Ceng 205 Computer Programming II
Final
Aug 3, 2004 09.00–11.00
Good Luck!

1 (30 Pts) Create a class **Rectangle** with attributes **length** and **width**, each of which defaults to 1. Provide member functions that calculate the **perimeter** and the **area** of the rectangle. Also, provide *set* and *get* functions for the **length** and **width** attributes. The set functions should verify that **length** and **width** are each floating-point numbers larger than 0.0 and less than 20.0. Write a complete program for the class **Rectangle** with the above capabilities.

```
#ifndef HEADER_H
#define HEADER_H
class Rectangle {
public:
   Rectangle( double = 1.0, double = 1.0 );  // default constructor
   double perimeter();  // perimeter
   double area();  // area
   void setWidth( double w ); // set width
   void setLength( double l ); // set length
   double getWidth();           // get width
   double getLength();          // get length
private:
   double length;           // 1.0 < length < 20.0
   double width;            // 1.0 < width < 20.0
}; // end class Rectangle
#endif

// member function definitions
#include "header.h"
Rectangle::Rectangle( double w, double l )
{
   setWidth(w);  // invokes function setWidth
   setLength(l); // invokes function setLength
}

double Rectangle::perimeter()
{
return 2 * ( width + length ); // returns perimeter
}

double Rectangle::area()
{
return width * length;          // returns area
}

void Rectangle::setWidth( double w )
```

```
{
width = w > 0 && w < 20.0 ? w : 1.0; // sets width
}

void Rectangle::setLength( double l )
{
length = l > 0 && l < 20.0 ? l : 1.0; // sets length
}

double Rectangle::getWidth()
{
return width;
}

double Rectangle::getLength()
{
return length;
}

// main
#include <iostream>
using std::cout; using std::endl; using std::fixed;
#include <iomanip>
using std::setprecision;
#include "header.h"

int main()
{
   Rectangle a, b( 4.0, 5.0 ), c( 67.0, 888.0 );
   cout << fixed;
   cout << setprecision( 1 );
   cout << "a: length = " << a.getLength()
    << "; width = " << a.getWidth()
        << "; perimeter = " << a.perimeter() << "; area = "
        << a.area() << '\n';
   cout << "b: length = " << b.getLength()
    << "; width = " << b.getWidth()
        << "; perimeter = " << b.perimeter() << "; area = "
        << b.area() << '\n';
   cout << "c: length = " << c.getLength()
    << "; width = " << c.getWidth()
        << "; perimeter = " << c.perimeter() << "; area = "
        << c.area() << endl;
   return 0;
}
```

2 (30 Pts) Create a class called **RationalNumber** (fractions) with the following capabilities:

- Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.
- Overload the addition, substraction, multiplication and division operators for this class.
- Overload the relational and equality operators for this class.

Write a complete program for the class **RationalNumber** with the above capabilities.

```
-----------------------------------------------
// rational.h
#ifndef RATIONAL_H
#define RATIONAL_H
class RationalNumber {
public:
   RationalNumber( int = 0, int = 1 ); // default constructor
   const RationalNumber operator+( const RationalNumber& );
   const RationalNumber operator-( const RationalNumber& );
   const RationalNumber operator*( const RationalNumber& );
   const RationalNumber operator/( RationalNumber& );
   bool operator>( const RationalNumber& ) const;
   bool operator<( const RationalNumber& ) const;
   bool operator>=( const RationalNumber& ) const;
   bool operator<=( const RationalNumber& ) const;
   bool operator==( const RationalNumber& ) const;
   bool operator!=( const RationalNumber& ) const;
   void printRational( void ) const;
private:
   int numerator;
   int denominator;
   void reduction( void );
};
#endif
-----------------------------------------------
// rational.cpp
#include <cstdlib>
#include <iostream>
using std::cout; using std::endl;
#include "rational.h"
RationalNumber::RationalNumber( int n, int d )
{
   numerator = n;
   denominator = d;
   reduction();
}
const RationalNumber RationalNumber::operator+(const RationalNumber &a )
{
```

```
   RationalNumber add;
   add.numerator = numerator*a.denominator+denominator*a.numerator;
   add.denominator = denominator*a.denominator;
   add.reduction();
   return add;
}
const RationalNumber RationalNumber::operator-(const RationalNumber &s )
{
   RationalNumber sub;
   sub.numerator = numerator*s.denominator-denominator*s.numerator;
   sub.denominator = denominator*s.denominator;
   sub.reduction();
   return sub;
}
const RationalNumber RationalNumber::operator*(const RationalNumber &m )
{
   RationalNumber mul;
   mul.numerator = numerator*m.numerator;
   mul.denominator = denominator*m.denominator;
   mul.reduction();
   return mul;
}
const RationalNumber RationalNumber::operator/( RationalNumber &d )
{
   RationalNumber divide;
   if ( /* write condition to test for zero numerator */ )
   if ( numerator!=0 )
‘  {
      divide.numerator = numerator*d.denominator;
      divide.denominator = denominator*d.numerator;
      divide.reduction();
   }
   else
   {
      cout << "Divide by zero error: terminating program\n";
      exit( 1 );   // cstdlib function
   }
   return divide;
}
bool RationalNumber::operator>(const RationalNumber &gr ) const
{
   if ( static_cast< double >( numerator ) / denominator >
        static_cast< double >( gr.numerator ) / gr.denominator )
      return true;
   else
      return false;
}
bool RationalNumber::operator<(const RationalNumber &lr ) const
```

```
{
   return !(*this > lr);
}
bool RationalNumber::operator>=(
   const RationalNumber &rat ) const
   { return *this == rat || *this > rat; }
bool RationalNumber::operator<=(const RationalNumber &lat ) const
{ return !(*this == lat || *this > lat); }
bool RationalNumber::operator==(const RationalNumber &eq ) const
   {
   if ( static_cast< double >( numerator ) == eq.numerator &&
        static_cast< double >( denominator ) == eq.denominator )
        return true;
   else
        return false;
   }
bool RationalNumber::operator!=(const RationalNumber &neq ) const
   { return !(*this == neq); }
void RationalNumber::printRational( void ) const
{
   if ( numerator == 0 )         // print fraction as zero
      cout << numerator;
   else if ( denominator == 1 )  // print fraction as integer
      cout << numerator;
   else
      cout << numerator << '/' << denominator;
}
void RationalNumber::reduction( void )
{
   int smallest, gcd = 1;  // greatest common divisor;
   smallest = ( numerator < denominator ) ? numerator : denominator;
   for ( int loop = 2; loop <= smallest; ++loop )
       if ( numerator % loop == 0 && denominator % loop == 0 )
          gcd = loop;
   numerator /= gcd;
   denominator /= gcd;
}
-----------------------------------------------
// driver.cpp
#include <iostream>
using std::cout; using std::endl;
#include "rational.h"
int main()
{
   RationalNumber c( 7, 3 ), d( 3, 9 ), x;
   c.printRational();
   cout << " + " ;
   d.printRational();
```

```
cout << " = ";
x = c + d;
x.printRational();
cout << '\n';
c.printRational();
cout << " - " ;
d.printRational();
cout << " = ";
x = c - d; /* subtract c from d and assign the result to x */
x.printRational();
cout << '\n';
c.printRational();
cout << " * " ;
d.printRational();
cout << " = ";
x = c * d; /* multiply c and d and assign the result to x */
x.printRational();
cout << '\n';
c.printRational();
cout << " / " ;
d.printRational();
cout << " = ";
x = c / d;
x.printRational();
cout << '\n';
c.printRational();
cout << " is:\n";
cout << ( ( c > d ) ? "  > " : "  <= " );
d.printRational();
cout << " according to the overloaded > operator\n";
cout << ( ( c < d ) ? "  < " : "  >= " );
d.printRational();
cout << " according to the overloaded < operator\n";
cout << ( ( c >= d ) ? "  >= " : "  < " );
d.printRational();
cout << " according to the overloaded >= operator\n";
cout << ( ( c <= d ) ? "  <= " : "  > " );
d.printRational();
cout << " according to the overloaded <= operator\n";
cout << ( ( c == d ) ? "  == " : "  != " );
d.printRational();
cout << " according to the overloaded == operator\n";
cout << ( ( c != d ) ? "  != " : "  == " );
d.printRational(); /* write statement to print d */
cout << " according to the overloaded != operator" << endl;
return 0;}
```

3 (30 Pts) Write an inheritance hierarchy for class **Quadrilateral**, **Trapezoid**, **Parallelogram**, **Rectangle** and **Square**. Use **Quadrilateral** as the base class of the hierarchy. Make the hierarchy as deep (i.e., as many levels) as possible. The **private** data of **Quadrilateral** should be the x-y coordinate pairs for the four endpoints of the **Quadrilateral**. Write a complete program that instantiates objects of each of the classes in your hierarchy and outputs each object's dimensions and area.

Hints: A quadrilateral is a four-sided polygon; A trapazoid is a quadrilateral having two and only two parallel sides; A rhombus is an equilateral parallelogram; A rectangle is an equiangular parallelogram; A square is an equiangular and equilateral parallelogram

```
Quadrilateral q( 1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4 );
Trapazoid t( 5.0, 0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0 );
Parallelogram p( 5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0 );
Rhombus rh( 0.0, 0.0, 5.0, 0.0, 8.5, 3.5, 3.5, 3.5 );
Rectangle r( 17.0, 14.0, 30.0, 14.0, 0.0, 28.0, 17.0, 28.0 );
Square s( 4.0, 0.0, 8.0, 0.0, 8.0, 4.0,  4.0, 4.0 );
```

```
// Header file for class Point
#ifndef pointh_H
#define pointh_H
#include <iostream>
using std::ostream;
class Point
{
   friend ostream &operator<<( ostream&, const Point& );
public:
   Point( double = 0, double = 0 );
   void setPoint( double, double );
   void print() const;
   double getX() const { return x; } // return x
   double getY() const { return y; } // return y
private:
   double x; // x coordinate
   double y; // y coordinate
};
#endif
```

```
// Header class for Quadrilateral
#ifndef quadrilateralh_H
#define quadrilateralh_H
#include "pointh.h"
#include <iostream>
using std::ostream;
class Quadrilateral
{
   friend ostream &operator<<( ostream&, Quadrilateral& );
public:
   Quadrilateral( double = 0, double = 0, double = 0, double = 0, double = 0,double = 0, double =
```

```
   void print() const;
protected:
   Point p1;
   Point p2;
   Point p3;
   Point p4;
};
#endif

// Header file for class Trapazoid
#ifndef trapazoidh_H
#define trapazoidh_H
#include "quadrilateralh.h"
#include <iostream>
using std::ostream;
class Trapazoid : public Quadrilateral
{
   friend ostream& operator<<( ostream&, Trapazoid& );
public:
   Trapazoid( double = 0, double = 0, double = 0, double = 0, double = 0,
              double = 0, double = 0, double = 0, double = 0 );
   void print() const;
   void setHeight( double h ) { height = h; }
   double getHeight() const { return height; }
private:
   double height;
};
#endif

// Header file for class Parallelogram
#ifndef parallelogramh_H
#define parallelogramh_H
#include "quadrilateralh.h"
#include <iostream>
using std::ostream;
class Parallelogram : public Quadrilateral
{
   friend ostream& operator<<( ostream&, Parallelogram& );
public:
   Parallelogram( double = 0, double = 0, double = 0, double = 0,
                  double = 0, double = 0, double = 0, double = 0 );
   void print() const;
private:
};
#endif

// Header file for class Rhombus
#ifndef rhombush_H
```

```
#define rhombush_H
#include "parallelogramh.h"
#include <iostream>
using std::ostream;
class Rhombus : public Parallelogram
{
   friend ostream& operator<<(ostream&, Rhombus&);
public:
   Rhombus( double = 0, double = 0, double = 0, double = 0, double = 0,
            double = 0, double = 0, double = 0 );
   void print() const { Parallelogram::print(); }
private:
};
#endif

// Header file for class Rectangle
#ifndef rectangleh_H
#define rectangleh_H
#include "parallelogramh.h"
#include <iostream>
using std::ostream;
class Rectangle : public Parallelogram
{
   friend ostream& operator<<( ostream&, Rectangle& );
public:
   Rectangle( double = 0, double = 0, double = 0, double = 0,
              double = 0, double = 0, double = 0, double = 0 );
   void print() const;
private:
};
#endif

// Header file for class Square
#ifndef squareh_H
#define squareh_H
#include "parallelogramh.h"
#include <iostream>
using std::ostream;
class Square : public Parallelogram
{
   friend ostream& operator<<( ostream&, Square& );
public:
   Square( double = 0, double = 0, double = 0, double = 0,
          double = 0, double = 0, double = 0, double = 0 );
   void print() const { Parallelogram::print(); }
private:
};
#endif
```

```
// member function defintions for class Point
#include <iostream>
using std::cout; using std::ios;
using std::ostream; using std::fixed;
#include <iomanip>
using std::setprecision;
#include "pointh.h"
Point::Point( double a, double b )
   { setPoint( a, b ); } // end Point constructor
void Point::setPoint( double a, double b )
{   x = a;
   y = b; }
ostream &operator<<( ostream &output, const Point &p )
{   output << "The point is: ";
   p.print();
   return output; }
void Point::print() const
{   cout << fixed
        << '[' << setprecision( 2 ) << getX()
        << ", " << setprecision( 2 ) << getY() << "]\n"
        << fixed;}

// member functions for class Quadrilateral
#include "quadrilateralh.h"
#include <iostream>
using std::cout; using std::ostream;
Quadrilateral::Quadrilateral( double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4 )   : p1( x1, y1 ), p2( x2, y2 ), p3( x3, y3 ), p4( x4,
ostream &operator<<( ostream& output, Quadrilateral& q )
{   output << "Coordinates of Quadrilateral are:\n";
   q.print();
   output << '\n';
   return output;}

// member function definitions for class Trapazoid
#include "trapazoidh.h"
#include <iostream>
using std::cout;
using std::ostream;
Trapazoid::Trapazoid( double h, double x1, double y1, double x2, double y2,double x3, double y3, d
          : Quadrilateral( x1, y1, x2, y2, x3, y3, x4, y4 )
   { setHeight( h ); }
ostream& operator<<( ostream& out, Trapazoid& t )
{   out << "The Coordinates of the Trapazoid are:\n";
   t.print();
   return out;}
```

```
// member function defintions for class Parallelogram
#include "quadrilateralh.h"
#include "parallelogramh.h"
#include <iostream>
using std::ostream;
Parallelogram::Parallelogram( double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4 ): Quadrilateral( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
ostream& operator<<( ostream& out, Parallelogram& pa )
{   out << "The coordinates of the Parallelogram are:\n";
   pa.print();
   return out;}
void Parallelogram::print() const
   {   Quadrilateral::print();   }
void Quadrilateral::print() const
{   cout << '(' << p1.getX()
        << ", " << p1.getY() << ") , (" << p2.getX() << ", " << p2.getY()
        << ") , (" << p3.getX() << ", " << p3.getY() << ") , ("
        << p4.getX() << ", " << p4.getY() << ")\n";}

// member function defintions for class Rhombus
#include "rhombush.h"
#include "parallelogramh.h"
#include <iostream>
using std::ostream;
Rhombus::Rhombus( double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4 ) : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ){ }
ostream& operator<<( ostream& out, Rhombus& r )
{   out << "\nThe coordinates of the Rhombus are:\n";
   r.print();
   return out;}

// member function defintions for class Rectangle
#include "rectangleh.h"
#include "parallelogramh.h"
#include <iostream>
using std::ostream;

Rectangle::Rectangle( double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4 ) : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
ostream& operator<<( ostream& out, Rectangle& r )
{   out << "\nThe coordinates of the Rectangle are:\n";
   r.print();
   return out;}

// member function defintions for class Square
#include "squareh.h"
#include "parallelogramh.h"
#include <iostream>
```

```
using std::ostream;
// default constructor calls base class constructor to set members
Square::Square( double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4 ) : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }
ostream& operator<<( ostream& out, Square& s )
{   out << "\nThe coordinates of the Square are:\n";
   s.print();
   return out;}

// driver
#include "pointh.h"
#include "quadrilateralh.h"
#include "trapazoidh.h"
#include "parallelogramh.h"
#include "rhombush.h"
#include "rectangleh.h"
#include "squareh.h"
#include <iostream>
using std::cout;
using std::endl;
int main()
{
// A quadrilateral is a four-sided polygon
Quadrilateral q( 1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4 );
// A trapazoid is a quadrilateral having  two and only two parallel sides
Trapazoid t( 5.0, 0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0 );
// A parallelogram is a quadrilateral whose opposite sides are parallel
Parallelogram p( 5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0 );
// A rhombus is an equilateral parallelogram
Rhombus rh( 0.0, 0.0, 5.0, 0.0, 8.5, 3.5, 3.5, 3.5 );
// A rectangle is an equiangular parallelogram
Rectangle r( 17.0, 14.0, 30.0, 14.0, 0.0, 28.0, 17.0, 28.0 );
// A square is an equiangular and equilateral parallelogram
Square s( 4.0, 0.0, 8.0, 0.0, 8.0, 4.0,  4.0, 4.0 );
cout << q << t << p << rh << r << s << endl;
return 0;
}
```

4 (30 Pts) Write down all the shapes you can think of both two-dimensional and three-dimensional and form those shapes into a shape hierarchy. Your hierarchy should have an abstract base class **Shape** from which class **TwoDimensionalShape** and class **ThreeDimensionalShape** are derived (these classes should also be abstract). Once you have developed the hierarchy, define each of the classes in the hierarchy. Use a **virtual print** function to output polymorphically the type and dimensions of each class. Also include **virtual area** and **volume** functions so these calculations can be performed for objects of each concrete class in the hierarchy. Write a driver program that tests the **Shape** class hierarchy.
Hints:

```
   shapes[ 0 ] = new Circle( 3.5, 6, 9 );
   shapes[ 1 ] = new Square( 12, 2, 2 );
   shapes[ 2 ] = new Sphere( 5, 1.5, 4.5 );
   shapes[ 3 ] = new Cube( 2.2 );
```

```
// Definition of base-class Shape
#ifndef SHAPE_H
#define SHAPE_H
#include <iostream>
using std::ostream;
class Shape {
   friend ostream & operator<<( ostream &, Shape & );
public:
   Shape( double = 0, double = 0 );
   double getCenterX() const;
   double getCenterY() const;
   virtual void print() const = 0;
protected:
   double xCenter;
   double yCenter;
};
#endif

// Member and friend definitions for Shape
#include "shape.h"
Shape::Shape( double x, double y )
{   xCenter = x;
   yCenter = y;}
double Shape::getCenterX() const { return xCenter; }
double Shape::getCenterY() const { return yCenter; }
ostream & operator<<( ostream &out, Shape &s )
{  s.print();
   return out;}

// Definition of class TwoDimensionalShape
#ifndef TWODIM_H
#define TWODIM_H
#include "shape.h"
```

```
class TwoDimensionalShape : public Shape {
public:
   TwoDimensionalShape( double x, double y ) : Shape( x, y ) { }
   virtual double area() const = 0;
};
#endif

// Defnition of class ThreeDimensionalShape
#ifndef THREEDIM_H
#define THREEDIM_H
#include "shape.h"
class ThreeDimensionalShape : public Shape {
public:
   ThreeDimensionalShape( double x, double y ) : Shape( x, y ) { }
   virtual double area() const = 0;
   virtual double volume() const = 0;
};
#endif

// Definition of class Circle
#ifndef CIRCLE_H
#define CIRCLE_H
#include "twodim.h"
class Circle : public TwoDimensionalShape {
public:
   Circle( double = 0, double = 0, double = 0 );
   double getRadius() const;
   double area() const;
   void print() const;
private:
   double radius;
};
#endif

// Member function definitions for Circle
#include "circle.h"
#include <iostream>
using std::cout;
Circle::Circle( double r, double x, double y )
   : TwoDimensionalShape( x, y ) { radius = r > 0 ? r : 0; }
double Circle::getRadius() const { return radius; }
double Circle::area() const { return 3.14159 * radius * radius; }
void Circle::print() const
{
cout << "Circle with radius " << radius << "; center at ("
    << xCenter << ", " << yCenter << ");\narea of " << area() <<'\n';}
```

```
// Definition of class Square
#ifndef SQUARE_H
#define SQUARE_H
#include "twodim.h"
class Square : public TwoDimensionalShape {
public:
   Square( double = 0, double = 0, double = 0 );
   double getSideLength() const;
   double area() const;
   void print() const;
private:
   double sideLength;
};
#endif

// Member function definitions for Square
#include "square.h"
#include <iostream>
using std::cout;
Square::Square( double s, double x, double y )
   : TwoDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }
double Square::getSideLength() const { return sideLength; }
double Square::area() const { return sideLength * sideLength; }
void Square::print() const
{
cout << "Square with side length " << sideLength << "; center at ("
     << xCenter << ", " << yCenter << ");\narea of " << area() << '\n';}

// Definition of class Shere
#ifndef SPHERE_H
#define SPHERE_H
#include "threedim.h"
class Sphere : public ThreeDimensionalShape {
public:
   Sphere( double = 0, double = 0, double = 0 );
   double area() const;
   double volume() const;
   double getRadius() const;
   void print() const;
private:
   double radius;
};
#endif

// Member function definitions for Sphere
#include "sphere.h"
#include <iostream>
using std::cout;
```

```
Sphere::Sphere( double r, double x, double y )
   : ThreeDimensionalShape( x, y ) { radius = r > 0 ? r : 0; }
double Sphere::area() const
   { return 4.0 * 3.14159 * radius * radius; }
double Sphere::volume() const
   { return 4.0/3.0 * 3.14159 * radius * radius * radius; }
double Sphere::getRadius() const { return radius; }
void Sphere::print() const
{
cout << "Sphere with radius " << radius << "; center at ("
     << xCenter << ", " << yCenter << ");\narea of "
     << area() << "; volume of " << volume() << '\n';}

// Definition of class Cube
#ifndef CUBE_H
#define CUBE_H
#include "threedim.h"
class Cube : public ThreeDimensionalShape {
public:
   Cube( double = 0, double = 0, double = 0 );
   double area() const;
   double volume() const;
   double getSideLength() const;
   void print() const;
private:
   double sideLength;
};
#endif

// Member function definitions for Cube
#include "cube.h"
#include <iostream>
using std::cout;
Cube::Cube( double s, double x, double y )
   : ThreeDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }
double Cube::area() const { return 6 * sideLength * sideLength; }
double Cube::volume() const
   { return sideLength * sideLength * sideLength; }
double Cube::getSideLength() const { return sideLength; }
void Cube::print() const
{
cout << "Cube with side length " << sideLength << "; center at ("
     << xCenter << ", " << yCenter << ");\narea of "
     << area() << "; volume of " << volume() << '\n';}

// Driver to test Shape hierarchy
#include <iostream>
using std::cout;
```

```
#include <vector>
using std::vector;
#include "shape.h"
#include "circle.h"
#include "square.h"
#include "sphere.h"
#include "cube.h"
int main()
{
   vector < Shape * > shapes( 4 );
   shapes[ 0 ] = new Circle( 3.5, 6, 9 );
   shapes[ 1 ] = new Square( 12, 2, 2 );
   shapes[ 2 ] = new Sphere( 5, 1.5, 4.5 );
   shapes[ 3 ] = new Cube( 2.2 );
   for ( int x = 0; x < 4; ++x )
      cout << *( shapes[ x ] ) << '\n';
   return 0;
}
```