```
/*
Convert given binary tree into threaded binary tree. Analyze time and space complexity of the
algorithm.
*/
#include<iostream>
using namespace std;

struct TTREE
{
   struct TTREE *lc;
   char lt;
   char data;
   char rt;
   struct TTREE *rc;
};

typedef struct TTREE ttree;

class myttree
{
   private :
       ttree *head;
   public :
       myttree();
       void create_btree();
       void insert_btree(ttree *);
      void thread_btree();
       void display_ttree();
       void display_btree();
};



myttree :: myttree()
{
   head = NULL;
}

void myttree :: insert_btree(ttree *node)
{
   if(head == NULL)
      head = node;
   else
   {
      int flag = 0;
      char ans;
      ttree *par;
      par = head;
      while(flag == 0)
      {
        cout<<"\nWhere to add (l/r) of "<<par->data<< " : ";
        cin>>ans;
        if(ans == 'l')
        {
           if(par->lc == NULL)
           {
              par->lc = node;
              flag = 1;
           }
           else
             par = par->lc;
        }
        else
        {
           if(par->rc == NULL)
           {
              par->rc = node;
              flag = 1;
           }
           else
             par = par->rc;
        }
      }
   }
}

void myttree :: create_btree()
{
   int i,n;
   ttree *node;
   cout<<"\nEnter the total no. of nodes in btree : ";
   cin>>n;
   for(i = 1; i<=n; i++)
   {
      node = new ttree;
      node->lc = NULL;  node->lt = 'f';
      node->rc = NULL;  node->rt = 'f';
      cout<<"\nEnter the data field of node "<<i<<" : ";
      cin>>node->data;
      insert_btree(node);
   }
   cout<<"\nBinary tree created successfully";
}


void preorder(ttree *node, ttree *A[],int & n)
{
   if(node != NULL)
   {
     A[n++] = node;
     preorder(node->lc,A,n);
     preorder(node->rc,A,n);
   }

}

void myttree :: thread_btree()
{
   ttree *A[20];
   ttree *node;
   int n = 0,i;
   if(head == NULL)
   {
      cout<<"\nBinary tree is empty";
   }
   else
   {
      preorder(head,A,n);
      cout<<"\nPreorder traversal is : ";
      for(i = 0 ; i < n ;i++)
      {
         cout<<A[i]->data<<"  ";
      }
      node = new ttree;
      node->lt = 'f';
      node->lc = head;
      node->data = '\0';
      node->rt = 't';
      node->rc = node;
      head = node;
      for(i = 0 ; i < n ;i++)
      {
         if(A[i]->lc == NULL)
         {
            if(i ==0)
              A[i]->lc = head;
            else
                A[i]->lc = A[i-1];
           A[i]->lt = 't';
         }
         if(A[i]->rc == NULL)
         {
            if(i == n - 1)
                A[i]->rc = head;
            else
                A[i]->rc = A[i+1];
            A[i]->rt = 't';
         }
      }
      cout<<"\nBinary Tree Threaded successfully\n";
   }
}

void myttree :: display_ttree()
{
   if(head == NULL)
     cout<<"\nBinary tree is empty";
   else
   {
      cout<<"Preorder Traversal of the tree : ";
      ttree *temp = head->lc;
      while( temp != head)
      {
         while(temp->lt == 'f')
         {
            cout<<temp->data<<" ";
            temp = temp->lc;
         }
         cout<<temp->data<<" ";
         while(temp->rt == 't' && temp->rc != head)
         {
            temp = temp->rc;
            cout<<temp->data<<" ";
         }
         if(temp->lt == 'f')
            temp = temp->lc;
         else
            temp = temp->rc;
      }
   }
}



void inorder(ttree *node)
{
   if(node != NULL)
   {
     inorder(node->lc);
     cout<<node->data<<"  ";
     inorder(node->rc);
   }
}

void myttree :: display_btree()
{
   if(head == NULL)
     cout<<"\nBinary tree is empty";
   else
   {
      cout<<"Inorder Traversal of the tree : ";
      inorder(head);
   }
}
int main()
{
  int ch;
  myttree t1;

  do
  {
    cout<<"\n\t1: Create Binary Tree";
    cout<<"\n\t2: Display Binary Tree";
    cout<<"\n\t3: Thread the Tree ";
    cout<<"\n\t4: Display the threaded tree";
    cout<<"\n\t5: Exit";
    cout<<"\n\nEnter ur choice : ";
    cin>>ch;
    switch(ch)
    {
     case 1 : t1.create_btree();
              break;
     case 2 : t1.display_btree();
              break;
     case 3 : t1.thread_btree();
              break;
     case 4 : t1.display_ttree();
              break;
     case 5 : cout<<"\nend\n";
              break;
      default: cout<<"\nTry again\n";
    }
  }while(ch != 5);
  return 0;
}
```