Illustrations List (Main Page)

Fig. 9.1	Some simple inheritance examples.
Fig. 9.2	An inheritance hierarchy for university community members.
Fig. 9.3	A portion of a Shape class hierarchy.
Fig. 9.4	Casting base-class pointers to derived-class pointers.
Fig. 9.5	Overriding a base-class member function in a derived class.
Fig. 9.6	Summary of base-class member accessibility in a derived class.
Fig. 9.7	Order in which base-class and derived-class constructors and destructors are called.
Fig. 9.8	Demonstrating class Point .
Fig. 9.9	Demonstrating class Circle .
Fig. 9.10	Demonstrating class Cylinder .
Fig. 9.11	Demonstrating multiple inheritance.

Base class	Derived classes
Student	GraduateStudent UndergraduateStudent
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeImprovementLoan MortgageLoan
Employee	FacultyMember StaffMember
Account	CheckingAccount SavingsAccount

Fig. 9.1 Some simple inheritance examples.

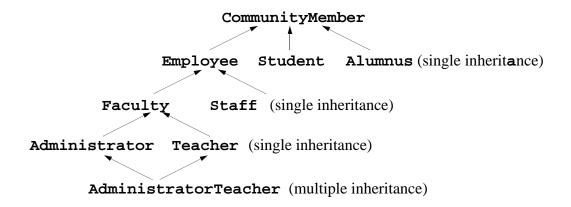


Fig. 9.2 An inheritance hierarchy for university community members.

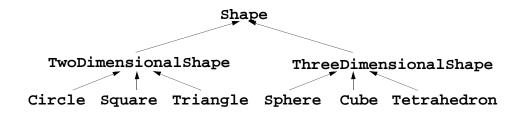


Fig. 9.3 A portion of a **Shape** class hierarchy.

```
// Fig. 9.4: point.h
    // Definition of class Point
    #ifndef POINT H
    #define POINT_H
 6
    class Point {
        friend ostream &operator<<( ostream &, const Point & );</pre>
 8
    public:
 9
       Point( int = 0, int = 0 );
                                            // default constructor
10
        void setPoint( int, int );
                                           // set coordinates
        int getX() const { return x; } // get x coordinate
int getY() const { return y; } // get y coordinate
11
12
13
                         // accessible by derived classes
    protected:
14
        int x, y;
                         // x and y coordinates of the Point
15
    };
16
17
    #endif
Fig. 9.4
        Casting base-class pointers to derived-class pointers (part 1 of 6).
    // Fig. 9.4: point.cpp
    // Member functions for class Point
20
    #include <iostream.h>
    #include "point.h"
23
    // Constructor for class Point
24
   Point::Point( int a, int b ) { setPoint( a, b ); }
25
26 // Set x and y coordinates of Point
27
    void Point::setPoint( int a, int b )
28
    {
29
        x = a;
30
        y = b;
31
32
33
    // Output Point (with overloaded stream insertion operator)
34
    ostream &operator<<( ostream &output, const Point &p )</pre>
35
36
        output << '[' << p.x << ", " << p.y << ']';
37
38
        return output; // enables cascaded calls
39
    }
Fig. 9.4 Casting base-class pointers to derived-class pointers (part 2 of 6).
40 // Fig. 9.4: circle.h
41
    // Definition of class Circle
42
   #ifndef CIRCLE_H
43 #define CIRCLE_H
44
45 #include <iostream.h>
46
    #include <iomanip.h>
47
    #include "point.h"
48
    class Circle : public Point {    // Circle inherits from Point
    friend ostream &operator<<( ostream &, const Circle & );</pre>
49
50
51
    public:
52
       // default constructor
53
        Circle( double r = 0.0, int x = 0, int y = 0);
54
55
        void setRadius( double ); // set radius
```

// return radius

// calculate area

56

57

double getRadius() const;

double area() const;

```
protected:
       double radius;
60
    };
62
    #endif
        Casting base-class pointers to derived-class pointers (part 3 of 6).
   // Fig. 9.4: circle.cpp
   // Member function definitions for class Circle
65
    #include "circle.h"
66
67
    // Constructor for Circle calls constructor for Point
68
    // with a member initializer then initializes radius.
69
    Circle::Circle( double r, int a, int b )
70
       : Point(a,b)
                             // call base-class constructor
71
    { setRadius( r ); }
72
73
    // Set radius of Circle
74
    void Circle::setRadius( double r )
75
       { radius = ( r >= 0 ? r : 0 ); }
76
77
    // Get radius of Circle
78
    double Circle::getRadius() const { return radius; }
80
    // Calculate area of Circle
81
    double Circle::area() const
82
       { return 3.14159 * radius * radius; }
83
84
   // Output a Circle in the form:
85
   // Center = [x, y]; Radius = #.##
86
   ostream &operator<<( ostream &output, const Circle &c )
87
88
       output << "Center = " << static_cast< Point >( c )
89
              << "; Radius = "
90
              << setiosflags( ios::fixed | ios::showpoint )
91
              << setprecision( 2 ) << c.radius;
92
93
       return output;
                       // enables cascaded calls
94
    }
Fig. 9.4 Casting base-class pointers to derived-class pointers (part 4 of 6).
   // Fig. 9.4: fig09_04.cpp
   // Casting base-class pointers to derived-class pointers
   #include <iostream.h>
98 #include <iomanip.h>
99 #include "point.h"
100 #include "circle.h"
101
102 int main()
103 {
104
       Point *pointPtr = 0, p(30, 50);
105
       Circle *circlePtr = 0, c( 2.7, 120, 89 );
106
107
       cout << "Point p: " << p << "\nCircle c: " << c << '\n';
108
109
       // Treat a Circle as a Point (see only the base class part)
110
       pointPtr = &c; // assign address of Circle to pointPtr
111
       cout << "\nCircle c (via *pointPtr): "</pre>
112
```

<< *pointPtr << '\n';

// Treat a Circle as a Circle (with some casting)

113 114

```
115
       pointPtr = &c; // assign address of Circle to pointPtr
116
       // cast base-class pointer to derived-class pointer
117
118
       circlePtr = static_cast< Circle * >( pointPtr );
119
       cout << "\nCircle c (via *circlePtr):\n" << *circlePtr</pre>
120
            << "\nArea of c (via circlePtr): "
121
            << circlePtr->area() << '\n';
122
123
       // DANGEROUS: Treat a Point as a Circle
124
       pointPtr = &p; // assign address of Point to pointPtr
125
126
       // cast base-class pointer to derived-class pointer
127
       circlePtr = static_cast< Circle * >( pointPtr );
128
       cout << "\nPoint p (via *circlePtr):\n" << *circlePtr</pre>
129
            << "\nArea of object circlePtr points to: "
130
            << circlePtr->area() << endl;
131
       return 0;
132 }
```

Fig. 9.4 Casting base-class pointers to derived-class pointers (part 5 of 6).

```
Point p: [30, 50]
Circle c: Center = [120, 89]; Radius = 2.70

Circle c (via *pointPtr): [120, 89]

Circle c (via *circlePtr):
Center = [120, 89]; Radius = 2.70
Area of c (via circlePtr): 22.90

Point p (via *circlePtr):
Center = [30, 50]; Radius = 0.00
Area of object circlePtr points to: 0.00
```

Fig. 9.4 Casting base-class pointers to derived-class pointers (part 6 of 6).

```
// Fig. 9.5: employ.h
   // Definition of class Employee
   #ifndef EMPLOY_H
   #define EMPLOY_H
6
   class Employee {
   public:
8
      Employee( const char *, const char * ); // constructor
      void print() const; // output first and last name
10
                           // destructor
      ~Employee();
11
  private:
12
      char *firstName;
                          // dynamically allocated string
13
      char *lastName;
                          // dynamically allocated string
14
   };
15
16
   #endif
```

Fig. 9.5 Overriding a base-class member function in a derived class (part 1 of 5).

```
// Fig. 9.5: employ.cpp
18
    // Member function definitions for class Employee
19
    #include <string.h>
20
   #include <iostream.h>
21
    #include <assert.h>
22
    #include "employ.h"
24
   // Constructor dynamically allocates space for the
25
   // first and last name and uses strcpy to copy
26
    // the first and last names into the object.
27
    Employee::Employee( const char *first, const char *last )
28
29
       firstName = new char[ strlen( first ) + 1 ];
30
       assert( firstName != 0 ); // terminate if not allocated
31
       strcpy( firstName, first );
32
33
       lastName = new char[ strlen( last ) + 1 ];
34
       assert( lastName != 0 ); // terminate if not allocated
35
       strcpy( lastName, last );
36
    }
37
38
    // Output employee name
39
    void Employee::print() const
40
       { cout << firstName << ' ' << lastName; }</pre>
41
   // Destructor deallocates dynamically allocated memory
42
43
   Employee::~Employee()
44
    {
45
       delete [] firstName; // reclaim dynamic memory
46
       delete [] lastName;
                              // reclaim dynamic memory
47
    }
       Overriding a base-class member function in a derived class (part 2 of 5).
Fig. 9.5
   // Fig. 9.5: hourly.h
49
   // Definition of class HourlyWorker
50
   #ifndef HOURLY_H
51
    #define HOURLY_H
52
53
    #include "employ.h"
54
55
    class HourlyWorker : public Employee {
56
57
       HourlyWorker( const char*, const char*, double, double );
58
       double getPay() const; // calculate and return salary
59
       void print() const;
                               // overridden base-class print
60
   private:
61
       double wage;
                               // wage per hour
62
       double hours;
                               // hours worked for week
63
    };
64
65 #endif
```

Fig. 9.5 Overriding a base-class member function in a derived class (part 3 of 5)

```
// Fig. 9.5: hourly.cpp
    // Member function definitions for class HourlyWorker
    #include <iostream.h>
68
69
    #include <iomanip.h>
70 #include "hourly.h"
71
72
    // Constructor for class HourlyWorker
73
    HourlyWorker::HourlyWorker( const char *first,
74
                                 const char *last,
75
                                 double initHours, double initWage )
76
       : Employee( first, last ) // call base-class constructor
77
78
       hours = initHours; // should validate
79
       wage = initWage;
                            // should validate
80
    }
81
82
    // Get the HourlyWorker's pay
83
    double HourlyWorker::getPay() const { return wage * hours; }
84
85 // Print the HourlyWorker's name and pay
86 void HourlyWorker::print() const
87
88
       cout << "HourlyWorker::print() is executing\n\n";</pre>
89
       Employee::print(); // call base-class print function
90
91
       cout << " is an hourly worker with pay of $"</pre>
92
            << setiosflags( ios::fixed | ios::showpoint )
93
            << setprecision( 2 ) << getPay() << endl;
94
    }
Fig. 9.5
        Overriding a base-class member function in a derived class (part 4 of 5).
   // Fig. 9.5: fig.09_05.cpp
   // Overriding a base-class member function in a
97
   // derived class.
98 #include <iostream.h>
99 #include "hourly.h"
100
101 int main()
102 {
103
       HourlyWorker h( "Bob", "Smith", 40.0, 10.00 );
104
       h.print();
105
       return 0;
106 }
         HourlyWorker::print() is executing
         Bob Smith is an hourly worker with pay of $400.00
```

Fig. 9.5 Overriding a base-class member function in a derived class (part 5 of 5).

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

Fig. 9.6 Summary of base-class member accessibility in a derived class.

```
// Fig. 9.7: point2.h
    // Definition of class Point
    #ifndef POINT2_H
    #define POINT2_H
 6
    class Point {
    public:
 8
       Point( int = 0, int = 0 ); // default constructor
 Q
       ~Point(); // destructor
10
   protected:
                    // accessible by derived classes
11
       int x, y;
                    // x and y coordinates of Point
12
    };
14
    #endif
Fig. 9.7 Order in which base-class and derived-class constructors and destructors are called (part 1 of 5).
   // Fig. 9.7: point2.cpp
   // Member function definitions for class Point
17
    #include <iostream.h>
18 #include "point2.h"
20
    // Constructor for class Point
21
    Point::Point( int a, int b )
    {
23
       x = a;
24
       y = b;
25
26
       cout << "Point constructor: "</pre>
27
            << '[' << x << ", " << y << ']' << endl;
28
    }
29
30
    // Destructor for class Point
31
    Point::~Point()
32
33
       cout << "Point destructor: "</pre>
34
             << '[' << x << ", " << y << ']' << endl;
35
    }
Fig. 9.7 Order in which base-class and derived-class constructors and destructors are called (part 2 of 5).
36 // Fig. 9.7: circle2.h
    // Definition of class Circle
38
   #ifndef CIRCLE2_H
30
    #define CIRCLE2_H
40
41
   #include "point2.h"
42
43 class Circle: public Point {
44 public:
45
       // default constructor
46
       Circle( double r = 0.0, int x = 0, int y = 0);
47
48
       ~Circle();
49
    private:
50
       double radius;
51
    };
53
    #endif
```

Fig. 9.7 Order in which base-class and derived-class constructors and destructors are called (part 3 of 5).

```
54 // Fig. 9.7: circle2.cpp
   // Member function definitions for class Circle
56 #include "circle2.h"
57
58 // Constructor for Circle calls constructor for Point
59 Circle::Circle( double r, int a, int b )
60
                        // call base-class constructor
       : Point(a,b)
61 {
62
       radius = r; // should validate
63
       cout << "Circle constructor: radius is "</pre>
64
            << radius << " [" << x << ", " << y << ']' << endl;
65
66
67
   // Destructor for class Circle
68 Circle::~Circle()
69
70
       cout << "Circle destructor: radius is "</pre>
71
            << radius << " [" << x << ", " << y << ']' << endl;
72
    }
Fig. 9.7 Order in which base-class and derived-class constructors and destructors are called (part 4 of 5).
73 // Fig. 9.7: fig09_07.cpp
    // Demonstrate when base-class and derived-class
    // constructors and destructors are called.
76 #include <iostream.h>
   #include "point2.h"
78 #include "circle2.h"
79
80 int main()
81
    {
82
       // Show constructor and destructor calls for Point
83
84
          Point p( 11, 22 );
85
       }
86
87
       cout << endl;</pre>
88
       Circle circle1( 4.5, 72, 29 );
89
       cout << endl;</pre>
90
       Circle circle2( 10, 5, 5 );
91
       cout << endl;</pre>
92
       return 0;
93
    }
         Point constructor: [11, 22]
         Point destructor: [11, 22]
         Point constructor: [72, 29]
         Circle constructor: radius is 4.5 [72, 29]
         Point constructor: [5, 5]
         Circle constructor: radius is 10 [5, 5]
         Circle destructor: radius is 10 [5, 5]
         Point destructor: [5, 5]
         Circle destructor: radius is 4.5 [72, 29]
         Point destructor: [72, 29]
```

Fig. 9.7 Order in which base-class and derived-class constructors and destructors are called (part 5 of 5).

```
// Fig. 9.8: point2.h
    // Definition of class Point
    #ifndef POINT2_H
    #define POINT2_H
 6
    class Point {
       friend ostream &operator<<( ostream &, const Point & );</pre>
 8
    public:
 9
       Point( int = 0, int = 0 );
                                          // default constructor
10
       void setPoint( int, int );
                                         // set coordinates
       int getX() const { return x; } // get x coordinate
int getY() const { return y; } // get y coordinate
11
12
                      // accessible to derived classes
13
    protected:
14
       int x, y;
                       // coordinates of the point
15
    };
16
17
    #endif
Fig. 9.8
        Demonstrating class Point (part 1 of 3).
   // Fig. 9.8: point2.cpp
    // Member functions for class Point
    #include <iostream.h>
    #include "point2.h"
23
    // Constructor for class Point
24
   Point::Point( int a, int b ) { setPoint( a, b ); }
26
   // Set the x and y coordinates
27
    void Point::setPoint( int a, int b )
28
    {
29
       x = a;
30
       y = b;
31
    }
32
33
    // Output the Point
34
    ostream &operator<<( ostream &output, const Point &p )
35
36
       output << '[' << p.x << ", " << p.y << ']';
37
38
                                // enables cascading
       return output;
39
    }
Fig. 9.8 Demonstrating class Point (part 2 of 3).
40 // Fig. 9.8: fig09_08.cpp
   // Driver for class Point
42
   #include <iostream.h>
43 #include "point2.h"
44
45
    int main()
46
    {
47
       Point p( 72, 115 ); // instantiate Point object p
48
49
       // protected data of Point inaccessible to main
       cout << "X coordinate is " << p.getX()</pre>
50
             << "\nY coordinate is " << p.getY();
51
52
53
       p.setPoint( 10, 10 );
54
       cout << "\n\nThe new location of p is " << p << endl;</pre>
55
56
       return 0;
57
    }
```

```
X coordinate is 72
Y coordinate is 115
The new location of p is [10, 10]
```

Fig. 9.8 Demonstrating class Point (part 3 of 3).

```
// Fig. 9.9: circle2.h
    // Definition of class Circle
    #ifndef CIRCLE2_H
    #define CIRCLE2_H
    #include "point2.h"
 8
    class Circle : public Point {
 9
       friend ostream &operator<<( ostream &, const Circle & );</pre>
10
   public:
11
       // default constructor
12
       Circle( double r = 0.0, int x = 0, int y = 0);
13
                                  // set radius
       void setRadius( double );
                                    // return radius
14
       double getRadius() const;
15
       double area() const;
                                     // calculate area
16
   protected:
                        // accessible to derived classes
17
       double radius;
                       // radius of the Circle
18
19
20
   #endif
Fig. 9.9
        Demonstrating class Circle (part 1 of 5).
    // Fig. 9.9: circle2.cpp
   // Member function definitions for class Circle
    #include <iostream.h>
24
    #include <iomanip.h>
25
    #include "circle2.h"
26
27
    // Constructor for Circle calls constructor for Point
28
    // with a member initializer and initializes radius
29
    Circle::Circle( double r, int a, int b )
30
                             // call base-class constructor
       : Point(a,b)
31
    { setRadius( r ); }
32
33
    // Set radius
34
    void Circle::setRadius( double r )
35
       { radius = ( r >= 0 ? r : 0 ); }
36
    // Get radius
38
    double Circle::getRadius() const { return radius; }
```

Fig. 9.9 Demonstrating class Circle (part 2 of 5).

```
39
40
    // Calculate area of Circle
    double Circle::area() const
42
       { return 3.14159 * radius * radius; }
43
44
   // Output a circle in the form:
45
   // Center = [x, y]; Radius = #.##
46
   ostream &operator << ( ostream &output, const Circle &c )
47
48
       output << "Center = " << static_cast< Point > ( c )
49
              << "; Radius = "
50
              << setiosflags( ios::fixed | ios::showpoint )
51
              << setprecision( 2 ) << c.radius;
53
       return output; // enables cascaded calls
54
    }
Fig. 9.9
       Demonstrating class Circle (part 3 of 5).
55 // Fig. 9.9: fig09_09.cpp
   // Driver for class Circle
    #include <iostream.h>
    #include "point2.h"
59
   #include "circle2.h"
60
61
    int main()
62
    {
63
       Circle c( 2.5, 37, 43 );
64
65
       cout << "X coordinate is " << c.getX()</pre>
66
            << "\nY coordinate is " << c.getY()
67
            << "\nRadius is " << c.getRadius();
68
69
       c.setRadius( 4.25 );
70
       c.setPoint( 2, 2 );
71
       cout << "\n new location and radius of c are\n"
72
            << c << "\nArea " << c.area() << '\n';
73
74
       Point &pRef = c;
75
       cout << "\nCircle printed as a Point is: " << pRef << endl;</pre>
76
77
       return 0;
78
    }
```

Fig. 9.9 Demonstrating class Circle (part 4 of 5).

```
X coordinate is 37
Y coordinate is 43
Radius is 2.5

The new location and radius of c are
Center = [2, 2]; Radius = 4.25
Area 56.74

Circle printed as a Point is: [2, 2]
```

Fig. 9.9 Demonstrating class Circle (part 5 of 5).

```
// Fig. 9.10: cylindr2.h
    // Definition of class Cylinder
    #ifndef CYLINDR2_H
    #define CYLINDR2_H
    #include "circle2.h"
 8
    class Cylinder : public Circle {
 Q
       friend ostream & operator << ( ostream &, const Cylinder & );
10
11
    public:
12
       // default constructor
13
       Cylinder( double h = 0.0, double r = 0.0,
14
                 int x = 0, int y = 0);
15
16
       void setHeight( double );
                                   // set height
17
       double getHeight() const; // return height
18
       double area() const;
                                   // calculate and return area
19
       double volume() const;
                                   // calculate and return volume
20
21
    protected:
       double height;
                                   // height of the Cylinder
23
    };
24
25 #endif
Fig. 9.10 Demonstrating class Cylinder (part 1 of 5)
26  // Fig. 9.10: cylindr2.cpp
27 // Member and friend function definitions
28 // for class Cylinder.
   #include <iostream.h>
   #include <iomanip.h>
    #include "cylindr2.h"
Fig. 9.10 Demonstrating class Cylinder (part 2 of 5)
   // Cylinder constructor calls Circle constructor
    Cylinder::Cylinder( double h, double r, int x, int y )
35
       : Circle( r, x, y ) // call base-class constructor
36
    { setHeight( h ); }
38
    // Set height of Cylinder
39
    void Cylinder::setHeight( double h )
40
       { height = ( h >= 0 ? h : 0 ); }
41
42
   // Get height of Cylinder
43
   double Cylinder::getHeight() const { return height; }
44
45
   // Calculate area of Cylinder (i.e., surface area)
46
    double Cylinder::area() const
47
    {
48
       return 2 * Circle::area() +
49
              2 * 3.14159 * radius * height;
50
    }
    // Calculate volume of Cylinder
53
    double Cylinder::volume() const
54
       { return Circle::area() * height; }
55
   // Output Cylinder dimensions
    ostream &operator<<( ostream &output, const Cylinder &c )
```

```
58
59
       output << static_cast< Circle >( c )
60
              << "; Height = " << c.height;
61
62
       return output; // enables cascaded calls
63
    }
Fig. 9.10 Demonstrating class Cylinder (part 3 of 5).
64 // Fig. 9.10: fig09_10.cpp
65 // Driver for class Cylinder
66 #include <iostream.h>
67
    #include <iomanip.h>
68 #include "point2.h"
   #include "circle2.h"
69
70 #include "cylindr2.h"
71
72
   int main()
73
74
       // create Cylinder object
75
       Cylinder cyl( 5.7, 2.5, 12, 23 );
76
Fig. 9.10 Demonstrating class Cylinder (part 4 of 5).
       // use get functions to display the Cylinder
78
       cout << "X coordinate is " << cyl.getX()</pre>
79
            << "\nY coordinate is " << cyl.getY()
80
             << "\nRadius is " << cyl.getRadius()
81
             << "\nHeight is " << cyl.getHeight() << "\n\n";
82
83
       // use set functions to change the Cylinder's attributes
84
       cyl.setHeight( 10 );
85
       cyl.setRadius( 4.25 );
86
       cyl.setPoint( 2, 2 );
87
       cout << "The new location, radius, and height of cyl are:\n"</pre>
88
             << cyl << '\n';
89
90
       // display the Cylinder as a Point
91
       Point &pRef = cyl; // pRef "thinks" it is a Point
92
       cout << "\nCylinder printed as a Point is: "</pre>
93
            << pRef << "\n\n";
94
95
       // display the Cylinder as a Circle
96
       Circle &circleRef = cyl; // circleRef thinks it is a Circle
97
       cout << "Cylinder printed as a Circle is:\n" << circleRef</pre>
98
            << "\nArea: " << circleRef.area() << endl;
99
100
       return 0;
101 }
```

```
X coordinate is 12
Y coordinate is 23
Radius is 2.5
Height is 5.7

The new location, radius, and height of cyl are:
Center = [2, 2]; Radius = 4.25; Height = 10.00

Cylinder printed as a Point is: [2, 2]

Cylinder printed as a Circle is:
Center = [2, 2]; Radius = 4.25
Area: 56.74
```

Fig. 9.10 Demonstrating class Cylinder (part 5 of 5).

```
// Fig. 9.11: base1.h
   // Definition of class Base1
   #ifndef BASE1_H
 4
    #define BASE1_H
 6
    class Base1 {
    public:
 8
       Basel( int x ) { value = x; }
       int getData() const { return value; }
10 protected: // accessible to derived classes
11
       int value; // inherited by derived class
12
    };
13
    #endif
Fig. 9.11 Demonstrating multiple inheritance (part 1 of 6).
15 // Fig. 9.11: base2.h
16 // Definition of class Base2
17 #ifndef BASE2_H
18 #define BASE2_H
19
20 class Base2 {
21 public:
22 Base:
       Base2( char c ) { letter = c; }
23
       char getData() const { return letter; }
24
   protected:
                     // accessible to derived classes
       char letter; // inherited by derived class
26
28 #endif
```

Fig. 9.11 Demonstrating multiple inheritance (part 2 of 6).

```
// Fig. 9.11: derived.h
30
   // Definition of class Derived which inherits
    // multiple base classes (Base1 and Base2).
    #ifndef DERIVED_H
33
    #define DERIVED_H
34
35
   #include "base1.h"
36 #include "base2.h"
37
38
   // multiple inheritance
39
    class Derived : public Base1, public Base2 {
40
       friend ostream &operator << ( ostream &, const Derived & );
41
42
    public:
43
       Derived( int, char, double );
44
       double getReal() const;
45
46 private:
47
       double real; // derived class's private data
48
    };
49
50
    #endif
Fig. 9.11 Demonstrating multiple inheritance (part 3 of 6).
   // Fig. 9.11: derived.cpp
   // Member function definitions for class Derived
   #include <iostream.h>
54
   #include "derived.h"
55
56 // Constructor for Derived calls constructors for
    // class Base1 and class Base2.
58
    // Use member initializers to call base-class constructors
59
   Derived::Derived( int i, char c, double f )
60
       : Base1( i ), Base2( c ), real ( f ) { }
61
62
   // Return the value of real
63
    double Derived::getReal() const { return real; }
64
65
   // Display all the data members of Derived
66
   ostream &operator<<( ostream &output, const Derived &d )
67
    {
68
       output << "
                      Integer: " << d.value</pre>
69
              << "\n Character: " << d.letter
70
              << "\nReal number: " << d.real;
72
       return output; // enables cascaded calls
73
    }
```

Fig. 9.11 Demonstrating multiple inheritance (part 4 of 6).

```
// Fig. 9.11: fig09_11.cpp
    // Driver for multiple inheritance example
76
   #include <iostream.h>
    #include "base1.h"
78 #include "base2.h"
79 #include "derived.h"
80
81
   int main()
82
    {
83
       Base1 b1( 10 ), *base1Ptr = 0; // create Base1 object
84
       Base2 b2( 'Z' ), *base2Ptr = 0; // create Base2 object
85
       Derived d( 7, 'A', 3.5 );
                                        // create Derived object
86
87
       // print data members of base class objects
88
       cout << "Object b1 contains integer " << b1.getData()</pre>
89
            << "\nObject b2 contains character " << b2.getData()
            << "\nObject d contains:\n" << d << "\n\n";
90
91
92
       // print data members of derived class object
93
       // scope resolution operator resolves getData ambiguity
94
       cout << "Data members of Derived can be"</pre>
95
            << " accessed individually:"
96
            << "\n
                     Integer: " << d.Base1::getData()</pre>
97
            << "\n Character: " << d.Base2::getData()
98
            << "\nReal number: " << d.getReal() << "\n\n";
99
100
       cout << "Derived can be treated as an "</pre>
101
            << "object of either base class:\n";
102
103
       // treat Derived as a Basel object
104
       base1Ptr = &d;
105
       cout << "baselPtr->getData() yields "
106
            << base1Ptr->getData() << '\n';
107
108
       // treat Derived as a Base2 object
109
      base2Ptr = &d;
110
       cout << "base2Ptr->getData() yields "
111
            << base2Ptr->getData() << endl;
112
113
       return 0;
114 }
```

Fig. 9.11 Demonstrating multiple inheritance (part 5 of 6).

```
Object b1 contains integer 10
Object b2 contains character Z
Object d contains:
    Integer: 7
    Character: A
Real number: 3.5

Data members of Derived can be accessed individually:
    Integer: 7
    Character: A
Real number: 3.5

Derived can be treated as an object of either base class: base1Ptr->getData() yields 7
base2Ptr->getData() yields A
```

Fig. 9.11 Demonstrating multiple inheritance (part 6 of 6).