Chapter 15 – Inheritance, Polymorphism, and Virtual Functions

15.1 What is Inheritance?

- Inheritance allows a new class to be based on an existing class. The new class inherits all the member variables and functions of the class it is based on.

Figure 15-1

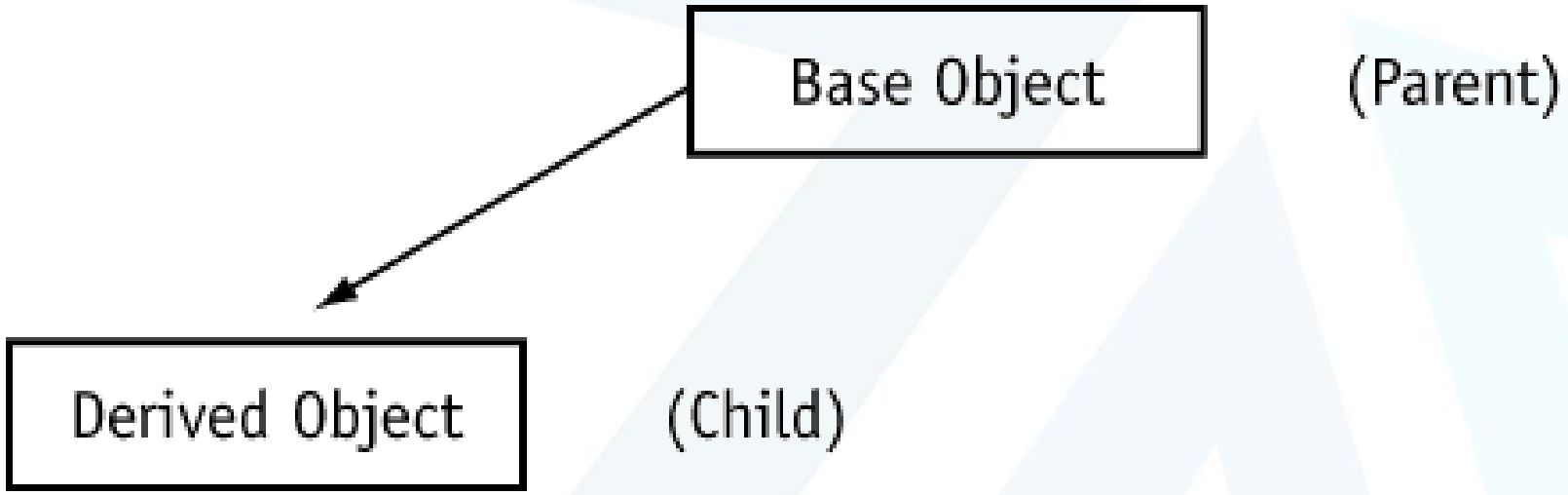


Figure 15-2

The individual class declarations contain their own members:

Grade Class

```
Private Members:
    char Letter;
    float Score;
    void CalcGrade(void);
Public Members:
    void SetScore(float);
    float GetScore(void);
    char GetLetter(void);
```

Test Class

```
Private Members:
    int NumQuestions;
    float PointsEach;
    int NumMissed;

Public Members:
    Test (int, int);
```

When the `Test` class is derived from the `Grade` class,
Objects of the `Test` class apppear to have the following members:

Test Class

```
Private Members:
    int NumQuestions;
    float PointsEach;
    int NumMissed;
Public Members:
    Test (int, int);
    void SetScore(float);
    float GetScore(void);
    char GetLetter(void);
```

Program 15-1

Contents of grade.h

```
#ifndef GRADE_H
#define GRADE_H
// Grade class declaration
class Grade
{
 private:
     char letter;
     float score;
     void calcGrade(void);
 public:
     void setScore(float s) { score = s; calcGrade();}
     float getScore(void) {return score; }
     char getLetter(void) { return letter; }
};
#endif
```

Program continues

Contents of grade.cpp

```
#include "grade.h"
// Definition of member function Grade::calcGrade
void Grade::calcGrade()
{
 if (score > 89)
     letter = 'A';
 else if (score > 79)
     letter = 'B';
 else if (score > 69)
     letter = 'C';
 else if (score > 59)
     letter = 'D';
 else
     letter = 'F';
}
```

Program continues

Contents of test.h

```
#ifndef TEST_H
#define TEST_H
#include "grade.h"  // Must include Grade class declaration.
// Test class declaration
class Test : public Grade
{
 private:
     int numQuestions;
     float pointsEach;
     int numMissed;
 public:
     Test(int, int);
};
#endif
```

Program continues

Contents of test.cpp

```
#include "test.h"

// Definition of Test class constructor
// Parameters: q = Number of questions, m = number of
// questions missed.
Test::Test(int q, int m)
{
 float numericGrade;
 numQuestions = q;
 numMissed = m;
 pointsEach = 100.0 / numQuestions;
 numericGrade = 100.0 - (numMissed * pointsEach);
 setScore(numericGrade);
}
```

Program continues

Contents of the main program pr15-1.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include "test.h"

void main(void)
{
  int questions, missed;

  cout << "How many questions are on the test? ";
  cin >> questions;
  cout << "How many questions did the student miss? ";
  cin >> missed;
  Test exam(questions, missed);
  cout.precision(2);
  cout << "The score is " << exam.getScore() << endl;
  cout << "The grade is " << exam.getLetter() << endl;
}
```

Program Output with Example Input

How many questions are on the test? **20** [**Enter**]

How many questions did the student miss? **3** [**Enter**]

The score is 85

The grade is B

15.2 Protected Members and Class Access

- Protected members of a base class are like private members, but they may be accessed by derived classes. The base class access specification determines how private, protected, and public base class members may be accessed by derived classes.

Program 15-2

Contents of grade2.h

```
#ifndef GRADE2_H
#define GRADE2_H
// Grade class declaration
class Grade
{
 protected:
     char letter;
     float score;
     void calcGrade(void);
 public:
     void setScore(float s) { score = s; calcGrade(); }
     float getScore(void) {return score; }
     char getLetter(void) { return letter; }
};
#endif
```

Program continues

Contents of grade2.cpp

```
#include "grade2.h"
// Definition of member function Grade::calcGrade
void Grade::calcGrade()
{
 if (score > 89)
     letter = 'A';
 else if (score > 79)
     letter = 'B';
 else if (score > 69)
     letter = 'C';
 else if (score > 59)
     letter = 'D';
 else
     letter = 'F';
}
```

Program continues

Contents of test2.h

```
#ifndef TEST2_H
#define TEST2_H
#include "grade2.h"

// Test class declaration
class Test : public Grade
{
 private:
      int numQuestions;
      float pointsEach;
      int numMissed;
 public:
      Test(int, int);
      void adjustScore();
};
#endif
```

Program continues

Contents of test2.cpp

```
#include "test2.h"

// Definition of Test constructor.
// Parameters: q = Number of questions, m = number of
// questions missed.
Test::Test(int q, int m)
{
 float numericGrade;
 numQuestions = q;
 numMissed = m;
 pointsEach = 100.0 / numQuestions;
 numericGrade = 100.0 - (numMissed * pointsEach);
 setScore(numericGrade);
}
```

Program continues

```
// Definition of Test::adjustScore. If score is within
// 0.5 points of the next whole point, it rounds the score up
// and recalculates the letter grade.
void Test::adjustScore()
{
  if ((score - int(score)) >= 0.5)
  {
      score += (score - int(score));
      calcGrade();
  }
}
```

Contents of the main program, pr15-2.cpp

```
// This program demonstrates a base class and a derived class
#include <iostream.h>
#include <iomanip.h>
#include "test2.h"

void main(void)
{
  int questions, missed;

  cout << "How many questions are on the test? ";
  cin >> questions;
  cout << "How many questions did the student miss? ";
  cin >> missed;

  // Declare a Test object
  Test exam(questions, missed);
  cout.precision(2);
```

Program continues

```
  cout << "Unadjusted score: " << exam.getScore() << endl;
  cout << "Unadjusted grade: " << exam.getLetter() << endl;
  exam.adjustScore();
  cout << "Adjusted score is: " << exam.getScore() << endl;
  cout << "Adjusted grade is: " << exam.getLetter() << endl;
}
```

Program Output with Example Input

How many questions are on the test? 200 [Enter]
How many questions did the student miss? 21 [Enter]
Unadjusted score: 89.5
Unadjusted grade: B
Adjusted score is: 90
Adjusted grade is: A

Table 15-1

Base Class Access Specification	How Members of the Base Class Appear in the Derived Class
private	•Private members of the base class are inaccessible to the derived class. •Protected members of the base class become private members of the derived class. •Public members of the base class become private members the derived class.

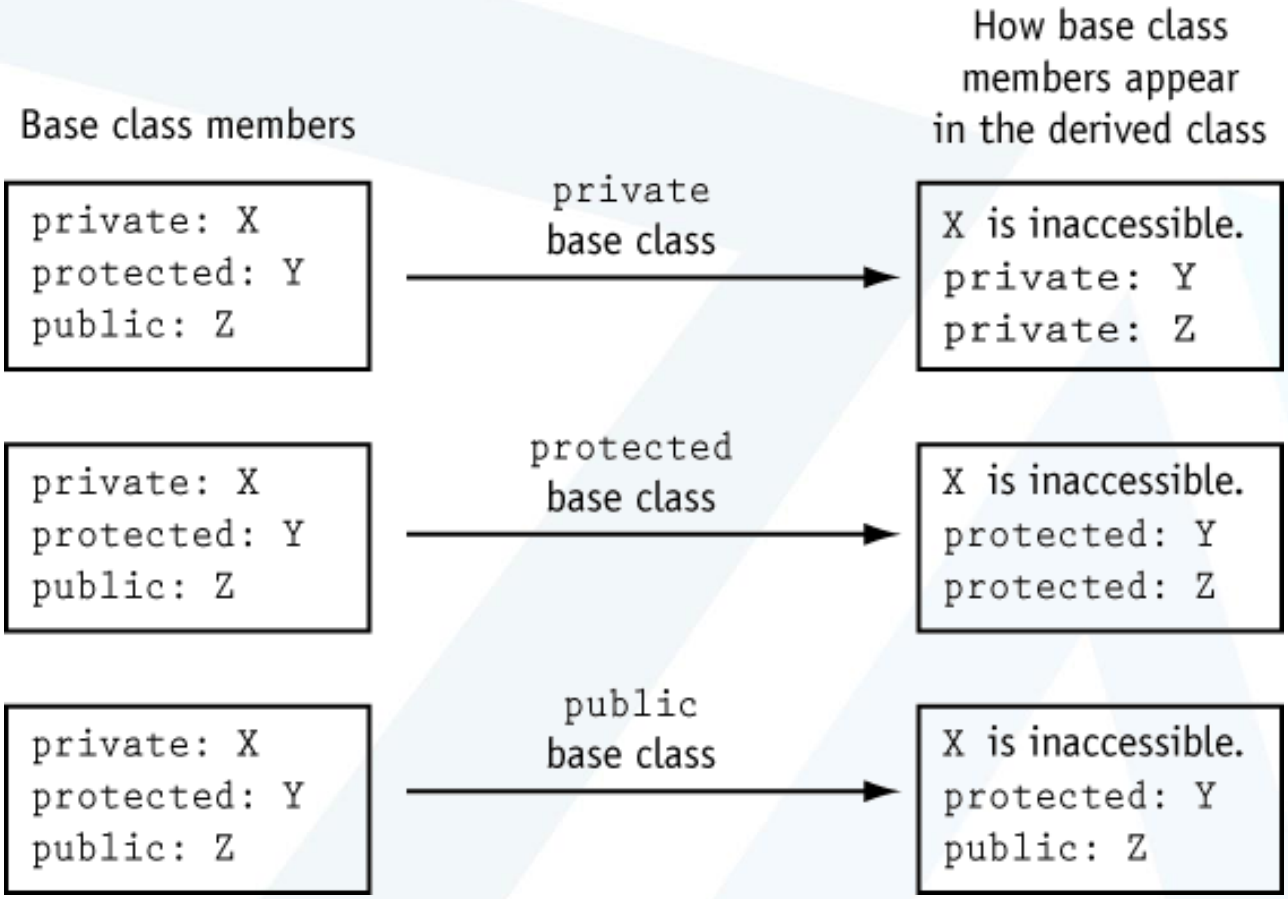
Table 15-1 *continued*

Base Class Access Specification	How Members of the Base Class Appear in the Derived Class
protected	•Private members of the base class are inaccessible to the derived class. •Protected members of the base class become protected members of the derived class. •Public members of the base class become protected members of the derived class.

Table 15-1 *continued*

Base Class Access Specification	How Members of the Base Class Appear in the Derived Class
public	•Private members of the base class are inaccessible to the derived class. •Protected members of the base class become protected members of the derived class. •Public members of the base class become public members of the derived class.

Figure 15-3


Base class members
How base class members appear in the derived class
private: X
protected: Y
public: Z
private base class
X is inaccessible.
private: Y
private: Z
private: X
protected: Y
public: Z
protected base class
X is inaccessible.
protected: Y
protected: Z
private: X
protected: Y
public: Z
public base class
X is inaccessible.
protected: Y
public: Z

15.3 Constructors and Destructors

- The base class's constructor is called before the derived class's constructor. The destructors are called in reverse order, with the derived class's destructor being called first.

Program 15-3

```
// This program demonstrates the order in which base and
// derived class constructors and destructors are called.
// For the sake of simplicity, all the class declarations
// are in this file.
#include <iostream.h>

// BaseDemo class
class BaseDemo
{
 public:
      BaseDemo(void)  // Constructor
         { cout << "This is the BaseDemo constructor.\n"; }
      ~BaseDemo(void) // Destructor
         { cout << "This is the BaseDemo destructor.\n"; }
};
```

Program continues

```
class DeriDemo : public BaseDemo
{
 public:
     DeriDemo(void)  //Constructor
        { cout << "This is the DeriDemo constructor.\n"; }
     ~DeriDemo(void)  // Destructor
        { cout << "This is the DeriDemo destructor.\n"; }
};

void main(void)
{
 cout << "We will now declare a DeriDemo object.\n";
 DeriDemo object;
 cout << "The program is now going to end.\n";
}
```

Program Output

We will now declare a DeriDemo object.
This is the BaseDemo constructor.
This is the DeriDemo constructor.
The program is now going to end.
This is the DeriDemo destructor.
This is the BaseDemo destructor.

Passing Arguments to Base Class Constructors

- Assume a class called Cube is derived from a class called Rect. Here is how the constructor looks in Rect:

```
Rect::Rect(float w, float l)
{
    width = w;
    length = l;
    area = length * width;
}
```

Cube constructor:

```
Cube::Cube(float wide, float long, float high) :
 Rect(wide, long)
{
   height = high;
   volume = area * high;
}
```

General Form:

```
<class name>::<class name>(parameter list) : <base
 class name> (parameter list)
```

- Note that the derived class constructor must take enough arguments to be able to pass to the base class constructor

Program 15-4

Contents of rect.h

```
#ifndef RECT_H
#define RECT_H
// Rect class declaration
class Rect
{
 protected:
      float width;
      float length;
      float area;
 public:
      Rect(void) { width = length = area = 0.0; }
      Rect(float, float);
      float getArea(void) { return area; }
      float getLen(void) { return length; }
      float getWidth(void) { return width; }
};
#endif
```

Program continues

Contents of rect.cpp

```
#include "rect.h"

// Definition of Rect constructor.
Rect::Rect(float w, float l)
{
 width = w;
 length = l;
 area = width * length;
}
```

Program continues

Contents of cube.h

```
#ifndef CUBE_H
#define CUBE_H
#include "rect.h"

// Cube class declaration
class Cube : public Rect
{
 protected:
     float height;
     float volume;
 public:
     Cube(float, float, float);
     float getHeight(void) { return height; }
     float getVol(void) { return volume; }
};
#endif
```

Program continues

Contents of cube.cpp

```
#include "cube.h"

// Definition of Cube constructor.
Cube::Cube(float wide, float long, float high) :
  Rect(wide, long)
{
  height = high;
  volume = area * high;
}
```

Contents of the main program, pr15-4.cpp

```
// This program demonstrates passing arguments to a base
// class constructor.

#include <iostream.h>
#include "cube.h"
```

Program continues

```
void main(void)
{
 float cubeWide, cubeLong, cubeHigh;
 cout << "Enter the dimensions of a Cube:\n";
 cout << "Width: ";  cin >> cubeWide;
 cout << "Length: "; cin >> cubeLong;
 cout << "Height: "; cin >> cubeHigh;
 Cube holder(cubeWide, cubeLong, cubeHigh);
 cout << "Here are the Cube's properties:\n";
 cout << "Width: " << holder.getWidth() << endl;
 cout << "Length: " << holder.getLen() << endl;
 cout << "Height: " << holder.getHeight() << endl;
 cout << "Base area: " << holder.getArea() << endl;
 cout << "Volume: " << holder.getVol() << endl;
}
```

Program Output with Example Input

Enter the dimensions of a Cube:
Width: **10** [**Enter**]
Length: **15** [**Enter**]
Height: **12** [**Enter**]
Here are the Cube's properties:
Width: 10
Length: 15
Height: 12
Base area: 150
Volume: 1800

15.4 Overriding Base Class Functions

- A member function of a derived class may have the same name as a member function of a base class.

Program 15-5

Contents of miledist.h

```
#ifndef MILEDIST_H
#define MILEDIST_H

// MileDist class declaration.
class MileDist
{
 protected:
     float miles;
 public:
     void setDist(float d) { miles = d; }
     float getDist(void) { return miles; }
};
#endif
```

Program continues

Contents of ftdist.h

```
#ifndef FTDIST_H
#define FTDIST_H
#include "miledist.h"

// FtDist class declaration/
class FtDist : public MileDist
{
 protected:
     float feet;
 public:
     void setDist(float);
     float getDist(void) { return feet; }
     float getMiles(void) { return miles; }
};
#endif
```

Program continues

Contents of ftdist.cpp

```
#include "ftdist.h"
void FtDist::setDist(float ft)
{
 feet = ft;
 MileDist::setDist(feet / 5280);  // Call base class function
}
```

Contents of the main program, pr15-5.cpp

```
// This program demonstrates a derived class with
// functions that override base class functions.
// NOTE: FTDIST.CPP must be compiled and linked with this
// program.
#include <iostream.h>
#include "ftdist.h"
```

Program continues

```
void main(void)
{
 FtDist feet;
 float ft;

 cout << "Enter a distance in feet and I will convert it\n";
 cout << "to miles: ";
 cin >> ft;
 feet.setDist(ft);
 cout.precision(1);
 cout.setf(ios::fixed);
 cout << feet.getDist(void) << " feet equals ";
 cout << feet.getMiles(void) << " miles.\n";
}
```

Program Output with Example Input

Enter a distance in feet and I will convert it
to miles: **12600** [**Enter**]
12600 feet equals 2.4 miles.

Program 15-6

```
// This program demonstrates that when a derived class function
// overrides a base class function, objects of the base class
// still call the base class version of the function.
#include <iostream.h>
class Base
{
 public:
     void showMsg(void)
          { cout << "This is the Base class.\n"; }
};
class Derived : public Base
{
 public:
     void showMsg(void)
          { cout << "This is the Derived class.\n"; }
};
```

Program continues

```
void main(void)
{
 Base b;
 Derived d;

 b.showMsg();
 d.showMsg();
}
```

Program Output

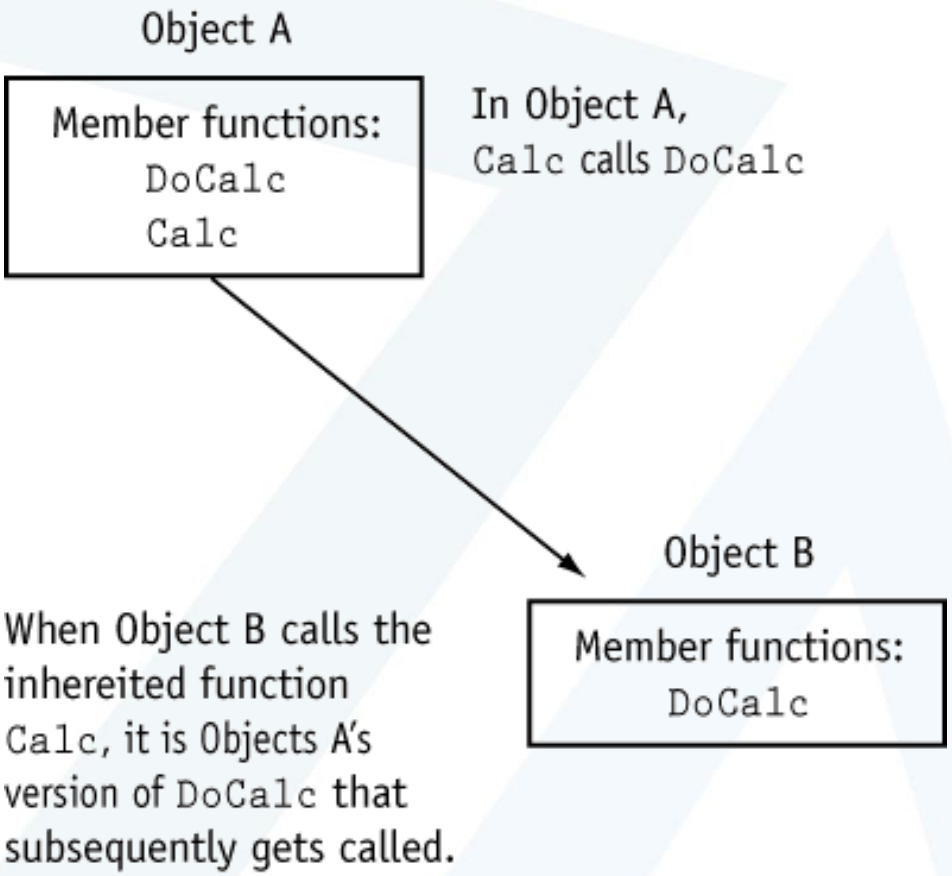
This is the Base class.

This is the Derived class.

15.5 Polymorphism and Virtual Member Functions

- A virtual member function in a base class expects to be overridden in a derived class

Figure 15-4


Object A
Member functions:
DoCalc
Calc
In Object A,
Calc calls DoCalc
Object B
Member functions:
DoCalc
When Object B calls the inhereited function Calc, it is Objects A's version of DoCalc that subsequently gets called.

Program 15-7

Contents of miledis2.h

```
#ifndef MILEDIS2_H
#define MILEDIS2_H

// MileDist class declaration.
class MileDist
{
 protected:
     float miles;
 public:
     void setDist(float d) { miles = d; }
     float getDist(void) { return miles; }
     float Square(void) { return getDist() * getDist();
 }
};
#endif
```

Program continues

Contents of ftdist2.h

```
#ifndef FTDIST2_H
#define FTDIST2_H
#include "miledis2.h"
// FtDist class declaration/
class FtDist : public MileDist
{
 protected:
     float feet;
 public:
     void setDist(float);
     float getDist(void) { return feet; }
     float getMiles(void) { return miles; }
};
#endif
```

Program continues

Contents of ftdist2.cpp

```
#include "ftdist2.h"
void FtDist::setDist(float ft)
{
 feet = ft;
 MileDist::setDist(feet / 5280); //Call base class function
}
```

Contents of the main program, PR15-7.CPP

```
// This program demonstrates a base class with an improperly
// overridden function.
#include <iostream.h>
#include "ftdist2.h"
```

Program continues

```
void main(void)
{
  FtDist feet;
  float ft;

  cout << "Enter a distance in feet and I will convert it\n";
  cout << "to miles: ";
  cin >> ft;
  feet.setDist(ft);
  cout.precision(1);
  cout.setf(ios::fixed);
  cout << feet.getDist(void) << " feet equals ";
  cout << feet.getMiles(void) << " miles.\n";
  cout << feet.getDist(void) << " square feet equals ";
  cout << feet.Square(void) << " total feet.\n";
}
```

Program Output with Example Input

Enter a distance in feet and I will convert it
to miles: **12600** [**Enter**]
12600 feet equals 2.4 miles.
12600 square feet equals 5.7 total feet.

Program 15-8

Contents of miledis3.h

```
#ifndef MILEDIS3_H
#define MILEDIS3_H

// MileDist class declaration.
class MileDist
{
 protected:
     float miles;
 public:
     void setDist(float d) { miles = d; }
     virtual float getDist(void) { return miles; }
     float Square(void) { return getDist() * getDist();
 }
};
#endif
```

Program continues

Contents of ftdist3.h

```
#ifndef FTDIST3_H
#define FTDIST3_H
#include "miledis3.h"

// FtDist class declaration/
class FtDist : public MileDist
{
 protected:
     float feet;
 public:
     void setDist(float);
     virtual float getDist(void) { return feet; }
     float getMiles(void) { return miles; }
};
#endif
```

Program continues

Contents of ftdist3.cpp

```
#include "ftdist3.h"

void FtDist::setDist(float ft)
{
 feet = ft;
 MileDist::setDist(feet / 5280);  // Call base class function
}
```

Contents of the main program, pr15-8.cpp

```
// This program demonstrates a base class with an improperly
// overriden function.

#include <iostream.h>
#include <iomanip.h>
#include "ftdist3.h"
```

Program continues

```
void main(void)
{
 FtDist distObject;
 float ft;
 cout << "Enter a distance in feet and I will convert it\n";
 cout << "to miles: ";
 cin >> ft;
 distObject.setDist(ft);
 cout.precision(1);
 cout.setf(ios::fixed);
 cout << distObject.getDist(void) << " feet equals ";
 cout << distObject.getMiles(void) << " miles.\n";
 cout << distObject.getDist(void) << " square feet equals ";
 cout << distObject.Square(void) << " total feet.\n";
}
```

Program Output with Example Input

Enter a distance in feet and I will convert it
to miles: **12600** [**Enter**]
12600 feet equals 2.4 miles.
12600 square feet equals 158760000 total feet.

15.6 Abstract Base Classes and Pure Virtual Functions

- An abstract base class is not instantiated, but other classes are derived from it.
- A pure virtual function is a virtual member function of a base class that must be overridden.

15.6 Abstract Base Classes and Pure Virtual Functions

- A class becomes an abstract base class when it contains one or more pure virtual functions.
- A pure virtual function is a virtual member function declared in a manner similar to the following:

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

Contents of student.h

```
#ifndef STUDENT_H
#define STUDENT_H
#include <string.h>    // For strcpy

class Student
{
protected:
        char name[51];
        char id[21];
        int yearAdmitted;
        int hoursCompleted;
public:

        Student(void)  // Constructor
                {name[0] = id[0] = yearAdmitted = hoursCompleted = 0; }
```

```
        void setName(char *n)
                { strcpy(name, n); }
        void setID(char *i)
                { strcpy(id, i); }
        void setYearAdmitted(int y)
                { yearAdmitted = y; }
        virtual void setHours(void) = 0;      // Pure virtual function
        virtual void showInfo(void) = 0;      // Pure virtual function
};

#endif
```

Contents of csstudent.h

```
ifndef CSSTUDENT_H
#define CSSTUDENT_H
#include "student.h"

class CsStudent : public Student
{
private:
        int mathHours;          // Hours of math taken
        int csHours;            // Hours of Computer Science taken
        int genEdHours;         // Hours of general education taken
public:
        void setMathHours(int mh)
                { mathHours = mh; }
        void setCsHours(int csh)
                { csHours = csh; }
        void setGenEdHours(int geh)
                { genEdHours = geh; }
```

```
        void setHours(void)
           { hoursCompleted = genEdHours + mathHours + csHours; }
        void showInfo(void);   // Defined in csstudent.cpp
};

#endif
```

Contents of csstudent.cpp

```
#include <iostream.h>
#include "csstudent.h"

void CsStudent::showInfo(void)
{
        cout << "Name: " << name << endl;
        cout << "Student ID: " << id << endl;
        cout << "Year admitted: " << yearAdmitted << endl;
        cout << "Summary of hours completed:\n";
        cout << "\tGeneral Education: " << genEdHours << endl;
        cout << "\tMath: " << mathHours << endl;
        cout << "\tComputer Science: " << csHours << endl << endl;
        cout << "\tTotal Hours Completed: " << hoursCompleted << endl;
}
```

Program 15-9

```
// This program demonstrates the CsStudent class, which i// derived
from the abstract base class, Student.

#include <iostream.h>
#include "csstudent.h"

void main(void)
{
        CsStudent student1;
        char chInput[51];       // Input buffer for entering C-strings.
        int intInput;           // Input buffer for entering integers.

        cout << "Enter the following student information:\n";
        // Set the student's name.
        cout << "Name: ";
        cin.getline(chInput, 51);
        student1.setName(chInput);
```

```
// Set the student's ID number.
cout << "Student ID: ";
cin.getline(chInput, 21);
student1.setID(chInput);

// Set the year admitted.
cout << "Year admitted: ";
cin >> intInput;
student1.setYearAdmitted(intInput);

// Set the # of general ed hours completed.
cout << "Number of general ed hours completed: ";
cin >> intInput;
student1.setGenEdHours(intInput);

// Set the # of math hours completed.
cout << "Number of math hours completed: ";
cin >> intInput;
student1.setMathHours(intInput);
```

```
        // Set the # of computer science hours completed.
        cout << "Number of computer science hours completed: ";
        cin >> intInput;
        student1.setCsHours(intInput);

        // Total the hours entered.
        student1.setHours();

        // Display the information provided.
        cout << "\nSTUDENT INFORMATION\n";
        student1.showInfo();

}
```

Program Output

```
Enter the following student information:
Name: Marty Stamey
Student ID: 167W98337
Year admitted: 2000
Number of general ed hours completed: 12
Number of math hours completed: 9
Number of computer science hours completed: 18
```

15.7 Base Class Pointers

- Pointer to a base class may be assigned the address of a derived class object. The pointer, however, will ignore any overrides the derived class performs

15.7 Base Class Pointers

Program 15-10

```
// This program demonstrates the behavior of a base class pointer
// when it is pointing to a derived class that overrides members
// of the base class.
// The class declarations are placed directly in the file for
// simplicity.

#include <iostream.h>

class Base
{
 public:
     void show(void)
          { cout << "This is from the Base class.\n"; }
};
```

Program continues

```
class Derived : public Base
{
 public:
     void show(void)
        { cout << "This is from the Derived class.\n"; }
};

void main(void)
{
 Base *bptr;
 Derived dObject;

 bptr = &dObject;
 bptr->show();  // Base class pointer, ignores override.
}
```

Program Output

This is from the Base class

15.8 Classes Derived from Derived Classes

- A base class can also be derived from another class.

Figure 15-5

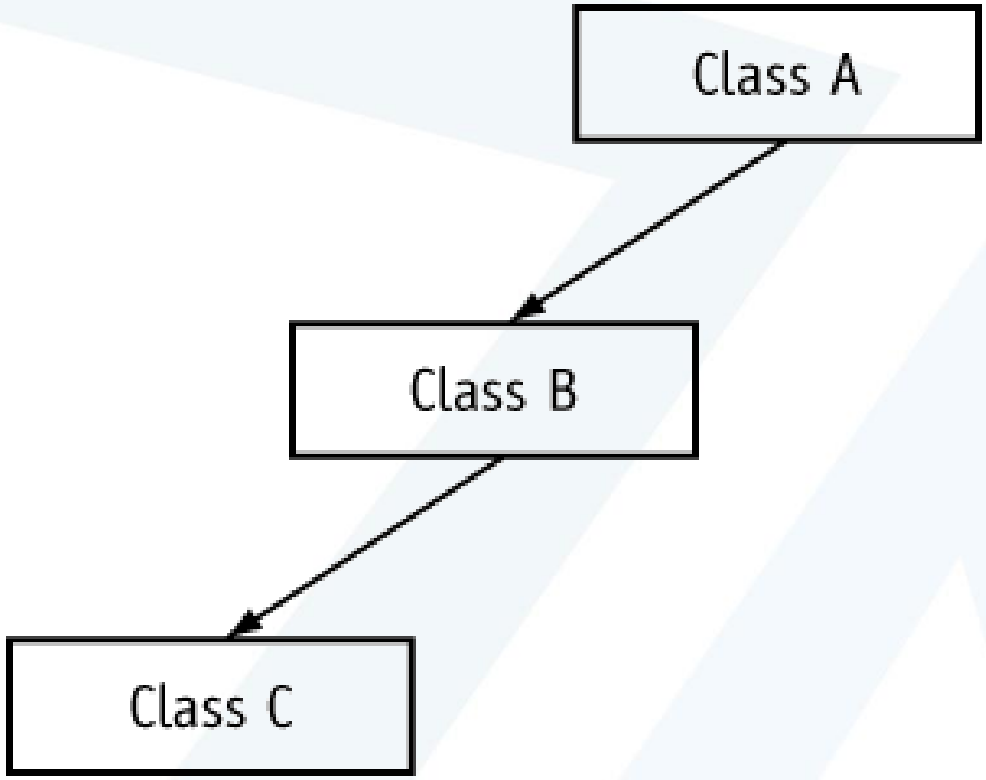


Table 15-2

Member	Access	Description
miles	protected	A member variable, inherited from the FtDist class, which inherited it from the MileDist class.
feet	protected	A member variable, inherited from the FtDist class.
Iinches	protected	A member variable.
setDist	public	A member function. This function overrides the SetDist functions in the base classes.
getDist	public	A member function. This function overrides the GetDist functions in the base classes.
getFeet	public	A member function, inherited from the FtDist class.
getMiles	public	A member function, inherited from the FtDist class, which inherited it MileDist class.

Program 15-11

Contents of inchdist.h

```
#ifndef INCHDIST_H
#define INCHDIST_H
#include "ftdist3.h" // Needed for FtDist class declaration.
// InchDist class declaration
class InchDist : public FtDist
{
 protected:
     float inches;
 public:
     void setDist(float);
     float getDist(void) { return inches; }
     float getFeet(void) { return feet; }
};
#endif
```

Program continues

Contents of inchdist.cpp

```
#include "inchdist.h"
// Definition of setDist member of InchDist class.
void InchDist::setDist(float in)
{
  inches = in;
  FtDist::setDist(inches / 12);  // Call base class function
}
```

Contents of the main program, pr15-11.cpp

```
// This program demonstrates a derived class derived from another
// derived class.
// NOTE: INCHDIST.CPP and FTDIST.CPP must be compiled and linked
// with this program.
#include <iostream.h>
#include <iomanip.h>
#include "inchdist.h"
```

Program continues

```
void main(void)
{
  InchDist inch;
  float in;

  cout << "Enter a distance in inches and I will convert\n";
  cout << "it to feet and miles: ";
  cin >> in;
  inch.setDist(in);
  cout.precision(1);
  cout.setf(ios::fixed);
  cout << inch.getDist() << " inches equals ";
  cout << inch.getFeet() << " feet.\n";
  cout << inch.getDist() << " inches equals ";
  cout << inch.getMiles() << " miles.\n";
}
```

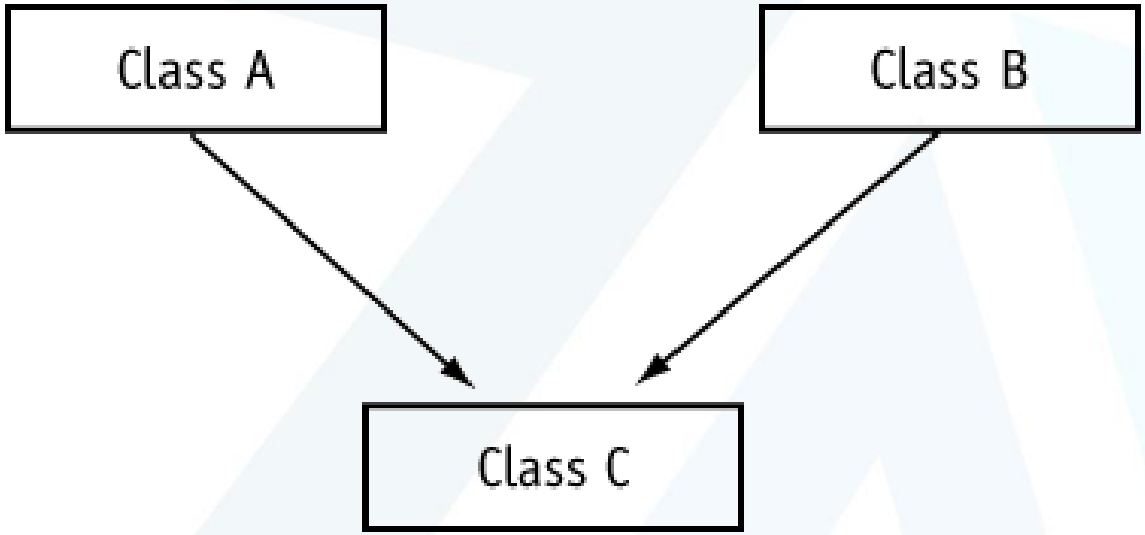
Program Output with Example Input

Enter a distance in inches and I will convert
it to feet and miles: **115900** [**Enter**]
115900 inches equals 9658.3 feet.
115900 inches equals 1.8 miles.

15.9 Multiple Inheritance

- Multiple inheritance is when a derived class has two or more base classes

Figure 15-6



Program 15-12

Contents of date.h

```
#ifndef DATE_H
#define DATE_H
class Date
{
 protected:
      int day;
      int month;
      int year;
 public:
      Date(int d, int m, int Y)
          { day = d; month = m; year = Y; }
      int getDay(void) { return day; }
      int getMonth(void) { return month; }
      int getYear(void) { return year; }
};
#endif
```

Program continues

Contents of time.h

```
#ifndef TIME_H
#define TIME_H
class Time
{
 protected:
      int hour;
      int min;
      int sec;
 public:
      Time(int h, int m, int s)
           { hour = h; min = m; sec = s; }
      int getHour(void) { return hour; }
      int getMin(void) { return min; }
      int getSec(void) { return sec; }
};
#endif
```

Program continues

Contents of datetime.h

```
#ifndef DATETIME_H
#define DATETIME_H
#include "date.h" // For Date class declaration
#include "time.h" // For Time class declaration

class DateTime : public Date, public Time
{
 protected:
     char dTString[20];
public:
 DateTime(int, int, int, int, int, int);
 void getDateTime(char *str) { strcpy(str, dTString); }
};
#endif
```

Program continues

Contents of datetime.cpp

```
#include “datetime.h”
#include <string.h> // For strcpy and strcat
#include <stdlib.h> // For itoa
void DateTime::DateTime(int dy, int mon, int yr, int hr, int mt,
  int sc) :
      Date(dy, mon, yr), Time(hr, mt, sc)
{
  char temp[10]; // Temporary work area for itoa()

  // Store the date in dTString, in the form MM/DD/YY
  strcpy(dTString, itoa(getMonth(), temp, 10));
  strcat(dTString, “/”);
  strcat(dTString, itoa(getDay(), temp, 10));
  strcat(dTString, “/”);
  strcat(dTString, itoa(getYear(), temp, 10));
  strcat(dTString, “ “);
```

Program continues

```
  // Store the time in dtString, in the form HH:MM:SS
  strcpy(dTString, itoa(getHour(), temp, 10));
  strcat(dTString, ":");
  strcat(dTString, itoa(getMin(), temp, 10));
  strcat(dTString, ":");
  strcat(dTString, itoa(getSec(), temp, 10));
}
```

Contents of the main program, pr15-12.cpp

```
// This program demonstrates a class with multiple inheritance.
// NOTE: datetime.cpp must be compiled and linked with this
// file.

#include <iostream.h>
#include "datetime.h"
```

Program continues

```
void main(void)
{
 char formatted[20];
 DateTime pastDay(4, 2, 60, 5, 32, 27);

 pastDay.getDateTime(formatted);
 cout << formatted << endl;
}
```

Program Output

4/2/60 5:32:27