Chapter 15: Polymorphism and Virtual Functions

Starting Out with C++
Early Objects
Global Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Addison-Wesley
is an imprint of
PEARSON

Topics

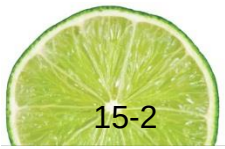
15.1 Type Compatibility in Inheritance Hierarchies

15.2 Polymorphism and Virtual Member Functions

15.3 Abstract Base Classes and Pure Virtual Functions

15.4 Composition Versus Inheritance

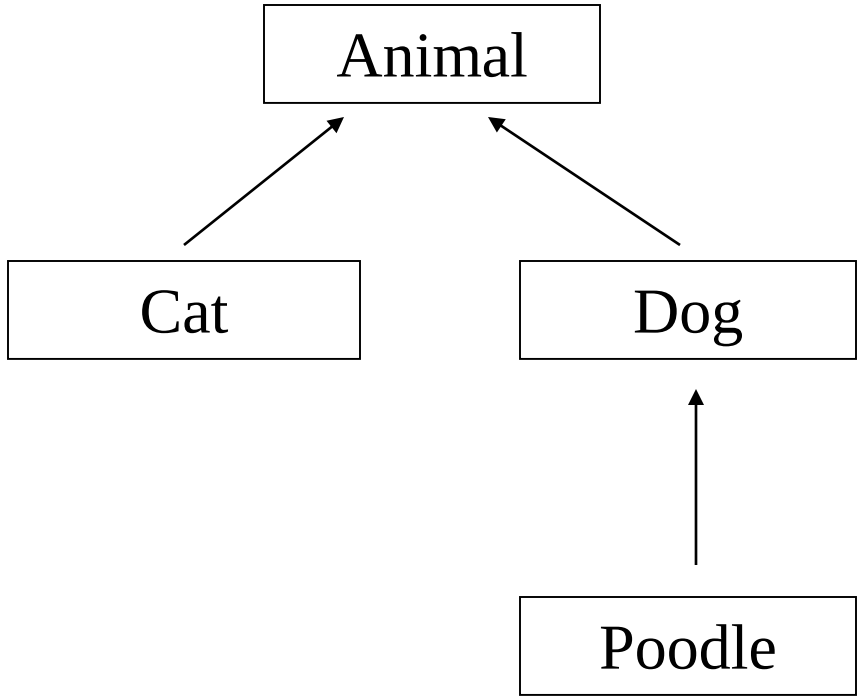
15.5 Secure Encryption System, Inc., Case Study



15.1 Type Compatibility in Inheritance Hierarchies

- Classes in a program may be part of an inheritance hierarchy

- Classes lower in the hierarchy are special cases of those above



Type Compatibility in Inheritance

- A pointer to a derived class can be assigned to a pointer to a base class. Another way to say this is:
- A base class pointer can point to derived class objects

```
Animal *pA = new Cat;
```

Type Compatibility in Inheritance

- Assigning a base class pointer to a derived class pointer requires a cast

```
Animal *pA = new Cat;
Cat *pC;
pC = static_cast<Cat *>(pA);
```

- The base class pointer must already point to a derived class object for this to work

P 984 (Figure 15-1) + P985 Inheritance4.h eP934~937

Using Type Casts with Base Class Pointers

- C++ uses the declared type of a pointer to determine access to the members of the pointed-to object

- If an object of a derived class is pointed to by a base class pointer, all members of the derived class may not be accessible

Using Type Casts with Base Class Pointers

- Type cast the base class pointer to the derived class (via **`static_cast`**) in order to access members that are specific to the derived class

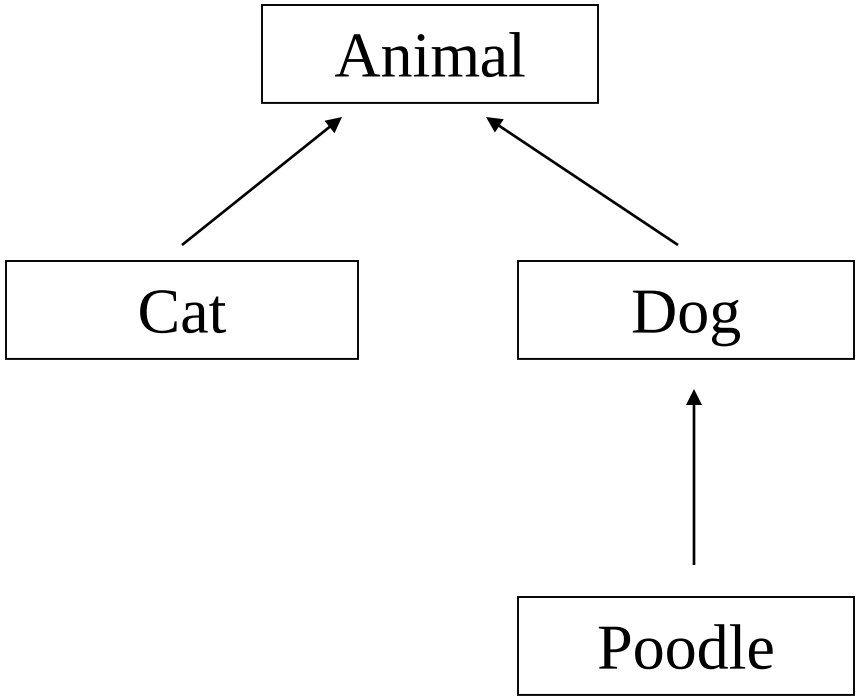
P 988 Program 15-1, eP938~939

15.2 Polymorphism and Virtual Member Functions

- Polymorphic code: Code that behaves differently when it acts on objects of different types
- Virtual Member Function: The C++ mechanism for achieving polymorphism

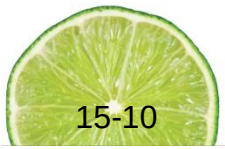
Polymorphism

Consider the Animal, Cat, Dog hierarchy where each class has its own version of the member function id( )



Polymorphism

```
class Animal{
 public: void id(){cout << "animal";}
}
class Cat : public Animal{
 public: void id(){cout << "cat";}
}
class Dog : public Animal{
 public: void id(){cout << "dog";}
}
```



Polymorphism

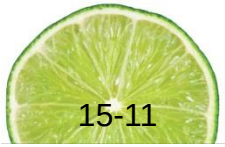
- Consider the collection of different Animal objects

```
Animal *pA[] = {new Animal, new Dog,
                new Cat};
```

and accompanying code

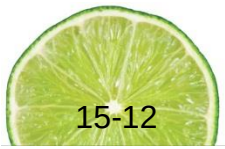
```
for(int k=0; k<3; k++)
    pA[k]->id();
```

- Prints: **`animal animal animal`**, ignoring the more specific versions of **`id()`** in **`Dog`** and **`Cat`**



Polymorphism

- The preceding code is not polymorphic: it behaves the same way even though **`Animal`**, **`Dog`** and **`Cat`** have different types and different **`id()`** member functions
- Polymorphic code would have printed "**`animal dog cat`**" instead of "**`animal animal animal`**"



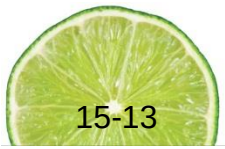
Polymorphism

- The code is not polymorphic because in the expression

  `pA[k]->id()`

  the compiler sees only the type of the pointer **`pA[k]`**, which is pointer to **`Animal`**
- Compiler does not see type of actual object pointed to, which may be **`Animal`**, or **`Dog`**, or **`Cat`**

P989 Program 15-2, eP939~940



Virtual Functions

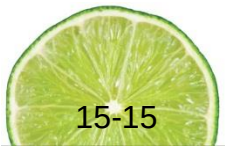
Declaring a function **`virtual`** will make the compiler check the type of each object to see if it defines a more specific version of the virtual function

Virtual Functions

If the member functions **`id()`** are declared virtual, then the code

```
Animal *pA[] = {new Animal,
                new Dog,new Cat};
for(int k=0; k<3; k++)
    pA[k]->id();
```

will print **`animal dog cat`**

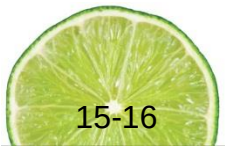


Virtual Functions

How to declare a member function virtual:

```
class Animal{
 public: virtual void id(){cout << "animal";}
}
class Cat : public Animal{
 public: virtual void id(){cout << "cat";}
}
class Dog : public Animal{
 public: virtual void id(){cout << "dog";}
}
```

P992 Program 15-3, eP941~943



Function Binding

- In **`pA[k]->id()`**, Compiler must choose which version of **`id()`** to use: There are different versions in the **`Animal`**, **`Dog`**, and **`Cat`** classes
- Function binding is the process of determining which function definition to use for a particular function call
- The alternatives are *static* and *dynamic* binding

Static Binding

- Static binding chooses the function in the class of the base class pointer, ignoring any versions in the class of the object actually pointed to
- Static binding is done at compile time

Dynamic Binding

- Dynamic Binding determines the function to be invoked at execution time
- Can look at the actual class of the object pointed to and choose the most specific version of the function
- Dynamic binding is used to bind virtual functions

15.3 Abstract Base Classes and Pure Virtual Functions

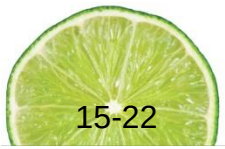
- An abstract class is a class that contains no objects that are not members of subclasses (derived classes)
- For example, in real life, Animal is an abstract class: there are no animals that are not dogs, or cats, or lions…

Abstract Base Classes and Pure Virtual Functions

- Abstract classes are an organizational tool. They are useful in organizing inheritance hierarchies
- Abstract classes can be used to specify an interface that must be implemented by all subclasses

Abstract Functions

- The member functions specified in an abstract class do not have to be implemented
- The implementation is left to the subclasses
- In C++, an abstract class is a class with at least one abstract member function

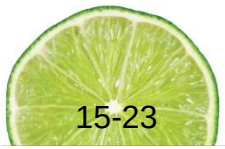


Pure Virtual Functions

- In C++, a member function of a class is declared to be an abstract function by making it virtual and replacing its body with `= 0;`

```
class Animal{
  public:
    virtual void id()=0;
};
```

- A virtual function with its body omitted and replaced with `=0` is called a pure virtual function, or an abstract function



Abstract Classes

- An abstract class can not be instantiated
- An abstract class can only be inherited from; that is, you can derive classes from it
- Classes derived from abstract classes must override all pure virtual functions with a concrete member functions before they can be instantiated.

P998 Program 15-6, eP945~946

15.4 Composition vs. Inheritance

- Inheritance models an 'is a' relation between classes. An object of a derived class 'is a(n)' object of the base class
- Example:
  - an **`UnderGrad`** is a **`Student`**
  - a **`Mammal`** is an **`Animal`**
  - a **`Poodle`** is a **`Dog`**

Composition vs. Inheritance

When defining a new class:

- Composition is appropriate when the new class needs to use an object of an existing class
- Composition models an ‘has a' relation between classes. An object of a composition class ‘has a(n)' object of another class

Composition vs. Inheritance

When defining a new class:

- Inheritance is appropriate when
  - objects of the new class are a subset of the objects of the existing class, or
  - objects of the new class will be used in the same ways as the objects of the existing class

P1005 Program 15-7, eP953~954

15.5 Secure Encryption System, Inc. : A Case Study

- A simple encryption/description framework
- Get char from input file → Encryption → Put the encrypted char into output file
- Class Encryption (P1008, eP956)

P1008 Program 15-8, eP956~958)

Homework 7

TEXT BOOK CH. 15 Programming Challenges

6. Removal of Line Breaks (P1020)

5. File Filter

A file filter reads an input file, transforms it in some way, and writes the results to an output file. Write an abstract file filter class that defines a pure virtual function for transforming a character. Create one subclass of your file filter class that performs encryption, another that transforms a file to all uppercase, and another that creates an unchanged copy of the original file.

The class should have a member function

```
void doFilter(ifstream &in, ofstream &out)
```

that is called to perform the actual filtering. The member function for transforming a single character should have the prototype

```
char transform(char ch)
```

The encryption class should have a constructor that takes an integer as an argument and uses it as the encryption key.

6. Removal of Line Breaks

Create a subclass of the abstract filter class of Programming Challenge 5 that replaces every line break in a file with a single space.