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

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

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

<table>
<thead>
<tr>
<th>Base Class</th>
<th>Derived Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>IndependentStudent</td>
</tr>
<tr>
<td>Shape</td>
<td>Triangle</td>
</tr>
<tr>
<td>Loan</td>
<td>HomeImprovementLoan</td>
</tr>
<tr>
<td>Employee</td>
<td>FacultyMember</td>
</tr>
<tr>
<td>Account</td>
<td>SavingsAccount</td>
</tr>
</tbody>
</table>

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
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;

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
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

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
  - A 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
9.7 public, private, and protected inheritance

<table>
<thead>
<tr>
<th>Base class</th>
<th>public accessibility</th>
<th>Type of inheritance</th>
<th>protected accessibility</th>
<th>private accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>public method scope.</td>
<td>public member functions, friend functions, and non-member functions.</td>
<td>protected in derived class.</td>
<td>protected in derived class.</td>
</tr>
<tr>
<td>Protected</td>
<td>can be accessed directly by the derived class.</td>
<td>can be accessed directly by all non-static member functions and friend functions.</td>
<td>can be accessed directly by all non-static member functions and friend functions.</td>
<td>can be accessed directly by all non-static member functions and friend functions.</td>
</tr>
<tr>
<td>Private</td>
<td>can be accessed directly by all non-static member functions and friend functions of the base class.</td>
<td>can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.</td>
<td>can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.</td>
<td>can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.</td>
</tr>
</tbody>
</table>

9.8 Direct and Indirect Base Classes

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

Employee is an indirect base class of MinuteWorker

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

- 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
  - Constructors 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).
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 object
    - 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 object
  - 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)

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.

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.

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

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
1.1 Function definitions

1.1.4 Function calls

Outline
9.15 Multiple Inheritance

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

Graded Exercises

- 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)