THE FOLLOWING CODE VERIFIES THE CAYLEY HAMILTON THEOREM FOR 3x3 COMPLEX MATRICES..

THIS CODE IS STRICTLY APPLICABLE FOR 3X3 MATRICES ONLY….

```
#include<iostream.h>
#include<conio.h>
class Complex;
template <class T>
class Matrix:public Complex
{
  private: T **M;
              int row;
              int column;
  public: Matrix(void);
              Matrix(const Matrix &);
              friend istream & operator>>(istream &,Matrix &);
              friend ostream & operator<<(ostream &,Matrix &);
              friend Matrix operator+(const Matrix &,const Matrix &);
              Matrix operator*(const Matrix &);
              void Scalar(T);
              void Coefficient(T *);
              void operator=(const Matrix &);
              ~Matrix(void);
};

template <class T>
Matrix<T>::Matrix(void)
```

DO NOT COPY

```
{
  M=NULL;
  row=column=0;
}

template <class T>
Matrix<T>::Matrix(const Matrix<T> & A)
{
  row=A.row;
  column=A.column;
  M=new T*;
  for(int i=0;i<row;i++)
     M[i]=new T[this->column];
  for(int j=0;j<row;j++)
  {
    for(int k=0;k<column;k++)
    {
      M[j][k]=A.M[j][k];
    }
  }
}

template <class T>
istream &operator>>(istream &in,Matrix<T> &A)
{
```

DO NOT COPY

```
 cout<<"\nEnter the order of Matrix:";
 in>>A.row>>A.column;
 A.M=new T*;
 for(int i=0;i<A.row;i++)
   A.M[i]=new T[A.column];
 for(i=0;i<A.row;i++)
 {
  for(int j=0;j<A.column;j++)
  {
    cout<<"\nEnter M["<<i<<"]["<<j<<"]:";
    in>>A.M[i][j];
  }
 }
 return in;
}

template <class T>
ostream & operator<<(ostream &out,Matrix<T> &A)
{
 out<<"\nThe details of the matrix are:\n";
 for(int i=0;i<A.row;i++)
 { out<<"*\t";
  for(int j=0;j<A.column;j++)
  {
     out<<A.M[i][j]<<'\t';
```

DO NOT COPY

```
    }
    out<<"*\n";
  }
  return out;
}

template<class T>
Matrix<T> operator+(const Matrix<T> &A,const Matrix<T> &B)
{
   Matrix<T> Ans;
   Ans.row=A.row;
   Ans.column=A.column;
   Ans.M=new T*;
   for(int i=0;i<A.row;i++)
      Ans.M[i]=new T[A.column];
   for(i=0;i<A.row;i++)
   {
     for(int j=0;j<A.column;j++)
     {
       Ans.M[i][j]=A.M[i][j]+B.M[i][j];
     }
   }
   return Ans;
}
template <class T>
```

DO NOT COPY

```
Matrix<T> Matrix<T>::operator*(const Matrix<T> &A)
{
 Matrix Ans;
 Ans.row=row;
 Ans.column=A.column;
 Ans.M=new T*;
 for(int i=0;i<row;i++)
   Ans.M[i]=new T[Ans.column];
 Complex C;
 for(i=0;i<Ans.row;i++)
 {
  for(int j=0;j<Ans.column;j++)
  {
    Ans.M[i][j]=C;
    for(int k=0;k<column;k++)
    {
          Ans.M[i][j]=Ans.M[i][j]+M[i][k]*A.M[k][j];
    }
  }
 }
 return Ans;
}
template <class T>
void Matrix<T>::operator=(const Matrix<T> &A)
{
```

DO NOT COPY

```
 this->row=A.row;
 this->column=A.column;
 M=new T*;
 for(int i=0;i<row;i++)
  M[i]=new T[column];
 for(i=0;i<row;i++)
 {
  for(int j=0;j<column;j++)
  {
    M[i][j]=A.M[i][j];
  }
 }
}

template <class T>
Matrix<T>::~Matrix(void)
{
 for(int i=0;i<row;i++)
   delete []M[i];
 delete M;
}
template <class T>
void Matrix<T>::Coefficient(T* A)
{
 Complex C;
```

DO NOT COPY

```
 cout<<"\nEnter C as -1,0:";
 cin>>C;
 A[0]=C;
 A[1]=M[0][0]+M[1][1]+M[2][2];

A[2]=M[2][1]*M[1][2]+M[0][1]*M[1][0]+M[0][2]*M[2][0]-M[1][1]*M[2][2]-M[0][0]*M[1][1]-M[0][0]*M[
2][2];

A[3]=M[0][0]*M[1][1]*M[2][2]-M[0][0]*M[2][1]*M[1][2]-M[0][1]*M[1][0]*M[2][2]+M[0][1]*M[2][0]*M[
1][2]+M[0][2]*M[1][0]*M[2][1]-M[0][2]*M[2][0]*M[1][1];
}
template <class T>
void Matrix<T>::Scalar(T a)
{
  for(int i=0;i<row;i++)
  {
    for(int j=0;j<column;j++)
    {
      M[i][j]=M[i][j]*a;
    }
  }
}

class Complex
{
  public: int real;
          int imag;
```

DO NOT COPY

```
  public: Complex(void);
          friend istream & operator>>(istream &,Complex &);
          friend ostream & operator<<(ostream &,Complex &);
          Complex operator+(const Complex &);
          Complex operator-(const Complex &);
          Complex operator*(const Complex &);
          void operator=(const Complex &);
          ~Complex(void);
};

Complex::Complex(void)
{
  real=0;
  imag=0;
}

istream & operator>>(istream &in,Complex &C)
{
  cout<<"\nEnter real and imaginary parts:";
  in>>C.real>>C.imag;
  return in;
}

ostream & operator<<(ostream & out,Complex &C)
{
```

DO NOT COPY

```
 out<<C.real<<"+"<<C.imag<<"i;";
 return out;
}

Complex Complex::operator+(const Complex &C)
{
 Complex Ans;
 Ans.real=real+C.real;
 Ans.imag=imag+C.imag;
 return Ans;
}

Complex Complex::operator-(const Complex &C)
{
 Complex Ans;
 Ans.real=real-C.real;
 Ans.imag=imag-C.imag;
 return Ans;
}

Complex Complex::operator*(const Complex &C)
{
 Complex Ans;
 Ans.real=real*C.real-imag*C.imag;
 Ans.imag=imag*C.real+real*C.imag;
```

DO NOT COPY

```
 return Ans;
}

void Complex::operator=(const Complex &C)
{
 real=C.real;
 imag=C.imag;
}

Complex::~Complex(void)
{
 //I'm in Destructor!
}

void main(void)
{
 clrscr();
 Matrix <Complex>A;
 Matrix <Complex>U1,U2,U3,U4;
 Complex C[4];
 cout<<"\nEnter matrix-A:";
 cin>>A;
 cout<<"\nEnter unit matrices:";
 cin>>U1;
 U2=U1;
```

DO NOT COPY

```
 U3=U1;
 U4=U1;
 cout<<"\nMatrix -A is:";
 cout<<A;
 cout<<"\nThe unit matrices are:";
 cout<<U1;
 cout<<U2;
 cout<<U3;
 cout<<U4;
 A.Coefficient(C);
 U1.Scalar(C[0]);
 U2.Scalar(C[1]);
 U3.Scalar(C[2]);
 U4.Scalar(C[3]);
/*  cout<<U1<<endl;
 cout<<"U2="<<U2<<endl;
 cout<<"U3="<<U3<<endl;
 cout<<"U4="<<U4<<endl;
 getch();    */
 Matrix<Complex> B;
 B=U1*A*A*A+U2*A*A+U3*A+U4;
 cout<<"\nThe matrix B is:";
 cout<<B;
 cout<<"\nHENCE THE CAYLEY HAMILTON THEOREM OF MATRICES IS VERFIED!";
 getch();
```

DO NOT COPY

```
}
```

DO NOT COPY