

W 9.1 - Inheritance

Outline

- 9.1 Introduction
- 9.2 Inheritance: Base Classes and Derived Classes
- 9.3 Protected Members
- 9.4 Casting Base-Class Pointers to Derived-Class Pointers
- 9.5 Using Member Functions
- 9.6 Overriding Base-Class Members in a Derived Class
- 9.7 Public, Protected and Private Inheritance
- 9.8 Direct Base Classes and Indirect Base Classes
- 9.9 Using Constructors and Destructors in Derived Classes
- 9.10 Implicit Derived-Class Object to Base-Class Object Conversion
- 9.11 Software Engineering with Inheritance
- 9.12 Composition vs. Inheritance
- 9.13 "Uses A" and "Knows A" Relationships
- 9.14 Case Study: Point, Circle, Cylinder
- 9.15 Multiple Inheritance

© 2000 Prentice Hall, Inc. All rights reserved.

9.1 Introduction

- Inheritance
 - New classes created from existing classes
 - Absorb attributes and behaviors
 - Derived class
 - Class that inherits data members and member functions from a previously defined base class
 - Single inheritance
 - Class inherits from one base class
 - Multiple inheritance
 - Class inherits from multiple base classes
 - Types of inheritance
 - public: Derived objects are accessible by the base class objects
 - private: Derived objects are inaccessible by the base class
 - protected: Derived classes and friends can access protected members of the base class

© 2000 Prentice Hall, Inc. All rights reserved.

9.1 Introduction

- Polymorphism
 - Write programs in a general fashion
 - Handle a wide variety of existing (and unspecified) related classes

© 2000 Prentice Hall, Inc. All rights reserved.

9.2 Inheritance: Base and Derived Classes

- Base and derived classes
 - Often an object from a derived class (subclass) is also an object of a base class (superclass)
 - A rectangle is a derived class in reference to a quadrilateral and a base class in reference to a square
- Inheritance examples

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

© 2000 Prentice Hall, Inc. All rights reserved.

9.2 Inheritance: Base and Derived Classes

- Implementation of **public** inheritance

```
class CommissionWorker : public Employee {
    ...
};
```

 - Class **CommissionWorker** inherits from class **Employee**
 - **friend** functions not inherited
 - **private** members of base class not accessible from derived class

© 2000 Prentice Hall, Inc. All rights reserved.

9.3 protected Members

- **protected** access
 - Intermediate level of protection between **public** and **private** inheritance
 - Derived-class members can refer to **public** and **protected** members of the base class simply by using the member names
 - Note that **protected** data "breaks" encapsulation

© 2000 Prentice Hall, Inc. All rights reserved.

9.4 Casting Base-Class Pointers to Derived Class Pointers

- Derived classes relationships to base classes
 - Objects of a derived class can be treated as objects of the base class
 - Reverse not true — base class objects cannot be derived-class objects
- Downcasting a pointer
 - Use an explicit cast to convert a base-class pointer to a derived-class pointer
 - If pointer is going to be dereferenced, the type of the pointer must match the type of object to which the pointer points
 - Format:


```
derivedPtr = static_cast< DerivedClass * > basePtr;
```

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

9.4 Casting Base-Class Pointers to Derived-Class Pointers

- The following example:
 - Demonstrates the casting of base class pointers to derived class pointers
 - Class **Circle** is derived from class **Point**
 - A pointer of type **Point** is used to reference a **Circle** object, and a pointer to type **Circle** is used to reference a **Point** object

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

```
1 // Fig. 9.4: point.h
2 // Definition of class Point
3 #ifndef POINT_H
4 #define POINT_H
5
6 #include <iostream>
7
8 using std::ostream;
9
10 class Point {
11     friend ostream& operator<<( ostream &, const Point & );
12 public:
13     Point( int = 0, int = 0 ); // default constructor
14     void setPoint( int, int ); // set coordinates
15     int getX() const { return x; } // get x coordinate
16     int getY() const { return y; } // get y coordinate
17 protected: // accessible by derived classes
18     int x, y; // x and y coordinates of the Point
19 };
20
21 #endif
22 // Fig. 9.4: point.cpp
23 // Member functions for class Point
24 #include <iostream>
25 #include "point.h"
26
27 // Constructor for class Point
28 Point::Point( int a, int b ) { setPoint( a, b ); }
29
30 // Set x and y coordinates of Point
31 void Point::setPoint( int a, int b )
32 {
33     x = a;
```

Outline

- 1. Point class definition
- 1. Load header
- 1.1 Function definitions

```
34     y = b;
35 }
36
37 // Output Point (with overloaded stream insertion operator)
38 ostream& operator<<( ostream soutput, const Point &p )
39 {
40     output << '[' << p.x << ", " << p.y << ']'>
41     return output; // enables cascaded calls
42 }
43
44 // Fig. 9.4: circle.h
45 // Definition of class Circle
46 #ifndef CIRCLE_H
47 #define CIRCLE_H
48
49 #include <iostream>
50
51 using std::ostream;
52
53 #include <iomanip>
54
55 using std::ios;
56 using std::setiosflags;
57 using std::setprecision;
58
59 #include "point.h"
60
61 class Circle : public Point { // Circle inherits from Point
62     friend ostream& operator<<( ostream &, const Circle & );
63 public:
64     // default constructor
```

Outline

- 1.1 Function definitions
- 1. Circle class definition

Class Circle publicly inherits from class Point, so it will have class Point's public and protected member functions and data.

```
65     Circle( double r = 0.0, int x = 0, int y = 0 );
66
67     void setRadius( double ); // set radius
68     double getRadius() const; // return radius
69     double area() const; // calculate area
70 protected:
71     double radius;
72 };
73
74 #endif
75 // Fig. 9.4: circle.cpp
76 // Member function definitions for class Circle
77 #include "circle.h"
78
79 // Constructor for Circle calls constructor for Point
80 // With a member initializer then initializes radius.
81 Circle::Circle( double r, int a, int b )
82     : Point( a, b ) // call base-class constructor
83     { setRadius( r ); }
84
85 // Set radius of Circle
86 void Circle::setRadius( double r )
87 { radius = ( r >= 0 ? r : 0 ); }
88
```

Outline

- 1. Circle definition
- 1. Load header
- 1.1 Function

Circle inherits from Point, and has Point's data members (which are set by calling Point's constructor).

```
89 // Get radius of Circle
90 double Circle::getRadius() const { return radius; }
91
92 // Calculate area of Circle
93 double Circle::area() const
94     { return 3.14159 * radius * radius; }
95
96 // Output a Circle in the form:
97 // Center = [x, y]; Radius = #.##
98 ostream& operator<<( ostream soutput, const Circle &c )
99 {
100     output << "Center = " << static_cast< Point >( c )
101     << "; Radius = "
102     << setiosflags( ios::fixed | ios::showpoint )
103     << setprecision( 2 ) << c.radius;
104
105     return output; // enables cascaded calls
106 }
107 // Fig. 9.4: Fig09_04.cpp
108 // Casting base-class pointers to derived-class pointers
109 #include <iostream>
110
111 using std::cout;
112 using std::endl;
113
114 #include <iomanip>
115
116 #include "point.h"
117 #include "circle.h"
118
119 int main()
120 {
121     Point *pointPtr = 0, p( 30, 50 );
```

Outline

- 1.1 Function Definitions
- Driver
- 1. Load headers
- 1.1 Initialize objects

```

122 Circle *circlePtr = 0, c( 2.7, 120, 89 );
123
124 cout << "Point p: " << p << "\nCircle c: " << c << '\n';
125
126 // Treat a Circle as a Point
127 pointPtr = &c; // assign address of Circle to pointPtr
128 cout << "\nCircle c (via *pointPtr): "
129 << *pointPtr << '\n';
130
131 // Assign pointPtr to a Point object. It has no derived-class information.
132 When it is cast to a Circle *, circlePtr is really assigned to a base-class object with no derived-class information. This is dangerous.
133
134 // DANGEROUS: Treat a Point as a Circle
135 pointPtr = &p; // assign address of Point to pointPtr
136
137 // cast base-class pointer to derived-class pointer
138 circlePtr = static_cast< Circle *>( pointPtr );
139 cout << "\nPoint p (via *circlePtr):\n" << *circlePtr
140 << "\nArea of object circlePtr points to: ";
141 circlePtr->area() << endl;
142 return 0;
143 }

```

Outline

1.2 Assign objects

2. Function calls

objects

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

Dynamic casting
Derived-class pointer
>(pointPtr);
Circle c (via *circlePtr):
Center = [120, 89]; Radius = 2.70
Area of c (via circlePtr): 22.90

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

The base class pointer only "sees" the base-class part of the object it points to.

Cast pointPtr into a Circle *, and assign to circlePtr.

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

```

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

```

Outline

Program Output

© 2000 Prentice Hall, Inc. All rights reserved.

9.5 Using Member Functions

- Derived class member functions
 - Cannot directly access **private** members of their base class
 - Maintains encapsulation
 - Hiding **private** members is a huge help in testing, debugging and correctly modifying systems

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

9.6 Overriding Base-Class Members in a Derived Class

- To override a base-class member function
 - In the derived class, supply a new version of that function with the same signature
 - same function name, different definition
 - When the function is then mentioned by name in the derived class, the derived version is automatically called
 - The scope-resolution operator may be used to access the base class version from the derived class

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

```

1 // Fig. 9.5: employ.h
2 // Definition of class Employee
3 #ifndef EMPLOY_H
4 #define EMPLOY_H
5
6 class Employee {
7 public:
8     Employee( const char *, const char * ); // constructor
9     void print() const; // output first and last name
10    ~Employee(); // destructor
11 private:
12    char *firstName; // dynamically allocated string
13    char *lastName; // dynamically allocated string
14 };
15
16 #endif
17 // Fig. 9.5: employ.cpp
18 // Member function definitions for class Employee
19 #include <iostream>
20
21 using std::cout;
22
23 #include <cstring>
24 #include <cassert>
25 #include "employ.h"
26
27 // Constructor dynamically allocates space for the
28 // first and last name and uses strcpy to copy
29 // the first and last names into the object.
30 Employee::Employee( const char *first, const char *last )
31 {
32     firstName = new char[ strlen( first ) + 1 ];

```

Outline

1. Employee class definition

1.1 Function definitions

```

33 assert( firstName != 0 ); // terminate if not allocated
34 strcpy( firstName, first );
35
36 lastName = new char[ strlen( last ) + 1 ];
37 assert( lastName != 0 ); // terminate if not allocated
38 strcpy( lastName, last );
39 }
40
41 // Output employee name
42 void Employee::print() const
43 { cout << firstName << " " << lastName; }
44
45 // Destructor deallocates dynamically allocated memory
46 Employee::~Employee()
47 {
48     delete [] firstName; // reclaim dynamic memory
49     delete [] lastName; // reclaim dynamic memory
50 }
51 // Fig. 9.5: hourly.h
52 // Definition of class HourlyWorker
53 #ifndef HOURLY_H
54 #define HOURLY_H
55
56 #include "employ.h"
57
58 class HourlyWorker : public Employee {
59 public:
60     HourlyWorker( const char*, const char*, double, double );
61     double getPay() const; // calculate and return salary
62     void print() const; // overridden base-class print
63 private:

```

Outline

1.1 Function definitions

1. HourlyWorker class definition

HourlyWorker inherits from Employee.

HourlyWorker will override the print function.

```

64 double wage;           // wage per hour
65 double hours;         // hours worked for week
66 };
67
68 #endif
69 // Fig. 9.5: hourly.cpp
70 // Member function definitions for class HourlyWorker
71 #include <iostream>
72
73 using std::cout;
74 using std::endl;
75
76 #include <iomanip>
77
78 using std::ios;
79 using std::setiosflags;
80 using std::setprecision;
81
82 #include "hourly.h"
83
84 // Constructor for class HourlyWorker
85 HourlyWorker::HourlyWorker( const char *first,
86                           const char *last,
87                           double initHours, double initWage )
88     : Employee( first, last ) // call base-class constructor
89 {
90     hours = initHours; // should validate
91     wage = initWage; // should validate
92 }
93
94 // Get the HourlyWorker's pay
95 double HourlyWorker::getPay() const { return wage * hours; }

```

Outline
1. Load header
1.1 Function definitions

```

96
97 // Print the HourlyWorker's name and pay
98 void HourlyWorker::print() const
99 {
100     cout << "HourlyWorker::print() is executing\n\n";
101     Employee::print(); // call base-class print function
102
103     cout << " is an hourly worker with pay of $"
104           << setiosflags( ios::fixed | ios::showpoint )
105           << setprecision( 2 ) << getPay() << endl;
106 }
107 // Fig. 9.5: fig09_05.cpp
108 // Overriding a base-class member function in a
109 // derived class.
110 #include "hourly.h"
111
112 int main()
113 {
114     HourlyWorker h( "Bob", "Smith", 40.0, 10.0 );
115     h.print();
116     return 0;
117 }

```

Outline
1.1 Function Definitions
1. Load header

The print function is overridden in HourlyWorker. However, the new function still can call the original print function using : :

HourlyWorker::print() is executing
Bob Smith is an hourly worker with pay of \$400.00
Program Output

9.7 public, private, and protected 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 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.	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.

© 2000 Prentice Hall, Inc. All rights reserved.

9.8 Direct and Indirect Base Classes

- Direct base class
 - Explicitly listed derived class's header with the colon (:) notation when that derived class is declared


```
class HourlyWorker : public Employee
```

 - Employee is a direct base class of HourlyWorker
- Indirect base class
 - Not listed in derived class's header
 - Inherited from two or more levels up the class hierarchy


```
class MinuteWorker : public HourlyWorker
```

 - Employee is an indirect base class of MinuteWorker

© 2000 Prentice Hall, Inc. All rights reserved.

9.9 Using Constructors and Destructors in Derived Classes

- Base class initializer
 - Uses member-initializer syntax
 - Can be provided in the derived class constructor to call the base-class constructor explicitly
 - Otherwise base class's default constructor called implicitly
 - Base-class constructors and base-class assignment operators are not inherited by derived classes
 - Derived-class constructors and assignment operators, however, can call base-class constructors and assignment operators

© 2000 Prentice Hall, Inc. All rights reserved.

9.9 Using Constructors and Destructors in Derived Classes

- A derived-class constructor
 - Calls the constructor for its base class first to initialize its base-class members
 - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor
- Destructors are called in the reverse order of constructor calls
 - So a derived-class destructor is called before its base-class destructor
- Destructor of Class Employee frees up the dynamically assigned arrays [] firstname and [] lastname, avoiding memory leaks (lines 48, 49).

© 2000 Prentice Hall, Inc. All rights reserved.

```

1 // Fig. 9.7: point2.h
2 // Definition of class Point
3 #ifndef POINT2_H
4 #define POINT2_H
5
6 class Point {
7 public:
8     Point( int = 0, int = 0 ); // default constructor
9     ~Point(); // destructor
10 protected: // accessible by derived classes
11     int x, y; // x and y coordinates of Point
12 };
13
14 #endif
15 // Fig. 9.7: point2.cpp
16 // Member function definitions for class Point
17 #include <iostream>
18
19 using std::cout;
20 using std::endl;
21
22 #include "point2.h"
23
24 // Constructor for class Point
25 Point::Point( int a, int b )
26 {
27     x = a;
28     y = b;
29
30     cout << "Point constructor: "
31         << '[' << x << ", " << y << ']' << endl;
32 }

```

25
Outline
 1. Point definition

 1. Load header

 1.1 Function definitions

```

33
34 // Destructor for class Point
35 Point::~Point()
36 {
37     cout << "Point destructor: "
38         << '[' << x << ", " << y << ']' << endl;
39 }
40 // Fig. 9.7: circle2.h
41 // Definition of class Circle
42 #ifndef CIRCLE2_H
43 #define CIRCLE2_H
44
45 #include "point2.h"
46
47 class Circle : public Point {
48 public:
49     // default constructor
50     Circle( double r = 0.0, int x = 0, int y = 0 );
51
52     ~Circle();
53 private:
54     double radius;
55 };
56
57 #endif

```

26
Outline
 1.1 Function definitions

 1. Load header

 1.1 Circle Definition

Circle inherits from Point.

```

58 // Fig. 9.7: circle2.cpp
59 // Member function definitions for class Circle
60 #include <iostream>
61
62 using std::cout;
63 using std::endl;
64
65 #include "circle2.h"
66
67 // Constructor for Circle calls constructor for Point
68 Circle::Circle( double r, int a, int b )
69 : Point( a, b ) // call base-class constructor
70 {
71     radius = r; // should validate
72     cout << "Circle constructor: radius is "
73         << radius << " [" << x << ", " << y << ']' << endl;
74 }
75
76 // Destructor for class Circle
77 Circle::~Circle()
78 {
79     cout << "Circle destructor: radius is "
80         << radius << " [" << x << ", " << y << ']' << endl;
81 }

```

27
Outline
 1. Load header

 1.1 Function Definitions

Constructor for Circle calls constructor for Point. first. Uses member-initializer syntax.

Destructor for Circle calls destructor for Point.

```

82 // Fig. 9.7: fig09_07.cpp
83 // Demonstrate when base-class and derived-class
84 // constructors and destructors are called.
85 #include <iostream>
86
87 using std::cout;
88 using std::endl;
89
90 #include "point2.h"
91 #include "circle2.h"
92
93 int main()
94 {
95     // Show constructor and destructor calls
96     {
97         Point p( 11, 22 );
98     }
99     cout << endl;
100     Circle circle1( 4.5, 72, 29 );
101     cout << endl;
102     Circle circle2( 10, 5, 5 );
103     cout << endl;
104     return 0;
105 }

```

28
Outline
 1. Load headers

 1.1 Initialize objects

 Objects enter and leave scope

Object created inside a block destroyed when it leaves scope.

Remember that the Point constructor is called for Point objects before the Circle objects before the Circle destructor: radius is 10 [5, 5]
 Point destructor: [5, 5]
 Circle destructor: radius is 4.5 [72, 29]
 Point destructor: [72, 29]

```

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]

```

29
Outline
 Program Output

9.10 Implicit Derived-Class Object to Base-Class Object Conversion

- Assignment of derived and base classes
 - Derived-class type and base-class type are different
 - Derived-class object can be treated as a base-class object
 - Derived class has members corresponding to all of the base class's members
 - Derived-class has more members than the base-class object
 - Base-class can be assigned a derived-class
 - Base-class object cannot be treated as a derived-class object
 - Would leave additional derived class members undefined
 - Derived-class cannot be assigned a base-class
 - Assignment operator can be overloaded to allow such an assignment

9.10 Implicit Derived-Class Object to Base-Class Object Conversion

- Mixing base and derived class pointers and objects
 - Referring to a base-class object with a base-class pointer
 - Allowed (straightforward)
 - Referring to a derived-class object with a derived-class pointer
 - Allowed (straightforward)
 - Referring to a derived-class object with a base-class pointer
 - Possible syntax error
 - Code can only refer to base-class members, or syntax error
 - Referring to a base-class object with a derived-class pointer
 - Syntax error
 - The derived-class pointer must first be cast to a base-class pointer
- Need way to resolve base-class Vs derived-class routines using base-class pointers (Virtual functions)

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

31

9.11 Software Engineering With Inheritance

- Classes are often closely related
 - “Factor out” common attributes and behaviors and place these in a base class
 - Use inheritance to form derived classes
- Modifications to a base class
 - Derived classes do not change as long as the **public** and **protected** interfaces are the same
 - Derived classes may need to be recompiled
- Use **Inheritance sparingly**, often times complexity is introduced needlessly. Can make for bad engineering thus hard to understand systems.

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

32

9.12 Composition vs. Inheritance

- “Is a” relationships
 - Inheritance
 - Relationship in which a class is derived from another class
- “Has a” relationships
 - Composition
 - Relationship in which a class contains other classes as members
- Has a**, is a composition. **Is a kind of**, is inheritance.
- Interchangeable! Careful.

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

33

9.13 “Uses A” And “Knows A” Relationships

- “Uses a”
 - One object issues a function call to a member function of another object. Limited!
- “Knows a”
 - One object is aware of another
 - Contains a pointer or handle to another object
 - Has access to all public stuff.
 - Also called an association

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

34

9.14 Case Study: Point, Circle, Cylinder

- Point, circle, cylinder hierarchy
 - Point** class is base class
 - Circle** class is derived from **Point** class
 - Cylinder** class is derived from **Circle** class

© 2000 Prentice Hall, Inc. All rights reserved. ■ ■

35

```

1 // Fig. 9.8: point2.h
2 // Definition of class Point
3 #ifndef POINT2_H
4 #define POINT2_H
5
6 #include <iostream>
7
8 using std::ostream;
9
10 class Point {
11     friend ostream operator<<( ostream &, const Point & );
12 public:
13     Point( int = 0, int = 0 ); // default constructor
14     void setPoint( int, int ); // set coordinates
15     int getX() const { return x; } // get x coordinate
16     int getY() const { return y; } // get y coordinate
17 protected: // accessible to derived classes
18     int x, y; // coordinates of the point
19 };
20
21 #endif
22 // Fig. 9.8: point2.cpp
23 // Member functions for class Point
24 #include "point2.h"
25
26 // Constructor for class Point
27 Point::Point( int a, int b ) { setPoint( a, b ); }
28
29 // Set the x and y coordinates
30 void Point::setPoint( int a, int b )
31 {
32     x = a;
    
```

Outline

- 1. Point definition
- 1.1 Function definitions

Point data members are protected to be made accessible by Circle

36

```

33 y = b;
34 }
35
36 // Output the Point
37 ostream operator<<( ostream &output, const Point &p )
38 {
39     output << '[' << p.x << ", " << p.y << ']' ;
40
41     return output; // enables cascading
42 }

```

Outline 37
1.1 Function definitions

```

1 // Fig. 9.9: circle2.h
2 // Definition of class Circle
3 #ifndef CIRCLE2_H
4 #define CIRCLE2_H
5
6 #include <iostream>
7
8 using std::ostream;
9
10 #include "point2.h"
11
12 class Circle : public Point {
13     friend ostream operator<<( ostream &, const Circle & );
14 public:
15     // default constructor
16     Circle( double r = 0.0, int x = 0, int y = 0 );
17     void setRadius( double ); // set radius
18     double getRadius() const; // return radius
19     double area() const; // calculate area
20 protected: // accessible to derived classes
21     double radius; // radius of the Circle
22 };
23
24 #endif
25 // Fig. 9.9: circle2.cpp
26 // Member function definitions for class Circle
27 #include <iomanip>
28
29 using std::ios;
30 using std::setiosflags;
31 using std::setprecision;
32
33 #include "circle2.h"

```

Outline 38
1. circle definition
1.1 Function definitions

Circle data members are protected to be made accessible by Cylinder.

```

34
35 // Constructor for Circle calls constructor for Point
36 // with a member initializer and initializes radius
37 Circle( double r, int a, int b )
38 : Point( a, b ) // call base-class constructor
39 { setRadius( r ); }
40
41 // Set radius
42 void Circle::setRadius( double r )
43 { radius = ( r >= 0 ? r : 0 ); }
44
45 // Get radius
46 double Circle::getRadius() const { return radius; }
47
48 // Calculate area of Circle
49 double Circle::area() const
50 { return 3.14159 * radius * radius; }
51
52 // Output a circle in the form:
53 // Center = [x, y]; Radius = ###
54 ostream operator<<( ostream &output, const Circle &c )
55 {
56     output << "Center = " << static_cast< Point >( c )
57           << "; Radius = "
58           << setiosflags( ios::fixed | ios::showpoint )
59           << setprecision( 2 ) << c.radius;
60
61     return output; // enables cascaded calls
62 }

```

Outline 39
1.1 Function definitions

```

1 // Fig. 9.10: cylindr2.h
2 // Definition of class Cylinder
3 #ifndef CYLINDER2_H
4 #define CYLINDER2_H
5
6 #include <iostream>
7
8 using std::ostream;
9
10 #include "circle2.h"
11
12 class Cylinder : public Circle {
13     friend ostream operator<<( ostream &, const Cylinder & );
14
15 public:
16     // default constructor
17     Cylinder( double h = 0.0, double r = 0.0,
18             int x = 0, int y = 0 );
19
20     void setHeight( double ); // set height
21     double getHeight() const; // return height
22     double area() const; // calculate and return area
23     double volume() const; // calculate and return volume
24
25 protected:
26     double height; // height of the Cylinder
27 };
28
29 #endif

```

Outline 40
1. Cylinder definition

```

30 // Fig. 9.10: cylindr2.cpp
31 // Member and friend function definitions
32 // for class Cylinder.
33 #include "cylindr2.h"
34
35 // Cylinder constructor calls Circle constructor
36 Cylinder( double h, double r, int x, int y )
37 : Circle( r, x, y ) // call base-class constructor
38 { setHeight( h ); }
39
40 // Set height of Cylinder
41 void Cylinder::setHeight( double h )
42 { height = ( h >= 0 ? h : 0 ); }
43
44 // Get height of Cylinder
45 double Cylinder::getHeight() const { return height; }
46
47 // Calculate area of Cylinder (i.e., surface area)
48 double Cylinder::area() const
49 {
50     return 2 * Circle::area() +
51            2 * 3.14159 * radius * height;
52 }
53
54 // Calculate volume of Cylinder
55 double Cylinder::volume() const
56 { return Circle::area() * height; }
57
58 // Output Cylinder dimensions
59 ostream operator<<( ostream &output, const Cylinder &c )
60 {

```

Outline 41
1.1 Function definitions

```

61     output << static_cast< Circle >( c )
62           << "; Height = " << c.height;
63
64     return output; // enables cascaded calls
65 }
66 // Fig. 9.10: fig09_10.cpp
67 // Driver for class Cylinder
68 #include <iostream>
69
70 using std::cout;
71 using std::endl;
72
73 #include "point2.h"
74 #include "circle2.h"
75 #include "cylindr2.h"
76
77 int main()
78 {
79     // creates Cylinder object
80     Cylinder cyl( 5.7, 2.5, 12, 23 );
81
82     // use get functions to display the Cylinder
83     cout << "X coordinate is " << cyl.getX()
84           << "\nY coordinate is " << cyl.getY()
85           << "\nRadius is " << cyl.getRadius()
86           << "\nHeight is " << cyl.getHeight() << "\n\n";
87
88     // use set functions to change the Cylinder's attributes
89     cyl.setHeight( 10 );
90     cyl.setRadius( 4.25 );
91     cyl.setPoint( 2, 2 );

```

Outline 42
1.1 Function definitions
1. Load headers
1.1 Initialize object
2. Function calls
2.1 Change attributes
3. Output data

Circle::area() is overridden.

X coordinate is 12
Y coordinate is 23
radius is 2.5
Height is 5.7

```

92 cout << "The new location, radius, and height of cyl are:\n"
93 << cyl << '\n';
94
95 cout << "The area of cyl is:\n"
96 << cyl.area() << '\n';
97
98 // display the Cylinder as a Point
99 Point pRef = cyl; // pRef "thinks"
100 cout << "\nCylinder printed as a Point is: "
101 << pRef << "\n";
102
103 // display the Cylinder as a Circle
104 Circle cRef = cyl; // cRef "thinks"
105 cout << "\nCylinder printed as a Circle is:\n"
106 << "Area: " << cRef.area()
107 << '\n';
108 return 0;
109 }

```

The new location, radius, and height of cyl are:
Center = [2, 2]; Radius = 4.25; Height = 10.00
The area of cyl is:
380.53
Cylinder printed as a Point is: [2, 2]
Cylinder printed as a Circle is:
Center = [2, 2]; Radius = 4.25
Area: 56.74

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
The area of cyl is:
380.53
Cylinder printed as a Point is: [2, 2]
Cylinder printed as a Circle is:
Center = [2, 2]; Radius = 4.25
Area: 56.74

9.15 Multiple Inheritance

- Multiple Inheritance
 - Derived-class inherits from multiple base-classes
 - Encourages software reuse, but can create ambiguities

© 2000 Prentice Hall, Inc. All rights reserved.

```

1 // Fig. 9.11: base1.h
2 // Definition of class Base1
3 #ifndef BASE1_H
4 #define BASE1_H
5
6 class Base1 {
7 public:
8     Base1( int x ) { value = x; }
9     int getData() const { return value; }
10 protected: // accessible to derived classes
11     int value; // inherited by derived class
12 };
13
14 #endif
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     Base2( char c ) { letter = c; }
23     char getData() const { return letter; }
24 protected: // accessible to derived classes
25     char letter; // inherited by derived class
26 };
27
28 #endif

```

1. Base1 definition

1. Base2 definition

```

29 // Fig. 9.11: derived.h
30 // Definition of class Derived which inherits
31 // multiple base classes (Base1 and Base2).
32 #ifndef DERIVED_H
33 #define DERIVED_H
34
35 #include <iostream>
36
37 using std::ostream;
38
39 #include "base1.h"
40 #include "base2.h"
41
42 // multiple inheritance
43 class Derived : public Base1, public Base2 {
44     friend ostream operator<<( ostream &, const Derived & );
45
46 public:
47     Derived( int, char, double );
48     double getReal() const;
49
50 private:
51     double real; // derived class's private data
52 };
53
54 #endif

```

1. Derived Definition

Derived inherits from Base1 and Base2.

```

55 // Fig. 9.11: derived.cpp
56 // Member function definitions for class Derived
57 #include "derived.h"
58
59 // Constructor for Derived calls constructors for
60 // class Base1 and class Base2.
61 // Use member initializers to call base-class constructors
62 Derived::Derived( int i, char c, double f )
63     : Base1( i ), Base2( c ), real( f ) {}
64
65 // Return the value of real
66 double Derived::getReal() const { return real; }
67
68 // Display all the data members of Derived
69 ostream operator<<( ostream &output, const Derived &d )
70 {
71     output << "Integer: " << d.value
72     << "\n Character: " << d.letter
73     << "\nReal number: " << d.real;
74     return output; // enables cascaded calls
75 }
76
77 // Fig. 9.11: fig09_11.cpp
78 // Driver for multiple inheritance example
79 #include <iostream>
80
81 using std::cout;
82 using std::endl;
83
84 #include "base1.h"
85 #include "base2.h"

```

1. Load header

1.1 Function Definitions

```

86 #include "derived.h"
87
88 int main()
89 {
90     Base1 b1( 10 ), *base1Ptr = 0; // create Base1 object
91     Base2 b2( 'Z' ), *base2Ptr = 0; // create Base2 object
92     Derived d( 7, 'A', 3.5 ); // create Derived object
93
94     // print data members of base class objects
95     cout << "Object b1 contains integer " << b1.getData()
96     << "\nObject b2 contains character " << b2.getData()
97     << "\nObject d contains:\n" << d << "\n";
98
99     // print data members of derived class object
100     // scope resolution operator resolves getData()
101     cout << "Data members of Derived can be"
102     << " accessed individually:\n";
103     cout << "Integer: " << d.Base1::getData()
104     << "\n Character: " << d.Base2::getData()
105     << "\nReal number: " << d.getReal() << "\n";
106
107     cout << "Derived can be treated as a"
108     << " object of either Base1 or Base2.\n";
109
110     // treat Derived as a Base1 object
111     base1Ptr = &d;
112     cout << "base1Ptr->getData() yields "
113     << base1Ptr->getData() << "\n";
114
115     // treat Derived as a Base2 object
116     base2Ptr = &d;

```

1.1 Create objects

2. Function calls

Data members of Derived can be accessed individually:
Integer: 7
Character: A
Real number: 3.5
get data Base1 object.
Derived can be treated as an object of either base class:
base1Ptr->getData() yields 7
base2Ptr->getData() yields 7


```

117 cout << "basePtr->getData() yields "
118     << basePtr->getData() << endl;
119
120 return 0;
121}

```

```

basePtr->getData() yields A

```

Outline

49

3. Output data

```

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:
basePtr->getData() yields 7
base2Ptr->getData() yields A

```

Program Output

© 2000 Prentice Hall, Inc. All rights reserved.

© 2000 Prentice Hall, Inc. All rights reserved.

Graded Exercises

50

- Read the summary of Ch 9 (pp 618..622)
- Do Self-Review exercises Ch 9 Deitel & Deitel
- Do following Exercises
 - 9.2 (make a diagram like on Fig 9.2, put in folder),
 - 9.12 (on paper & put in folder)