0.0.1 Point/Circle Hierarchy using private data

Class **Point3** (Figs. 1-2) declares data members **x** and **y** as **private** and exposes member functions **setX, getX, setY, getY** and **print** for manipulating these values.

1 Case Study: Three-Level Inheritance Hierarchy

Three level point/circle/cylinder hierarchy

- Point
  - x-y coordinate pair
- Circle
  - x-y coordinate pair
  - Radius
- Cylinder
  - x-y coordinate pair
  - Radius
  - Height

Derive class **Cylinder** from class **Circle4**. Class **Cylinder** should redefine member functions **getArea** and **print** member functions. Figs. 7-8 present class **Cylinder**, which inherits from class **Circle4**. We were able to develop classes Circle4 and Cylinder much more quickly by using inheritance than if we had developed these classes "from scratch". Inheritance avoids duplicating code and the associated code-maintenance problems.

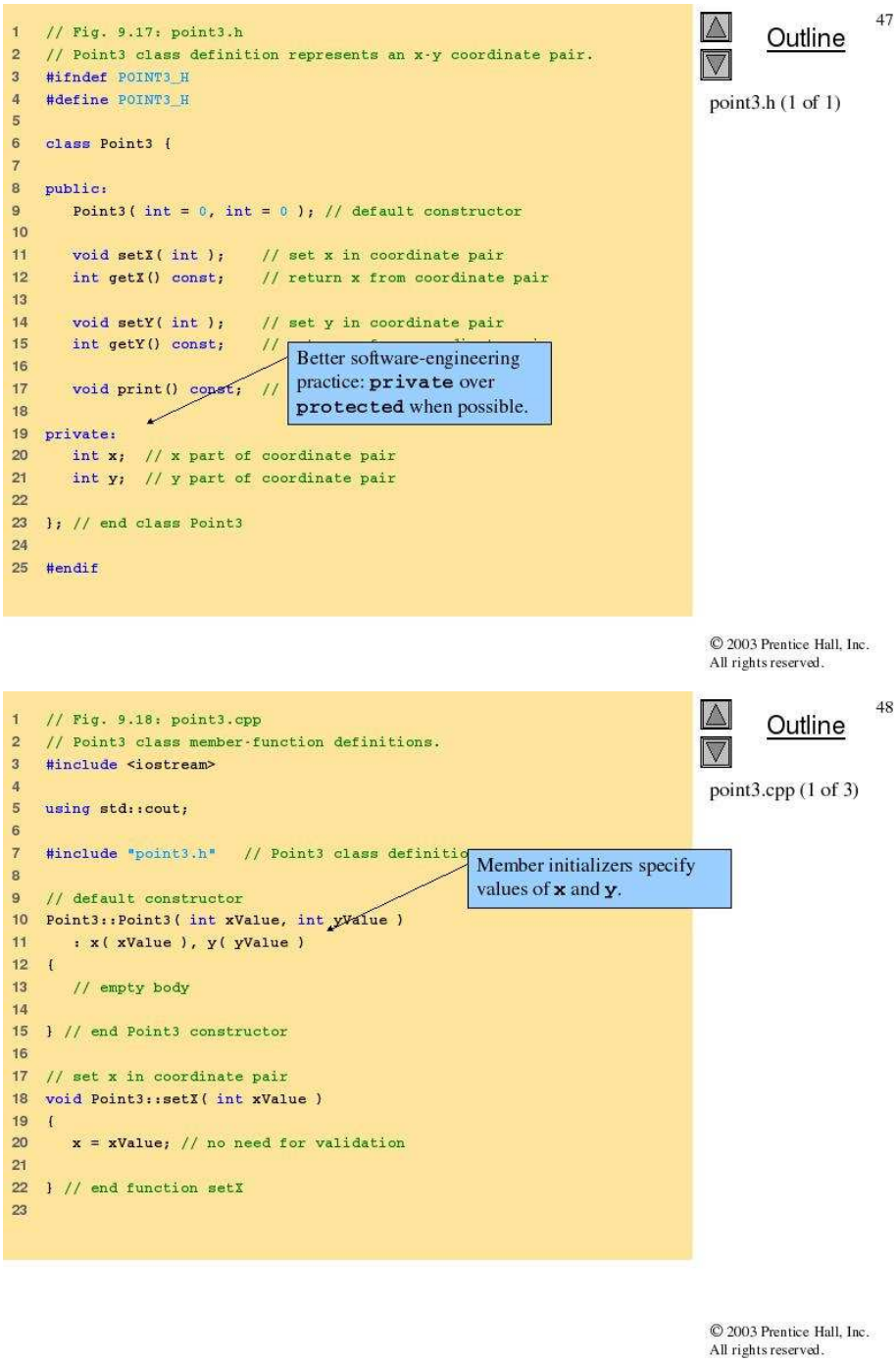


Figure 1: **Point3** class header file. Point/Circle Hierarchy Using **private** Data

Figure 2: **Point3** class uses member functions to manipulate its **private** data.

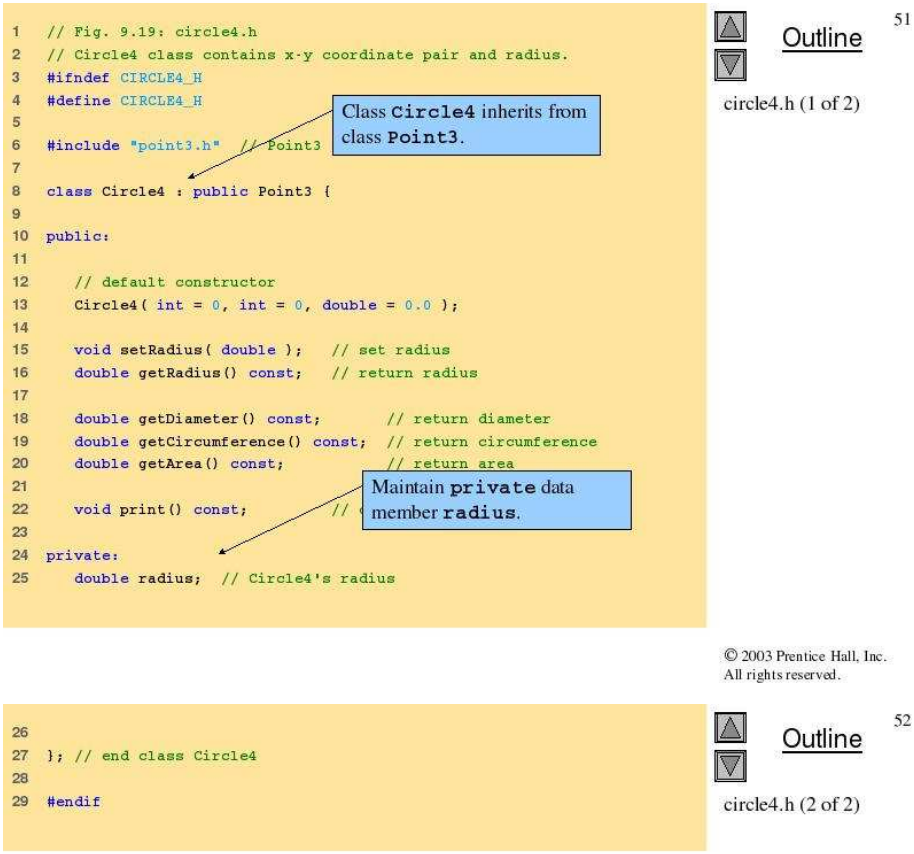


Figure 3: **Circle4** class header file.


```
1  // Fig. 9.20: circle4.cpp
2  // Circle4 class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "circle4.h"    // Circle4 cl
8
9  // default constructor
10 Circle4::Circle4( int xValue, int yV
11    : Point3( xValue, yValue )  // call base-class constructor
12 {
13    setRadius( radiusValue );
14
15 } // end Circle4 constructor
16
17 // set radius
18 void Circle4::setRadius( double radiusValue )
19 {
20    radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
21
22 } // end function setRadius
23
```

```
24 // return radius
25 double Circle4::getRadius() const
26 {
27    return radius;
28
29 } // end function getRadius
30
31 // calculate and return diameter
32 double Circle4::getDiameter() const
33 {
34    return 2 * getRadius();
35
36 } // end function getDiameter
37
38 // calculate and return circumference
39 double Circle4::getCircumference() const
40 {
41    return 3.14159 * getDiameter();
42
43 } // end function getCircumference
44
```

Figure 4: **Circle4** class that inherits from class **Point3**, which does not provide **protected** data. (part 1 of 2)

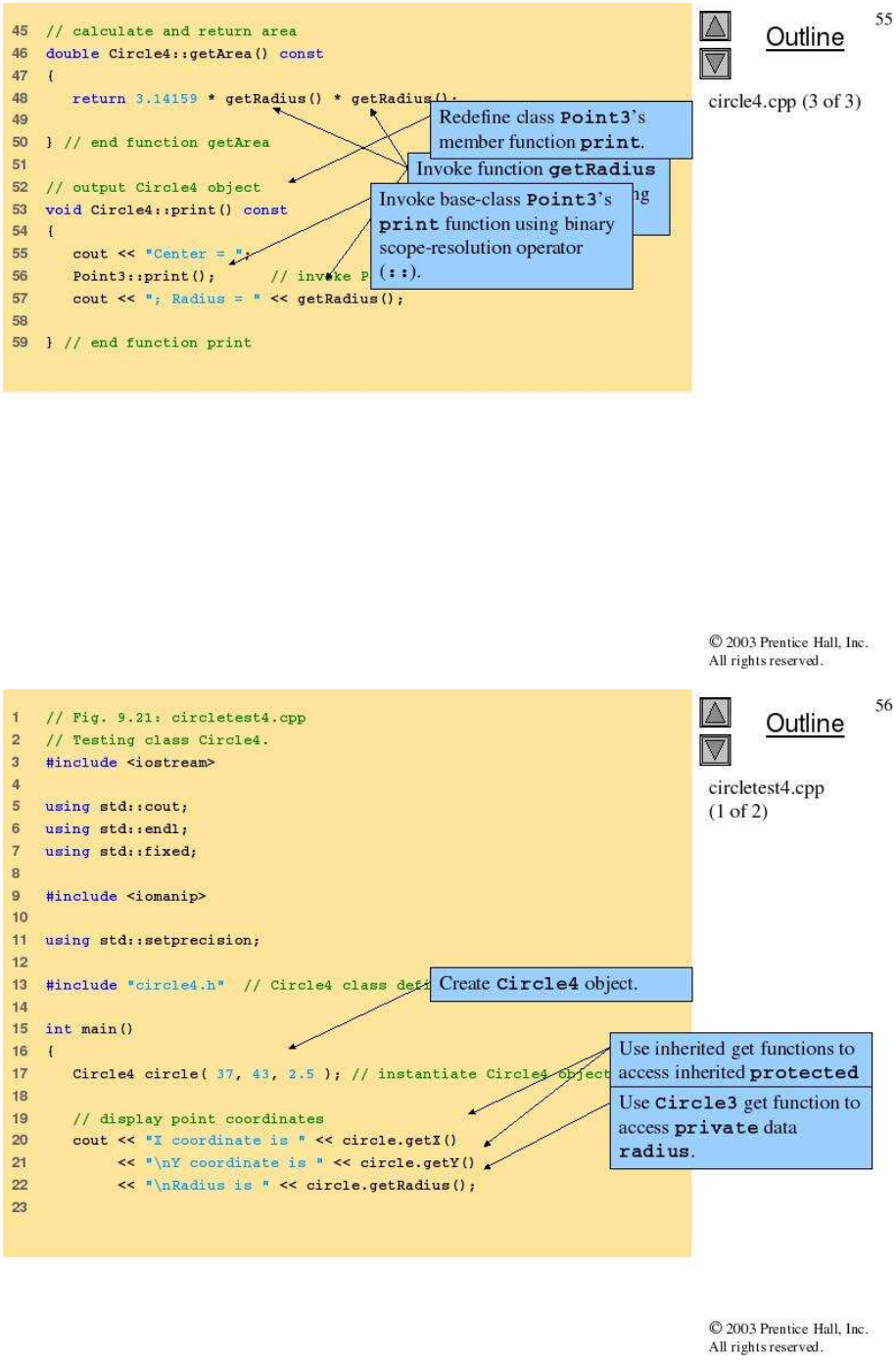
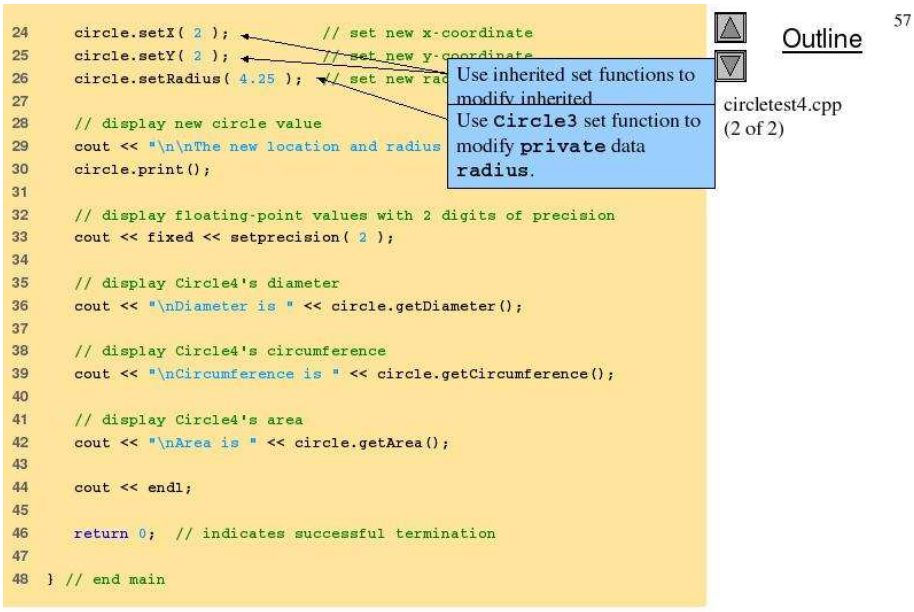


Figure 5: **Circle4** class that inherits from class **Point3**, which does not provide **protected** data. (part 2 of 2)



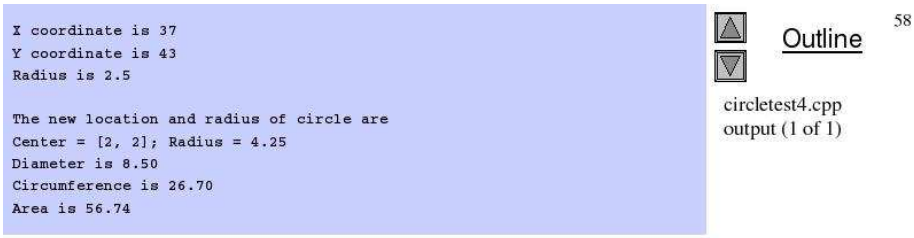
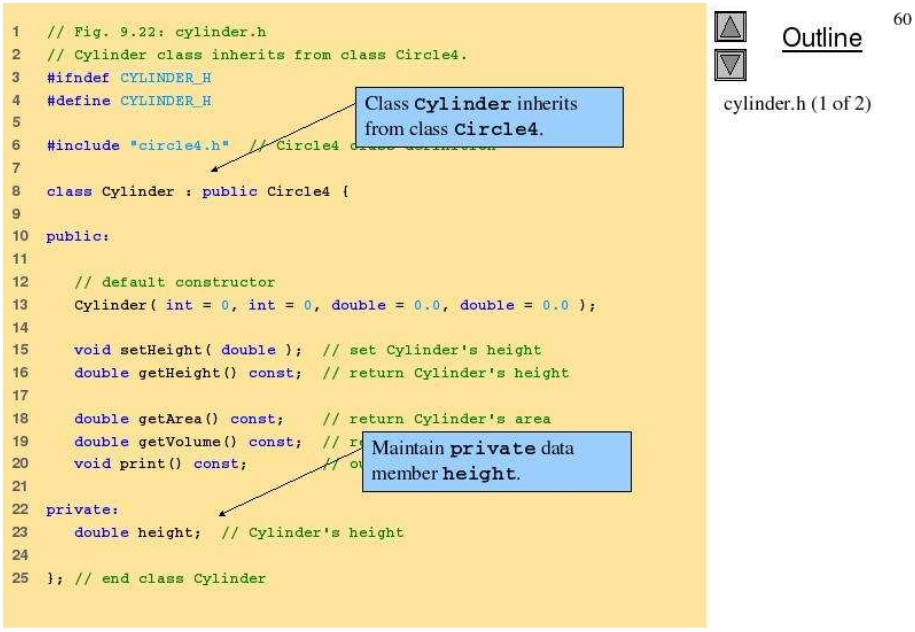


Figure 6: Base class **private** data is accessible to a derived class via **public** or **protected** member function inherited by the derived class.



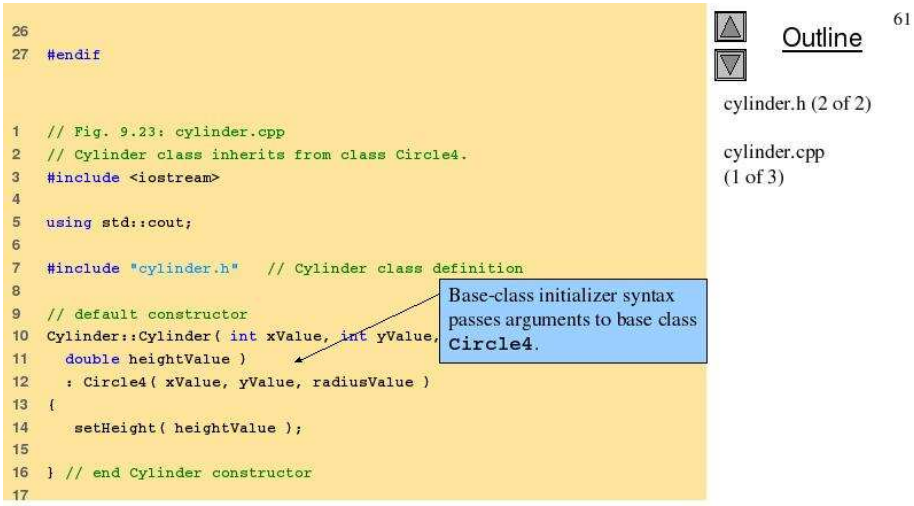


Figure 7: **Cylinder** class header file.

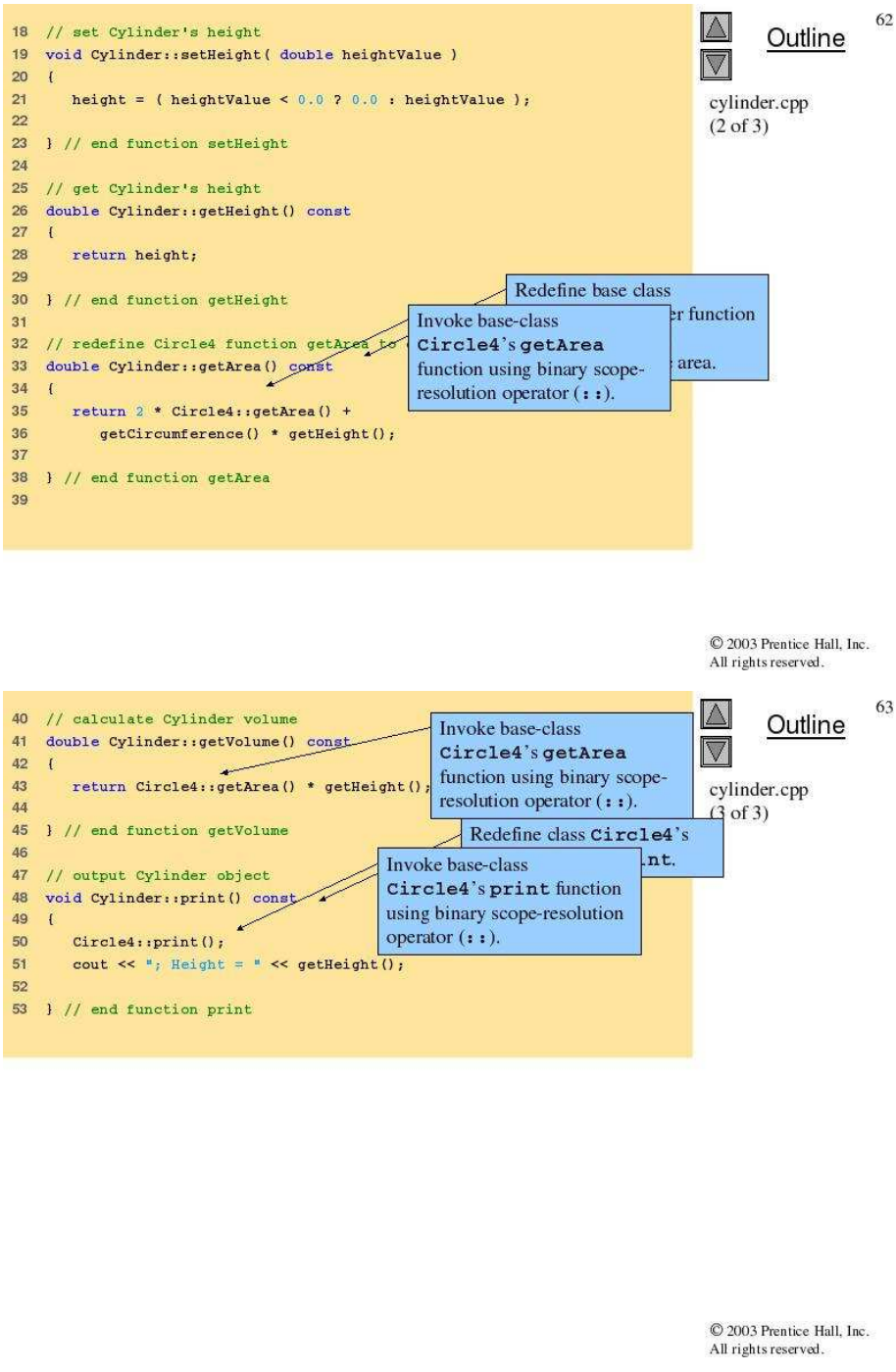


Figure 8: **Cylinder** class inherits from class **Circle4** and redefines member function **getArea**.

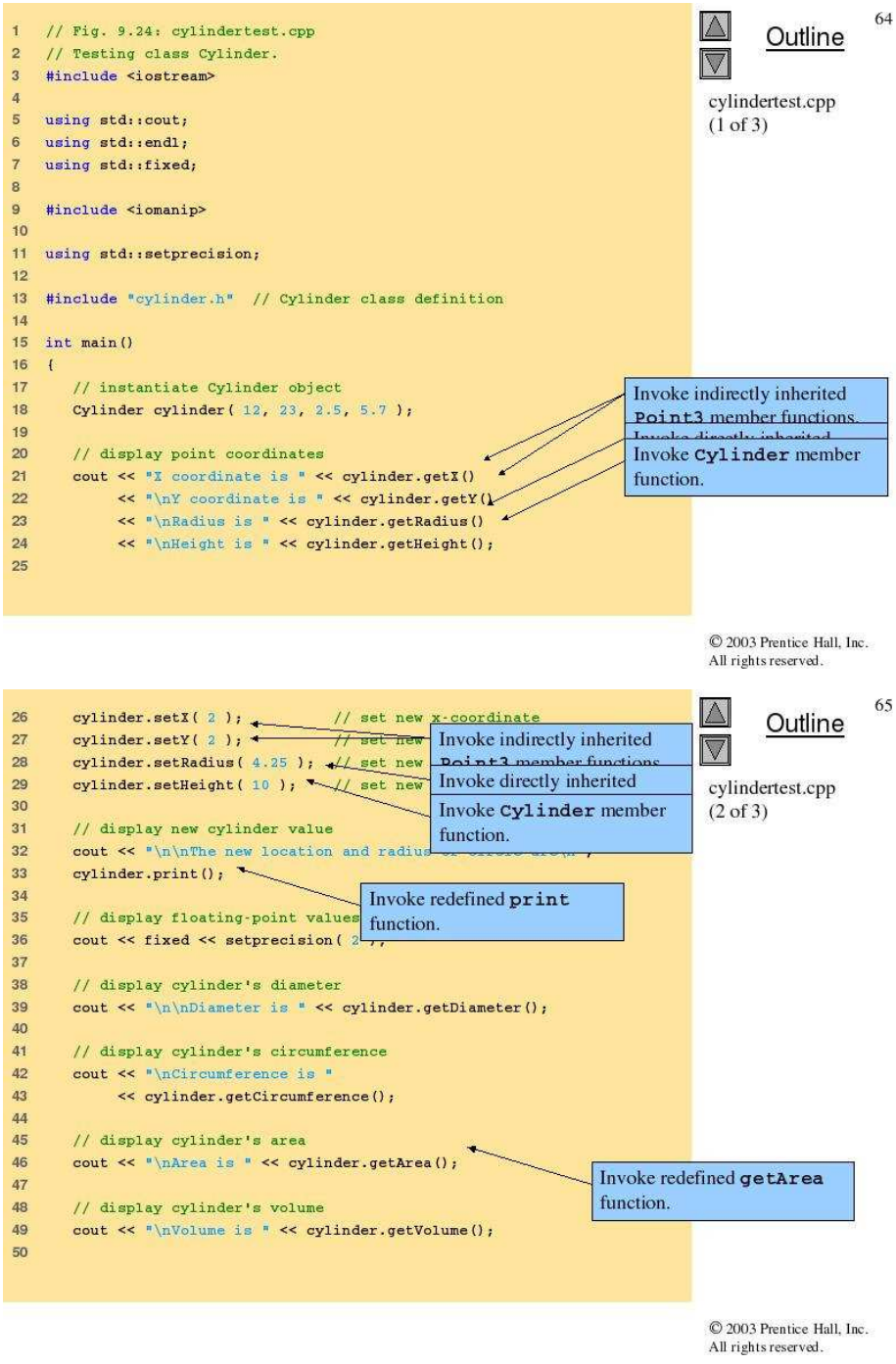


Figure 9: **Point/Circle/Cylinder** hierarchy test program. (part 1 of 2)

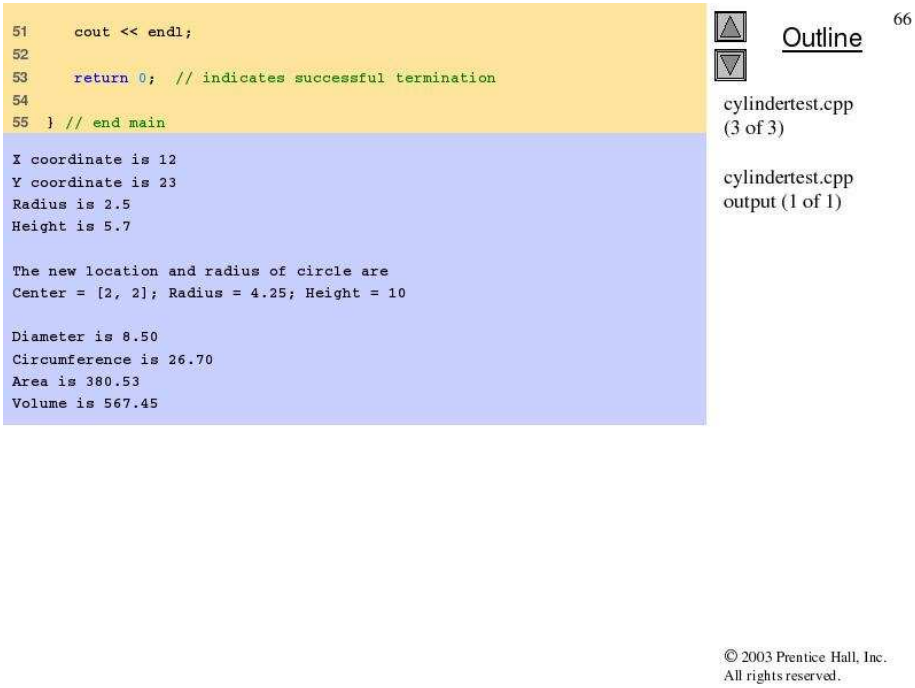


Figure 10: **Point/Circle/Cylinder** hierarchy test program. (part 2 of 2)

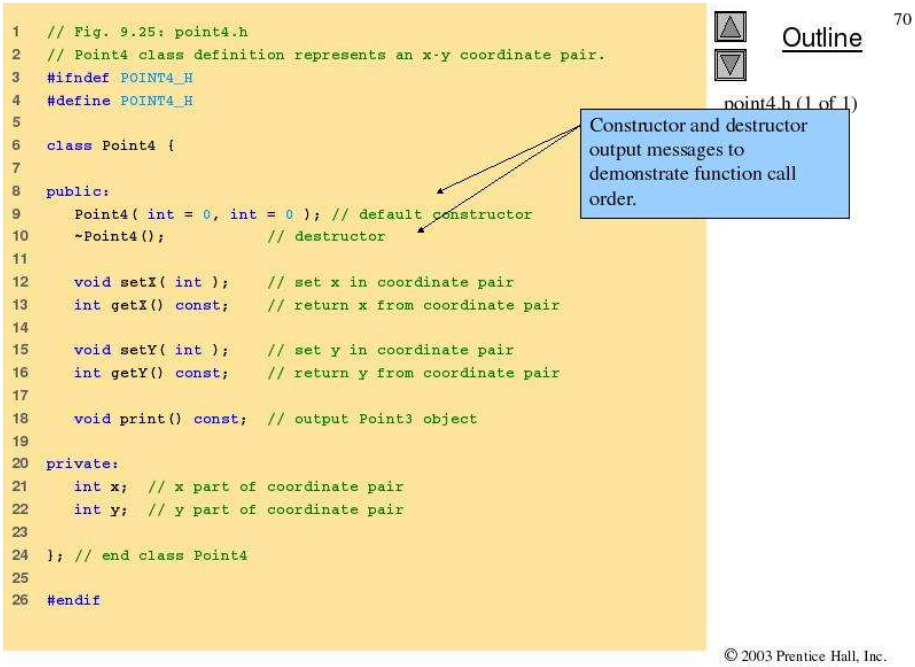
1.1 Constructors and Destructors in Derived Classes

- Instantiating derived-class object
  - Chain of constructor calls
    - Derived-class constructor invokes base class constructor
      - Implicitly or explicitly
    - Base of inheritance hierarchy
      - Last constructor called in chain
      - First constructor body to finish executing
      - Example: **Point3/Circle4/Cylinder** hierarchy
        Point3 constructor called last
        Point3 constructor body finishes execution first
    - Initializing data members
      - Each base-class constructor initializes data members
        Inherited by derived class
- Destroying derived-class object

  - Chain of destructor calls
    * Reverse order of constructor chain
    * Destructor of derived-class called first
    * Destructor of next base class up hierarchy next
      · Continue up hierarchy until final base reached; After final base-class destructor, object removed from memory

- Base-class constructors, destructors, assignment operators
  - Not inherited by derived classes
  - Derived class constructors, assignment operators can call
    * Constructors
    * Assignment operators

Next example revisits the point/circle hierarchy by defining class **Point4** (11-12) and class **Circle5** (13-15) that contain constructors and destructors, each of which prints a message when it is invoked.



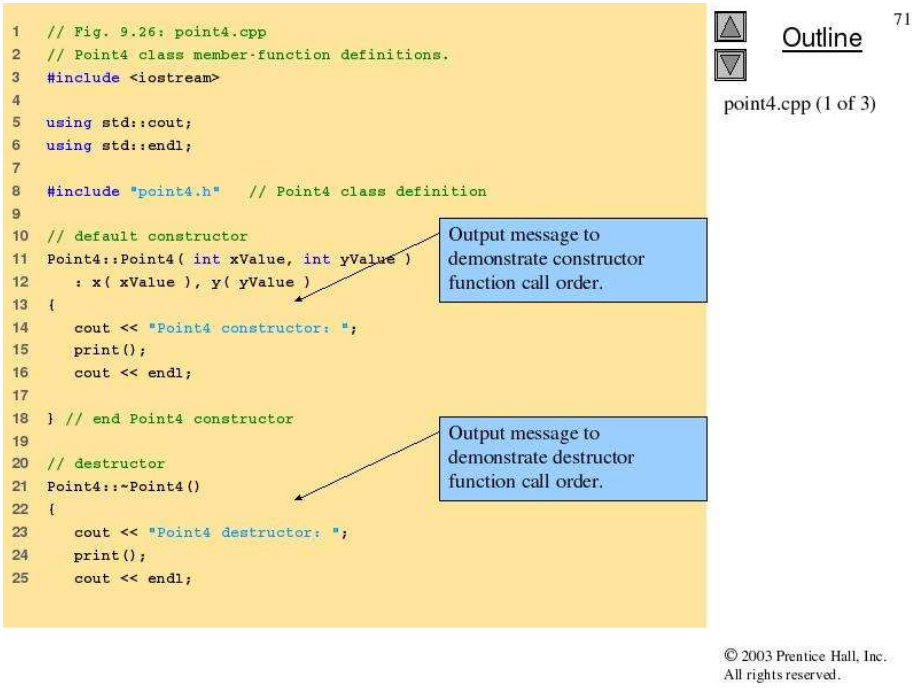


Figure 11: **Point4** class header file and **Point4** base class contains a constructor and a destructor. (part 1 of 2)

```
26
27 } // end Point4 destructor
28
29 // set x in coordinate pair
30 void Point4::setX( int xValue )
31 {
32    x = xValue; // no need for validation
33
34 } // end function setX
35
36 // return x from coordinate pair
37 int Point4::getX() const
38 {
39    return x;
40
41 } // end function getX
42
43 // set y in coordinate pair
44 void Point4::setY( int yValue )
45 {
46    y = yValue; // no need for validation
47
48 } // end function setY
49
```

Outline

72

point4.cpp (2 of 3)


```
50 // return y from coordinate pair
51 int Point4::getY() const
52 {
53    return y;
54
55 } // end function getY
56
57 // output Point4 object
58 void Point4::print() const
59 {
60    cout << '[' << getX() << ", " << getY() << ']';
61
62 } // end function print
```

Outline

73

point4.cpp (3 of 3)

Figure 12: **Point4** base class contains a constructor and a destructor. (part 2of 2)

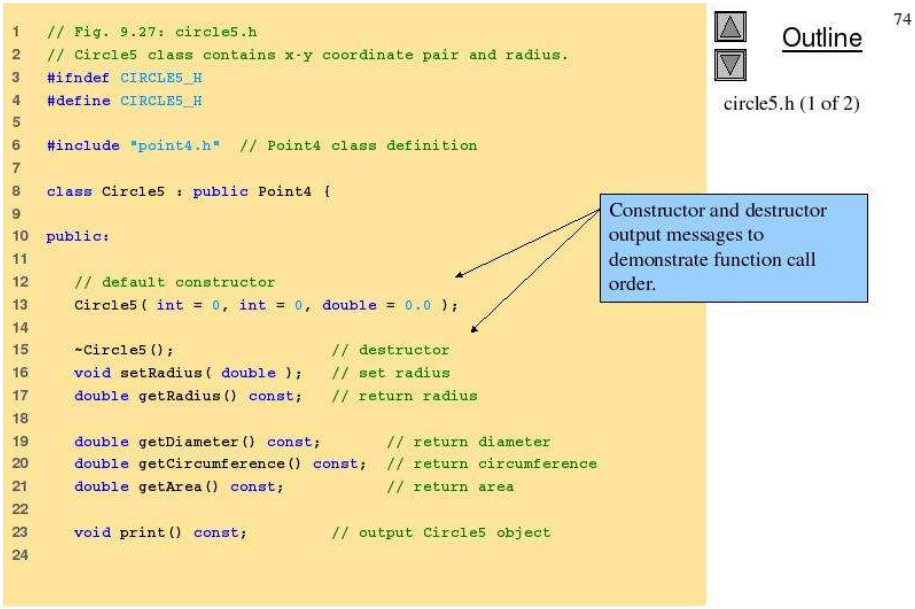
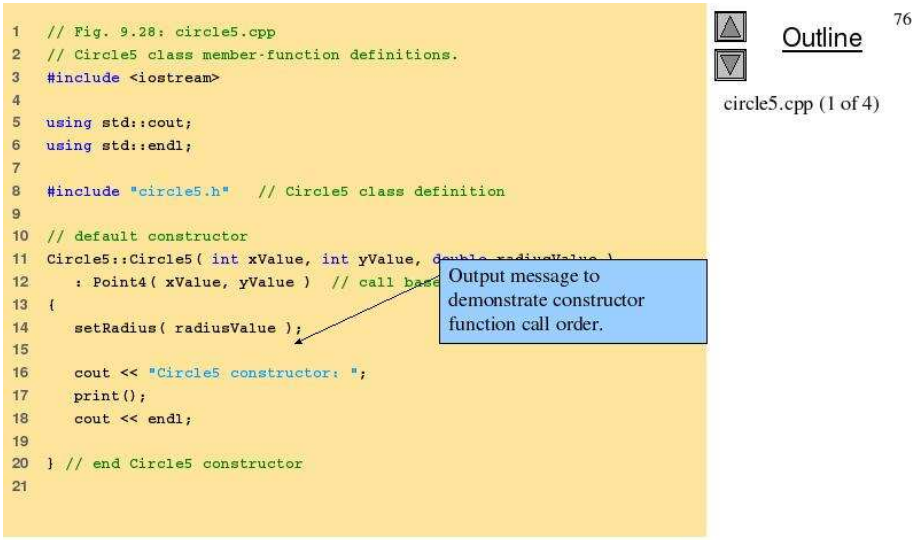


Figure 13: **Circle5** class header file.



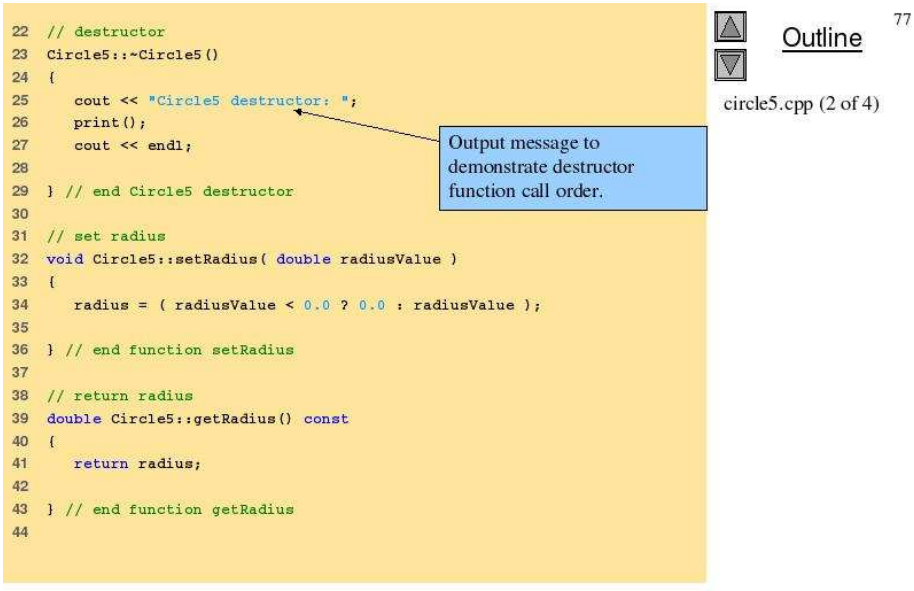


Figure 14: **Circle5** class inherits from class **Point4**. (part 1 of 2)

```
45 // calculate and return diameter
46 double Circle5::getDiameter() const
47 {
48    return 2 * getRadius();
49
50 } // end function getDiameter
51
52 // calculate and return circumference
53 double Circle5::getCircumference() const
54 {
55    return 3.14159 * getDiameter();
56
57 } // end function getCircumference
58
59 // calculate and return area
60 double Circle5::getArea() const
61 {
62    return 3.14159 * getRadius() * getRadius();
63
64 } // end function getArea
65
```

Outline

78

circle5.cpp (3 of 4)


```
66 // output Circle5 object
67 void Circle5::print() const
68 {
69    cout << "Center = ";
70    Point4::print();      // invoke Point4's print function
71    cout << "; Radius = " << getRadius();
72
73 } // end function print
```

Outline

79

circle5.cpp (4 of 4)

Figure 15: **Circle5** class inherits from class **Point4**. (part 2 of 2)

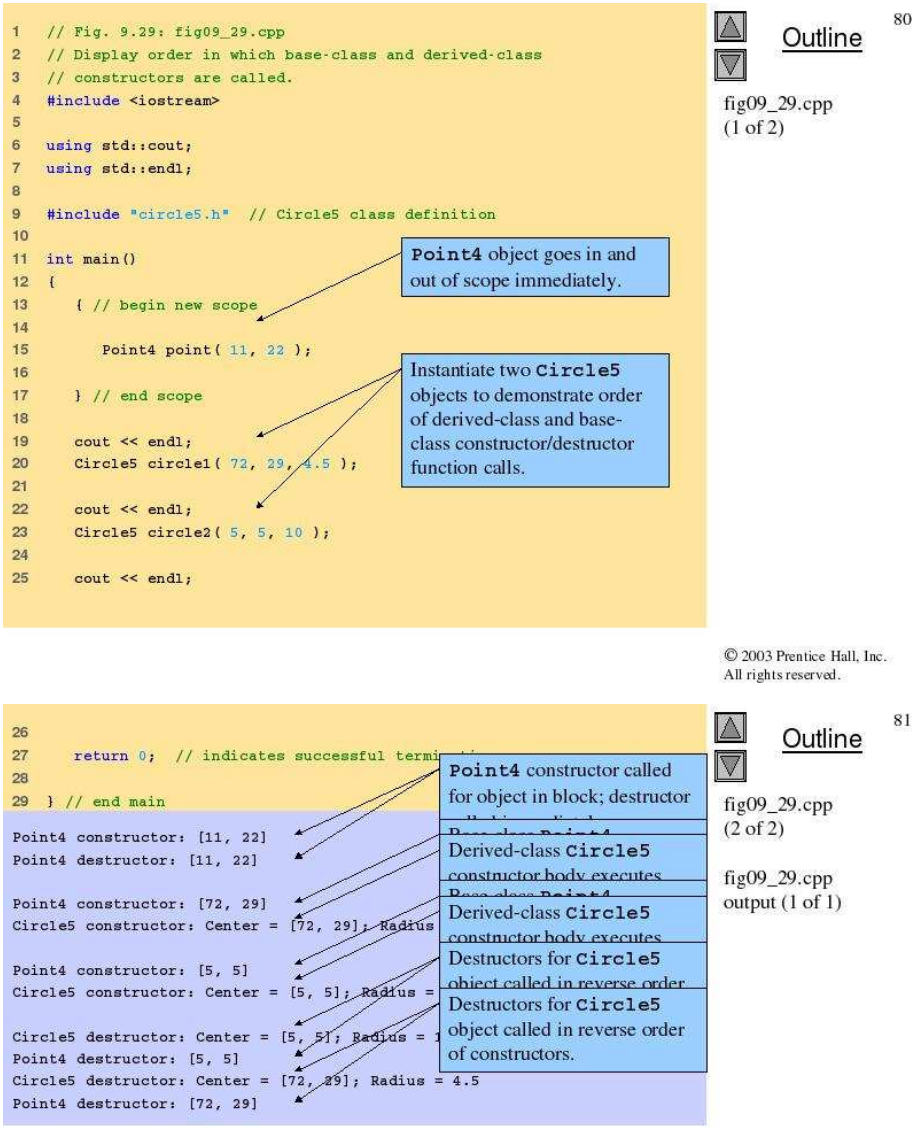


Figure 16: Constructor and destructor call order.

1.2 "Uses A" and "Knows A" Relationships

- **"Uses a"**
  - Object uses another object
    - Call non-**private** member function; using pointer, reference or object name
- **"Knows a"** (association)
  - Object aware of another object; contain pointer handle or reference handle
  - Knowledge networks

1.3 public, protected and private Inheritance

83

9.8 public, protected and private Inheritance

Base class member access specifier	Type of inheritance		
	public inheritance	protected inheritance	private inheritance
Public	public in derived class. Can be accessed directly by any non-static member functions, friend functions and non-member functions.	protected in derived class. Can be accessed directly by all non-static member functions and friend functions.	private in derived class. Can be accessed directly by all non-static member functions and friend functions.
Protected	protected in derived class. Can be accessed directly by all non-static member functions and friend functions.	protected in derived class. Can be accessed directly by all non-static member functions and friend functions.	private in derived class. Can be accessed directly by all non-static member functions and friend functions.
Private	Hidden in derived class. Can be accessed by non-static member functions and friend functions through public or protected member functions of	Hidden in derived class. Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.	Hidden in derived class. Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.

Figure 17: Summary of base–class member accessibility in a derived class.

1.4 Software Engineering with Inheritance

Customizing existing software

- Inherit from existing classes
  - Include additional members
  - Redefine base-class members
  - No direct access to base class's source code; Link to object code
- Independent software vendors (ISVs)
  - Develop proprietary code for sale/license; available in object-code format
  - Users derive new classes; without accessing ISV proprietary source code