Experiment no 1

1. 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 any collisionhandling techniques. Operations- 1. Insert 2. Search 3. Display

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

 struct node
{
 int value;
              node* next;
}*HashTable[10];

class hashing
{
public:
hashing()
{
for(int i=0 ; i<10 ; i++)
{
                            HashTable[i]=NULL;
              }
 }

int HashFunction(int value)
{
  return (value%10);
 }
 node* create_node(int x)
{
               node* temp=new node;
              temp->next=NULL;
              temp->value=x;
              return temp;
 }

 void display()
{
               for(int i=0 ; i< 10; i++)
{
                            node * temp=new node;
                            temp=HashTable[i];
                            cout<<"a["<<i<<"] : ";
                            while(temp !=NULL)
```

```
{
                              cout<<" ->"<<temp->value;
                              temp=temp->next;
                    }
                    cout<<"\n";
}
 }

int searchElement(int value)
{
          bool flag = false;
         int hash_val = HashFunction(value);
         node* entry = HashTable[hash_val];
         cout<<"\nElement found at : ";
         while (entry != NULL)
          {
                    if (entry->value==value)
                     {
                              cout<<hash_val<<" : "<<entry->value<<endl;
                              flag = true;
                     }
                    entry = entry->next;
         }
         if (!flag)
         return -1;
 }


 void insertElement(int value)
{
          int hash_val = HashFunction(value);
         node* temp=new node;
         node* head=new node;
         head = create_node(value);
         temp=HashTable[hash_val];
         if (temp == NULL)
                      {
                     HashTable[hash_val] =head;
                     }
         else
{
          while (temp->next != NULL)
         {
                     temp = temp->next;
```

```
                              }
                            temp->next =head;
              }
 }
};
int main()
{
              int ch;
 int data,search,del;
              hashing h;
              do
{
cout<<"\nTelephone : \n1.Insert \n2.Display \n3.Search  \n4.Exit";
                            cin>>ch;
                            switch(ch)
{
                                          case 1:cout<<"\nEnter phone no. to be inserted : ";
                                                        cin>>data;
h.insertElement(data);
                                                         break;
                                          case 2:h.display();
                                                        break;
                                          case 3:cout<<"\nEnter the no to be searched : ";
                                                        cin>>search;

                                                         if (h.searchElement(search) == -1)
                                                        {
                                                                      cout<<"No element found at key ";
                                                                       continue;
                                                        }
                                                        break;

                                                        }
                            }while(ch!=5);
                            return 0;

}
```

Experiment no 2

2. 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 any collisionhandling techniques. Operations –

1. Insert 2. Delete 3. Display

```
#include<iostream>

#include<string.h>

using namespace std;


 struct node

{

 int value;

              node* next;

}*HashTable[10];


class hashing

{

public:

hashing()

{

for(int i=0 ; i<10 ; i++)

{

                              HashTable[i]=NULL;

              }

 }


int HashFunction(int value)

{

  return (value%10);

 }

 node* create_node(int x)
```

```
{
         node* temp=new node;
        temp->next=NULL;
        temp->value=x;
        return temp;
 }

 void display()
{
         for(int i=0 ; i< 10; i++)
{
                node * temp=new node;
                temp=HashTable[i];
                cout<<"a["<<i<<"] : ";
                while(temp !=NULL)
{
                        cout<<" ->"<<temp->value;
                        temp=temp->next;
                 }
                cout<<"\n";
}
 }

void deleteElement(int value)
{
         int hash_val = HashFunction(value);
        node* entry = HashTable[hash_val];
        if (entry == NULL )
        {
         cout<<"No Element found ";
                 return;
        }

        if(entry->value==value)
{
                HashTable[hash_val]=entry->next;
                return;
        }
        while ((entry->next)->value != value)
        {
           entry = entry->next;
        }
        entry->next=(entry->next)->next;
 }

 void insertElement(int value)
{
         int hash_val = HashFunction(value);
        node* temp=new node;
        node* head=new node;
        head = create_node(value);
        temp=HashTable[hash_val];
        if (temp == NULL)
                      {
                 HashTable[hash_val] =head;
                      }
        else
{
```

```
               while (temp->next != NULL)
              {
                              temp = temp->next;
                               }
                             temp->next =head;
              }
 }
};
int main()
{
              int ch;
 int data,search,del;
              hashing h;
              do
{
cout<<"\nTelephone : \n1.Insert \n2.Display \n3.Delete \n4.Exit";
                            cin>>ch;
                            switch(ch)
{
                                          case 1:cout<<"\nEnter phone no. to be inserted : ";
                                                        cin>>data;
h.insertElement(data);
                                                         break;
                                          case 2:h.display();
                                                        break;

                                          case 3:cout<<"\nEnter the phno. to be deleted : ";
                                                         cin>>del;
                                                         h.deleteElement(del);
```

```
                          cout<<"Phno. Deleted"<<endl;
                         break;
                         }
                }while(ch!=5);
                return 0;

        }
```

Exp3 Experiment no 3

Implement all the functions of a dictionary (ADT) using hashing and handle collisionsusing chaining with / without replacement. Data: Set of (key, value) pairs, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Display(key)

//Dictionary ADT by Chaining with or without replacement.

```
#include<iostream>
#include<string.h>
using namespace std;
class HashFunction
 {
               typedef struct hash
              {
                             long key;
                             char name[10];
}hash;
 hash h[10];
                public:
 HashFunction();
                            void insert();
                            void display();
                            int find(long);
 void Delete(long);
 };
HashFunction::HashFunction()
 {
              int i;
              for(i=0;i<10;i++)
               {
                            h[i].key=-1;
                            strcpy(h[i].name,"NULL");
               }
 }
void HashFunction::Delete(long k)
 {
 int index=find(k);
 if(index==-1)
               {
  cout<<"\n\tKey Not Found";
               }
 else
               {
                            h[index].key=-1;
                            strcpy(h[index].name,"NULL");
  cout<<"\n\tKey is Deleted";
               }
 }
```

```
int HashFunction::find(long k)
 {
            int i;
            for(i=0;i<10;i++)
            {
                        if(h[i].key==k)
                       {
cout<<"\n\t"<<h[i].key<<" is Found at "<<i<<" Location With Name "<<h[i].name;
                        return i;
                        }
             }
 if(i==10)
             {
                       return -1;
  }
 }
void HashFunction::display()
 {
            int i;
            cout<<"\n\t\tKey\t\tName";
            for(i=0;i<10;i++)
             {
                        cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t\t"<<h[i].name;
             }
 }
void HashFunction::insert()
 {
            char ans,n[10],ntemp[10];
            long k,temp;
 int v,hi,cnt=0,flag=0,i;
 do
            {
                        if(cnt>=10)
                       {
                                   cout<<"\n\tHash Table is FULL";
                                   break;
                        }
                       cout<<"\n\tEnter a Telephone No: ";
                        cin>>k;
                        cout<<"\n\tEnter a Client Name: ";
                        cin>>n;
                       hi=k%10;// hash function
                       if(h[hi].key==-1)
                         {
                                   h[hi].key=k;
                                    strcpy(h[hi].name,n);
                        }
                       else
```

```
{
    if(h[hi].key%10!=hi)
    {
        temp=h[hi].key;
        strcpy(ntemp,h[hi].name);
        h[hi].key=k;
        strcpy(h[hi].name,n);
        for(i=hi+1;i<10;i++)
        {
            if(h[i].key==-1)
            {
                h[i].key=temp;
                strcpy(h[i].name,ntemp);
                flag=1;
                break;
            }
        }
        for(i=0;i<hi && flag==0;i++)
        {
            if(h[i].key==-1)
            {
                h[i].key=temp;
                strcpy(h[i].name,ntemp);
                break;
            }
        }
    }
    else
    {
        for(i=hi+1;i<10;i++)
        {
            if(h[i].key==-1)
            {
                h[i].key=k;
                strcpy(h[i].name,n);
                flag=1;
                break;
            }
        }
        for(i=0;i<hi && flag==0;i++)
        {
            if(h[i].key==-1)
            {
                h[i].key=k;
                strcpy(h[i].name,n);
                break;
            }
        }
```

```
                              }
                        }
                        flag=0;
                        cnt++;
                        cout<<"\n\t..... Do You Want to Insert More Key: y/n";
                        cin>>ans;
               }while(ans=='y'||ans=='Y');
  }
int main()
  {
 long k;
 int ch,index;
 char ans;
 HashFunction obj;
               do
                {
                         cout<<"\n\t*** Telephone (ADT) *****";
                        cout<<"\n\t1. Insert\n\t2. Display\n\t3. Find\n\t4. Delete\n\t5. Exit";
                        cout<<"\n\t..... Enter Your Choice: ";
                         cin>>ch;
                         switch(ch)
                        {
                              case 1:  obj.insert();
                                       break;
                              case 2: obj.display();
                                        break;
                              case 3: cout<<"\n\tEnter a Key Which You Want to Search: ";
                                       cin>>k;
                                       index=obj.find(k);
                                       if(index==-1)
                                        {
                                              cout<<"\n\tKey Not Found";
                                       }
                                        break;
                              case 4: cout<<"\n\tEnter a Key Which You Want to Delete: ";
                                        cin>>k;
                                       obj.Delete(k);
                                       break;
                              case 5:
                                        break;
                        }
                        cout<<"\n\t..... Do You Want to Continue in Main Menu:y/n ";
                        cin>>ans;
                }while(ans=='y'||ans=='Y');
  }
```

----------OUTPUT----------

```
*** Telephone (ADT) *****
        1. Insert
        2. Display
        3. Find
        4. Delete
        5. Exit
        ..... Enter Your Choice: 1
        Enter a Telephone No: 2456
        Enter a Client Name: abc
        ..... Do You Want to Insert More Key: y/ny
        Enter a Telephone No: 2466
        Enter a Client Name: sdf
        ..... Do You Want to Insert More Key: y/nn
        ..... Do You Want to Continue in Main Menu:y/n y
        *** Telephone (ADT) *****
        1. Insert
        2. Display
        3. Find
        4. Delete
        5. Exit
        ..... Enter Your Choice: 2
        Key             Name
        h[0]    -1              NULL
        h[1]    -1              NULL
        h[2]    -1              NULL
        h[3]    -1              NULL
        h[4]    -1              NULL
        h[5]    -1              NULL
        h[6]    2456            abc
        h[7]    2466            sdf
        h[8]    -1              NULL
        h[9]    -1              NULL
        ..... Do You Want to Continue in Main Menu:y/n y
        *** Telephone (ADT) *****
        1. Insert
        2. Display
        3. Find
        4. Delete
        5. Exit
        ..... Enter Your Choice: 1
        Enter a Telephone No: 2457
        Enter a Client Name: klh
        ..... Do You Want to Insert More Key: y/ny
        Enter a Telephone No: 8888
        Enter a Client Name: opp
        ..... Do You Want to Insert More Key: y/nn
        ..... Do You Want to Continue in Main Menu:y/n y
        *** Telephone (ADT) *****
```

```
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 2
Key             Name
h[0]    -1              NULL
h[1]    -1              NULL
h[2]    -1              NULL
h[3]    -1              NULL
h[4]    -1              NULL
h[5]    -1              NULL
h[6]    2456            abc
h[7]    2457            klh
h[8]    8888            opp
h[9]    2466            sdf
..... Do You Want to Continue in Main Menu:y/n
```

Exp4: Experiment no 4

Implement all the functions of a dictionary (ADT) using hashing and handle collisionsusing chaining with / without replacement. Data: Set of (key, value) pairs, Keys must be comparable, Keys must be unique. Standard Operations: Insert(key, value), Delete(key), Display(key)

//Dictionary ADT by Chaining with or without replacement.

```
#include<iostream>
#include<string.h>
using namespace std;
class HashFunction
  {
               typedef struct hash
              {
                             long key;
                             char name[10];
}hash;
 hash h[10];
                public:
 HashFunction();
                            void insert();
                            void display();
                            int find(long);
 void Delete(long);
  };
HashFunction::HashFunction()
  {
              int i;
              for(i=0;i<10;i++)
               {
                            h[i].key=-1;
                            strcpy(h[i].name,"NULL");
               }
  }
void HashFunction::Delete(long k)
  {
 int index=find(k);
 if(index==-1)
               {
  cout<<"\n\tKey Not Found";
               }
 else
               {
                            h[index].key=-1;
```

```
                        strcpy(h[index].name,"NULL");
  cout<<"\n\tKey is Deleted";
             }
  }
int HashFunction::find(long k)
  {
            int i;
            for(i=0;i<10;i++)
            {
                         if(h[i].key==k)
                        {
cout<<"\n\t"<<h[i].key<<" is Found at "<<i<<" Location With Name "<<h[i].name;
                         return i;
                         }
             }
 if(i==10)
             {
                        return -1;
   }
  }
void HashFunction::display()
  {
            int i;
            cout<<"\n\t\tKey\t\tName";
            for(i=0;i<10;i++)
             {
                         cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t\t"<<h[i].name;
             }
  }
void HashFunction::insert()
  {
            char ans,n[10],ntemp[10];
            long k,temp;
 int v,hi,cnt=0,flag=0,i;
 do
            {
                         if(cnt>=10)
                        {
                                    cout<<"\n\tHash Table is FULL";
                                    break;
                         }
                        cout<<"\n\tEnter a Telephone No: ";
                         cin>>k;
                         cout<<"\n\tEnter a Client Name: ";
                         cin>>n;
                        hi=k%10;// hash function
                        if(h[hi].key==-1)
                          {
                                    h[hi].key=k;
                                     strcpy(h[hi].name,n);
                         }
                        else
                        {
                                     if(h[hi].key%10!=hi)
                                    {
                                                 temp=h[hi].key;
                                                strcpy(ntemp,h[hi].name);
                                                h[hi].key=k;
                                                strcpy(h[hi].name,n);
                                                for(i=hi+1;i<10;i++)
                                                 {
                                                            if(h[i].key==-1)
                                                             {
                                                                        h[i].key=temp;
                                                                        strcpy(h[i].name,ntemp);
                                                                        flag=1;
                                                                        break;
                                                            }
                                                 }
                                                for(i=0;i<hi && flag==0;i++)
                                                {
                                                            if(h[i].key==-1)
                                                            {
                                                                        h[i].key=temp;
                                                                        strcpy(h[i].name,ntemp);
                                                                        break;
                                                             }
                                                }
                                     }
                                    else
                                    {
                                                for(i=hi+1;i<10;i++)
                                                {
                                                            if(h[i].key==-1)
                                                             {
                                                                         h[i].key=k;
                                                                        strcpy(h[i].name,n);
                                                                         flag=1;
```

```
                                                        break;
                                                 }
                                          }
                                          for(i=0;i<hi && flag==0;i++)
                                           {
                                                 if(h[i].key==-1)
                                                 {
                                                        h[i].key=k;
                                                        strcpy(h[i].name,n);
                                                        break;
                                                 }
                                           }
                                   }
                           }
                           flag=0;
                           cnt++;
                           cout<<"\n\t..... Do You Want to Insert More Key: y/n";
                           cin>>ans;
                    }while(ans=='y'||ans=='Y');
      }
     int main()
      {
      long k;
      int ch,index;
      char ans;
      HashFunction obj;
                    do
                     {
                            cout<<"\n\t*** Telephone (ADT) *****";
                           cout<<"\n\t1. Insert\n\t2. Display\n\t3. Find\n\t4. Delete\n\t5. Exit";
                           cout<<"\n\t..... Enter Your Choice: ";
                            cin>>ch;
                            switch(ch)
                           {
                                   case 1:  obj.insert();
                                          break;
                                   case 2: obj.display();
                                           break;
                                   case 3: cout<<"\n\tEnter a Key Which You Want to Search: ";
                                           cin>>k;
                                          index=obj.find(k);
                                          if(index==-1)
                                           {
                                                 cout<<"\n\tKey Not Found";
```

```
                                                }
                                                 break;
                                    case 4: cout<<"\n\tEnter a Key Which You Want to Delete: ";
                                                  cin>>k;
                                                 obj.Delete(k);
                                                 break;
                                    case 5:
                                                  break;
                        }
                        cout<<"\n\t..... Do You Want to Continue in Main Menu:y/n ";
                        cin>>ans;
            }while(ans=='y'||ans=='Y');
 }
```

```
----------OUTPUT----------
*** Telephone (ADT) *****
            1. Insert
            2. Display
            3. Find
            4. Delete
            5. Exit
            ..... Enter Your Choice: 1
            Enter a Telephone No: 2456
            Enter a Client Name: abc
            ..... Do You Want to Insert More Key: y/ny
            Enter a Telephone No: 2466
            Enter a Client Name: sdf
            ..... Do You Want to Insert More Key: y/nn
            ..... Do You Want to Continue in Main Menu:y/n y
            *** Telephone (ADT) *****
            1. Insert
            2. Display
            3. Find
            4. Delete
            5. Exit
            ..... Enter Your Choice: 2
            Key                     Name
            h[0]      -1                    NULL
            h[1]      -1                    NULL
            h[2]      -1                    NULL
            h[3]      -1                    NULL
            h[4]      -1                    NULL
            h[5]      -1                    NULL
```

```
h[6]    2456        abc
h[7]    2466        sdf
h[8]    -1          NULL
h[9]    -1          NULL
..... Do You Want to Continue in Main Menu:y/n y
*** Telephone (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 1
Enter a Telephone No: 2457
Enter a Client Name: klh
..... Do You Want to Insert More Key: y/ny
Enter a Telephone No: 8888
Enter a Client Name: opp
..... Do You Want to Insert More Key: y/nn
..... Do You Want to Continue in Main Menu:y/n y
*** Telephone (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 2
Key         Name
h[0]    -1          NULL
h[1]    -1          NULL
h[2]    -1          NULL
h[3]    -1          NULL
h[4]    -1          NULL
h[5]    -1          NULL
h[6]    2456        abc
h[7]    2457        klh
h[8]    8888        opp
h[9]    2466        sdf
..... Do You Want to Continue in Main Menu:y/n
```

Exp5: Experiment no 5

A book consists of chapters, chapters consist of sections Construct a tree and print the nodes

Time Comp: O(n) Chapter insertion
Time Comp: O(n*m) Section insertion
Space Comp: O(n*m)

```
#include<iostream>
#include<stdlib.h>
#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(struct 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;
}
```

Exp6 and Exp7 and Exp8: Experiment no 6,7,8

Exp6: Beginning with an empty binary search tree, Construct binary search tree operations to be performed is

1. **Insert new node 2. Find number of nodes in longest path from root 3. Display**

Exp7: Beginning with an empty binary search tree, Construct binary search tree operations to be performed is 1. Insert new node 2. Change a tree so that the roles of the left and right pointers are swapped at every node 3. Display

Exp8: Beginning with an empty binary search tree, Construct binary search tree operations to be performed is 1. Insert new node 2. Search a value 3. Display

```
#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;
}
```

Exp9: Experiment no 9

Construct an expression tree from the given prefix expression eg. +--a*bc/def and Traverse it using post order traversal (non recursive) and then delete the entire tree.

```
/*
 Construct an expression tree from the given prefix expression eg. +--
a*bc/def and traverse it using post order traversal (non recursive) and
then delete the entire tree.
 */

#include<iostream>
#include<stack>
#include<string>

using namespace std;

// Structure of the node of the expression tree
struct Node {
    char data;
    Node* left;
    Node* right;
};

// Function to create a new node of the expression tree
Node* newNode(char data) {
    Node* node = new Node;
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

// Function to construct an expression tree from the given prefix
expression
Node* constructExpressionTree(string prefix) {
    stack<Node*> s;
    int len = prefix.length();
    for (int i = len - 1; i >= 0; i--) {
        char ch = prefix[i];
        if (isdigit(ch) || isalpha(ch)) {
            s.push(newNode(ch));
        } else {
            Node* node = newNode(ch);
            node->left = s.top();
            s.pop();
            node->right = s.top();
            s.pop();
            s.push(node);
        }
    }
    return s.top();
}
```

```
// Function to traverse the expression tree in postorder without recursion
void postorderTraversal(Node* root) {
    if (root == NULL) return;
    stack<Node*> s;
    s.push(root);
    Node* prev = NULL;
    while (!s.empty()) {
        Node* curr = s.top();
        if (prev == NULL || prev->left == curr || prev->right == curr) {
            if (curr->left) {
                s.push(curr->left);
            } else if (curr->right) {
                s.push(curr->right);
            }
        } else if (curr->left == prev) {
            if (curr->right) {
                s.push(curr->right);
            }
        } else {
            cout << curr->data << " ";
            s.pop();
        }
        prev = curr;
    }
    cout << endl;
}

// Function to delete the expression tree
void deleteTree(Node* root) {
    if (root == NULL) return;
    stack<Node*> s;
    s.push(root);
    Node* prev = NULL;
    while (!s.empty()) {
        Node* curr = s.top();
        if (prev == NULL || prev->left == curr || prev->right == curr) {
            if (curr->left) {
                s.push(curr->left);
            } else if (curr->right) {
                s.push(curr->right);
            }
        } else if (curr->left == prev) {
            if (curr->right) {
                s.push(curr->right);
            }
        } else {
            delete curr;
            s.pop();
        }
        prev = curr;
    }
}

int main() {
    string prefix = "+--a*bc/def";
    Node* root = constructExpressionTree(prefix);
    cout << "Postorder Traversal: ";
    postorderTraversal(root);
    deleteTree(root);
    return 0;
}
```

Exp10 : Experiment no 10

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. 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.

```
#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(struct node);
         p->next=NULL;
         p->vertex=v[i];
         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(struct 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;
}
```

Exp11: Experiment no 11

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. Find Minimum cost of the graph.

```
#include<iostream>
using namespace std;

class tree
{
        int a[20][20],l,u,w,i,j,v,e,visited[20];
public:
                void input();
                void display();
                void minimum();
};

void tree::input()
{
        cout<<"Enter the no. of branches: ";
        cin>>v;

        for(i=0;i<v;i++)
        {
                visited[i]=0;
                for(j=0;j<v;j++)
                {
                        a[i][j]=999;
                }
        }

        cout<<"\nEnter the no. of connections: ";
        cin>>e;

        for(i=0;i<e;i++)
        {
                cout<<"Enter the end branches of connections:  "<<endl;
                cin>>l>>u;
                cout<<"Enter the phone company charges for this connection:
";
                cin>>w;
                a[l-1][u-1]=a[u-1][l-1]=w;
        }
}

void tree::display()
{
        cout<<"\nAdjacency matrix:";
        for(i=0;i<v;i++)
        {
                cout<<endl;
                for(j=0;j<v;j++)
                {
                        cout<<a[i][j]<<"   ";
                }
                cout<<endl;
        }
}

void tree::minimum()
{
        int p=0,q=0,total=0,min;
        visited[0]=1;
        for(int count=0;count<(v-1);count++)
        {
                min=999;
                for(i=0;i<v;i++)
                {
                        if(visited[i]==1)
                        {
                                for(j=0;j<v;j++)
                                {
                                        if(visited[j]!=1)
                                        {
                                                if(min > a[i][j])
                                                {
                                                        min=a[i][j];
                                                        p=i;
                                                        q=j;
                                                }
                                        }
                                }
                        }
                }
                visited[p]=1;
                visited[q]=1;
                total=total + min;
                cout<<"Minimum cost connection is"<<(p+1)<<" -> "<<(q+1)<<"
with charge : "<<min<< endl;

        }
        cout<<"The minimum total cost of connections of all branches is:
"<<total<<endl;
}

int main()
{
        int ch;
        tree t;
        do
        {
                cout<<"==========PRIM'S ALGORITHM================="<<endl;
                cout<<"\n1.INPUT\n \n2.DISPLAY\n \n3.MINIMUM\n"<<endl;
                cout<<"Enter your choice :"<<endl;
                cin>>ch;

        switch(ch)
        {
        case 1: cout<<"*******INPUT YOUR VALUES*******"<<endl;
               t.input();
               break;

        case 2: cout<<"*******DISPLAY THE CONTENTS********"<<endl;
               t.display();
               break;

        case 3: cout<<"*********MINIMUM************"<<endl;
               t.minimum();
               break;
        }

        }while(ch!=4);
        return 0;
}
```

```
-----------------OUTPUT-----------------------------
==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :
1
*******INPUT YOUR VALUES*******
Enter the no. of branches: 4
Enter the no. of connections: 5
Enter the end branches of connections:
1
2
Enter the phone company charges for this connection:  100
Enter the end branches of connections:
1
3
Enter the phone company charges for this connection:  150
Enter the end branches of connections:
3
4
Enter the phone company charges for this connection:  300
Enter the end branches of connections:
2
4
Enter the phone company charges for this connection:  250
Enter the end branches of connections:
1
4
Enter the phone company charges for this connection:  50
==========PRIM'S ALGORITHM=================
1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :
2
*******DISPLAY THE CONTENTS********

Adjacency matrix:
999   100   150   50

100   999   999   250

150   999   999   300

50   250   300   999
==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :
3
*********MINIMUM************
Minimum cost connection is1 -> 4  with charge : 50
Minimum cost connection is1 -> 2  with charge : 100
Minimum cost connection is1 -> 3  with charge : 150
The minimum total cost of connections of all branches is: 300
==========PRIM'S ALGORITHM=================

1.INPUT

2.DISPLAY

3.MINIMUM

Enter your choice :
```

Exp 12: Experiment no 12

Given sequence k = k1 < … <kn of n sorted keys, with a search probability pi for each key ki . Build the O p t i m a l Binary search tree that has the least search cost.

```
#include<iostream>

using namespace std;

void con_obst(void);

void print(int,int);

float a[20],b[20],wt[20][20],c[20][20];

int r[20][20],n;

int main()

 {

        int i;

        cout<<"\n****** PROGRAM FOR OBST ******\n";

        cout<<"\nEnter the no. of nodes : ";

        cin>>n;cout<<"\nEnter the probability for successful search :: ";

        cout<<"\n————————————————\n";

        for(i=1;i<=n;i++)

         {

                cout<<"p["<<i<<"]";

                cin>>a[i];

         }

        cout<<"\nEnter the probability for unsuccessful search :: ";

        cout<<"\n—————————————————\n";

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

         {

                cout<<"q["<<i<<"]";

                cin>>b[i];

         }

        con_obst();
        print(0,n);
        cout<<endl;
}
void con_obst(void)
{
        int i,j,k,l,min;
        for(i=0;i<n;i++)
         { //Initialisation
                c[i][i]=0.0;
                r[i][i]=0;
                wt[i][i]=b[i];
                // for j-i=1 can be j=i+1
                wt[i][i+1]=b[i]+b[i+1]+a[i+1];
                c[i][i+1]=b[i]+b[i+1]+a[i+1];
                r[i][i+1]=i+1;
         }
        c[n][n]=0.0;
        r[n][n]=0;
        wt[n][n]=b[n];
        //for j-i=2,3,4....,n
        for(i=2;i<=n;i++)
         {
                for(j=0;j<=n-i;j++)
                 {
                        wt[j][j+i]=b[j+i]+a[j+i]+wt[j][j+i-1];
                        c[j][j+i]=9999;
                        for(l=j+1;l<=j+i;l++)
                         {
                                if(c[j][j+i]>(c[j][l-1]+c[l][j+i]))
```

```
                                    {
                                            c[j][j+i]=c[j][l-1]+c[l][j+i];
                                            r[j][j+i]=l;
                                    }
                             }
                            c[j][j+i]+=wt[j][j+i];
                     }
                    cout<<endl;
          }
          cout<<"\n\nOptimal BST is :: ";
          cout<<"\nw[0]["<<n<<"] :: "<<wt[0][n];
          cout<<"\nc[0]["<<n<<"] :: "<<c[0][n];
          cout<<"\nr[0]["<<n<<"] :: "<<r[0][n];
 }
void print(int l1,int r1)
 {
          if(l1>=r1)
                    return;
          if(r[l1][r[l1][r1]-1]!=0)
                    cout<<"\n Left child of "<<r[l1][r1]<<" :: "<<r[l1][r[l1][r1]-1];
          if(r[r[l1][r1]][r1]!=0)
                    cout<<"\n Right child of "<<r[l1][r1]<<" :: "<<r[r[l1][r1]][r1];
          print(l1,r[l1][r1]-1);
          print(r[l1][r1],r1);
          return;
}
```

------------------------ OUTPUT -----------------------------

```
****** PROGRAM FOR OBST ******

Enter the no. of nodes : 4

Enter the probability for successful search ::

————————————————

p[1]3

p[2]3

p[3]1

p[4]1

Enter the probability for unsuccessful search ::

—————————————————

q[0]2

q[1]3

q[2]1

q[3]1

q[4]1

Optimal BST is ::

w[0][4] :: 16

c[0][4] :: 32

r[0][4] :: 2

 Left child of 2 :: 1

 Right child of 2 :: 3

 Right child of 3 :: 4
```

Exp13: ## Experiment no 13

A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Use Height balance tree.

```
#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;
}
```

Exp14 and Exp15: Exp 14,15

Exp14: Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum marks obtained in that subject. Use Heap data structure.

Exp15:Read the marks obtained by students of second year in an online examination of particular subject. Find out minimum marks obtained in that subject. Use Heap data structure.

```
#include<iostream>

using namespace std;

# define max 20

class stud

{

int mks[max];

public:

   stud()

   {

   for(int i=0;i<max;i++)

     mks[i]=0;

   }

   void insertheap(int tot);

   void displayheap(int tot);

   void showmax(int tot);

   void showmin();

};


void  stud::insertheap(int tot)

{

 for(int i=1;i<=tot;i++)

 {

  cout<<"enter marks";

  cin>>mks[i];
```

```
    int j=i;
    int par=j/2;
    while(mks[j]<=mks[par] && j!=0)
     {
        int tmp = mks[j];
        mks[j]=mks[par];
        mks[par]=tmp;
        j=par;
        par=j/2;
     }
 }
}

void stud::displayheap(int tot)
{
int i=1,space=6;
cout<<endl;
while(i<=tot)
{
    if(i==1 || i==2 || i==4 || i==8 || i==16)
    {
    cout<<endl<<endl;
    for(int j=0;j<space;j++)
         cout<<" ";
    space-=2;
    }
   cout<<" "<<mks[i];i++;


}
```

```
}



void stud::showmax(int tot)
{
int max1=mks[1];
for(int i=2;i<=tot;i++)
{
 if(max1<mks[i])
  max1= mks[i];
}
cout<<"Max marks:"<<max1;
}

void stud::showmin()
{
cout<<"Min marks:"<<mks[1];
}

int main()
{
int ch,cont,total,tmp;
int n;
stud s1;
do
{
cout<<endl<<"Menu";
cout<<endl<<"1.Read marks of the student ";
cout<<endl<<"2.Display  Min heap";
cout<<endl<<"3.Find Max Marks";
cout<<endl<<"4.Find Min Marks";
cout<<endl<<"Enter Choice";
cin>>ch;
switch(ch)
{
case 1:
     cout<<"how many students";
     cin>>total;
    s1.insertheap(total);
    break;
case 2:
    s1.displayheap(total);
    break;
case 3:    s1.showmax(total);
    break;
case 4:
    s1.showmin();
    break;
}
cout<<endl<<"do u want to continue?(1 for continue)";
cin>>cont;
}while(cont==1);
return 0;
}
```

------------------- OUTPUT-------------------------

Exp16: Experiment no 16

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 Students. If record of student does not exist an appropriate Message is displayed. Use sequential file to maintain the data.

```
#include<iostream>
#include<fstream>
#include<cstring>
using namespace std;
class tel
 {

 public:
 int rollNo,roll1;
 char name[10];
 char div;
 char address[20];
 void accept()
 {
  cout<<"\n\tEnter Roll Number : ";
  cin>>rollNo;
  cout<<"\n\tEnter the Name : ";
  cin>>name;
  cout<<"\n\tEnter the Division:";
  cin>>div;
  cout<<"\n\tEnter the Address:";
  cin>>address;
 }
        void accept2()
        {
               cout<<"\n\tEnter the Roll No. to modify : ";
               cin>>rollNo;
        }
        void accept3()
        {
              cout<<"\n\tEnter the name to modify : ";
              cin>>name;
        }
        int getRollNo()
        {
         return rollNo;
        }
  void show()
  {

  cout<<"\n\t"<<rollNo<<"\t\t"<<name<<"\t\t"<<div<<"\t\t"<<address;
  }
};
int main()
{
 int i,n,ch,ch1,rec,start,count,add,n1,add2,start2,n2,y,a,b,on,oname,add3,start3,n3,y1,add4,start4,n4;
 char name[20],name2[20];
 tel t1;
 count=0;
 fstream g,f;
 do
 {
  cout<<"\n>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<";
  cout<<"\n1.Insert and overwrite\n2.Show\n3.Search & Edit(number)\n4.Search &
Edit(name)\n5.Search & Edit(onlynumber)\n6.Search & edit(only name)\n 7.Delete a Student Record\n
8.Exit\n\tEnter the Choice\t:";
  cin>>ch;
  switch(ch)
  {
  case 1:
   f.open("StuRecord.txt",ios::out);
   x:t1.accept();
   f.write((char*) &t1,(sizeof(t1)));
   cout<<"\nDo you want to enter more records?\n1.Yes\n2.No";
   cin>>ch1;
    if(ch1==1)
     goto x;
    else
    {
     f.close();
     break;
    }

  case 2:
   f.open("StuRecord.txt",ios::in);
   f.read((char*) &t1,(sizeof(t1)));
   //cout<<"\n\tRoll No.\t\tName \t\t Division \t\t Address";
   while(f)
   {
    t1.show();
    f.read((char*) &t1,(sizeof(t1)));
   }
   f.close();
   break;
  case 3:
   cout<<"\nEnter the roll number you want to find";
   cin>>rec;
   f.open("StuRecord.txt",ios::in|ios::out);
   f.read((char*)&t1,(sizeof(t1)));
   while(f)
   {
    if(rec==t1.rollNo)
    {
     cout<<"\nRecord found";
     add=f.tellg();
     f.seekg(0,ios::beg);
           start=f.tellg();
     n1=(add-start)/(sizeof(t1));
     f.seekp((n1-1)*sizeof(t1),ios::beg);
     t1.accept();
     f.write((char*) &t1,(sizeof(t1)));
     f.close();
     count++;
     break;
    }
    f.read((char*)&t1,(sizeof(t1)));
       }
   if(count==0)
          cout<<"\nRecord not found";
   f.close();
   break;

  case 4:
    cout<<"\nEnter the name you want to find and edit";
    cin>>name;
   f.open("StuRecord.txt",ios::in|ios::out);
   f.read((char*)&t1,(sizeof(t1)));
   while(f)
   {
    y=(strcmp(name,t1.name));
    if(y==0)
    {
     cout<<"\nName found";
     add2=f.tellg();
     f.seekg(0,ios::beg);
     start2=f.tellg();
    n2=(add2-start2)/(sizeof(t1));
    f.seekp((n2-1)*sizeof(t1),ios::beg);
    t1.accept();
    f.write((char*) &t1,(sizeof(t1)));
    f.close();
    break;
   }
        f.read((char*)&t1,(sizeof(t1)));
  }
  break;
     case 5:
          cout<<"\n\tEnter the roll number you want to modify";
          cin>>on;
          f.open("StuRecord.txt",ios::in|ios::out);
          f.read((char*) &t1,(sizeof(t1)));
          while(f)
          {
            if(on==t1.rollNo)
            {
              cout<<"\n\tNumber found";
              add3=f.tellg();
              f.seekg(0,ios::beg);
              start3=f.tellg();
              n3=(add3-start3)/(sizeof(t1));
              f.seekp((n3-1)*(sizeof(t1)),ios::beg);
              t1.accept2();
              f.write((char*)&t1,(sizeof(t1)));
              f.close();
              break;
            }
            f.read((char*)&t1,(sizeof(t1)));
         }
         break;
     case 6:
          cout<<"\nEnter the name you want to find and edit";
   cin>>name2;
  f.open("StuRecord.txt",ios::in|ios::out);
  f.read((char*)&t1,(sizeof(t1)));
  while(f)
  {
   y1=(strcmp(name2,t1.name));
   if(y1==0)
   {
    cout<<"\nName found";
```

```
    add4=f.tellg();
    f.seekg(0,ios::beg);
    start4=f.tellg();
    n4=(add4-start4)/(sizeof(t1));
    f.seekp((n4-1)*sizeof(t1),ios::beg);
    t1.accept3();
    f.write((char*) &t1,(sizeof(t1)));
    f.close();
    break;
   }
        f.read((char*)&t1,(sizeof(t1)));
  }
  break;
   case 7:
     int roll;
     cout<<"Please Enter the Roll No. of Student Whose Info You Want to Delete: ";
    cin>>roll;
    f.open("StuRecord.txt",ios::in);
    g.open("temp.txt",ios::out);
    f.read((char *)&t1,sizeof(t1));
    while(!f.eof())
    {
       if (t1.getRollNo() != roll)
          g.write((char *)&t1,sizeof(t1));
        f.read((char *)&t1,sizeof(t1));
    }
   cout << "The record with the roll no. " << roll << " has been deleted " << endl;
    f.close();
    g.close();
    remove("StuRecord.txt");
    rename("temp.txt","StuRecord.txt");
     break;
   case 8:
      cout<<"\n\tThank you";
      break;


       }
  }while(ch!=8);
}
```