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

#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;
}
```