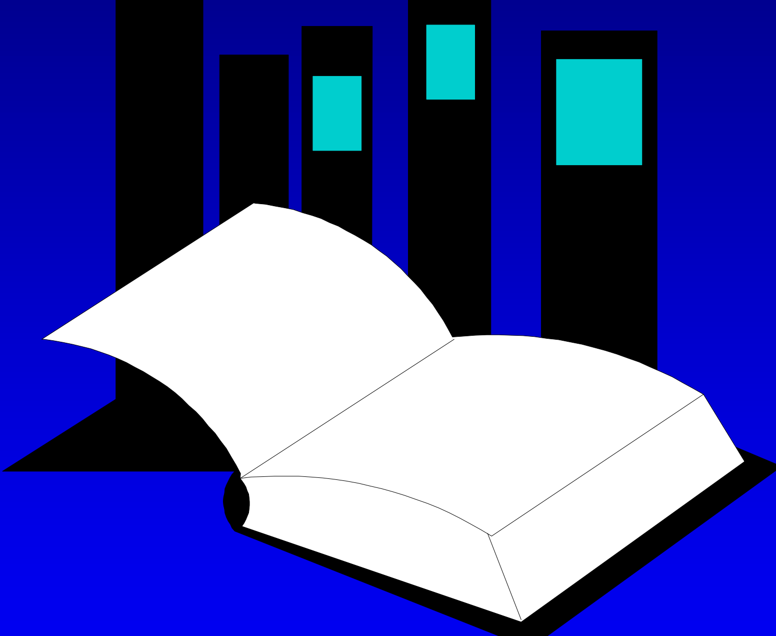
Inheritance



Data Structures and Other Objects Using C++

- Chapter 14 discuss Derived classes, Inheritance, and Polymorphism
- Inheritance Basics
- Inheritance Details
- Polymorphism
- Virtual Functions

Inheritance Basics

- Inheritance is the process by which a new class, called a derived class, is created from another class, called the base class
  - A derived class automatically has all the member variables and functions of the base class
  - A derived class can have additional member variables and/or member functions
  - The derived class is a child of the base or parent class

A Base Class – an example

- We will define a class called clock for all possible clocks
- The clock class will be used to define classes that keep track of the time

A clock Class

- A Clock class keeps track of a time value such as 9:48 pm

```
class Clock
    {
    public:
        // CONSTRUCTOR
        clock( );
        // MODIFICATION FUNCTIONS
        void set_time(int hour, int minute, bool morning);
        void advance(int minutes);
        // CONSTANT FUNCTIONS
        int get_hour( ) const;
        int get_minute( ) const;
        bool is_morning( ) const;
    private:
        int my_hour;
        int my_minute;
        int my_morning;
    };
```

A derived class

- A derived class inherits all the members of the parent class
  - The derived class does not re-declare or re-define members inherited from the parent, except…
    - The derived class re-declares and re-defines member functions of the parent class that will have a different definition in the derived class
    - The derived class can add member variables and functions

Examples of derived classes

```
class Cuckoo_Clock : public Clock
// a cuckoo_clock
    {
    public:
        bool is_cuckooing( ) const;
    };


class Clock24 : public Clock
// a military clock
    {
    public:
        int get_hour( ) const;
    };
```

Class cuckoo_clock

- `Cuckoo_Clock` is derived from class `Clock`
  - `Cuckoo_Clock` inherits all member functions and member variables of `Clock`
  - The class definition begins

```
class Cuckoo_Clock : public Clock
```

    - :public clock shows that Cuckoo_Clock is derived from class Clock
  - A derived class can declare, in necessary, additional member variables

Implementing a Derived Class

- Any member functions added in the derived class are defined in the implementation file for the derived class
  - Definitions are not given for inherited functions that are not to be changed

```
bool cuckoo_clock::is_cuckooing( ) const
    {
  return (get_minute( ) == 0);
    }
```

Parent and Child Classes

- Recall that a child class automatically has all the members of the parent class
- The parent class is an ancestor of the child class
- The child class is a descendent of the parent class
- The parent class (`Clock`) contains all the code common to the child classes
  - You do not have to re-write the code for each child

Derived Class Types

- A cuckoo clock is a clock
  - In C++, an object of type Cuckoo_Clock can be used where an object of type Clock can be used
  - An object of a class type can be used wherever any of its ancestors can be used
  - An ancestor **cannot** be used wherever one of its descendents can be used

Default Initialization

- If a derived class does not declare any constructor for the derived class, C++ provides automatically the default constructor:

- Steps performed by C++:
  1. Activate the default constructor for the base class to initialize the member variable of the base class
  2. Activate the default constructor for any new member variable of the derived class.

Example

- If class B is derived from class A and class C is derived from class B
  - When a object of class C is created
    - The base class A's constructor is the first invoked
    - Class B's constructor is invoked next
    - C's constructor completes execution

Assignment Operator

- If a derived class does not define its own assignment operator C++ will automatically provide one.

- Steps for the application of the assignment operator:
  1. Activate the assignment operator for the base class
  2. Activate the assignment operator for any new member variable that is in the derived class but not in the base class.

Destructor

- If a derived class does not have a declared destructor C++ will automatically provide one.

- Steps for the application of the destructor:
  1. The destructor will be called for any member variable of the derived class not in the base class
  2. The destructor is called for the base class

Example of Destruction Sequence

- If class B is derived from class A and class C is derived from class B…
  - When the destructor of an object of class C goes out of scope
    - The destructor of class C is called
    - Then the destructor of class B
    - Then the destructor of class A
  - Notice that destructors are called in the reverse order of constructor calls

Derived Class Constructors

- A derived class can define its own constructor
  - The base class constructor can be invoked by the constructor of the derived class, if required.

```
class Animal : public Organism
    {
    public:
    // CONSTRUCTOR
    Animal(double init_size = 1, double init_rate = 0,
            double init_need = 0);
    }
```

Member Functions Overriding

- When defining a derived class, only list the inherited functions that you wish to change for the derived class (override)

```
int Clock24::get_hour( ) const
    {
        int ordinary_hour;
        ordinary_hour = clock::get_hour( );
        if (is_morning( ))
        {
            if (ordinary_hour == 12)
                return 0;
            else
                return ordinary_hour;
        }
        else
        {
            if (ordinary_hour == 12)
                return 12;
            else
                return ordinary_hour + 12;
        }
    }
```

```
int Clock::get_hour( ) const
    {
        return my_hour;
    }
```

Redefining or Overloading?

- A function redefined in a derived class has the same number and type of parameters
  - The derived class has only one function with the same name as the base class
- An overloaded function has a different number and/or type of parameters than the base class
  - The derived class has two functions with the same name as the base class
    - One is defined in the base class, one in the derived class

Access to a Redefined Base Function

- When a base class function is redefined in a derived class, the base class function can still be used
  - To specify that you want to use the base class version of the redefined function:

```
Clock24 tick;
tick.Clock::get_hour( );
```

Polymorphism

- Polymorphism refers to the ability to associate multiple meanings with one function name using a mechanism called late binding
- Polymorphism is a key component of the philosophy of object oriented programming

A Late Binding Example

- Imagine a graphics program with several types of figures
  - Each figure may be an object of a different class, such as a circle, oval, rectangle, etc.
  - Each shape is a descendant of a class Figure
  - Each shape has a function draw( ) implemented with code specific to each shape
  - Class Figure has functions common to all figures

A Problem

- Suppose that class Figure has a function center()
  - Function center() moves a figure to the center of the screen by erasing the figure and redrawing it in the center of the screen
  - Function center() is inherited by each of the derived classes
    - Function center() uses each derived object's draw function to draw the figure
    - The Figure class does not know about its derived classes, so it cannot know how to draw each figure

Virtual Functions

- Because the Figure class includes a method to draw figures, but the Figure class cannot know how to draw the figures, virtual functions are used
- Making a function virtual tells the compiler that you don't know how the function is implemented and to wait until the function is used in a program, then get the implementation from the object.
  - This is called late binding

Benefits of a well-written Base class

- With a well written base class, a programmer can write derived classes without worrying about how the base class accomplishes its work.
- When to require virtual member function?
  - When one member function activates another member function and the programmer anticipates that the other member function will be overridden in the future.

Private and Protected

- A private member variable or function in the parent class is not accessible to the child class
  - The public parent class member functions must be used to access the private members of the parent
- protected members of a class appear to be private outside the class, but are accessible by derived classes
  - Protected members are not necessary but are a convenience to facilitate writing the code of derived classes

Pure Virtual Functions and Abstract Classes

- A pure virtual function is indicated by =0 before the end of the semicolon in the prototype.
  - The class does not provide any implementation of a pure virtual function.
  - Because there is no implementation, any class with a pure virtual function is called an abstract class and no instances of an abstract class may appear in a program
- Abstract classes are used as base classes, and it is up to the derived class to provide the implementation for each pure virtual function

Virtual Destructors

- Whenever a class has virtual methods it is a good idea to also provide a destructor, even if the base class has no need for a destructor.
- With a virtual destructor the compiler arranges for the right destructor to be called at run time, even when the compiler is uncertain about the exact data type of the object.
  - Ex. virtual ~game()

Summary

- A derived class inherits all the members of the parent class
- A derived class can add new members and/or override existing members of the parent class
- Polymorphism refers to the ability to associate multiple meanings with one function name using a mechanism called late binding
- Late binding is achieved by declaring a function as virtual
  - A virtual function tells the compiler that you don't know how the implementation of the function and that you need to wait until the function is used in a program to get the implementation from the object.

Chapter 14 -- End