Data Structure Lab Manual

Winter 2019

Session 3

This should be the general structure of your code on which we have been working on since last two weeks.

```
#include<iostream>

using namespace std;

//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
// Definition of complex
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------

class complex
{
        double real;
        double imag;
        static int numberOfObjects;

public:
        //default constructor
        complex() { real = 0; imag = 0; numberOfObjects++; };
        //copy constructor
        complex(complex &c) { real = c.real; imag = c.imag; numberOfObjects++;};
        //parameterized constructor
        //complex(double r, double i) { real = r; imag = i; numberOfObjects++;}
        complex(double r, double i):real(r), imag(i) {  numberOfObjects++; };
        static int getNumberOfObjects() { return numberOfObjects; };
        void setReal(double);
        void setImag(double);
        double getReal();
        double getImag();

        void add(const complex &, const complex &);
        complex & maxComplex(complex &a, complex &b);
};

int complex::numberOfObjects = 0;
```

```
void complex::setReal(double a) { real = a; }
void complex::setImag(double a) { imag = a; }
double complex::getReal() { return real; }
double complex::getImag() { return imag; }
void complex::add(const complex &a , const complex &b)
{
        //cout << "The address of a is:" << &a;
        real = a.real + b.real;
        imag = a.imag + b.imag;
}
complex & complex::maxComplex(complex &a, complex &b)
{
        double magnitude1, magnitude2;

        //compute the square of the magnitude of two complex numbers
        magnitude1 = a.real * a.real + a.imag*a.imag;
        magnitude2 = b.real * b.real + b.imag*b.imag;

        if (magnitude1 > magnitude2)
                return a;
        else
                return b;
}

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
// Main Program
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
int main()
{

        double  r, im;
        complex c1, c2, c3;

        cout << "Enter the first complex number:";
        cin >> r >> im;
        c1.setReal(r);
        c1.setImag(im);

        cout << "Enter the second complex number:";
        cin >> r >> im;
        c2.setReal(r);
        c2.setImag(im);

        //cout << "The address of c1 is:" << &c1;
        c3.add(c1, c2);
        cout << "Sum: " << c3.getReal() << "+i"<<c3.getImag()<<endl;
```

```
        complex &c4 = c1.maxComplex(c1, c2);
        cout << "Maximum: " << c4.getReal() << "+i" << c4.getImag()<<endl;
        cout << "The address of c1 is:" << &c1<<endl;
        cout << "The address of c2 is:" << &c2 << endl;
        cout << "The address of c4 is:" << &c4<<endl;

        complex * p;
        p = &c3;
        p->add(c1, c2);
        cout << "Sum: " << p->getReal() << "+i" << p->getImag();

        int size;
        double a, b;
        cout << "Enter the size of the array:";
        cin >> size;
        p = new complex[size];
        //this loops takes the input
        c3.setImag(0);
        c3.setReal(0);
        for (int i = 0; i < size; i++)
        {
                cout << "Enter the " << i + 1 << " complex number";
                cin >> a >> b;
                p[i].setReal(a);
                p[i].setImag(b);
        }

        cout << "The number of objects created from the complex class
        are" << complex::getNumberOfObjects();

        cin >> r;
        return 0;
}
```

First Step

The aim of this step is to learn operator overloading. I want to overload the "+" operator to sum two complex numbers. First define the function as a member of class complex as

```
complex operator+(complex & c);
```

Define the following function outside the class as

```
complex complex::operator+(complex & c)
{
        complex s;
        s.real = real + c.real;
        s.imag = imag + c.imag;
```

```
        return s;
}
```

In the main function the sum can be done for two complex number as

```
c3 = c1 + c2;
cout << "Sum: " << c3.getReal() << "+i" << c3.getImag() << endl;
```

This is nothing but we have overloaded "+" operator to add two objects the logic of this addition is defined in the function "operator+". Following lines are equivalent

```
c3 = c1 + c2;
c3 = c1.operator+(c2); \\both these lines are equivalent
```

Second Step

This step deals with converting a predefined datatype to the user defined datatype. In this type conversion, we want to convert a real number to a complex number. In order to do this define a constructor inside the class complex as

```
complex(double r) :real(r), imag(0.0) { };
```

The above line converts the real number to complex object. This can be tested in main program with these lines

```
        double  r, im;
        complex c1, c2, c3;

        cout << "Enter the first complex number:";
        cin >> r >> im;
        c1.setReal(r);
        c1.setImag(im);

        c1 = r;
        cout << "c1: " << c1.getReal() << "+i" << c1.getImag() << endl;
```

This way any real number can be converted to complex objects just with the help on "=" inside the main.

Third Step

In this step, we convert a complex object into a real number. This automatically should set the magnitude of the complex number in the real variable. In order to acheive this define following function as the member function of the class complex

```
complex::operator double()
{
        double s;
        s = sqrt(real*real + imag*imag);
        return s;
}
```

Write following lines in the main to test this functionality

```
r = c1;
cout << "Magnitude of c1: " << r << endl;
```

Fourth Step

Similarly, we can define type conversions from one class to the other user defined class. We want to convert from a class complex to class polar complex. So define following polar class with only getter functions and default constructor.

```
class polarcomplex
{
        double r;
        double theta;
public:
        polarcomplex() { r = 0; theta = 0; };
        double getR() { return r; };
        double getTheta() { return theta; };
};
```

The type conversion involved here needs to identify complex class as source and polarcomplex as destination class. Now the logic of this conversion can either be in the source class or in the destination class. For simplification let us keep the logic of this conversion in the polarcomplex. So, we need to define a constructor which will do this job in polar complex as

```
polarcomplex(complex &c)
        {
                r = sqrt(c.getReal()*c.getReal() + c.getImag()*c.getImag());
                theta = atan(c.getImag() / c.getReal());
        }
```

In the main program, it is tested using

```
polarcomplex pc;
pc = c1;
cout << "Polar representation: r:" << pc.getR() << " theta:" <<
pc.getTheta() << endl;
```

Fifth Step

In the fifth step, we want to convert the polarcomplex object back to the original one. The logic for this conversion should be defined in the polarcomplex. The source class is polarcomplex and the destination is complex. Therefore we need an overloaded complex operator in the polarcomplex class as

```
        operator complex()
        {
                complex c;
                c.setReal(r*cos(theta));
                c.setImag(r*sin(theta));
                return c;
        }
```

In the main program, we need to test it using these line

```
c1.setImag(0.0);
c1.setReal(0.0);
```

```
c1 = pc;
cout << "c1: " << c1.getReal() << "+i" << c1.getImag() <<endl;
```

Sixth Step

Now we change our code to include templates. The templates act as a place holder for the datatype. In our case the real and imag of the complex object can be replaced with a place holder so modify the complex class as following

```
template<typename data>
class complex
{
        data real;
        data imag;
        static int numberOfObjects;

public:
        //default constructor
        complex() { real = 0; imag = 0; numberOfObjects++; };
        //copy constructor
        complex(const complex &c) { real = c.real; imag = c.imag; numberOfObjects++;
        };
        //parameterized constructor
        //complex(double r, double i) { real = r; imag = i; numberOfObjects++;}
        complex(data r, data i):real(r), imag(i) {  numberOfObjects++; };
        complex(data r) :real(r), imag(0.0) { };
        static int getNumberOfObjects() { return numberOfObjects; };
        void setReal(data);
        void setImag(data);
        double getReal();
        double getImag();

        void add(const complex &, const complex &);
        complex & maxComplex(complex &a, complex &b);

        complex operator+(complex & c);
        operator data();
};
```

The member function will also change so that the placeholder for datatype can be used. One such example is here

```
template<typename data>
complex<data> & complex<data>::maxComplex(complex &a, complex &b)
{
        data magnitude1, magnitude2;

        //compute the square of the magnitude of two complex numbers
        magnitude1 = a.real * a.real + a.imag*a.imag;
        magnitude2 = b.real * b.real + b.imag*b.imag;
```

```
        if (magnitude1 > magnitude2)
                return a;
        else
                return b;
}
```

In the main program the changes would be as shown below:

```
complex<double> c1, c2, c3;

        cout << "Enter the first complex number:";
        cin >> r >> im;
        c1.setReal(r);
        c1.setImag(im);

        polarcomplex<double> pc;
```

The final code looks something like this

```
#include<iostream>

using namespace std;

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Definition of complex
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

template<typename data>
class complex
{
        data real;
        data imag;
        static int numberOfObjects;

public:
        //default constructor
        complex() { real = 0; imag = 0; numberOfObjects++; };
        //copy constructor
        complex(const complex &c) { real = c.real; imag = c.imag; numberOfObjects++;
        };
        //parameterized constructor
        //complex(double r, double i) { real = r; imag = i; numberOfObjects++;}
        complex(data r, data i):real(r), imag(i) {  numberOfObjects++; };
        complex(data r) :real(r), imag(0.0) { };
        static int getNumberOfObjects() { return numberOfObjects; };
        void setReal(data);
        void setImag(data);
```

```
        data getReal();
        data getImag();

        void add(const complex &, const complex &);
        complex & maxComplex(complex &a, complex &b);

        complex operator+(complex & c);
        operator data();
};

template<typename data>
int complex<data>::numberOfObjects = 0;

template<typename data>
void complex<data>::setReal(data a) { real = a; }

template<typename data>
void complex<data>::setImag(data a) { imag = a; }

template<typename data>
data complex<data>::getReal() { return real; }

template<typename data>
data complex<data>::getImag() { return imag; }

template<typename data>
void complex<data>::add(const complex &a , const complex &b)
{
        //cout << "The address of a is:" << &a;
        real = a.real + b.real;
        imag = a.imag + b.imag;
}

template<typename data>
complex<data> & complex<data>::maxComplex(complex &a, complex &b)
{
        data magnitude1, magnitude2;

        //compute the square of the magnitude of two complex numbers
        magnitude1 = a.real * a.real + a.imag*a.imag;
        magnitude2 = b.real * b.real + b.imag*b.imag;

        if (magnitude1 > magnitude2)
                return a;
        else
                return b;
}
```

```
template<typename data>
complex<data> complex<data>::operator+(complex & c)
{
        complex s;
        s.real = real + c.real;
        s.imag = imag + c.imag;
        return s;
}

template<typename data>
complex<data>::operator data()
{
        data s;
        s = sqrt(real*real + imag*imag);
        return s;
}


template<typename data>
class polarcomplex
{
        double r;
        double theta;
public:
        polarcomplex() { r = 0; theta = 0; };
        double getR() { return r; };
        double getTheta() { return theta; };
        polarcomplex(complex<data> &c)
        {
                r = sqrt(c.getReal()*c.getReal() + c.getImag()*c.getImag());
                theta = atan(c.getImag() / c.getReal());
        }
        operator complex<data>()
        {
                complex<data> c;
                c.setReal(r*cos(theta));
                c.setImag(r*sin(theta));
                return c;
        }
};

//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
// Main Program
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
int main()
{
```

```
double  r, im;
complex<double> c1, c2, c3;

cout << "Enter the first complex number:";
cin >> r >> im;
c1.setReal(r);
c1.setImag(im);

polarcomplex<double> pc;
pc = c1;
cout << "Polar representation: r:" << pc.getR() << " theta:" << pc.getTheta() << end

c1.setImag(0.0);
c1.setReal(0.0);
c1 = pc;
cout << "c1: " << c1.getReal() << "+i" << c1.getImag() <<endl;

r = c1;
cout << "Magnitude of c1: " << r << endl;

c1 = r;
cout << "c1: " << c1.getReal() << "+i" << c1.getImag() << endl;

cout << "Enter the second complex number:";
cin >> r >> im;
c2.setReal(r);
c2.setImag(im);

c3 = c1 + c2;
cout << "Sum: " << c3.getReal() << "+i" << c3.getImag() << endl;

cin >> r;
return 0;

//cout << "The address of c1 is:" << &c1;
c3.add(c1, c2);
cout << "Sum: " << c3.getReal() << "+i"<<c3.getImag()<<endl;

complex<double> &c4 = c1.maxComplex(c1, c2);
cout << "Maximum: " << c4.getReal() << "+i" << c4.getImag()<<endl;
cout << "The address of c1 is:" << &c1<<endl;
cout << "The address of c2 is:" << &c2 << endl;
cout << "The address of c4 is:" << &c4<<endl;

complex<double> * p;
p = &c3;
p->add(c1, c2);
```

```
        cout << "Sum: " << p->getReal() << "+i" << p->getImag();

        int size;
        double a, b;
        cout << "Enter the size of the array:";
        cin >> size;
        p = new complex<double>[size];
        //this loops takes the input
        c3.setImag(0);
        c3.setReal(0);
        for (int i = 0; i < size; i++)
        {
                cout << "Enter the " << i + 1 << " complex number";
                cin >> a >> b;
                p[i].setReal(a);
                p[i].setImag(b);
        }

        cout << "The number of objects created from the complex class are " << complex<doubl

        cin >> r;
        return 0;
}
```