
        }
        else
        {
                if ( searchnode ( val, n, &k ) )
                        cout << endl << "Key value already exists." << endl ;
                if ( setval ( val, n -> child[k], p, c ) )
                {
                        if ( n -> count < MAX )
                        {
                                fillnode ( *p, *c, n, k ) ;
                                return 0 ;
                        }
                        else
                        {
                                split ( *p, *c, n, k, p, c ) ;
                                return 1 ;
                        }
```

```
                }
                return 0 ;
        }
}
btnode * btree :: search ( int val, btnode *root, int *pos )
{
        if ( root == NULL )
                return NULL ;
        else
        {
                if ( searchnode ( val, root, pos ) )
                        return root ;
                else
                        return search ( val, root -> child[*pos], pos ) ;
        }
}
int btree :: searchnode ( int val, btnode *n, int *pos )
{
        if ( val < n -> value[1] )
        {
                *pos = 0 ;
                return 0 ;
        }
        else
        {
                *pos = n -> count ;
                while ( ( val < n -> value[*pos] ) && *pos > 1 )
                        ( *pos )-- ;
                if ( val == n -> value[*pos] )
                        return 1 ;
                else
                        return 0 ;
        }
}
void btree :: fillnode ( int val, btnode *c, btnode *n, int k )
{
        int i ;
        for ( i = n -> count ; i > k ; i-- )
        {
                n -> value[i + 1] = n -> value[i] ;
```

```
                n -> child[i + 1] = n -> child[i] ;
        }
        n -> value[k + 1] = val ;
        n -> child[k + 1] = c ;
        n -> count++ ;
}
void btree :: split ( int val, btnode *c, btnode *n,
                int k, int *y, btnode **newnode )
{
        int i, mid ;

        if ( k <= MIN )
                mid = MIN ;
        else
                mid = MIN + 1 ;

        *newnode = new btnode ;

        for ( i = mid + 1 ; i <= MAX ; i++ )
        {
                ( *newnode ) -> value[i - mid] = n -> value[i] ;
                ( *newnode ) -> child[i - mid] = n -> child[i] ;
        }

        ( *newnode ) -> count = MAX - mid ;
        n -> count = mid ;

        if ( k <= MIN )
                fillnode ( val, c, n, k ) ;
        else
                fillnode ( val, c, *newnode, k - mid ) ;

        *y = n -> value[n -> count] ;
        ( *newnode ) -> child[0] = n -> child[n -> count] ;
        n -> count-- ;
}
void btree :: del ( int val )
{
        btnode * temp ;
```

```
        if ( ! delhelp ( val, root ) )
                cout << endl << "Value " << val << " not found." ;
        else
        {
                if ( root -> count == 0 )
                {
                        temp = root ;
                        root = root -> child[0] ;
                        delete temp ;
                }
        }
}
int btree :: delhelp ( int val, btnode *root )
{
        int i ;
        int flag ;

        if ( root == NULL )
                return 0 ;
        else
        {
                flag = searchnode ( val, root, &i ) ;
                if ( flag )
                {
                        if ( root -> child[i - 1] )
                        {
                                copysucc ( root, i ) ;
                                flag = delhelp ( root -> value[i], root -> child[i] ) ;
                                if ( !flag )
                                        cout << endl << "Value " << val << " not
found." ;
                        }
                        else
                                clear ( root, i ) ;
                }
                else
                        flag = delhelp ( val, root -> child[i] ) ;
                if ( root -> child[i] != NULL )
                {
                        if ( root -> child[i] -> count < MIN )
```

```
                                        restore ( root, i ) ;
                        }
                        return flag ;
                }
}
void btree :: clear ( btnode *root, int k )
{
        int i ;
        for ( i = k + 1 ; i <= root -> count ; i++ )
        {
                root -> value[i - 1] = root -> value[i] ;
                root -> child[i - 1] = root -> child[i] ;
        }
        root -> count-- ;
}
void btree :: copysucc ( btnode *root, int i )
{
        btnode *temp = root -> child[i] ;

        while ( temp -> child[0] )
                temp = temp -> child[0] ;

        root -> value[i] = temp -> value[1] ;
}
void btree :: restore ( btnode *root, int i )
{
        if ( i == 0 )
        {
                if ( root -> child [1] -> count > MIN )
                        leftshift ( 1 ) ;
                else
                        merge ( 1 ) ;
        }
        else
        {
                if ( i == root -> count )
                {
                        if ( root -> child[i - 1] -> count > MIN )
                                rightshift ( i ) ;
                        else
```

```
                        merge ( i ) ;
                }
                else
                {
                        if ( root -> child[i - 1] -> count > MIN )
                                rightshift ( i ) ;
                        else
                        {
                                if ( root -> child[i + 1] -> count > MIN )
                                        leftshift ( i + 1 ) ;
                                else
                                        merge ( i ) ;
                        }
                }
        }
}
void btree :: rightshift ( int k )
{
        int i ;
        btnode *temp ;

        temp = root -> child[k] ;

        for ( i = temp -> count ; i > 0 ; i-- )
        {
                temp -> value[i + 1] = temp -> value[i] ;
                temp -> child[i + 1] = temp -> child[i] ;
        }

        temp -> child[1] = temp -> child[0] ;
        temp -> count++ ;
        temp -> value[1] = root -> value[k] ;
        temp = root -> child[k - 1] ;
        root -> value[k] = temp -> value[temp -> count] ;
        root -> child[k] -> child [0] = temp -> child[temp -> count] ;
        temp -> count-- ;
}
void btree :: leftshift ( int k )
{
        btnode *temp ;
```

```
        temp = root -> child[k - 1] ;
        temp -> count++ ;
        temp -> value[temp -> count] = root -> value[k] ;
        temp -> child[temp -> count] = root -> child[k] -> child[0] ;
        temp = root -> child[k] ;
        root -> value[k] = temp -> value[1] ;
        temp -> child[0] = temp -> child[1] ;
        temp -> count-- ;
        for ( int i = 1 ; i <= temp -> count ; i++ )
        {
                temp -> value[i] = temp -> value[i + 1] ;
                temp -> child[i] = temp -> child[i + 1] ;
        }
}
void btree :: merge ( int k )
{
        btnode *temp1, *temp2 ;
        temp1 = root -> child[k] ;
        temp2 = root -> child[k - 1] ;
        temp2 -> count++ ;
        temp2 -> value[temp2 -> count] = root -> value[k] ;
        temp2 -> child[temp2 -> count] = root -> child[0] ;
        for ( int i = 1 ; i <= temp1 -> count ; i++ )
        {
                temp2 -> count++ ;
                temp2 -> value[temp2 -> count] = temp1 -> value[i] ;
                temp2 -> child[temp2 -> count] = temp1 -> child[i] ;
        }
        for ( i = k ; i < root -> count ; i++ )
        {
                root -> value[i] = root -> value[i + 1] ;
                root -> child[i] = root -> child[i + 1] ;
        }
        root -> count-- ;
        delete temp1 ;
}
void btree :: show( )
{
        display ( root ) ;
```

```
}
void btree :: display ( btnode *root )
{
        if ( root != NULL )
        {
                for ( int i = 0 ; i < root -> count ; i++ )
                {
                        display ( root -> child[i] ) ;
                        cout << root -> value[i + 1] << "\t" ;
                }
                display ( root -> child[i] ) ;
        }
}
void btree :: deltree ( btnode *root )
{
        if ( root != NULL )
        {
                for ( int i = 0 ; i < root -> count ; i++ )
                {
                        deltree ( root -> child[i] ) ;
                        delete ( root -> child[i] ) ;
                }
                deltree ( root -> child[i] ) ;
                delete ( root -> child[i] ) ;
        }
}

btree :: ~btree( )
{
        deltree ( root ) ;
}

void main( )
{
        btree b ;
        int arr[ ] = { 27, 42, 22, 47, 32, 2, 51, 40, 13 } ;
        int sz = sizeof ( arr ) / sizeof ( int ) ;
        for ( int i = 0 ; i < sz ; i++ )
                b.insert ( arr[i] ) ;
        cout << "B-tree of order 5:" << endl ;
```

```
        b.show( ) ;
        b.del ( 22 ) ;
        b.del ( 11 ) ;
        cout << "\n\nB-tree after deletion of values:" << endl ;
        b.show( ) ;
}
```