9

Classes: A Deeper Look, Part I

My object all sublime
I shall achieve in time.
—W. S. Gilbert

Is it a world to hide virtues in?
—William Shakespeare

Don't be "consistent," but be simply true.
—Oliver Wendell Holmes, Jr.

This above all: to thine own self be true.
—William Shakespeare

OBJECTIVES

In this chapter you'll learn:

- How to use a preprocessor wrapper to prevent multiple definition errors caused by including more than one copy of a header file in a source-code file.
- To understand class scope and accessing class members via the name of an object, a reference to an object or a pointer to an object.
- To define constructors with default arguments.
- How destructors are used to perform "termination housekeeping" on an object before it is destroyed.
- When constructors and destructors are called and the order in which they are called.
- The logic errors that may occur when a `public` member function of a class returns a reference to `private` data.
- To assign the data members of one object to those of another object by default memberwise assignment.

Assignment Checklist

Name: ____________________________ Date: ____________________________

Section: ____________________________

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	11, 12, 13, 14, 15, 16, 17, 18, 19, 20	
Short Answer	21, 22, 23, 24	
Programming Output	25, 26, 27, 28	
Correct the Code	29, 30, 31, 32, 33, 34, 35	
Lab Exercises		
Lab Exercise 1 — Complex Numbers	YES NO	
Follow-Up Questions and Activities	1, 2, 3	
Lab Exercise 2 — Dates	YES NO	
Follow-Up Questions and Activities	1, 2, 3, 4	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3, 4, 5, 6, 7, 8	
Programming Challenges	1, 2, 3, 4	

Prelab Activities

Matching

Name: ______________________________ **Date:** ______________________________

Section: ______________________________

After reading Chapter 9 of *C++ How to Program, Seventh Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

	Term	Description
___	1. Data members	a) `public` or `private`.
___	2. Scope resolution operator	b) Making data accessible only to the class in which it's defined.
___	3. Constructor	c) Data components of a class.
___	4. Class interface	d) Function components of a class.
___	5. Destructor	e) Initializes a class's data members to appropriate values.
___	6. Member functions	f) `::`.
___	7. Information hiding	g) `public` member functions.
___	8. Member-access operators	h) Carries out "termination housekeeping."
___	9. Message	i) Member function call sent from one object to another.
___	10. Member-access specifiers	j) Dot operator (`.`) or the arrow operator (`->`).

Prelab Activities

Name:

Fill in the Blank

Name: ____________________ **Date:** ____________________

Section: ____________________

Fill in the blanks in each of the following statements:

11. ____________ enable the programmer to model objects that have attributes and behaviors or operations.

12. Generally, ____________ calls are made in the reverse order of the corresponding constructor calls.

13. A(n) ____________ initializes objects of a class.

14. A(n) ____________ function is a `private` member function that is intended to be used only by other member functions of the class.

15. Member-access specifiers always end with a(n) ____________ and can appear multiple times and in any order in a class definition.

16. When a member function is defined outside the class definition, the function name is preceded by the ____________ name and the ____________ operator.

17. A fundamental principle of good software engineering is separating ____________ from ____________.

18. ____________ of a class normally are made `private` and ____________ of a class normally are made `public`.

19. Constructors may not specify a(n) ____________.

20. The members of one object are assigned to the members of another object of the same type with ____________ assignment.

Prelab Activities

Name:

Short Answer

Name: ______________________________ **Date:** ______________________________

Section: ______________________________

In the space provided, answer each of the given questions. Your answers should be concise; aim for two or three sentences.

21. What is information hiding? Why is it important?

22. What are preprocessor wrappers used for? Give an example of a preprocessor wrapper.

23. Explain when to use the dot operator (`.`) and when to use the arrow operator (`->`).

24. What is the difference between class scope and file scope?

Prelab Activities

Name:

Programming Output

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

For *Programming Output Exercises 25* and *26*, use the class definition in Fig. L 9.1.

```
// Time abstract data type (ADT) definition
class Time
{
public:
   Time();                        // constructor
   Time( int, int, int );         // three-argument constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printUniversal();         // print universal time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23 ( 24-hour clock format )
   int minute;   // 0 - 59
   int second;   // 0 - 59
}; // end class Time

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time()
{
   hour = minute = second = 0;
} // end Time constructor

// Time constructor initializes each data member as specified.
Time::Time( int h, int m, int s )
{
   setTime( h, m, s );
} // end Time constructor

// Set a new Time value using universal time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
} // end function setTime

```

Fig. L 9.1 | `Time` class. (Part 1 of 2.)

Prelab Activities

Name:

Programming Output

```
// Print Time in universal format
void Time::printUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
        << setw( 2 ) << minute << ":"
        << setw( 2 ) << second;
} // end function printUniversal

// Print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << setfill( '0' ) << setw( 2 ) << minute
        << ":" << setw( 2 ) << second
        << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Fig. L 9.1 | Time class. (Part 2 of 2.)

25. What is output by the following code segment? Use the definition of the class Time shown in Fig. L 9.1.

```
Time t1();

t1.setTime( 18, 22, 9 );
cout << "The time is: ";
t1.printStandard();
```

Your answer:

26. What is the output of the following program segment?

```
Time t( 3, 4, 5 );

t.printStandard();
cout << endl;

t.printUniversal();
cout << endl;

t.setTime( 99, 3, 4 );

t.printUniversal();
cout << endl;
```

Prelab Activities

Name:

Programming Output

Your answer::

27. What is output by the following program? Use the `Time` class shown in Fig. L 9.1.

```
#include <iostream>
using namespace std;

class M
{
public:
   M( int );
   int mystery( int );
private:
   int data;
   double number;
}; // end class M

// constructor
M::M( int q )
{
   data = q;
   number = .5;
} // end class M constructor

// function mystery definition
int M::mystery( int q )
{
   data += q;
   return data * number;
} // end function mystery

int main()
{
   M stuff( 44 );
   cout << stuff.mystery( 78 );
} // end main
```

Your answer:

28. What is output by the following program?

```
#include <iostream>
using namespace std;
```

Prelab Activities

Name:

Programming Output

```

class M
{
public:
   M( int );
   int mystery( int );
private:
   int data;
   int number;

}; // end class M

// constructor
M::M( int q = 0 )
{
   data = q;
   number = 2;
} // end class M constructor

// function mystery definition
int M::mystery( int q )
{
   data += q;
   return data;
} // end function mystery

int main()
{
   M mObject( 2 );
   M *mPtr = &mObject;

   cout << mObject.mystery( 20 ) << endl;
   cout << mPtr->mystery( 30 );
} // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

Name: ______________________ Date: ______________________

Section: ______________________

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic error or a compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note:* It is possible that a program segment may contain multiple errors.]

29. The following code should set the `hour`, `minute`, and `second` variables within a `Time` class. Use the definition for the `Time` class defined in Fig. L 9.2.

```
class Time
{
public:
   int hour;     // 0-23
   int minute;   // 0-59
   int second;   // 0-59
}; // end class Time
```

Fig. L 9.2 | `Time` class definition.

```
Time clock;
Time *clockPtr = &clock;

clock.hour = 8;
clock.minute = 12
*clockPtr.second = 0;
```

Your answer:

Prelab Activities

Name:

Correct the Code

30. The following should define class `Time`:

```cpp
class Time
{
public:
   Time( int = 0, int = 0, int );
   void setTime( int, int, int );
   void printUniversal();
   void printStandard();
private:
   int hour;
   int minute;
   int second;
} // end class Time
```

Your answer:

31. The following code defines class `Q`:

```cpp
class Q
{
public:
   int Q( int );
   void setQ( int );
   void printQ();
   int operateQ( int );
private:
   int qData;
}; // end class Q
```

Prelab Activities

Name:

Correct the Code

Your answer:

32. The following is another version of class `Q`'s definition:

```
class Q
{
public:
   Q( int );
   void setQ( int );
   void printQ();
   int operateQ( int );
private:
   int qData = 1;
}; // end class Q
```

Your answer:

Prelab Activities

Name:

Correct the Code

33. The following defines `Q`’s `setQ` method. This definition resides outside class `Q`’s definition. Use the corrected class `Q` from *Correct the Code Exercise 35*:

```
void setQ( int input )
{
   qData = input;
}
```

Your answer:

34. The following defines `setHour`, a member function of the `Time` class, Fig. L 9.1.

```
int &Time::setHour( int hh )
{
   hour = ( hh >= 0 && hh < 24 ) ? hh : 0;

   return hour;
}
```

Your answer:

Prelab Activities

Name:

Correct the Code

35. The following code should call member function `printUniversal` of the `Time` class defined in Fig. L 9.1.

```
Time clock( 11, 22, 43 );
Time *clockPtr = &clock;

clockPtr.printUniversal();
```

Your answer:

Lab Exercises

Lab Exercise 1 — Complex Numbers

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 9.3–Fig. L 9.5)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 9 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Creating new data types by writing class definitions.
- Defining member functions of programmer-defined classes.
- Instantiating objects from programmer-defined classes.
- Calling member functions of programmer-defined classes.

The follow-up questions and activities will also give you practice:

- Initializing programmer-defined class data members with class constructors.

Description of the Problem

Create a class called `Complex` for performing arithmetic with complex numbers. Write a program to test your class.

Complex numbers have the form

`realPart + imaginaryPart *` *i*

where *i* is

$$\sqrt{-1}$$

Use `double` variables to represent the `private` data of the class. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should contain default values in case no initializers are provided. Provide `public` member functions that perform the following tasks:

a) Adding two `Complex` numbers: The real parts are added together and the imaginary parts are added together.

Lab Exercises

Name:

Lab Exercise 1 — Complex Numbers

b) Subtracting two Complex numbers: The real part of the right operand is subtracted from the real part of the left operand and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.

c) Printing Complex numbers in the form (a, b) where a is the real part and b is the imaginary part.

Sample Output

```
(1, 7) + (9, 2) = (10, 9)
(10, 1) - (11, 5) = (-1, -4)
```

Template

```cpp
// Lab 1: Complex.h
#ifndef COMPLEX_H
#define COMPLEX_H

/* Write class definition for Complex */

#endif
```

Fig. L 9.3 | Complex.h.

```cpp
// Lab 1: Complex.cpp
// Member-function definitions for class Complex.
#include <iostream>
using namespace std;

#include "Complex.h"

Complex::Complex( double real, double imaginary )
{
   setComplexNumber( real, imaginary );
} // end Complex constructor

Complex Complex::add( const Complex &right )
{
   /* Write a statement to return a Complex object. Add
      the realPart of right to the realPart of this Complex
      object and add the imaginaryPart of right to the
      imaginaryPart of this Complex object */
} // end function add

Complex Complex::subtract( const Complex &right )
{
   /* Write a statement to return a Complex object. Subtract
      the realPart of right from the realPart of this Complex
      object and subtract the imaginaryPart of right from
      the imaginaryPart of this Complex object */
} // end function subtract

void Complex::printComplex()
{
```

Fig. L 9.4 | complex.cpp. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Complex Numbers

```
   cout << '(' << realPart << ", " << imaginaryPart << ')';
} // end function printComplex

void Complex::setComplexNumber( double rp, double ip )
{
   realPart = rp;
   imaginaryPart = ip;
} // end function setComplexNumber
```

Fig. L 9.4 | `complex.cpp`. (Part 2 of 2.)

```
// Lab 1: ComplexTest.cpp
#include <iostream>
using namespace std;

#include "Complex.h"

int main()
{
   Complex a( 1, 7 ), b( 9, 2 ), c; // create three Complex objects

   a.printComplex(); // output object a
   cout << " + ";
   b.printComplex(); // output object b
   cout << " = ";
   c = a.add( b ); // invoke add function and assign to object c
   c.printComplex(); // output object c

   cout << '\n';
   a.setComplexNumber( 10, 1 ); // reset realPart and
   b.setComplexNumber( 11, 5 ); // and imaginaryPart
   a.printComplex(); // output object a
   cout << " - ";
   b.printComplex(); // output object b
   cout << " = ";
   c = a.subtract( b ); // invoke add function and assign to object c
   c.printComplex(); // output object c
   cout << endl;
} // end main
```

Fig. L 9.5 | `ComplexTest.cpp`.

Problem-Solving Tips

1. In this lab, you must write the definition for class `Complex`. Use the details provided in the member definition (`Complex.cpp`) file to assist you.
2. Remember to use member-access specifiers `public` and `private` to specify the access level of data members and functions. Carefully consider which access specifier to use for each class member. In general, data members should be `private` and member functions should be `public`.

Lab Exercises

Name:

Lab Exercise 1 — Complex Numbers

Follow-Up Questions and Activities

1. Why do you think `const` was used in the parameter list of `add` and `subtract`?

2. Can `add` and `subtract`'s parameters be passed by value instead of by reference? How might this affect the design of class `Complex`? Write a new class definition that illustrates how the parameters would be passed by value.

3. Declare a `Complex` number, as follows, without passing any arguments to the constructor. What happens? Does the default constructor get called?

Lab Exercises

Name:

Lab Exercise 2 — Dates

Name: ______________________ **Date:** ______________________

Section: ______________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 9.6–Fig. L 9.8)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 9 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Using access functions and utility functions so that it is not necessary for non-member functions to be able to access a class' data members.

The follow-up questions and activities also will give you practice:

- Overloading constructors and using default arguments with constructors.
- Defining a destructor.

Description of the Problem

Modify the `Date` class of Fig. 9.17–Fig. 9.18 of *C++ How to Program, Seventh Edition* to provide a member function `nextDay` to increment the day by one. The `Date` object should always remain in a consistent state. Write a program that tests function `nextDay` in a loop that prints the date during each iteration to illustrate that the `nextDay` function works correctly. Be sure to test the following cases:

a) Incrementing into the next month.

b) Incrementing into the next year.

Lab Exercises

Name:

Lab Exercise 2 — Dates

Sample Output

```
12-24-2004
12-25-2004
12-26-2004
12-27-2004
12-28-2004
12-29-2004
12-30-2004
12-31-2004
1-1-2005
1-2-2005
1-3-2005
1-4-2005
1-5-2005
1-6-2005
1-7-2005
1-8-2005
```

Template

```cpp
// Lab 2: Date.h
#ifndef DATE_H
#define DATE_H

class Date
{
public:
   Date( int = 1, int = 1, int = 2000 ); // default constructor
   void print(); // print function
   void setDate( int, int, int ); // set month, day, year
   void setMonth( int ); // set month
   void setDay( int ); // set day
   void setYear( int ); // set year
   int getMonth(); // get month
   int getDay(); // get day
   int getYear(); // get year
   /* Write a member function prototype for nextDay,
      which will increment the Date by one day */
private:
   int month; // 1-12
   int day; // 1-31 (except February(leap year), April, June, Sept, Nov)
   int year; // 1900+
   bool leapYear(); // leap year
   int monthDays(); // days in month
}; // end class Date

#endif
```

Fig. L 9.6 | Date.h.

Lab Exercises

Name:

Lab Exercise 2 — Dates

```
// Lab 2: Date.cpp
// Member-function definitions for class Date.
#include <iostream>
using namespace std;

#include "Date.h" // include definition of class Date

Date::Date( int m, int d, int y )
{
   setDate( m, d, y ); // sets date
} // end Date constructor

void Date::setDate( int mo, int dy, int yr )
{
   setMonth( mo ); // invokes function setMonth
   setDay( dy ); // invokes function setDay
   setYear( yr ); // invokes function setYear
} // end function setDate

void Date::setDay( int d )
{
   if ( month == 2 && leapYear() )
      day = ( d <= 29 && d >= 1 ) ? d : 1;
   else
      day = ( d <= monthDays() && d >= 1 ) ? d : 1;
} // end function setDay

void Date::setMonth( int m )
{
   month = m <= 12 && m >= 1 ? m : 1; // sets month
} // end function setMonth

void Date::setYear( int y )
{
   year = y >= 1900 ? y : 1900; // sets year
} // end function setYear

int Date::getDay()
{
   return day;
} // end function getDay

int Date::getMonth()
{
   return month;
} // end function getMonth

int Date::getYear()
{
   return year;
} // end function getYear

void Date::print()
{
   cout << month << '-' << day << '-' << year << '\n'; // outputs date
} // end function print
```

Fig. L 9.7 | Date.cpp. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Dates

```cpp

/* Write code to define member function nextDay;
   make sure to check if the new day is the start of
   a new month or a new year */

bool Date::leapYear()
{
   if ( getYear() % 400 == 0 || ( getYear() % 4 == 0 && getYear() % 100 != 0 ) )
         return true; // is a leap year
      else
         return false; // is not a leap year
} // end function leapYear

int Date::monthDays()
{
   const int days[ 12 ] =
     { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

   return getMonth() == 2 && leapYear() ? 29 : days[ getMonth() - 1 ];
} // end function monthDays
```

Fig. L 9.7 | Date.cpp. (Part 2 of 2.)

```cpp
// Lab 2: DateTest.cpp
#include <iostream>
using namespace std;

#include "Date.h" // include definitions of class Date

int main()
{
   const int MAXDAYS = 16;
   Date d( 12, 24, 2004 ); // instantiate object d of class Date

   // output Date object d's value
   for ( int loop = 1; loop <= MAXDAYS; ++loop )
   {
      d.print(); // invokes function print
      /* Write call to nextDay */
   } // end for

   cout << endl;
} // end main
```

Fig. L 9.8 | DateTest.cpp.

Problem-Solving Tips

1. In this lab you will implement function `nextDay`. This function should increment the day and determine whether the month should also be incremented. If so, the function should determine whether the next also must be incremented.
2. Use functions `setDay`, `setMonth` and `setYear` as part of your `nextDay` implementation.

Lab Exercises

Name:

Lab Exercise 2 — Dates

Follow-Up Questions and Activities

1. The `Date` class has only one constructor. Is it possible to have more than one constructor?

2. What happens when a member function that takes no arguments is called without the parentheses (i.e., `dateObject.nextDay`)?

3. Write a destructor for the `Date` class. The destructor should print text indicating that the `Date` class destructor was called successfully.

4. In `main`, try to change `d`’s `year` to `2003` using an assignment statement. Do not call function `setYear`. What happens? Are you able to change the value?

Lab Exercises

Name:

Debugging

Name: ______________________ Date: ______________________

Section: ______________________

The program (Fig. L 9.9–Fig. L 9.11) in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code has been corrected.

Sample Output

```
This is the: Ace of spades
This is the: 4 of hearts
This card is not valid
This is the: 4 of hearts

This is the: Ace of hearts
This is the: 5 of hearts
This is the: Queen of clubs
This is the: 5 of hearts

The destructor has been invoked
The destructor has been invoked
The destructor has been invoked
```

Broken Code

```cpp
// Debugging: Card.h

#ifndef CARD_H
#define CARD_H

// class card definition
class Card {

public
   void Card();
   void Card( int, int );
   void ~Card();

   void setSuit( int );
   int getSuit() const;

   void setValue( int );
   int getValue() const;

   void print() const;

```

Fig. L 9.9 | Card.h. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
private
   int suit = 4;
   int value = 1;
   bool validCard() const;

} // end class Card

#endif // CARD_H
```

Fig. L 9.9 | `Card.h`. (Part 2 of 2.)

```
// Debugging: Card.cpp
#include <iostream>
using namespace std;

// default constructor
void Card::Card()
{
   suit = 4;
   value = 1;

} // end class Card constructor

// constructor
Card::Card( int s, int v )
{
   suit = s; value = v;

} // end class Card constructor

// destructor
Card::~Card()
{
   cout << "The destructor has been invoked\n";

} // end class Card destructor

// set suit
void Card::setSuit( int s )
{
   suit = s;

} // end function setSuit

// set value
void Card::setValue( int v )
{
   value = v;

} // end function setValue

```

Fig. L 9.10 | `Card.cpp`. (Part 1 of 3.)

Lab Exercises

Name:

Debugging

```
// function print definition
void print()
{
   // is card valid
   if ( !validCard() ) {
      cout << "This card is not valid\n";
      return;

   } // end if

   cout << "This is the: ";

   // determine face of card
   switch ( value ) {
      case 1:
         cout << "Ace ";
         break;

      case 11:
         cout << "Jack ";
         break;

      case 12:
         cout << "Queen ";
         break;

      case 13:
         cout << "King ";
         break;

      default:
         cout << value << " ";

   } // end switch

   // determine suit
   switch ( suit ) {
      case 1:
         cout << "of clubs\n";
         break;

      case 2:
         cout << "of diamonds\n";
         break;

      case 3:
         cout << "of hearts\n";
         break;

      case 4:
         cout << "of spades\n";
         break;

```

Fig. L 9.10 | `Card.cpp`. (Part 2 of 3.)

Lab Exercises

Name:

Debugging

```
      default:
         cout << "\ninvalid suit\n";

   } // end switch

} // end function print

// return suit
int Card::getSuit()
{
   return suit;

} // end function getSuit

// return value
int Card::getValue()
{
   return value;

} // end function getValue

// function validCard definition
bool validCard()
{
   return value >= 1 && value <= 13 && suit >= 1 && suit <= 4;

} // end function validCard
```

Fig. L 9.10 | Card.cpp. (Part 3 of 3.)

```
// Debugging: CardTest.cpp
#include <iostream>
using namespace std;

int main()
{
   Card c1;
   Card c2( 3, 4 );
   Card c3( 1, 14 );

   Card *p1 = &c2;

   c1.print();
   c2.print();
   c3.print();
   p1->print();
   cout << endl;

   c1.setSuit( p1->getSuit() );
   c3.value = 12;
   p1->value = 5;

```

Fig. L 9.11 | CardTest.cpp. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
   c1.print();
   c2.print();
   c3.print();
   *p1.print();
   cout << endl;
} // end main
```

Fig. L 9.11 | CardTest.cpp. (Part 2 of 2.)

Postlab Activities

Coding Exercises

Name: ______________________ **Date:** ______________________

Section: ______________________

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For each of the following problems, write a program or a program segment that performs the specified action:

1. Write the class definition (do not define any methods) for a polynomial of the form

   $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$

   where n is the degree of the `Polynomial`. Assume that the largest polynomial to be used has degree `10`. The class definition should contain a constructor, data members and member function prototypes.

2. Instantiate an object of type `Polynomial` with degree `3`.

Postlab Activities

Name:

Coding Exercises

3. Set the coefficients of your `Polynomial` object to `3`, `-10`, `4` and `1`, respectively.

4. Write the class definition for class `BookIndex`, which contains information found in a library's card catalog. It should contain the title, author and copyright year (in the form *yyyy*) as well as member functions for retrieving and manipulating data.

Postlab Activities

Name:

Coding Exercises

5. The title and author of most books never change after the initial publication. However, copyrights get updated for new editions. Redefine the class `BookIndex` with this information in mind. [*Hint:* Use keyword `const` where appropriate.]

6. Instantiate an object of type `BookIndex`, then declare a pointer and assign the object's address to it. Change the book's copyright via the pointer and -> operator.

7. Change the copyright through the pointer again, this time using the * operator instead of the -> operator.

Postlab Activities

Name:

Coding Exercises

8. Change the copyright a third time using a reference to the `BookIndex` object.

Postlab Activities

Name:

Programming Challenges

Name: ______________________________ **Date:** ______________________________

Section: ______________________________

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a C++ program for each of the problems in this section. The answers to these problems are available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. Provide a constructor that is capable of using the current time from the `time()` function—declared in the C++ Standard Library header `<ctime>`—to initialize an object of the `Time` class.

Hints:

- Write a new constructor that sets the members of the `time` function to the current time.
- Determine the current year using the following formula: *current year – 1970*.
- Depending on the time zone you are in, you must shift the time by a certain number of hours. For this problem, 5 hours (or 4 hours during Daylight Savings) is the current shift for Eastern Standard Time (EST).
- Sample output:

```
The universal time is 14:54:06
The standard time is 2:54:06 PM
```

2. Create a class called `Rational` for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the `private` data of the class—the `numerator` and the `denominator`. Provide a constructor that enables an object of this class to be initialized when it is instantiated. The constructor should contain default values in case no initializers are provided and should store the fraction in reduced form. For example, the fraction

$$\frac{2}{4}$$

would be stored in the object as 1 in the `numerator` and 2 in the `denominator`. Provide `public` member functions that perform each of the following tasks:

a) Adding two `Rational` numbers. The result should be stored in reduced form.

b) Subtracting two `Rational` numbers. The result should be stored in reduced form.

c) Multiplying two `Rational` numbers. The result should be stored in reduced form.

d) Dividing two `Rational` numbers. The result should be stored in reduced form.

e) Printing `Rational` numbers in the form `a/b` where `a` is the numerator and `b` is the denominator.

f) Printing `Rational` numbers in floating-point format.

Postlab Activities

Name:

Programming Challenges

Hints:

- The parameters of the functions that perform addition, subtraction, multiplication and division should all be `const`.
- Write a `private` utility function to perform reduction and call this function after every operation to ensure that the fraction is stored in reduced form.
- Sample output:

```
1/3 + 7/8 = 29/24
29/24 = 1.20833

1/3 - 7/8 = -13/24
-13/24 = -0.541667

1/3 x 7/8 = 7/24
7/24 = 0.291667

1/3 / 7/8 = 8/21
8/21 = 0.380952
```

3. Modify the `Time` class of Fig. 9.8–9.9 of *C++ How to Program, Seventh Edition* to include a `tick` member function that increments the time stored in a `Time` object by one second. The `Time` object should always remain in a consistent state. Write a program that tests the `tick` member function in a loop that prints the time in standard format during each iteration of the loop to illustrate that the `tick` member function works correctly. Be sure to test the following cases:

   a) Incrementing into the next minute

   b) Incrementing into the next hour

   c) Incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM)

Hints:

- `tick` should increment `second` by one (use ++).
- Determine whether the next minute has begun. Remember, when `setSecond` gets an invalid value (i.e., 60 or higher), it sets second to `0`.
- Do not forget to provide a similar implementation for hours.
- Sample output:

```
11:59:57 PM
11:59:58 PM
11:59:59 PM
12:00:00 AM
12:00:01 AM
.
.
.
```

Postlab Activities

Name:

Programming Challenges

4. Create a class `Rectangle` with attributes `length` and `width`, each of which defaults to `1`. Provide member functions that calculate the `perimeter` and the `area` of the rectangle. Also, provide *set* and *get* functions for the `length` and `width` attributes. The *set* functions should verify that `length` and `width` are each floating-point numbers larger than 0.0 and less than 20.0.

Hints:

- *Perimeter* = *2* × *(length* + *width)*
- *Area* = *length* × *width*
- Sample output:

```
a: length = 1.0; width = 1.0; perimeter = 4.0; area = 1.0
b: length = 5.0; width = 4.0; perimeter = 18.0; area = 20.0
c: length = 1.0; width = 1.0; perimeter = 4.0; area = 1.0
```