**Aim :** Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client's telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers.

---

**Program :**

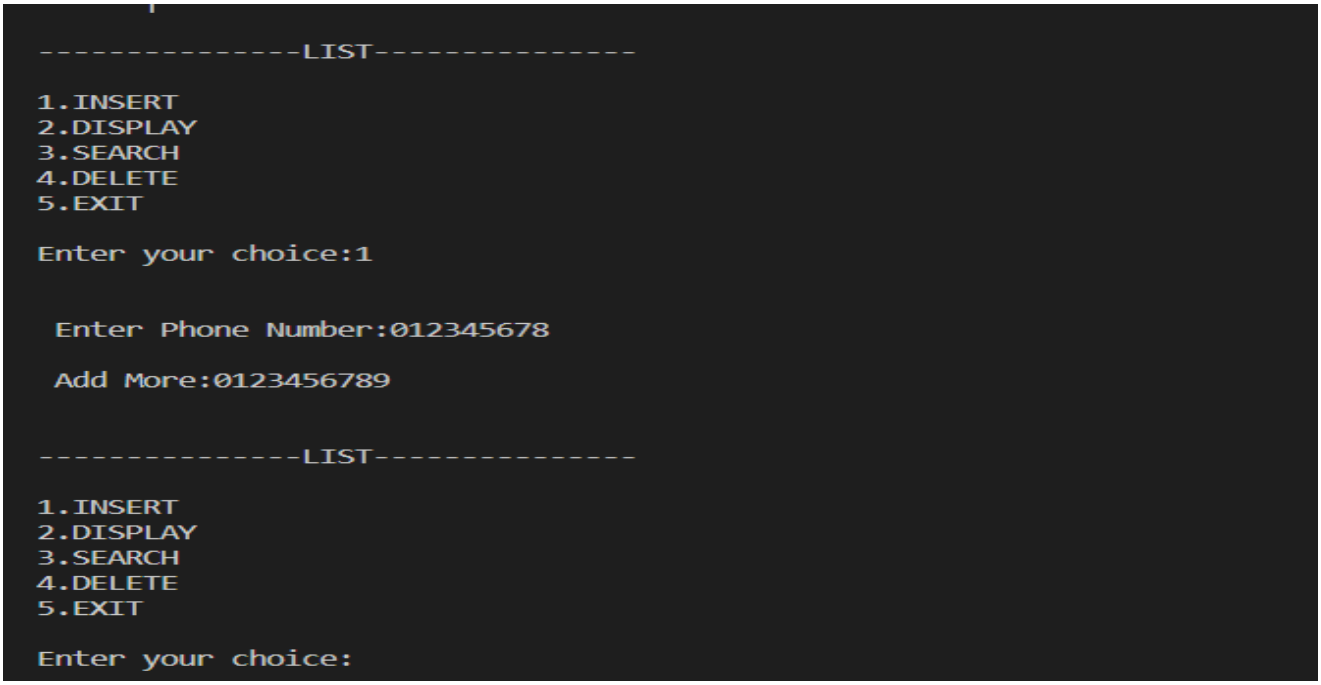
```
 #include<iostream>
#include<cstring>
#define max 10
using namespace std;
struct client
{
long int iPhno;
char name[20];
};//end of structure
class hashtable
{
client ht[max];
public:
hashtable() //constructor
{
for(int i=0;i<max;i++)
{
ht[i].iPhno=0;
}
}
void insert();
void display();
int search(int);
int del(int);
int hash(long int);
};//end of class
void hashtable::insert()
{
client c;
int iPos;
char cAns;
do
{
cout<<"\n Enter Phone Number:";
cin>>c.iPhno;
iPos=hash(c.iPhno);
```

```
if(ht[iPos].iPhno==0)
{
ht[iPos]=c;
}
else
{
for(int i=iPos+1;i%max!=iPos;i++)
{
ht[i]=c;
break;
}
}
cout<<"\n Add More:";
cin>>cAns;
}while(cAns=='y' || cAns=='Y');
}//end of insert
int hashtable::hash(long int key)
{
return(key%max);
}//end of hash
void hashtable::display()
{
cout<<"-----------------------------------";
cout<<"\nSrno\tPhone number\n";
cout<<"-----------------------------------\n";
for(int i=0;i<max;i++)
{
cout<<i<<"\t"<<ht[i].iPhno<<endl;
}
cout<<"-----------------------------------\n";
}//end of display
int hashtable::search(int x)
{
int iFlag=0;
cout<<"Enter Phone number to be searched:";
cin>>x;
for(int i=0;i<max;i++)
{
if(ht[i].iPhno==x)
{
cout<<"\n Phone Number Found at position "<<i;
iFlag=1;
```

```
}
}
if(iFlag==0)
cout<<"\n Phone Number Not Found";
}

//end of search
int hashtable::del(int s)
{
int iF=0;
cout<<"\n Enter phone number to be deleted:";
cin>>s;
for(int i=0;i<max;i++)
{
if(ht[i].iPhno==s)
{
ht[i].iPhno=0;
cout<<"\n Phone number found and deleted";
iF=1;
}
}
if(iF==0)
cout<<"\n Phone number not found";
}//end of del
int main()
{
int y,s,iCh;
hashtable h;
do
{
cout<<"\n----------LIST----------\n";
cout<<"\n1.INSERT\n2.DISPLAY\n3.SEARCH\n4.DELETE\n5.EXIT\n\n";
cout<<"Enter your choice:";
cin>>iCh;
cout<<"\n";
switch(iCh)
{
case 1://insert
h.insert();
cout<<"\n";
break;
case 2://display
```

```
h.display();
cout<<"\n";
break;
case 3://search
h.search(y);
cout<<"\n";
break;
case 4://delete
h.del(s);
cout<<"\n";
break;
case 5://exit
break;
}//end of switch
}while(iCh!=5);//end of do
return 0;
}
```

**Output :**



```
----------------LIST----------------

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

Enter your choice:3

Enter Phone number to be searched:12345678

 Phone Number Found at position 8

----------------LIST----------------

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

Enter your choice:
```

```
---------------LIST---------------

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

Enter your choice:4


 Enter phone number to be deleted:12345678

 Phone number found and deleted

---------------LIST---------------

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

Enter your choice:
```

**Aim :** Implements all the function of a dictionary (ADT) using hashing and handle collision using chaining with / without replacements.Data: Set of (key, value) pairs, keys are mapped to values, key must be comparable, keys must be unique. Standard Operations; Insert(Key ,vlues), Find(Key), Delete(Key).

---

**Program :**

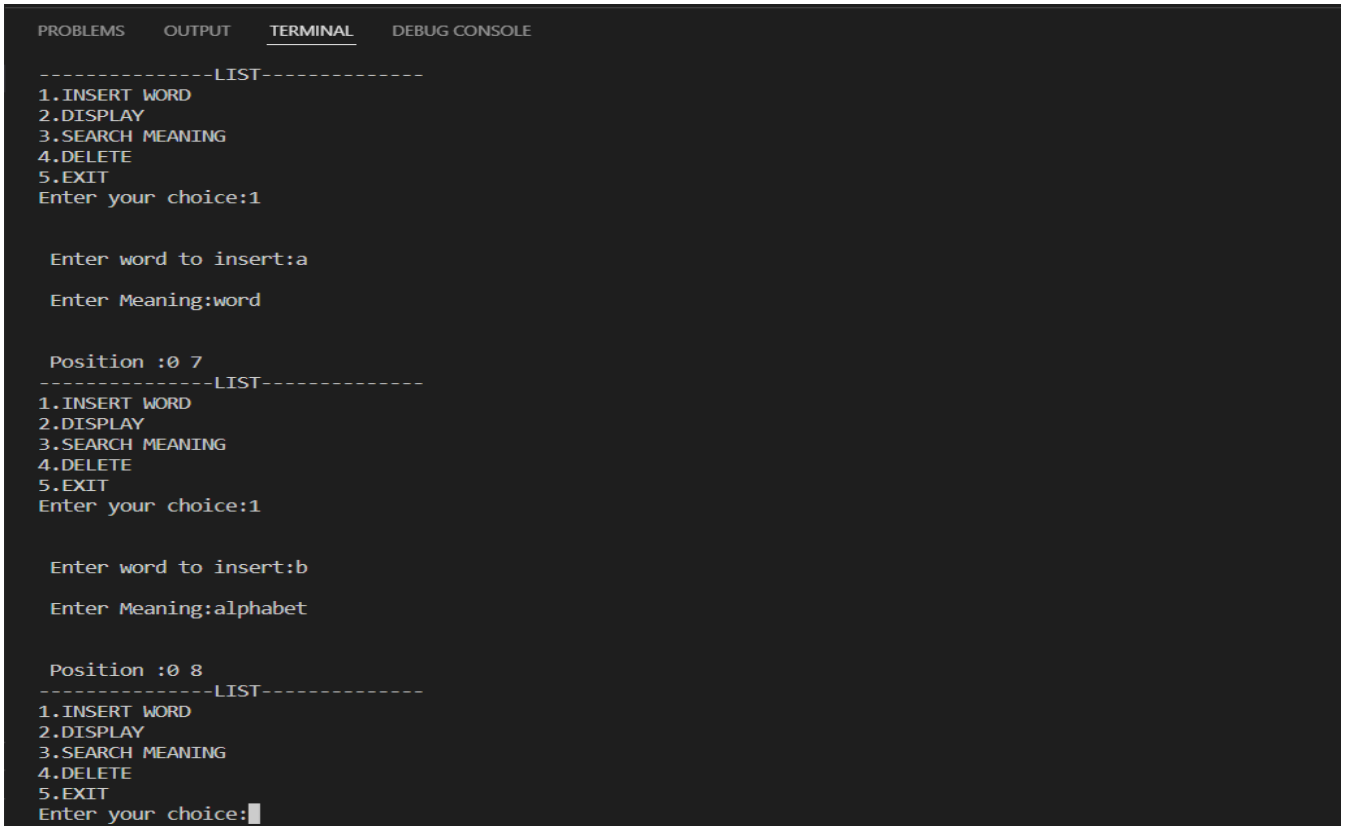
```
#include<iostream>
#include<cstring>
#define max 10
using namespace std;
struct dic
{
char cWord[20];
char cMeaning[20];
};//end of structure
class hashtable
{
public:
dic ht[max];
hashtable() //constructor
{
for(int i=0;i<max;i++)
{
strcpy(ht[i].cWord,"");
strcpy(ht[i].cMeaning,"");
}
}
int hash(char ckey[10])
{
int i,s=0;
for(i=0;ckey[i]!='\0';i++)
{
s=s+ckey[i];
}
return(s%max);
}//end of hash
void insert_word(dic d);
void display();
int search_word(char cW[]);
void del_word(char cW[]);
};//end of class
void hashtable::insert_word(dic d)
{
int iIndex=10;
```

```
for(int i=0;i%max!=iIndex;i=(i+1)%max)
{
iIndex=(hash(d.cWord)+i*i)%max;
cout<<"\n\n Position :"<<i<<" "<<iIndex;
if(i>0)
cout<<"\n Collision at "<<iIndex;
if(strcmp(ht[iIndex].cWord,"")==0)
{
ht[iIndex]=d;
break;
}
}
}//end of insert
void hashtable::display()
{
cout<<"index\t\tWord\t\tmeaning";
for(int i=0;i<max;i++)
{
cout<<"\n"<<i<<"\t\t"<<ht[i].cWord<<"\t\t"<<ht[i].cMeaning<<"\n";
}
}//end of display
int hashtable::search_word(char cW[10])
{
int iIndex,iFlag=0;
for(int i=0;i%max!=iIndex;i=(i+1)%max)
{
iIndex=(hash(cW)+i*i)%max;
if(strcmp(ht[iIndex].cWord,cW)==0)
{
cout<<"\nWord Found and Meaning is :"<<ht[iIndex].cMeaning;
iFlag=1;
break;
}
}
if(iFlag==0)
cout<<"\nWord Not Found";
}//end of search_word
void hashtable::del_word(char cW[10])
{
int iIndex,iFlag=0;
for(int i=0;i%max!=iIndex;i=(i+1)%max)
{
iIndex=(hash(cW)+i*i)%max;
if(strcmp(ht[iIndex].cWord,cW)==0)
{
```

```
cout<<"\nWord Found and deleted :"<<ht[iIndex].cMeaning;
strcpy(ht[iIndex].cWord,"");
strcpy(ht[iIndex].cMeaning,"");
iFlag=1;
break;
}
}
if(iFlag==0)
cout<<"\n Word Not Found";
}//end of del_word
int main()
{
char cW[10];
int iCh,iFlag=0;
hashtable h;
dic d;
do
{
cout<<"\n----------LIST----------\n";
cout<<"1.INSERT WORD\n2.DISPLAY\n3.SEARCH MEANING\n4.DELETE\n5.EXIT\n";
cout<<"Enter your choice:";
cin>>iCh;
cout<<"\n";
switch(iCh)
{
case 1://insert
cout<<"\n Enter word to insert:";
cin>>d.cWord;
cout<<"\n Enter Meaning:";
cin>>d.cMeaning;
h.insert_word(d);
break;
case 2://display
h.display();
break;
case 3://search
cout<<"\n Enter word to be searched:";
cin>>cW;
h.search_word(cW);
break;
case 4://delete
cout<<"\n Enter the word to be deleted:";
cin>>cW;
h.del_word(cW);
break;
```

```
case 5://exit
break;
}//end of switch
}while(iCh!=5);
return 0;
}
```

**Output :**



```
---------------LIST--------------
1.INSERT WORD
2.DISPLAY
3.SEARCH MEANING
4.DELETE
5.EXIT
Enter your choice:2

index           Word            meaning
0

1

2

3

4

5

6

7               a               word

8               b               alphabet

9

---------------LIST--------------
1.INSERT WORD
2.DISPLAY
3.SEARCH MEANING
4.DELETE
5.EXIT
Enter your choice:4


 Enter the word to be deleted:b

Word Found and deleted :alphabet
```

**Aim :** Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node. ii. Find number of nodes in longest path. iii. Minimum data value found in the tree. iv. Change a tree so that the roles of the left and right pointers are swapped at every node. v. Search a value.

**Program :**

```
#include<iostream>
using namespace std;
class node
{
 public:
 int data;
node *left;
node *right;
};
class bst
{
public:
node *root;
bst()
{
root=NULL;
}
void create();
void insert();
void postorder(node*);
void inorder(node *);
 void preorder(node *);
 void search(int key);
 int search(node*, int key);
void minimum();
int height(node*);
};
void bst::minimum()
{
 node *temp;
int min;
 temp=root;
 while(temp->left!=NULL)
 {
 min=temp->data;
```

```
temp=temp->left;
if(temp->data<min)
{
min=temp->data;
}
else
{
temp=temp->left;
}
}
cout<<"minimum no. is:"<<min;
}
int bst::height(node *root)
{
if(root==NULL)
{
return 0;
}
else
{
if(height(root->right)>height(root->left)) //right tree is longer
{
return (1+height(root->right));
}
else
{
return (1+height(root->left));
}
}
}
void bst::create()
{
node *curr,*temp;
int ans=1;
cout<<"enter data:";
do
{
curr=new node;
cin>>curr->data;
curr->left=curr->right=NULL;
if(root==NULL)
{
```

```
root=curr;
}
else
{
temp=root;
while(1)
{
if(curr->data<=temp->data)
{
if(temp->left==NULL)
{
temp->left=curr;
break;
}
else
{
temp=temp->left;
}
}
else
{
if(temp->right==NULL)
{
temp->right=curr;
break;
}
else
{
temp=temp->right;
}
}
}
}
cout<<"want to continue:";
cin>>ans;
}while(ans==1);
}
void bst::inorder(node *root)
{
if(root!=NULL)
{
inorder(root->left);
```

```
cout<<" "<<root->data;
inorder(root->right);
}
}
void bst::preorder(node *root)
{
if(root!=NULL)
{
cout<<" "<<root->data;
preorder(root->left);
preorder(root->right);
}
}
void bst::postorder(node *root)
{
if(root!=NULL)
{
postorder(root->left);
postorder(root->right);
cout<<" "<<root->data;
}
}
void bst::insert()
{
node *curr,*temp;
int ans=1;
cout<<"enter data:";
curr=new node;
cin>>curr->data;
curr->left=curr->right=NULL;
if(root==NULL)
{
root=curr;
}
else
{
temp=root;
while(1)
{
if(curr->data<=temp->data)
{
if(temp->left==NULL)
```

```
{
temp->left=curr;
break;
}
else
{
temp=temp->left;
}
}
else
{
if(temp->right==NULL)
{
temp->right=curr;
break;
}
else
{
temp=temp->right;
}
}
}//end of while
}
}
void bst::search(int key)
{
node *curr;
curr=root;
while(curr!=NULL){
if(curr->data==key){
cout<<"found";
break;
}
else{
if(key<curr->data){
curr=curr->left;
}
else{
curr=curr->right;
}}}
if(curr==NULL) //not found even at the end of the tree.{
cout<<"not found";
```

```
}
}
int main(){
bst b;
int key,ch;
do{
cout<<"\n1.create\n2.insert\n3.inorder\n4.preorder\n5.postorder\n6.search\n7.minimum\n
8.height\npress 0 to exit\n";
cout<<"enter your choice:";
cin>>ch;
switch(ch){
case 1:b.create();
break;
case 2:b.insert();
break;
case 3:cout<<"inorder traversal is\n";
b.inorder(b.root);
break;
case 4:cout<<"preorder traversal is\n";
b.preorder(b.root);
break;
case 5:cout<<"postorder traversal is\n";
b.postorder(b.root);
break;
case 6:cout<<"\nenter key:";
cin>>key;
b.search(key);
break;
case 7:b.minimum();
break;
case 8:cout<<"height of tree: "<<b.height(b.root);
break;}
}while(ch!=0);
return 0;
}
```

## Output :

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:2
enter data:2

1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:2
enter data:5

1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:2
enter data:9

1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:2
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:3
inorder traversal is
 2 5 9
1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:4
preorder traversal is
 2 5 9
1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:5
postorder traversal is
 9 5 2
1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:
```

```
1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:6

enter key:5
found
1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:8
height of tree: 3
1.create
2.insert
3.inorder
4.preorder
5.postorder
6.search
7.minimum
8.height
press 0 to exit
enter your choice:
```

**Aim :** Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.

---

**Program :**

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct node
{
   int data;
   node *left,*right;
   int lbit,rbit;
};
class tbt
{
  node *temp=NULL,*t1=NULL,*s=NULL,*head=NULL,*t=NULL;
  public:
  node *create();
  void insert();
  node *insuc(node*);
  node *inpre(node*);
  void dis();
  void display(node*);
  void thr();
  void thread(node*);
};
node *tbt::create()
{
  node *p=new(struct node);
  p->left=NULL;
  p->right=NULL;
  p->lbit=0;
  p->rbit=0;
  cout<<"\n enter the data";
  cin>>p->data;
  return p;
}
void tbt::insert()
{
   temp=create();
   if(head==NULL)
```

```
  {  node *p=new(struct node);
     head=p;
     head->left=temp;
     head->right=head;
     head->lbit=1;
     head->rbit=0;
     temp->left=head;
     temp->right=head;
     temp->lbit=0;
     temp->rbit=0;
   }else
  {     t1=head;
        t1=t1->left;
        while(t1!=NULL)
        {   s=t1;
            if(((temp->data)>(t1->data))&&t1->rbit==1)
            {   t1=t1->right;   }
            else if(((temp->data)<(t1->data))&&t1->lbit==1)
            {   t1=t1->left;     }
            else
            { break;       }
        }
        if(temp->data>s->data)
        {
          s->right=temp;
          s->rbit=1;
          temp->left=inpre(head->left);
          temp->right=insuc(head->left);
        }
        else
        {
          s->left=temp;
          s->lbit=1;
          temp->left=inpre(head->left);
          temp->right=insuc(head->left);
        }
    }
  }
node *tbt::inpre(node *m){
   if(m->lbit==1)    {
     inpre(m->left);
   }
```

```
   if(m->data==temp->data&&t==NULL)
   {  return head;      }
   if(m->data==temp->data)
   {   return t;        }
   t=m;
   if(m->rbit==1)
   { inpre(m->right);
   }
}
node *tbt::insuc(node *m)
{
   if(m->lbit==1)
   {  t=m;
     insuc(m->left);
   }
   if(m->data==temp->data&&t==NULL)
   {  return head;      }
   if(m->data==temp->data)
   {   return t;        }

   if(m->rbit==1)
   { insuc(m->right);
   }
}
void tbt::dis()
{
  cout<<"\n Elements are :";
   display(head->left);
}
void tbt::display(node *m)
{
      if(m->lbit==1)
      {   display(m->left);            }
      cout<<"\n"<<m->data;
      if(m->rbit==1)
      {    display(m->right);           }
}
void tbt::thr()
{  cout<<"\n thread are";
  thread(head->left);
}
void tbt::thread(node *m)
```

```
{
      if(m->lbit==1)
      {   thread(m->left);             }
      if(m->lbit==0||m->rbit==0)
      {
      cout<<"\n"<<m->data;
      }
      if(m->rbit==1)
      {    thread(m->right);            }
}
int main()
{  tbt t;  int ch;
   while(1)
   {
   cout<<"\n 1.insert data";
   cout<<"\n 2.display all data";
   cout<<"\n 3.display threaded node";
   cout<<"\n 4.exit \n";
   cout<<"\n enter the choice : \n";
   cin>>ch;
   switch(ch)
   {
      case 1:
               t.insert();
               break;
      case 2:
               t.dis();
               cout<<"\n";
               break;
      case 3:
               t.thr();
               break;
      case 4:  exit(0);

      default:
              cout<<"\n invalid entry";
    }
    }
   return 0;
}
```

## Output :

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE



 enter the choice :
1

 enter the data6

 1.insert data
 2.display all data
 3.display threaded node
 4.exit

 enter the choice :
1

 enter the data9

 1.insert data
 2.display all data
 3.display threaded node
 4.exit

 enter the choice :
1

 enter the data7

 1.insert data
 2.display all data
 3.display threaded node
 4.exit

 enter the choice :
1

 enter the data8

 1.insert data
 2.display all data
 3.display threaded node
 4.exit
```

```
 enter the data8

 1.insert data
 2.display all data
 3.display threaded node
 4.exit

 enter the choice :
2

 Elements are :
5
6
7
8
9

 1.insert data
 2.display all data
 3.display threaded node
 4.exit

 enter the choice :
3

 thread are
5
6
7
8
9
 1.insert data
 2.display all data
 3.display threaded node
 4.exit

 enter the choice :
```

**Aim :** There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.

---

**Program :**

```
#include<iostream>
#include<queue>
#include<stack>
using namespace std;
class Graph
{
 string city[10];
 int a[10][10];
 int n;
public:
 void input();
 void display();
 void BFS();
 void DFS();
};
void Graph::input()
{
 cout<<"\nEnter no. of cites: ";
 cin>>n;
 cout<<"\nEnter the names of cities: ";
 for(int i=0 ; i<n ; i++)
 cin >> city[i];
 cout<<"\nEnter the distances: ";
 for(int i=0 ; i<n ; i++)
 for(int j=i ; j<n ; j++)
 {
 if(i==j)
 {
 a[i][j] = 0;
 continue;
 }
 cout<<"\nEnter the distance between " << city[i] <<" and " << city[j]<< " : ";
 cin >> a[i][j];
```

```
 a[j][i] = a[i][j];
 }
}
void Graph::display()
{
 for(int i=0 ; i<n ; i++)
 {
 cout<<"\n";
 for(int j=0 ; j<n ; j++)
 {
 cout<<a[i][j] << "\t";
 }
 }
}
void Graph::BFS()
{
 cout<<"\n\nBFS Traversal: ";
 queue<int> q;
 int visit[n];
 for(int i=0 ; i<n ; i++)
 visit[i] = 0;
 string start;
 int index;
 cout<<"\nEnter starting city: ";
 cin>>start;
 for(int i=0 ; i<n ; i++)
 if(start == city[i])
index =i;
 visit[index] = 1;
 cout<<city[index]<<" -> ";
 int current = index;
 while(1)
 {
 for(int i=0 ; i<n ; i++)
 {
 if(a[current][i]!=0 && visit[i] == 0)
 {
 visit[i] = 1;
 q.push(i);
 cout<<city[i]<<" -> ";
 }
 }
```

```
 if(q.empty()!=0)
 break;
 else
 {
 current = q.front();
 q.pop();
 }
 }
}
void Graph::DFS()
{
 cout<<"\n\nDFS Traversal: ";
 stack<int> s;
 int visit[n];
 for(int i=0 ; i<n ; i++)
 visit[i] = 0;
 string start;
 int index;
 cout<<"\nEnter starting city: ";
 cin>>start;
 for(int i=0 ; i<n ; i++)
 if(start == city[i])
 index =i;
 s.push(index);
 visit[index] = 1;
 int current = index;
 cout << city[index]<<" -> ";
 while(1)
 {
 for(int i=0 ; i<n ; i++)
 {
 if(a[current][i]!=0 && visit[i]==0)
 {
 s.push(i);
 cout<<city[i]<<" -> ";
 visit[i] = 1;
 current = i;
 i=0;
 }
 }
 if(s.empty()!=0)
 break;
```

```
 else
 {
 current = s.top();
 s.pop();
 }
 }
}
int main()
{
 Graph g1;
 int choice;
MENU:
 cout<<"\n\nGRAPH TRAVERSAL";
 cout<<"\n1. Input data";
 cout<<"\n2. Display data";
 cout<<"\n3. DFS Traversal";
 cout<<"\n4. BFS Traversal";
 cout<<"\n5. Exit";
 cout<<"\nEnter your choice: ";
 cin >> choice;
 switch(choice){
 case 1:
 g1.input();
break;
 case 2:
 g1.display();
break;
 case 3:
 g1.DFS();
break;
 case 4:
 g1.BFS();
break;
 case 5:
 return 0;
 default:
 cout<<"\nInvalid choice.Try again!";
 }
 if(choice != 5)
 goto MENU;
 return 0;
}
```

## Output :

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

GRAPH TRAVERSAL
1. Input data
2. Display data
3. DFS Traversal
4. BFS Traversal
5. Exit
Enter your choice: 1

Enter no. of cites: 4

Enter the names of cities: pune
mumbai
kolhapur
delhi

Enter the distances:
Enter the distance between pune and mumbai : 120

Enter the distance between pune and kolhapur : 240

Enter the distance between pune and delhi : 700

Enter the distance between mumbai and kolhapur : 500

Enter the distance between mumbai and delhi : 600

Enter the distance between kolhapur and delhi : 1000


GRAPH TRAVERSAL
1. Input data
2. Display data
3. DFS Traversal
4. BFS Traversal
5. Exit
Enter your choice: 2

0       120     240     700
120     0       500     600
240     500     0       1000
700     600     1000    0
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

GRAPH TRAVERSAL
1. Input data
2. Display data
3. DFS Traversal
4. BFS Traversal
5. Exit
Enter your choice: 3


DFS Traversal:
Enter starting city: pune
pune -> mumbai -> kolhapur -> delhi ->

GRAPH TRAVERSAL
1. Input data
2. Display data
3. DFS Traversal
4. BFS Traversal
5. Exit
Enter your choice: 4


BFS Traversal:
Enter starting city: kolhapur
kolhapur -> pune -> mumbai -> delhi ->

GRAPH TRAVERSAL
1. Input data
2. Display data
3. DFS Traversal
4. BFS Traversal
5. Exit
Enter your choice:
```

**Aim :** You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

---

**Program :**

```
 #include<iostream>
using namespace std;
class Office
{
 int n;
 int a[10][10];
 string office[10];
public:
 void input();
 void display();
 void Prims();
};
void Office::input()
{
 cout<<"\nEnter no. of offices: ";
 cin>>n;
 cout<<"\nEnter the names of offices: ";
 for(int i=0 ; i<n ; i++)
 cin >> office[i];
 cout<<"\nEnter the cost to connect the offices: ";
 for(int i=0 ; i<n ; i++)
 for(int j=i ; j<n ; j++)
 {
 if(i==j)
 {
 a[i][j] = 0;
 continue;
 }
 cout<<"\nEnter the cost to connect " << office[i] <<" and " <<
office[j]<< " : ";
 cin >> a[i][j];
 a[j][i] = a[i][j];
 }
}
void Office::display()
```

```
{
 for(int i=0 ; i<n ; i++)
 {
 cout<<"\n";
 for(int j=0 ; j<n ; j++)
 {
 cout<<a[i][j] << "\t";
 }
 }
}
void Office::Prims()
{
 int visit[n], minCost=0, count=1, minIndex, cost=0;
 for(int i=0 ; i<n ; i++)
 visit[i] = 0;
 cout<<"\n\nShortest path: ";
 visit[0]=1;
 cout<<office[0] << " -> ";
 while(1)
 {
 minCost = 10000;
 for(int i=0 ; i<n ; i++)
 {
 for(int j=0 ; j<n ; j++)
 {
 if(visit[i]==1 && a[i][j]!=0 && a[i][j]< minCost && visit[j]==0) {
 minCost = a[i][j];
 minIndex = j;
 }
 }
 }
 visit[minIndex]=1;
 cout<<office[minIndex] << " -> ";
 cost = cost + minCost;
 count++;
 if(count==n)
 break;
 }
 cout<<"\nMinimum cost: "<<cost;
}
int main()
{
```

```
 Office o1;
 int choice;
MENU:
 cout<<"\n\nMINIMUM SPANNING TREE";
 cout<<"\n1. Input data";
 cout<<"\n2. Display data";
 cout<<"\n3. Calculate minimum cost";
 cout<<"\n4. Exit";
 cout<<"\nEnter your choice: ";
 cin >> choice;
 switch(choice)
 {
 case 1:
 o1.input();
break;
 case 2:
 o1.display();
break;
 case 3:
 o1.Prims();
break;
 case 4:
 return 0;
 default:
 cout<<"\nInvalid choice.Try again!";
 }
 if(choice != 5)
 goto MENU;
 return 0;
}
```

## Output :

```
MINIMUM SPANNING TREE
1. Input data
2. Display data
3. Calculate minimum cost
4. Exit
Enter your choice: 1

Enter no. of offices: 5

Enter the names of offices: Infosys
Wipro
HCL
Cognizant
Accenture

Enter the cost to connect the offices:
Enter the cost to connect Infosys and Wipro : 500

Enter the cost to connect Infosys and HCL : 300

Enter the cost to connect Infosys and Cognizant : 400

Enter the cost to connect Infosys and Accenture : 200

Enter the cost to connect Wipro and HCL : 450

Enter the cost to connect Wipro and Cognizant : 350

Enter the cost to connect Wipro and Accenture : 700

Enter the cost to connect HCL and Cognizant : 150

Enter the cost to connect HCL and Accenture : 250

Enter the cost to connect Cognizant and Accenture : 450


MINIMUM SPANNING TREE
1. Input data
2. Display data
3. Calculate minimum cost
4. Exit
Enter your choice:
```

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE


MINIMUM SPANNING TREE
1. Input data
2. Display data
3. Calculate minimum cost
4. Exit
Enter your choice: 2

0       500     300     400     200
500     0       450     350     700
300     450     0       150     250
400     350     150     0       450
200     700     250     450     0

MINIMUM SPANNING TREE
1. Input data
2. Display data
3. Calculate minimum cost
4. Exit
Enter your choice: 3


Shortest path: Infosys -> Accenture -> HCL -> Cognizant -> Wipro ->
Minimum cost: 950

MINIMUM SPANNING TREE
1. Input data
2. Display data
3. Calculate minimum cost
4. Exit
Enter your choice:
```

**Aim :** Given sequence k = $k_1$ <$k_2$<… <kn of n sorted keys, with a search probability pi for each key $k_i$ . Build the Binary search tree that has the least search cost given the access probability for each key?

---

**Program :**

```
#include <iostream>
#define SIZE 10
using namespace std;
class optimal
{
 public:
 int p[SIZE];
 int q[SIZE];
 int a[SIZE];
 int w[SIZE][SIZE];
 int c[SIZE][SIZE];
 int r[SIZE][SIZE];
 int n;
 int front,rear,queue[20];

 optimal() //default constructor
 {
 front=rear=-1;
 }
 void getdata();
 int minvalue(int,int);
 void OBST();
 void buildtree();
 };
void optimal::getdata()
{
 int i;
 cout<<"\n Optimal Binary search tree";
 cout<<"\n Enter the number of nodes :";
 cin>>n;
 cout<<"\n Enter the data : \n";
 for (i=1;i<=n;i++)
 {
 cout<<"\n a["<<i<<"]:";
 cin>>a[i];
 }
```

```
 cout<<"\n Enter probalities for successful search \n";
 for(i=1;i<=n;i++)
 {
 cout<<"p["<<i<<"]:";
 cin>>p[i];
 }
 cout<<"\n Enter probalities for unsuccessful search \n";
 for(i=1;i<=n;i++)
 {
 cout<<"q["<<i<<"]:";
 cin>>q[i];
 }
 }
 /* This function returns a value in range r[i][j-1] to r[i+1][j] so that cost c[i][k-1]+ c[k][j]
is minimum
*/
 int optimal::minvalue(int i,int j)
 {
 int m,k;
 int min=32000;
 for(m=r[i][j-1];m<=r[i+1][j];m++)
 {
 if((c[i][m-1]+c[m][j])<min)
 {
 min=c[i][m-1]+c[m][j];
 k=m;
 }
 }
 return k;
 }
 /* This function builds table from all given probalities. it basically computes C,r,w value
*/
 void optimal::OBST()
 {
 int i,j,k,m;
 for(i=0;i<n;i++)
 {
 //initialize
 w[i][i]=q[i];
 r[i][i]=c[i][i]=0;
 //optimal trees with one node
 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;
//find optimal trees with m nodes
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=minvalue(i,j);
c[i][j]=w[i][j]+c[i][k-1]+c[k][j];
r[i][j]=k;
}
}
}
/* This function builds tree from table made by OBST function */
void optimal::buildtree()
{
int i,j,k;
cout<<"\n The optimal Binary search tree for given nodes is : \n";
cout<<"\n The root of this OBST is :"<<r[0][n];
cout<<"\n The cost of this OBST is: "<<c[0][n];
cout<<"\n\n Node \t Left child \t Right child";
cout<<"\n _____"<<endl;
queue[++rear]=0;
queue[++rear]=n;
while(front!=rear)
{
i=queue[++front];
j=queue[++front];
k=r[i][j];
cout<<"\n\t"<<k;
if(r[i][k-1]!=0)
{
cout<<" "<<r[i][k-1];
queue[++rear]=i;
queue[++rear]=k-1;
}
else
```

```
            cout<<" ";
            if(r[k][j]!=0)
            {
            cout<<" "<<r[k][j];
            queue[++rear]=k;
            queue[++rear]=j;
            }
            else
            cout<<" ";
            }
            cout<<endl;
            }
            /* This is main function */
            int main() {
            optimal obj;
            obj.getdata();
            obj.OBST();
            obj.buildtree();
            return 0;
            }
```

**Output :**

```
 Optimal Binary search tree
 Enter the number of nodes :4

 Enter the data :

 a[1]:1

 a[2]:2

 a[3]:3

 a[4]:4

 Enter probalities for successful search
p[1]:3
p[2]:3
p[3]:1
p[4]:1

 Enter probalities for unsuccessful search
q[1]:2
q[2]:3
q[3]:1
q[4]:1

 The optimal Binary search tree for given nodes is :

 The root of this OBST is :2
 The cost of this OBST is: 30

 Node    Left child      Right child
 ------------------------------------

       2 1 3
       1
       3  4
       4
PS C:\Users\Admin\OneDrive\Desktop\New folder> cd "c:\Users\Admin\OneDrive\Desktop\New folder\"
```

**Aim :** A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.

---

**Program :**

```
#include<iostream>
#include<cstring>
#include<cstdlib>
#define MAX 50
#define SIZE 20
using namespace std;
struct AVLnode
{
public:
char cWord[SIZE], cMeaning[MAX];
AVLnode *left,*right;
int iB_fac,iHt;
};
class AVLtree
{
public:
AVLnode *root;
AVLtree()
{
root=NULL;
}
int height(AVLnode*);
int bf(AVLnode*);
AVLnode* insert(AVLnode*,char[SIZE],char[MAX]);
AVLnode* rotate_left(AVLnode*);
AVLnode* rotate_right(AVLnode*);
AVLnode* LL(AVLnode*);
AVLnode* RR(AVLnode*);
AVLnode* LR(AVLnode*);
AVLnode* RL(AVLnode*);
AVLnode* delet(AVLnode*,char x[SIZE]);
void inorder(AVLnode*);
};
AVLnode *AVLtree::delet(AVLnode *curr,char x[SIZE])
{
AVLnode *temp;
if(curr==NULL)
```

```
return(0);
else
if(strcmp(x,curr->cWord)>0)
{
curr->right=delet(curr->right,x);
if(bf(curr)==2)
if(bf(curr->left)>=0)
curr=LL(curr);
else
curr=LR(curr);
}
else
if(strcmp(x,curr->cWord)<0)
{
curr->left=delet(curr->left,x);
if(bf(curr)==-2)
if(bf(curr->right)<=0)
curr=RR(curr);
else
curr=RL(curr);
}
else
{
if(curr->right!=NULL)
{
temp=curr->right;
while(temp->left!=NULL)
temp=temp->left;
strcpy(curr->cWord,temp->cWord);
curr->right=delet(curr->right,temp->cWord);
if(bf(curr)==2)
if(bf(curr->left)>=0)
curr=LL(curr);
else
curr=LR(curr);
}
else
return(curr->left);
}
curr->iHt=height(curr);
return(curr);
}
AVLnode* AVLtree :: insert(AVLnode*root,char newword[SIZE],char newmeaning[MAX]){
if(root==NULL)
{
```

```
root=new AVLnode;
root->left=root->right=NULL;
strcpy(root->cWord,newword);
strcpy(root->cMeaning,newmeaning);
}
else if(strcmp(root->cWord,newword)!=0)
{
if(strcmp(root->cWord,newword)>0)
{
root->left=insert(root->left,newword,newmeaning);
if(bf(root)==2)
{
if (strcmp(root->left->cWord,newword)>0)
root=LL(root);
else
root=LR(root);
}
}
else if(strcmp(root->cWord,newword)<0)
{
root->right=insert(root->right,newword,newmeaning);
if(bf(root)==-2)
{
if(strcmp(root->right->cWord,newword)>0)
root=RR(root);
else
root=RL(root);
}
}
}
else
cout<<"\nRedundant AVLnode";
root->iHt=height(root);
return root;
}
int AVLtree :: height(AVLnode* curr)
{
int lh,rh;
if(curr==NULL)
return 0;
if(curr->right==NULL && curr->left==NULL)
return 0;
else
{
lh=lh+height(curr->left);
```

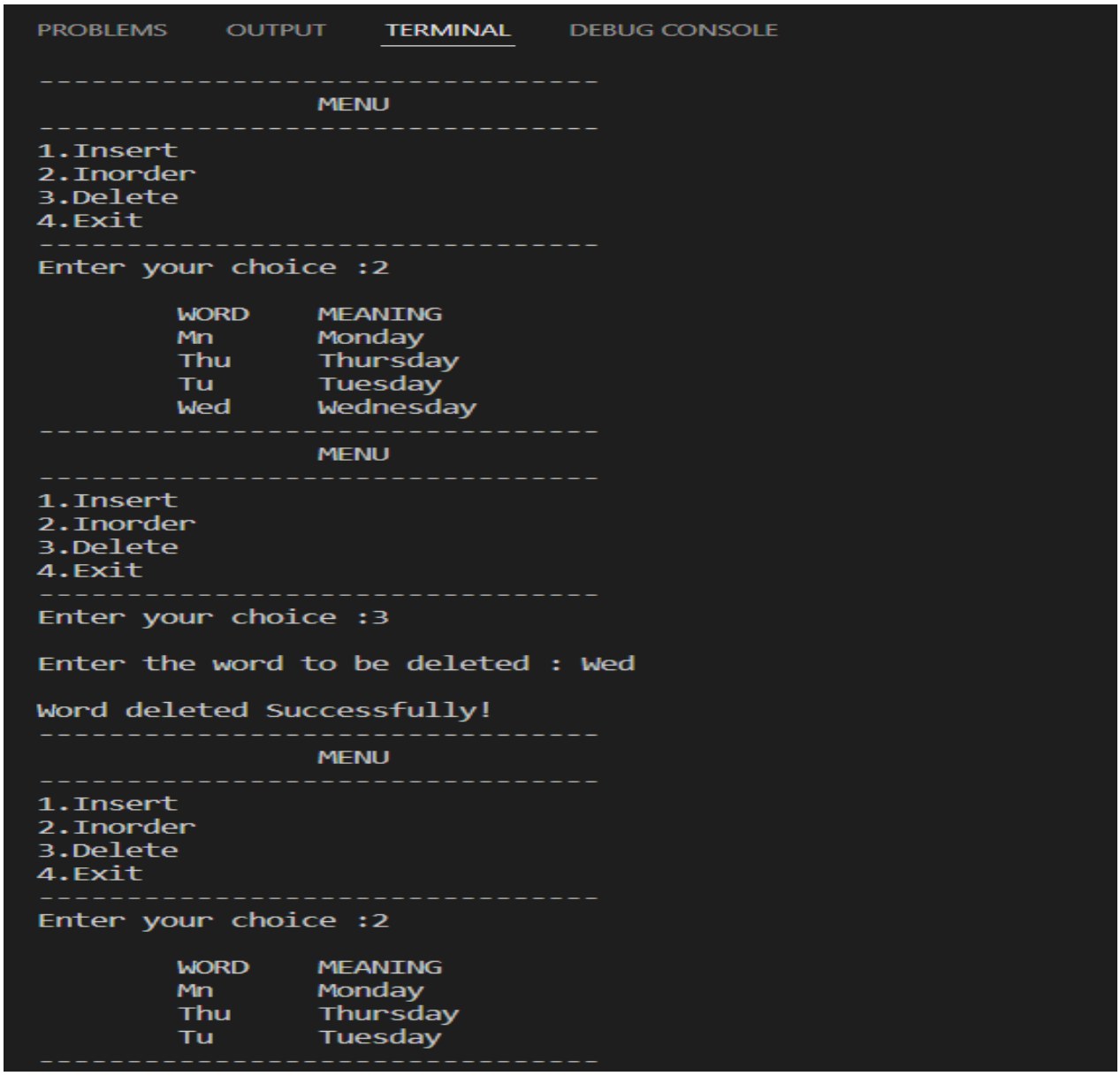
```
rh=rh+height(curr->right);
if(lh>rh)
return lh+1;
return rh+1;
}}
int AVLtree :: bf(AVLnode* curr){
int lh,rh;
if(curr==NULL)
return 0;
else{
if(curr->left==NULL)
lh=0;
else
lh=1+curr->left->iHt;
if(curr->right==NULL)
rh=0;
else
rh=1+curr->right->iHt;
return(lh-rh);
}}
AVLnode* AVLtree :: rotate_right(AVLnode* curr){
AVLnode* temp;
temp=curr->left;
curr->left=temp->right;
temp->left=curr;
curr->iHt=height(curr);
temp->iHt=height(temp);
return temp;
}
AVLnode* AVLtree :: rotate_left(AVLnode* curr){
AVLnode* temp;
temp=curr->right;
curr->right=temp->left;
temp->left=curr;
curr->iHt=height(curr);
temp->iHt=height(temp);
return temp;
}
AVLnode* AVLtree :: RR(AVLnode* curr){
curr=rotate_left(curr);
return curr;
}
AVLnode* AVLtree :: LL(AVLnode* curr){
curr=rotate_right(curr);
return curr;
```

```
}
AVLnode* AVLtree :: RL(AVLnode* curr){
curr->right=rotate_right(curr->right);
curr=rotate_left(curr);
return curr;
}
AVLnode* AVLtree::LR(AVLnode* curr){
curr->left=rotate_left(curr->left);
curr=rotate_right(curr);
return curr;
}
void AVLtree :: inorder(AVLnode* curr){
if(curr!=NULL){
inorder(curr->left);
cout<<"\n\t"<<curr->cWord<<"\t"<<curr->cMeaning;
inorder(curr->right);
}}
int main(){
int iCh;
AVLtree a;
AVLnode *curr=NULL;
char cWd[SIZE],cMean[MAX];
cout<<"\n-------------------------------------";
cout<<"\n\tAVL TREE IMPLEMENTATION";
cout<<"\n-------------------------------------";
do
{ cout<<"\n--------------------------------";
cout<<"\n\t\tMENU";
cout<<"\n--------------------------------";
cout<<"\n1.Insert\n2.Inorder\n3.Delete\n4.Exit";
cout<<"\n--------------------------------";
cout<<"\nEnter your choice :";
cin>>iCh;
switch(iCh){
case 1: cout<<"\nEnter Word : ";
cin>>cWd;
/*for(int i;cMean[i]!='\0';i++){
if(cMean[i]>='A'&& cMean[i]<='Z'){
cMean[i]=cMean[i]+32;
}}
cout<<cMean;*/
cout<<"\nEnter Meaning : ";
cin.ignore();
cin.getline(cMean,MAX);
a.root=a.insert(a.root,cWd,cMean);
```

```
break;
case 2: cout<<"\n\tWORD\tMEANING";
a.inorder(a.root);
break;
case 3: cout<<"\nEnter the word to be deleted : ";
cin>>cWd;
curr=a.delet(a.root,cWd);
if(curr==NULL)
cout<<"\nWord not present!";
else
cout<<"\nWord deleted Successfully!";curr=NULL;
break;
case 4: exit(0);
}
}while(iCh!=4);
return 0;}
```

**Output :**

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE


----------------------------------------
         AVL TREE IMPLEMENTATION
----------------------------------------
---------------------------------
                  MENU
---------------------------------
1.Insert
2.Inorder
3.Delete
4.Exit
---------------------------------
Enter your choice :1

Enter Word : Mn

Enter Meaning : Monday

---------------------------------
                  MENU
---------------------------------
1.Insert
2.Inorder
3.Delete
4.Exit
---------------------------------
Enter your choice :1

Enter Word : Tu

Enter Meaning : Tuesday

---------------------------------
                  MENU
---------------------------------
1.Insert
2.Inorder
3.Delete
4.Exit
---------------------------------
Enter your choice :1

Enter Word : Wed

Enter Meaning : Wednesday
```


PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
---
MENU
---
1.Insert
2.Inorder
3.Delete
4.Exit
---
Enter your choice :2
WORD MEANING
Mn Monday
Thu Thursday
Tu Tuesday
Wed Wednesday
---
MENU
---
1.Insert
2.Inorder
3.Delete
4.Exit
---
Enter your choice :3
Enter the word to be deleted : Wed
Word deleted Successfully!
---
MENU
---
1.Insert
2.Inorder
3.Delete
4.Exit
---
Enter your choice :2
WORD MEANING
Mn Monday
Thu Thursday
Tu Tuesday
---

**Aim :** Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language .

---

**Program :**

```
import java.util.Scanner;
import java.io.*;
public class shell
{
	static void display(int arr[])
	{
		int i;
		int n=arr.length;
	for(i=0;i<n;i++)
		System.out.print(arr[i]+" ");
	System.out.println();
	}
	static int sort(int arr[])
	{
		int i,j,gap,temp;
		int n=arr.length;
		for(gap=n/2;gap>0;gap/=2)
		{
			for(i=gap;i<n;i++)
			{
				for(j=i-gap;j>=0 && arr[j]>arr[j+gap];j-=gap)
				{
					temp=arr[j];
					arr[j]=arr[j+gap];
					arr[j+gap]=temp;
				}
			}
		System.out.print("Pass "+gap+":");
		for(int k=0;k<n;k++)
			System.out.print(" "+arr[k]+" ");
		System.out.println();
		}
		return 0;
	}
	public static void main(String[] args)
	{
```

```
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the no. of elements:");
                int n=sc.nextInt();
                int arr[]=new int[n];
                int i;
                System.out.println("Enter the elements:");
                for(i=0;i<n;i++)
                        arr[i]=sc.nextInt();
                System.out.println("Array before sorting:");
                display(arr);
                sort(arr);
                System.out.println("\nAfter sorting :");
                display(arr);
        }
}
```

**Output :**

Output Clear

```
java -cp /tmp/cSOZ4vFu78 shell
Enter the no. of elements:5
Enter the elements:23
65
96
32
45
Array before sorting:
23 65 6 32 45
Pass 2:6  32 23 65  45
Pass 1: 6  23  32  45  65

After sorting :
6 23 32 45 65
```

**Aim :** Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

---

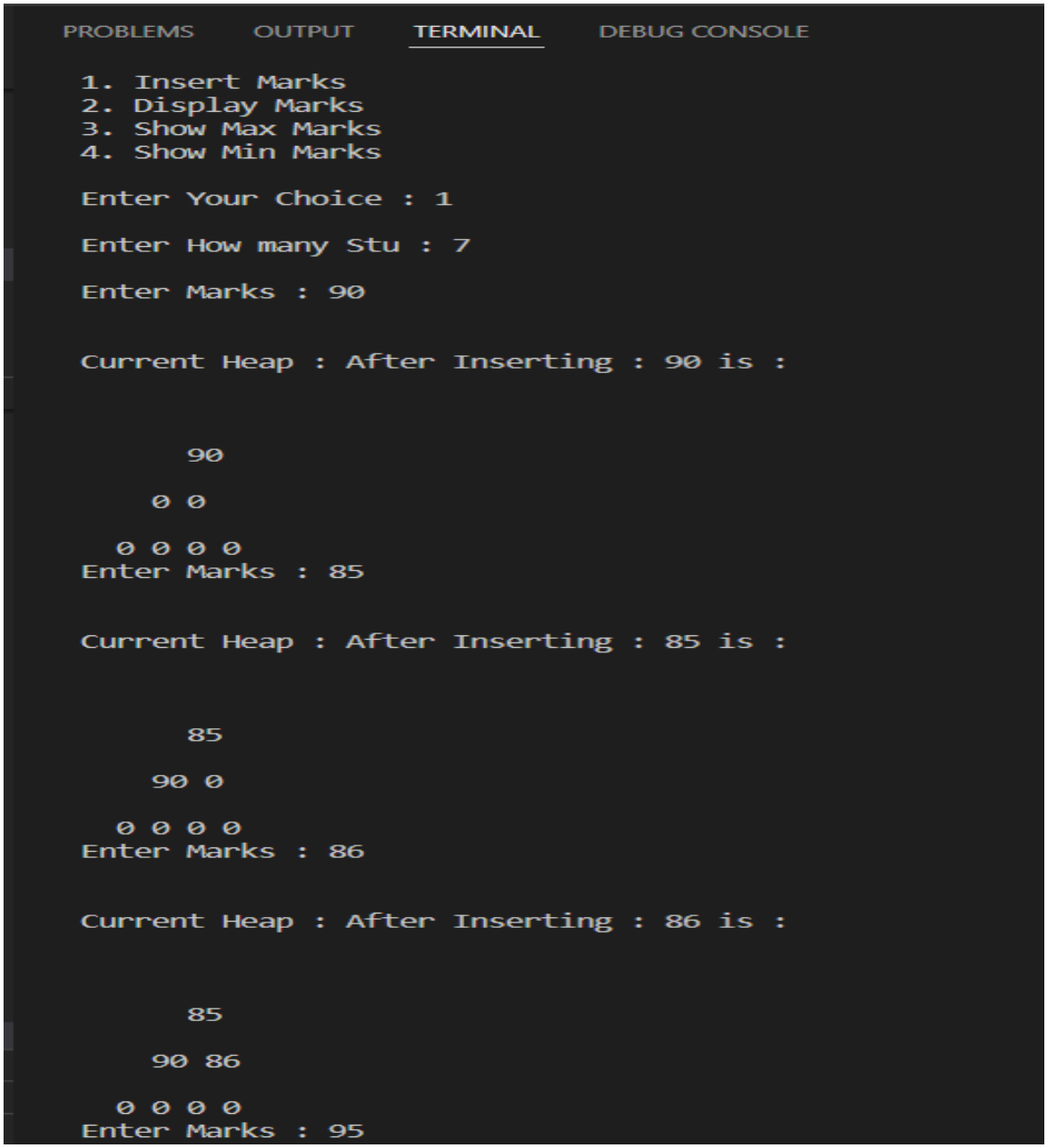
**Program :**

```
#include<iostream>
#include<math.h>
using namespace std;
# define max1 20
class stud{
public:
int marks[max1],total;
 stud(){
 for(int i=0;i<max1;i++)
 marks[i]=0;
 }
void createHeap();
void displayHeap();
void showmax();
void showmin();
};
void stud::createHeap(){
int i,j,par,temp,M;
cout<<"\n Enter How many Stu : ";
cin>>total; //5
for(i=0;i<total;i++) //0-4=5 times{
cout<<"\n Enter Marks : ";
cin>>marks[i];
M=marks[i];
j=i;//j is child
par=floor((j-1)/2);
 while(marks[j] < marks[par] && j!=0) {
 temp=marks[j];
 marks[j]=marks[par];
 marks[par]=temp;
 j=par;
 par=floor((j-1)/2);
 }
cout<<"\n \n Current Heap : After Inserting : " <<M<<" is : \n ";
displayHeap();
```

```
}}
void stud::displayHeap(){
int i=0,space=6;
cout<<endl;
while(i<total){
 if(i==0 || i==1 || i==3 || i==7 || i==15) {
 cout<<endl<<endl;
 for(int j=0;j<space;j++)
 cout<<" ";
 space-=2;
 }
 cout<<" "<<marks[i];i++;
 }}
void stud::showmin(){
cout<<marks[0];
}
void stud::showmax(){
int max,i;
max=marks[0];
for(i=1;i<total;i++){
if(max < marks[i])
max=marks[i];
}
cout<<max;
}
int main(){
stud s1;
int ch, ans;
do{
cout<<"\n 1. Insert Marks ";
cout<<"\n 2. Display Marks ";
cout<<"\n 3. Show Max Marks ";
cout<<"\n 4. Show Min Marks ";
cout<<"\n\n Enter Your Choice : ";
cin>>ch;
switch(ch){
case 1:
s1.createHeap();
break;
case 2:
s1.displayHeap();
break;
```

```
case 3: s1.showmax();
break;
case 4: s1.showmin();
break;
}
cout<<" \n Do u want to continue : (1 for continue )";
cin>>ans;
}while(ans==1);
return 0;
}
```

**Output :**




PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
1. Insert Marks
2. Display Marks
3. Show Max Marks
4. Show Min Marks
Enter Your Choice : 2
80
85 82
95 90 86 98
Do u want to continue : (1 for continue )1
1. Insert Marks
2. Display Marks
3. Show Max Marks
4. Show Min Marks
Enter Your Choice : 3
98
Do u want to continue : (1 for continue )1
1. Insert Marks
2. Display Marks
3. Show Max Marks
4. Show Min Marks
Enter Your Choice : 4
80
Do u want to continue : (1 for continue )1
1. Insert Marks
2. Display Marks
3. Show Max Marks
4. Show Min Marks
Enter Your Choice :

**Aim :** Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

---

**Program :**

```
#include<iostream>
#include<fstream>
#include<string.h>
using namespace std;
class Student
{
int rno;
char sname[20];
char subject[20];
int sub_code;
float internal;
float university;
public:
Student()
{
rno=0;
strcpy(sname,"\0");
strcpy(subject,"\0");
sub_code=0;
internal=0.0;
university=0.0;
}
int acceptRollno()
{
return(rno);
}
void acceptDetails()
{
cout<<"\nEnter Details : ";
cout<<"\nRoll no : ";
cin>>rno;
cout<<"Sname : ";
cin>>sname;
cout<<"Subject Code : ";
cin>>sub_code;
cout<<"Subject Sname : ";
cin>>subject;
```

```
cout<<"Internal AssessMent Marks : ";
cin>>internal;
cout<<"University Exam Marks : ";
cin>>university;
}
void putData()
{
cout<<"\nRoll No. : ";
cout<<rno;
cout<<"\t\tSname : ";
cout<<sname;
cout<<"\nSubject Code : ";
cout<<sub_code;
cout<<"\tSubject Sname : ";
cout<<subject;
cout<<"\nInternal AssessMent Marks : ";
cout<<internal;
cout<<"\nUniversity Exam Marks : ";
cout<<university<<"\n\n";
}
};
class Operations
{
ifstream fin;
ofstream fout;
fstream fs;
public:
void addRecord( );
void show();
void search(int );
int DelRecord(int );
int edit(int );
};
void Operations::addRecord()
{
Student r;
r.acceptDetails();
fout.open("Data",ios::ate|ios::app);
fout.write((char *)&r,sizeof(r));
fout.close();
}
void Operations::show()
{
Student r;
fin.open("Data");
```

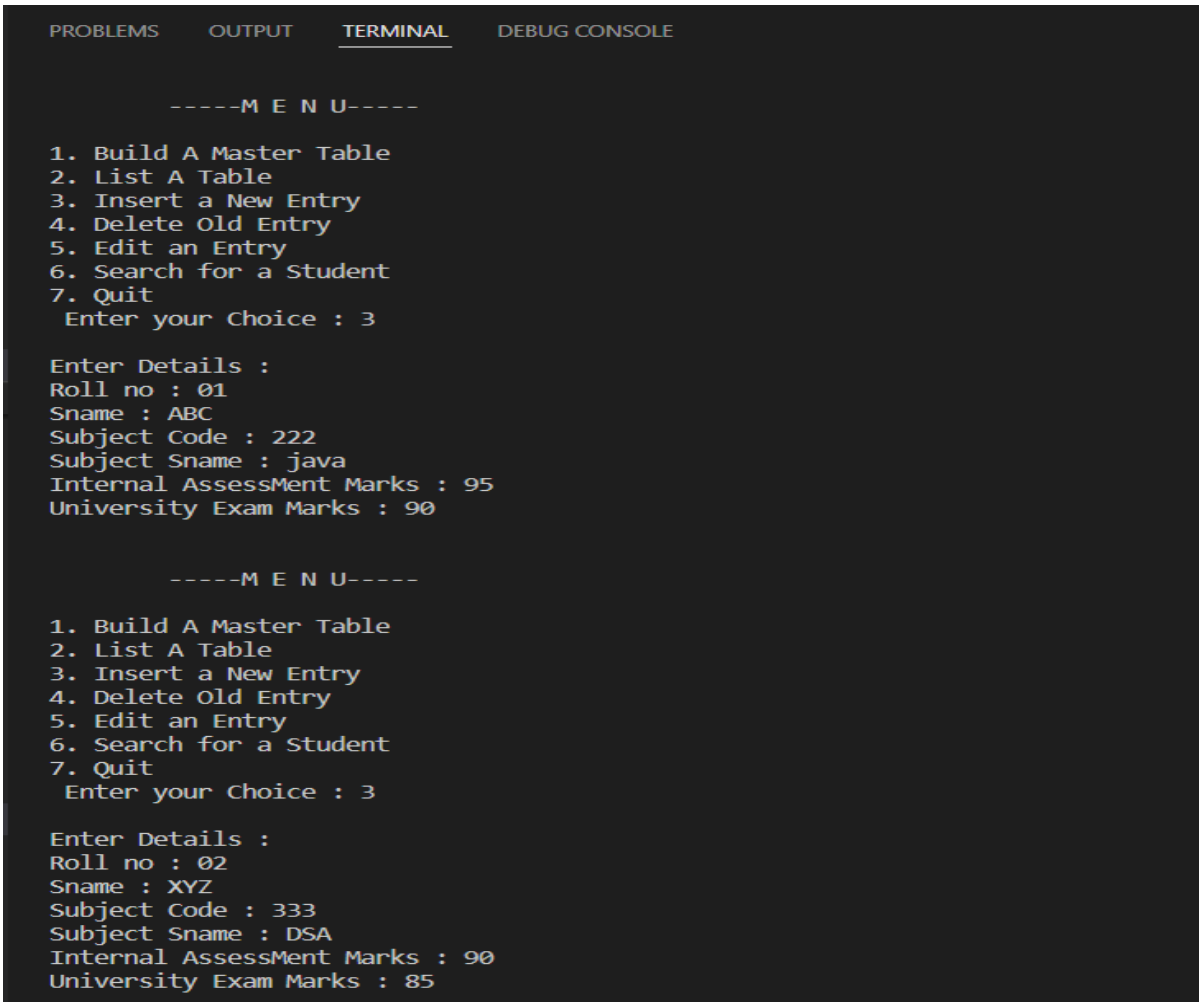
```
fin.seekg(0, ios::beg);
while(fin.read((char *)&r,sizeof(r)))
{
r.putData();
}
fin.close();
}
void Operations::search(int rno)
{
Student r;
int flag=0;
fin.open("Data");
fin.seekg(0, ios::beg);
while(fin.read((char *)&r,sizeof(r)))
{
if(r.acceptRollno()==rno)
{
flag=1;
break;
}
}
fin.close();
if(flag==1)
{
cout<<"\nStudent Found :";
r.putData();
}
else
{
cout<<"\nStudent not Found ";
}
}
int Operations::DelRecord(int rno)
{
Student r;
int flag=0;
fin.open("Data");
fout.open("Temp",ios::ate|ios::app);
fin.seekg(0, ios::beg);
while(fin.read((char *)&r,sizeof(r)))
{
if(r.acceptRollno()==rno)
{
flag=1;
}
```

```
else
{
fout.write((char *)&r,sizeof(r));
}
}
fin.close();
fout.close();
remove("Data");
rename("Temp","Data");
return(flag);
}
int Operations::edit(int rno)
{
Student r;
int flag=0;
fs.open("Data");
fs.seekg(0, ios::beg);
while(fs.read((char *)&r,sizeof(r)))
{
if(r.acceptRollno()==rno)
{
flag=1;
cout<<"\nEnter New Details : ";
r.acceptDetails();
fs.seekp((int)fs.tellg()-sizeof(r),ios::beg);
fs.write((char *)&r,sizeof(r));
}
}
fs.close();
return(flag);
}
int main()
{
Operations f;
int ch,n,i,flag=0;
do
{
cout<<"\n\n\t-----M E N U-----";
cout<<"\n\n1. Build A Master Table";
cout<<"\n2. List A Table";
cout<<"\n3. Insert a New Entry";
cout<<"\n4. Delete Old Entry";
cout<<"\n5. Edit an Entry";
cout<<"\n6. Search for a Student";
cout<<"\n7. Quit";
```

```
cout<<"\n Enter your Choice : ";
cin>>ch;
switch(ch)
{
case 1:if(flag==0)
{
cout<<"\n Enter No of Students to insert : ";
cin>>n;
for(i=0; i<n; i++)
{
f.addRecord();
}
flag=1;
}
else
{
cout<<"\n Sorry.. Table is Already build... \n If want to add Student please select Insert a New
Entry in option.....";
}
break;
case 2:f.show();
break;
case 3:f.addRecord();
break;
case 4:cout<<"\nEnter Roll No of Student Whoes Student is to be Deleted : ";
cin>>n;
i=f.DelRecord(n);
if(i==1)
{
cout<<"\nStudent Deleted Successfully";
}
else
{
cout<<"\nStudent not Found ";
}
break;
case 5:cout<<"\nEnter Roll No of Student Whoes Student is to be Edit : ";
cin>>n;
i=f.edit(n);
if(i==1)
{
cout<<"\nStudent Modified Successfully";
}
else
{
```

```
cout<<"\nStudent not Found ";
}
break;
case 6:cout<<"\nEnter Roll No of Student to be Searched :";
cin>>n;
f.search(n);
break;
case 7:
break;
default:cout<<"\n Invalid Choice.....";
}
}while(ch!=7);
return(0);
}
```

## Output :



```
        -----M E N U-----

1. Build A Master Table
2. List A Table
3. Insert a New Entry
4. Delete Old Entry
5. Edit an Entry
6. Search for a Student
7. Quit
 Enter your Choice : 6

Enter Roll No of Student to be Searched :02

Student Found :
Roll No. : 2            Sname : xyz
Subject Code : 666      Subject Sname : cpp
Internal AssessMent Marks : 85
University Exam Marks : 80
```

**Aim :** Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

---

**Program :**

```
#include<iostream>
#include<fstream>
#include<stdio.h>
using namespace std;
//Employee class Declaration
class Employee{
private:
      int code;
      char name[20];
      float salary;
public:
      void read();
      void display();
      //will return employee code
      int getEmpCode()          { return code;}
      //will return employee salary
      int getSalary()           { return salary;}
      //will update employee salary
      void updateSalary(float s)  { salary=s;}
};
//Read employee record
void Employee::read(){
   cout<<"Enter employee code: ";
   cin>>code;
   cout<<"Enter name: ";
   cin.ignore(1);
   cin.getline(name,20);
   cout<<"Enter salary: ";
   cin>>salary;
}
//Display employee record
void Employee::display()
{
   cout<<code<<" "<<name<<"\t"<<salary<<endl;
}

//global declaration
```

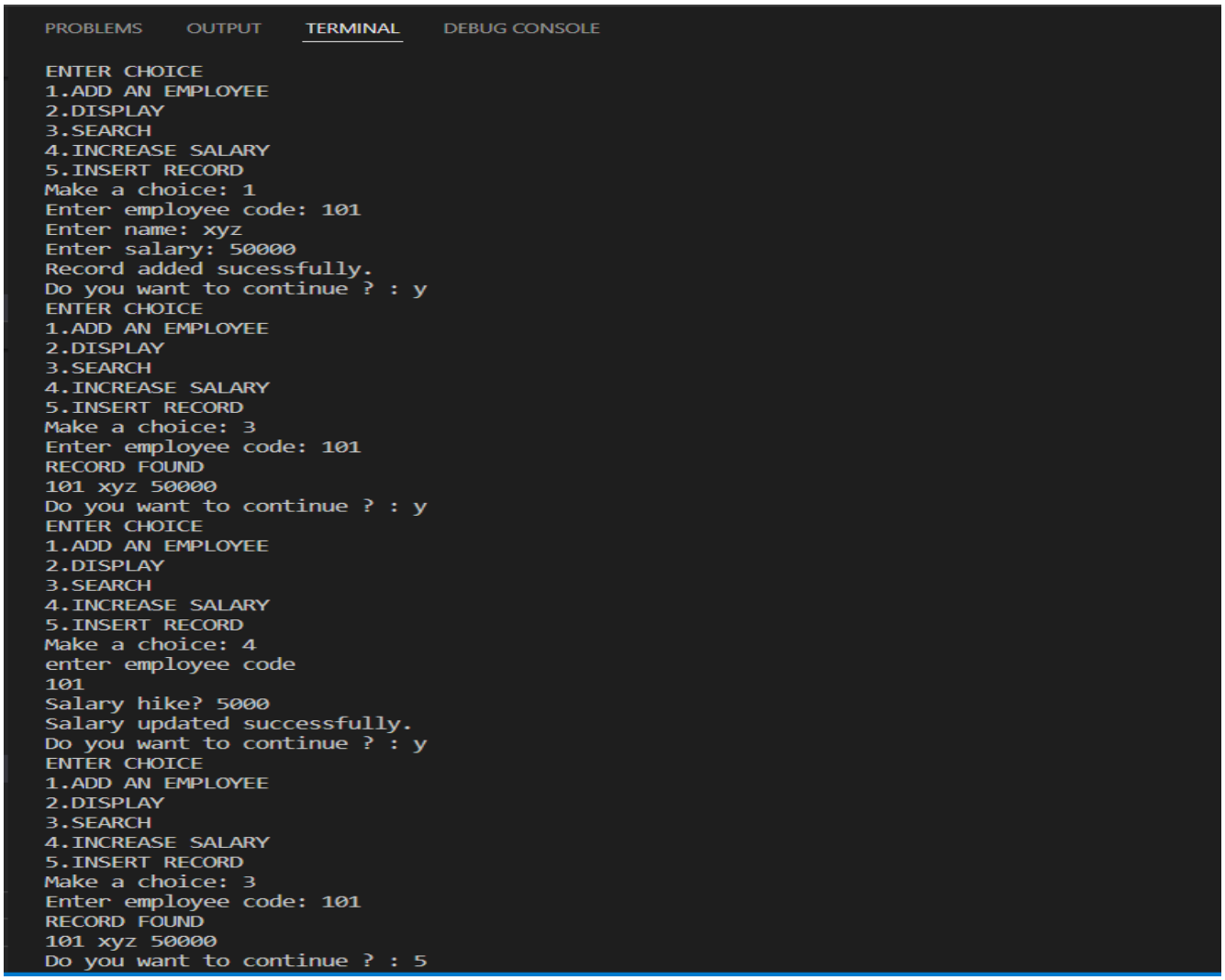
```
fstream file;

//Will delete file when program is being executed
//because we are create file in append mode
void deleteExistingFile(){
   remove("EMPLOYEE.DAT");
}
//function to append record into file
void appendToFille(){
   Employee    x;
   //Read employee record from user
   x.read();
   file.open("EMPLOYEE.DAT",ios::binary|ios::app);
   if(!file){
      cout<<"ERROR IN CREATING FILE\n";
      return;
   }
   //write into file
   file.write((char*)&x,sizeof(x));
   file.close();
   cout<<"Record added sucessfully.\n";
}

void displayAll(){
   Employee    x;

   file.open("EMPLOYEE.DAT",ios::binary|ios::in);
   if(!file){
      cout<<"ERROR IN OPENING FILE \n";
      return;
   }
   while(file){
   if(file.read((char*)&x,sizeof(x)))
      if(x.getSalary()>=10000 && x.getSalary()<=20000)
         x.display();
   }
  file.close();
}

void searchForRecord(){
   //read employee id
   Employee    x;
   int c;
   int isFound=0;
```

```
    cout<<"Enter employee code: ";
    cin>>c;


    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()==c){
                cout<<"RECORD FOUND\n";
                x.display();
                isFound=1;
                break;
            }
        }
    }
    if(isFound==0){
        cout<<"Record not found!!!\n";
    }
    file.close();
}

//Function to increase salary
void increaseSalary(){
    //read employee id
    Employee    x;
    int c;
    int isFound=0;
    float sal;

    cout<<"enter employee code \n";
    cin>>c;


    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()==c){
```

```
            cout<<"Salary hike? ";
            cin>>sal;
            x.updateSalary(x.getSalary()+sal);
            isFound=1;
            break;
         }
      }
   }
   if(isFound==0){
      cout<<"Record not found!!!\n";
   }
   file.close();
   cout<<"Salary updated successfully."<<endl;
}

//Insert record by assuming that records are in
//ascending order
void insertRecord(){
   //read employee record
   Employee    x;
   Employee newEmp;

   //Read record to insert
   newEmp.read();

   fstream fin;
   //read file in input mode
   file.open("EMPLOYEE.DAT",ios::binary|ios::in);
   //open file in write mode
   fin.open("TEMP.DAT",ios::binary|ios::out);
   if(!file){
      cout<<"Error in opening EMPLOYEE.DAT file!!!\n";
      return;
   }
   if(!fin){
      cout<<"Error in opening TEMP.DAT file!!!\n";
      return;
   }
   while(file){
      if(file.read((char*)&x,sizeof(x))){
         if(x.getEmpCode()>newEmp.getEmpCode()){
            fin.write((char*)&newEmp, sizeof(newEmp));
         }
         //no need to use else
         fin.write((char*)&x, sizeof(x));
```

```
        }
    }
    fin.close();
    file.close();
    rename("TEMP.DAT","EMPLOYEE.DAT");
    remove("TEMP.DAT");
    cout<<"Record inserted successfully."<<endl;
}
int main()
{
     char ch;
    //if required then only remove the file
     deleteExistingFile();
     do{
     int n;
     cout<<"ENTER CHOICE\n"<<"1.ADD AN
EMPLOYEE\n"<<"2.DISPLAY\n"<<"3.SEARCH\n"<<"4.INCREASE
SALARY\n"<<"5.INSERT RECORD\n";
     cout<<"Make a choice: ";
     cin>>n;
     switch(n){
         case 1:
           appendToFille();
           break;
         case 2 :
           displayAll();
           break;
         case 3:
           searchForRecord();
           break;
        case 4:
           increaseSalary();
           break;
        case 5:
           insertRecord();
           break;

          default :
               cout<<"Invalid Choice\n";
     }
     cout<<"Do you want to continue ? : ";
     cin>>ch;
     }while(ch=='Y'||ch=='y');
    return 0;
}
```

Output :


PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 1
Enter employee code: 101
Enter name: xyz
Enter salary: 50000
Record added sucessfully.
Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 3
Enter employee code: 101
RECORD FOUND
101 xyz 50000
Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 4
enter employee code
101
Salary hike? 5000
Salary updated successfully.
Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 3
Enter employee code: 101
RECORD FOUND
101 xyz 50000
Do you want to continue ? : 5