Assignment 1

```
size = 3
client_list = [None] * size

def add_client():
    client_id = int(input("client id"))
    name = input("client name")
    telephone = input("client telephone")
    client_details = [client_id, name, telephone]

    index = client_id % size
    # Inserting record using linear
    # probing in case of collision
    for i in range(size):
        if client_list[index] == None:
            client_list[index] = client_details
            print("adding data", index, client_details)
            break
        else:
            index = (index + 1) % size


def search_client():
    client_id = int(input("client id"))
    index = client_id % size
    for i in range(size):
        if client_list[index] != None:
            if client_list[index][0] == client_id:
                print("client is a found at index ", index, client_list[index])
                break
        index = (index + 1) % size
    else:
        print("element is not found")

def delete_client():
    client_id = int(input("client id"))
    index = client_id % size
    for i in range(size):
        if client_list[index] != None:
            if client_list[index][0] == client_id:
                client_list[index] = None
                print("cliet delted")
                break
```

```
        index = (index + 1) % size
    else:
        print("element is not found")

add_client()
add_client()
add_client()
print("serach client")
search_client()
print("deleted client")
delete_client()
print("search client")
search_client()
```

Output

```
client id44
client namea
client telephone4874
adding data 2 [44, 'a', '4874']
client id55
client nameb
client telephone333
adding data 1 [55, 'b', '333']
client id4
client namec
client telephone444
adding data 0 [4, 'c', '444']
serach client
client id4
client is a found at index  0 [4, 'c', '444']
deleted client
client id4
cliet delted
search client
client id4
element is not found
```

Assignment 2

'''
implement all the functions of dictionary (adt) using hashing and handle collisions using with/without replacement
data is set of key,value pairs. keys are mapped to values, keys must be comparbale and unique. functions to implement insert(key,value), find(key) and delete(key)
'''

```
def insert(data,mp):
    h=data[0]%10
    if(mp[h]==None):
        mp[h]=data
    elif(mp[h]!=None and mp[h][2]==-1):
        i=h+1
        while(i!=h):
            if(i==10):
                i=0
            elif(mp[i]!=None):
                i+=1
            elif(mp[i]==None):
                mp[i]=data
                mp[h][2]=i
                break
    elif(mp[h]!=None and mp[h][2]!=-1):
        k=mp[h][2]
        while(mp[k][2]!=-1):
            k=mp[k][2]
        i=k+1
        while(i!=k):
            if(i==10):
                i=0
            elif(mp[i]!=None):
                i+=1
            elif(mp[i]==None):
                mp[i]=data
                mp[k][2]=i
                break
def find(key,mp):
    h=key%10
    if(mp[h]!=None and mp[h][0]==key):
        print(mp[h])
    elif(mp[h]!=None and(mp[h][0]!=key and mp[h][2]!=-1)):
        k=mp[h][2]
```

```
                while(mp[k]!=None and(mp[k][0]!=key or mp[k][2]!=-1)):
                        k=mp[k][2]
                if(mp[k]!=None and mp[k][0]==key):
                        print(mp[k])
                else:
                        print("Not found")
        elif(mp[h]==None):
                print("Not Found")
def main():
        mp=[None]*10
        while(True):
                print("1. Insert")
                print("2. Delete")
                print("3. Find")
                c=int(input("Enter your choice: "))
                if(c==1):
                        key=int(input("Enter the key: "))
                        val=int(input("Enter the value: "))
                        temp=[key,val,-1]
                        insert(temp,mp)
                        print(mp)
                elif(c==3):
                        key=int(input("Enter the key to be searched: "))
                        find(key,mp)
                else:
                        break;
main()
```

output

```
1. Insert
2. Delete
3. Find
Enter your choice: 1
Enter the key: 8
Enter the value: 54
[None, None, None, None, None, None, None, None, [8, 54, -1], None]
1. Insert
2. Delete
3. Find
Enter your choice: 3
Enter the key to be searched: 8
[8, 54, -1]
```

Assignment 3

```
#include<iostream>
#include<string.h>
using namespace std;
struct node
{   char name[20];
    node *next;
    node *down;
     int flag;
};
class Gll
{    char ch[20];
    int n,i;
    node *head=NULL,*temp=NULL,*t1=NULL,*t2=NULL;
    public:
    node *create();
    void insertb();
    void insertc();
    void inserts();
    void insertss();
    void displayb();

};
node *Gll::create()
{
   node *p=new node;
   p->next=NULL;
   p->down=NULL;
   p->flag=0;
   cout<<"\n enter the name";
   cin>>p->name;
   return p;
}
void Gll::insertb()
{

       if(head==NULL)
        {    t1=create();
            head=t1;
        }
        else
        {
```

```
          cout<<"\n book exist";
        }
}
void Gll::insertc()
{
    if(head==NULL)
       {
            cout<<"\n there is no book";
        }
        else
        {   cout<<"\n how many chapters you want to insert";
            cin>>n;
            for(i=0;i<n;i++)
            {
            t1=create();
            if(head->flag==0)
            {  head->down=t1;
              head->flag=1;
            }
            else
            {   temp=head;
                temp=temp->down;
               while(temp->next!=NULL)
                   temp=temp->next;
                temp->next=t1;
            }
            }
       }


}
void Gll::inserts()
{

       if(head==NULL)
        {
            cout<<"\n there is no book";
        }
        else
        {   cout<<"\n Enter the name of chapter on which  you want to enter the section";
            cin>>ch;

            temp=head;
```

```
            if(temp->flag==0)
            {   cout<<"\n their are no chapters on in book";
            }
            else
            {    temp=temp->down;
             while(temp!=NULL)
              {
                 if(!strcmp(ch,temp->name))
                 {
                        cout<<"\n how many sections you want to enter";
                        cin>>n;
                        for(i=0;i<n;i++)
                        {

                              t1=create();
                                 if(temp->flag==0)
                                 {     temp->down=t1;

                                       temp->flag=1;  cout<<"\n******";
                                       t2=temp->down;

                                 }
                                 else
                                 {
                                          cout<<"\n#####";
                                           while(t2->next!=NULL)
                                           {    t2=t2->next;        }
                                                t2->next=t1;
                                  }
                         }
                           break;
                 }
                       temp=temp->next;
              }
            }
      }
}
void Gll::insertss()
{

      if(head==NULL)
       {
            cout<<"\n there is no book";
```

```
        }
        else
        {   cout<<"\n Enter the name of chapter on which  you want to enter the section";
            cin>>ch;

            temp=head;
           if(temp->flag==0)
           {   cout<<"\n their are no chapters on in book";
           }
           else
           {    temp=temp->down;
            while(temp!=NULL)
             {
                if(!strcmp(ch,temp->name))
                {
                  cout<<"\n enter name of section in which you want to enter the sub
section";
                  cin>>ch;

                 if(temp->flag==0)
                 {   cout<<"\n their are no sections ";   }
                  else
                  {      temp=temp->down;
                        while(temp!=NULL)
                        {
                           if(strcmp(ch,temp->name))
                           {
                            cout<<"\n how many subsections you want to enter";
                              cin>>n;
              for(i=0;i<n;i++)
                         {

                              t1=create();
                                 if(temp->flag==0)
                                 {     temp->down=t1;

                                       temp->flag=1;  cout<<"\n******";
                                       t2=temp->down;

                                 }
                                else
                                 {
                                          cout<<"\n#####";
```

```
                                    while(t2->next!=NULL)
                                    {    t2=t2->next;        }
                                          t2->next=t1;
                                }
                            }
                             break;
                          }     temp=temp->next;
                         }
                    }
                   }

                        temp=temp->next;
              }
             }
        }
}
void Gll::displayb()
{

             if(head==NULL)
             {  cout<<"\n book not exist";
             }
             else
             {
              temp=head;

                cout<<"\n NAME OF BOOK:  "<<temp->name;
                   if(temp->flag==1)
                   {
                   temp=temp->down;

                    while(temp!=NULL)
                    {    cout<<"\n\t\tNAME OF CHAPTER:  "<<temp->name;
                        t1=temp;
                        if(t1->flag==1)
                        {  t1=t1->down;
                          while(t1!=NULL)
                          {    cout<<"\n\t\t\t\tNAME OF SECTION:  "<<t1->name;
                              t2=t1;
                              if(t2->flag==1)
                              {  t2=t2->down;
                              while(t2!=NULL)
                              {    cout<<"\n\t\t\t\t\t\tNAME OF SUBSECTION:  "<<t2->name;
```

```
                        t2=t2->next;
                        }
                        }
                        t1=t1->next;
                    }
                   }
                    temp=temp->next;
                }

               }
          }



}
int main()
{   Gll g;   int x;
     while(1)
    {   cout<<"\n\n enter your choice";
        cout<<"\n 1.insert book";
        cout<<"\n 2.insert chapter";
        cout<<"\n 3.insert section";
        cout<<"\n 4.insert subsection";
        cout<<"\n 5.display book";
        cout<<"\n 6.exit";
        cin>>x;
       switch(x)
       {  case 1:       g.insertb();
                         break;
          case 2:       g.insertc();
                         break;
          case 3:       g.inserts();
                         break;
          case 4:       g.insertss();
                         break;
          case 5:       g.displayb();
                         break;
          case 6:  exit(0);
       }
    }
    return 0;
}
```

Output

```
enter your choice
 1.insert book
 2.insert chapter
 3.insert section
 4.insert subsection
 5.display book
 6.exit1

 enter the namebook1


 enter your choice
 1.insert book
 2.insert chapter
 3.insert section
 4.insert subsection
 5.display book
 6.exit2

 how many chapters you want to insert3

 enter the namechp 1

 enter the name
 enter the namechap2


 enter your choice
 1.insert book
 2.insert chapter
 3.insert section
 4.insert subsection
 5.display book
 6.exit5

 NAME OF BOOK:  book1
        NAME OF CHAPTER:  chp
        NAME OF CHAPTER:  1
        NAME OF CHAPTER:  chap2
```

Assignment 4

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct node
{  int a;
  node *left,*right;
 };
class Bt
{
        node *root=NULL,*temp=NULL,*t1=NULL,*s=NULL, *t=NULL;
        int count;
        public:
        Bt(){  count=0;         }
        node *create();
        void insert();
        void del();
        node *delet(node*,int);
        void find();
        void search();
        void sw();
        void swap(node*);
        void height();
        int he(node*,int);
        void disp(node*);
        void display();
        node *findmin(node*);

};
node *Bt::create()
{
        node *p=new(struct node);
        p->left=NULL;
        p->right=NULL;
        cout<<"\n enter the data";
        cin>>p->a;
        return p;
}
void Bt::insert()
{
        temp=create();
        if(root==NULL)
        {       root=temp;    }
```

```
        else
        {       t1=root;
        while(t1!=NULL)
        {       s=t1;
        if((temp->a)>(t1->a))
        {       t1=t1->right;  }
        else
        {       t1=t1->left;    }
        }
        if((temp->a)>(s->a))
        {       s->right=temp;          }
        else
        {  s->left=temp;        }
        }
}
void Bt::find()
{
        if(root==NULL)
        {       cout<<"\n tree not exist";  }
        else
        {
        t1=root;

        while(t1->left!=NULL)
        {
        t1=t1->left;
        }

        cout<<"\n smallest no."<<t1->a;
        t1=root;

        while(t1->right!=NULL)
        {
        t1=t1->right;
        }
        cout<<"\n largest no."<<t1->a;
        }
}
void Bt::search()
{
        int m,f=0;
        if(root==NULL)
        {  cout<<"\n tree not exist";
```

```
        }
        else
        {
        cout<<"\n enter data to be searched";
        cin>>m;
        if(root->a==m)
        {   cout<<"\ndata found";   }
        else
        {         t1=root;
        while(t1->a!=m)
        {
        if((m)>(t1->a))
        {         t1=t1->right;  }
        else
        {   t1=t1->left;           }
        if(t1==NULL)
        {   cout<<"\n data not found";         f=1;
                  break;
        }
        }
        if(f==0)
        {   cout<<"\n data found";    }

        }
        }
}
 void Bt::sw()
 {
        if(root==NULL)
        {  cout<<"\n tree not exist";
        }
        else
        {
        swap(root);
        }
}
void Bt::swap(node *q)
{
  if(q->left!=NULL)
  swap(q->left);
  if(q->right!=NULL)
  swap(q->right);
  t=q->left;
```

```
  q->left=q->right;
  q->right=t;
}
void Bt::height()
{
        count=0;
        if(root==NULL)
        {  cout<<"\n tree not exist";
        }
        else
        {
        he(root,0); cout<<"\n height of the tree is"<<count;
        }

}
int Bt::he(node *q,int c)  // he is a function that will be used to calculate height of the
tree. Can be called using root and counter intilized to 0
{
        c++;
        // cout<<"\n*"<<q->a<<"*"<<c<<"*\n";
        if(q->left!=NULL)
        {       he(q->left,c);
        }
        if(q->right!=NULL)
        {
        he(q->right,c);
        }
        if(count<c)
        {
        count=c;
        }


        return 0;

}
void Bt::del()
{       int x;
  cout<<"\n enter data to be deleted";
  cin>>x;
  delet(root,x);

}
```

```
node *Bt::delet(node *T,int x)
{
        if(T==NULL)
        {
        cout<<"\n element not found";
        return(T);
        }
        if(x<T->a)
        {
        T->left=delet(T->left,x);
        return (T);
        }
        if(x>T->a)
        {
        T->right=delet(T->right,x);
        return T;
        }
        if(T->left==NULL&&T->right==NULL)
        {
        temp=T;
        free(temp);
        return(NULL);
        }
        if(T->left==NULL)
        {
        temp=T;
        T=T->right;
        delete temp;
        return T;
        }
        if(T->right==NULL)
        {
        temp=T;
        T=T->left;
        delete temp;
        return T;
        }
        temp=findmin(T->right);
        T->a=temp->a;
        T->right=delet(T->right,temp->a);
        return T;
}
node *Bt::findmin(node *T)
```

```
{
        while(T->left!=NULL)
        {  T=T->left;    }
        return T;
}


void Bt::display()
{

        if(root==NULL)
        {  cout<<"\n tree not exist";
        }
        else
        {
        disp(root);
        }

}
void Bt::disp(node *q)
{
        cout<<"\n*"<<q->a;
        if(q->left!=NULL)
        {         disp(q->left);
        }
        if(q->right!=NULL)
        {
        disp(q->right);
        }

 }
int main()
{
        Bt b;  int x;      char ch;
        while(1)
        {
        cout<<"\n enter your choice";
        cout<<"\n 1.insert";
        cout<<"\n 2.find";
        cout<<"\n 3.search";
        cout<<"\n 4.swap";
        cout<<"\n 5.height";
        cout<<"\n 6.delete";
```

```
        cout<<"\n 7.display";
        cout<<"\n 8.exit";
        cin>>x;
        switch(x)
        {       case 1: b.insert();
                        break;
                case 2: b.find();
                        break;
                case 3: b.search();
                        break;

                case 4: b.sw();
                        break;
                case 5: b.height();
                        break;
                case 6: b.del();
                        break;
                case 7: b.display();
                        break;
                case 8:exit(0);
        }

        }

        return 0;
}
```

Output

```
 enter your choice
 1.insert
 2.find
 3.search
 4.swap
 5.height
 6.delete
 7.display
 8.exit1

 enter the data55

 enter your choice
```

1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit1

enter the data78

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit1

enter the data56

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit1

enter the data53

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display

8.exit1

enter the data32

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit2

smallest no.32
largest no.78

Assignment 5

```
#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()
{   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 enter the choice";
  cout<<"\n 1.insert data";
  cout<<"\n 2.display all data";
  cout<<"\n 3.display threaded node";
  cout<<"\n 4.exit";
  cin>>ch;
  switch(ch)
  {
        case 1:
        t.insert();
        break;
        case 2:
        t.dis();
        break;
        case 3:
        t.thr();
        break;
        case 4:  exit(0);
        default:
        cout<<"\n invalid entry";
        }
        }
  return 0;
}
```

Output:

```
enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit1

enter the data45

enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit1

enter the data71

enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit1

enter the data61

enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit1

enter the data43

enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit1

enter the data46

enter the choice
1.insert data
```

2.display all data
3.display threaded node
4.exit2

43
45
46
61
71
enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit3

thread are
43
46
61
71
enter the choice
1.insert data
2.display all data
3.display threaded node
4.exit4

Assignment 6

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
using namespace std;
struct node
{   string vertex;
   int time;
   node *next;
};
class adjmatlist
{
    int m[10][10],n,i,j;
    char ch;
    string v[20];
    node *head[20];
    node *temp=NULL;

    public:
    adjmatlist()
    {     for(i=0;i<20;i++)
          {
             head[i]=NULL;
          }
    }
    void getgraph();
    void adjlist();

    void displaym();
    void displaya();
};
void adjmatlist::getgraph()
{
  cout<<"\n enter no. of cities(max. 20)";
  cin>>n;
  cout<<"\n enter name of cities";
  for(i=0;i<n;i++)
    cin>>v[i];
  for(i=0;i<n;i++)
  {
     for(j=0;j<n;j++)
```

```
    {  cout<<"\n if path is present between city "<<v[i]<<" and "<<v[j]<<" then press
enter y otherwise n";
       cin>>ch;
       if(ch=='y')
       {
        cout<<"\n enter time required to reach city "<<v[j]<<" from "<<v[i]<<" in
minutes";
        cin>>m[i][j];
       }
       else if(ch=='n')
       {  m[i][j]=0;  }
       else
       { cout<<"\n unknown entry";  }
    }
  }
    adjlist();

}
void adjmatlist::adjlist()
{     cout<<"\n ****";
      for(i=0;i<n;i++)
      {  node *p=new node;
        p->next=NULL;
        p->vertex=v[i];
        p->time=m[i][j];
        head[i]=p;
        cout<<"\n"<<head[i]->vertex;
      }

      for(i=0;i<n;i++)
      {  for(j=0;j<n;j++)
        {
              if(m[i][j]!=0)
              {
                  node *p=new node;
                  p->vertex=v[j];
                  p->time=m[i][j];
                  p->next=NULL;
                  if(head[i]->next==NULL)
                  {
                     head[i]->next=p;
                  }
                  else
```

```
                {
                   temp=head[i];
                   while(temp->next!=NULL)
                   {
                     temp=temp->next;
                   }
                   temp->next=p;
                }

            }

       }
     }

}
void adjmatlist::displaym()
{   cout<<"\n";
    for(j=0;j<n;j++)
    {  cout<<"\t"<<v[j];  }

    for(i=0;i<n;i++)
    {  cout<<"\n "<<v[i];
       for(j=0;j<n;j++)
       {   cout<<"\t"<<m[i][j];
       }
          cout<<"\n";
    }
}
void adjmatlist::displaya()
{
     cout<<"\n adjacency list is";

     for(i=0;i<n;i++)
     {


                if(head[i]==NULL)
                {   cout<<"\n adjacency list not present";  break;   }
                else
                {
                  cout<<"\n"<<head[i]->vertex;
                temp=head[i]->next;
                while(temp!=NULL)
```

```
                { cout<<"-> "<<temp->vertex;
                  temp=temp->next; }

                }




    }

      cout<<"\n path and time required to reach cities is";

    for(i=0;i<n;i++)
    {


                if(head[i]==NULL)
                {  cout<<"\n adjacency list not present";  break;  }
                else
                {

                temp=head[i]->next;
                while(temp!=NULL)
                { cout<<"\n"<<head[i]->vertex;
                  cout<<"-> "<<temp->vertex<<"\n   [time required: "<<temp->time<<"
min ]";
                  temp=temp->next; }

                }




    }
}
int main()
{
  int m;
  adjmatlist a;

  while(1)
  {
  cout<<"\n\n enter the choice";
```

```
  cout<<"\n 1.enter graph";
  cout<<"\n 2.display adjacency matrix for cities";
  cout<<"\n 3.display adjacency list for cities";
  cout<<"\n 4.exit";
  cin>>m;

     switch(m)
    {          case 1: a.getgraph();
                        break;
               case 2: a.displaym();
                        break;

               case 3: a.displaya();
                        break;
               case 4: exit(0);

                    default:  cout<<"\n unknown choice";
     }
  }
  return 0;
}
```

Output

```
enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit1

 enter no. of cities(max. 20)4

 enter name of citiesa
b
c
d

 if path is present between city a and a then press enter y otherwise nn

 if path is present between city a and b then press enter y otherwise ny

 enter time required to reach city b from a in minutes5

 if path is present between city a and c then press enter y otherwise ny
```

enter time required to reach city c from a in minutes15

if path is present between city a and d then press enter y otherwise ny

enter time required to reach city d from a in minutes10

if path is present between city b and a then press enter y otherwise ny

enter time required to reach city a from b in minutes15

if path is present between city b and b then press enter y otherwise nn

if path is present between city b and c then press enter y otherwise ny

enter time required to reach city c from b in minutes5

if path is present between city b and d then press enter y otherwise ny

enter time required to reach city d from b in minutes5

if path is present between city c and a then press enter y otherwise ny

enter time required to reach city a from c in minutes15

if path is present between city c and b then press enter y otherwise ny

enter time required to reach city b from c in minutes20

if path is present between city c and c then press enter y otherwise nn

if path is present between city c and d then press enter y otherwise ny

enter time required to reach city d from c in minutes5

if path is present between city d and a then press enter y otherwise nn

if path is present between city d and b then press enter y otherwise nn

if path is present between city d and c then press enter y otherwise ny

enter time required to reach city c from d in minutes10

if path is present between city d and d then press enter y otherwise nn

****
a
b
c
d

enter the choice
1.enter graph
2.display adjacency matrix for cities
3.display adjacency list for cities
4.exit2

	a	b	c	d
a	0	5	15	10
b	15	0	5	5
c	15	20	0	5
d	0	0	10	0

enter the choice
1.enter graph
2.display adjacency matrix for cities
3.display adjacency list for cities
4.exit3

adjacency list is
a-> b-> c-> d
b-> a-> c-> d
c-> a-> b-> d
d-> c
path and time required to reach cities is
a-> b
[time required: 5 min]
a-> c
[time required: 15 min]
a-> d
[time required: 10 min]
b-> a

[time required: 15 min]
b-> c
[time required: 5 min]
b-> d
[time required: 5 min]
c-> a
[time required: 15 min]
c-> b
[time required: 20 min]
c-> d
[time required: 5 min]
d-> c
[time required: 10 min]

Assignment 7

```
#include<iostream>
#include<climits>
using namespace std;
template <class T>
class Graph
{   int ** AM,num;
        T * data;
public:
        Graph(int n)
        {   AM=new int*[n];
        for(int i=0;i<n;i++)
        AM[i]=new int[n];
        num=n;
        data=new T[n];
        cout<<"Enter names of all cities : ";
        for(int i=0;i<n;i++)
        cin>>data[i];
        cout<<"Enter cost if you want to connect cities else enter 0: \n";
        for(int j=0;j<n;j++)
        cout<<data[j]<<" ";
        cout<<endl;
        for(int i=0,cost=0;i<n;i++)
        {   cout<<"Nodes connected to "<<data[i]<<" :\n";
        for(int j=0;j<i;j++)
        cout<<AM[i][j]<<"\t";
        for(int j=i;j<n;j++)
        if(j==i) {cout<<"0\t";AM[i][j]=AM[j][i]=0;}
        else {cin>>cost;AM[i][j]=AM[j][i]=cost;}
        }
        for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
        if(AM[i][j]==0)AM[i][j]=INT_MAX;
        }
        void prims()
        {
        cout<<"\nCities that we need to connect:\n";
        int *visited=new int[num](),*distance=new int[num],*from=new
int[num](),cost=0;
        visited[0]=1;
        for(int i=0;i<num;i++)
        distance[i]=AM[0][i];
        int u,v;
```

```
        for(int count=num-1;count>0;count--)
        {   int min=INT_MAX;
        for(int j=1;j<num;j++)
        if(visited[j]==0&&distance[j]<min)
        {v=j;min=distance[j];}
        u=from[v];
        cout<<data[u]<<"==>"<<data[v]<<"\tcost: "<<AM[u][v]<<endl;
        visited[v]=1;
        for(int j=1;j<num;j++)
        if(visited[j]==0&&AM[j][v]<distance[j])
        {distance[j]=AM[j][v];from[j]=v;}
        cost+=AM[u][v];
        }
        cout<<"Total cost of connecting all cities : "<<cost<<endl;
        }
};
int main()
{   int n;
        cout<<"Enter number of cities: ";
        cin>>n;
        Graph<string> gr(n);
        gr.prims();
        return 0;
}
```

Output

```
Enter number of cities: 5
Enter names of all cities : a
b
c
d
e
Enter cost if you want to connect cities else enter 0:
a b c d e
Nodes connected to a :
0       2
3
4
5
Nodes connected to b :
2       0       0
0
5
```

Nodes connected to c :
3 0 0 3
1
Nodes connected to d :
4 0 3 0 0
Nodes connected to e :
5 5 1 0 0
Cities that we need to connect:
a==>b cost: 2
a==>c cost: 3
c==>e cost: 1
c==>d cost: 3
Total cost of connecting all cities : 9

Assignment 8

```
#include <iostream>
using namespace std;
class obst
{
   int a[10], r[10][10], n;
   float p[10], q[10], w[10][10], c[10][10];
   public:
   void accept();
   void cons_obst();
   int knuthmin(int, int);
   void tree(int i, int j);
};
void obst::accept()
{
   int i;
   cout << "how many elements are there in the tree?\n";
   cin >> n;
   cout << "enter" << n << "elements \n";
   for (i = 1; i <= n; i++)
   cin >> a[i] ;
   cout << "enter" << n << "their probabilities\n";
   for (i = 1; i <= n; i++)
   cin >>  p[i];
   cout << "enter" << n + 1 << "failure probabilities\n";
   for (i = 0; i <= n; i++)
   cin >> q[i];
}
void obst::cons_obst()
{
   int i, m, j, k;
   for (i = 0; i < n; i++) /* Initialize the weight and cost matrices */
   {
      w[i][i] = q[i];
      r[i][i] = c[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] = w[i][i + 1];
   }
   w[n][n] = q[n];
   r[n][n] = c[n][n] = 0;
   for (m = 2; m <= n; m++) /* calculate the weight and cost matrices */
   {
```

```
      for (i = 0; i <= n - m; i++)
      {
         j = i + m;
         w[i][j] = w[i][j - 1] + p[j] + q[j];
         k = knuthmin(i, j); /* find minimum value in the range r[i-1][j] to r[i][j-1] */
         c[i][j] = w[i][j] + c[i][k - 1] + c[k][j];
         r[i][j] = k;
      }
   }
   cout << "root node is " << a[r[0][n]];
   cout << "\nleft child of " << a[r[0][n]] << " is ";
   tree(0, r[0][n] - 1);
   cout << "\nright child of " << a[r[0][n]] << " is ";
   tree(r[0][n], n);
}
int obst::knuthmin(int i, int j)
{
   int min = 999, k, z;
   for (k = r[i][j - 1]; k <= r[i + 1][j]; k++)
   {
      if (min > c[i][k - 1] + c[k][j])
      {
         min = c[i][k - 1] + c[k][j];
         z = k;
      }
   }
   return (z);
}
void obst::tree(int i, int j)
{
   if (r[i][j] == 0)
   {
   cout<<" NULL\n";
   return;
   }
   cout << " :: " << a[r[i][j]];
   cout << "\n left child of is ::" << a[r[i][j]];
   tree(i, r[i][j] - 1);
   cout << "\n right child of is :: " << a[r[i][j]];
   tree(r[i][j], j);
}
int main()
{
```

```
   obst o;
   o.accept();
   o.cons_obst();
}
```

Output

```
how many elements are there in the tree?
5
enter5elements
6
12
56
78
41
enter5their probabilities
0.5
0.9
0.3
0.4
0.6
enter6failure probabilities
0.5
0.1
0.7
0.6
0.4

0.1
root node is 12
left child of 12 is  :: 6
 left child of is ::6 NULL

 right child of is :: 6 NULL
right child of 12 is  :: 78
 left child of is ::78 :: 56
 left child of is ::56 NULL

 right child of is :: 56 NULL

 right child of is :: 78 :: 41
 left child of is ::41 NULL

 right child of is :: 41 NULL
```

Assignment 9

```
#include<iostream>
#include<string.h>
using namespace std;
class dict
{
        dict *root,*node,*left,*right,*tree1;
        string s1,s2;
        int flag,flag1,flag2,flag3,cmp;
public:
        dict()
        {
        flag=0,flag1=0,flag2=0,flag3=0,cmp=0;
        root=NULL;
        }
        void input();
        void create_root(dict*,dict*);
        void check_same(dict*,dict*);
        void input_display();
        void display(dict*);
        void input_remove();
        dict* remove(dict*,string);
        dict* findmin(dict*);
        void input_find();
        dict* find(dict*,string);
        void input_update();
        dict* update(dict*,string);

};

        void dict::input()
        {
                        node=new dict;
                        cout<<"\nEnter the keyword:\n";
                        cin>>node->s1;
                        cout<<"Enter the meaning of the keyword:\n";
                        cin.ignore();
                        getline(cin,node->s2);
                        create_root(root,node);
        }


                        void dict::create_root(dict *tree,dict *node1)
```

```
{
        int i=0,result;
        char a[20],b[20];
        if(root==NULL)
        {
        root=new dict;
        root=node1;
        root->left=NULL;
        root->right=NULL;
        cout<<"\nRoot node created successfully"<<endl;
        return;
        }
        for(i=0;node1->s1[i]!='\0';i++)
        {
                a[i]=node1->s1[i];
        }
        for(i=0;tree->s1[i]!='\0';i++)
        {
                b[i]=tree->s1[i];
        }
        result=strcmp(b,a);
        check_same(tree,node1);
        if(flag==1)
        {
        cout<<"The word you entered already exists.\n";
        flag=0;
        }
        else
        {
        if(result>0)
        {
        if(tree->left!=NULL)
        {
        create_root(tree->left,node1);
        }
        else
        {
        tree->left=node1;
        (tree->left)->left=NULL;
                        (tree->left)->right=NULL;
                cout<<"Node added to left of "<<tree->s1<<"\n";
                return;
                }
```

```
                                        }
                                        else if(result<0)
                                        {
                                         if(tree->right!=NULL)
                                         {
                                                create_root(tree->right,node1);
                                         }
                                         else
                                         {
                                                tree->right=node1;
                                                (tree->right)->left=NULL;
                                                (tree->right)->right=NULL;
                                                cout<<"Node added to right of
"<<tree->s1<<"\n";
                                                return;
                                         }
                                        }
                                        }
                                }


void dict::check_same(dict *tree,dict *node1)
{
   if(tree->s1==node1->s1)
   {
          flag=1;
          return;
   }
   else if(tree->s1>node1->s1)
         {
    if(tree->left!=NULL)
   {
           check_same(tree->left,node1);
   }
         }
         else if(tree->s1<node1->s1)
         {
          if(tree->right!=NULL)
          {
          check_same(tree->right,node1);
          }
         }
}
```

```
void dict::input_display()
{
        if(root!=NULL)
        {
                cout<<"The words entered in the dictionary are:\n\n";
                display(root);
        }
        else
        {
        cout<<"\nThere are no words in the dictionary.\n";
        }
}


                void dict::display(dict *tree)
                {
                        if(tree->left==NULL&&tree->right==NULL)
                        {
                                cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
                        }
                        else
                        {
                if(tree->left!=NULL)
                {
                        display(tree->left);
                }
                cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
                if(tree->right!=NULL)
                {
                        display(tree->right);
                }
                        }
                }


void dict::input_remove()
{
   char t;
   if(root!=NULL)
   {
     cout<<"\nEnter a keyword to be deleted:\n";
```

```
    cin>>s1;
    remove(root,s1);
    if(flag1==0)
    {
                cout<<"\nThe word '"<<s1<<"' has been deleted.\n";
    }
    flag1=0;
   }
   else
   {
        cout<<"\nThere are no words in the dictionary.\n";
   }
}


        dict* dict::remove(dict *tree,string s3)
        {
                dict *temp;
                if(tree==NULL)
                {
                        cout<<"\nWord not found.\n";
                        flag1=1;
                        return tree;
                }
                else if(tree->s1>s3)
                {
                        tree->left=remove(tree->left,s3);
                        return tree;
                }
                else if(tree->s1<s3)
                        {
                        tree->right=remove(tree->right,s3);
                        return tree;
                }
                else
                {
                        if(tree->left==NULL&&tree->right==NULL)
                        {
                                delete tree;
                                tree=NULL;
                        }
                        else if(tree->left==NULL)
                        {
```

```
                temp=tree;
                tree=tree->right;
                delete temp;
        }
        else if(tree->right==NULL)
        {
                temp=tree;
                tree=tree->left;
                delete temp;
        }
        else
        {
                temp=findmin(tree->right);
                tree=temp;
                tree->right=remove(tree->right,temp->s1);
        }
    }
    return tree;
}


        dict* dict::findmin(dict *tree)
        {
                while(tree->left!=NULL)
                {
                        tree=tree->left;
                }
                return tree;
        }


void dict::input_find()
{
     flag2=0,cmp=0;
     if(root!=NULL)
     {
     cout<<"\nEnter the keyword to be searched:\n";
     cin>>s1;
    find(root,s1);
    if(flag2==0)
    {
        cout<<"Number of comparisons needed: "<<cmp<<"\n";
        cmp=0;
```

```
        }
        }
        else
        {
                cout<<"\nThere are no words in the dictionary.\n";
        }
}


                dict* dict::find(dict *tree,string s3)
                {
                        if(tree==NULL)
                        {
                                cout<<"\nWord not found.\n";
                                flag2=1;
                                flag3=1;
                                cmp=0;
                        }
                        else
                        {
                                if(tree->s1==s3)
                                {
                                        cmp++;
                                        cout<<"\nWord found.\n";
                                        cout<<tree->s1<<": "<<tree->s2<<"\n";
                                        tree1=tree;
                                        return tree;
                                }
                                else if(tree->s1>s3)
                                {
                                        cmp++;
                                        find(tree->left,s3);
                                }
                                else if(tree->s1<s3)
                                {
                                        cmp++;
                                        find(tree->right,s3);
                                }
                        }
                        return tree;
        }
```

```
void dict::input_update()
{
  if(root!=NULL)
  {
  cout<<"\nEnter the keyword to be updated:\n";
  cin>>s1;
        update(root,s1);
  }
  else
  {
        cout<<"\nThere are no words in the dictionary.\n";
  }
}


        dict* dict::update(dict *tree,string s3)
        {
                flag3=0;
                find(tree,s3);
                if(flag3==0)
                {
                cout<<"\nEnter the updated meaning of the keyword:\n";
                cin.ignore();
                getline(cin,tree1->s2);
                cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";
                }
                return tree;
        }




                        int main()
                         {
                                int ch;
                                dict d;
                                do
                                {

cout<<"\n========================================\n"
                                        "\n********DICTIONARY***********:\n"
                                 "\nEnter your choice:\n"
```

```
                    "1.Add new keyword.\n"
                    "2.Display the contents of the Dictionary.\n"
                "3.Delete a keyword.\n"
                "4.Find a keyword.\n"
                "5.Update the meaning of a keyword.\n"
                "6.Exit.\n"

"================================================\n";
                cin>>ch;
                switch(ch)
                {
                    case 1:d.input();
                    break;
                    case 2:d.input_display();
                                break;
                    case 3:d.input_remove();
                    break;
                    case 4:d.input_find();
                    break;
                    case 5:d.input_update();
                                break;
                    default:cout<<"\nPlease enter a valid option!\n";
                                break;
                }
                 }while(ch!=6);
                return 0;
            }
```

Output

```
==========================================

********DICTIONARY***********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
================================================
1
```

Enter the keyword:
f
Enter the meaning of the keyword:
the 6th letter

Root node created successfully

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
1

Enter the keyword:
a
Enter the meaning of the keyword:
the 1st letter
Node added to left of f

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
1

Enter the keyword:

c
Enter the meaning of the keyword:
the 3rd letter
Node added to right of a

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
2
The words entered in the dictionary are:

a = the 1st letter

c = the 3rd letter

f = the 6th letter

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
2
The words entered in the dictionary are:

a = the 1st letter

c = the 3rd letter

f = the 6th letter

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
4

Enter the keyword to be searched:
c

Word found.
c: the 3rd letter
Number of comparisons needed: 3

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
5

Enter the keyword to be updated:
c

Word found.
c: the 3rd letter

Enter the updated meaning of the keyword:
the letter after b

The meaning of 'c' has been updated.

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
4

Enter the keyword to be searched:
c

Word found.
c: the letter after b
Number of comparisons needed: 3

==========

********DICTIONARY**********:

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========

Assignment 10

```
#include <iostream>
using namespace std;
void min_heapify(int *a,int i,int n)
{
   int j, temp;
   temp = a[i];
   j = 2 * i;
   while (j <= n)
   {
      if (j < n && a[j+1] < a[j])
         j = j + 1;
      if (temp < a[j])
         break;
      else if (temp >= a[j])
      {
         a[j/2] = a[j];
         j = 2 * j;
      }
   }
   a[j/2] = temp;

}
void max_heapify(int *a,int i,int n)
{
   int j, temp;
   temp = a[i];
   j = 2 * i;
   while (j <= n)
   {
      if (j < n && a[j+1] > a[j])
         j = j + 1;
      if (temp > a[j])
         break;
      else if (temp <= a[j])
      {
         a[j/2] = a[j];
         j = 2 * j;
      }
   }
   a[j/2] = temp;
   return;
}
```

```
void build_minheap(int *a, int n)
{
   int i;
   for(i = n/2; i >= 1; i--)
   {
      min_heapify(a,i,n);
   }
}
void build_maxheap(int *a, int n)
{
   int i;
   for(i = n/2; i >= 1; i--)
   {
      max_heapify(a,i,n);
   }
}
int main()
{
   int n, i, x,ch;
   char choice;
   cout<<"Enter no of marks of array\n";
   cin>>n;
   int a[20];
   for (i = 1; i <= n; i++)
   {
      cout<<"Enter marks"<<(i)<<endl;
      cin>>a[i];
   }
   do
   {
         cout<<"\n***Enter the choice***\n1.MIN Heap\n2.MAX Heap\n";
         cin>>ch;
         switch(ch)
         {
                  case 1:
                           build_minheap(a, n);
                           cout<<"Min Heap\n";
                            for (i = 1; i <= n; i++)
                            {
                           cout<<a[i]<<endl;
                            }
                            break;
                  case 2:
```

```
                        build_maxheap(a, n);
                        cout<<"Max Heap\n";
                         for (i = 1; i <= n; i++)
                         {
                        cout<<a[i]<<endl;
                         }
                         break;
        }
        cout<<"Do you wany to continue (Y/N) : ";
        cin>>choice;
   }while(choice=='Y'||choice=='y');
}
```

Output

```
Enter no of marks of array
8
Enter marks1
45
Enter marks2
78
Enter marks3
14
Enter marks4
59
Enter marks5
46
Enter marks6
23
Enter marks7
18
Enter marks8
64

***Enter the choice***
1.MIN Heap
2.MAX Heap
1
Min Heap
14
46
18
59
78
```

```
23
45
64
Do you wany to continue (Y/N) : y

***Enter the choice***
1.MIN Heap
2.MAX Heap
2
Max Heap
78
64
45
59
46
23
18
14
Do you wany to continue (Y/N) : n
```

Assignment 11

```
#include<iostream>
#include<fstream>
#include<string.h>
using namespace std;
class student
 {
        typedef struct stud
        {
                int roll;
                char name[10];
                char div;
                char add[10];
        }stud;
        stud rec;
        public:
          void create();
          void display();
          int search();
          void Delete();
 };
void student::create()
 {
        char ans;
        ofstream fout;
        fout.open("stud.dat",ios::out|ios::binary);
        do
         {
                cout<<"\n\tEnter Roll No of Student    : ";
                cin>>rec.roll;
                cout<<"\n\tEnter a Name of Student     : ";
                cin>>rec.name;
                cout<<"\n\tEnter a Division of Student : ";
                cin>>rec.div;
                cout<<"\n\tEnter a Address of Student  : ";
                cin>>rec.add;
                fout.write((char *)&rec,sizeof(stud))<<flush;
                cout<<"\n\tDo You Want to Add More Records: ";
                cin>>ans;
          }while(ans=='y'||ans=='Y');
        fout.close();
 }
void student::display()
```

```
 {
        ifstream fin;
        fin.open("stud.dat",ios::in|ios::binary);
        fin.seekg(0,ios::beg);
        cout<<"\n\tThe Content of File are:\n";
        cout<<"\n\tRoll\tName\tDiv\tAddress";
        while(fin.read((char *)&rec,sizeof(stud)))
         {
                if(rec.roll!=-1)

cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;
         }
        fin.close();
 }
int student::search()
 {
        int r,i=0;
        ifstream fin;
        fin.open("stud.dat",ios::in|ios::binary);
        fin.seekg(0,ios::beg);
        cout<<"\n\tEnter a Roll No: ";
        cin>>r;
        while(fin.read((char *)&rec,sizeof(stud)))
         {
                if(rec.roll==r)
                 {
                        cout<<"\n\tRecord Found...\n";
                        cout<<"\n\tRoll\tName\tDiv\tAddress";

cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;
                        return i;
                 }
                i++;
         }
        fin.close();
        return 0;
 }
void student::Delete()
 {
        int pos;
        pos=search();
        fstream f;
        f.open("stud.dat",ios::in|ios::out|ios::binary);
```

```
        f.seekg(0,ios::beg);
        if(pos==0)
         {
                cout<<"\n\tRecord Not Found";
                return;
         }
        int offset=pos*sizeof(stud);
        f.seekp(offset);
        rec.roll=-1;
        strcpy(rec.name,"NULL");
        rec.div='N';
        strcpy(rec.add,"NULL");
        f.write((char *)&rec,sizeof(stud));
        f.seekg(0);
        f.close();
        cout<<"\n\tRecord Deleted";
 }

int main()
 {
        student obj;
        int ch,key;
        char ans;
        do
         {
                cout<<"\n\t***** Student Information *****";
                cout<<"\n\t1. Create\n\t2. Display\n\t3. Delete\n\t4. Search\n\t5. Exit";
                cout<<"\n\t..... Enter Your Choice: ";
                cin>>ch;
                switch(ch)
                 {
                        case 1: obj.create();
                                break;
                        case 2: obj.display();
                                break;
                        case 3: obj.Delete();
                                break;
                        case 4: key=obj.search();
                                if(key==0)
                                  cout<<"\n\tRecord Not Found...\n";
                                break;
                        case 5:
                                break;
```

```
                }
                cout<<"\n\t..... Do You Want to Continue in Main Menu: ";
                cin>>ans;
          }while(ans=='y'||ans=='Y');
return 1;
 }
```

Output

```
        ***** Student Information *****
        1. Create
        2. Display
        3. Delete
        4. Search
        5. Exit
        ..... Enter Your Choice: 1

        Enter Roll No of Student       : 1

        Enter a Name of Student       : a

        Enter a Division of Student : 1

        Enter a Address of Student  : pune

        Do You Want to Add More Records: y

        Enter Roll No of Student       : 2

        Enter a Name of Student       : b

        Enter a Division of Student : 1

        Enter a Address of Student  : pune

        Do You Want to Add More Records: y

        Enter Roll No of Student       : 3

        Enter a Name of Student       : c

        Enter a Division of Student : 2

        Enter a Address of Student  : pune
```

Do You Want to Add More Records: n

..... Do You Want to Continue in Main Menu: y

***** Student Information *****
1. Create
2. Display
3. Delete
4. Search
5. Exit
..... Enter Your Choice: 2

The Content of File are:

Roll	Name	Div	Address
1	a	1	pune
2	b	1	pune
3	c	2	pune

..... Do You Want to Continue in Main Menu: y

***** Student Information *****
1. Create
2. Display
3. Delete
4. Search
5. Exit
..... Enter Your Choice: 4

Enter a Roll No: 1

Record Found...

Roll	Name	Div	Address
1	a	1	pune

Record Not Found...

..... Do You Want to Continue in Main Menu: n

Assignment 12

```
#include<iostream>
#include<iomanip>
#include<fstream>
#include<cstring>

using namespace std;

class EMP_CLASS
{
 typedef struct EMPLOYEE
 {
 char name[10];
 int emp_id;
 int salary;
 }Rec;

 typedef struct INDEX
 {
 int emp_id;
 int position;
 }Ind_Rec;

 Rec Records;
 Ind_Rec Ind_Records;

 public:
  EMP_CLASS();
  void Create();
  void Display();
  void Update();
  void Delete();
  void Append();
  void Search();
};

EMP_CLASS::EMP_CLASS()//constructor
 {
  strcpy(Records.name,"");
 }

void EMP_CLASS::Create()
{
```

```
 int i,j;
 char ch='y';
 fstream seqfile;
 fstream indexfile;
 i=0;
 indexfile.open("IND.DAT",ios::out);
 seqfile.open("EMP.DAT",ios::out);
 do
 {
  cout<<"\n Enter Name: ";
  cin>>Records.name;
  cout<<"\n Enter Emp_ID: ";
  cin>>Records.emp_id;
  cout<<"\n Enter Salary: ";
  cin>>Records.salary;
  seqfile.write((char*)&Records,sizeof(Records));
  Ind_Records.emp_id=Records.emp_id;
  Ind_Records.position=i;
  indexfile.write((char*)&Ind_Records,sizeof(Ind_Records));
  i++;
  cout<<"\nDo you want to add more records?";
  cin>>ch;
  }while(ch=='y');
  seqfile.close();
  indexfile.close();
}

void EMP_CLASS::Display()
{
 fstream seqfile;
 fstream indexfile;
 int n,i,j;
 seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
 indexfile.open("IND.DAT",ios::in|ios::out|ios::binary);
 indexfile.seekg(0,ios::beg);
 seqfile.seekg(0,ios::beg);
 cout<<"\n The Contents of file are ..."<<endl;
 i=0;
 while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
 {

   i=Ind_Records.position*sizeof(Rec);//getting pos from index file
   seqfile.seekg(i,ios::beg);//seeking record of that pos from seq.file
```

```
  seqfile.read((char *)&Records,sizeof(Records));//reading record

  if(Records.emp_id!=-1)//if rec. is not deleted logically
  {   //then display it
  cout<<"\nName: "<<Records.name<<flush;
  cout<<"\nEmp_ID: "<<Records.emp_id;
  cout<<"\nSalary: "<<Records.salary;
  cout<<"\n";
          }

 }
 seqfile.close();
 indexfile.close();
}

void EMP_CLASS::Update()
{
 int pos,id;
 char New_name[10];
 int New_emp_id;
 int New_salary;
 cout<<"\n For updation,";
 cout<<"\n Enter the Emp_ID for for searching ";
 cin>>id;
 fstream seqfile;
 fstream indexfile;
 seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
 indexfile.open("IND.DAT",ios::in|ios::out|ios::binary);
 indexfile.seekg(0,ios::beg);

 pos=-1;
 //reading index file for getting the index
 while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
 {
  if(id==Ind_Records.emp_id)//the desired record is found
  {
   pos=Ind_Records.position;//getting the position
   break;
  }
 }
 if(pos==-1)
 {
```

```
 cout<<"\n The record is not present in the file";
 return;
}
else
{
 cout<<"\n Enter the values for updation...";
 cout<<"\n Name: ";cin>>New_name;
 cout<<"\n Salary: ";cin>>New_salary;
//calculating the position of record in seq. file using the pos of ind. file
 int offset=pos*sizeof(Rec);
 seqfile.seekp(offset);//seeking the desired record for modification
 strcpy(Records.name,New_name);//can be updated
 Records.emp_id=id;//It's unique id,so don't change
 Records.salary=New_salary;//can be updated
 seqfile.write((char*)&Records,sizeof(Records))<<flush;
 cout<<"\n The record is updated!!!";
}
seqfile.close();
indexfile.close();

}
void EMP_CLASS::Delete()
{
 int id,pos;
 cout<<"\n For deletion,";
 cout<<"\n Enter the Emp_ID for for searching ";
 cin>>id;
 fstream seqfile;
 fstream indexfile;
 seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
 indexfile.open("IND.DAT",ios::in|ios::out|ios::binary);
 seqfile.seekg(0,ios::beg);
 indexfile.seekg(0,ios::beg);
 pos=-1;
 //reading index file for getting the index
 while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
 {
  if(id==Ind_Records.emp_id) //desired record is found
  {
   pos=Ind_Records.position;
   Ind_Records.emp_id=-1;
   break;
  }
```

```
}
if(pos==-1)
{
 cout<<"\n The record is not present in the file";
 return;
}
//calculating the position of record in seq. file using the pos of ind. file
int offset=pos*sizeof(Rec);
seqfile.seekp(offset);//seeking the desired record for deletion
strcpy(Records.name,"");
Records.emp_id=-1; //logical deletion
Records.salary=-1; //logical deletion
seqfile.write((char*)&Records,sizeof(Records))<<flush;//writing deleted status
        //From index file also the desired record gets deleted as follows

offset=pos*sizeof(Ind_Rec);//getting position in index file
indexfile.seekp(offset); //seeking that record
Ind_Records.emp_id=-1; //logical deletion of emp_id
Ind_Records.position=pos;//position remain unchanged
indexfile.write((char*)&Ind_Records,sizeof(Ind_Records))<<flush;
seqfile.seekg(0);
indexfile.close();
seqfile.close();
cout<<"\n The record is Deleted!!!";
}
void EMP_CLASS::Append()
{
fstream seqfile;
fstream indexfile;
int pos;
indexfile.open("IND.DAT",ios::in|ios::binary);
indexfile.seekg(0,ios::end);
pos=indexfile.tellg()/sizeof(Ind_Records);
indexfile.close();

indexfile.open("IND.DAT",ios::app|ios::binary);
seqfile.open("EMP.DAT",ios::app|ios::binary);

cout<<"\n Enter the record for appending";
cout<<"\nName: ";cin>>Records.name;
cout<<"\nEmp_ID: ";cin>>Records.emp_id;
cout<<"\nSalary: ";cin>>Records.salary;
seqfile.write((char*)&Records,sizeof(Records));//inserting rec at end in seq. file
```

```
 Ind_Records.emp_id=Records.emp_id;       //inserting rec at end in ind. file

 Ind_Records.position=pos;                //at calculated pos
 indexfile.write((char*)&Ind_Records,sizeof(Ind_Records))<<flush;
 seqfile.close();
 indexfile.close();
 cout<<"\n The record is Appended!!!";
}
void EMP_CLASS::Search()
{
 fstream seqfile;
 fstream indexfile;
 int id,pos,offset;
 cout<<"\n Enter the Emp_ID for searching the record ";
 cin>>id;
 indexfile.open("IND.DAT",ios::in|ios::binary);
 pos=-1;
 //reading index file to obtain the index of desired record
 while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
 {
  if(id==Ind_Records.emp_id)//desired record found
  {
   pos=Ind_Records.position;//seeking the position
   break;
  }
 }
  if(pos==-1)
  {
  cout<<"\n Record is not present in the file";
  return;
  }
//calculate offset using position obtained from ind. file
  offset=pos*sizeof(Records);
  seqfile.open("EMP.DAT",ios::in|ios::binary);
//seeking the record from seq. file using calculated offset
  seqfile.seekg(offset,ios::beg);//seeking for reading purpose
  seqfile.read((char *)&Records,sizeof(Records));
  if(Records.emp_id==-1)
  {
  cout<<"\n Record is not present in the file";
  return;
  }
  else //emp_id=desired record's id
```

```
 {
 cout<<"\n The Record is present in the file and it is...";
 cout<<"\n Name: "<<Records.name;
 cout<<"\n Emp_ID: "<<Records.emp_id;
 cout<<"\n Salary: "<<Records.salary;
 }
 seqfile.close();
 indexfile.close();
}

int main()
{
 EMP_CLASS List;
 char ans;
 int choice;
 do
 {
 cout<<"\n                Main Menu                "<<endl;
 cout<<"\n 1.Create";
 cout<<"\n 2.Display";
 cout<<"\n 3.Update";
 cout<<"\n 4.Delete";
 cout<<"\n 5.Append";
 cout<<"\n 6.Search";
 cout<<"\n 7.Exit";
 cout<<"\n Enter your choice: ";
 cin>>choice;
 switch(choice)
 {
 case 1:List.Create();
          break;
 case 2:List.Display();
          break;
 case 3:List.Update();
          break;
 case 4:List.Delete();
          break;
 case 5:List.Append();
          break;
 case 6:List.Search();
          break;

 }
```

```
 cout<<"\n\t Do you want to go back to Main Menu?";
 cin>>ans;
 }while(ans=='y');
}
```

Output

```
        Main Menu

 1.Create
 2.Display
 3.Update
 4.Delete
 5.Append
 6.Search
 7.Exit
 Enter your choice: 1

 Enter Name: a

 Enter Emp_ID: 1

 Enter Salary: 4000

Do you want to add more records?y

 Enter Name: b

 Enter Emp_ID: 2

 Enter Salary: 4800

Do you want to add more records?y

 Enter Name: c

 Enter Emp_ID: 3

 Enter Salary: 5000

Do you want to add more records?n

        Do you want to go back to Main Menu?y
```

Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice: 2

The Contents of file are ...

Name: a
Emp_ID: 1
Salary: 4000

Name: b
Emp_ID: 2
Salary: 4800

Name: c
Emp_ID: 3
Salary: 5000

Do you want to go back to Main Menu?y

Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice: 3

For updation,
Enter the Emp_ID for for searching 3

Enter the values for updation...
Name: c

Salary: 5200

The record is updated!!!
Do you want to go back to Main Menu?y

Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice: 4

For deletion,
Enter the Emp_ID for for searching 2

The record is Deleted!!!
Do you want to go back to Main Menu?y

Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice: 5

Enter the record for appending
Name: a

Emp_ID: 1

Salary: 4500

The record is Appended!!!
Do you want to go back to Main Menu?y

Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice: 6

Enter the Emp_ID for searching the record 3

The Record is present in the file and it is...
Name: c
Emp_ID: 3
Salary: 5200
Do you want to go back to Main Menu?n