OBJECT ORIENTED PROGRAMMING USING C++

Polymorphism

- **Polymorphism** occurs with objects instantiated from a set of classes related by inheritance to the same ancestor class.
  - Each class provides its own implementation of the (virtual) functions.
  - Therefore, objects respond differently to the same set of messages originally declared in the ancestor.

R Gopalakrishna
Assistant Professor
DCE, Gurgaon

Polymorphism

- **Polymorphism** is most useful when there are a series of related classes that require the “same” behaviors.
  - The individual classes typically define their own specific behavior implementation.

Polymorphism

- Example #1:
  - A company where there is an abstract class Employee. Every employee must earn money but the way earnings are calculated may be different based upon the type of employee (manager, commission worker, hourly worker).

Polymorphism

- Example #2:
  - Most 2-dimensional shapes have a position, area, and perimeter. The specific shape subclasses then define the particular implementations of these functions.

Polymorphism

- Example #3:
  - All customers at a store are allowed to pay with cash.
  - Some customers have a charge account at the store, and therefore have an alternate way to pay.
  - When information on customers is printed, the ones with charge accounts should show their balance, etc.

Virtual Functions

- A C++ **virtual function** is a function that can be made polymorphic by redefinition.
- A non-virtual function
  - can still be redefined
  - causes the function that is called on an object to depend solely on the object's compile-time type
  - is therefore not polymorphic.

Declaring Virtual Functions

```
class Parent {
    virtual void poly();
    void strict();
};
class Child: public Parent {
    virtual void poly(); // redefinition
    void strict(); // redefinition
};
```

Calling Virtual Functions

```
Parent *p( new Parent() );
Parent *c( new Child() );
p->poly(); // calls Parent::poly()
p->strict(); // calls Parent::strict()
c->poly();  // calls Child::poly()
c->strict(); // calls Parent::strict() !
```

Constructors / Destructors

- By definition a constructor function **cannot** be defined as a virtual function.
  - See *Factory* design patterns for ways around this.
- A class that contains **virtual functions** should also contain a **virtual destructor.**

Abstract Classes

- An **abstract class** is a base class that will never have an object instantiated from it.
  - **Abstract classes** are used only for inheritance, they are not well-defined enough for instantiation.
  - **Abstract classes** provide a generic base class that can be used by concrete classes to provide an interface and/or implementation.

Abstract Classes

- An abstract class in C++ is defined as any class that contains at least one **pure virtual function**.

Pure Virtual Functions

- A **pure virtual function** associates an "initializer" with the virtual function declaration.

  virtual returnType functionName () const = 0;

- Derived classes may define an implementation for the pure virtual function.
  - If the derived class does **not** include an implementation for the pure virtual function, the derived class is also abstract.

Pure Virtual Functions

- Defining pure virtual functions in the abstract base class:
  - Abstract.h: class declaration

    virtual ret-type functionName(..,..,..) const = 0;
  - Abstract.C: class definition
    - There is no implementation in the *.C file.
      - (although no one will stop you!)

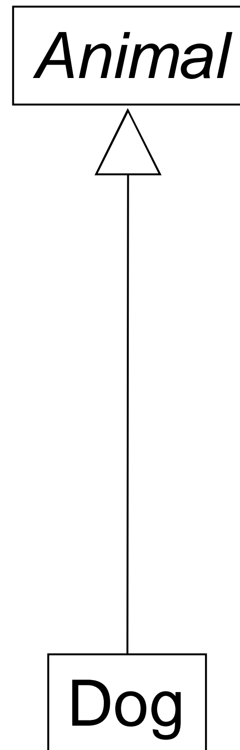
Pure Virtual Functions

- Implementing virtual functions in a concrete derived class:
  - Concrete.h

    virtual ret-type functionName(..,..,..) const;
  - Concrete.C

```
ret-type Concrete::functionName(..,..,..) const {
  // …
}
```

Pure Virtual Functions



Pure Virtual Functions

- Animal.h

```
virtual void speak() = 0;
```

- Animal.C
  - No implementation
- Dog.h

```
virtual void speak();
```

- Dog.C

```
void speak() {
   cout << “Arf!”;
}
```

Here we define a group of classes known collectively as animals. All animals can speak, but each one speaks differently. If we ask an animal to speak, it will make the appropriate response.

For The Curious: What is happening?

- Any time the C++ compiler encounters a class definition containing a virtual function, the compiler creates a virtual function table (vtable) for the class.
  - The vtable is used in the program to determine which function implementation is to be executed when the function is called.
  - It also helps realize RTTI (not used here)